3-2018

USING AUTOENCODER TO REDUCE THE LENGTH OF THE AUTISM DIAGNOSTIC OBSERVATION SCHEDULE (ADOS)

Sara Hussain Daghustani
California State University - San Bernardino

USING AUTOENCODER TO REDUCE THE LENGTH OF THE AUTISM DIAGNOSTIC OBSERVATION SCHEDULE (ADOS)

A Thesis

Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

---

by

Sara Hussain Daghustani

March 2018

USING AUTOENCODER TO REDUCE THE LENGTH OF THE AUTISM DIAGNOSTIC OBSERVATION SCHEDULE (ADOS)

A Thesis

Presented to the

Faculty of

California State University,

San Bernardino

---

by

Sara Daghustani

March 2018

Approved by:

Dr. Kerstin Voigt, Adviser, Computer Science and Engineering

Dr. George Georgiou, Committee Member

Dr. Qingquan Sun, Committee Member

ABSTRACT

This thesis uses autoencoders to explore the possibility of reducing the length of the Autism Diagnostic Observation Schedule (ADOS), which is a series of tests and observations used to diagnose autism spectrum disorders in children, adolescents, and adults of different developmental levels. The length of the ADOS, directly and indirectly, causes barriers to its access for many individuals, which means that individuals who need testing are unable to get it. Reducing the length of the ADOS without significantly sacrificing its accuracy would increase its accessibility. The autoencoders used in this thesis have specific connections between layers that mimic the sectional structure of the original ADOS. Autoencoders reduce the length of the ADOS by conducting its dimensionality through combining original variables into new variables. By examining the weights of variables entering the reduced diagnostic, this thesis explores which variables are prioritized and deprioritized by the autoencoder. These information yields insights as to which variables, and underlying concepts, should prioritize in a shorter ADOS. After training, all autoencoders used were able to reduce dimensionality with minimal accuracy losses. Examination of weights yielded many keen insights as to which ADOS variables are the least important to their modules and can thus be eliminated or deprioritized in a reduced diagnostic. In particular, the observation of self-injurious behavior was declared entirely unnecessary in the first three modules of the ADOS, a finding that corroborates

other recent experimental results in the domain. This observation suggests that the solutions converged upon by the model have real-world significance.

ACKNOWLEDGEMENTS

I dedicate my thanks to my beloved parents for their encouragement and believing in me. Many thanks to my motivational source my sister Wid Daghustani for being such an inspiration to me. My thanks to all my friends who helped me pass all the obstacles in my master's degree journey. Thanks to everyone I meet through my studying years for his or her help. I am also grateful to Mr. Mohammad Al-Hijjawi for providing me Nvidia GPU Hardware. My appreciations to my adviser Dr. Voigt for her precious supervision and suggestions during the thesis progress. Moreover, thanks to the committee members Dr. Georgiou and Dr. Sun for their cooperation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

1.1 Introduction

Autism spectrum disorders (ASD) can pose tremendous developmental challenges to children who have them. When parents suspect that their child's neurological development is abnormal, it is essential for them to have their child examined and assessed by experts to verify if an autism spectrum disorder causes their troubles. Unfortunately, the autism diagnostic process is often cumbersome and costly, and many parents faced with significant barriers to its access. Many of these obstacles, directly and indirectly, result from the length of the autism diagnostic process. If the process could shorten without decreasing its effectiveness, many parents and their children could greatly benefit from the erosion of barriers to accessing autism diagnostic services.

Autism spectrum disorders (ASD) are a group of complex neurological conditions of brain development. These disabilities are characterized by weakness of social interaction and communication, repetitive and stereotypic behaviors, reorganized in children during the first years [1]. The symptoms of ASD are apparent in children before three years of age, but in a rare situation, it can diagnosed at even earlier stages.

According to a US government survey of American parents, in the United States, for every 45 children, at least one is diagnosed with ASD [2]. This number

is higher than the Center for Disease Control and Prevention's (CDC) estimation, which claims that one in 68 children is diagnosed with ASD [3]. The incidence rate of autism worldwide is approximately 20 per 10,000 children, and it is four times greater in male children than in female children [4].

Autism diagnosis is a process executed by specialists to verify the existence of autism, identify its causes, and to propose appropriate therapeutic intervention programs. Autism is difficult to diagnose, and the process of diagnosis is an ongoing research challenge for specialists in the field. The current standard for autism diagnosis is the Autism Diagnostic Observation Schedule (ADOS), which is a series of tests and observations administered to children, adolescents, and adults of different stages of development. The length and complexity of the ADOS logistically limits the number of individuals that can be tested for autism. If the ADOS was shorter and less complex, evidence exists to suggest that a more substantial amount of individuals could access and afford autism diagnosis services.

1.2 Thesis Scope

his thesis uses autoencoders to test if all four modules of the Autism Diagnostic Observation Schedule can be reduced in dimensionality, and to examine which variables are prioritized and deprioritized in the dimensionality reduction process. The impact of original ADOS variables in the reduced

diagnostic is reviewed, and all insights gained from this process are catalogued and explored.

1.3 Purpose

The purpose of this thesis is to create an algorithm that reduces the length of the Autism Diagnostic Observation Schedule by lowering its dimensionality without significantly sacrificing its accuracy. The algorithm is designed to incorporate the sectional structure of the ADOS. When reducing dimensionality, the algorithm prioritizes and de-prioritizes different variables. Examining this information will yield insights regarding the ideal conceptual focus of potentially reduced versions of the ADOS.

The ADOS is currently a long and cumbersome test to administer, which is problematic from an access standpoint. There is plenty of evidence that indicates that autism is under-diagnosed in developing countries. Part of the problem is that there are too many barriers to administering the test. According to a paper covering the hurdles of implementing the test in Jamaica, high administration costs and long waiting lists prevent many children from being tested for autism [5]. The high administration cost and long waiting lists can likely be at least partially attributed to the length of the test. More extended tests reduce the number of tests that can be administered in one day, which undoubtedly exacerbates long waiting lists. Likewise, the length and complexity of the test

drive up administration costs [6]. Finding a way to reduce the test without sacrificing its accuracy could allow for a newer version of the ADOS to be accessed by more significant numbers of potential patients worldwide. Any project that assists in this endeavor is a worthwhile task. In the United States, a full-length neuropsychological evaluation of the potential for autism spectrum disorder within a child can cost between $1,200 to $2,500 and take up to five hours. For a bare bones autism diagnostic process, without any interaction with parents or detailed feedback, the price can range from $500 to $800, and can take up to 2 hours. The price of these procedures depends upon the length and amount of labor required from highly-compensated specialists in the field [7].

1.4 Approach

This thesis employs an autoencoder with connections that mirror the sectional structure of each ADOS module. The extent which each variable in the input influenced the reduced-length diagnostic is assessed and their relative impacts are ranked. This information might yield insights as to which aspects of the ADOS are the most and least important. Furthermore, this thesis aims to provide as much useful information as possible to psychologists designing the next version of the ADOS. All unusual patterns and trends from the results of the autoencoders are cataloged. It is possible that this information could shed light

on the way that different tests within the diagnostic could be fused for the sake of efficiency.

CHAPTER TWO

BACKGROUND

2.1 Introduction

This section is an overview of the field of machine learning and the background mathematical and algorithmic techniques needed to comprehend the research methods used in this thesis.

2.2 Machine Learning

Machine learning is artificial intelligence research branch that involves a combination of statistics, mathematics, and programming. Machine learning algorithms train a computer to correctly execute a task on its own, without explicit programming. There are three types of machine learning tasks: supervised learning, unsupervised learning, and reinforcement learning.

2.3 Supervised Learning

Supervised learning involves training a machine learning algorithm to find a target function that reliably maps an input (or set of inputs) to an output (or set of outputs). The algorithm is trained with a dataset that contains the correct outputs. This dataset is typically referred to as the training set. During training, the algorithm increases the accuracy of its target function iteratively. Once the algorithm trained, the efficiency of the algorithm is tested on a new dataset called

the test set. There are two common types of supervised learning problems: classification and regression.

2.3.1 Classification

In a classification problem, an algorithm attempts to map an input to a categorical variable. The goal of classification is to figure out which category an input falls into, given a set of options. An algorithm is judged by the accuracy by which it can correctly predict which group an input should place too.

2.3.2 Regression

Regression tasks involve mapping an input to a continuous variable. In this case, the algorithm attempts to predict the correct output value. The accuracy of the algorithm is judged by the average error between the network's output predictions and the correct values.

2.4 Regularization

Regularization, in machine learning, refers to a process by which overfitting can be avoided. Overfitting occurs when a model adapts too closely to the peculiarities of an individual data set, rather than learning a target function that can be used for different data sets of the same category.

2.5 Neural Network

Neural networks are formally referred to as Artificial Neural Networks (ANN). They are a class of machine learning algorithm that has risen steadily in popularity over the previous decade. Training neural networks to solve problems is a process that is often referred to as “deep learning” colloquially. A neural network consists of connections of units or nodes called artificial neurons arranged in layers.

Neural networks can describe as computational graphs, which place mathematical operations into nodes that connect to each other. When the output of one node is the input of another, an arrow can be drawn from the first node to the second[8]. Every neural network diagram displays the computational graph for that network. Nodes in a neural network are referred to as neurons. There are many different variations of neural networks that currently used for various tasks.

2.5.1 Deep Feedforward Networks

The most basic version of a neural network is called a feedforward neural network, or a multilayer perceptron. A multilayer perceptron consists of artificial neurons arranged in sequential layers. The first layer of the network is comprised of the data, which is the input for the model. Each neuron typically corresponds to a dimension or variable in that dataset. The first layer takes the form of a matrix, which is transformed by matrix multiplication with a set of numbers referred to as weights that are often randomly initialized. A further level of

transformation can be provided to introduce non-linearity if an activation function is implemented. The neurons in the following layer of the neural network represent the transformed version of the first layer, following matrix multiplication with weights. Through this process, data is continually transformed in each consecutive layer, until it reaches the final layer or the output layer. This process collectively referred as a forward pass [8].

In supervised learning, the output of a neural network is checked for accuracy through a specified criterion. An algorithm is then implemented to adjust the weights of each layer by a small amount, with the goal of producing a more accurate output. Another forward pass is then conducted, and this process repeats in many iterations until a neural network has “learned” how to transform data to make consistently accurate predictions.

2.6 Tensor

A tensor is a mathematical object that can be used to represent any N-dimensional data structure. A tensor is analogous to a container for data that can apply mathematical operations. A1-dimensional tensor is a vector; a 2-dimensional tensor is a matrix, etc. Machine learning algorithms involve mathematical operations on data contained within tensors.

1. One-dimensional tensor = vector.

$$\begin{bmatrix} 2 \\ 5 \\ 1 \end{bmatrix}$$

2. Two-dimensional tensor = matrix.

$$\begin{bmatrix} 1 & 5 & 7 \\ 4 & 4 & 3 \\ 2 & 6 & 9 \end{bmatrix}$$

2.7 Matrix Multiplication

If matrix A is a matrix with dimensions of n x m and matrix B is a matrix of m x p dimensions, then the product of A X B is a matrix of n x p dimensions. Each entry in the new matrix is given by multiplying the entries of a row in A by a column in B and summing the results. Matrix multiplication is not a commutative operation: the product of A X B does not equal the product of B X A. Additionally, matrix multiplication can only work if the column number in the first matrix matches the row number in the second matrix [9].

$$\begin{bmatrix} 2 & 4 & 5 \\ 1 & 0 & 7 \end{bmatrix} \begin{bmatrix} x & a \\ y & b \\ z & c \end{bmatrix} = \begin{bmatrix} 2x + 4y + 5z & 2a + 4b + 5c \\ x + 7z & a + 7c \end{bmatrix}$$

2.8 Hadamard Product

The Hadamard Product is a mathematical operation that is conducted between two matrices. It can only be calculated for two matrices of the same

dimensions. The Hadamard Product of two matrices is a matrix in which each element is the product of corresponding elements in the first two matrices.

$$\begin{bmatrix} 2 & 7 \\ 3 & 5 \end{bmatrix} \odot \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} (2*1) & (7*1) \\ (3*0) & (5*0) \end{bmatrix} = \begin{bmatrix} 2 & 7 \\ 0 & 0 \end{bmatrix}$$

2.9 Activation Function

Activation functions are transformations that influence the output from a layer of neurons in a neural network. Activation functions can act as gates that block or allow neurons pass their values to the next layer based on a threshold, or they can transform the output of a neuron based on a continuous or discrete limit.

Activation functions are vital in neural networks because they introduce non-linearity into the network. Besides, without activation functions between layers, each neural network layer would be a linear transformation of the previous layer. If this is the case, the target function between input and output is limited in its complexity, and the network likely fails to converge when tasked with solving a difficult non-convex optimization problem. By introducing non-linearity into a neural network training process, activation functions allow neural networks to find significantly more complex methods of mapping an input to an output [10].

2.10 Loss Function

Loss functions are a crucial part of neural networks. Loss function can also be referred to as cost functions. A loss function is a mathematical representation of the gap between a network's predictions and the correct values. The value of a loss function will always be positive, as it represents the inconsistency between forecasts and right values. Many different loss functions can be used in different situations [11]. For example, standard loss functions for regression problems are mean squared error and L1 loss. There is no way to quickly know which loss function will provide the best results for a given neural network experiment. Experimenting with multiple viable loss functions is a good practice for smaller scale problems where training time is not too cumbersome.

If a neural network has at least one hidden layer, its loss function will almost certainly be non-convex. For clarity, a convex function has a single global minimum. Minimizing a convex loss function is relatively simple, as an algorithm needs merely to descend towards a single global minimum. Unfortunately, neural networks have non-convex loss functions that are marked by multiple local minima, rather than a single global minimum.

2.11 Optimizer

If a neural network minimizes a loss function to the best of its abilities, it must acquire a way to find the lowest point on a very complex non-convex loss

function. Finding a minimum point is a complicated task, and this is the purpose of optimization algorithms [12].

Optimization algorithms can be separated into first-order optimization algorithms and second-order optimization algorithms. First-order optimization algorithms minimize a loss function concerning its gradient. A gradient is merely a vector containing the partial derivatives of a function. In other words, the gradient is the multivariable equivalent of a derivative in single variable calculus, which is the slope or instantaneous rate of changing the output regarding its input. Second order methods use the Hessian, which is the multivariable equivalent of the second derivative. A Hessian can be as the gradient of a function's gradient. First-order optimization methods are significantly less computationally costly than second-order optimization methods, and the following explanations and algorithms covered are all first-order optimization methods [13].

2.11.1 Gradient Descent

All of the popular first-order optimization algorithms use an approach known as gradient descent. Gradient descent optimizers, following a forward pass, calculate the gradient of the loss function concerning the network's parameters, mainly the weights. This gradient is calculated with an algorithm called backpropagation, which is a computational implementation of reverse mode differentiation. After the slope calculated, the weights are then adjusted in small steps in the opposite direction of the loss function gradients [14]. Then

another forward pass is calculated, and the loss function's gradient is again calculated, and weights are then adjusted. The goal of gradient descent algorithms is to descend along the slope to find an optimal solution.

2.11.2 Learning Rate

The size of each adjustment, or step along the gradient, is called the learning rate of an optimizer. If the learning rate is too small, an optimizer might take too long to train, as its progress towards an optimal solution will take too long. If the learning rate is too large, the optimizer might "overshoot" the local region that leads to a minimum and skips over that area of the loss function entirely. Finding the best learning rate is often a matter of intuition and experimentation [13].

2.11.2 Batch Size

The batch size of a neural network in training refers to the number of data samples that are sent through the network in a single forward pass to calculate the gradient of the loss function. A forward pass of all data samples in a neural network referred to as an epoch [11].

2.11.3 Stochastic Gradient Descent

The equation for stochastic gradient descent is:

$$\theta = \theta - \alpha * \nabla J(\theta)$$

In this equation, θ refers to the parameters updated. α refers to the learning rate or the size of the adjustment. ∇J(θ) refers to the gradient of the loss

function concerning the gradient. Collectively, α·∇J(θ) refers to the update of the parameters by the optimizer. In other words, stochastic gradient descent updates the parameters opposite the gradient of the loss function scaled by a constant called the learning rate [14].

2.11.4 Momentum

If a gradient descends much more steeply in one dimension than in others, stochastic gradient descent tends to oscillate when updating, and often fails to drop in the correct amount towards a local minimum. Momentum is a technique that can be added to stochastic gradient descent to address this. Momentum mitigates oscillation and emphasizes the correct direction for the algorithm to descend in. It does this by incorporating information from the last update into the current update [15].

In mathematical terms, if the update of stochastic gradient descent is called v, then the new update can be formulated via the following equation:

$$v_t = \mu * v_{t-1} + \alpha * \nabla J(\theta)$$

In this equation, μ refers to a fraction of the previous update. μ is often a value near 1. Simply put, the last update, or a scaled version almost equal the previous update is added to the learning rate times the gradient to create the new update. The algorithm then updates the parameters with the equation [14]:

$$\theta = \theta - v_t$$

or

$$\theta = \theta - (\mu * v_{t-1} + \alpha * \nabla J(\theta))$$

In simpler terms, momentum works much as it does in its classical physics definition. As the algorithm descends the gradient, it theoretically descends faster and faster. Momentum increases for terms with gradients pointing in the same direction and reduces the rate of updates for terms with gradients leading in opposite directions, which effectively accelerates the algorithms descent along the gradient and minimizes the oscillation of the optimizer [15].

2.12 Backpropagation

Neural networks are made computationally tractable with the implementation of the back-propagation algorithm. Backpropagation is a neural network specific implementation of a technique called reverse mode differentiation, which allows for the rapid calculation of derivatives [16].

A partial derivative shows how altering one value affects a-partially dependent value. Within the context of a computational graph, it shows the expansive effects of altering different node. To compute partial derivatives between nodes that are not directly connected, one must sum over all possible paths from one node to another while multiplying the partial derivatives of each edge of a path together [17]. This is one way to think about how the chain rule in multivariable calculus works.

This problem here is that the number of paths between different nodes explodes as a neural network gets larger, which quickly makes a typical application of the multivariate chain rule computationally intractable. Forward mode differentiation and reverse mode differentiation are two techniques that help address this. Both techniques merge paths together at each node, rather than summing over all possible paths individually. Forward mode differentiation starts at input to the graph, and tracks how it affects every node forward. The derivative of every node with respect to the input is calculated. Once this is done, the derivative of the output with respect to every input has been calculated. Reverse mode differentiation starts at the graphs output, and then moves backwards, giving the derivative of the output with respect to every node in one calculation [16].

If a graph has 100 inputs and one output, forward mode differentiation would have to move through the graph 100 times to get the derivative of the output with respect to all outputs. Reverse mode differentiation is able to get this same derivative while moving through the graph only once. For most neural networks, reverse mode differentiation is exponentially faster. The backpropagation algorithm, which applies reverse mode differentiation to neural networks, makes most neural networks computationally tractable.

CHAPTER THREE

AUTISM DIAGNOSTIC OBSERVATION SCHEDULE

3.1 Introduction

This section describes each module of the Autism Diagnostic Observation Schedule (ADOS) in detail. The section communicates the origin and parameters of the data used in this thesis. It includes the purpose of each module and the concepts that it covers. Each variable gets explained thoroughly. The section contains the significance of different scores for each variable, and lists the shortened encodings used in the data.

3.2 Autism Diagnostic Observation Schedule (ADOS)

ADOS is a tool for diagnosing and assessing autism. The procedure consists of a sequence of structured and semi-structured tasks that involve social interaction between the inspector and the participant. It is uesed to evaluate the potential for autism spectrum disorders within an individual [18].

3.3 ADOS Modules

ADOS modules are activities of social communicative sequences, structured situation, and unstructured situations that allow the examiner to

observe whether any behaviors occur that determine the presence of autism and other pervasive developmental disorders. The activities are tasks runs by the examiner to allow for a consistent observation of the participant. These activities contain several observationa items. The items are features of behavior that the inspector focuses on detecting throughout the task. The examiners rate each observation according to the coding rules and specific coding standards [19].

The ADOS has four modules which are Module 1, Module 2, Module 3 , and Module 4. The examiners select it based on the individual's expressive language level or chronological age level. Each module can be administered in the range of 45 minutes to 100 minutes. It is suitable for children and adults of different ages and language levels reaching non-speaking to fluently [19].

In general Module 1 and Module 2 are designed for children who are not verbally fluent. Module 3 and Module 4 developed for participants who are verbally fluent. However, the difference between Module 3 and Module 4 is that Module 3 observes through play along with interview questions to collect information about social communication, while Module 4 entirely depends on interview questions and conversation [19].

The table below shows the different ages and language levels between the modules [19].

Table 1. ADOS Modules Ages and Language Levels

Language level	Chronological age range	ADOS Module
No speech/ simple phrase	31 months and older	Module 1
Phrase speech not verbally fluent	Any age	Module 2
Fluent speech	Children/ Adolescent Usually under 16	Module 3
Fluent speech	Adolescent / Adult	Module 4

As mentioned earlier, the examiner runs different tasks on the participant for observation which are call activities. The table below shows a list of the activities between modules [19].

Table 2. Modules Activities

Module 1	Module 2	Module 3	Module 4
1. Free Play 2. Response to Name 3. Response to Joint Attention 4. Buble play 5. Anticipation of a Routine With objects 6. Responsive social smile 7. Anticipation of a Social Routine 8. Functional and Symbolic Imitation 9.Birthday Party 10.Snack	1. Construction Task 2. Response to Name 3. Make-Believe Play 4. Joint Interactive Play 5. Conversation 6. Response to Joint Attention 7. Demonstration Task 8. Description of Picture 9.Telling a Story From a Book 10. Free Play	1. Construction Task 2. Make-Believe Play 3. Joint Interactive Play 4. Demonstration Task 5. Description of a Picture 6. Telling a Story From a Book 7. Cartoons 8. Conversation and Reporting 9. Emotions 10. Social Difficulties and	1. Construction Task* 2. Telling a Story From a Book 3. Description of a Picture* 4. Conversation and Reporting 5. Current Work or School* 6. Social Difficulties and Annoyance 7. Emotions 8. Demonstration Task 9. Cartoons* 10.Break

	11.Birthday Party 12.Snack 13. Anticipation of a Routine with Objects 14. Buble Play	Annoyance 11. Break 12.Friends, Relationship and Marriage 13.Loneliness 14. Creating a Story	11. Daily Living* 12. Friends, Relationship and Marriage 13.Loneliness 14.Plans and Hopes 15. Creating a Story

***** optional

3.3.1 Module 1

Module 1 is dedicated to children ages 31 months and older who have language levels of no speech to a simple phrase or single words. This module also can be used for adolescent and adult who have the same language level.

Module 1 of the ADOS consists of 10 activities with 29 observation items. Each item is rated by the examiner according to 5 main categories, "Language and Communication," "Reciprocal Social Interaction," "Play," "Stereotyped Behaviors and Restricted Interests," and "Other Abnormal Behaviors." [18]

The table below shows Observation Items for Language and Communication section. In this section, there are 8 observation items for the examiner to rate according specific coding standards [19].

Table 3. Module 1 Section 1 Observation Items

Section A: Language and Communication	
Observation Items	**Abbreviation**
A1. Overall level of non-echoed spoken language	(OLANG)
A2. Frequency of Spontaneous Vocalization Directed to Others	(FVOC)

A3. Intonation of Vocalizations or Verbalizations	(INTON)
A4. Immediate Echolalia	(IECHO)
A5. Stereotyped/Idiosyncratic Use of Words or Phrases	(STEREO)
A6. Use of Another's Body	(UOTHER)
A7. Pointing	(POINT)
A8. Gestures	(GEST)

The table below shows Observation Items for Reciprocal Social Interaction section. In this section, there are 12 observation items for the examiner to rate according specific coding standards [19].

Table 4. Module 1 Section 2 Observation Items

Section B: Reciprocal Social Interaction	
Observation Items	**Abbreviation**
B1. Unusual Eye Contact	(UEYE)
B2. Responsive Social Smile	(SSMILE)
B3. Facial Expressions Directed to Others	(FACEO)
B4. Integration of Gaze and Other Behaviors During Social Overtures	(GZSOV)
B5. Other Behaviors During Social Overtures, Shared Enjoyment in Interaction	(SHRNJ)
B6. Response to Name	(RNAME)
B7. Requesting	(REQ)
B8. Giving	(GIVE)
B9. Showing	(SHOW)
B10. Spontaneous Initiation of Joint Attention	(SIJNT)
B11. Response to Joint Attention	(RJNT)
B12. Quality of Social Overtures	(QSOV)

The table below shows Observation Items for Play section. In this section, there are 2 observation items the examiner evaluates according to the specific coding standards [19].

Table 5. Module 1 Section 3 Observation Items

Section C: Play	
Observation Items	**Abbreviation**
C1. Functional Play with Objects	(FPLAY)
C2. Imagination/Creativity	(IMGCR)

The table below shows Observation Items for Stereotyped Behaviors and Restricted Interests section. In this section, there are 4 observation items for the examiner to rate according specific coding standards [19].

Table 6. Module 1 Section 4 Observation Items

Section D: Stereotyped Behaviors and Restricted Interests	
Observation Items	**Abbreviation**
D1. Unusual Sensory Interest in Play Material/Person	(USENS)
D2. Hand and Finger and Other Complex Mannerisms	(OMAN)
D3. Self-Injurious Behavior	(SELFINJ)
D4. Unusually Repetitive Interests or Stereotyped Behaviors	(URBEH)

The table below shows Observation Items for Other Abnormal Behaviors section. In this section, there are 3 observation items for the examiner to rate according specific coding standards [19].

Table 7. Module 1 Section 5 Observation Items

Section E: Other Abnormal Behaviors	
Observation Items	**Abbreviation**
E1.Overactivity	(ACTIVE)
E2. Tantrums, Aggression, Negative or Disruptive Behavior	(AGG)
E3. Anxiety	(ANXTY)

3.3.2 Module 2

Module 2 is dedicated to children of any age who have language level of some phrase speech but not verbally fluent. This module also can be used for adolescent and adult who have the same language level. Children under age of 3 who are verbally fluent are examined with this module even if they meet Module 3 standards.

Module 2 of the ADOS consists of 14 activities with 28 observation items. Each item is rated by the examiner according to 5 main categories, “Language and Communication,” ” Reciprocal Social Interaction,” ” Play,” ” Stereotyped Behaviors and Restricted Interests,” and ” Other Abnormal Behaviors.”[19]

The table below shows Observation Items for Language and Communication section. In this section, there are 7 observation items for the examiner to rate according specific coding standards [19].

Table 8. Module 2 Section 1 Observation Items

Section A: Language and Communication	
Observation Items	**Abbreviation**
A1. Overall level of non-echoed spoken language	(OLANG)
A2. Speech Abnormalities Associated With Autism	(SPABN)
A3. Immediate Echolalia	(IECHO)
A4. Stereotyped/Idiosyncratic Use of Words or Phrases	(STEREO)
A5. Conversation	(CONVS)
A6. Pointing	(POINT)
A7. Descriptive, Conventional, Instrumental or informational Gestures	(DGEST)

The table below shows Observation Items for Reciprocal Social Interaction section. In this section, there are 12 observation items for the examiner to rate according specific coding standards [19].

Table 9. Module 2 Section 2 Observation Items

Section B: Reciprocal Social Interaction	
Observation Items	**Abbreviation**
B1. Unusual Eye Contact	(UEYE)
B2. Facial Expressions Directed to Others	(FACEO)
B3. Shared Enjoyment in Interaction	(SHRNJ)
B4. Response to Name	(RNAME)
B5. Showing	(SHOW)
B6. Spontaneous Initiation of Joint Attention	(SIJNT)
B7. Response to Joint Attention	(RJNT)
B8. Quality of Social Overtures	(QSOV)
B9. Amount of Social Overtures	(ASOV)
B10. Quality of Social Response	(QSRES)
B11. Amount of Reciprocal Social Communication	(ARSOC)
B12. Overall Quality of Rapport	(OQRAP)

The table below shows Observation Items for Play section. In this section, there are 2 observation items for the examiner to rate according specific coding standards [19].

Table 10. Module 2 Section 3 Observation Items

Section C: Play	
Observation Items	**Abbreviation**
C1. Functional Play with Objects	(FPLAY)
C2. Imagination/Creativity	(IMGCR)

The table below shows Observation Items for Stereotyped Behaviors and Restricted Interests section. In this section, there are 4 observation items for the examiner to rate according specific coding standards [19].

Table 11. Module 2 Section 4 Observation Items

Section D: Stereotyped Behaviors and Restricted Interests	
Observation Items	**Abbreviation**
D1. Unusual Sensory Interest in Play Material/Person	(USENS)
D2. Hand and Finger and Other Complex Mannerisms	(OMAN)
D3. Self-Injurious Behavior	(SELFINJ)
D4. Unusually Repetitive Interests or Stereotyped Behaviors	(URBEH)

The table below shows Observation Items for Other Abnormal Behaviors section. In this section, there are 3 observation items for the examiner to rate according specific coding standards [19].

Table 12. Module 2 Section 5 Observation Items

Section E: Other Abnormal Behaviors	
Observation Items	**Abbreviation**
E1.Overactivity	(ACTIVE)
E2. Tantrums, Aggression, Negative or Disruptive Behavior	(AGG)
E3. Anxiety	(ANXTY)

3.3.3 Module 3

Module 3 is dedicated to children and adolescent between 4 to 15 years of age who have verbally fluent language levels. As mentioned previously, children

under age of 2 who are verbally fluent will be examined by Module 2 even if they meet Module 3 standards.

Module 3 of the ADOS consists of 14 activities with 28 observation items. Each item is rated by the examiner according to 5 main categories, “Language and Communication,” ” Reciprocal Social Interaction,” ” Imagination,” ” Stereotyped Behaviors and Restricted Interests,” and ” Other Abnormal Behaviors.” [19]

The table below shows Observation Items for Language and Communication section. In this section, there are 9 observation items for the examiner to rate according specific coding standards [19].

Table 13. Module 3 Section 1 Observation Items

Section A: Language and Communication	
Observation Items	**Abbreviation**
A1. Overall level of non-echoed spoken language	(OLANG)
A2. Speech Abnormalities Associated With Autism	(SPABN)
A3. Immediate Echolalia	(IECHO)
A4. Stereotyped/Idiosyncratic Use of Words or Phrases	(STEREO)
A5. Offers Information	(OINFO)
A6. Asks for Information	(AINFO)
A7. Reporting of Events	(REPRT)
A8. Conversation	(CONVS)
A9. Descriptive, Conventional, Instrumental or informational Gestures	(DGEST)

The table below shows Observation Items for Reciprocal Social Interaction section. In this section, there are 10 observation items for the examiner to rate according specific coding standards [19].

Table 14. Module 3 Section 2 Observation Items

Section B: Reciprocal Social Interaction	
Observation Items	**Abbreviation**
B1. Unusual Eye Contact	(UEYE)
B2. Facial Expressions Directed to Others	(FACEO)
B3.Language Production and Linked Nonverbal Communication	(LLNVC)
B4. Shared Enjoyment in Interaction	(SHIRJ)
B5. Comment on Others' Emotions/Empathy	(EMPTH)
B6. Insight Into Typical Social Situations and Relationships	(INSIG)
B7. Quality of Social Overtures	(QSOV)
B8. Quality of Social Response	(QSRES)
B9. Amount of Reciprocal Social Communication	(ARSOC)
B10. Overall Quality of Rapport	(OQRAP)

The table below shows Observation Items for Imagination section. In this section, there is 1 observation item for the examiner to rate according specific coding standards [19].

Table 15. Module 3 Section 3 Observation Items

Section C: Imagination	
Observation Items	**Abbreviation**
C2. Imagination/Creativity	(IMGCR)

The table below shows Observation Items for Stereotyped Behaviors and Restricted Interests section. In this section, there are 5 observation items for the examiner to rate according specific coding standards [19].

Table 16. Module 3 Section 4 Observation Items

Section D: Stereotyped Behaviors and Restricted Interests	
Observation Items	**Abbreviation**
D1. Unusual Sensory Interest in Play Material/Person	(USENS)
D2. Hand and Finger and Other Complex Mannerisms	(OMAN)
D3. Self-Injurious Behavior	(SELFINJ)
D4. Excessive Interest in or References to Unusual or Highly Specific Topics or Objects or Repetitive Behaviors	(TOPIC)
D5. Compulsions or Rituals	(RITL)

The table below shows Observation Items for Other Abnormal Behaviors section. In this section, there are 3 observation items for the examiner to rate according specific coding standards [19].

Table 17. Module 3 Section 5 Observation Items

Section E: Other Abnormal Behaviors	
Observation Items	**Abbreviation**
E1.Overactivity/Agitation	(ACTIVE)
E2. Tantrums, Aggression, Negative or Disruptive Behavior	(AGG)
E3. Anxiety	(ANXTY)

3.3.4 Module 4

Module 4 is dedicated to older adolescent and adult of age16 years and older who have verbally fluent language level. As mentioned previously there are slight differences between Module 4 and 3. Module 4 depends on interview questions and conversation while Module 3 depends on the play along with interview questions.

Module 4 of the ADOS consists of 15 activities with 31 observation items. Each item is rated by the examiner according to 5 main categories, “Language and Communication,” ” Reciprocal Social Interaction,” ” Imagination,” "Stereotyped Behaviors and Restricted Interests,” and ” Other Abnormal Behaviors.”[19]

The table below shows Observation Items for Language and Communication section. In this section, there are 10 observation items for the examiner to rate according specific coding standards [19].

Table 18. Module 4 Section 1 Observation Items

Section A: Language and Communication	
Observation Items	**Abbreviation**
A1. Overall level of non-echoed spoken language	(OLANG)
A2. Speech Abnormalities Associated With Autism	(SPABN)
A3. Immediate Echolalia	(IECHO)
A4. Stereotyped/Idiosyncratic Use of Words or Phrases	(STEREO)
A5. Offers Information	(OINFO)

A6. Asks for Information	(AINFO)
A7. Reporting of Events	(REPRT)
A8. Conversation	(CONVS)
A9. Descriptive, Conventional, Instrumental or informational Gestures	(DGEST)
A10. Emphatic or Emotional Gestures	(EGEST)

The table below shows Observation Items for Reciprocal Social Interaction section. In this section, there are 12 observation items for the examiner to rate according specific coding standards [19].

Table 19. Module 4 Section 2 Observation Items

Section B: Reciprocal Social Interaction	
Observation Items	**Abbreviation**
B1. Unusual Eye Contact	(UEYE)
B2. Facial Expressions Directed to Others	(FACEO)
B3.Language Production and Linked Nonverbal Communication	(LLNVC)
B4. Shared Enjoyment in Interaction	(SEI)
B5. Communication of Own Affect	(CAFF)
B6. Comment on Others' Emotions/Empathy	(EMPTH)
B7. Insight Into Typical Social Situations and Relationships	(INSIG)
B8. Responsibility	(RESP)
B9. Quality of Social Overtures	(QSOV)
B10. Quality of Social Response	(QSRES)
B11. Amount of Reciprocal Social Communication	(ARSOC)
B12. Overall Quality of Rapport	(OQRAP)

The table below shows Observation Items for Imagination section. In this section, there are 1 observation item for the examiner to rate according specific coding standards [19].

Table 20. Module 4 Section 3 Observation Items

Section C: Imagination	
Observation Items	**Abbreviation**
C2. Imagination/Creativity	(IMGCR)

The table below shows Observation Items for Stereotyped Behaviors and Restricted Interests section. In this section, there are 5 observation items for the examiner to rate according specific coding standards [19].

Table 21. Module 4 Section 4 Observation Items

Section D: Stereotyped Behaviors and Restricted Interests	
Observation Items	**Abbreviation**
D1. Unusual Sensory Interest in Play Material/Person	(USENS)
D2. Hand and Finger and Other Complex Mannerisms	(OMAN)
D3. Self-Injurious Behavior	(SELFINJ)
D4. Excessive Interest in or References to Unusual or Highly Specific Topics or Objects or Repetitive Behaviors	(TOPIC)
D5. Compulsions or Rituals	(RITL)

The table below shows Observation Items for Other Abnormal Behaviors section. In this section, there are 3 observation items for the examiner to rate according specific coding standards [19].

Table 22. Module 4 Section 5 Observation Items

Section E: Other Abnormal Behaviors	
Observation Items	**Abbreviation**
E1.Overactivity/Agitation	(ACTIVE)
E2. Tantrums, Aggression, Negative or Disruptive Behavior	(AGG)

E3. Anxiety	(ANXTY)

3.4 ADOS Coding Conventions

The table below shows coding conventions applied to ADOS modules coding sections [19].

Table 23. ADOS Coding Convention Modules

Coding Rate	Convention
0	No evidence of abnormality in the behavior according to the specifications. The absence of defect as specified does not imply the expression is normal.
1	The behavior is slightly or mildly unusual yet not an exact match to the type specified but not grossly abnormal.
2	The behavior is an exact match to the defined abnormality. At this level, the severity of coding defect will vary according to the item.
3	The behavior is blatantly abnormal and interferes with the assessment, or the behavior is limited therefore making it impossible to make a qualitative evaluation.
4	It shows that test subject displayed no instinctive use of words or anything close to a word during the entire ADOS administration
7	an abnormal behavior yet, it not covered by the other ratings.
8	The behavior in question did not occur and/or the rating is not applicable

9	The item cannot rate due to some reason other than that listed for a code 8, such as if examiner commits an error and does not administer a particular ADOS activity. This code provides examiners with a consistent way to rate items that cannot measure and allowing it to use for any item.

The coding ratings range from 0 (abnormality specified is absent) through 2 or 3 (defect determined is present). Optimal or expected performance detailed in the code for a rating of 0, and partial, minimal, and/or wavering production described in the rest of the codes. In clinical practice, codes of 0, 1, 2, and 3 are the ones often considered most. Ratings of 8 or 9 can use for anything that is not applicable or that otherwise cannot be coded, as such it should be used sparingly because of all the missing data, including 8s or 9s, must change to scores of 0 on the algorithm [19].

3.5 ADOS DATA

The ADOS records of the participants can found in the Autism Genetic Resource Exchange (AGRE).

3.5.1 AGRE

AGRE is a non-profit DNA repository and family registry. It has a database of biomaterials, genotypic and phenotypic data that is made available to researchers. It is a central shared resource for studies on autism and related

disorders. It has data from 1700+ families with 3300+ affected individuals with an Autism Spectrum Disorder. It contains Clinical and biomaterial data for over 500 twin families [20]. However, accessing the data requires approval. Approval information can be found in Appendix A.

3.5.2 Data information

The AGRE provided all records following the approvel prosses. Patient evaluation results for each module were included.

Module 1 data contains1055 individuals, Module 2 contains 602 individuals, Module 3 contains 1158 individuals, and Module 4 contains 264 individuals. The data was collected between the years 2002 and 2015.

CHAPTER FOUR

MODEL DESIGN

4.1 Introduction

This section covers all details relevant to the design of the model used in this thesis. The information includes details about the Python machine learning ecosystem, the type of model used, the unique architectural aspects of the model used, the reasoning behind these aspects, and their implementation in math and code.

4.2 Python Machine Learning Tools Used

4.2.1 Numpy

Numpy is a Python package used for scientific computing. It contains functions that allow for robust array computations, advanced math such as linear algebra and Fourier transformations, and can act as an efficient container of different types of data. Numpy can very easily integrate with other Python libraries and frameworks for specific purposes, such as neural networks and machine learning [21].

4.2.1 Pandas

Pandas is a Python library that provides useful data structures and tools for data analysis. In some ways, pandas is analogous to Excel for Python [22].

4.2.2 Scikit-Learn

Scikit-learn is a Python library for machine learning tasks. It provides many functions for regression, classification, dimensionality reduction, clustering, model selection, and preprocessing [23].

4.2.3 Matplotlib

Matplotlib is a Python library that is used for convenient graphing and data visualizations [24].

4.2.4 Jupyter Notebook

A Jupyter Notebook is a document, which can contain both computer code in specific languages (Python included) and rich text elements (charts, graphs, and so forth). In the Jupyter Notebook, code is can be written and executed in the same document. Lines of code can divide into modular cells that can be executed individually. Jupyter notebooks run in a server-client application that operated through a web browser online or offline. When a notebook document is opened, a kernel corresponding to the language of choice (in this case, the iPython kernel) gets launched. When a cell of code is executed, the kernel runs the computations and produces the output underneath the block [25].

4.3 PyTorch

PyTorch is a Python constructed deep learning research framework, which offers flexibility and high speed. It consists of several libraries such as tensor computation that uses strong GPU integration, which is the key of PyTorch components. Also, it contains tape-based autograd that supports all differentiable tensor operations. Moreover, it contains a neural networks library that is tightly integrated with automatic differentiation and optimization, which features famous optimizers such as Stochastic Gradient Descent, RMSprop, and Adam [26].

4.3.1 Pytorch Components

4.3.1.1 Tensor Computations

Tensors are the core of PyTorch. They are n-dimensional arrays and form the fundamental building blocks for many algorithms including neural networks. PyTorch tensors are functionally equivalent to numpy arrays , but can be operated upon by GPU's provide significantly faster calculations for many machine-learning tasks, including neural networks. A PyTorch tensor can cast into a GPU-specific data type, and a model comprised of these tensors can be trained on a GPU significantly faster than on a CPU [27].

4.3.1.2 Autograd Mechanics

The autograd package in PyTorch provides the functionality of automatic differentiation. Automatic differentiation is used to quickly evaluate the derivative

of a function specified by a computer program by repeatedly applying the chain rule to elementary operations [27].

Neural networks are typically trained with back-propagation using a unique form of automatic differentiation called reverse mode automatic differentiation. When using the autograd package, each forward pass of a neural network defines a computational graph with tensors as nodes. Each tensor is wrapped in a Variable object that represents a node in the computational graph [27]. Back-propagation through this graph allows for the quick computation of gradients. PyTorch autograd implementation is not unique; however, it is swift compared to what is offered by many competing frameworks.

4.3.1.3 Torch.nn

Torch.nn is a library in PyTorch, which allows the user to build neural networks conveniently. It provides a high degree of abstraction over raw computational graphs, which makes it simpler to develop and organize neural networks of all sizes based on layer types, dimensions, and activation functions [26]. Torch.nn is similar to Keras, which providesthe same functionality to users of other frameworks like TensorFlow and Theano.

The nn package contains Modules, which are similar to neural network layers. The modules take input and compute an output, and it can hold an internal state if needed. Also, nn contains a multitude of loss functions and activations for timely implementation. All commonly implemented loss and

activation functions are present in PyTorch, allowing for easy experimentation with different combinations [27].

It is possible to build custom nn modules that are more complex than existing modules, and this can be done by creating a subclass and defining a specific forward operation that receives inputs and computes outputs.

4.3.1.4 Torch.optim

The optim package provides an abstraction for optimization algorithms rather than manually writing code to update weights for Variables in a model. Powerful and popular optimization algorithms such as AdaGrad and Adam can conveniently implement in a model [27].

4.3.2 PyTorch Unique Features and Advantages

4.3.2.1 Tight Integration with Python Language

PyTorch has multiple of advantages over other deep learning frameworks. It is more tightly integrated with the Python language than competitors such as TensorFlow. PyTorch is underlying C/C++ code is tailored explicitly for conveniently working with Python. By comparison, TensorFlow was first built entirely in C/C++ and then bound to Python. Using PyTorch is like using other Python libraries, such as numpy or scipy. Layers can be written from scratch in Python and incorporated into PyTorch models. The code can execute on a line-by-line basis, which allows for easier debugging. PyTorch also displays the

dimensionality of all tensors within a model, which also assists with debugging [27].

PyTorch can easily be extended with custom Python code. This allows users to build models of any level of uniqueness and complexity while still enjoying PyTorch's lightning fast computational speed. The tighter integration of PyTorch with the Python language makes innovation within PyTorch significantly more comfortable than in other frameworks, like TensorFlow.

4.3.2.2 Dynamic Computational Graphs

PyTorch defines its computational graphs dynamically, which contrasts from the static graph methods used by other Python deep learning frameworks. All deep learning frameworks define neural networks as directed acyclic graphs. Most structures generate a graph once, and then continually re-use it when training the model. PyTorch, on the other hand, creates a graph with each forward pass. This dynamic computational graph generation enables a higher degree of flexibility in training in many ways, such as allowing inputs of different sizes dynamically within the same data set. Any aspect of the computational graph can be adjusted on the fly with control flow, from the dimensions of hidden layers to the number of hidden layers, to the activation function used between layers [27].

4.3.2.3 Exceptional Speed and Flexibility

PyTorch also has a low framework overhead and support for acceleration libraries featured by hardware manufacturers. Integration with Nvidia's acceleration libraries CUDA and CuDNN is a notable perk of PyTorch.

PyTorch is also exceptionally fast for training small and large neural networks. Memory allocation is also hyper-efficient, as PyTorch contains custom memory allocators for GPU's [27]. PyTorch tensors use the same memory allocation as numpy arrays, which allows for data to be converted between PyTorch and numpy structures significantly faster than it can in TensorFlow. Within the subjective experience of this experiment, PyTorch trained small neural networks markedly quicker than TensorFlow did.

4.3.3 CUDA

CUDA is a parallel computing platform developed by NVIDIA for use on GPU's. CUDA allows for significantly faster computing with GPU's and is tightly integrated with several popular machine learning frameworks, including PyTorch [28].

4.4 Autoencoder

Autoencoders are a particular type of feedforward neural network. They recreate the model's input as their target. Naturally, the output layer of an autoencoder has the same dimensions as the input layer. Autoencoders have

hidden layers smaller than either the input or output layer. An autoencoder compresses its inputs in the hidden layers and then attempts to reconstruct the input as its output [29].

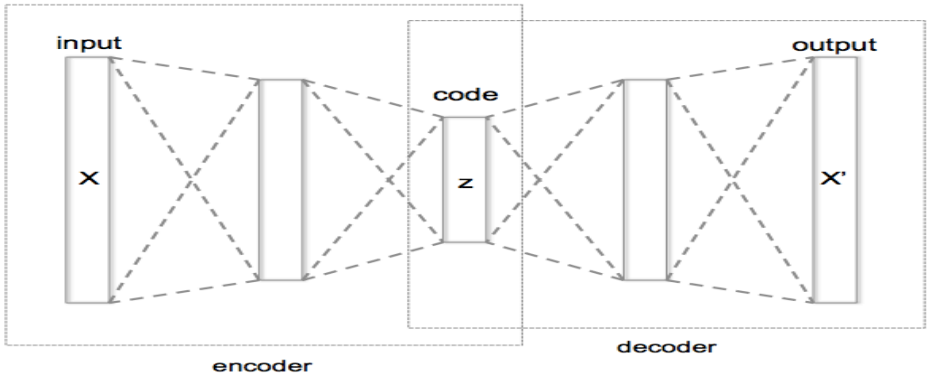


Figure 1. Autoencoder [30]

The first half of an autoencoder is called the encoder. The encoder includes the input layer and any hidden layers up to the smallest hidden layer. The most smallest hidden layer in an autoencoder represents the most compressed representation of the input. The first half of an autoencoder is called the encoder because by compressing the input, it is effectively creating a concise, encoded representation of the input [31].

The second half of an autoencoder is called the decoder. The decoder includes progressively wider hidden layers following the encoded layer, all the way until the output layer while attempting to reconstruct the input. This section of an autoencoder is called a decoder because it tries to recreate the original

input from its encoded representation, without “knowing” what the original input looked like, therefore acting as a “decoder” of sorts [31].

Autoencoders have very few practical applications currently. These neural networks mostly used for experimental causes. They are attractive to researchers because of their potential for large-scale, intelligent dimensionality reduction. If neural networks are used for large-scale unsupervised learning, autoencoders might be able to function as a memory of sorts by compressing the massive amounts of data an algorithm might parse through into a condensed representation, and then deconstructing that representation to restore the full dimensionality of the original data.

4.4.1 Reasons for Using Autoencoder

Applying autoencoders to the Autism Diagnostic Observation Schedule is a novel application of autoencoders. Since autoencoders are a type of neural network, their effectiveness scales very well with the amount of data available. Neural networks also scale in effectiveness with more significant computational resources. Since computing power is increasing, and the amount of available ADOS data is increasing as well, autoencoders have the potential to be an algorithm of considerable interest in the future. Other dimensionality reduction methods such as principal components analysis (PCA) have been applied to the ADOS by other researchers in recent years [32]. These algorithms, as well as all different “shallow” machine learning algorithms, are increasingly less relevant

with every year, whereas neural networks are frequently interesting, relevant, and efficient. For this reason, autoencoders chosen for this thesis.

4.5 Autoencoder with Special Connection

4.5.1 Special Connections with Binary Mask Matrix

Typically, the computational graph of a neural network features fully connected layers. That is, each node in a layer connects to each node in the following layer. In effect, that means that each neuron in one layer has an impact on every neuron in the next layer. In this experiment, each neuron in the input layer corresponds to a variable within ADOS. Similarly, each neuron in the hidden layer of the autoencoder corresponds to a new variable in the reduced length diagnostic. Every neuron in the input layer that connects with a neuron in the hidden layer represents the content of an ADOS items impacting the content of reduced diagnostic items.

The ADOS features different sections with significantly different subject material. If an autoencoder with fully connected layers were to be used, that would indicate that questions from all sections should be indiscriminately combined in a reduced diagnostic. Given the conceptual differences between sections, this does not seem like a logical approach. Therefore, it is necessary to enforce specific connections between layers in the autoencoder.

In the experiment's design, the reduced length diagnostic has the same sectional structure as the original diagnostic. The only difference is that each section has fewer variables. Each variable in the original diagnostic influences every variable in the corresponding section of the reduced diagnostic, and no other variables in the reduced diagnostic whatsoever.

In other words, connections between neurons in each layer occur in discrete groups, rather than ubiquitously. This is achieved through the use of a binary adjacency matrix called a mask. The mask matrix exclusively consists of 0's and 1's. The 0's and 1's correspond with connections that are to be removed and enforced respectively.

During each forward pass for the network, the binary mask matrix is applied to the weight matrix before the weights and the inputs are multiplied together. The mask matrix and the weight matrix have the same dimensions, and they are combined with an element wise multiplication operation known as the Hadamard product. In this operation, corresponding values in each matrix are multiplied together. The result of this operation is the preservation of certain weights, and the removal of others by setting them to a value of 0. Once the weight matrix is combined with the inputs through matrix multiplication, every weight corresponding to desired connections is preserved and every weight corresponding to undesired connections is set to 0. The diagrams below are Hadamard Product of Weight Matrix and Binary Mask Matrix for Module 1, 2, 3 and 4. Find the other Hadamard Product Diagrams in Appendix B.

Module1 Hadamard Product for 2920 Dimension

$$\begin{bmatrix} w_{1,1} & \cdots & w_{1,5} & w_{1,6} & \cdots & w_{1,14} & w_{1,15} & \cdots & w_{1,20} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{8,1} & \cdots & w_{8,5} & w_{8,6} & \cdots & w_{8,14} & w_{8,15} & \cdots & w_{8,20} \\ w_{9,1} & \cdots & w_{9,5} & w_{9,6} & \cdots & w_{9,14} & w_{9,15} & \cdots & w_{9,20} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{20,1} & \cdots & w_{20,5} & w_{20,6} & \cdots & w_{20,14} & w_{20,15} & \cdots & w_{20,20} \\ w_{21,1} & \cdots & w_{21,5} & w_{21,6} & \cdots & w_{21,14} & w_{21,15} & \cdots & w_{21,20} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{29,1} & \cdots & w_{29,5} & w_{29,6} & \cdots & w_{29,14} & w_{29,15} & \cdots & w_{29,20} \end{bmatrix} \odot \begin{bmatrix} 1_{1,1} & \cdots & 1_{1,5} & 0_{1,6} & \cdots & 0_{1,14} & 0_{1,15} & \cdots & 0_{1,20} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1_{8,1} & \cdots & 1_{8,5} & 0_{8,6} & \cdots & 0_{8,14} & 0_{8,15} & \cdots & 0_{8,20} \\ 0_{9,1} & \cdots & 0_{9,5} & 1_{9,6} & \cdots & 1_{9,14} & 0_{9,15} & \cdots & 0_{9,20} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0_{20,1} & \cdots & 0_{20,5} & 1_{20,6} & \cdots & 1_{20,14} & 0_{20,15} & \cdots & 0_{20,20} \\ 0_{21,1} & \cdots & 0_{21,5} & 0_{21,6} & \cdots & 0_{21,14} & 1_{21,15} & \cdots & 1_{21,20} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0_{29,1} & \cdots & 0_{29,5} & 0_{29,6} & \cdots & 0_{29,14} & 1_{29,15} & \cdots & 1_{29,20} \end{bmatrix}$$

$$= \begin{bmatrix} w_{1,1} & \cdots & w_{1,5} & 0_{1,6} & \cdots & 0_{1,14} & 0_{1,15} & \cdots & 0_{1,20} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{8,1} & \cdots & w_{8,5} & 0_{8,6} & \cdots & 0_{8,14} & 0_{8,15} & \cdots & 0_{8,20} \\ 0_{9,1} & \cdots & 0_{9,5} & w_{9,6} & \cdots & w_{9,14} & 0_{9,15} & \cdots & 0_{9,20} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0_{20,1} & \cdots & 0_{20,5} & w_{20,6} & \cdots & w_{20,14} & 0_{20,15} & \cdots & 0_{20,20} \\ 0_{21,1} & \cdots & 0_{21,5} & 0_{21,6} & \cdots & 0_{21,14} & w_{21,15} & \cdots & w_{21,20} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0_{29,1} & \cdots & 0_{29,5} & 0_{29,6} & \cdots & 0_{29,14} & w_{29,15} & \cdots & w_{29,20} \end{bmatrix}$$

Figure 2. Module 1 Hadamard Product for 2920 Dimensions

Module2 Hadamard Product for 2819 Dimension

$$\begin{bmatrix} w_{1,1} & \cdots & w_{1,4} & w_{1,5} & \cdots & w_{1,13} & w_{1,14} & \cdots & w_{1,19} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{7,1} & \cdots & w_{7,4} & w_{7,5} & \cdots & w_{7,13} & w_{7,14} & \cdots & w_{7,19} \\ w_{8,1} & \cdots & w_{8,4} & w_{8,5} & \cdots & w_{8,13} & w_{8,14} & \cdots & w_{8,19} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{19,1} & \cdots & w_{19,4} & w_{19,5} & \cdots & w_{19,13} & w_{19,14} & \cdots & w_{19,19} \\ w_{20,1} & \cdots & w_{20,4} & w_{20,5} & \cdots & w_{20,13} & w_{20,14} & \cdots & w_{20,19} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{28,1} & \cdots & w_{28,4} & w_{28,5} & \cdots & w_{28,13} & w_{28,14} & \cdots & w_{28,19} \end{bmatrix} \odot \begin{bmatrix} 1_{1,1} & \cdots & 1_{1,4} & 0_{1,5} & \cdots & 0_{1,13} & 0_{1,14} & \cdots & 0_{1,19} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1_{7,1} & \cdots & 1_{7,4} & 0_{7,5} & \cdots & 0_{7,13} & 0_{7,14} & \cdots & 0_{7,19} \\ 0_{8,1} & \cdots & 0_{8,4} & 1_{8,5} & \cdots & 1_{8,13} & 0_{8,14} & \cdots & 0_{8,19} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0_{19,1} & \cdots & 0_{19,4} & 1_{19,5} & \cdots & 1_{19,13} & 0_{19,14} & \cdots & 0_{19,19} \\ 0_{20,1} & \cdots & 0_{20,4} & 0_{20,5} & \cdots & 0_{20,13} & 1_{20,14} & \cdots & 1_{20,19} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0_{28,1} & \cdots & 0_{28,4} & 0_{28,5} & \cdots & 0_{28,13} & 1_{28,14} & \cdots & 1_{28,19} \end{bmatrix}$$

$$= \begin{bmatrix} w_{1,1} & \cdots & w_{1,4} & 0_{1,5} & \cdots & 0_{1,13} & 0_{1,14} & \cdots & 0_{1,19} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{7,1} & \cdots & w_{7,4} & 0_{7,5} & \cdots & 0_{7,13} & 0_{7,14} & \cdots & 0_{7,19} \\ 0_{8,1} & \cdots & 0_{8,4} & w_{8,5} & \cdots & w_{8,13} & 0_{8,14} & \cdots & 0_{8,19} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0_{19,1} & \cdots & 0_{19,4} & w_{19,5} & \cdots & w_{19,13} & 0_{19,14} & \cdots & 0_{19,19} \\ 0_{20,1} & \cdots & 0_{20,4} & 0_{20,5} & \cdots & 0_{20,13} & w_{20,14} & \cdots & w_{20,19} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0_{28,1} & \cdots & 0_{28,4} & 0_{28,5} & \cdots & 0_{28,13} & w_{28,14} & \cdots & w_{28,19} \end{bmatrix}$$

Figure 3. Module 2 Hadamard Product for 2819 Dimensions

Module3 Hadamard Product for 2819 Dimension

$$\begin{bmatrix} w_{1,1} & \cdots & w_{1,6} & w_{1,7} & \cdots & w_{1,13} & w_{1,14} & \cdots & w_{1,19} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{9,1} & \cdots & w_{9,6} & w_{9,7} & \cdots & w_{9,13} & w_{9,14} & \cdots & w_{9,19} \\ w_{10,1} & \cdots & w_{10,6} & w_{10,7} & \cdots & w_{10,13} & w_{10,14} & \cdots & w_{10,19} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{19,1} & \cdots & w_{19,6} & w_{19,7} & \cdots & w_{19,13} & w_{19,14} & \cdots & w_{19,19} \\ w_{20,1} & \cdots & w_{20,6} & w_{20,7} & \cdots & w_{20,13} & w_{20,14} & \cdots & w_{20,19} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{28,1} & \cdots & w_{28,6} & w_{28,7} & \cdots & w_{28,13} & w_{28,14} & \cdots & w_{28,19} \end{bmatrix} \odot \begin{bmatrix} 1_{1,1} & \cdots & 1_{1,6} & 0_{1,7} & \cdots & 0_{1,13} & 0_{1,14} & \cdots & 0_{1,19} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1_{9,1} & \cdots & 1_{9,6} & 0_{9,7} & \cdots & 0_{9,13} & 0_{9,14} & \cdots & 0_{9,19} \\ 0_{10,1} & \cdots & 0_{10,6} & 1_{10,7} & \cdots & 1_{10,13} & 0_{10,14} & \cdots & 0_{10,19} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0_{19,1} & \cdots & 0_{19,6} & 1_{19,7} & \cdots & 1_{19,13} & 0_{19,14} & \cdots & 0_{19,19} \\ 0_{20,1} & \cdots & 0_{20,6} & 0_{20,7} & \cdots & 0_{20,13} & 1_{20,14} & \cdots & 1_{20,19} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0_{28,1} & \cdots & 0_{28,6} & 0_{28,7} & \cdots & 0_{28,13} & 1_{28,14} & \cdots & 1_{28,19} \end{bmatrix}$$

$$= \begin{bmatrix} w_{1,1} & \cdots & w_{1,6} & 0_{1,7} & \cdots & 0_{1,13} & 0_{1,14} & \cdots & 0_{1,19} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{9,1} & \cdots & w_{9,6} & 0_{9,7} & \cdots & 0_{9,13} & 0_{9,14} & \cdots & 0_{9,19} \\ 0_{10,1} & \cdots & 0_{10,6} & w_{10,7} & \cdots & w_{10,13} & 0_{10,14} & \cdots & 0_{10,19} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0_{19,1} & \cdots & 0_{19,6} & w_{19,7} & \cdots & w_{19,13} & 0_{19,14} & \cdots & 0_{19,19} \\ 0_{20,1} & \cdots & 0_{20,6} & 0_{20,7} & \cdots & 0_{20,13} & w_{20,14} & \cdots & w_{20,19} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0_{28,1} & \cdots & 0_{28,6} & 0_{28,7} & \cdots & 0_{28,13} & w_{28,14} & \cdots & w_{28,19} \end{bmatrix}$$

Figure 4. Module 3 Hadamard Product for 2819 Dimensions

Module4 Hadamard Product for 3119 Dimension

$$\begin{bmatrix} w_{1,1} & \cdots & w_{1,6} & w_{1,7} & \cdots & w_{1,13} & w_{1,14} & \cdots & w_{1,19} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{10,1} & \cdots & w_{10,6} & w_{10,7} & \cdots & w_{10,13} & w_{10,14} & \cdots & w_{10,19} \\ w_{11,1} & \cdots & w_{11,6} & w_{11,7} & \cdots & w_{11,13} & w_{11,14} & \cdots & w_{11,19} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{22,1} & \cdots & w_{22,6} & w_{22,7} & \cdots & w_{22,13} & w_{22,14} & \cdots & w_{22,19} \\ w_{23,1} & \cdots & w_{23,6} & w_{23,7} & \cdots & w_{23,13} & w_{23,14} & \cdots & w_{23,19} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{31,1} & \cdots & w_{31,6} & w_{31,7} & \cdots & w_{31,13} & w_{31,14} & \cdots & w_{31,19} \end{bmatrix} \odot \begin{bmatrix} 1_{1,1} & \cdots & 1_{1,6} & 0_{1,7} & \cdots & 0_{1,13} & 0_{1,14} & \cdots & 0_{1,19} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1_{10,1} & \cdots & 1_{10,6} & 0_{10,7} & \cdots & 0_{10,13} & 0_{10,14} & \cdots & 0_{10,19} \\ 0_{11,1} & \cdots & 0_{11,6} & 1_{11,7} & \cdots & 1_{11,13} & 0_{11,14} & \cdots & 0_{11,19} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0_{22,1} & \cdots & 0_{22,6} & 1_{22,7} & \cdots & 1_{22,13} & 0_{22,14} & \cdots & 0_{22,19} \\ 0_{23,1} & \cdots & 0_{23,6} & 0_{23,7} & \cdots & 0_{23,13} & 1_{23,14} & \cdots & 1_{23,19} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0_{31,1} & \cdots & 0_{31,6} & 0_{31,7} & \cdots & 0_{31,13} & 1_{31,14} & \cdots & 1_{31,19} \end{bmatrix}$$

$$= \begin{bmatrix} w_{1,1} & \cdots & w_{1,6} & 0_{1,7} & \cdots & 0_{1,13} & 0_{1,14} & \cdots & 0_{1,19} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{10,1} & \cdots & w_{10,6} & 0_{10,7} & \cdots & 0_{10,13} & 0_{10,14} & \cdots & 0_{10,19} \\ 0_{11,1} & \cdots & 0_{11,6} & w_{11,7} & \cdots & w_{11,13} & 0_{11,14} & \cdots & 0_{11,19} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0_{22,1} & \cdots & 0_{22,6} & w_{22,7} & \cdots & w_{22,13} & 0_{22,14} & \cdots & 0_{22,19} \\ 0_{23,1} & \cdots & 0_{23,6} & 0_{23,7} & \cdots & 0_{23,13} & w_{23,14} & \cdots & w_{23,19} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0_{31,1} & \cdots & 0_{31,6} & 0_{31,7} & \cdots & 0_{31,13} & w_{31,14} & \cdots & w_{31,19} \end{bmatrix}$$

Figure 5. Module 4 Hadamard Product for 3119 Dimensions

The first matrix, is the weight matrix. This second matrix, the binary mask matrix, is comprised of ones and zeros corresponding to the desired and undesired connections. Before the weights are applied to the inputs, the network will constrain the weights through element-wise multiplication between the mask and the weight matrices. The new weight matrix is the Hadamard product of the mask and weight matrices.

As can be seen in the diagram below, implementing the mask matrix in a forward pass erases all undesired connections. The only connections between nodes in layers are connections that correspond to specific sections of the ADOS. Find The diagrams in Appendix C.

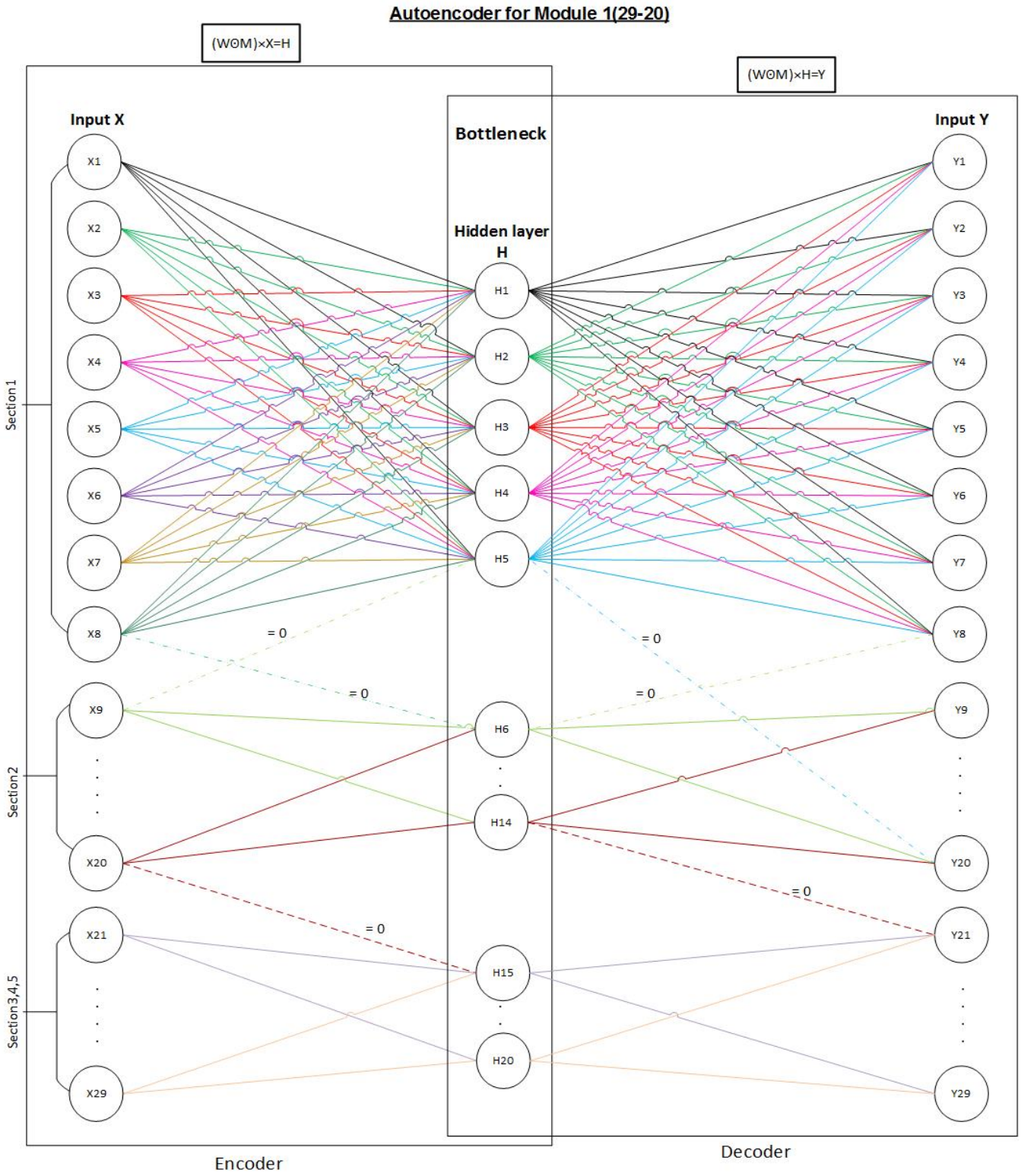


Figure 6. Module 1 Autoencoder with Hidden Layer of Size 20

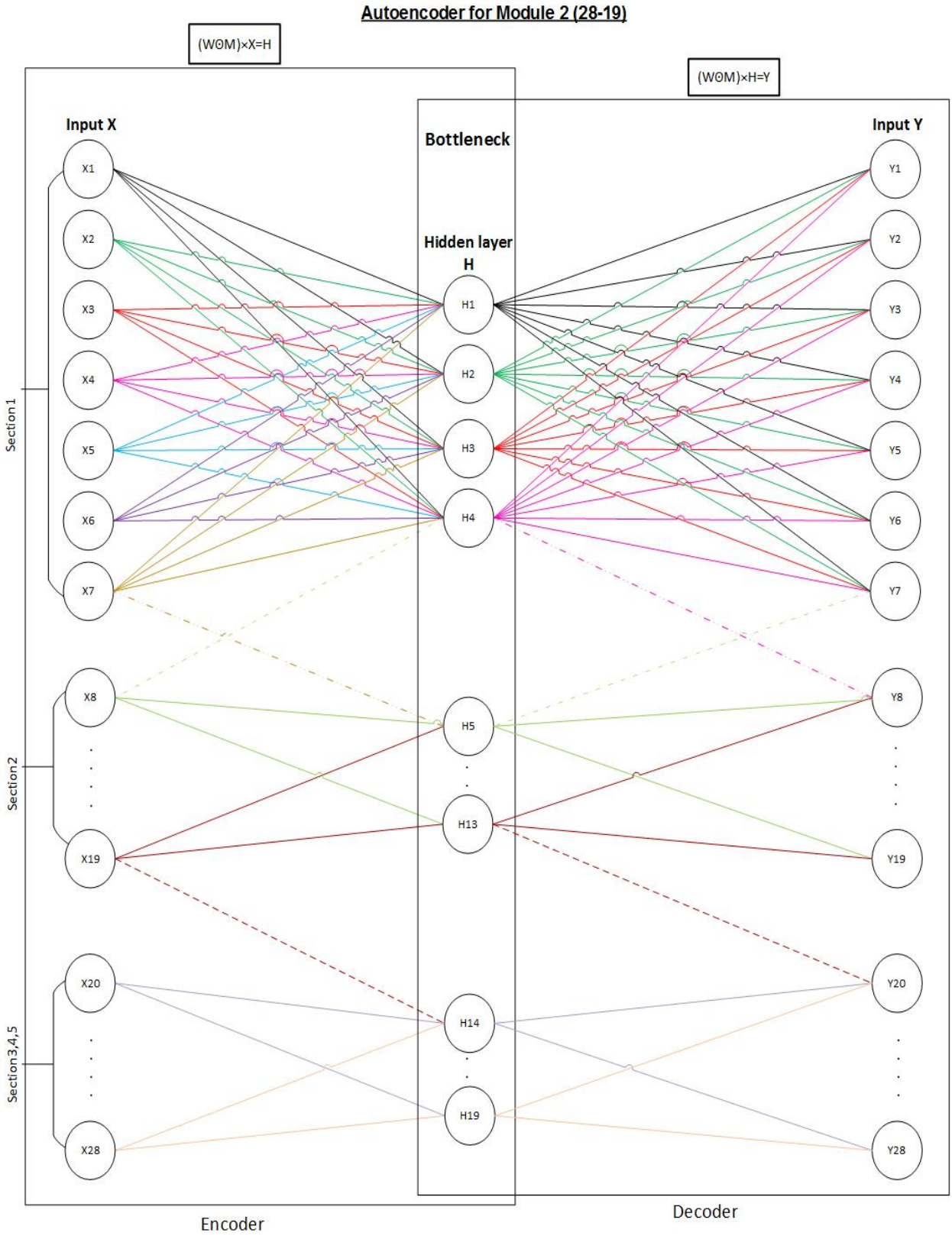


Figure 7. Module 2 Autoencoder with Hidden Layer of Size 19

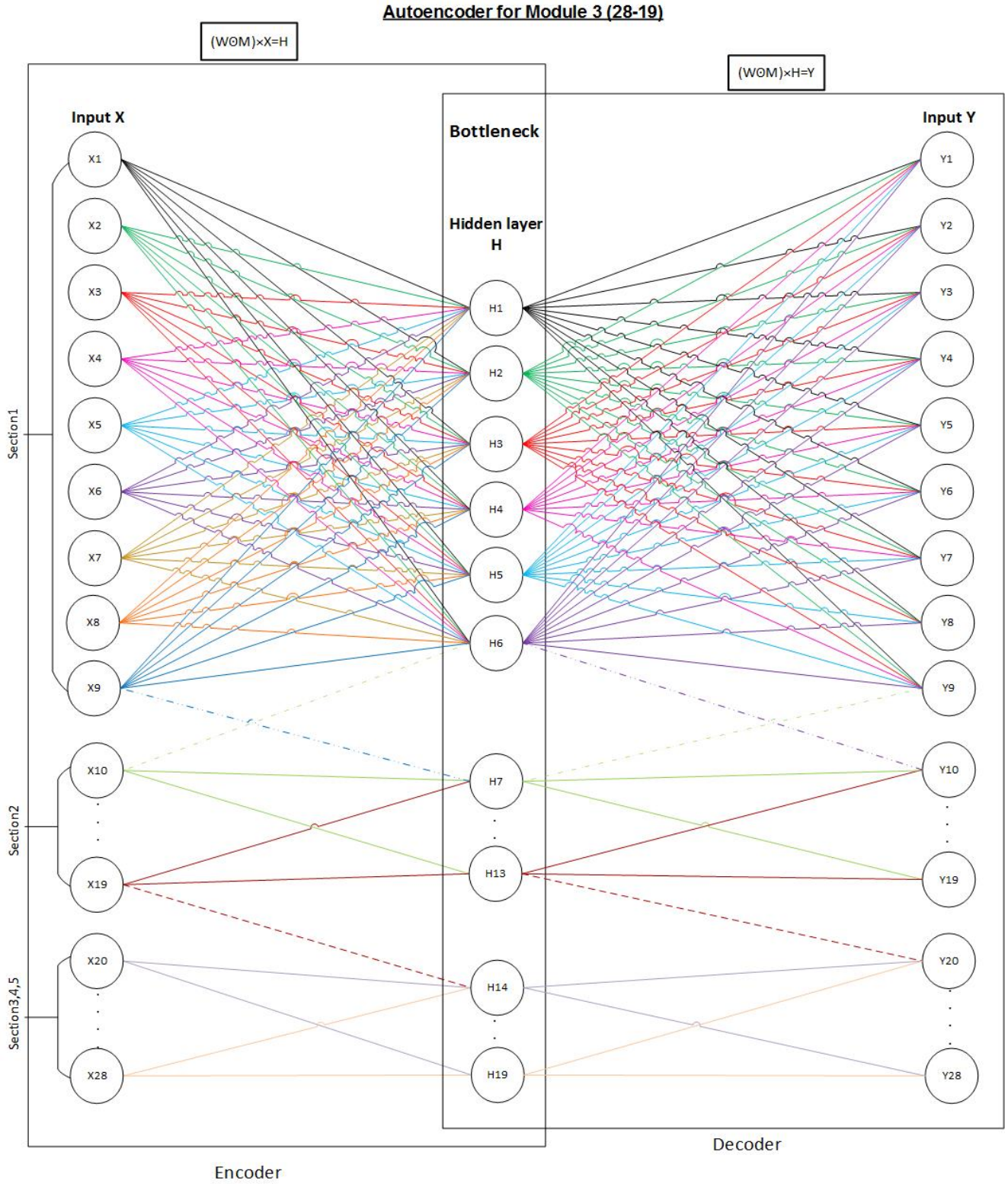


Figure 8. Module 3 Autoencoder with Hidden Layer of Size 19

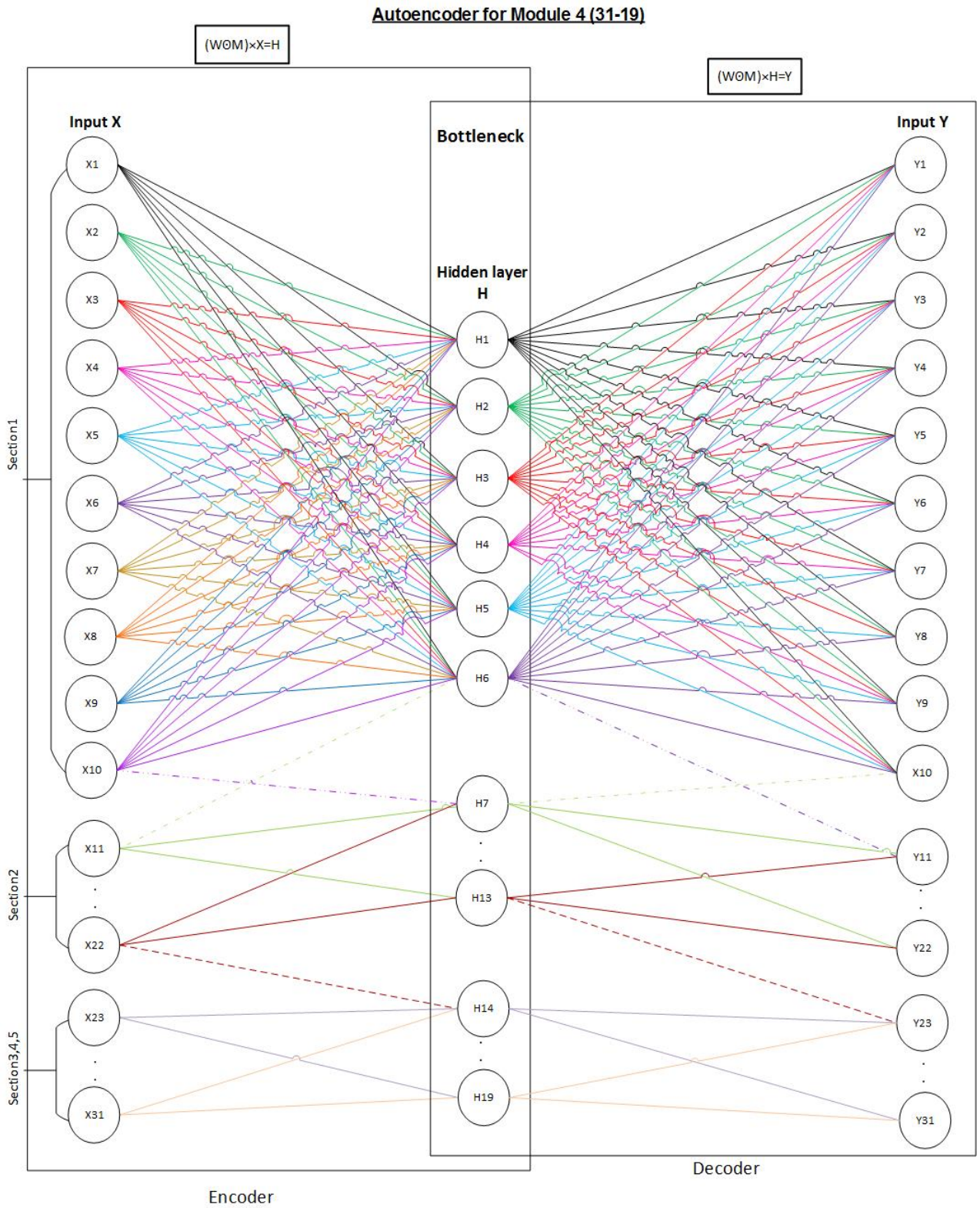


Figure 9. Module 4 Autoencoder with Hidden Layer of Size 19

4.5.2 Creating Mask in PyTorch

The binary mask used for the special connections between layers was originally created as a *numpy* array prior to its importation into PyTorch. PyTorch tensors are generally created from *numpy* arrays, so this intermediate step was required. The mask tensor was created as a PyTorch “FloatTensor” in order to match with the tensor type of the model’s weights. The tensor was then converted into a CUDA-specified tensor type stored on the GPU.

The mask for each autoencoder decoder layer is simply a transposition of the mask for each autoencoder encoder layer. For example, the mask that works for a reduction from 29 to 23 dimensions is simply a transposition of the mask that expands 23 dimensions for 29 dimensions. Therefore, for the sake of convenience mask, each decoder mask was simply created by transposing the corresponding encoder mask.

4.5.3 Incorporating Mask into Model

PyTorch allows a user to build custom modules for its nn package. Modules in nn function as neural network layers. Building a new module requires creating a custom subclass with a unique forward pass that receives Variables as inputs and computes Variables as outputs.

This experiment features custom layers that feature the aforementioned binary mask to selectively enforce and eliminate connections in a manner consistent with the sectional structure of the Autism Diagnostic Observation

Schedule. The forward pass of each layer is altered to update the weights by taking the Hadamard Product, or element-wise multiplication product, of the weight matrix and the binary mask matrix before applying the weights to the input Variable via matrix multiplication.

Otherwise, the custom module functions exactly like a standard feedforward neural network layer. The layer takes an array as an input and multiplies it through a weight matrix. The output of this operation becomes the input of the next layer in the neural network. In the case of PyTorch, the input array is a Torch tensor for GPU computation wrapped as a differentiable object called a Variable. The tensor inside this variable represent the data, which is multiplied by a weight matrix in which the undesired connections have already been eliminated through element-wise multiplication with the mask matrix or in other words the Hadamard Product.

The custom module for the masked layers created by altering the source code for PyTorch regular linear layers used in a standard feedforward neural network. These layers involve matrix multiplication between weights and inputs. Before this step, the code was added to modify the weight matrix through element-wise multiplication with the corresponding mask matrix.

4.6 Autoencoder Architecture

The input layer of each module's architecture corresponds to the number of variables in that module. Since autoencoders attempt the reconstruct their input as their output, the output layer for each autoencoder has to have the same number of neurons as the input layer. The only things to determine, in terms of autoencoder architecture, were the number of hidden layers, and the size of each hidden layer, with the smallest hidden layer being the most important, since it represents the encoded representation of the diagnostic.

When evaluating the architecture of the autoencoder, it is critical to keep the project's larger purpose in mind. The purpose of this thesis is to use an autoencoder to develop a viable reduced length diagnostic, rather than find the smallest possible encoded representation that an autoencoder can encode and then accurately decode. Likewise, the goal is to subjectively comprehend the variables that go into the encoded representation, and enforce specialized connections between layers to stay true to the sectional structure of the diagnostic.

In light of the goal of making sense of the variables that go into the reduced diagnostic, it made sense to have as few hidden layers as possible. First, autoencoders with three hidden layers were tested for Module 1. An intermediate encoding layer and an intermediate decoding layer helped bridge the gaps between the input, encoding, and reconstructed output. While this approach yielded a very accurate reconstruction, it was difficult to keep track of

the underlying concepts behind variables, or neurons, in the reduced diagnostic within the context of the original variables of the diagnostic. For that reason, autoencoders with a single hidden layer were chosen, since it is easier to keep track of the presence of input variables within the reduced diagnostic.

4.6.1 Autoencoder Pipeline

4.6.1.1 Building Autoencoders in PyTorch

The *nn* abstraction method was used to build the autoencoders. *NN* allows for a model to built as a combination of layers and activation functions. Since the masked layers had built as custom *nn* modules, these layers could simply be stacked to form the autoencoder in Pytorch.

Over the course of this thesis, multiple autoencoders were trained for different reduced length diagnostics for each module. The dimensions of every autoencoder are listed below.

4.6.2 List of Autoencoder Architectures Tested

4.6.2.1 Module 1

Module 1 contains 29 variables that are measured. The input layer of each autoencoder for Module 1 therefore contains 29 neurons to represent these original 29 variables. The dimensions of the output layer of each autoencoder match the dimensions of the input layer. Over the course of this thesis, four separate autoencoder structures for this module were tested, each with a single

hidden layer representing the reduced diagnostic. The only differences between these four autoencoders were the connections and the hidden layer size. Hidden layer sizes of 20 neurons, 21 neurons, 22 neurons, and 23 neurons were tested. During an earlier phase of experimentation, hidden layer sizes slightly smaller than 20 neurons were tested. During these trials, the accuracy of the reconstruction was significantly worse. Therefore, nothing below 20 neurons was used in the final phase of the experiment for Module 1.

4.6.2.1 Modules 2, 3 and 4

Modules 2, 3, and 4 each had four autoencoders trained for them. All autoencoders consisted of a single hidden layer. Each module's experimentation involved training autoencoders with hidden layer dimensions of 19, 20, 21, and 22 neurons. The only difference between the autoencoders for the different modules involved the dimensions of the input and output layers. Module 2 contains 28 variables. Module 3 also contains 28 variables. Module 4 contains 31 variables. The input and output dimensions of their respective autoencoders contain numbers of neurons that match the number of variables.

4.6.3 Reasons for the Lower Limit in Dimensionality Reduction

The purpose of this experiment was to test the possibility of a coherent reduced version of the Autism Diagnostic Observation Schedule. Therefore, it did not make sense to decrease the dimensionality of the data as much as algorithmically possible, since a reduction of too large a magnitude would be

inconsistent with the project's goal. The smallest hidden layer sizes of the autoencoders tested in this experiment represent multiple variables eliminated from each section of the corresponding module. Reducing dimensionality further seemed excessive from a subjective standpoint within the context of finding a viable reduced diagnostic size. The lowest dimensionality reductions involved removing multiple variables per section. Removing even more variables would not have left enough variables to cover the diversity of different concepts on the ADOS.

4.6.4 Selecting Regression Over Classification

This experiment was treated as a regression problem rather than a classification problem. Regression problems involve making predictions about a continuous variable, whereas classification problems involve making predictions about categorical variables. In regression problems, loss functions involve errors between predictions and targets. In classification, loss functions attempt to optimize classification accuracy as a binary procedure.

The scores of different variables on the ADOS all take the form of ordinal variables. Only integer values exist, but the hierarchy of the integers has a meaning, unlike in a strict classification problem. Because of the ordinal nature of ADOS scores, reconstructing the input was treated as a high dimensional regression problem, with appropriate loss functions. To make sure that the

reconstructions were comparable to the input, values were rounded to the nearest integer.

4.7 Hyperparameter Tuning

The most important part of training this autoencoder was picking the correct hyperparameters for the model, which refers to the choice of loss function, optimizer, and activation function. While intuition can certainly guide which loss functions, optimizers, and activation functions are tested out, there are multiple options for each of these hyperparameters that are plausible. The only way to determine which hyperparameter settings allow for the best convergence, and therefore most accurate reconstruction of an encoded, reduced representation is to test out different combinations.

4.7.1 Tested Loss Functions

The loss functions that were tested areaveraged version of Least Absolute Error (L1 loss) and Least Squared Error (L2 loss). L2 loss is also called Mean Squared Error Loss (MSELoss). Both of these functions are typically used for regression tasks. Since the numbers that represent scores on the different tests on the ADOS have an ordinal relationship, this problem was treated as a regression problem instead of a discrete classification problem. At the end, reconstructed values could be rounded in order to represent the integer nature of the scores for each variable on the diagnostic.

1- L1 Loss

L1loss is simple error measurement typically used for regression problems. It takes the absolute value of each error, sums them up. In PyTorch, users have the option to then average this sum on a per-sample basis. In other words, this the average absolute value difference between the network's predictions and the correct results [33]. The formula for averaged L1 loss is as follows:

$$L1 = \frac{1}{n}\sum_{i=1}^{n}\left|y^{(i)} - \hat{y}^{(i)}\right|$$

2- Mean Squared Error Loss (MSELoss)

Mean squared error, which is also commonly referred to as L2 loss, is a very simple error measurement. It takes the difference between the estimation and the correct value of each, and then squares it. These squares are all summed together, and then divided by the sample size [33]. The formula for mean squared error is as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2$$

4.7.1.1 Reasons for Testing Both L1 & L2 Loss

At first glance, it might seem like it is unnecessary to test the performance of both L1 and L2 loss functions. Since both results were averaged out, they basically seem to be measuring the same thing. The only difference is

that mean squared error or L2 loss measures a squared version of the average absolute value error or L1 loss.

While this is true, the shape that both loss functions take for a given experiment might be different. Since each of the functions will take the form of a very complex multivariable equation, differences in function shape can make a huge difference.

For any given problem, one function might have a gradient that is easier for an optimizer to navigate than the other. Not only that, but one function might work better for one set of optimizers and activation functions, and the other function might work better for another set of optimizers and activation functions. There is no way to know beforehand which will work best, so different combinations need to be tested.

4.7.2 Tested Activation Functions

The activation functions that were tested are Rectified Linear Units (ReLU), Exponential Linear Units (ELU), Scaled Exponential Linear Units (SELU), and Softplus.

1- Rectified Linear Units (ReLU)

Rectified Linear Units or ReLU is a simple activation function that fires a neuron if its output is greater than 0, and does not activate a neuron if its value is less than 0. ReLU is the most popular activation function used by researchers in the field of deep learning. While it can

help implement a nonlinear target function that allows a network to converge upon a very difficult to reach solution to a complex problem, ReLU can be prone to backfiring in an inconvenient manner from time to time. A ReLU neuron with a large gradient might cause its weights to update in a way that permanently set that neuron's value at 0 or below, causing it to "die" and never fire again. This problem can be avoided by setting a proper learning rate that is not too high [34]. The threshold for ReLU is very simple:

$$f(x) = \max(0, x)$$

Graphically, ReLU [35] looks like this

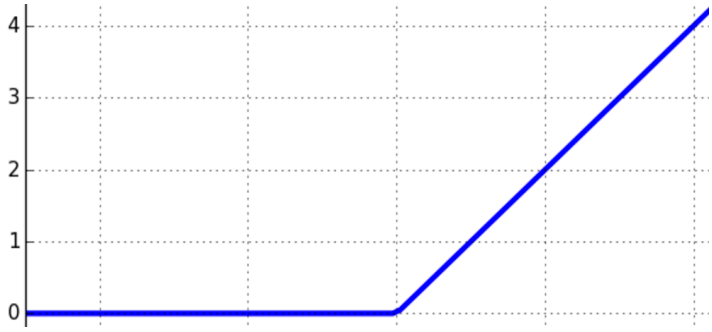


Figure 10. ReLu Activation Function

2- Exponential Linear Units (ELU)

For positive values, exponential linear units function are identical to rectified linear units. For negative values, the function has a boundary of -1 for an α value of 1.0, which the default used by PyTorch. Exponential linear units have a greater percentage of neurons fire, which ends up helping a neural network train faster [36]. The formula for exponential linear units is as follows:

$$f(x)\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

Graphically, exponential linear units look like this:

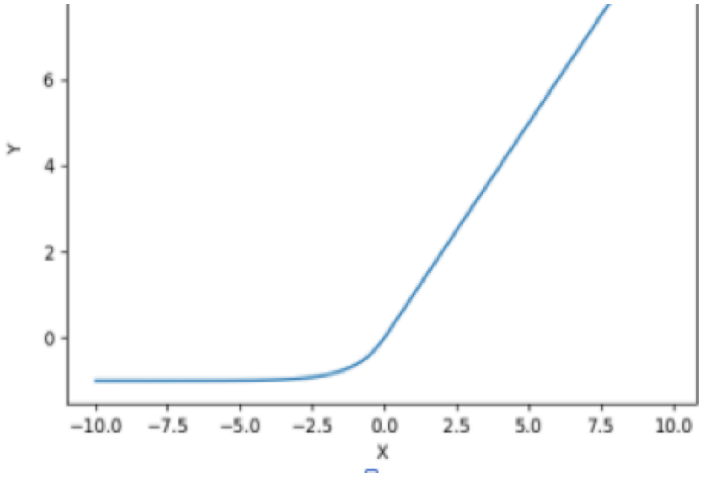


Figure 11. ELU Activation Function

3- Scaled Exponential Linear Units (SELU)

SELU works very similarly to exponential linear units (ELU). The only difference is that the exponential linear units are scaled based on two constant parameters not subject to gradients, α & λ. The default value for α is 1.6732632423543772848170429916717. The default value for λ is 1.0507009873554804934193349852946. The formula for scaled exponential linear units is as follows [37]:

$$selu(x) = \lambda \begin{cases} x & if x > 0 \\ \alpha e^x - \alpha & if x \leq 0 \end{cases}$$

Graphically, SELU looks like this:

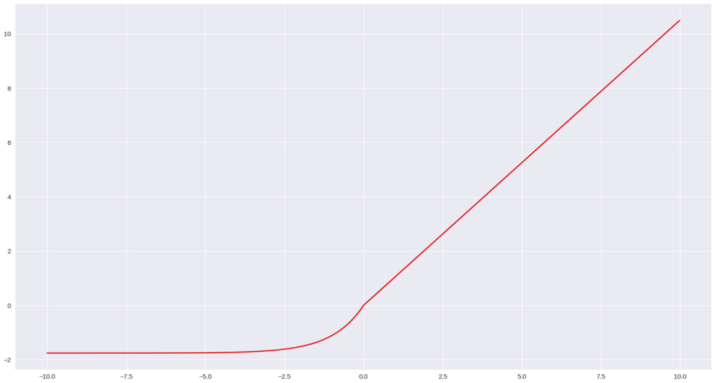


Figure 12. SELU Activation Function

4- Softplus

The softplus activation function is simply a smooth approximation of the rectified linear unit activation function (ReLU). The softplus activation function is:

$$f(x) = \ln(1 + e^x)$$

Graphically, softplus looks very similar to ReLu, but is smooth [38].

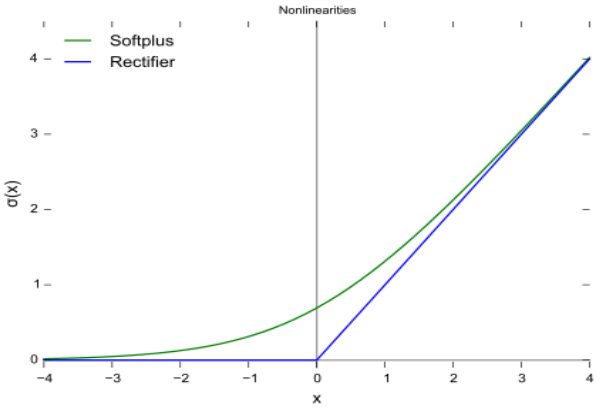


Figure13. Softplus Activation Function

4.7.3 Tested Optimizers

The optimizers that were tested are Adaptive Moment Estimation (Adam), RMSProp and Average Stochastic Gradient Descent (ASGD). In order to understand Adam and RMSProp, it is necessary to understand the optimizer Adaptive Gradient Boost.

4.7.3.1 Adaptive Gradient Boost (Adagrad)

Adagrad attempts to improve upon traditional stochastic gradient descent by normalizing updates for each parameter. After it is done, parameters with larger gradients have smaller updates, and parameters with smaller or less used gradients have larger updates [14]. The formulation of Adagrad is shown below:

$$0_{t+1,i} = 0_{t,i} - \frac{n}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

The problem with Adagrad is that the cache eventually becomes too large and ceases to function properly. This process occurs because the squared gradients accumulated in the denominator, which is referred to as the cache. As the accumulated sum grows, the learning rate progressively shrinks until the algorithm learns at an infinitesimally small and computationally intractable rate [14].

1- RMSProp

The RMSProp optimization algorithm addresses the aforementioned weakness of adaptive gradient boost by decaying the size of the cache by

multiplying it by a constant fraction less than but close to 1. RMSProp also divides the learning rate for a given weight by a running average of the magnitudes of recent gradients for that weight [14].

$$cache_t = \gamma * cache_{t-1} + (1 - y) * \nabla J(\theta_t)^2$$

2- Adaptive Moment Estimation (Adam)

Adam optimizer improves RMSProp by incorporating information from past updates, by using momentum. It accumulates past squared gradients in an exponentially decaying average like RMSProp. In addition to this decaying average of squared gradients, Adam also accumulates an exponentially decaying average of previous gradients without squaring them. These two caches are running estimates of the mean and variance of past gradients, which are referred to as the first and second moments respectively. Adam then implements a bias correction to these estimates, and then uses this information to calculate new updates [14]. The Adam optimizer has the formulation:

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * J(\theta_t)$$

$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * \nabla J(\theta_t)^2$$

$$\theta_t = \theta_{t-1} - \frac{\alpha * m_t}{(\sqrt{v_t})}$$

3- ASGD (Averaged Stochastic Gradient Descent)

This algorithm represents an accelerated version of stochastic gradient descent. A normal stochastic gradient descent update is performed, and then an average is calculated efficiently using a recursive formula [39].

$$\overline{w}_t = \frac{1}{t - t_0} \sum_{i=t_{0+1}}^{t} w_t$$

This average is then used to influence future updates, which can accelerate training significantly in some cases.

$$w_{t+1} = (1 - \gamma_t \lambda) w_t - \gamma_t y_t x_t l'(y_t x_t w_t)$$

$$\overline{w}_{t+1} = \overline{w}_t + \mu_t (w_{t+1} - \overline{w}_t)$$

The averaging rate is:

$$\mu_t = 1/max\{1, t - t_0\}$$

4.7.4 Batch Size, Learning Rate and Scheduler Tuning

Different batch sizes were tested throughout the training process. The batch size of a neural network refers to the number of data samples that are simultaneously sent forward through the network at one time.

During hyperparameter tuning, different initial learning rates were tested for each optimizer. The learning rate determines the size of a weight update for the optimizer. The learning rates tried altered by a factor of 10, and ranged from .01 to .00001.

An algorithm called a scheduler, which dynamically adjusts the learning rate of an optimizer once descent on the loss function has stagnated, was also tested. Results with the scheduler, for all optimizers, were significantly worse than without a scheduler. This was true with all combinations of loss and activation functions, and was true regardless of the optimizer used.

The most likely explanation for this phenomenon involves local minima within the loss function. In all likelihood, the scheduler ended up getting stuck in a suboptimal local minimum during training. Since the scheduler decreases the learning rate once learning stagnates, it is very unlikely to “escape” a local minimum during gradient descent.

4.8 Data Pipeline Prior to Training.

4.8.1 Data Preprocessing

The Autism Diagnostic Observation schedule data, covering Module 1, Module 2, Module 3, and Module 4, was supplied in the form of Excel spreadsheets with a “.xlsx” file extension. The first step in the data pipeline was to convert these “.xlsx” files to comma separated value files “.csv”. This format imports into python more cleanly.

Prior to doing this, the files were edited, as there were a small but insignificant number of samples that did not contain full ADOS data. There were also several columns in the spreadsheet that were completely unnecessary to the project. These columns included patient identification numbers and the date at which the tests were administered. Besides this, the data required very little preprocessing.

4.8.2 Moving Data into Python Environment

The programming and machine learning part of this thesis was mostly conducted within the confines of a Jupyter notebook. The data was imported into

the Jupyter notebook with a library called *Pandas*. *Pandas* is a Python open source library that provides high performance data structures and data analysis tools for the purpose of data science and machine learning projects. *Pandas* library contains a function called “read_csv”, which takes the contents of a comma separated value file and imports it into a Python programming environment.

Following this, the data in the file were quickly isolated and placed into a structure called a *Pandas* dataframe. A *Pandas* dataframe can be thought of as an analogue to an Excel spreadsheet in Python.

4.8.3 Conversion from Pandas Dataframe to Numpy Array

Numpy is a very popular Python library that allows for the computation of arrays. It is entirely possible to code a neural network from scratch using *numpy*. *Numpy* tends to be an integral part of almost every machine-learning project in Python, and supports a host of mathematical operations that are commonly used in data science and machine learning. *Numpy* arrays are stored and operated upon on CPU’s however, so for any project that requires high performance computing, *numpy* might not be sufficient.

The Pandas Dataframe needed to be converted to a *numpy* array prior to its importation into PyTorch. PyTorch tensors are generally created from *numpy* arrays, so this intermediate step was required. Also, binary mask used for the

special connections between layers was also originally created as a *numpy* array.

4.8.4 Conversion from Numpy Array to PyTorch Tensor

PyTorch tensors are commonly created from *Numpy* arrays. Both the dataset and the binary mask matrix were quickly converted from 2-dimensional *numpy* arrays to PyTorch tensors. A PyTorch tensor is conceptually equivalent to a *numpy* array.

In order for the algorithm to work, the mask and dataset had been stored in the exact same type of PyTorch tensor data structure. These tensors were first stored on the machine's CPU.

In order for the algorithm to work, the mask and dataset had to be stored in the exact same type of PyTorch tensor data structure. Furthermore, this data type had to match the data type of the weights for the network. By default, the weights of the neural network take the form of a "Float Tensor". The mask and the dataset, when imported from numpy and pandas, were initially stored as "Long Tensors". These "Long Tensors" were converted to "Float Tensors" so that the weights, mask, and data all matched. These tensors were first stored on the machine's CPU.

4.8.5 Moving PyTorch Tensors to GPU

PyTorch allows a user to run algorithms on GPU rather than CPU for the sake of efficient computing. The next step in the pipeline was to convert the

PyTorch CPU tensors currently in use to PyTorch GPU tensors, which effectively moves the data storage from the CPU to GPU.

PyTorch take advantage of NVIDIA's CUDA library for computations run on NVIDIA GPU, so the PyTorch tensors were redefined as a specialized tensor optimized for CUDA computations. These steps were administered for both the mask and for the dataset.

4.8.6 Wrapping GPU Tensors in PyTorch Variables

PyTorch features the convention of wrapping tensor data structures into objects called Variables prior to using them in a model. These Variable objects have an attribute called ".data" that refers to the tensor held within the Variable.

Once the data and the mask were wrapped in Variables while stored on the GPU, the model was ready to be trained.

CHAPTER FIVE

MODEL TRAINING

5.1 Introduction

This section describes the training process for all autoencoders used in this thesis. This information includes the hyperparameters used, training procedures for avoiding overfitting, the logic behind these procedures, the duration of training, and the implementation of training within PyTorch.

Once the correct configuration of hyperparameters had been selected for the autoencoders, each module was ready to be trained. The GPU used for training these models was an Nvidia GTX 870, which allows for accelerated computations using NVIDIA's CUDA framework integrated into PyTorch. Neural networks train significantly faster on GPU, so training these models on GPU's allowed for a greater amount of experimentation.

5.2 Final Hyperparameters Used

With the different combination of the loss, activation and optimizer tested, L1 loss function, SELU activation function and Adam optimizer yielded the best results. This combination was used for all autoencoder architectures in all modules.

5.2.1 L1 Loss

L1 loss function as mentioned previously measures average absolute value between the networks predictions and the correct result then sum them up. In PyTorch, users have the option to then average this sum on a per-sample basis. In other words, this represents the average absolute value difference between the network's predictions and the correct results [29]. The formula for averaged L1 loss is as follows:

$$L1 = \frac{1}{n}\sum_{i=1}^{n}\left|y^{(i)} - \hat{y}^{(i)}\right|$$

In this case, L1's gradient took a shape that was easier for the Adam optimizer to descend upon.

5.2.2 SELU

SELU as mentioned previously the exponential linear units are scaled based on two constant parameters not subject to gradients, α & λ [33]. SELU do not transform positive neuron outputs, and simply bound negative neuron outputs at -1 with a biased exponential transformation. As will be discussed later, it is unlikely that scaled exponential units transformed weighted outputs to a significant degree. For whatever reason, scaled exponential linear units worked better than competing activation functions. The formula for scaled exponential linear units is as follows:

$$selu(x) = \lambda \begin{cases} x & if x > 0 \\ \alpha e^{x} - \alpha & if x \leq 0 \end{cases}$$

4.2.3 Adam

Adam as mentioned previously accumulates past squared gradients in an exponentially decaying average. In addition to this decaying average of squared gradients, Adam also accumulates an exponentially decaying average of previous gradients without squaring them. These two caches are running estimates of the mean and variance of past gradients, which are referred to as the first and second moments respectively. Adam then implements a bias correction to these estimates, and then uses this information to calculate new updates [17]. The Adam optimizer has the formulation:

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * J(\theta_t)$$

$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * \nabla J(\theta_t)^2$$

$$\theta_t = \theta_{t-1} - \frac{\alpha * m_t}{(\sqrt{v_t})}$$

It adapts updates on a per-parameter basis, and is able to incorporate a concept of momentum as well. These traits, in all likelihood, decreased the probability of the Adam optimizer getting stuck in a suboptimal local minimum.

4.2.4 Final Batch Size, Learning Rate and Scheduler

As mentioned earlier, different batch sizes and learning rate were tested. The best final set of autoencoders was batch of size 10 and a learning rate of size ($1.0*10^{-4}$) or .0001 across all architecture modules. Implementing a learning rate scheduler led to worse performance across the board in every architecture, so the scheduler was completely omitted from training.

5.3 Overfitting

Overfitting is when a model's target function maps inputs to outputs in a way that too closely matches the training data. When this happens, the model learns a pattern that is overly specific, and matches the training data, but would fail to regularize and apply to other similar data sets. Overfitting in neural networks is often mitigated by the implementation of a technique called dropout.

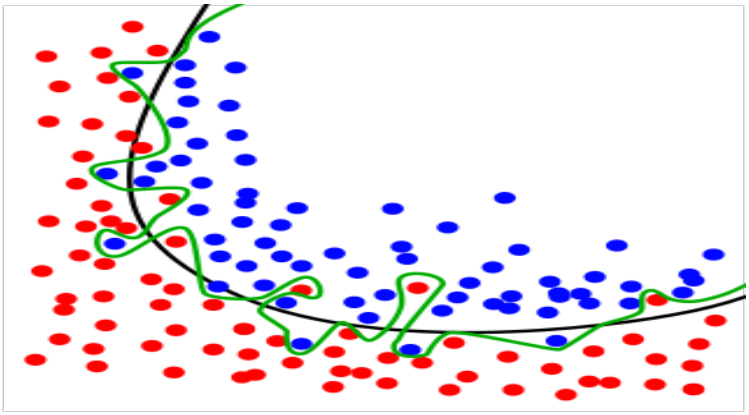


Figure 14. Overfitting [40]

4.3.1 Dropout

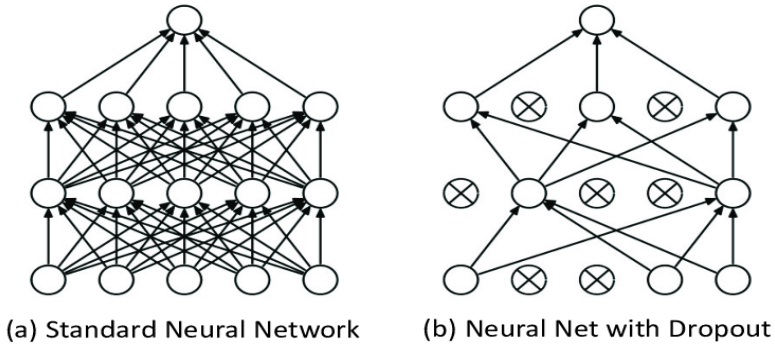


Figure 15. Dropout [41]

Researchers who train neural networks typically use dropout to avoid overfitting. Dropout is a procedure by which certain numbers of neurons in each layer are removed from any given forward pass [42]. The neurons removed will vary on each forward pass. The idea behind this is that the model is forced to learn a target function despite different neurons being deactivated at different times. Doing this should help prevent the neural network from developing a target function that is overly specific to the dataset. Dropout, however, was not used in this project for the reason mentioned below.

4.3.1.1 Reason not Using Dropout

Since the autoencoders employed in this project already had a large number of neurons removed from training due to the specialized connections between layers enforced by the binary mask matrix, dropout seemed like it would be less effective than normal. Simply dropout removes neurons, and the project's design also removes neurons, and there seemed to be a degree of redundancy there.

4.3.2 K-Fold Cross Validation

K-fold cross validation is a technique used to prevent overfitting. In k-fold cross validation the data set is randomly partitioned into a training set and a validation set [43]. The diagram below shows k-fold cross validation.

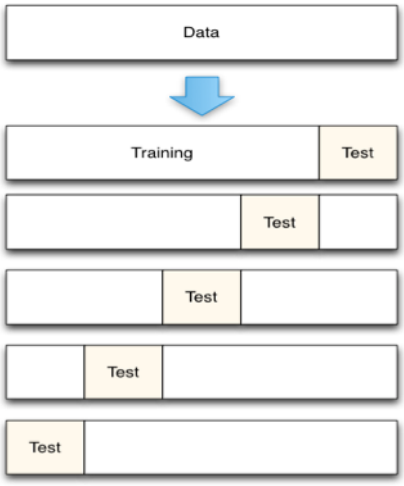


Figure 16. K-Fold Cross Validation [43]

K-fold cross validation in this experiment was implemented as follows. The data set was shuffled and randomly split into a training set and a test set. 90% of the data was used for training, and 10% for a test. The model was trained using the training data for 100,000 epochs, and then tested on the remaining test data. This represents a single fold in k-fold cross validation. Afterwards, entire data set was shuffled and randomly split again into a different training and test set of the same portions. Another fold of training and testing was carried out, followed by another random shuffle and split, until 10 folds of cross validation had been executed. This procedure theoretically prevents the model from approximating a target function that becomes overly reliant on a certain subsection of the dataset, thereby acting as a bulwark against overfitting.

5.3.3 Code for K-Fold Cross Validation and Model Training

Because PyTorch is very tightly integrated with the Python language, it was possible to implement k-fold cross validation by using traditional Python

control flow methods. The 10 splits of cross validation were written in the form of a Python *for* loop of 10 iterations. In each loop, the dataset was randomly split with a 9:1 training to validation ratio. Within each split, the model was trained for 100,00 epochs. The training was also represented as a *for* loop within the cross-validation *for* loop. The training loop consisted of 100,000 iterations.

5.4 Training

Each autoencoder was trained for 1,000,000 epochs consisting of 10 splits of 100,000 epochs for k-fold cross validation. Since the datasets were small and PyTorch is exceptionally efficient, the models were able to converge upon local minima very easily within this time.

For each autoencoder structure in each module, training was conducted 10 times. The results of each training session were saved, and the final weights for each structure represented an average of these 10 iterations.

5.4.1 Training Time

Each architecture took roughly 1 hour to train. This extrapolates to roughly 160 hours of total training time over the course of the experiment for all iterations of autoencoders on different modules.

5.4 Data Pipeline After Training

5.4.1 Pipeline for Reconstructed Data Set and Target

5.4.1.1 Moving Tensor to CPU

The first step in the pipeline, following training of the model, was to remove the data held in the Variable object representing the reconstructed ADOS dataset and the target. The next step was to convert the GPU tensor for the data into a CPU tensor, since the only purpose of using the GPU was training the model. These two steps were conducted in a single line of code for the reconstruction and target.

5.4.1.2 Convert PyTorch CPU Tensor to Numpy Array

Since the end goal of the pipeline was to convert the reconstructed data into a csv, similar to the original data, the Pytorch tensor needed to be converted to a Numpy array as an intermediate step between PyTorch and Pandas.

5.4.1.3 Convert Numpy Array to Pandas Data Frame

The next step in the pipeline was to convert the Numpy array into a Pandas data frame. The column names of the dataset were specified when creating the dataframe because this information was lost during training, as the model did not include column labels with the input.

5.4.1.4 Convert Pandas Dataframe to Comma Separated Value File

The final step in the reconstruction and target pipeline was to convert a pandas data frame into a comma separated value file so that it could be compared to the input.

5.4.2 Pipeline for Weights

The parameters of the model in PyTorch are stored in an iterable object referred to as “model.parameters”. The parameters object of the model has an attribute referred to as “param.data”.

To remove the weights from this object, a *for* loop was constructed to iterate through “model.parameters” and convert the data within into a PyTorch tensor.

From there, the weights followed the exact same pipeline as the reconstructed dataset did.

CHAPTER SIX

MODEL VALIDATION

6.1 Introduction

Validating a model involves insuring that the model can reliably and accurately achieve its purpose on relevant datasets. In this chapter, the accuracy of autoencoders at reconstructing their inputs will be evaluated. Since the purpose of an autoencoder is to accurately reconstruct its input as its target, validation for this project involved testing if the autoencoders used were actually able to accurately reconstruct their inputs. Accuracy was measured for each training iteration of autoencoders used in this project. Average accuracies are calculated for each combination of autoencoder architecture and module. A table of all autoencoder training iterations and their respective reconstruction accuracies can be found in Appendix D.

6.2 Validation Method

Validation was conducted by testing how accurately an autoencoder was able to reconstruct its input. Reconstructing an input from a smaller hidden layer effectively represents an autoencoders ability to decode its hidden layer and recover as much information about its input as possible. All analysis of variables that go into a reduced representation of the ADOS is only relevant if the

autoencoder can reliably reconstruct the original higher-dimensional sample from that reduced representation.

6.2.1.1 Compare Output with Target

As mentioned previously, in the model the data was split into 90 percent training and 10 percent validation. Following this division, the model trained normally for a certain number of epochs.

After training was completed for this partition, the dataset is then reshuffled and divided into a different random partition of 90/10. The model is then trained again, and then shuffled and randomly divided again, for an arbitrary number of splits.

The final outputs of the autoencoder represent reconstructed versions of the training data and the test data. Collectively, all samples from the original dataset are in the output, the only difference is that order is switched. For the sake of examining the autoencoder output, the training set reconstructions were compared to the training set splits, since these represent the overwhelming majority of the input data. The data for every individual patient remains completely unchanged, so comparing the reconstruction to the shuffled data set is a completely valid method for evaluating the reconstruction performance of the autoencoder. However, since the order of the patients has been changed, the reconstruction compared to the original data set as supplied by the AGRE in its original patient order.

K-fold cross validation was conducted to attempt to limit the potential for overfitting by this model.

6.2.1.2 Calculating the Accuracy

Since the output was treated as a regression problem, all values in the output had to be rounded to the nearest integer prior to comparison with the input. After rounding, overlapping comma separated value files were compared, and all discrepancies highlighted. The number of discrepancies represents the number of observations that the autoencoder failed to accurately reconstruct. By dividing the number of discrepancies by the total number of observations, the percentage of wrong observations was calculated. Subtracting this percentage from 1 gave an accuracy percentage for the autoencoder.

6.2.1.3 Average the Accuracy

Accuracy numbers were calculated for all 10 autoencoders for a single architecture and module. To determine the reconstruction accuracy of a given autoencoder, the number of correct values divided by total values was calculated for each training session. After accuracy percentages were calculated for all 10 training versions of a single architecture and module combination, those accuracies were averaged to estimate the aggregate average accuracy of a given architecture in reconstructing a module.

6.3 Autoencoder Reconstruction Accuracy

6.3.1 Module 1 Autoencoder Reconstruction Accuracy

The diagram below shows the average accuracy across Module 1 architectures.

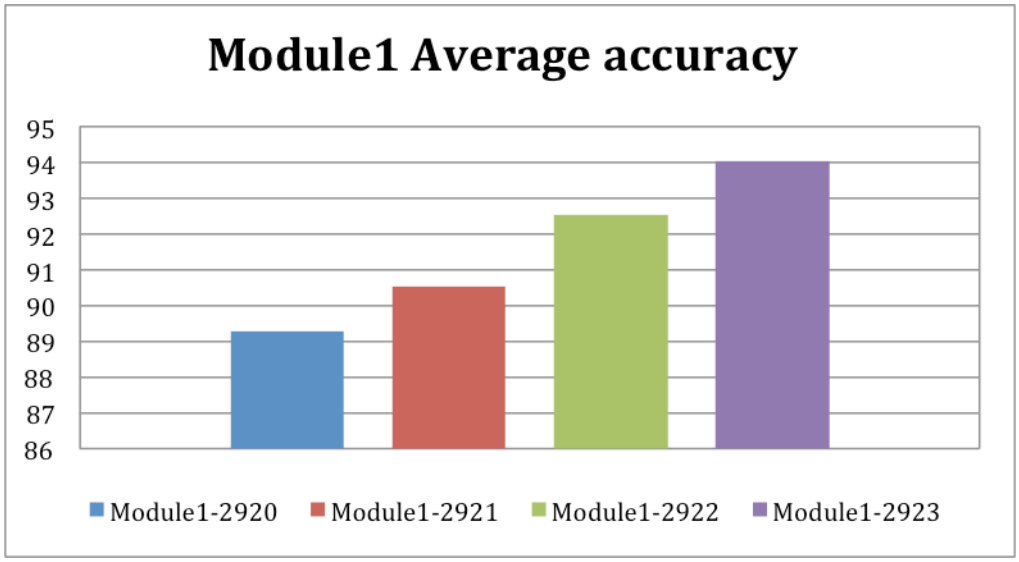


Figure 17. Average Accuracy Across Module 1

All autoencoders tested ended up with accurate reconstructions. Within Module 1, the autoencoder that reduced the 29-variable diagnostic to 20 variables ended up reconstructing its inputs with 89.3% accuracy. This number, and each of the following average accuracy measures, describes the average accuracy of 10 identically designed and trained autoencoders. The least accurate of the 10 autoencoders reconstructed its input with 87.56% accuracy, and the most accurate reconstructed its input with 90% accuracy.

The autoencoder architecture that reduced the 29-variable Module 1 to 21 variables had an average accuracy of 90.5%. The least accurate iteration scored at 89.35% accuracy, and most accurate iteration had accuracy of 91.74%.

The autoencoder architecture that reduced the 29-variable Module 1 to 22 variables had an average accuracy of 92.5%. The least accurate iteration scored at 91.54% accuracy, and most accurate iteration had accuracy of 93.36%.

The autoencoder architecture that reduced the 29-variable Module 1 to 23 variables had an average accuracy of 94%. The least accurate iteration scored at 93.35% accuracy, and most accurate iteration had accuracy of 94.53%.

As one might expect, Module 1 autoencoders with larger hidden layers were more accurate at reconstructing than autoencoder architectures with smaller hidden layers.

6.3.2 Module 2 Autoencoder Reconstruction Accuracy

The diagram below shows the average accuracy across Module 2 architecture**.**

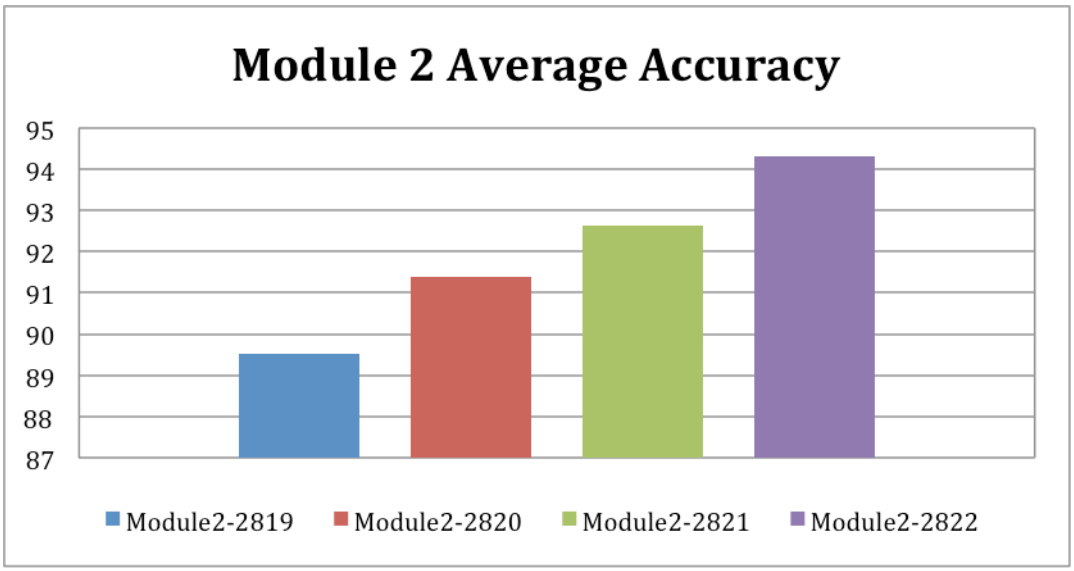


Figure 18. Average Accuracy Across Module 2

On Module 1, the autoencoder that reduced the 28-variable diagnostic to 19 variables ended up reconstructing its inputs with 90.5% accuracy. The least accurate of the 10 autoencoders reconstructed its input with 88.15% accuracy, and the most accurate reconstructed its input with 91.12% accuracy.

The autoencoder architecture that reduced the 28-variable Module 2 to 21 variables had an average accuracy of 91.4%. The least accurate iteration scored at 90.97% accuracy, and most accurate iteration had accuracy of 91.86%.

The autoencoder architecture that reduced the 28-variable Module 2 to 22 variables had an average accuracy of 92.6%. The least accurate iteration scored at 92.03% accuracy, and most accurate iteration had accuracy of 93.59%.

The autoencoder architecture that reduced the 29-variable Module 2 to 23 variables had an average accuracy of 94.3%. The least accurate iteration scored at 92.22% accuracy, and most accurate iteration had accuracy of 95.08%.

Module 2 autoencoders showed decreased accuracy with smaller hidden layers, as is expected.

6.3.3 Module 3 Autoencoder Reconstruction Accuracy

The diagram below shows the average accuracy across Module 3 architecture.

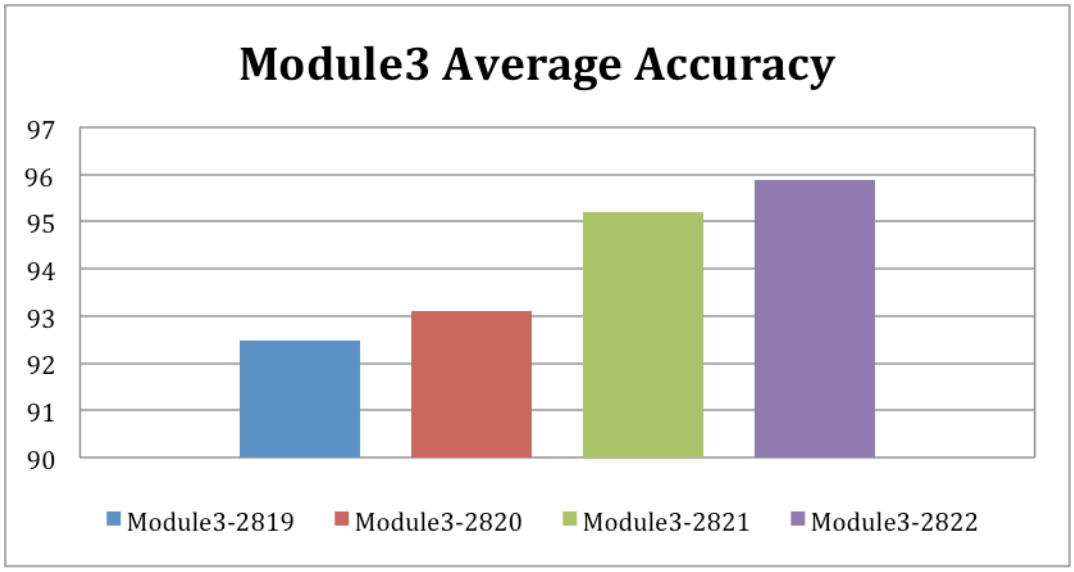


Figure 19. Average Accuracy Across Module 3

On Module 3, the autoencoder that reduced the 28-variable diagnostic to 19 variables ended up reconstructing its inputs with 92.5% accuracy. The least accurate of the 10 autoencoders reconstructed its input with 91.60% accuracy, and the most accurate reconstructed its input with 93.37% accuracy.

The autoencoder architecture that reduced the 28-variable Module 3 to 20 variables had an average accuracy of 93.1%. The least accurate iteration scored at 92.48% accuracy, and most accurate iteration had accuracy of 93.67%.

The autoencoder architecture that reduced the 28-variable Module 3 to 21 variables had an average accuracy of 95.2%. The least accurate iteration scored at 94.23% accuracy, and most accurate iteration had accuracy of 96.09%.

The autoencoder architecture that reduced the 29-variable Module 3 to 22 variables had an average accuracy of 95.9%. The least accurate iteration scored at 95.07% accuracy, and most accurate iteration had accuracy of 96.82%.

Module 3 autoencoders showed decreased accuracy with smaller hidden layers, as is expected.

6.3.4 Module 4 Autoencoder Reconstruction Accuracy

The diagram below shows the average accuracy across Module 3 architecture.

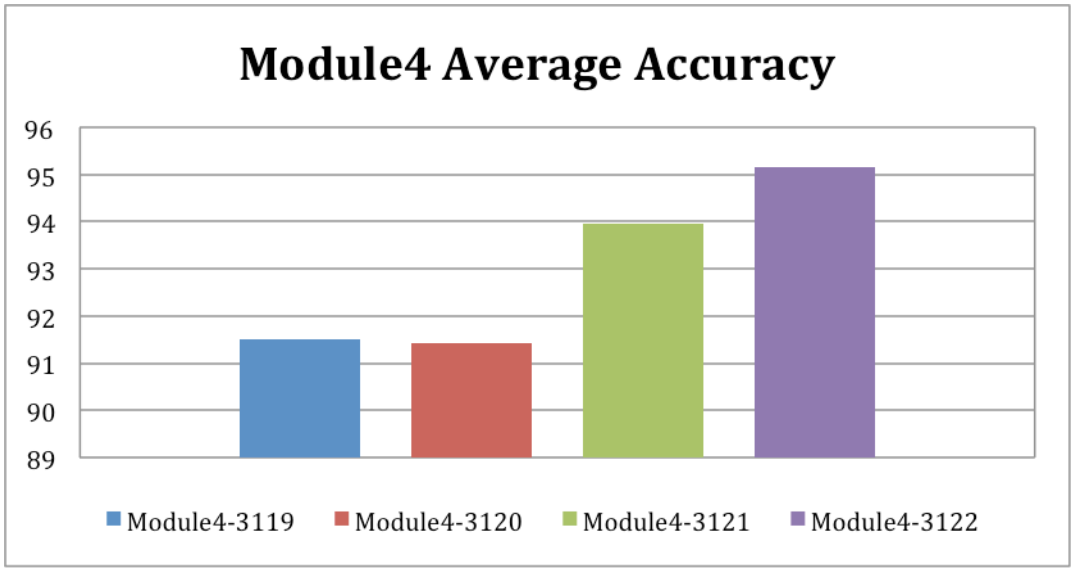


Figure 20. Average Accuracy Across Module 4

On Module 4, the autoencoder that reduced the 31-variable diagnostic to 19 variables ended up reconstructing its inputs with 91.5% accuracy. The least

accurate of the 10 autoencoders reconstructed its input with 90.93% accuracy, and the most accurate reconstructed its input with 91.15% accuracy.

The autoencoder architecture that reduced the 31-variable Module 4 to 20 variables had an average accuracy of 91.4%. The least accurate iteration scored at 91.00% accuracy, and most accurate iteration had accuracy of 92.21%.

The autoencoder architecture that reduced the 31-variable Module 4 to 21 variables had an average accuracy of 94%. The least accurate iteration scored at 93.52% accuracy, and most accurate iteration had accuracy of 94.31%.

The autoencoder architecture that reduced the 31-variable Module 4 to 22 variables had an average accuracy of 95.1%. The least accurate iteration scored at 94.48% accuracy, and most accurate iteration had accuracy of 95.82%.

Module 4 autoencoders mostly showed decreased accuracy with smaller hidden layers, with one exception. The 19-variable hidden layer actually had an accuracy that was on average 0.1% higher than the 20-variable hidden layer. With the exception of this discrepancy, the trend held true throughout all autoencoders.

CHAPTER SEVEN

MODEL ANALYSIS

7.1 Introduction

This section explores the weights of autoencoders to extract all possible insights regarding the focal points of a reduced diagnostic. The lowest dimensional reconstruction is evaluated for each module. Additionally, weights are examined across all architectures within a module, and across all modules to see if there are any consistent overarching trends.

7.2 Analysis Method

7.2.1 Comma Separated Value File

After Convert Pandas data frame to csv file, weights can be accessed in Excel. The encoder weights pulled from each autoencoder represent the weights used in the final epoch of training, prior to being element-wise multiplied with the binary mask matrix. Therefore, a large number of the weights in the comma separated value file were not used. It is simple to identify these weights based on their rows and columns, and delete them. Weights that were removed by the mask on the final forward pass were deleted prior to weight analysis for the purpose of clarity.

7.2.2 Averaging Weights

After 10 autoencoders for each architecture and module were trained, the weights from each autoencoder were converted to absolute values. The absolute value of a variable's weights determines its impact in the reduced diagnostic, so whether a weight was positive or negative was irrelevant in terms of its impact. After conversion to absolute values, the weights for all 10 identical autoencoders were averaged before analysis.

7.2.3 Reduced Diagnostic Analysis

Since all autoencoders tested demonstrated a satisfactory level of accuracy, the following analysis focuses on the autoencoder with the lowest dimensional hidden layer for each module. Therefor, a reduced diagnostic size of 20 variables for Module 1, and 19 variables for Module 2, Module 3, and Module 4. The weight data for the other architectures and modules can be found in Appendix E.

7.2.4 Explanation of Graphs for Nodes

The following pie charts show the relative strength of connections between nodes in the input layer and nodes in the hidden layer. With the type of autoencoders employed in this project, nodes corresponding to original diagnostic variables influence, through weights, nodes corresponding to condensed variables in the reduced diagnostic. Each of the following pie charts represents a single node in the reduced section and shows the relative extent to

which different variables in the original diagnostic influenced the node through their respective weights.

7.2.5 Explanation of Graphs for Sections

The following bar chart compares the impact of the weights of each variable in terms of impact across an entire reduced diagnostic section. These numbers are simply calculated by summing the weights of a single variable in every question in its relevant section. These bar graphs display which variables had the greatest impact on the reduced diagnostic shown here, and allows for their comparison.

7.2.6 Explanation for Correlations and Correlational Graphs

The following graphs show correlational relationships between the weights of variables in each section of the autoencoder. Values approaching 1 show a near perfect positive correlation, values approaching -1 show a near perfect inverse correlation. Variable pairs with values of 0 demonstrate no discernable correlational relationship. When two variables correlate in the positive direction, it means that increased presence of one variable generally accompanies increased presence of another. For this experiment, correlations with strength of 0.7 or greater were treated as strong positive relationships, and correlations greater than 0.9 were treated as almost perfectly correlated.

7.2.7 Meaning of Weights

The weight composition of a node in the hidden layer shows the extent to which variables in the input layer affected that node. Since weights represent coefficients for different inputs, weights with the highest absolute values represent a greater strength of connection between input nodes and hidden layer nodes. Absolute values close to zero represent minimal influence between nodes. Therefore, by examining the weight composition of each hidden layer node, in terms of the relative weight strength of input variables, it is possible to examine the conceptual makeup of each node in the hidden layer. On a section-by-section basis, seeing which input variables were the most represented and least represented in the reduced diagnostic can give insights as to how ADOS might be reduced into a shorter, viable diagnostic.

7.3 Module 1 Analysis

The diagram below shows the Module 1 Autoencoder with a hidden layer of size 20.

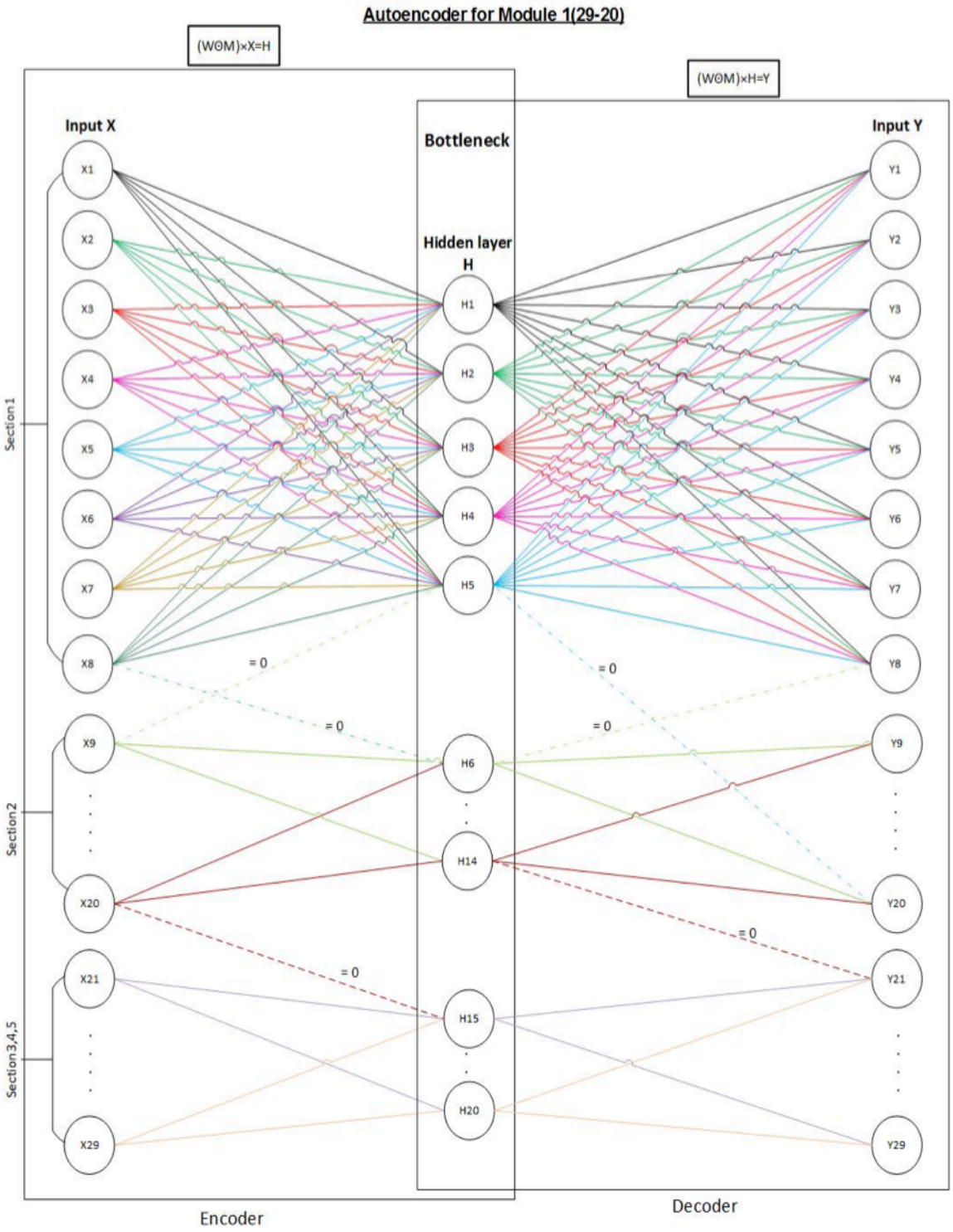


Figure 21. Module 1 Autoencoder with a Hidden Layer of Size 20

7.3.1 Module 1 Section 1 Analysis

The autoencoder reduced the first section of Module 1 from 8 variables to 5 variable sections. Section 1 subject is Language and Communication. The 8 variables in this section are Overall Level of Non-Echoed Spoken Language (OLANG), Frequency of Spontaneous Vocalization Directed to Others (FVOC), Intonation of Vocalizations or Verbalizations (INTON), Immediate Echolalia (IECHO), Stereotyped/Idiosyncratic Use of Words or Phrases (STEREO), Use of Another's Body (UOTHER), Pointing (POINT), and Gestures (GEST). The diagrams below show the weights of the reduced diagnostic of this section.

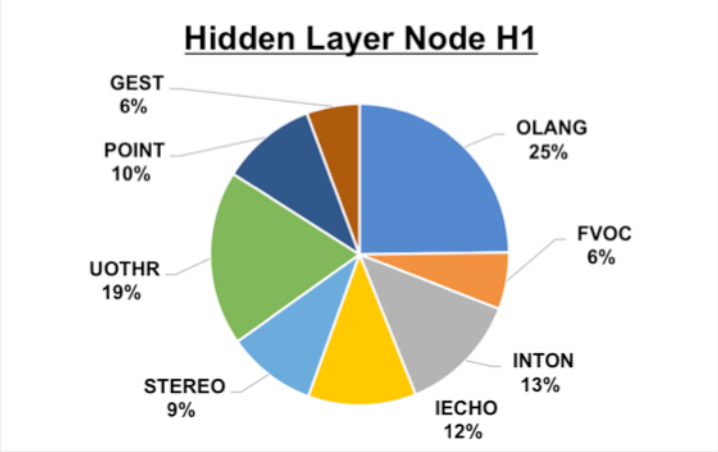


Figure 22. Relative Weights Impacts Node H1 – Module 1

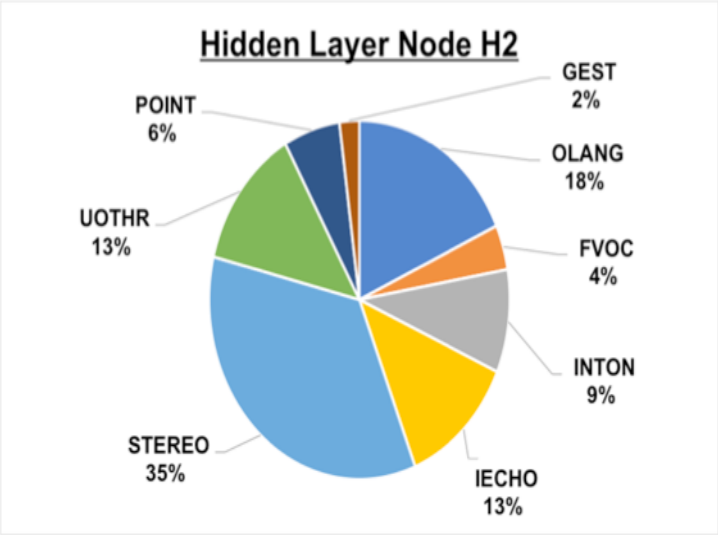


Figure 23. Relative Weights Impacts Node H2 – Module 1

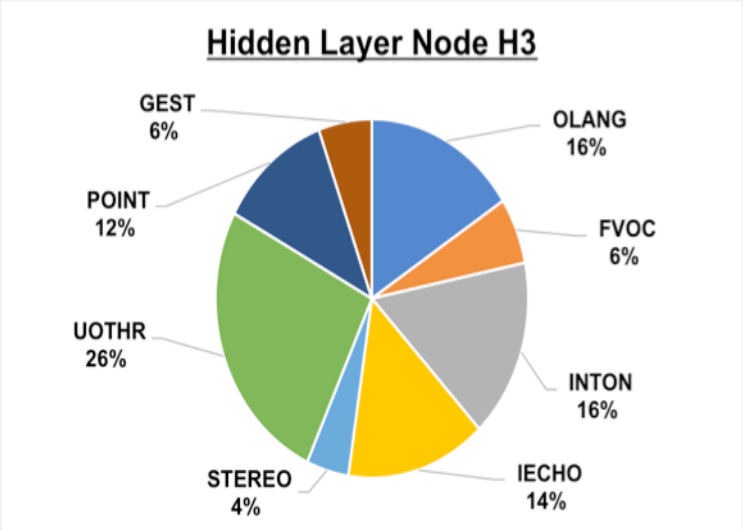


Figure 24. Relative Weights Impacts Node H3 – Module 1

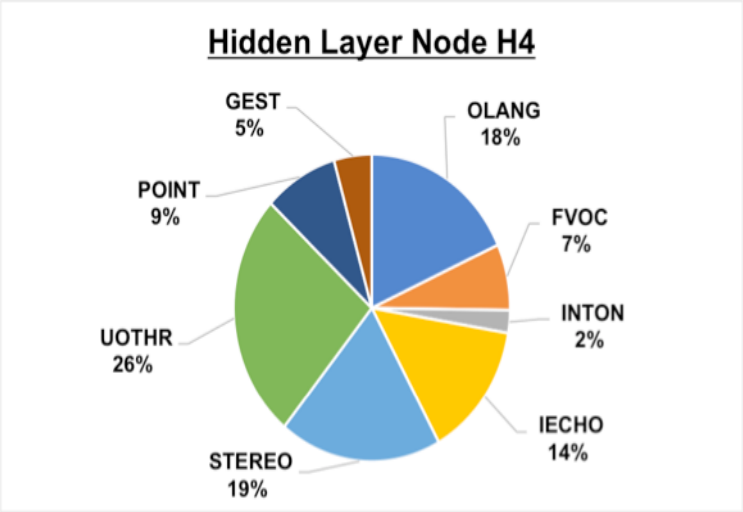


Figure 25. Relative Weights Impacts Node H4 – Module 1

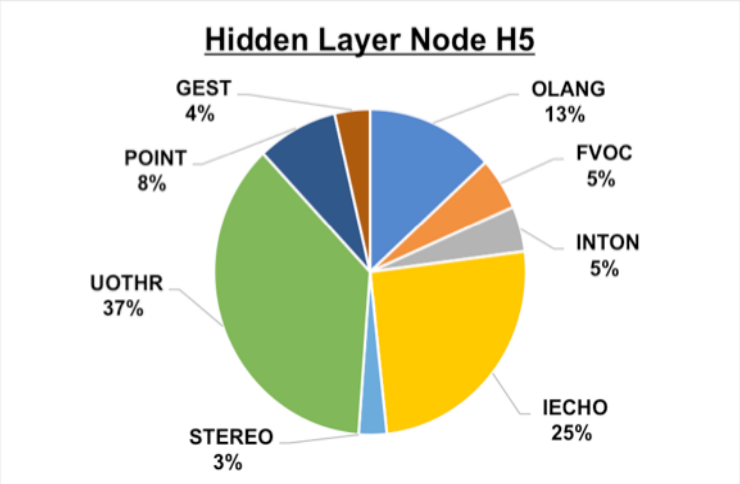


Figure 26. Relative Weights Impacts Node H5 – Module 1

The variables with the least representation in the reduced diagnostic are POINT, FVOC, and GEST. UOTHER" has the greatest presence by a significant margin. OLANG, IECHO and STEREO also had strong presences in the reduced diagnostic.

Variables with strong weights can potentially serve as focal points for a reduced ADOS with a 20-variable Module 1. POINT and GEST might be redundant variables, given there near perfect correlations. It should be noted that the variables with the smallest weight presences in this section still had significant absolute values, especially in comparison to weights analyzed in other sections and other modules. It is difficult to advocate completely disregarding any variables in the first section, but there might exist possibilities for more concise observation variable lists.

7.3.1.1 Correlation Between Variables

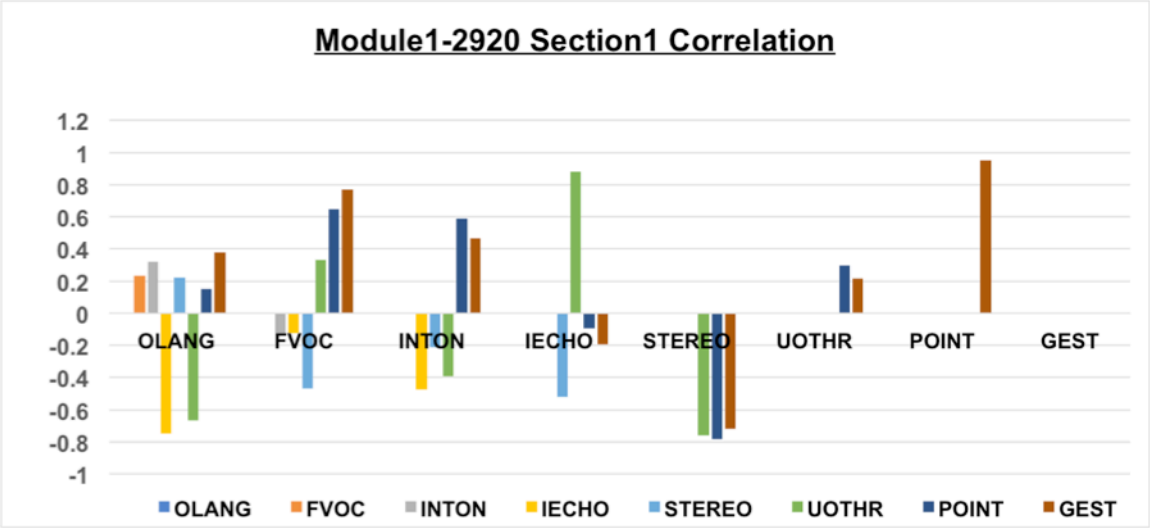


Figure 27. Weights Correlations – Module 1 Section 1

Within the reduced diagnostic nodes, the weights of IECHO and UOTHER almost perfectly correlate in their presence. Likewise, the weights of POINT and GEST have a correlation of almost 1.

7.3.2 Module 1 Section 2 Analysis

Section 2 of Module 1 of the ADOS involves the subject of Reciprocal Social Interaction. The autoencoder reduced this section from 12 variables to 9 variables. The original 12 variables were Unusual Eye Contact (UEYE), Responsive Social Smile (SSMILE), Facial Expressions Directed to Others (FACEO), Integration of Gaze (GZSOV) and Other Behaviors During Social Overtures, Shared Enjoyment in Interaction (SHRNJ), Response to Name (RNAME), Requesting (REQ), Giving (GIVE), Showing (SHOW), Spontaneous Initiation of Joint Attention (SIJNT), Response to Joint Attention (RJNT), and Quality of Social Overtures (QSOV). The diagrams below shows the weights of the reduced diagnostic of this section.

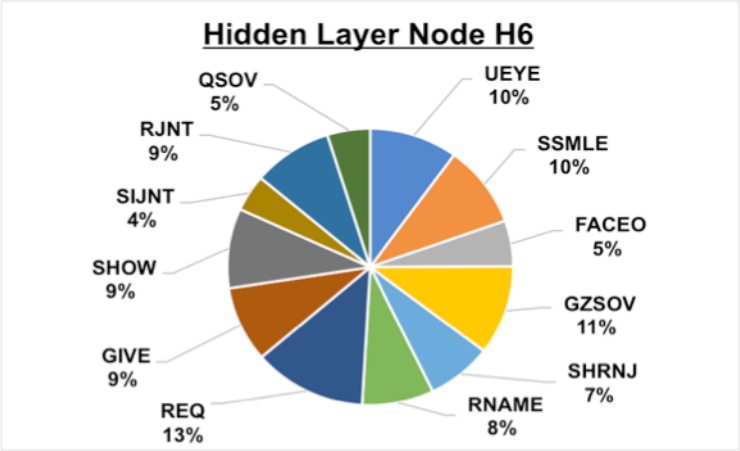


Figure 28. Relative Weights Impacts Node H6 – Module 1

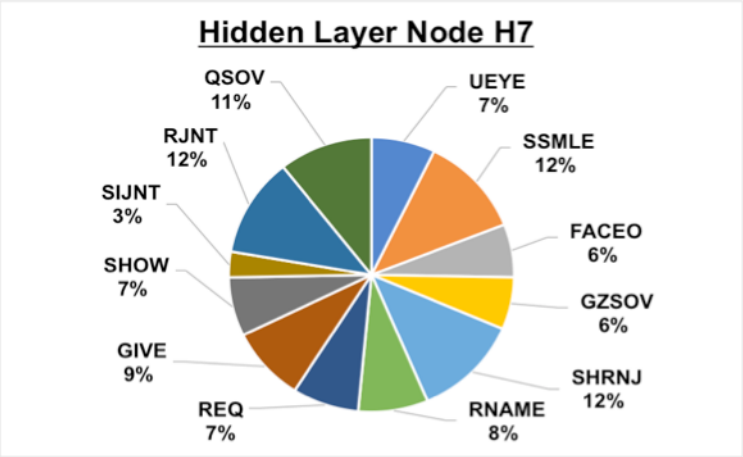


Figure 29. Relative Weights Impacts Node H7 – Module 1

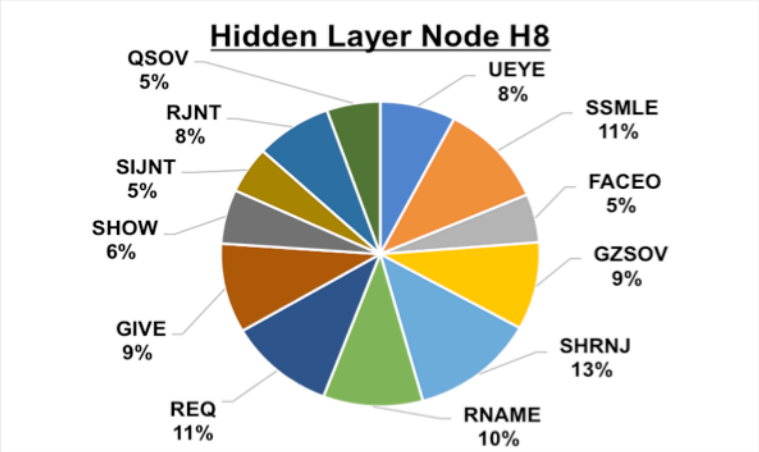


Figure 30. Relative Weights Impacts Node H8 – Module 1

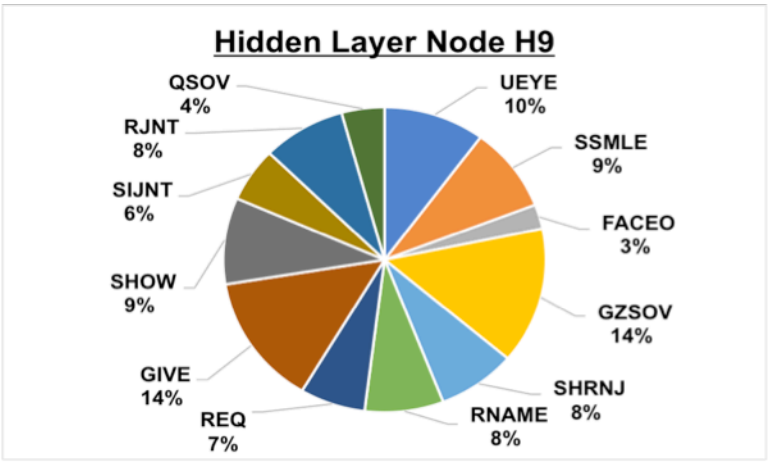


Figure 31. Relative Weights Impacts Node H9 – Module 1

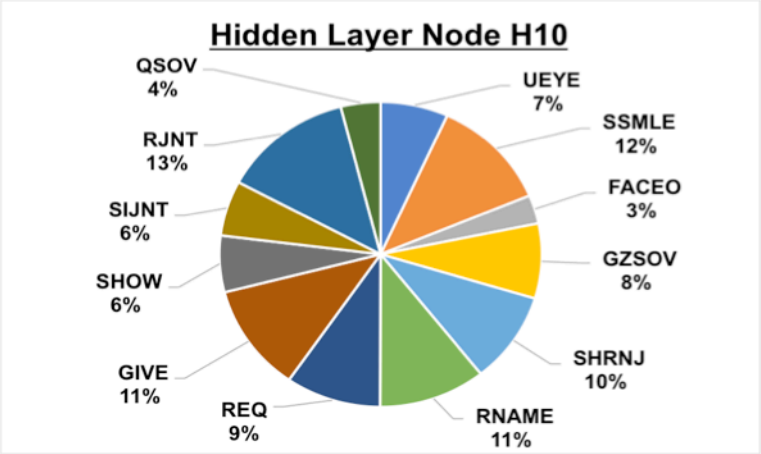


Figure 32. Relative Weights Impacts Node H10 – Module 1

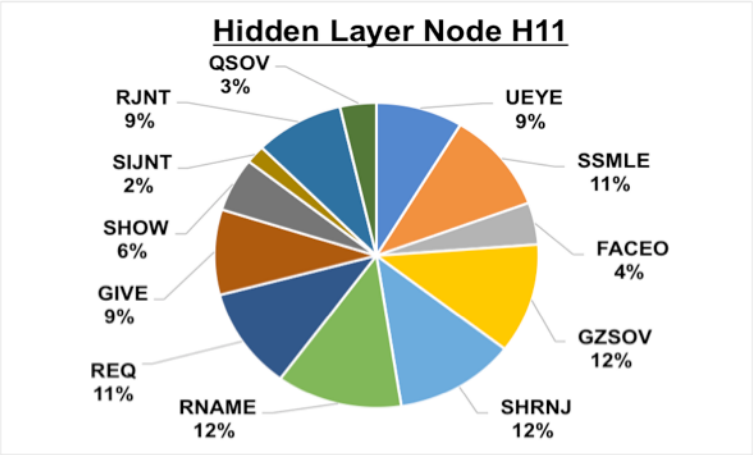


Figure 33. Relative Weights Impacts Node H11 – Module 1

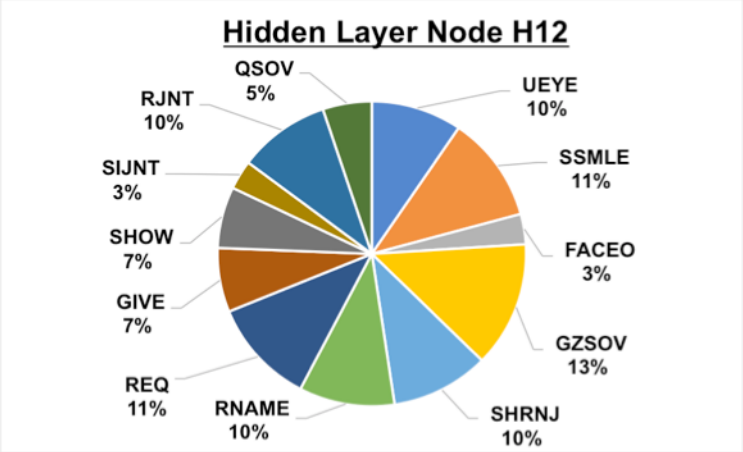


Figure 34. Relative Weights Impacts Node H12 – Module 1

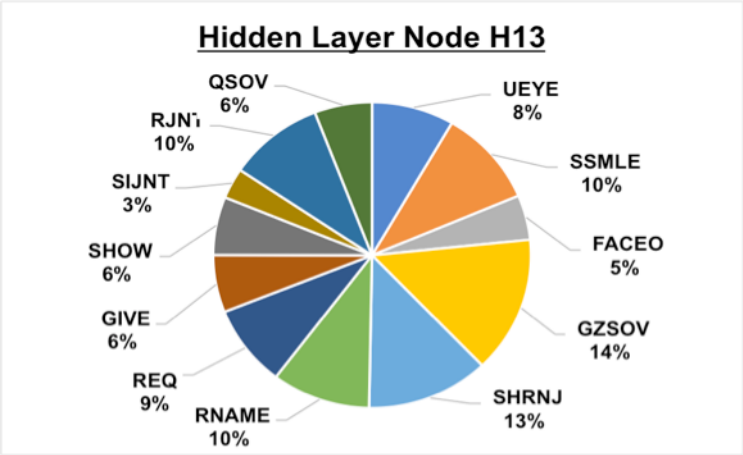


Figure 35. Relative Weights Impacts Node H13 – Module 1

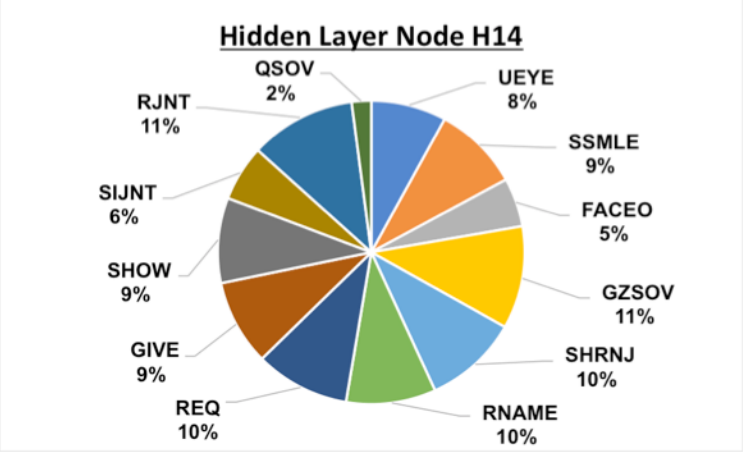


Figure 36. Relative Weights Impacts Node H14 – Module 1

In this section, none of the weights were negligible, but there did exist a clear hierarchy in variable importance. GZSOV, SSMILE and SHRNJ were the three most important variables by a significant margin. RJNT, REQ, RNAME, and GIVE formed the next tier of importance. UEYE and SHOW were of middling importance. SIJNT, FACEO, and QSOV were the three least represented

variables by a significant margin. Again, this information might have utility in identifying focal points for condensed observations.

7.3.2.1 Correlation Between Variables

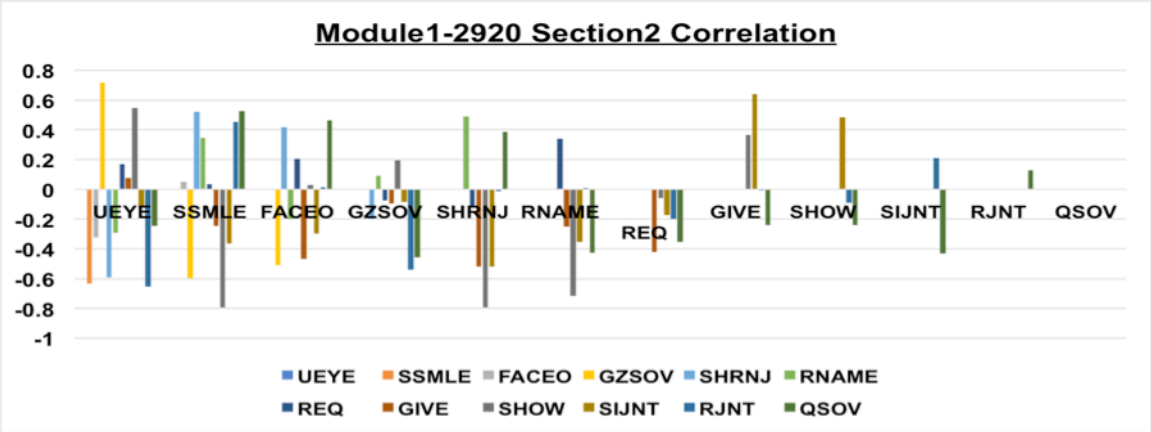


Figure 37. Weights Correlations – Module 1 Section 2

Correlations of 0.5 are considered weak according to conventional statistics.

7.3.3 Module 1 Section 3 Analysis

The third section of the reduced Module 1 autoencoder involved reducing 9 original ADOS variables into 6 new ones. The 9 original variables spanned the subjects of Play, Stereotyped Behaviors/Restricted Interests, and Other Abnormal Behaviors. The original variables were Functional Play with Objects (FPLAY), Imagination/Creativity (IMGCR), Unusual Sensory Interest in Play Material/Person (USENS), Hand and Finger and Other Complex Mannerisms (OMAN), Self-Injurious Behavior (SELFINJ), Unusually Repetitive Interests or Stereotyped Behaviors (URBEH), Overactivity (ACTIVE), Tantrums, Aggression,

Negative or Disruptive Behavior (AGG), and Anxiety (ANXTY). The diagrams below shows the weights of the reduced diagnostic of this section.

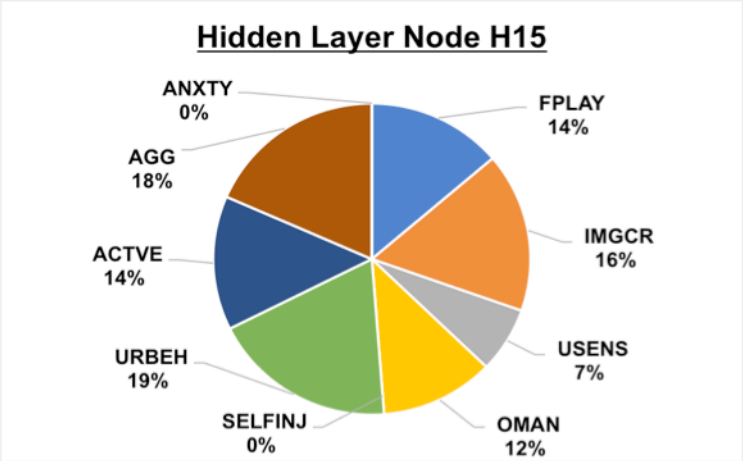


Figure 38. Relative Weights Impacts Node H15 – Module 1

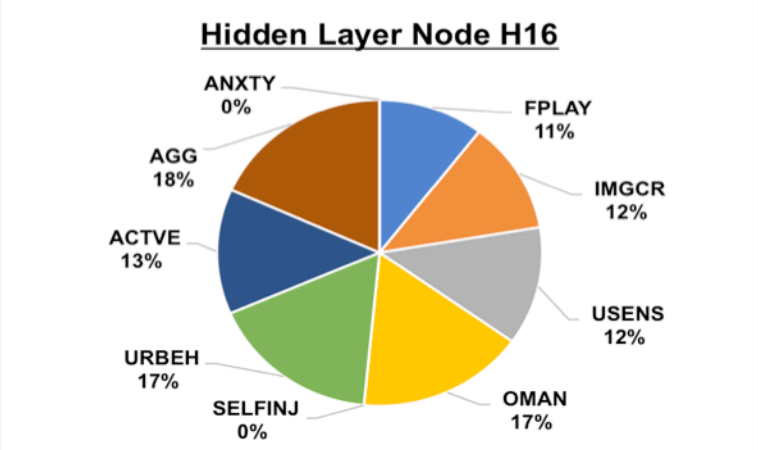


Figure 39. Relative Weights Impacts Node H16 – Module 1

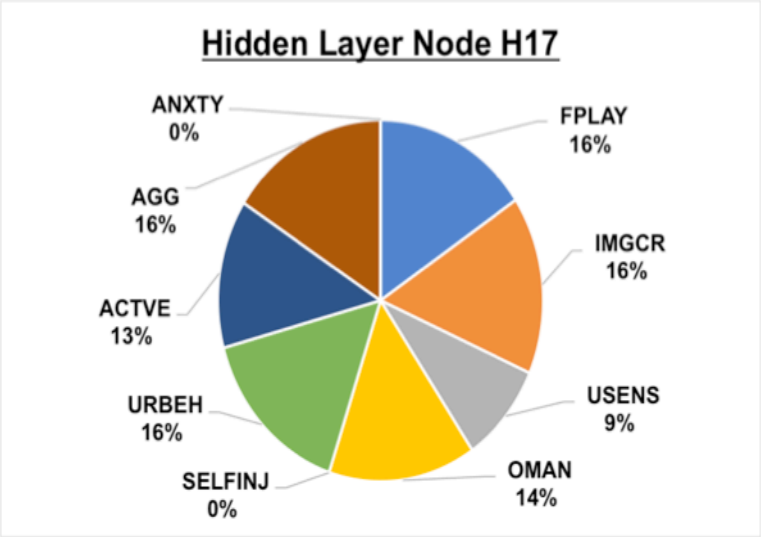


Figure 40. Relative Weights Impacts Node H17 – Module 1

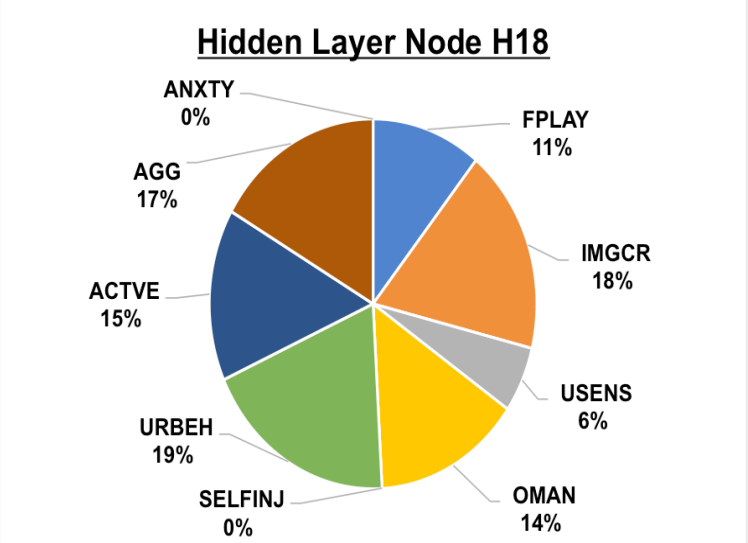


Figure 41. Relative Weights Impacts Node H18 – Module 1

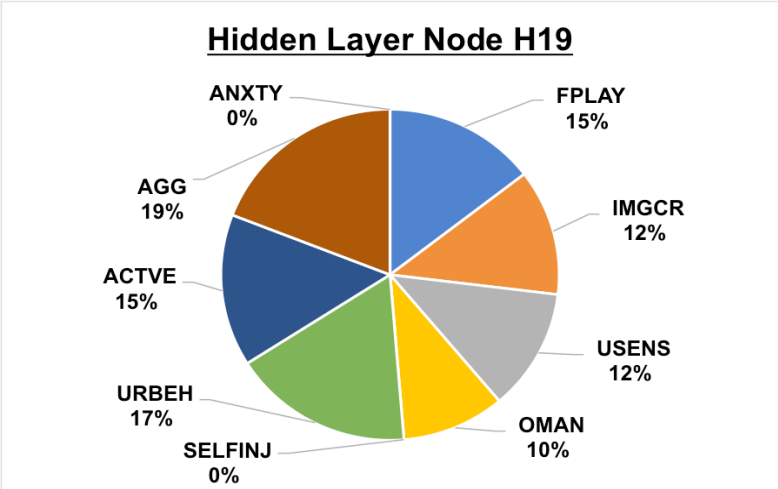


Figure 42. Relative Weights Impacts Node H19 – Module 1

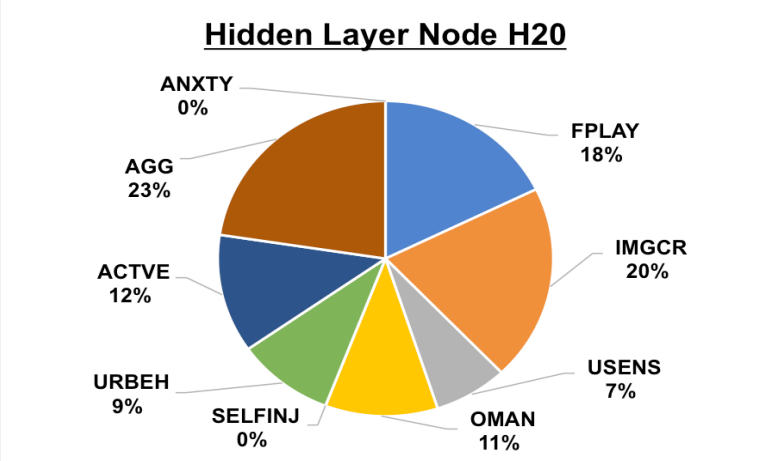


Figure 43. Relative Weights Impacts Node H20 – Module 1

In this section SELFINJ and ANXTY were by far the two variables with the smallest strength weights. The weights for these variables were orders of magnitudes smaller than anything else in the diagnostic and were effectively eliminated from the reduced diagnostic. For reference, the weights of these two variables were over 4,000 times smaller than the third-least represented variable.

This suggests it might be possible to eliminate or disregard these variables in a reduced diagnostic.

Every other variable in the original section had a significant weight presence in the reduced diagnostic. The most significant variables were AGG, URBEH, and IMGCR.

7.3.3.1 Correlation Between Variables

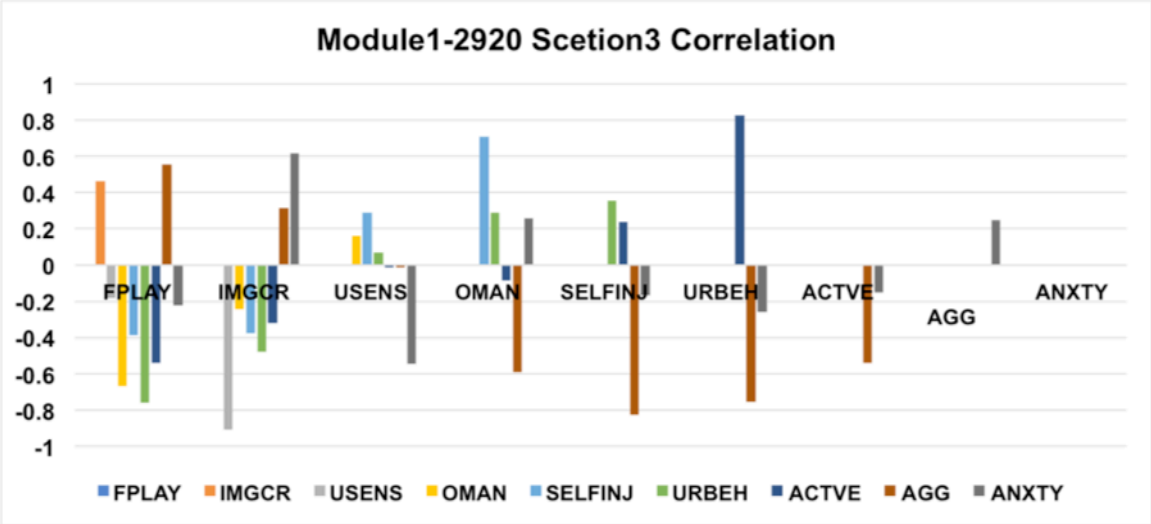


Figure 44. Weights Correlations – Module 1 Section 3

ACTIVE strongly correlates with URBEH, which suggests that increased presence of one variable in a node generally accompanies increased presence of another.

7.4 Module 2 Analysis

The diagram below shows Module 2 Autoencoder with Hidden layer of size 19.

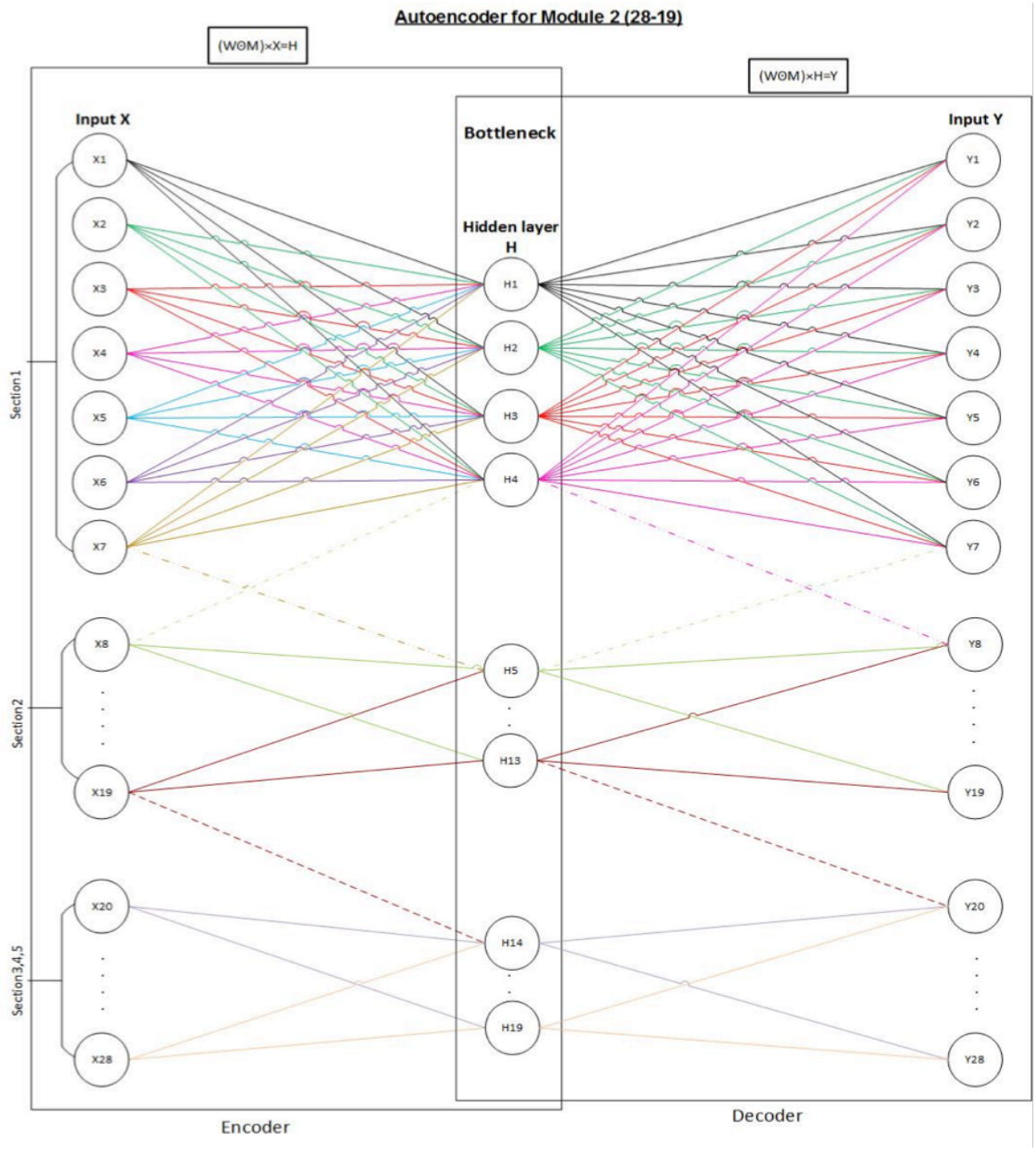


Figure 45. Module 2 Autoencoder with Hidden Layer of Size 19

7.4.1 Module 2 Section 1 Analysis

The first section of Module 2 corresponds to Language and Communication. The 7-variable original section was reduced to a 4-variable section by the autoencoder. The 7 variables in this section are Overall Level of Non-Echoed Spoken Language (OLANG), Speech Abnormalities (SPABN), Immediate Echolalia (IECHO), Stereotyped/Idiosyncratic Use of Words or Phrases (STEREO), Conversation (CONVS), Pointing (POINT), and Descriptive Gestures (DGEST). The diagrams below shows the weights of the reduced diagnostic of this section.

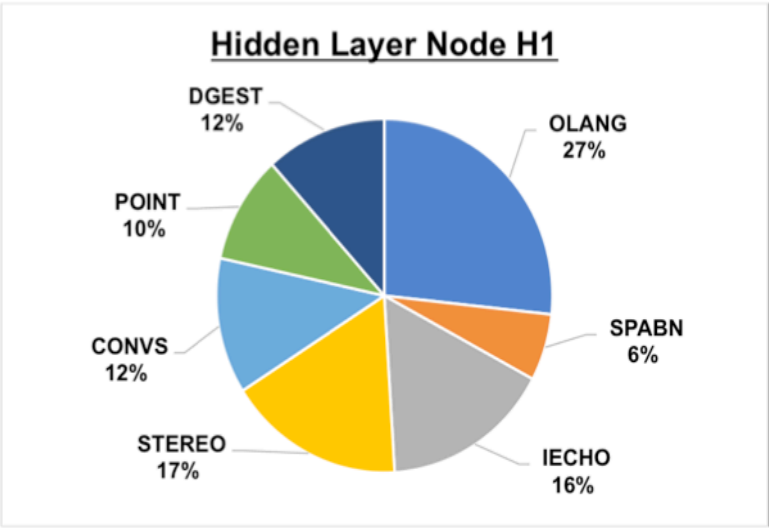


Figure 46. Relative Weights Impacts Node H1 – Module 2

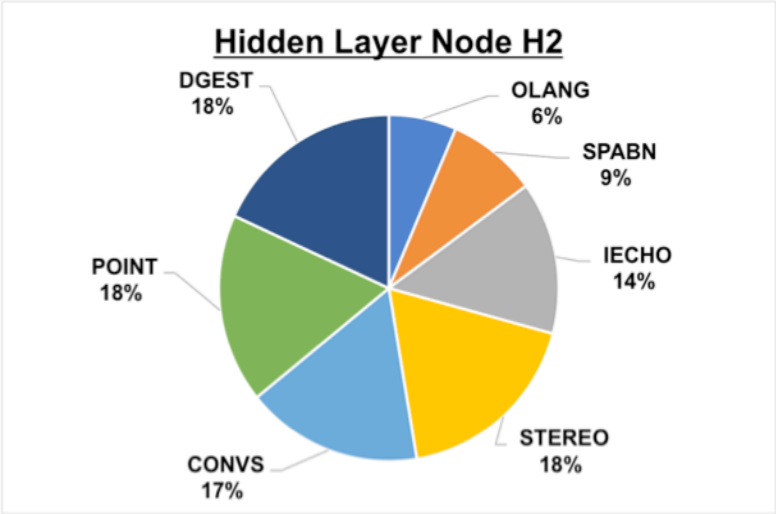


Figure 47. Relative Weights Impacts Node H2 – Module 2

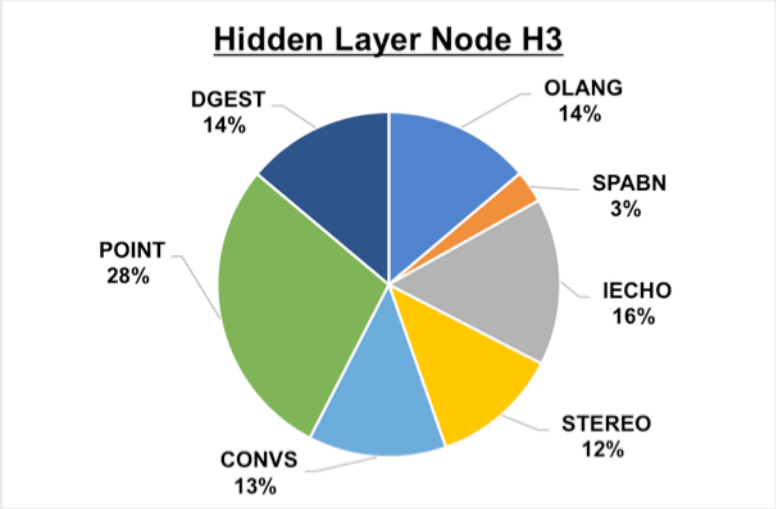


Figure 48. Relative Weights Impacts Node H3 – Module 2

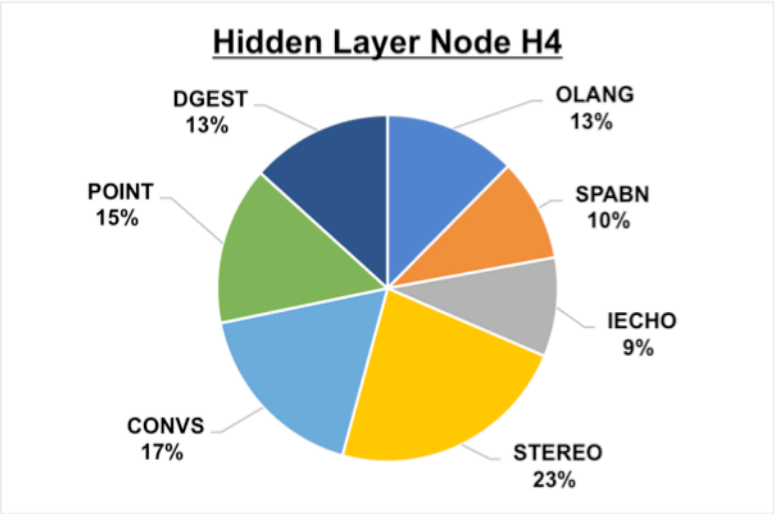


Figure 49. Relative Weights Impacts Node H4 – Module 2

SPABN is the least represented variable by a significant margin. This data suggests that this variable could likely be deprioritized in a reduced diagnostic. None of the variables in this section have negligible weights however. The most represented variables are POINT and STEREO by a significant margin. The other variables, with the exception of SPABN are all closely clustered in the strength of their weights.

7.4.1.1 Correlation Between Variables

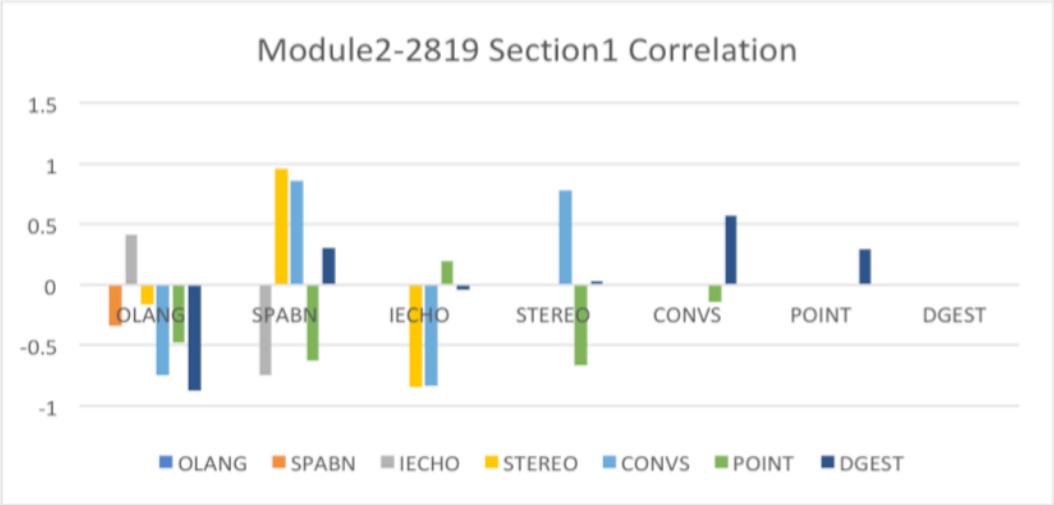


Figure 50. Weights Correlations – Module 2 Section 1

SPABN and STEREO correlate almost perfectly in their weights. SPABN also correlates strongly with CONVS.

7.4.2 Module 2 Section 2 Analysis

Section 2 of Module 2 of the ADOS involves the Subject of Reciprocal Social Interaction. The autoencoder reduced this section from 12 variables to 9 variables. The original 12 variables were Unusual Eye Contact (UEYE), Facial Expressions Directed to Others (FACEO), Shared Enjoyment in Interaction (SHRNJ), Response to Name (RNAME), Showing (SHOW), Spontaneous Initiation of Joint Attention (SIJNT), Response to Joint Attention (RJNT), Amount of Social Overtures (ASOV), Quality of Social Response (QSOV), Amount of Reciprocal Social Communication (ARSOC) and Overall Quality of Rapport

(OQRAP). The diagrams below shows the weights of the reduced diagnostic of this section.

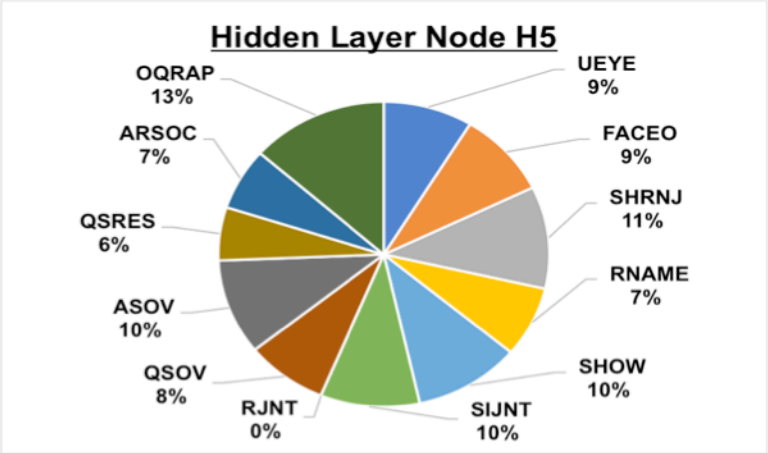


Figure 51. Relative Weights Impacts Node H5 – Module 2

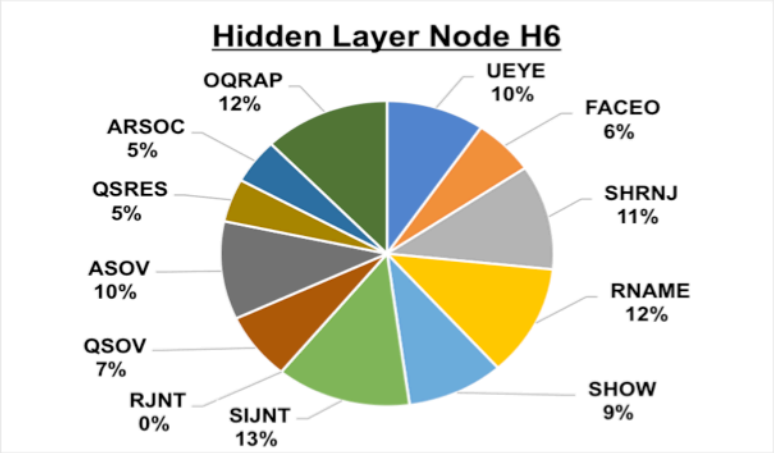


Figure 52. Relative Weights Impacts Node H6 – Module 2

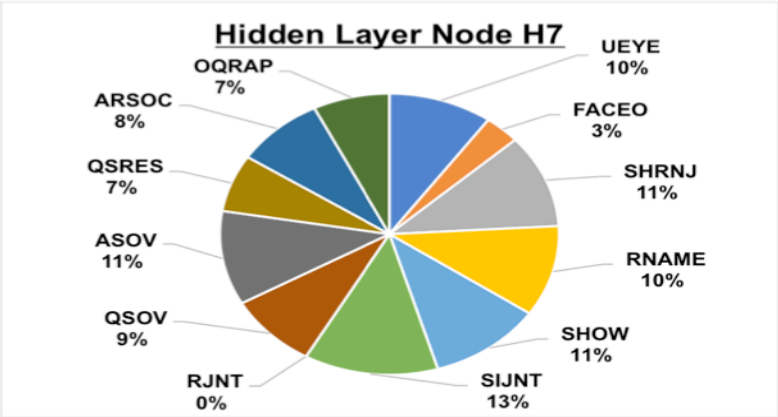


Figure 53. Relative Weights Impacts Node H7 – Module 2

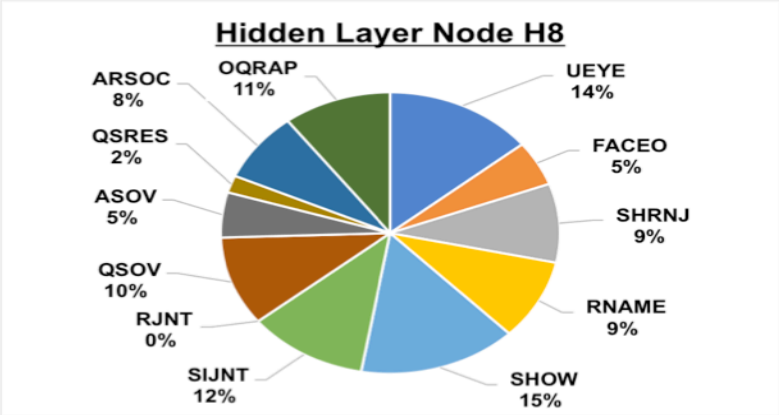


Figure 54. Relative Weights Impacts Node H8 – Module 2

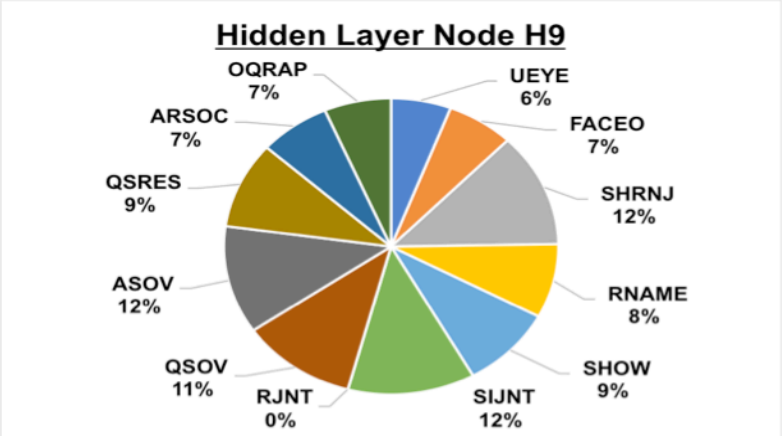


Figure 55. Relative Weights Impacts Node H9 – Module 2

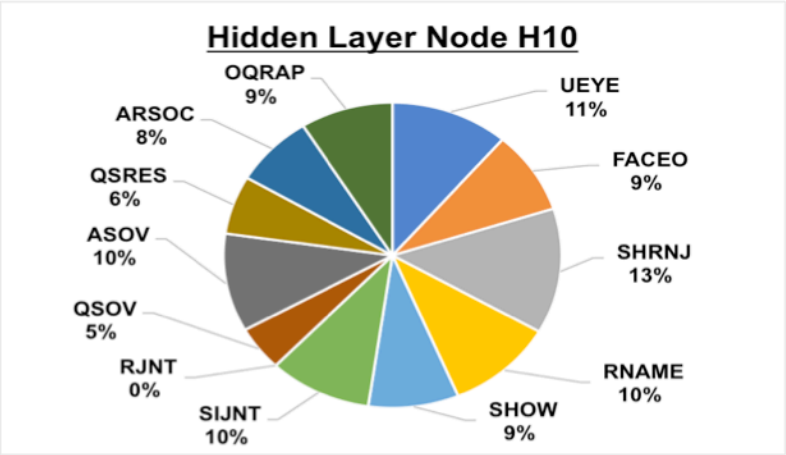


Figure 56. Relative Weights Impacts Node H10 – Module 2

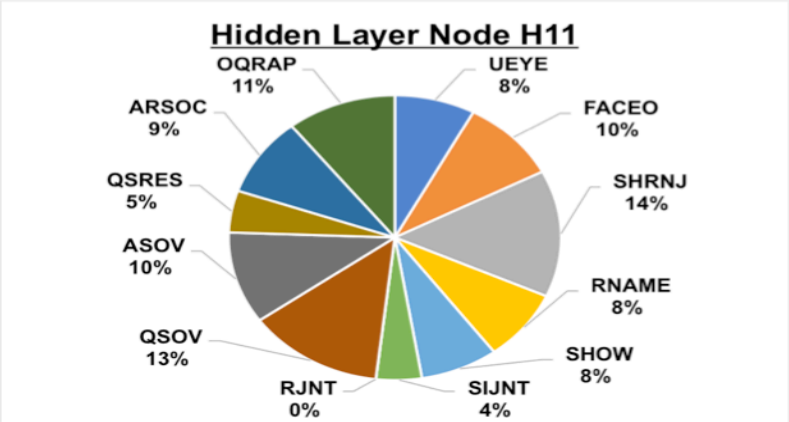


Figure 57. Relative Weights Impacts Node H11 – Module 2

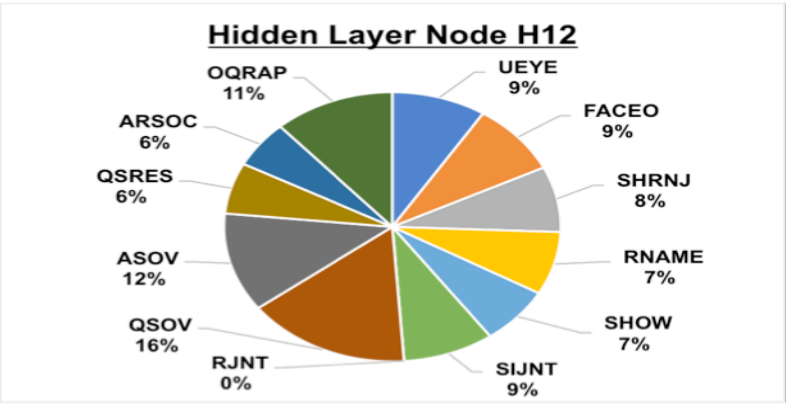


Figure 58. Relative Weights Impacts Node H12 – Module 2

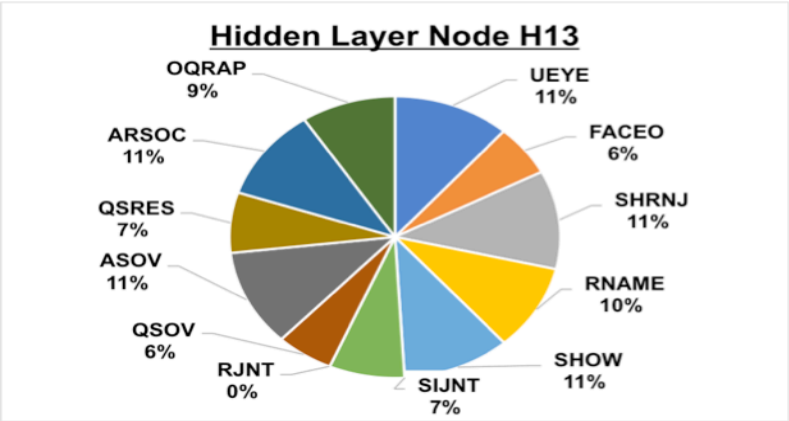


Figure 59. Relative Weights Impacts Node H13 – Module 2

Within the reduced diagnostic nodes, the weights of SIJNT and ARSOC almost perfectly correlate have a correlation of almost 1. Also, ASOV and QSRES correlate with each other with a near perfect correlation.

RJIT was by far the least represented variable in the reduced diagnostic for this section. Its weights were over 100 times smaller than the second least represented variable. Every other variable in this section had significant weights. SHRNJ had the most significant presence. ASOV, OQRAP, SIJNT, SHOW and UEYE followed, and were also closely clustered.

7.4.2.1 Correlation Between Variables

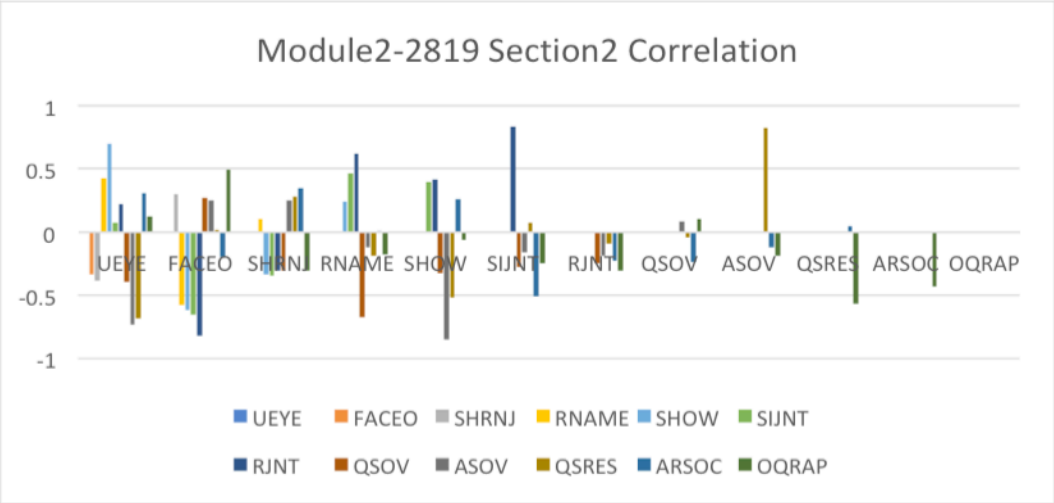


Figure 60. Weights Correlations – Module 2 Section 2

UEYE and SHOW demonstrate a strong positive correlation in weight strength. SIJNT and RJNT also strongly correlate in weight presence, which makes sense given their similar subject nature. Last, OQRAP and FACEO have a strong correlational relationship.

7.4.3 Module 2 Section 3 Analysis

The third section of the reduced Module 2 autoencoder involved reducing 9 original ADOS variables into 6 new ones. The 9 original variables spanned the subjects of Play, Stereotyped Behaviors/Restricted Interests, and Other Abnormal Behaviors. The original variables were Functional Play with Objects (FPLAY), Imagination/Creativity (IMGCR), Unusual Sensory Interest in Play Material/Person (USENS), Hand and Finger and Other Complex Mannerisms (OMAN), Self-Injurious Behavior (SELFINJ), Unusually Repetitive

Interests or Stereotyped Behaviors (URBEH), Overactivity (ACTIVE), Tantrums, Aggression, Negative or Disruptive Behavior (AGG), and Anxiety (ANXTY). The diagrams below shows the weights of the reduced diagnostic of this section.

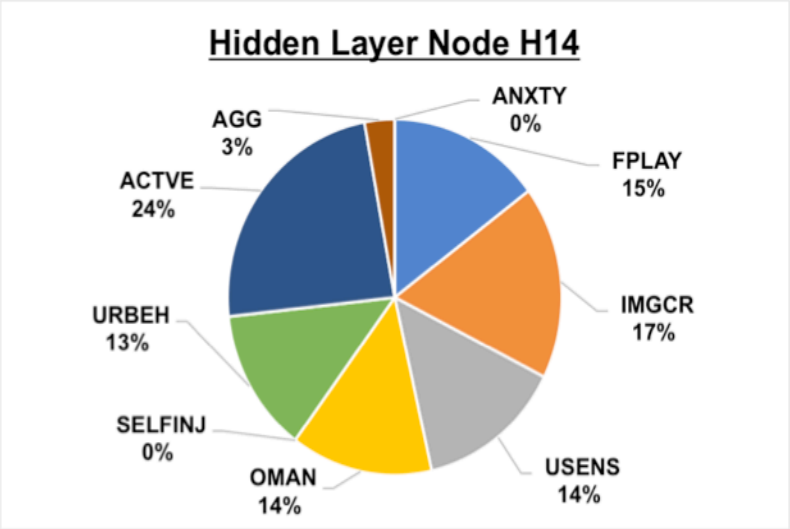


Figure 61. Relative Weights Impacts Node H14 – Module 2

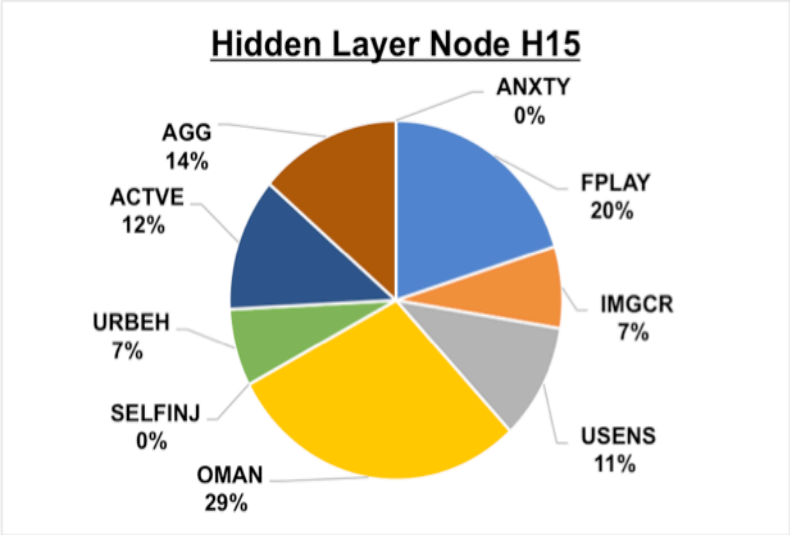


Figure 62. Relative Weights Impacts Node H15 – Module 2

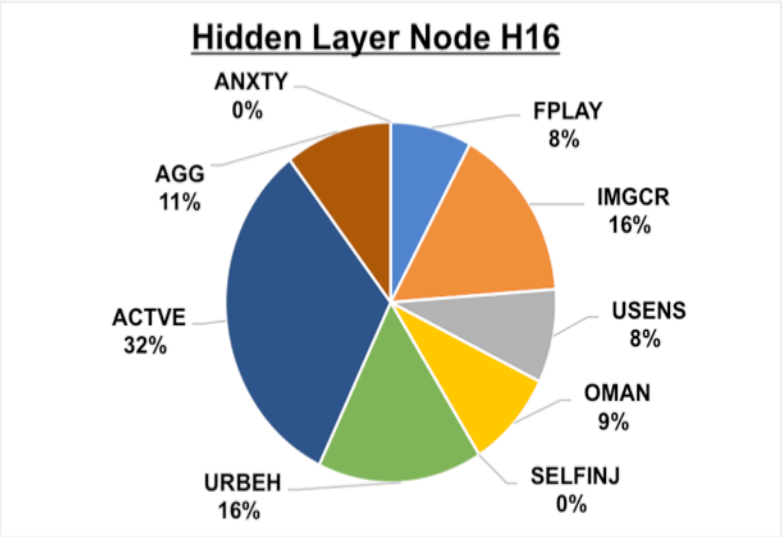


Figure 63. Relative Weights Impacts Node H16 – Module 2

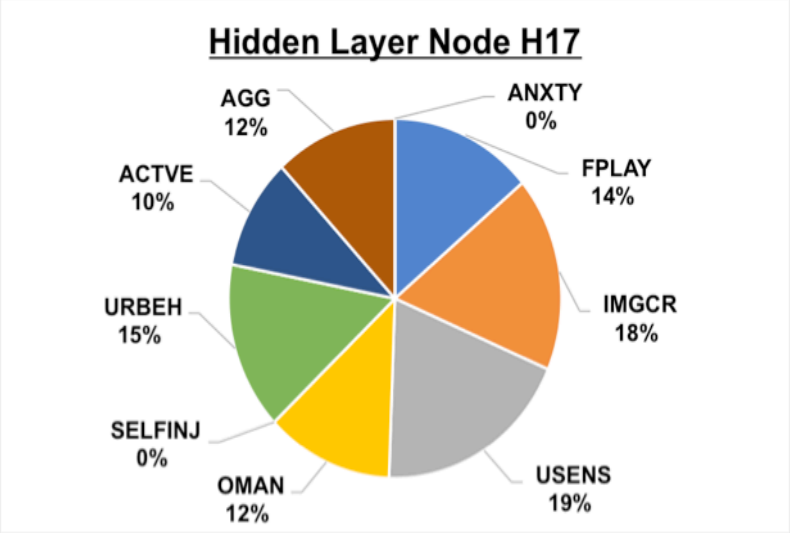


Figure 64. Relative Weights Impacts Node H17 – Module 2

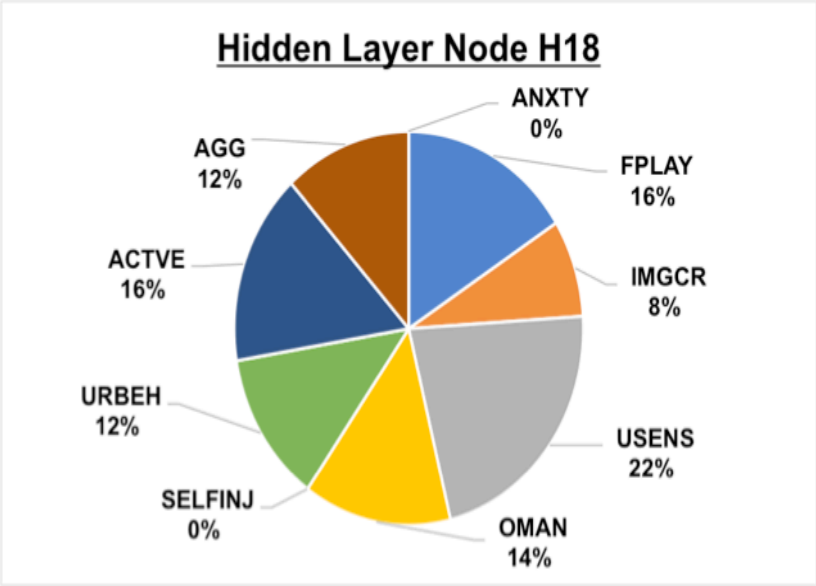


Figure 65. Relative Weights Impacts Node H18 – Module 2

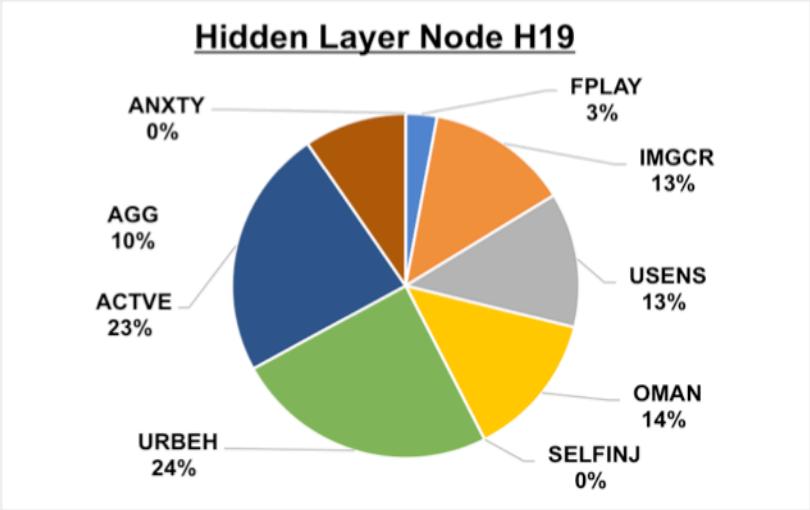


Figure 66. Relative Weights Impacts Node H19 – Module 2

Once again, SELFINJ and ANXTY were by far the least represented variables, as they had completely negligible weights. They were over 1,000 times less influential than any other variables, and were effectively dropped from the diagnostic by the autoencoder. ACTIVE and “OMAN” were the two variables with

greatest representation, and all variables besides the two that were mentioned were represented adequately.

7.4.3.1 Correlation Between Variables

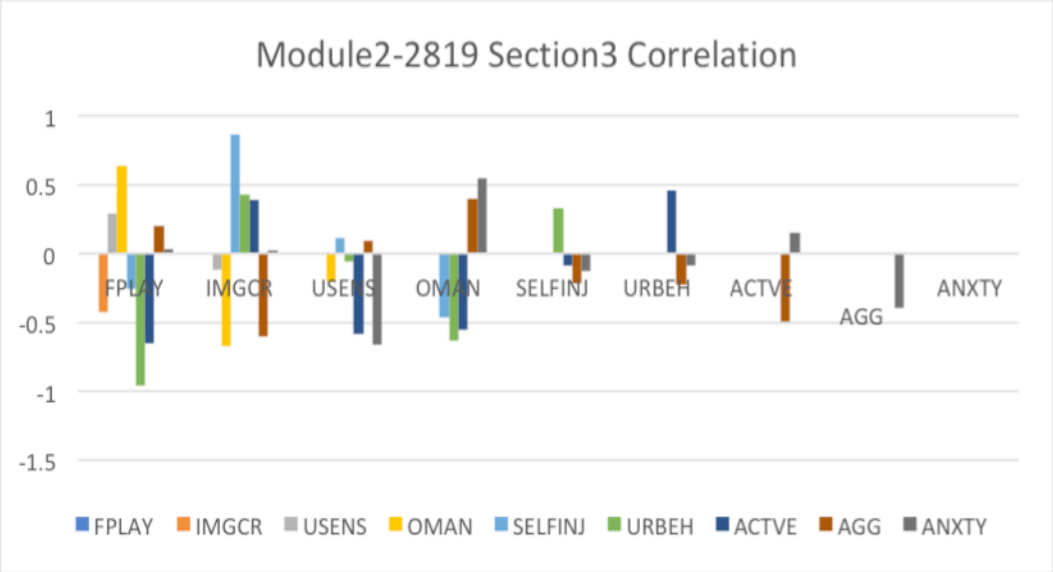


Figure 67. Weights Correlations – Module 2 Section 3

IMGCE and SELFINJ have a strong correlational relationship, although this is probably meaningless given the latter variable's negligible presence in nodes.

7.5 Module 3 Analysis

The diagram below shows Module 3 Autoencoder with Hidden layer of size 19.

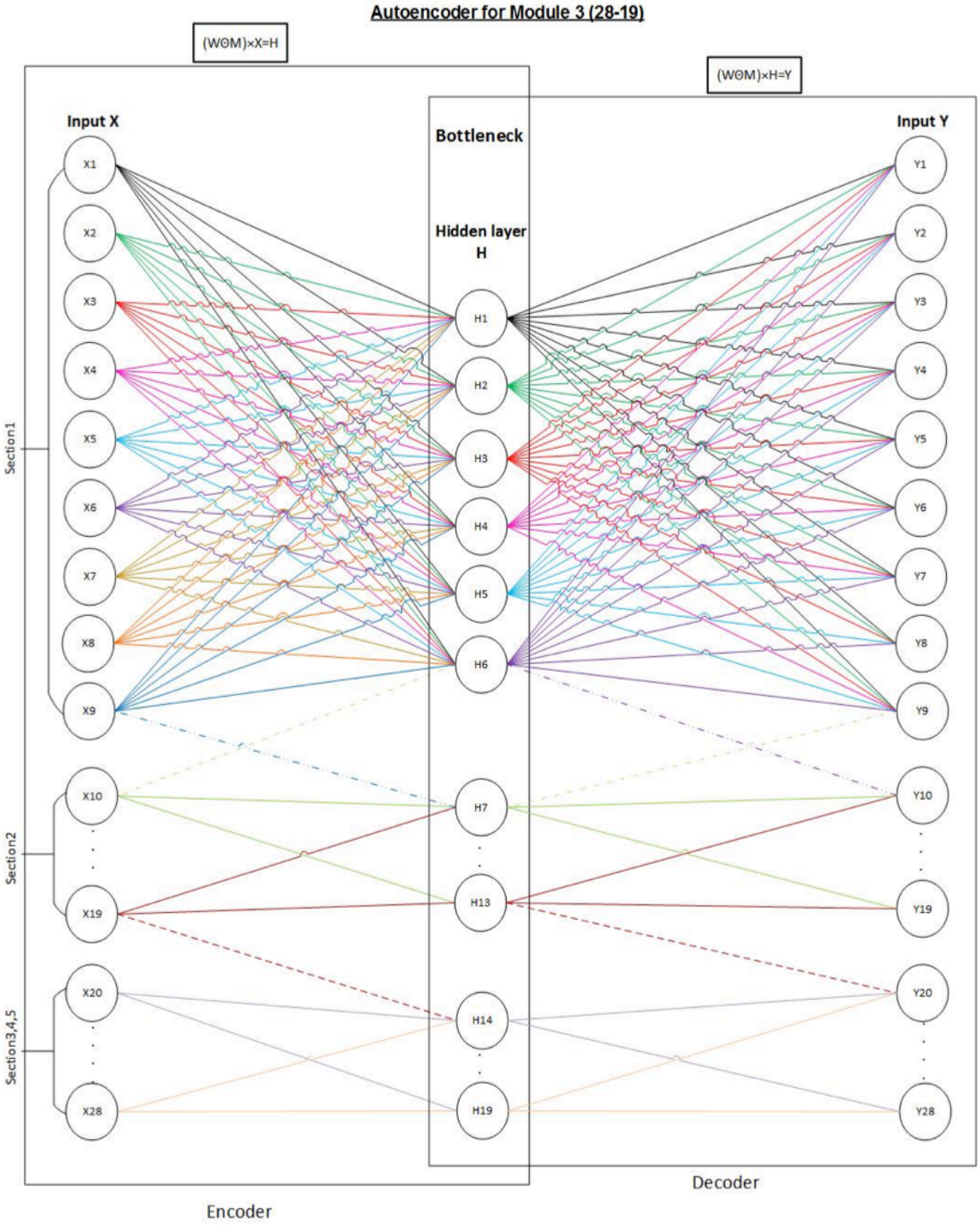


Figure 68. Module 3 Autoencoder with Hidden Layer of Size 19

7.5.1 Module 3 Section 1 Analysis

The first section of Module 3 corresponds to Language and Communication. The 9-variable original section was reduced to a 6-variable section by the autoencoder. The 9 variables in this section are Overall Level of Non-Echoed Spoken Language (OLANG), Speech Abnormalities (SPABN), Immediate Echolalia (IECHO), Stereotyped/Idiosyncratic Use of Words or Phrases (STEREO), Offers Information (OINFO), Asks for Information (AINFO), Reporting of Events (REPRT), Conversation (CONVS), and Descriptive Gestures (DGEST). The diagrams below shows the weights of the reduced diagnostic of this section.

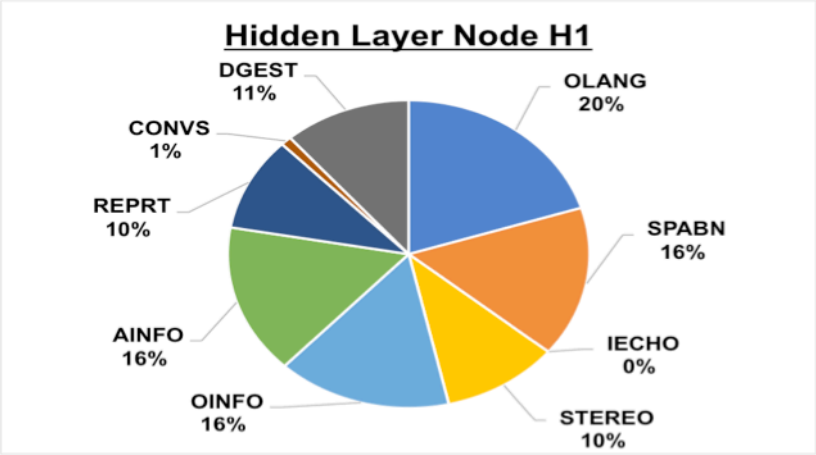


Figure 69. Relative Weights Impacts Node H1 – Module 3

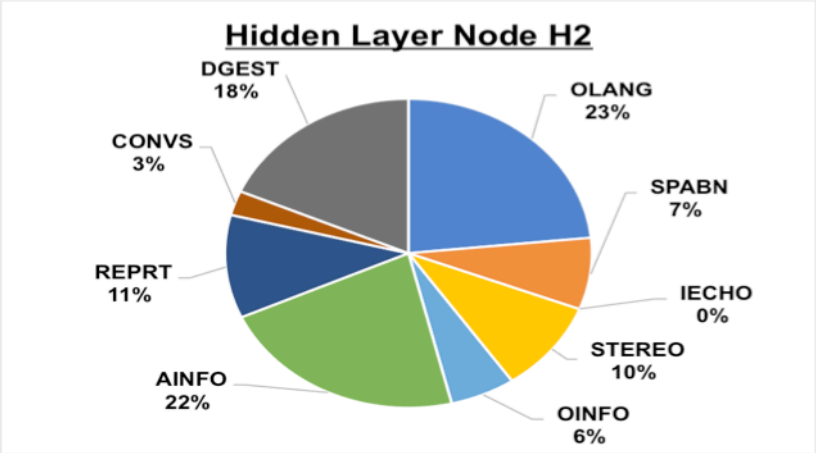


Figure 70. Relative Weights Impacts Node H2 – Module 3

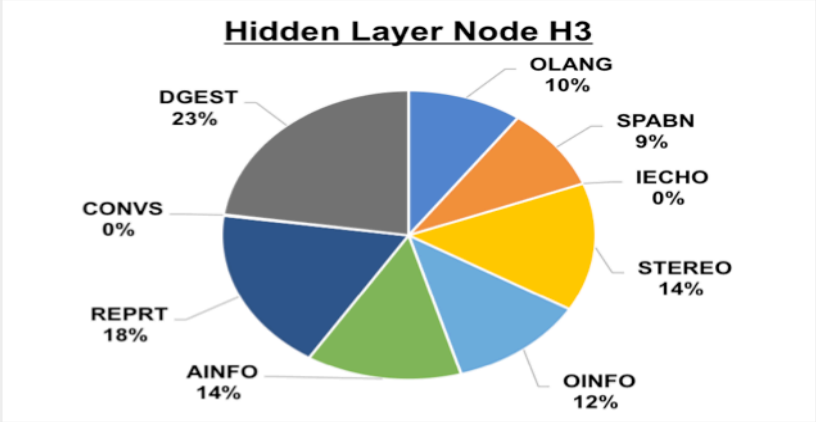


Figure 71. Relative Weights Impacts Node H3 – Module 3

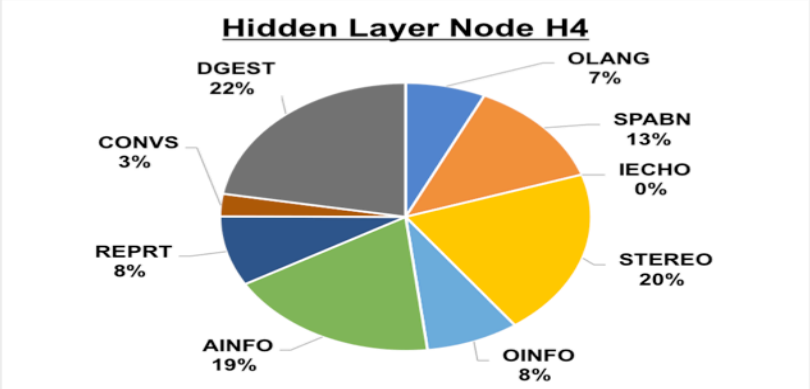


Figure 72. Relative Weights Impacts Node H4 – Module 3

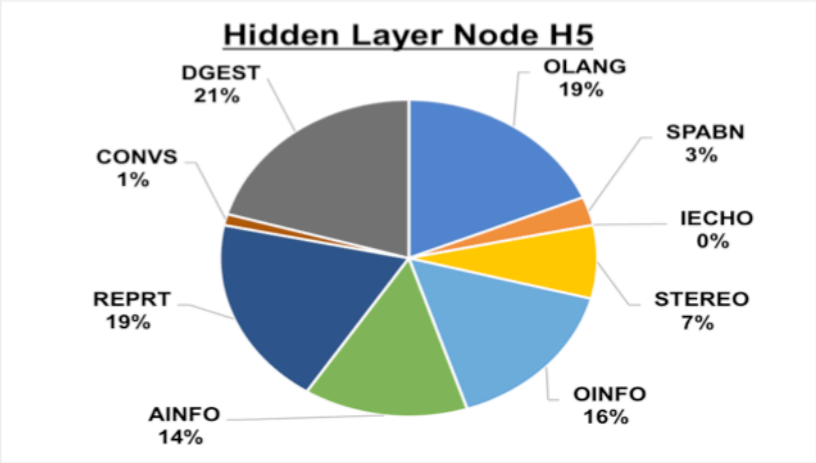


Figure 73. Relative Weights Impacts Node H5 – Module 3

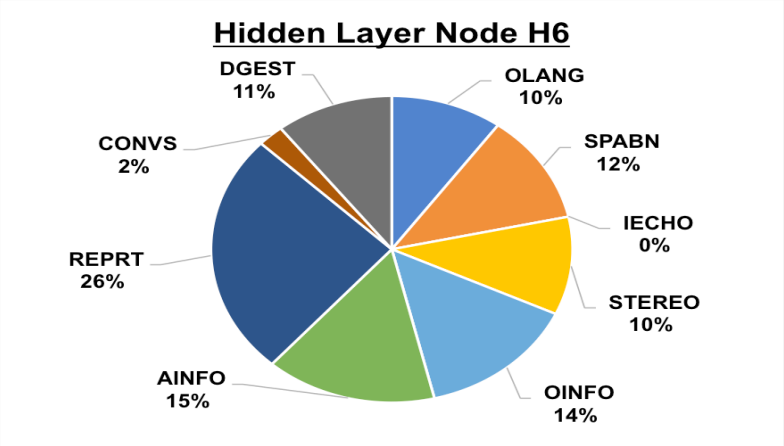


Figure 74. Relative Weights Impacts Node H6 – Module 3

IECHO was almost completely dropped from the diagnostic, as its weights were several orders of magnitude smaller than any others in the section. CONVS was the second least represented variable. While its weights were significantly higher than IECHO, it lagged significantly behind the other variables.

All other weights were fairly strong, with DGEST as the highest, followed by AINFO, REPRT and OLANG. These variables had weights significantly than the other variables.

7.5.1.1 Correlation Between Variables

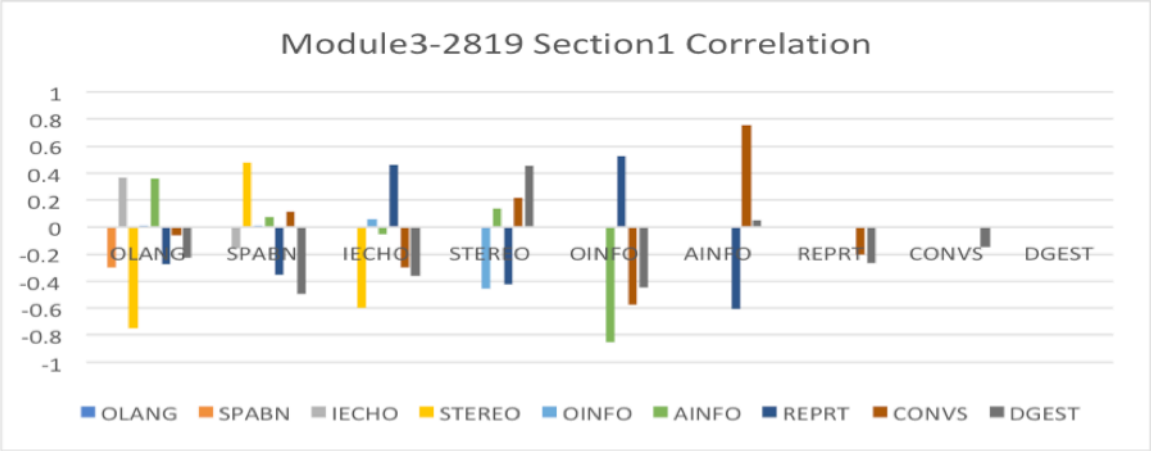


Figure 75. Weights Correlations – Module 3 Section 1

AINFO and CONVS have a strong positive correlational relationship.

7.5.2 Module 3 Section 2 Analysis

Section 2 of Module 3 involves Reciprocal Social Interaction. The original diagnostic included 10 variables, and the reduced diagnostic contains 7 variables. The variables are Unusual Eye Contact (UEYE), Facial Expressions Directed to Examiner (FACEO), Language Production and Linked Nonverbal Communication (LLNVC), Shared Enjoyment in Interaction (SHRNJ), Comments on Others' Emotions/Empathy (EMPTH), Insight into Typical Social Situations and Relationships (INSIG), Quality of Social Overtures (QSOV), Quality of Social

Response (QSRES), Amount of Reciprocal Social Communication (ARSOC), and Overall Quality of Rapport (OQRAP). The diagrams below shows the weights of the reduced diagnostic of this section.

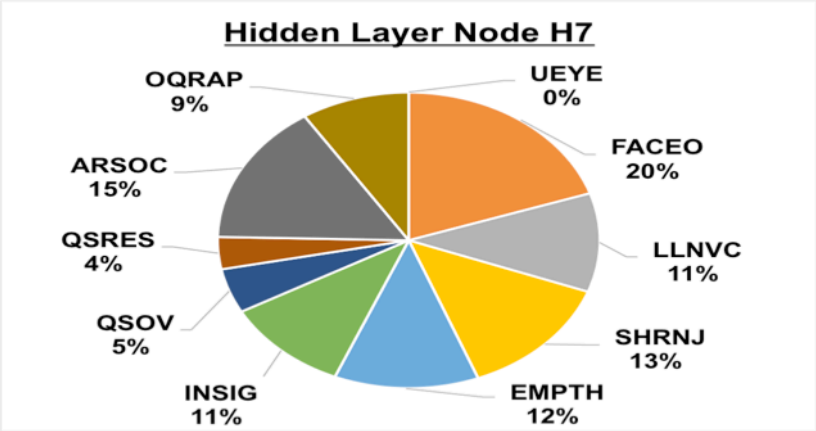


Figure 76. Relative Weights Impacts Node H7 – Module 3

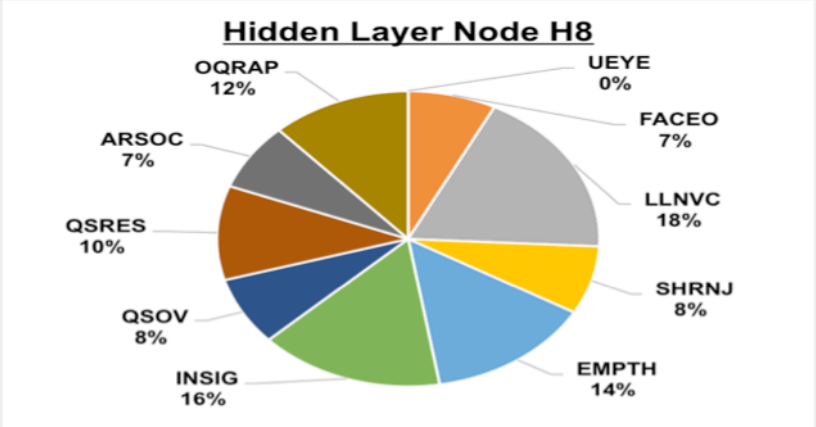


Figure 77. Relative Weights Impacts Node H8 – Module 3

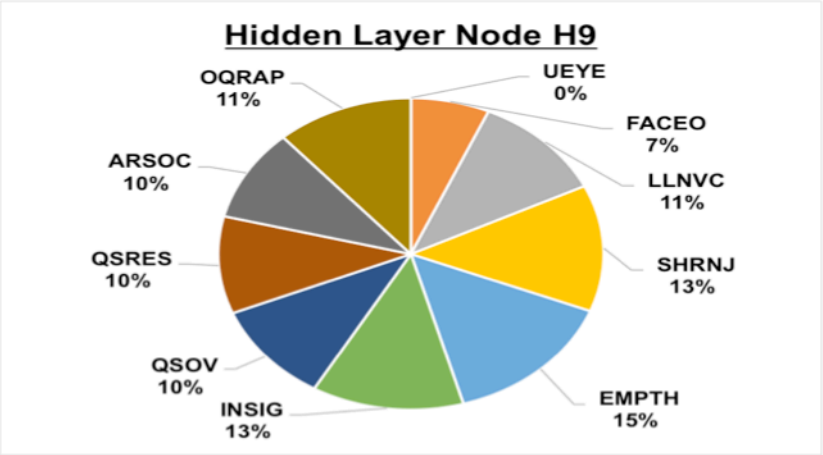


Figure 78. Relative Weights Impacts Node H9 – Module 3

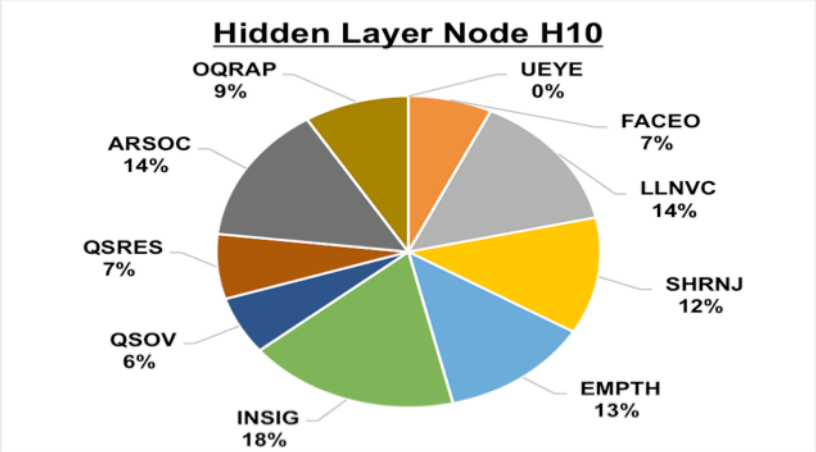


Figure 79. Relative Weights Impacts Node H10 – Module 3

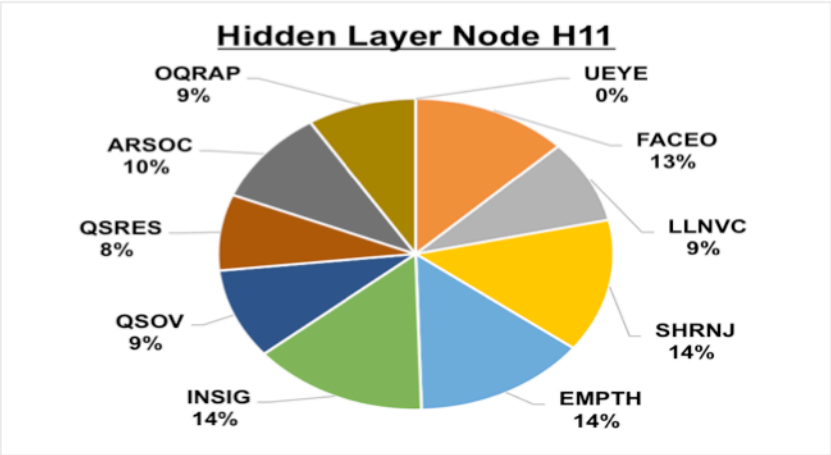


Figure 80. Relative Weights Impacts Node H11 – Module 3

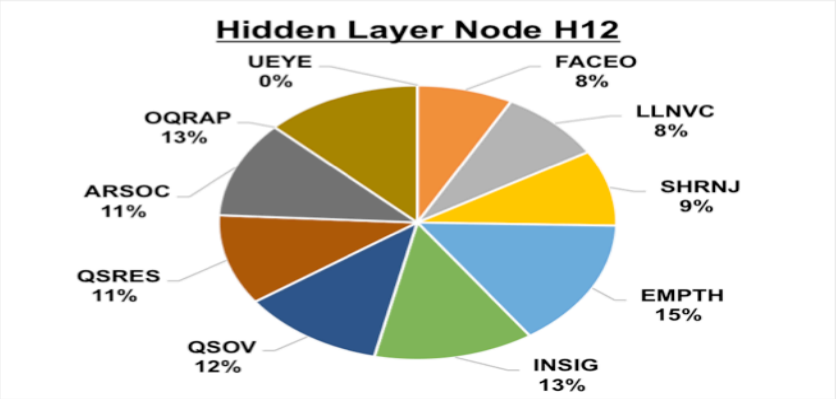


Figure 81. Relative Weights Impacts Node H12 – Module 3

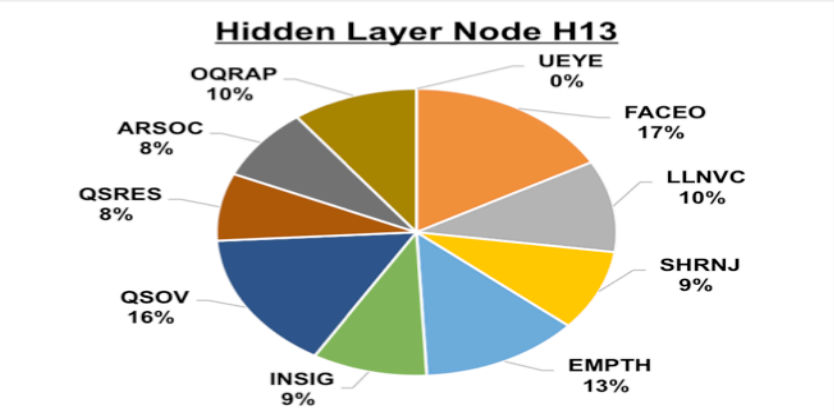


Figure 82. Relative Weights Impacts Node H13 – Module 3

UEYE was the least represented variable by a large margin and was effectively dropped from the reduced diagnostic in this instance. All other variables had significant weights attached. EMPTHY and INSIG were the most impactful variables by a significant margin. The majority of the other variables were clustered closely together in weight strength, with QSRES and QSOV slightly lagging behind the others.

75.2.1 Correlation Between Variables

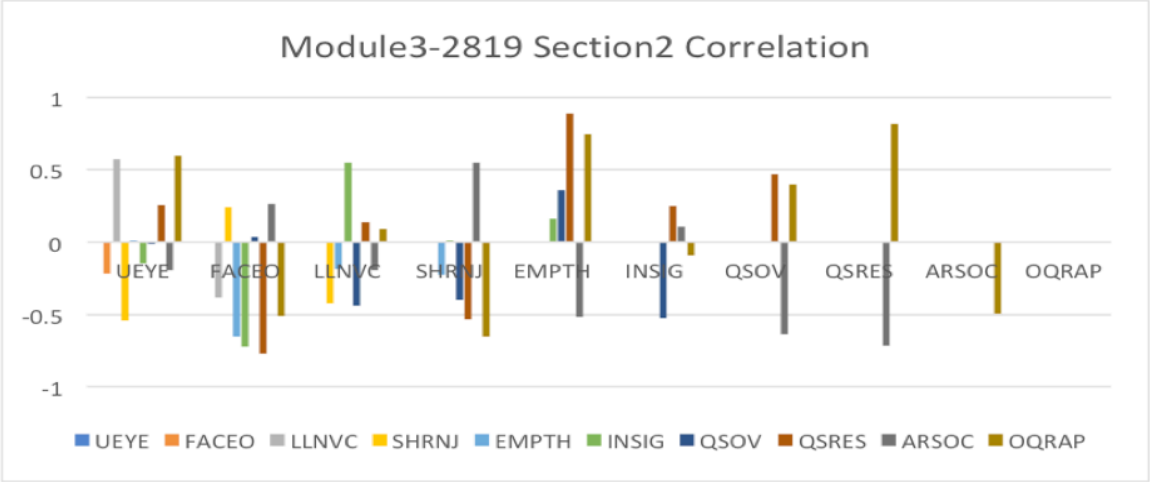


Figure 83. Weights Correlations – Module 3 Section 2

EMPTH and QSRES have a strong correlational relationship in weight presence. OQRAP also correlates strongly with both EMPTH and QSRES. These correlations indicate shared occurrences between these three variables in some capacity.

7.5.3 Module 3 Section 3 Analysis

The third section of the reduced diagnostic combines sections in the original that cover Imagination, Stereotyped Behaviors/Restricted Interests, and Other Abnormal Behaviors. There were 9 variables originally in these sections, and these were reduced to 6 variables by the autoencoders. The variables in these sections are Imagination/Creativity (IMGCR), Unusual Sensory Interest In Play Material/Person (USENS), Hand and Finger and Other Complex Mannerisms (OMAN), Self-Injurious Behavior (SELFINJ), Excessive Interest in or References to Unusual or Highly Specific Topics or Objects or Repetitive Behaviors (TOPIC), Compulsions or Rituals (RITL), Overactivity/Agitation (ACTIVE), Tantrums/Aggression (AGG), and Anxiety (ANXTY). The diagrams below shows the weights of the reduced diagnostic of this section.

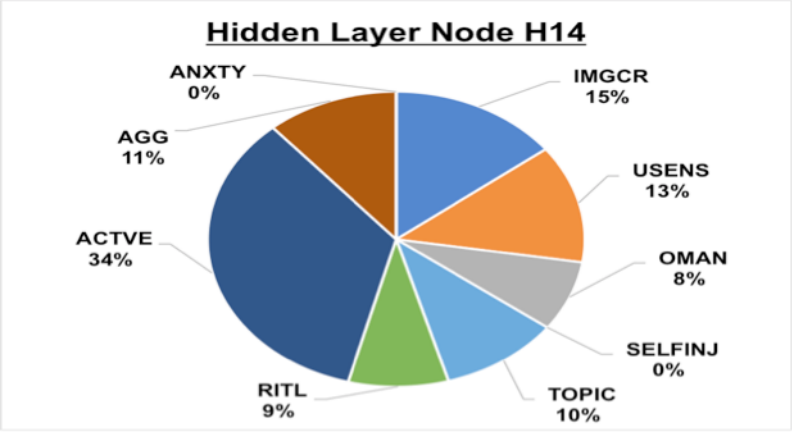


Figure 84. Relative Weights Impacts Node H14 – Module 3

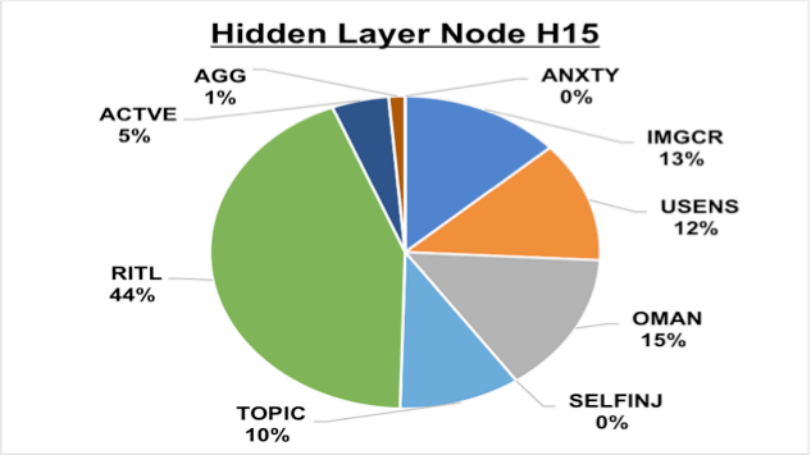


Figure 85. Relative Weights Impacts Node H15 – Module 3

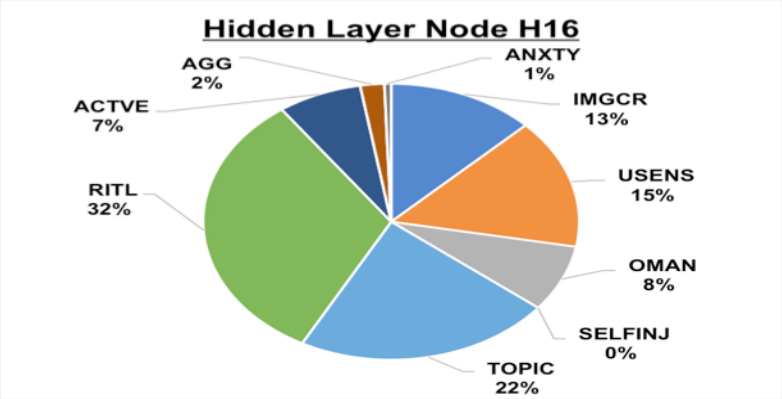


Figure 86. Relative Weights Impacts Node H16 – Module 3

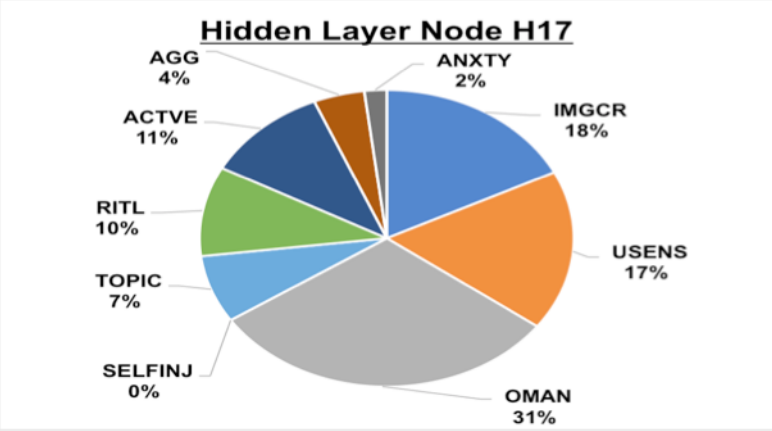


Figure 87. Relative Weights Impacts Node H17 – Module 3

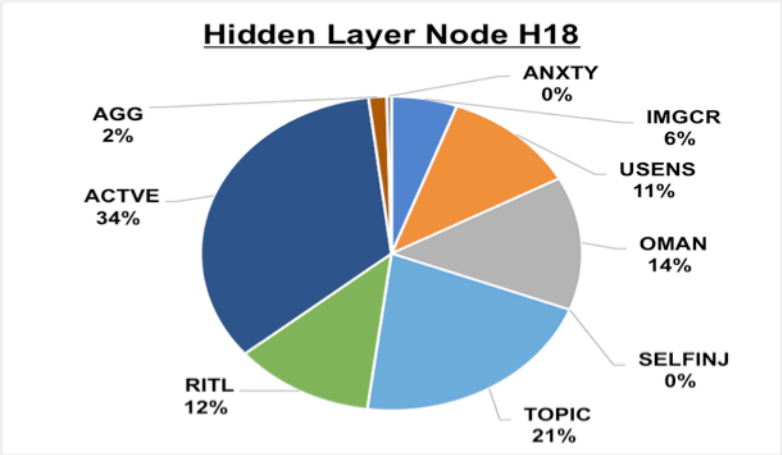


Figure 88. Relative Weights Impacts Node H18 – Module 3

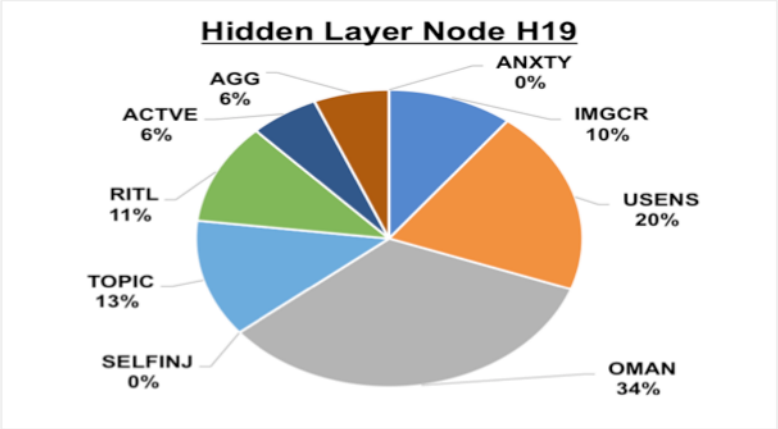


Figure 89. Relative Weights Impacts Node H19 – Module 3

As in earlier cases, SELFINJ had by far the weakest weights, orders of magnitude below anything else. ANXTY also had very weak weights that were significantly below everything else. Nonetheless, there was a sizeable gap between ANXTY and SELFINJ". All other weights were fairly strong, with TOPIC or RITL, OMAN, and ACTIVE demonstrating the strongest connections by a solid margin. The other variables were clustered close together with the exception of

AGG, which lagged behind other variables but ahead of the two least important ones.

7.5.3.1 Correlation Between Variables

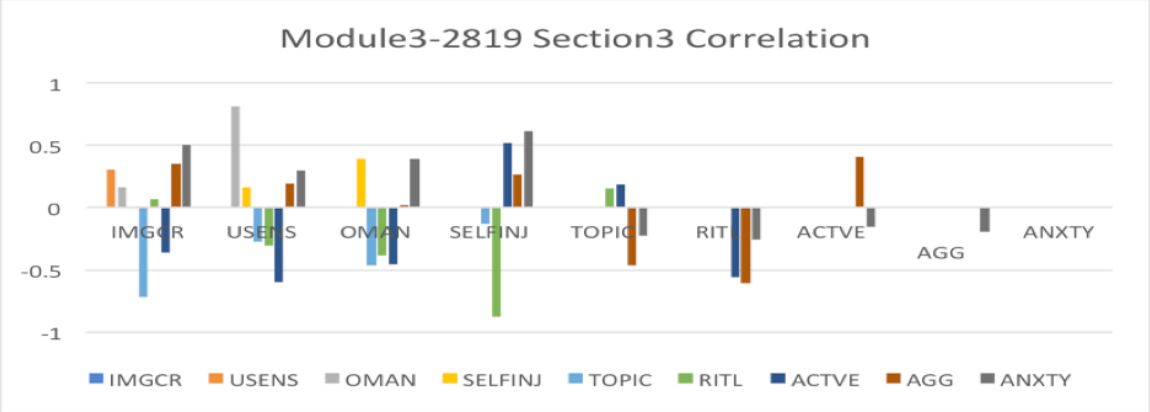


Figure 90. Weights Correlations – Module 3 Section 3

USENS and OMAN show a strong positive correlational relationship.

7.6 Module 4 Analysis

The diagram below shows Module 4 Autoencoder with Hidden layer of size 19.

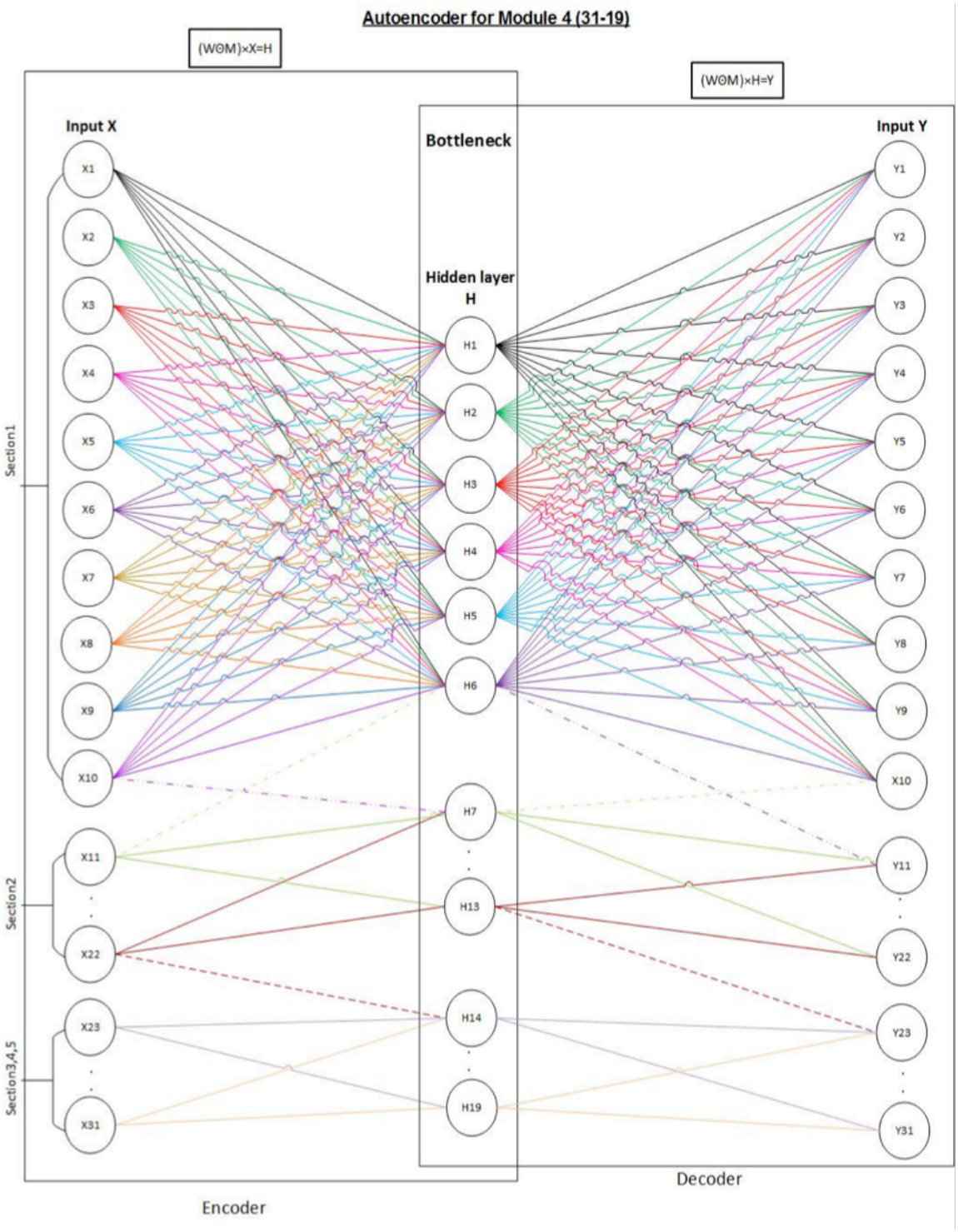


Figure 91. Module 4 Autoencoder with Hidden Layer of Size 19

7.6.1 Module 4 Section 1 Analysis

The first section of Module 4 corresponds to Language and Communication. The 10-variable original section was reduced to a 6-variable section by the autoencoder. The 10 variables in this section are Overall Level of Non-Echoed Spoken Language (OLANG), Speech Abnormalities (SPABN), Immediate Echolalia (IECHO), Stereotyped/Idiosyncratic Use of Words or Phrases (STEREO), Offers Information (OINFO), Asks for Information (AINFO), Reporting of Events (REPRT), Conversation (CONVS), Descriptive Gestures (DGEST), Emphatic or Emotional Gestures (EGEST). The diagrams below shows the weights of the reduced diagnostic of this section.

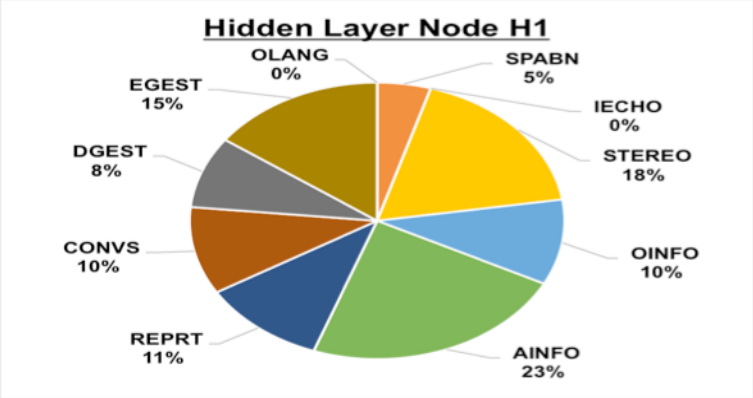


Figure 92. Relative Weights Impacts Node H1 – Module 4

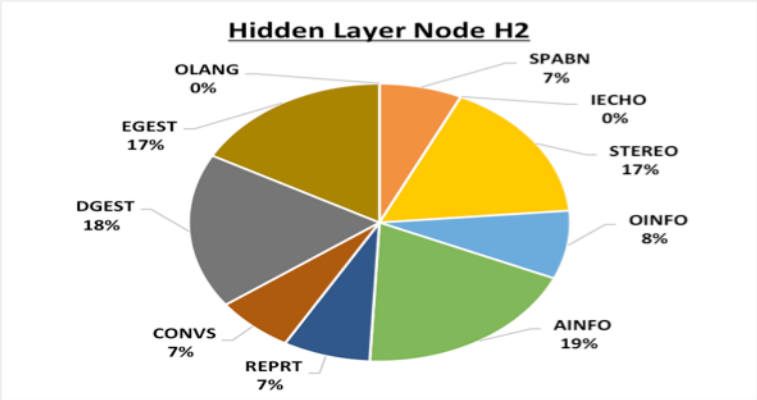


Figure 93. Relative Weights Impacts Node H2 – Module 4

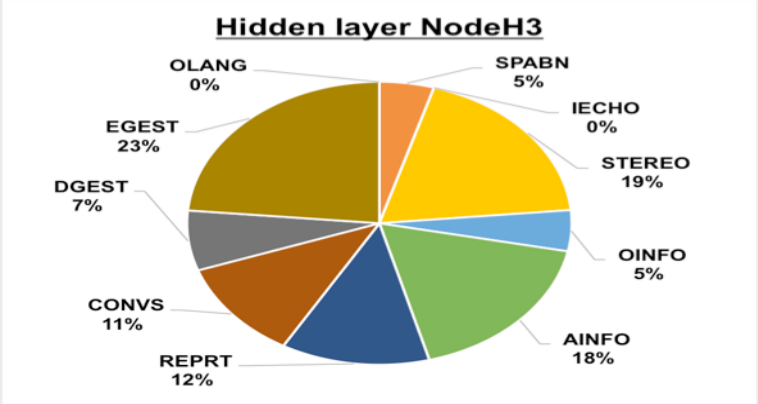


Figure 94. Relative Weights Impacts Node H3 – Module 4

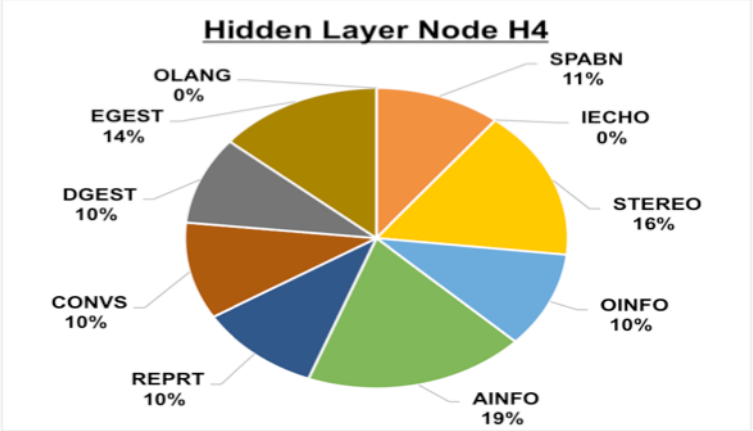


Figure 95. Relative Weights Impacts Node H4 – Module 4

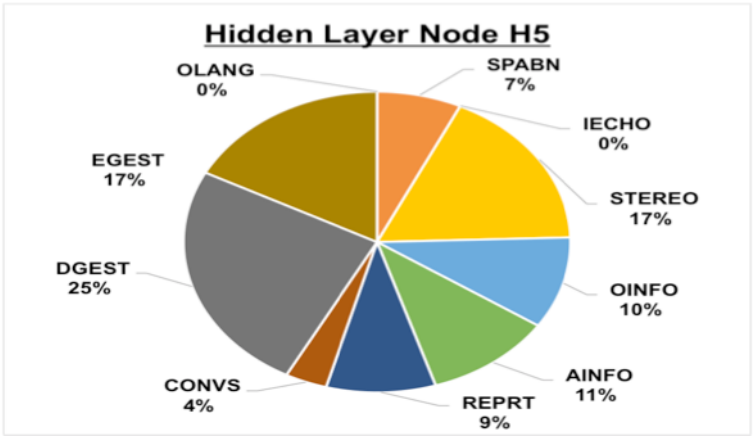


Figure 96. Relative Weights Impacts Node H5 – Module 4

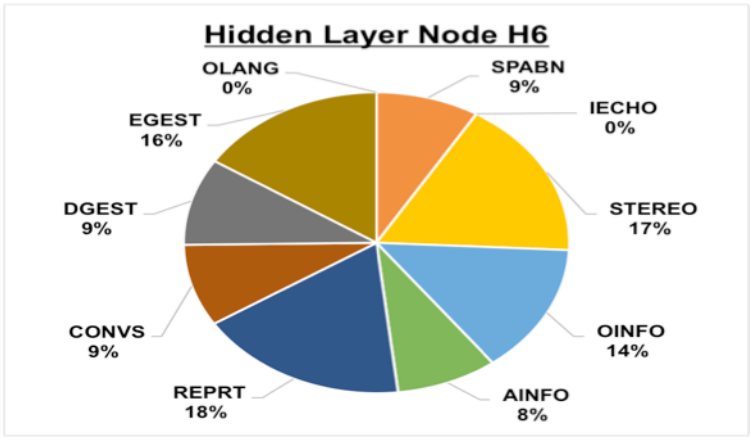


Figure 97. Relative Weights Impacts Node H6 – Module 4

OLANG and IECHO were the two least represented variables and had weights with absolute values several orders of magnitude smaller than the other variables. All other variables were represented with significant weights, with STEREO, EGEST, and AINFO leading the pack. Those three variables had weights higher than the others by a notable amount.

7.6.1.1 Correlation Between Variables

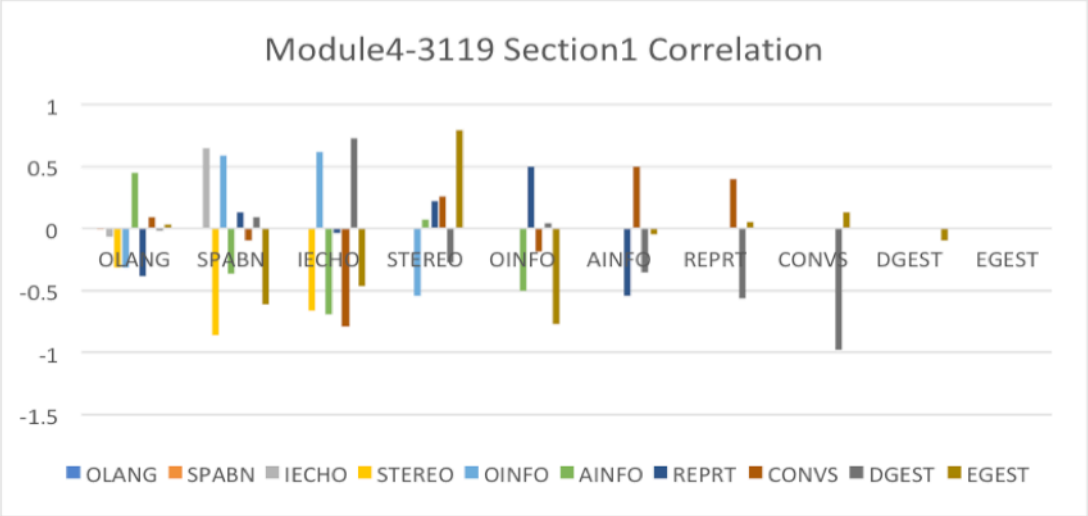


Figure 98. Weights Correlations – Module 4 Section 1

STEREO AND EGEST correlated positively to a strong degree.

7.6.2 Module 4 Section 2 Analysis

Section 3 of Module 4 involves Reciprocal Social Interaction. The original diagnostic included 12 variables, and the reduced diagnostic contains 7 variables. The variables are Unusual Eye Contact (UEYE), Facial Expressions Directed to Examiner (FACEO), Language Production and Linked Nonverbal Communication (LLNVC), Shared Enjoyment in Interaction (SEI), Communication of Own Affect (CAFF), Comments on Others' Emotions/Empathy (EMPTH), Insight into Typical Social Situations and Relationships (INSIG), Responsibility (RESP), Quality of Social Overtures (QSOV), Quality of Social Response (QSRES), Amount of Reciprocal Social Communication (ARSOC), and Overall

Quality of Rapport (OQRAP). The diagrams below shows the weights of the reduced diagnostic of this section.

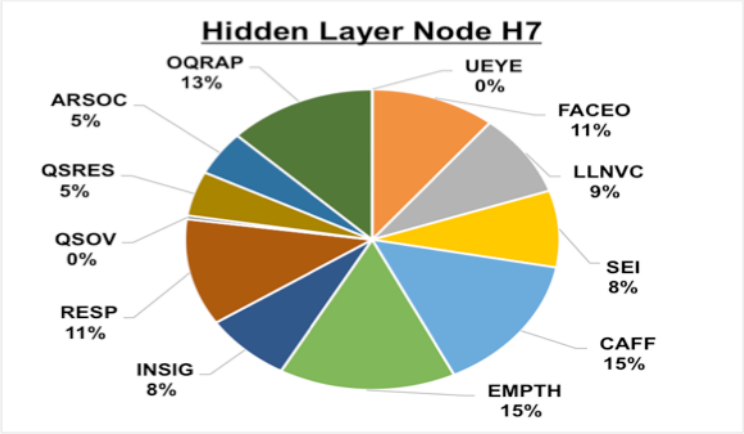


Figure 99. Relative Weights Impacts Node H7 – Module 4

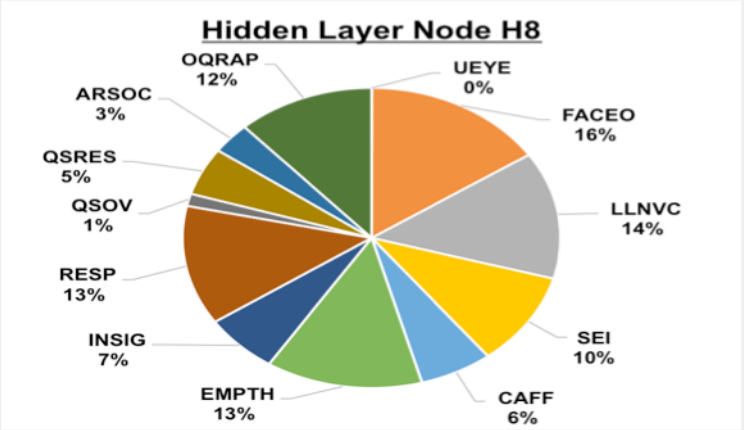


Figure 100. Relative Weights Impacts Node H8 – Module 4

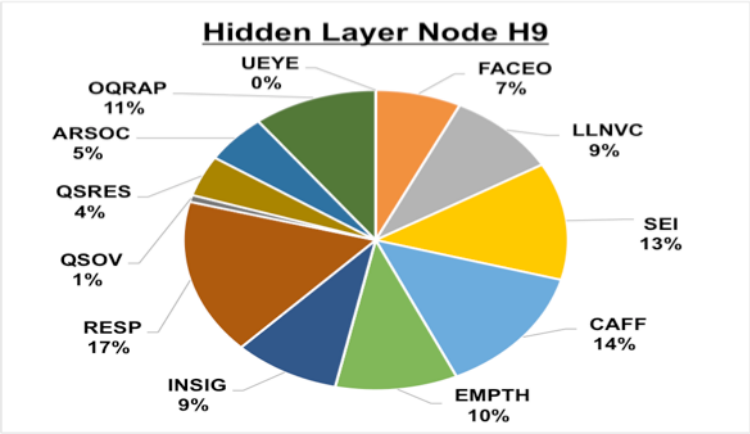


Figure 101. Relative Weights Impacts Node H9 – Module 4

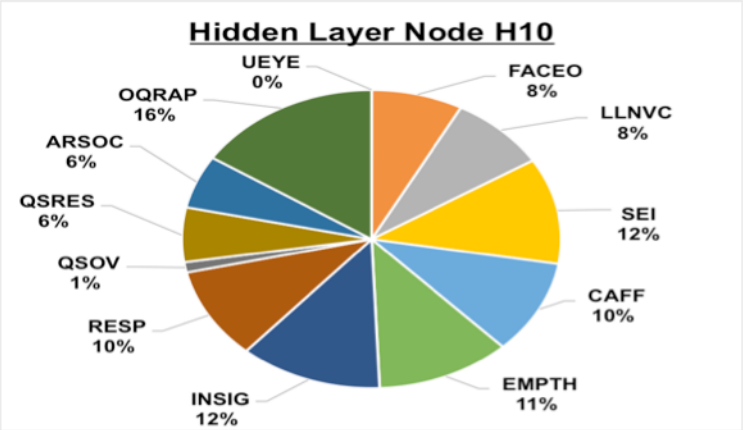


Figure 102. Relative Weights Impacts Node H10 – Module 4

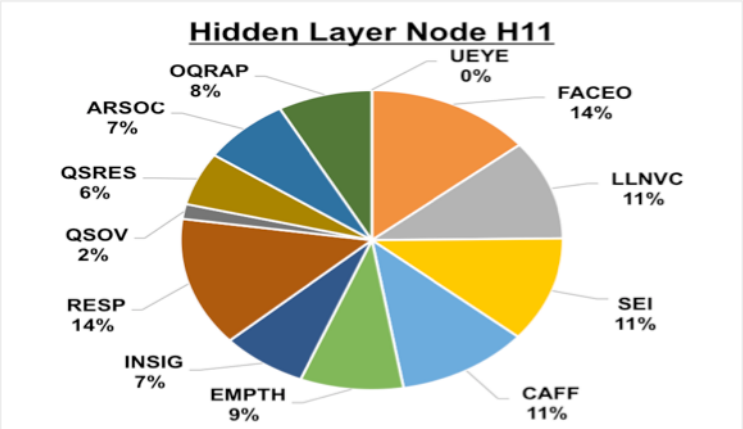


Figure 103. Relative Weights Impacts Node H11 – Module 4

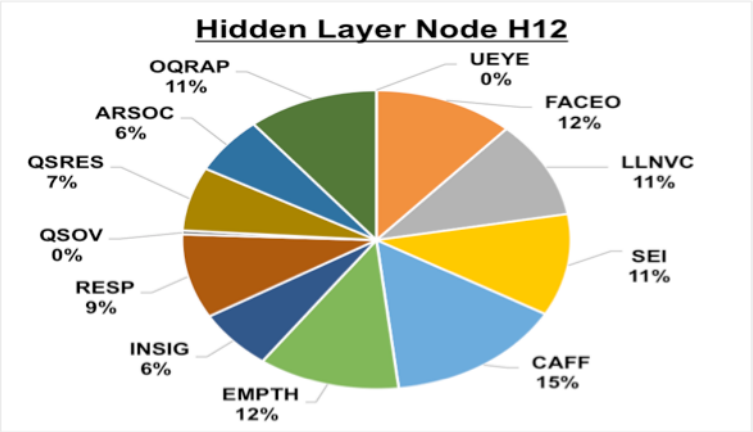


Figure 104. Relative Weights Impacts Node H12 – Module 4

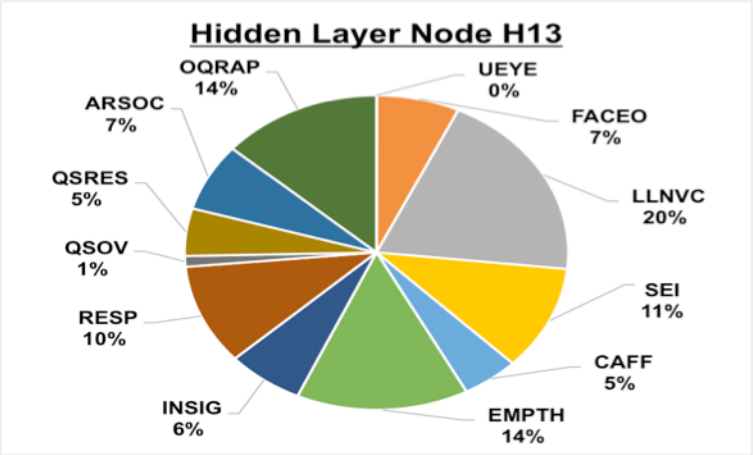


Figure 105. Relative Weights Impacts Node H13 – Module 4

UEYE had the lowest absolute value weights by a large amount. It was effectively removed from the reduced diagnostic. QSOV was clearly the second least important input variable to the reduced diagnostic. QSRES and ARSOC were the next two variables in order of ascending absolute value weights, and had almost identical weights, at a significantly higher level than QSOV. INSIG was the next variable in ascending order. All other variables were closely

clustered together, with Others EMPTH featuring the highest absolute value weights.

7.6.2.1 Correlation Between Variables

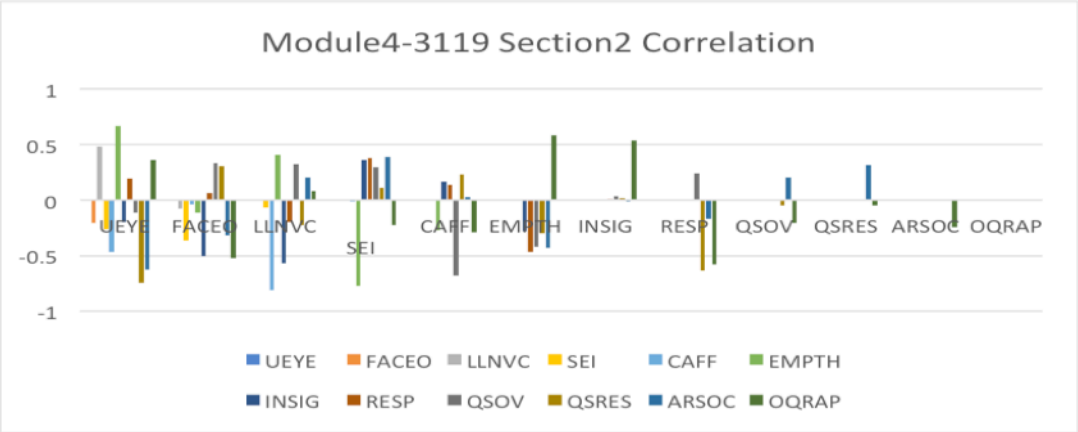


Figure 106. Weights Correlations – Module 4 Section 2

All correlation in this section are weak.

7.6.3 Module 4 Section 3 Analysis

The third section of the reduced diagnostic combines sections in the original that cover Imagination, Stereotyped Behaviors/Restricted Interests, and Other Abnormal Behaviors. There were 9 variables originally in these sections, and these were reduced to 6 variables by the autoencoders. The variables in these sections are Imagination/Creativity (IMGCR), Unusual Sensory Interest In Play Material/Person (USENS), Hand and Finger and Other Complex Mannerisms (OMAN), Self-Injurious Behavior (SELFINJ), Excessive Interest in or References to Unusual or Highly Specific Topics or Objects or Repetitive Behaviors (TOPIC), Compulsions or Rituals (RITL), Overactivity/Agitation

(ATIVE), Tantrums/Aggression (AGG), and Anxiety (ANXTY). The diagrams below shows the weights of the reduced diagnostic of this section.

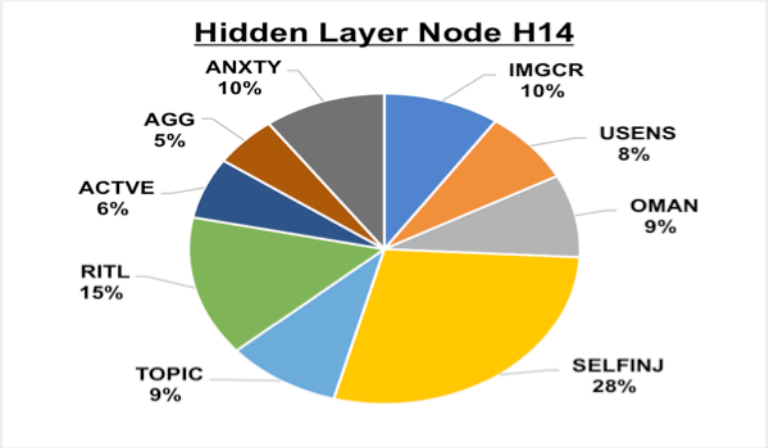


Figure 107. Relative Weights Impacts Node H14 – Module 4

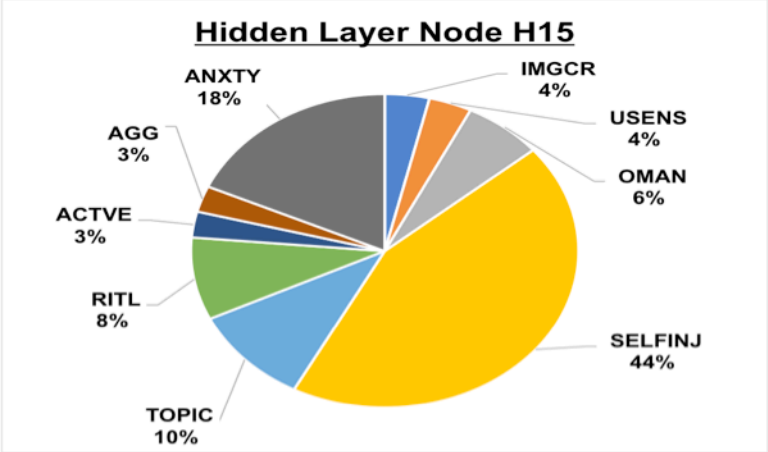


Figure 108. Relative Weights Impacts Node H15 – Module 4

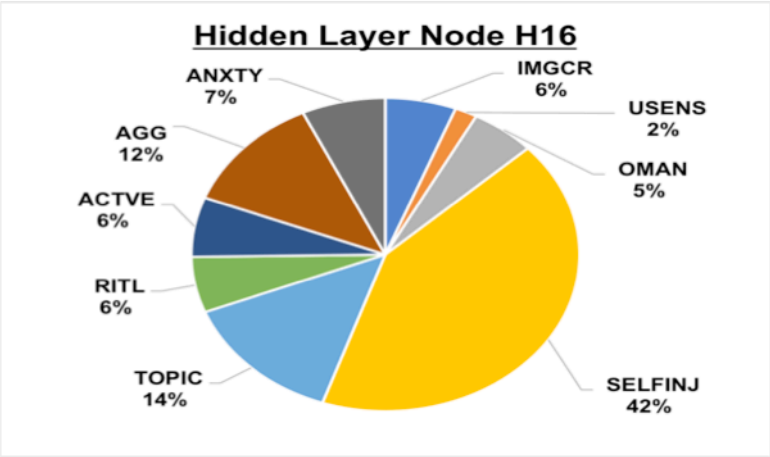


Figure 109. Relative Weights Impacts Node H16 – Module 4

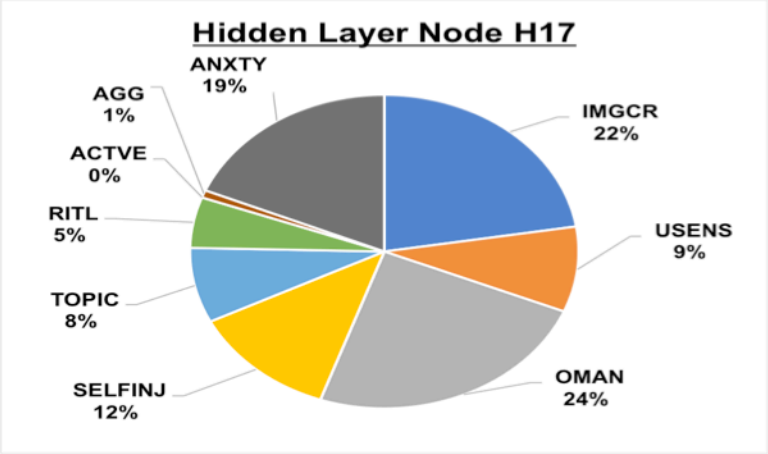


Figure 110. Relative Weights Impacts Node H17 – Module 4

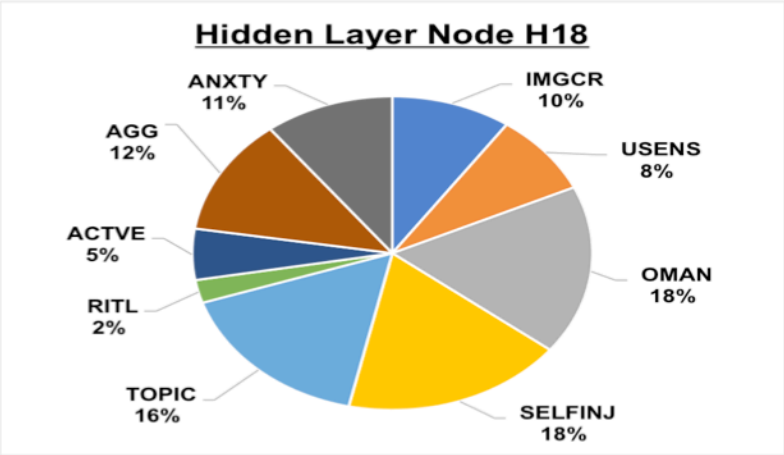


Figure 111. Relative Weights Impacts Node H18 – Module 4

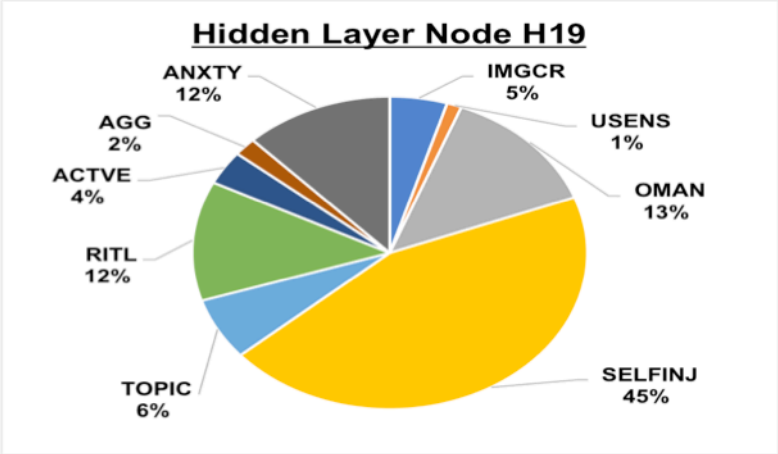


Figure 112. Relative Weights Impacts Node H19 – Module 4

The weights for this section were unusually balanced, and no variables were completely dropped. The nodes in this section were shared between input variables far more than many other sections of other modules. Interestingly enough, SELFINJ and ANXTY were the two variables with the highest strength of weights. SELFINJ in particular had cumulative weights that were more than

double the next most impactful variable. OMAN had weight strength almost equivalent to ANXTY, followed by TOPIC and IMGCR.

7.6.3.1 Correlation Between Variables

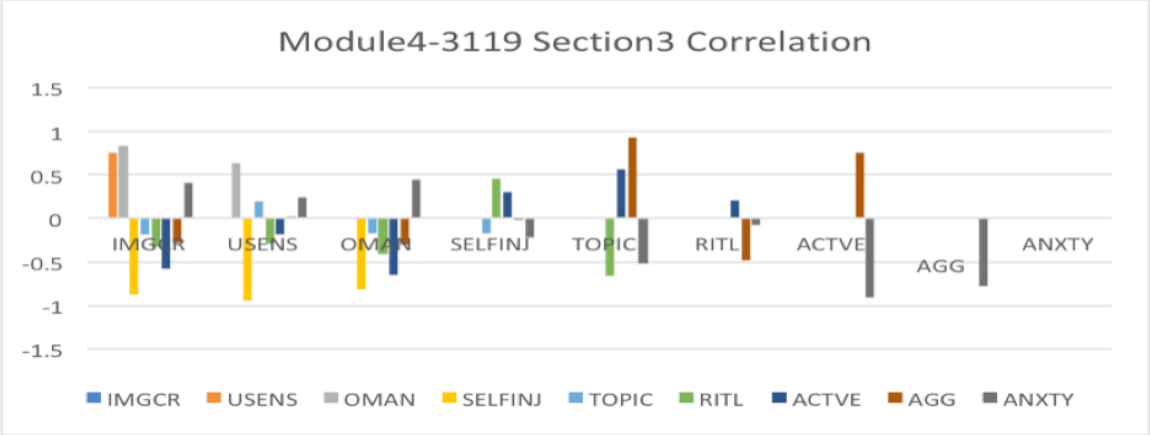


Figure 113. Weights Correlations – Module 4 Section 3

IMGCE and OMAN had a strong positive correlational relationship. (ACTIVE) and AGG also correlated strongly in the positive direction.

7.7 Insights Across Module 1 Architectures

7.7.1 Module 1 Section 1 Insights

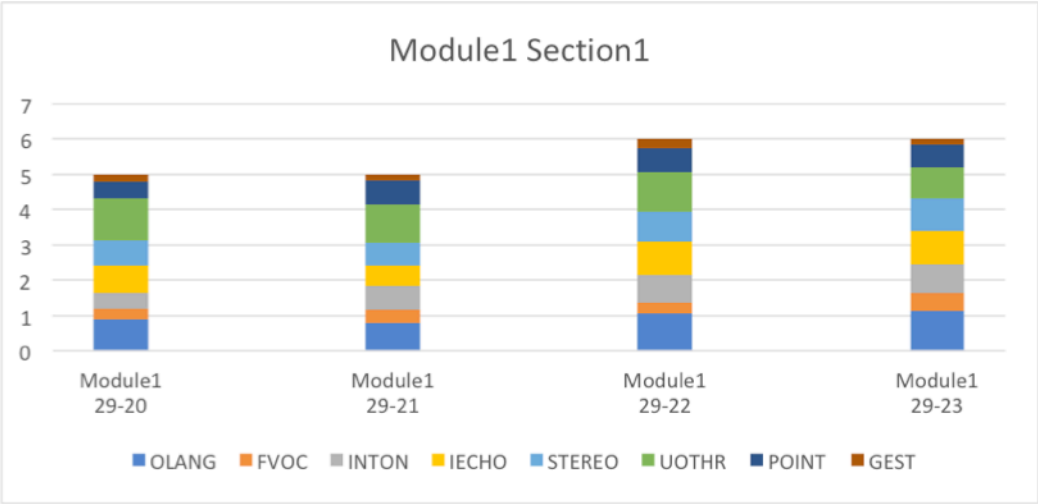


Figure 114. Module 1 Section 1 Insights

GEST was the least important variable across the board. None of its weights were negligible, but it was nonetheless the lowest ranked in terms of absolute value weights across the board. FVOC had the second lowest absolute value weights across the board, and displayed a greater degree of weight variance than GEST.

UOTHER was ranked highest in absolute value weights in three of the four autoencoders, representing the lowest dimensional reductions. OLANG was ranked second in absolute value weights in three lowest dimensional reductions, and was ranked first in the other.

7.7.2 Module 1 Section 2 Insights

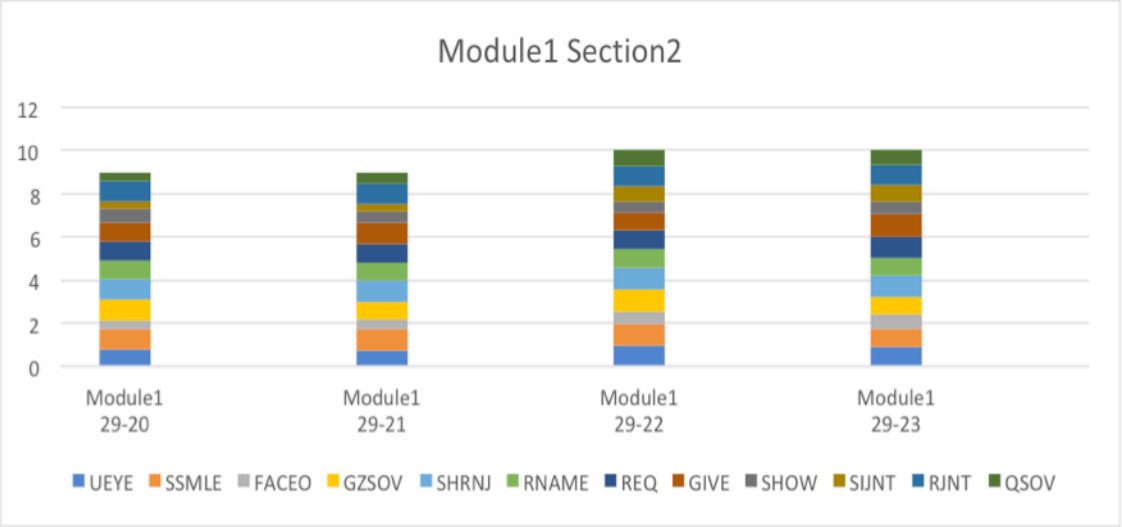


Figure 115. Module 1 Section 2 Insights

The different architectures in Section 2 of Module 1 differed to a far greater degree than in many other sections. It seems that different autoencoders converged upon radically different minima. GIVE and “GZSOV” had high absolute value weights across the board.

Interestingly enough, the 29-20 architecture showed a greater degree of similarity with the 29-22 architecture. A similar trend was seen between the 29-21 and 29-23 architectures.

7.7.3 Module 1 Section 3 Insights

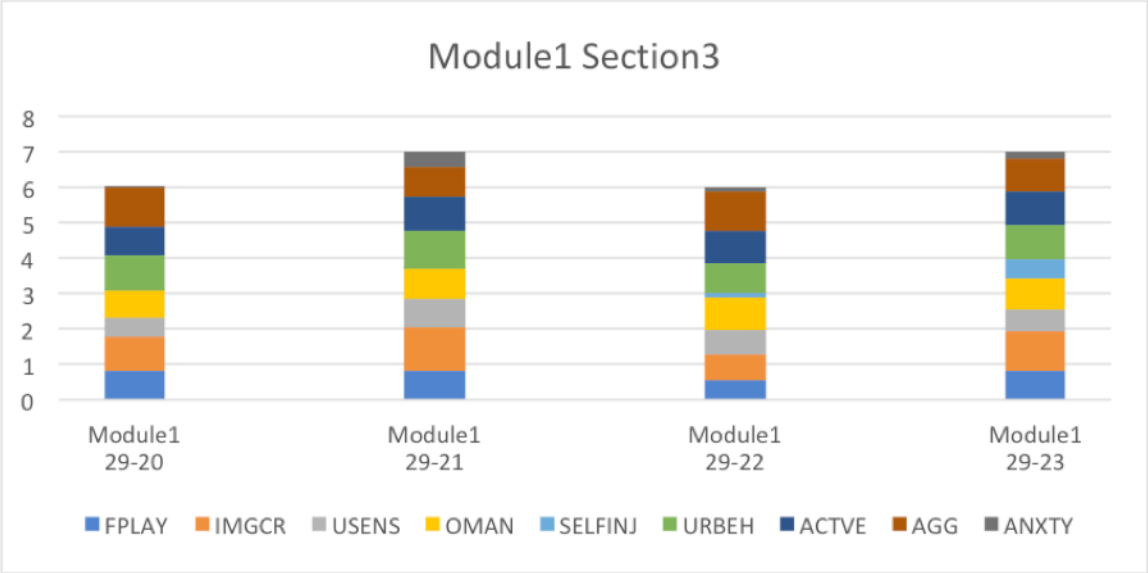


Figure 116. Module 1 Section 3 Insights

SELFINJ and ANXTY were consistently the least impactful architectures. The more significant the dimensionality reduction of an autoencoder architecture, the higher the extent to which these two variables dropped. In the lowest dimension architecture, these two variables almost wholly dropped.

AGG was the most impactful variable in the 20-variable reduced diagnostic and the 22-variable compressed diagnostic architectures. IMGCR was the most impactful variable in the other two architectures.

7.8 Insights Across Module 2 Architectures

7.8.1 Module 2 Section 1 Insights

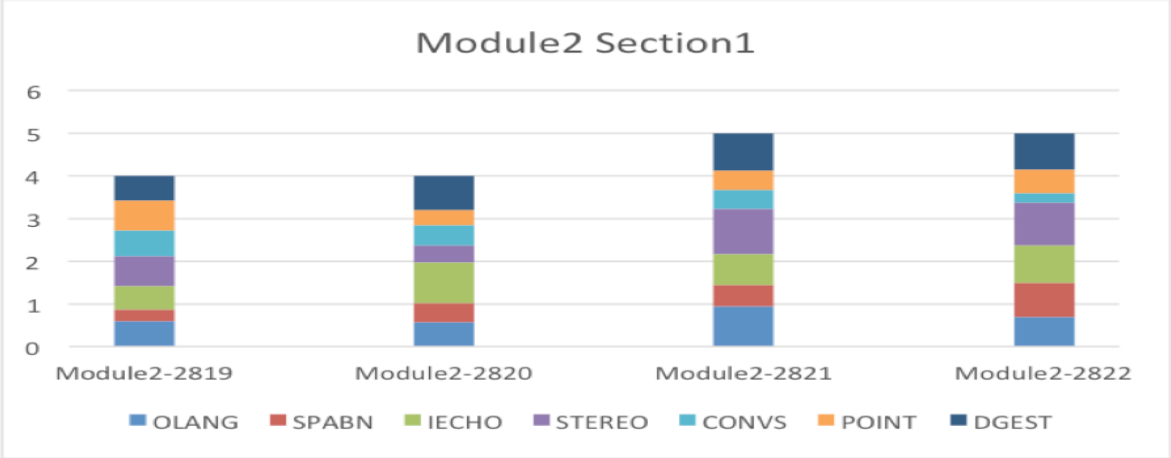


Figure 117. Module 2 Section 1 Insights

There were very few consistent trends between architectures in Module 2 Section 1. Different autoencoders found different combinations of weights to reduce dimensionality successfully.

7.8.2 Module 2 Section 2 Insights

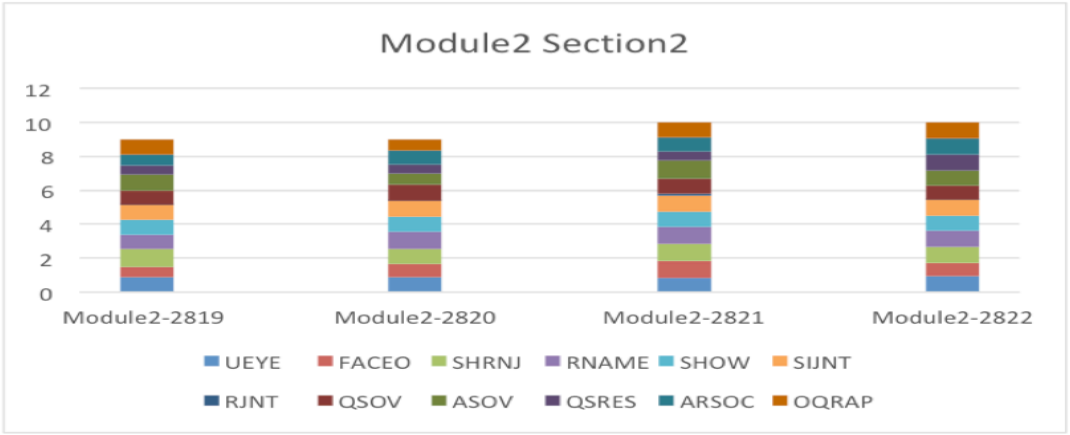


Figure 118. Module 2 Section 2 Insights

RJNT had the lowest absolute value weights across all autoencoder architectures.

7.8.3 Module 2 Section 3 Insights

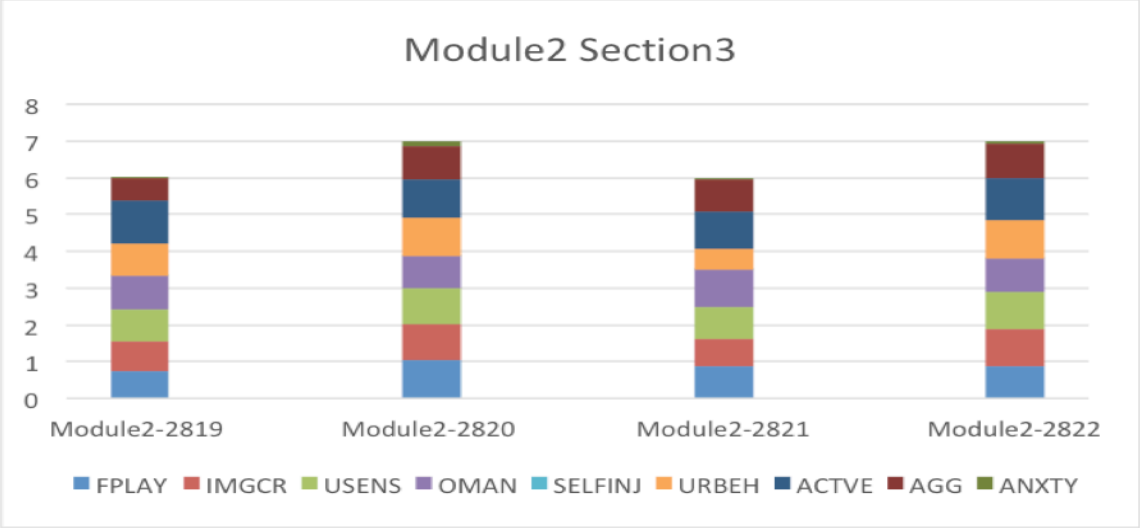


Figure 119. Module 2 Section 3 Insights

Once again, ANXTY and SELFINJ were the least represented variables in all architectures of autoencoder. ACTIVE had a strong weight presence throughout all reduced diagnostics.

7.9 Insights Across Module 3 Architectures

7.9.1 Module 3 Section 1 Insights

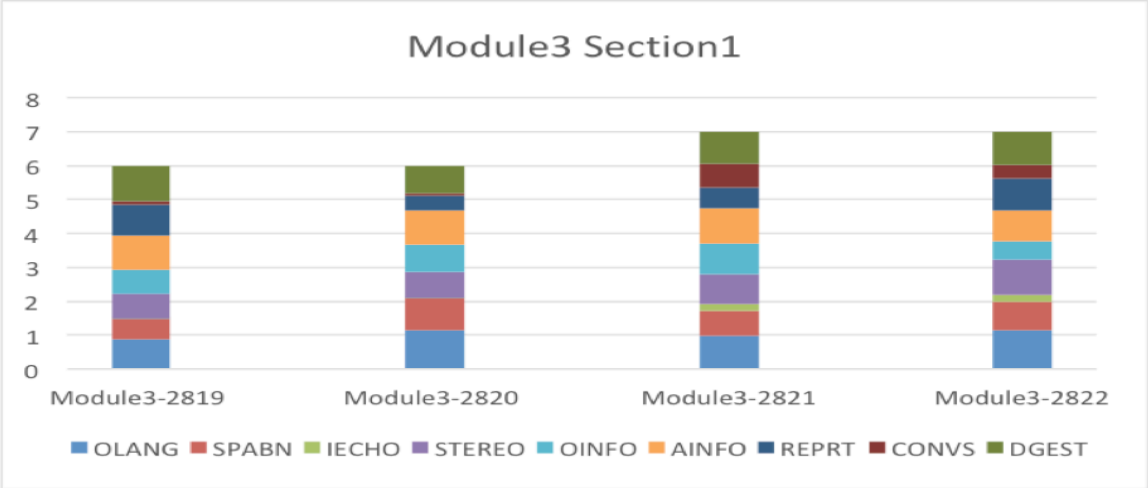


Figure 120. Module 3 Section 1 Insights

IECHO had the lowest absolute value weights in all reduced diagnostics. The absolute value of its weights was significantly smaller in the more moderate dimension reduced diagnostics (19 and 20 variables) than in the larger ones (21 and 22 variables). CONVS had consistently low absolute value weights and was the second lowest in 3 out of 4 autoencoders.

7.8.2 Module 3 Section 2 Insights

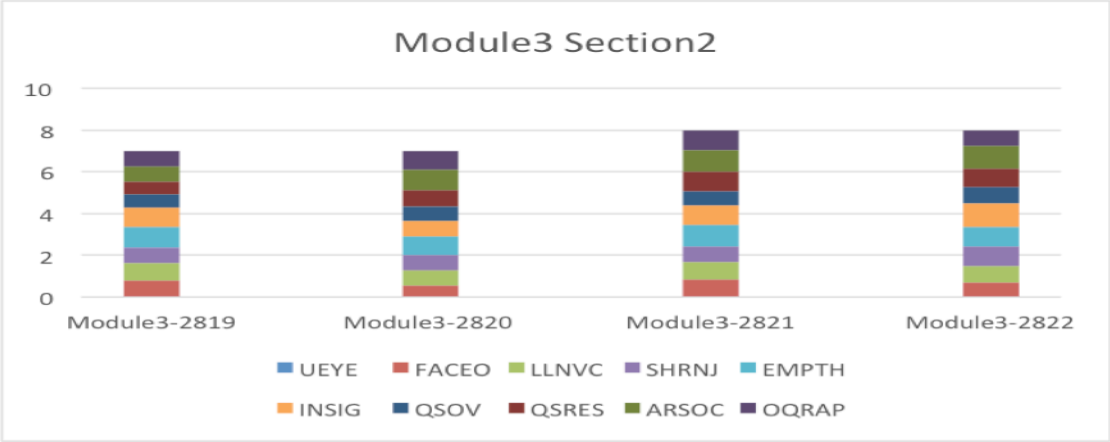


Figure 121. Module 3 Section 2 Insights

UEYE had the lowest absolute value weights across the board in all reduced diagnostics.

7.8.3 Module 3 Section 3 Insights

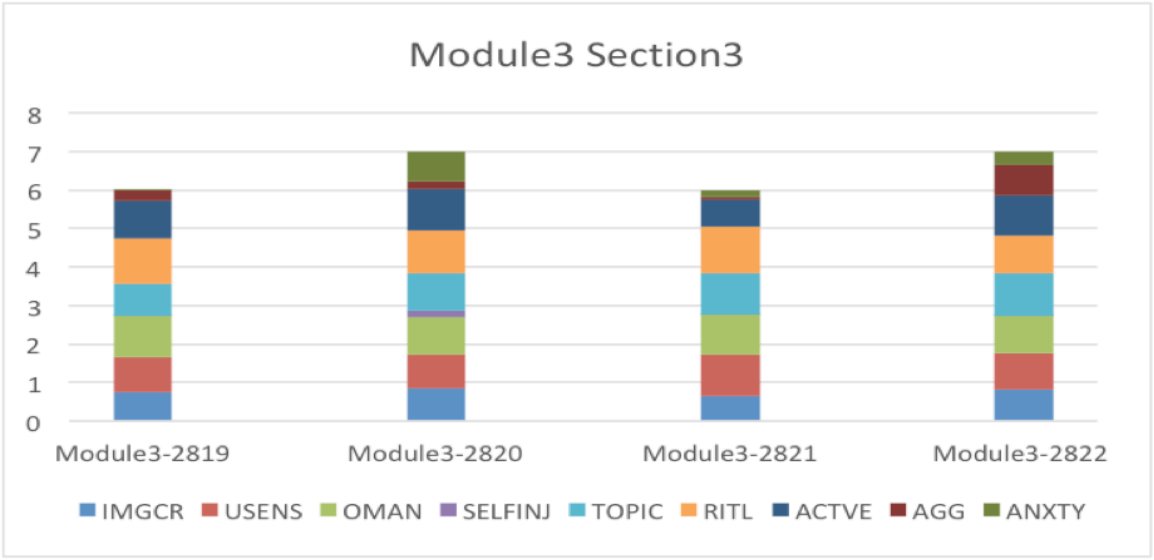


Figure 122. Module 3 Section 3 Insights

SELFINJ once again had the lowest absolute value weights in all architectures.

7.10 Insights Across Module 4 Architectures

7.10.1 Module 4 Section 1 Insights

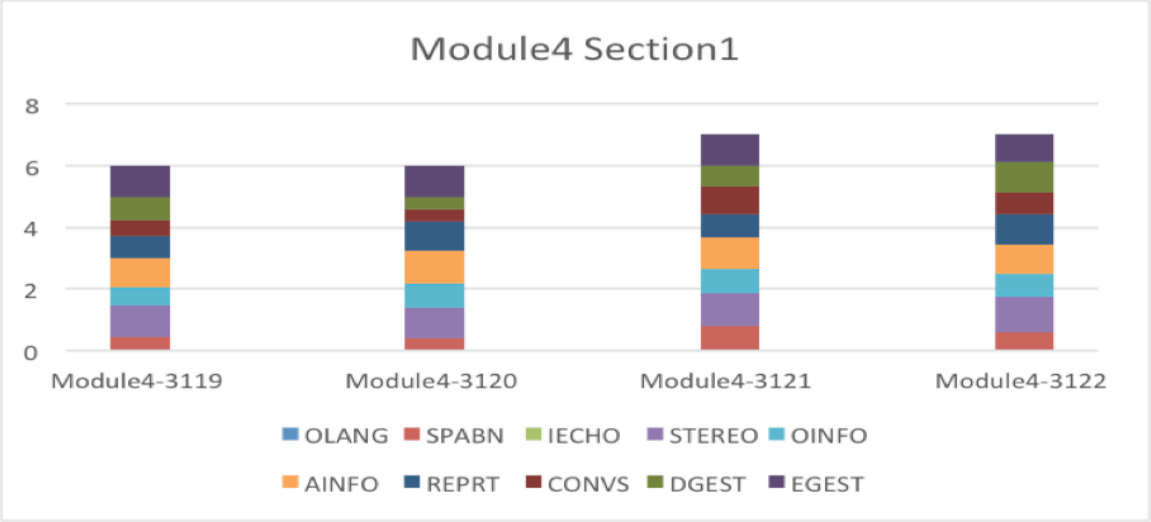


Figure 123. Module 4 Section 1 Insights

OLANG had the lowest absolute value weights in all architectures and negligible absolute value weights in all structures. IECHO had the second most moderate absolute value weights and was also insignificant in all constructions. STEREO had a strong presence in all architectures and had the most substantial absolute value weights in 3 out of 4 reduced diagnostics.

7.10.2 Module 4 Section 2 Insights

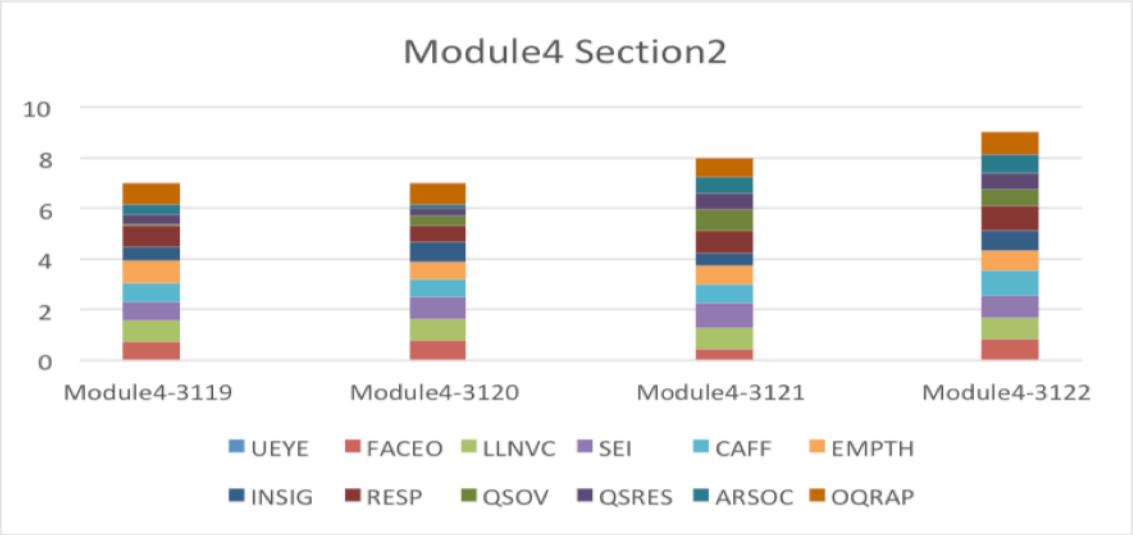


Figure 124. Module 4 Section 2 Insights

UEYE had the lowest absolute value weights in all architectures and negligibly represented.

7.10.3 Module 4 Section 3 Insights

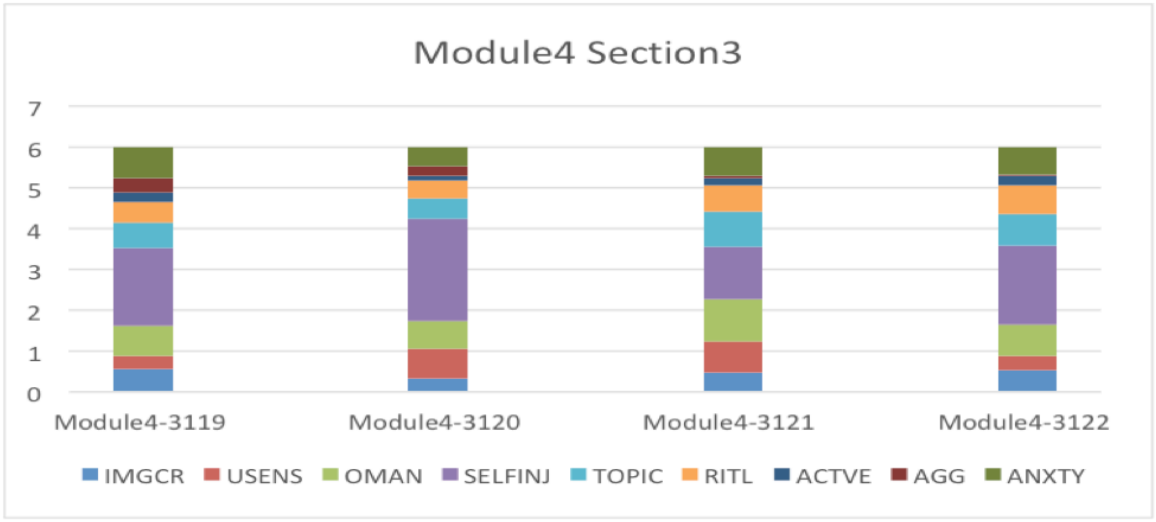


Figure 125. Module 4 Section 3 Insights

SELFINJ had by far the highest absolute value weights for all architectures in this section.

7.11 Insights Across Modules

7.11.1 Modules 1, 2 and 3

In the Modules 1, 2, and 3, SELFINJ was the lowest ranked variable across the board in its section. Its absolute value weights were negligible, and it effectively eliminated from most reduced diagnostics. In Module 4 however, it had the highest presence by a massive amount.

A study published after the date of most patient data used for this project indicated that self-injury is not a symptom of autism [44]. The fact that autoencoders repeatedly converged on solutions that eliminated self-injury might not be a coincidence. For that reason, it might be worth examining if self-injury is a reliable indicator of autism for adolescents and adults capable of having a functional conversation with an interviewer. The nature of Modules 1-3 differs significantly from Modules 4, and it is possible that self-injury's relationship with autism is somewhat nuanced, in that it is not a predictor for autism in earlier stages of social development, but is a definite symptom for autism in later stages of social development.

7.11.2 Modules 1 and 2

ANXTY had low absolute value weights consistently in Modules 1 and 2.

7.11.3 Modules 3 and 4

IECHO had low absolute values and efficiently eliminated from diagnostics in both Modules 3 and 4. UEYE also had consistently low absolute value weights.

7.12 Relative Significant of ADOS Observation Items

7.12.1 Normalization with Feature-Scaling

To provide additional context for comparing the relative strengths of weights, weight amounts were also feature scaled to fit a range between 0 and 1. The formula for feature scaling for a given section of weights looks like this.

$$\frac{X_i - X_{min}}{X_{max} - X_{min}}$$

The smallest absolute value weight for a section is set to 0 under this formula, and the largest is set to 1. All others are placed on a relative scale between 0 and 1. Normalized plots of weight values allows for one to conveniently observe the degree of separation between the strengths of different weights.

7.12.2 Module 1

The table below shows relative significant of ADOS Observation Items for Module 1 Section 1.

Table 24. Module 1 Section 1 Relative Significant

Module 1 (29-20) Section 1		
Observation Items	**Weights**	**Normalized Weights**
UOTHR	1.205708516	1
OLANG	0.904170549	0.69581462
IECHO	0.781549337	0.572116831
STEREO	0.708023612	0.497945572
INTON	0.453141831	0.240826005
POINT	0.450138779	0.237796587
FVOC	0.282855572	0.069044684
GEST	0.214411804	0

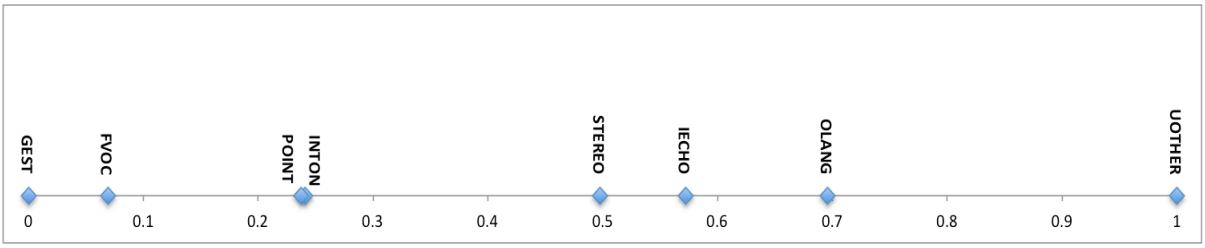


Figure 126. One-dimensional Scatter Plot - Module 1 Section 1

The table below shows relative significant of ADOS Observation Items for Module 1 Section 2.

Table 25. Module 1 Section 2 Relative Significant

Module 1 (29-20) Section 2		
Observation Items	**Weights**	**Normalized Weights**
GZSOV	0.986883532	1
SSMLE	0.946474784	0.933358854
SHRNJ	0.945726214	0.93212433
RJNT	0.8943154	0.847338838
REQ	0.87838718	0.821070396
RNAME	0.868602947	0.804934472
GIVE	0.833063511	0.746323679
UEYE	0.76341717	0.63146459

SHOW	0.645348437	0.436748443
QSOV	0.458394155	0.128427886
FACEO	0.398866478	0.030256257
SIJNT	0.380520193	0

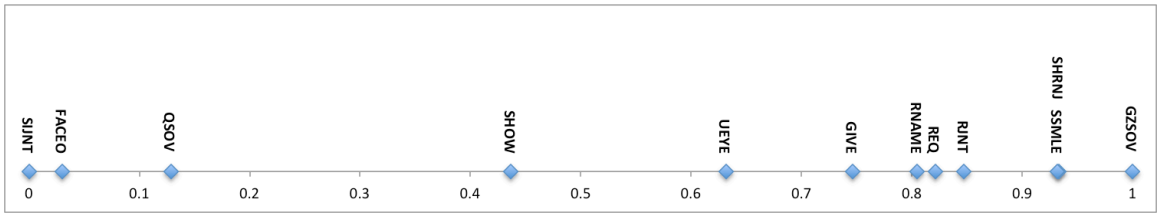


Figure 127. One-dimensional Scatter Plot - Module 1 Section 2

The table below shows relative significant of ADOS Observation Items for Module 1 Section 3.

Table 26. Module 1 Section 3 Relative Significant

Module 1 (29-20) Section 3		
Observation Items	**Weights**	**Normalized Weights**
AGG	1.110117879	1
URBEH	0.974269508	0.877620065
IMGCR	0.948730169	0.854612777
FPLAY	0.830749743	0.748329303
ACTVE	0.824988178	0.743138958
OMAN	0.781285844	0.703769411
USENS	0.529718704	0.477143431
ANXTY	7.63822E-05	1.15212E-05
SELFINJ	6.3593E-05	0

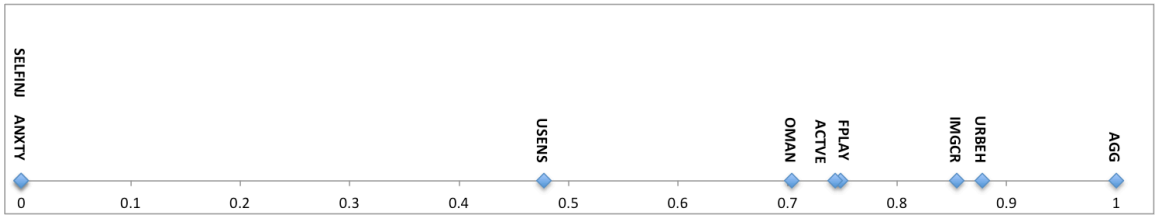


Figure 128. One-dimensional Scatter Plot - Module 1 Section 3

The table below shows relative significant of ADOS Observation Items for Module 1.

Table 27. Module 1 Relative Significant

Module 1 (29-20)	
Observation Items	**Normalized Weights**
UOTHR	1
GZSOV	1
AGG	1
SSMLE	0.933358854
SHRNJ	0.93212433
URBEH	0.877620065
IMGCR	0.854612777
RJNT	0.847338838
REQ	0.821070396
RNAME	0.804934472
FPLAY	0.748329303
GIVE	0.746323679
ACTVE	0.743138958
OMAN	0.703769411
OLANG	0.69581462
UEYE	0.63146459
IECHO	0.572116831
STEREO	0.497945572
USENS	0.477143431
SHOW	0.436748443
INTON	0.240826005
POINT	0.237796587
QSOV	0.128427886
FVOC	0.069044684
FACEO	0.030256257
ANXTY	1.15212E-05
GEST	0
SIJNT	0
SELFINJ	0

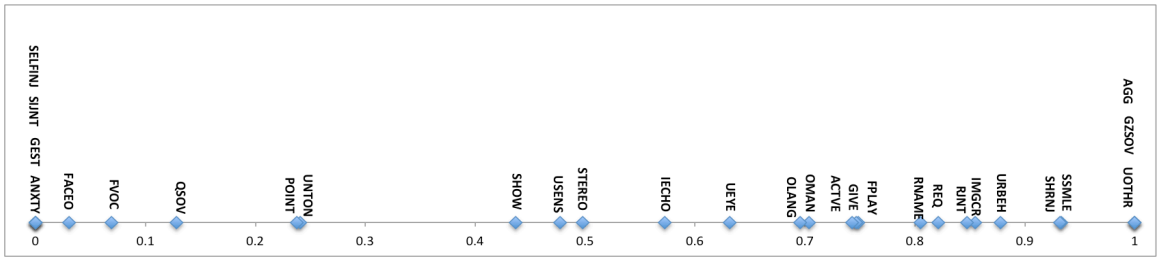


Figure 129. One-dimensional Scatter Plot - Module 1

7.12.3 Module 2

The table below shows relative significant of ADOS Observation Items for Module 2 Section 1.

Table 28. Module 2 Section 1 Relative Significant

Module 2 (28-19) Section 1		
Observation Items	**Weights**	**Normalized Weights**
POINT	0.708659958	1
STEREO	0.700268554	0.980743087
CONVS	0.597751086	0.745482102
OLANG	0.595364234	0.740004664
DGEST	0.571544413	0.685342033
IECHO	0.553512397	0.643961476
SPABN	0.272899357	0

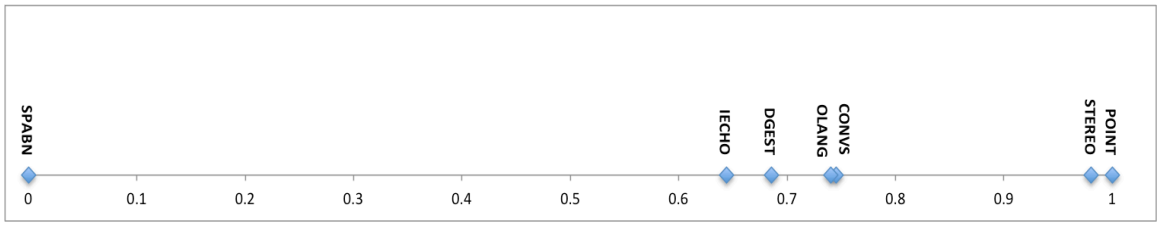


Figure 130. One-dimensional Scatter Plot - Module 2 Section 1

The table below shows relative significant of ADOS Observation Items for Module 2 Section 2.

Table 29. Module 2 Section 2 Relative Significant

Module 2 (28-19) Section 2		
Observation Items	**Weights**	**Normalized Weights**
SHRNJ	1.010939116	1
ASOV	0.921786728	0.911798545
OQRAP	0.901903429	0.892127329
SIJNT	0.898558484	0.888818063
SHOW	0.884995778	0.875400022
UEYE	0.877366224	0.867851847
QSOV	0.846978244	0.837787997
RNAME	0.82933398	0.820331934
ARSOC	0.675532079	0.668170542
FACEO	0.629712293	0.622839488
QSRES	0.522735909	0.517004155
RJNT	0.000157736	0

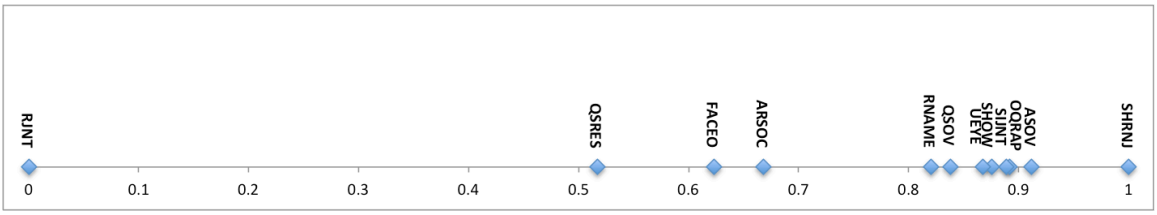


Figure 131. One-dimensional Scatter Plot - Module 2 Section 2

The table below shows relative significant of ADOS Observation Items for Module 2 Section 3.

Table 30. Module 2 Section 3 Relative Significant

Module 2 (28-19) Section 3		
Observation Items	**Weights**	**Normalized Weights**
ACTVE	1.175469252	1

OMAN	0.914500552	0.777978982
URBEH	0.883581232	0.751674146
USENS	0.86840694	0.738764506
IMGCR	0.797130439	0.678125505
FPLAY	0.756933467	0.643927639
AGG	0.603862908	0.513701749
SELFINJ	6.93597E-05	1.99992E-05
ANXTY	4.58522E-05	0

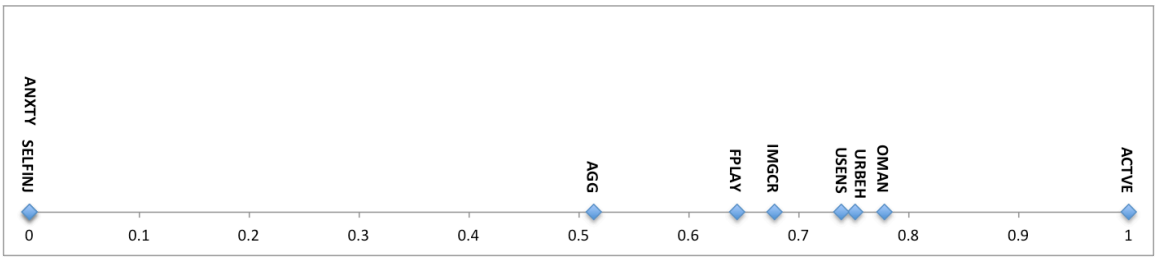


Figure 132. One-dimensional Scatter Plot - Module 2 Section 3

The table below shows relative significant of ADOS Observation Items for Module 2.

Table 31. Module 2 Relative Significant

Module 2 (28-19)	
Observation Items	**Normalized Weights**
POINT	1
SHRNJ	1
ACTVE	1
STEREO	0.980743087
ASOV	0.911798545
OQRAP	0.892127329
SIJNT	0.888818063
SHOW	0.875400022
UEYE	0.867851847
QSOV	0.837787997
RNAME	0.820331934
OMAN	0.777978982
URBEH	0.751674146

CONVS	0.745482102
OLANG	0.740004664
USENS	0.738764506
DGEST	0.685342033
IMGCR	0.678125505
ARSOC	0.668170542
IECHO	0.643961476
FPLAY	0.643927639
FACEO	0.622839488
QSRES	0.517004155
AGG	0.513701749
SELFINJ	1.99992E-05
SPABN	0
RJNT	0
ANXTY	0

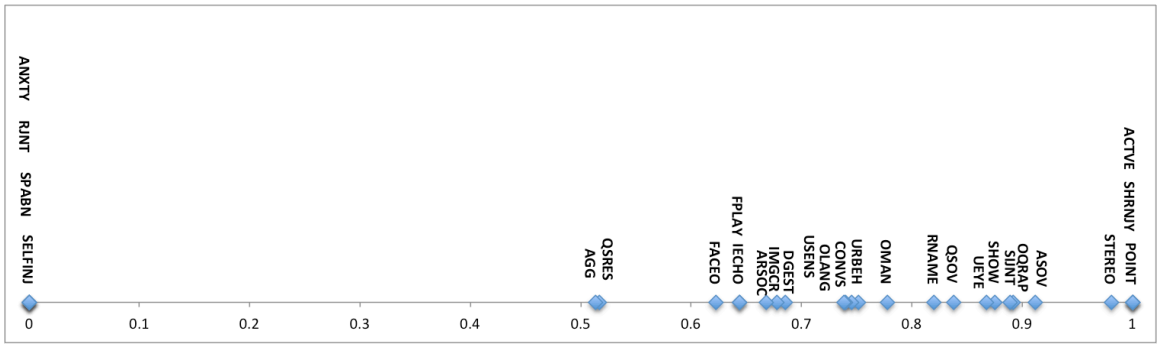


Figure 133. One-dimensional Scatter Plot - Module 2

7.12.3 Module 3

The table below shows relative significant of ADOS Observation Items for Module 3 Section 1.

Table 32. Module 3 Section 1 Relative Significant

Module 3 (28-19) Section 1		
Observation Items	**Weights**	**Normalized Weights**
DGEST	1.057394016	1
AINFO	0.991035334	0.937240375
REPRT	0.92237092	0.872300072
OLANG	0.891673245	0.843267329
STEREO	0.723490444	0.684206159
OINFO	0.715556231	0.676702269
SPABN	0.599730539	0.567158547
CONVS	0.098701942	0.093303942
IECHO	4.73278E-05	0

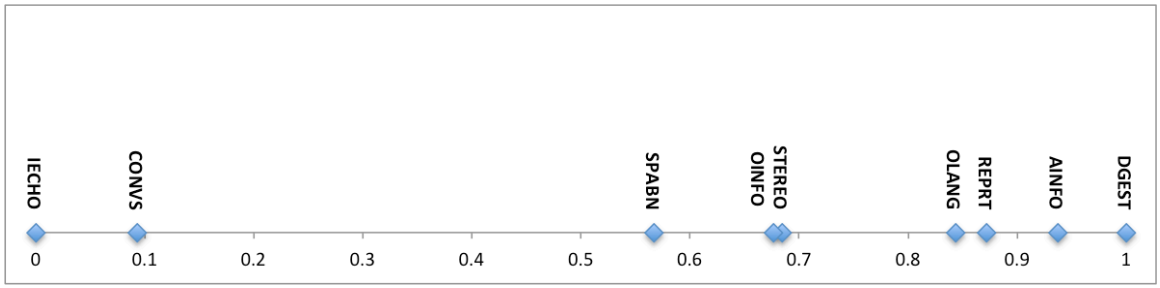


Figure 134. One-dimensional Scatter Plot - Module 3 Section 1

The table below shows relative significant of ADOS Observation Items for Module 3 Section 2.

Table 33. Module 3 Section 2 Relative Significant

Module 3 (28-19) Section 2		
Observation Items	**Weights**	**Normalized Weights**
EMPTH	0.957367727	1
INSIG	0.936019453	0.977697431
LLNVC	0.823743481	0.860402581
FACEO	0.78472377	0.81963864
SHRNJ	0.781059777	0.815810862
ARSOC	0.761125309	0.794985298
OQRAP	0.732458418	0.765036961
QSOV	0.655085127	0.684204986

QSRES	0.568260651	0.593499346
UEYE	0.000156287	0

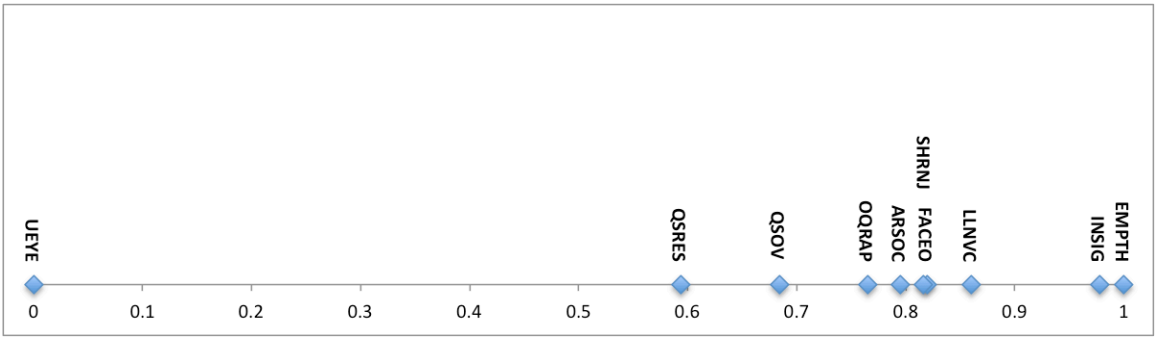


Figure 135. One-dimensional Scatter Plot - Module 3 Section 2

The table below shows relative significant of ADOS Observation Items for Module 3 Section 3.

Table 34. Module 3 Section 3 Relative Significant

Module 3 (28-19) Section 3		
Observation Items	**Weights**	**Normalized Weights**
RITL	1.170622428	1
OMAN	1.078962543	0.921694967
ACTVE	0.971970664	0.830291812
USENS	0.896979931	0.766227234
TOPIC	0.839487573	0.717111517
IMGCR	0.742854752	0.634558105
AGG	0.270061152	0.230650566
ANXTY	0.028987615	0.024701461
SELFINJ	7.3343E-05	0

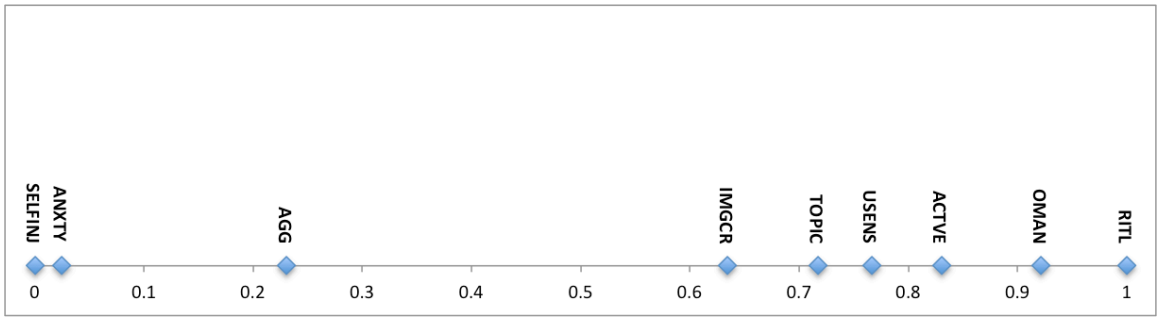


Figure 136. One-dimensional Scatter Plot - Module 3 Section 3

The table below shows relative significant of ADOS Observation Items for Module 3.

Table 35. Module 3 Relative Significant

Module 3 (28-19)	
Observation Items	**Normalized Weights**
DGEST	1
EMPTH	1
RITL	1
INSIG	0.977697431
AINFO	0.937240375
OMAN	0.921694967
REPRT	0.872300072
LLNVC	0.860402581
OLANG	0.843267329
ACTVE	0.830291812
FACEO	0.81963864
SHRNJ	0.815810862
ARSOC	0.794985298
USENS	0.766227234
OQRAP	0.765036961
TOPIC	0.717111517
STEREO	0.684206159
QSOV	0.684204986
OINFO	0.676702269
IMGCR	0.634558105

QSRES	0.593499346
SPABN	0.567158547
AGG	0.230650566
CONVS	0.093303942
ANXTY	0.024701461
IECHO	0
UEYE	0
SELFINJ	0

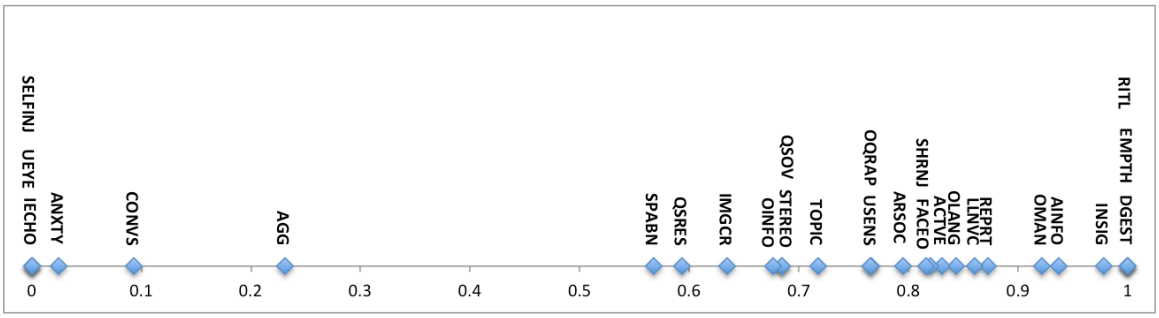


Figure 137. One-dimensional Scatter Plot - Module 3

7.12.3 Module 4

The table below shows relative significant of ADOS Observation Items for Module 4 Section 1.

Table 36. Module 4 Section 1 Relative Significant

Module 4 (31-19) Section 1		
Observation Items	**Weights**	**Normalized Weights**
STEREO	1.041436433	1
EGEST	1.026833667	0.985977423
AINFO	0.97720692	0.938322419
DGEST	0.769717302	0.739076667
REPRT	0.681088622	0.653969334
OINFO	0.567627342	0.545016037
CONVS	0.510251157	0.489919491
SPABN	0.425571861	0.408604628
IECHO	0.00020563	0.000138819

OLANG	6.10677E-05	0

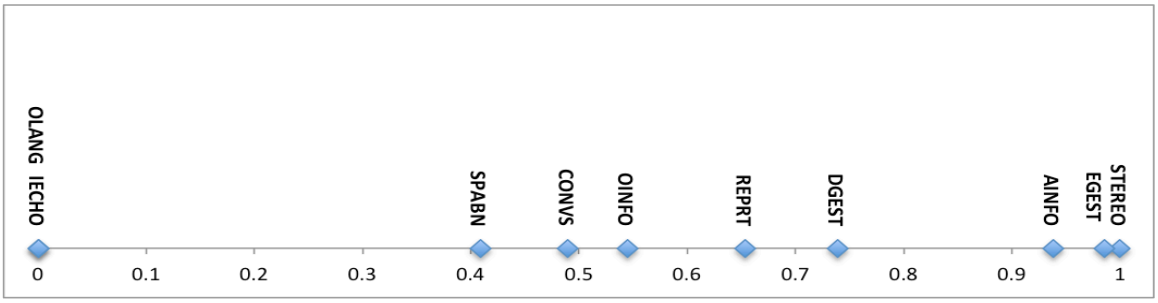


Figure 138. One-dimensional Scatter Plot - Module 4 Section 1

The table below shows relative significant of ADOS Observation Items for Module 4 Section 2.

Table 37. Module 4 Section 2 Relative Significant

Module 4 (31-19) Section 2		
Observation Items	**Weights**	**Normalized Weights**
EMPTH	0.854731274	1
RESP	0.84517449	0.988804041
OQRAP	0.832105734	0.973493738
LLNVC	0.813874191	0.95213513
CAFF	0.760248374	0.889311438
SEI	0.756331895	0.884723206
FACEO	0.740745094	0.866462965
INSIG	0.55350398	0.647106344
ARSOC	0.39763268	0.464500072
QSRES	0.377828497	0.441299085
QSOV	0.066684778	0.076788143
UEYE	0.001139013	0

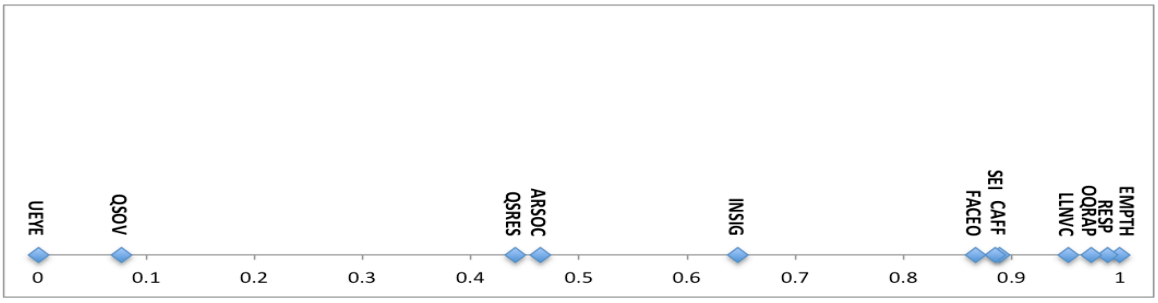


Figure 139. One-dimensional Scatter Plot - Module 4 Section 2

The table below shows relative significant of ADOS Observation Items for Module 4 Section 3.

Table 38. Module 4 Section 3 Relative Significant

Module 4 (31-19) Section 3		
Observation Items	**Weights**	**Normalized Weights**
SELFINJ	1.889942607	1
ANXTY	0.765446827	0.318667147
OMAN	0.754011647	0.311738563
TOPIC	0.63886953	0.241973866
IMGCR	0.560970038	0.194774505
RITL	0.488984408	0.151158356
AGG	0.347731121	0.065572882
USENS	0.314536498	0.045460234
ACTVE	0.239507323	0

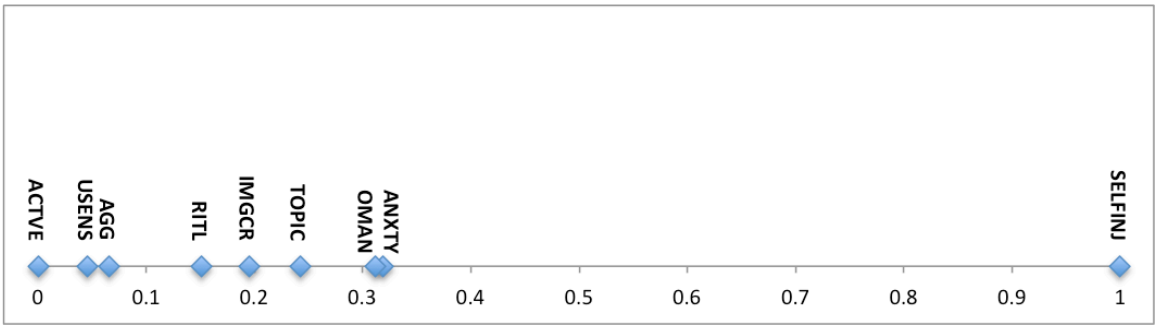


Figure 140. One-dimensional Scatter Plot - Module 4 Section 3

The table below shows relative significant of ADOS Observation Items for Module 4.

Table 39. Module 4 Relative Significant

Module 4 (31-19)	
Observation Items	**Normalized Weights**
STEREO	1
EMPTH	1
SELFINJ	1
RESP	0.988804041
EGEST	0.985977423
OQRAP	0.973493738
LLNVC	0.95213513
AINFO	0.938322419
CAFF	0.889311438
SEI	0.884723206
FACEO	0.866462965
DGEST	0.739076667
REPRT	0.653969334
INSIG	0.647106344
OINFO	0.545016037
CONVS	0.489919491
ARSOC	0.464500072
QSRES	0.441299085
SPABN	0.408604628
ANXTY	0.318667147
OMAN	0.311738563
TOPIC	0.241973866
IMGCR	0.194774505
RITL	0.151158356
QSOV	0.076788143
AGG	0.065572882
USENS	0.045460234
IECHO	0.000138819
OLANG	0
UEYE	0
ACTVE	0

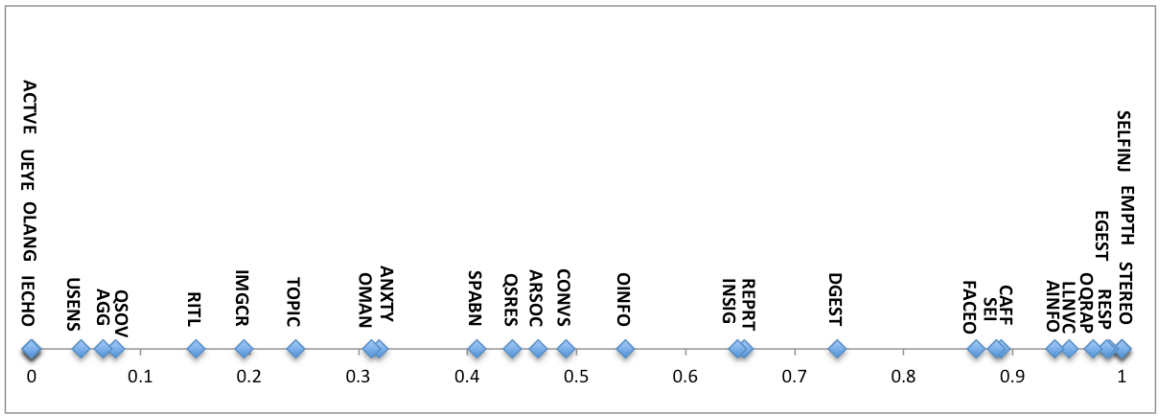


Figure 141. One-dimensional Scatter Plot - Module 4

CHAPTER EIGHT

CONCLUSION

8.1 Conclusion

Following the training of each iteration of all autoencoder architectures on all ADOS modules, a number of clear insights were provided by examining the absolute value of each variable's weights for the encoder section of each model. The effectiveness of all autoencoders was validated by their accuracy at reconstructing their input. As expected, the accuracy of a reconstruction directly correlated with the dimensionality of a hidden layer. By comparing the cumulative absolute value weights of variables, it was possible to rank the extent to which each variable impacted its relevant section in the reduced diagnostic. "Self-Injurious Behavior" was consistently eliminated in all autoencoder architectures in Modules 1-3, which suggests that a reduced diagnostic could avoid including observations for this behavior in Modules 1-3 (which differ greatly from Module 4 in their administration). This trend starkly reversed in Module 4, which relevant autoencoders consistently treated "Self-Injurious Behavior" as the highest priority when reducing the diagnostic. "Anxiety" was consistently deprioritized in Module 1 and Module 2, suggesting that it might be not be necessary to monitor for this symptom when evaluating examinees with less developed language fluency and communication abilities. "Immediate echolalia" was eliminated from Module 3 and

Module 4, suggesting that this symptom might not be necessary to include in examinations of individuals with more advanced language fluency and communication abilities.

8.2 Discussion of Context

This thesis attempted to use the weights of autoencoders to assess which variables in the Autism Diagnostic Observation Schedule would be integral to a shorter version of the diagnostic. Analyzing the weights of these autoencoders did allow for insights to be drawn regarding the extent to which variables represented in the reduced diagnostic. As such, the variables the autoencoders chose to prioritize can be treated as suggestions for focal points of a reduced diagnostic.

The fact that the autoencoders were able to reconstruct the diagnostic with a reasonable degree of accuracy suggests that machine learning approaches might be valuable in using the smaller amount of information in a shorter diagnostic to create a more extensive, more comprehensive patient profile. With sufficiently advanced machine learning approaches trained on enough large datasets, it might be possible that many types of shorter observation schedules could be used to create more significant patient profiles. This experiment suggests that machine learning approaches can enable the possibility of shorter autism diagnostics through their inferential capabilities.

It is worth discussing if the concepts that autoencoders' chose represent the foundation of a functional reduced diagnostic. There is, unfortunately, no way to currently confirming this. The possibility is worth considering though. The fact that the autoencoder reliably removed "Self-Injurious Behavior" from the diagnostic for the first three modules suggests that this might be a possibility. According to study, self-injury is no longer considered a symptom of autism[44]. The fact that every autoencoder trained on the first three modules repeatedly and reliably converged upon the solution of eliminating self-injury from the diagnostic suggests that it can isolate the variables that are most important to the diagnostic from the perspective of autism psychology.

Autoencoders might be valuable as tools to identify areas of further research within fields of psychology. If autoencoders consistently remove the same variable from a data set, it might be worth examining if that variable represents a concept that is related to the domain. In the case of these autoencoders, self-injury was repeatedly removed, and it turns out that there is experimental evidence to suggest that self-injury is not a symptom of autism. Similarly, if autoencoders regularly prioritize specific variables, that might indicate that those variables are a more integral aspect of autism than realized. Alone, these trends are entirely insufficient evidence, which is why further psychology research and experimentation would be needed.

In the big picture, psychological data contains a degree of inherent unreliability. Observation diagnoses are somewhat unreliable by nature, given

the degree of subjectivity included within them. This, unfortunately, limits the extent to which machine learning algorithms applied to psychology data can yield valuable insights about the underlying domain.

Nonetheless, this thesis indicates that machine learning algorithms applied to psychology data can potentially uncover trends that accurately portray facts about human psychology. As more powerful models are applied to more massive data sets, it is possible that machine learning algorithms can act as a bulwark against the unreliability of psychology data by identifying consistent and essential patterns that separate useful variables from misleading variables. By reducing patient profiles to critical components, autoencoders certainly have potential in this regard, especially when it comes to the domain of the Autism Diagnostic Observation Schedule.

8.3 Future Work

It would be valuable to test if autoencoders trained on a different and larger samples of ADOS data converged upon the same solutions by prioritizing and deprioritizing the same variables. Furthermore, researchers in the field of psychology might evaluate if a shorter diagnostic process can be created by designing a series of observation activities that focus on the concepts prioritized by the autoencoders in this thesis.

The fact that self-injury was, in contrast to Modules 1-3, heavily prioritized in Module 4 suggests that further research into the relationship between self-harm tendencies and adult autism might be valuable. Since the autoencoder correctly removed self-injury in Modules 1-3, a degree of credibility lent to its suggestion regarding self-injury in Module 4.

Similarly, the autoencoder also strongly highlights “Anxiety” as a target for removal in Module 1 and Module 2, and highlights “Immediate Echolalia” as a target for removal from Module 3 and Module 4. This represents another potential research area for psychologists, as it can be examined if anxiety is really a symptom of autism for individuals with underdeveloped communication abilities, and if immediate echolalia is really a symptom of autism for individuals with more developed communication abilities.

APPENDIX A

ADOS DATA APPROVAL

From: mgillesp@csusb.edu
Subject: IRB-FY2017-145 - Initial: IRB Expedited Review Approval Letter
Date: May 16, 2017 at 11:00 AM
To: 005070711@coyote.csusb.edu, kvoigt@csusb.edu

May 16, 2017

CSUSB INSTITUTIONAL REVIEW BOARD
Expedited Review
IRB# FY2017-145
Status: Approved

Ms. Sara Daghustani and Prof. Kerstin Voigt
School of Computer Science
California State University, San Bernardino
5500 University Parkway
San Bernardino, California 92407

Dear Ms. Daghustani and Prof. Voigt:

Your application to use human subjects, titled, "FINDING PREDICTIVE VALUE USING MACHINE LEARNING FOR THE AUTISM DIAGNOSTIC OBSERVATION SCHEDULE," has been reviewed and approved by the Institutional Review Board (IRB). The informed consent document you submitted is the official version for your study and cannot be changed without prior IRB approval. A change in your informed consent (no matter how minor the change) requires resubmission of your protocol as amended using the IRB Cayuse system protocol change form. **Your application is approved for one year from May 16, 2017 through May 15, 2018. Please note the Cayuse IRB system will notify you when your protocol is up for renewal and ensure you file it before your protocol study end date.**

Your responsibilities as the researcher/investigator reporting to the IRB Committee include the following 4 requirements as mandated by the Code of Federal Regulations 45 CFR 46 listed below. Please note that the protocol change form and renewal form are located on the IRB website under the forms menu. Failure to notify the IRB of the above may result in disciplinary action. You are required to keep copies of the informed consent forms and data for at least three years. Please notify the IRB Research Compliance Officer for any of the following:

1) Submit a protocol change form if any changes (no matter how minor) are proposed in your research protocol for review and approval of the IRB before implemented in your research,
2) If any unanticipated/adverse events are experienced by subjects during your research,
3) To apply for renewal and continuing review of your protocol one month prior to the protocols end date,
4) When your project has ended by emailing the IRB Research Compliance Officer.

The CSUSB IRB has not evaluated your proposal for scientific merit, except to weigh the risk to the human participants and the aspects of the proposal related to potential risk and benefit. This approval notice does not replace any departmental or additional approvals which may be required. If you have any questions regarding the IRB decision, please contact Michael Gillespie, the IRB Compliance Officer. Mr. Michael Gillespie can be reached by phone at (909) 537-7588, by fax at (909) 537-7028, or by email at mgillesp@csusb.edu. Please include your application approval identification number (listed at the top) in all correspondence.

Best of luck with your research.

Sincerely,

Caroline Vickers

Caroline Vickers, Ph.D., IRB Chair
CSUSB Institutional Review Board

CV/MG

APPENDIX B

HADAMARD PRODUCT

Hadamard Product of Weight Matrix and Binary Mask Matrix for Module 1 (29 to 21)

$$
\begin{bmatrix}
w_{1,1} & \dots & w_{1,5} & w_{1,6} & \dots & w_{1,14} & w_{1,15} & \dots & w_{1,21} \\
\vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\
w_{8,1} & \dots & w_{8,5} & w_{8,6} & \dots & w_{8,14} & w_{8,15} & \dots & w_{8,21} \\
w_{9,1} & \dots & w_{9,5} & w_{9,6} & \dots & w_{9,14} & w_{9,15} & \dots & w_{9,21} \\
\vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\
w_{20,1} & \dots & w_{20,5} & w_{20,6} & \dots & w_{20,14} & w_{20,15} & \dots & w_{20,21} \\
w_{21,1} & \dots & w_{21,5} & w_{21,6} & \dots & w_{21,14} & w_{21,15} & \dots & w_{21,21} \\
\vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\
w_{29,1} & \dots & w_{29,5} & w_{29,6} & \dots & w_{29,14} & w_{29,15} & \dots & w_{29,21}
\end{bmatrix}
\odot
\begin{bmatrix}
1_{1,1} & \dots & 1_{1,5} & 0_{1,6} & \dots & 0_{1,14} & 0_{1,15} & \dots & 0_{1,21} \\
\vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\
1_{8,1} & \dots & 1_{8,5} & 0_{8,6} & \dots & 0_{8,14} & 0_{8,15} & \dots & 0_{8,21} \\
0_{9,1} & \dots & 0_{9,5} & 1_{9,6} & \dots & 1_{9,14} & 0_{9,15} & \dots & 0_{9,21} \\
\vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\
0_{20,1} & \dots & 0_{20,5} & 1_{20,6} & \dots & 1_{20,14} & 0_{20,15} & \dots & 0_{20,21} \\
0_{21,1} & \dots & 0_{21,5} & 0_{21,6} & \dots & 0_{21,14} & 1_{21,15} & \dots & 1_{21,21} \\
\vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\
0_{29,1} & \dots & 0_{29,5} & 0_{29,6} & \dots & 0_{29,14} & 1_{29,15} & \dots & 1_{29,21}
\end{bmatrix}
$$

$$
=
\begin{bmatrix}
w_{1,1} & \dots & w_{1,5} & 0_{1,6} & \dots & 0_{1,14} & 0_{1,15} & \dots & 0_{1,21} \\
\vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\
w_{8,1} & \dots & w_{8,5} & 0_{8,6} & \dots & 0_{8,14} & 0_{8,15} & \dots & 0_{8,21} \\
0_{9,1} & \dots & 0_{9,5} & w_{9,6} & \dots & w_{9,14} & 0_{9,15} & \dots & 0_{9,21} \\
\vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\
0_{20,1} & \dots & 0_{20,5} & w_{20,6} & \dots & w_{20,14} & 0_{20,15} & \dots & 0_{20,21} \\
0_{21,1} & \dots & 0_{21,5} & 0_{21,6} & \dots & 0_{21,14} & w_{21,15} & \dots & w_{21,21} \\
\vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\
0_{29,1} & \dots & 0_{29,5} & 0_{29,6} & \dots & 0_{29,14} & w_{29,15} & \dots & w_{29,21}
\end{bmatrix}
$$

Hadamard Product of Weight Matrix and Binary Mask Matrix for Module 1 (29 to 22)

$$
\begin{bmatrix}
w_{1,1} & \cdots & w_{1,6} & \cdots & w_{1,14} & w_{1,17} & \cdots & w_{1,22} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{8,1} & \cdots & w_{8,6} & \cdots & w_{8,14} & w_{8,17} & \cdots & w_{8,22} \\
w_{9,1} & \cdots & w_{9,6} & \cdots & w_{9,14} & w_{9,17} & \cdots & w_{9,22} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{20,1} & \cdots & w_{20,6} & \cdots & w_{20,14} & w_{20,17} & \cdots & w_{20,22} \\
w_{21,1} & \cdots & w_{21,6} & \cdots & w_{21,14} & w_{21,17} & \cdots & w_{21,22} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{29,1} & \cdots & w_{29,6} & \cdots & w_{29,14} & w_{29,17} & \cdots & w_{29,22}
\end{bmatrix}
\odot
\begin{bmatrix}
1_{1,1} & \cdots & 1_{1,6} & 0_{1,7} & \cdots & 0_{1,16} & 0_{1,17} & \cdots & 0_{1,22} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
1_{8,1} & \cdots & 1_{8,6} & 0_{8,7} & \cdots & 0_{8,16} & 0_{8,17} & \cdots & 0_{8,22} \\
0_{9,1} & \cdots & 0_{9,6} & 1_{9,7} & \cdots & 1_{9,16} & 0_{9,17} & \cdots & 0_{9,22} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{20,1} & \cdots & 0_{20,6} & 1_{20,7} & \cdots & 1_{20,16} & 0_{20,17} & \cdots & 0_{20,22} \\
0_{21,1} & \cdots & 0_{21,6} & 0_{21,7} & \cdots & 0_{21,16} & 1_{21,17} & \cdots & 1_{21,22} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{29,1} & \cdots & 0_{29,6} & 0_{29,7} & \cdots & 0_{29,16} & 1_{29,17} & \cdots & 1_{29,22}
\end{bmatrix}
$$

$$
=
\begin{bmatrix}
w_{1,1} & \cdots & w_{1,6} & 0_{1,7} & \cdots & 0_{1,16} & 0_{1,17} & \cdots & 0_{1,22} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{8,1} & \cdots & w_{8,6} & 0_{8,7} & \cdots & 0_{8,16} & 0_{8,17} & \cdots & 0_{8,22} \\
0_{9,1} & \cdots & 0_{9,6} & w_{9,7} & \cdots & w_{9,16} & 0_{9,17} & \cdots & 0_{9,22} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{20,1} & \cdots & 0_{20,6} & w_{20,7} & \cdots & w_{20,16} & 0_{20,17} & \cdots & 0_{20,22} \\
0_{21,1} & \cdots & 0_{21,6} & 0_{21,7} & \cdots & 0_{21,16} & w_{21,17} & \cdots & w_{21,22} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{29,1} & \cdots & 0_{29,6} & 0_{29,7} & \cdots & 0_{29,16} & w_{29,17} & \cdots & w_{29,22}
\end{bmatrix}
$$

Hadamard Product of Weight Matrix and Binary Mask Matrix for Module 1 (29 to 23)

$$
\begin{bmatrix}
w_{1,1} & \dots & w_{1,6} & \dots & w_{1,14} & w_{1,17} & \dots & w_{1,23} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{8,1} & \dots & w_{8,6} & \dots & w_{8,14} & w_{8,17} & \dots & w_{8,23} \\
w_{9,1} & \dots & w_{9,6} & \dots & w_{9,14} & w_{9,17} & \dots & w_{9,23} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{20,1} & \dots & w_{20,6} & \dots & w_{20,14} & w_{20,17} & \dots & w_{20,23} \\
w_{21,1} & \dots & w_{21,6} & \dots & w_{21,14} & w_{21,17} & \dots & w_{21,23} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{29,1} & \dots & w_{29,6} & \dots & w_{29,14} & w_{29,17} & \dots & w_{29,23}
\end{bmatrix}
\odot
\begin{bmatrix}
1_{1,1} & \dots & 1_{1,6} & 0_{1,7} & \dots & 0_{1,16} & 0_{1,17} & \dots & 0_{1,23} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
1_{8,1} & \dots & 1_{8,6} & 0_{8,7} & \dots & 0_{8,16} & 0_{8,17} & \dots & 0_{8,23} \\
0_{9,1} & \dots & 0_{9,6} & 1_{9,7} & \dots & 1_{9,16} & 0_{9,17} & \dots & 0_{9,23} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{20,1} & \dots & 0_{20,6} & 1_{20,7} & \dots & 1_{20,16} & 0_{20,17} & \dots & 0_{20,23} \\
0_{21,1} & \dots & 0_{21,6} & 0_{21,7} & \dots & 0_{21,16} & 1_{21,17} & \dots & 1_{21,23} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{29,1} & \dots & 0_{29,6} & 0_{29,7} & \dots & 0_{29,16} & 1_{29,17} & \dots & 1_{29,23}
\end{bmatrix}
$$

$$
=
\begin{bmatrix}
w_{1,1} & \dots & w_{1,6} & 0_{1,7} & \dots & 0_{1,16} & 0_{1,17} & \dots & 0_{1,23} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{8,1} & \dots & w_{8,6} & 0_{8,7} & \dots & 0_{8,16} & 0_{8,17} & \dots & 0_{8,23} \\
0_{9,1} & \dots & 0_{9,6} & w_{9,7} & \dots & w_{9,16} & 0_{9,17} & \dots & 0_{9,23} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{20,1} & \dots & 0_{20,6} & w_{20,7} & \dots & w_{20,16} & 0_{20,17} & \dots & 0_{20,23} \\
0_{21,1} & \dots & 0_{21,6} & 0_{21,7} & \dots & 0_{21,16} & w_{21,17} & \dots & w_{21,23} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{29,1} & \dots & 0_{29,6} & 0_{29,7} & \dots & 0_{29,16} & w_{29,17} & \dots & w_{29,23}
\end{bmatrix}
$$

Hadamard Product of Weight Matrix and Binary Mask Matrix for Module 2 (28 to 20)

$$
\begin{bmatrix}
w_{1,1} & \dots & w_{1,4} & w_{1,5} & \dots & w_{1,13} & w_{1,14} & \dots & w_{1,20} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{7,1} & \dots & w_{7,4} & w_{7,5} & \dots & w_{7,13} & w_{7,14} & \dots & w_{7,20} \\
w_{8,1} & \dots & w_{8,4} & w_{8,5} & \dots & w_{8,13} & w_{8,14} & \dots & w_{8,20} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{19,1} & \dots & w_{19,4} & w_{19,5} & \dots & w_{19,13} & w_{19,14} & \dots & w_{19,20} \\
w_{20,1} & \dots & w_{20,4} & w_{20,5} & \dots & w_{20,13} & w_{20,14} & \dots & w_{20,20} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{28,1} & \dots & w_{28,4} & w_{28,5} & \dots & w_{28,13} & w_{28,14} & \dots & w_{28,20}
\end{bmatrix}
\odot
\begin{bmatrix}
1_{1,1} & \dots & 1_{1,4} & 0_{1,5} & \dots & 0_{1,13} & 0_{1,14} & \dots & 0_{1,20} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
1_{7,1} & \dots & 1_{7,4} & 0_{7,5} & \dots & 0_{7,13} & 0_{7,14} & \dots & 0_{7,20} \\
0_{8,1} & \dots & 0_{8,4} & 1_{8,5} & \dots & 1_{8,13} & 0_{8,14} & \dots & 0_{8,20} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{19,1} & \dots & 0_{19,4} & 1_{19,5} & \dots & 1_{19,13} & 0_{19,14} & \dots & 0_{19,20} \\
0_{20,1} & \dots & 0_{20,4} & 0_{20,5} & \dots & 0_{20,13} & 1_{20,14} & \dots & 1_{20,20} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{28,1} & \dots & 0_{28,4} & 0_{28,5} & \dots & 0_{28,13} & 1_{28,14} & \dots & 1_{28,20}
\end{bmatrix}
$$

$$
=
\begin{bmatrix}
w_{1,1} & \dots & w_{1,4} & 0_{1,5} & \dots & 0_{1,13} & 0_{1,14} & \dots & 0_{1,20} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{7,1} & \dots & w_{7,4} & 0_{7,5} & \dots & 0_{7,13} & 0_{7,14} & \dots & 0_{7,20} \\
0_{8,1} & \dots & 0_{8,4} & w_{8,5} & \dots & w_{8,13} & 0_{8,14} & \dots & 0_{8,20} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{19,1} & \dots & 0_{19,4} & w_{19,5} & \dots & w_{19,13} & 0_{19,14} & \dots & 0_{19,20} \\
0_{20,1} & \dots & 0_{20,4} & 0_{20,5} & \dots & 0_{20,13} & w_{20,14} & \dots & w_{20,20} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{28,1} & \dots & 0_{28,4} & 0_{28,5} & \dots & 0_{28,13} & w_{28,14} & \dots & w_{28,20}
\end{bmatrix}
$$

Hadamard Product of Weight Matrix and Binary Mask Matrix for Module 2 (28 to 21)

$$
\begin{bmatrix}
w_{1,1} & \dots & w_{1,5} & w_{1,6} & \dots & w_{1,15} & w_{1,16} & \dots & w_{1,21} \\
\vdots & \dots & \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots \\
w_{7,1} & \dots & w_{7,5} & w_{7,6} & \dots & w_{7,15} & w_{7,16} & \dots & w_{7,21} \\
w_{8,1} & \dots & w_{8,5} & w_{8,6} & \dots & w_{8,15} & w_{8,16} & \dots & w_{8,21} \\
\vdots & \dots & \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots \\
w_{19,1} & \dots & w_{19,5} & w_{19,6} & \dots & w_{19,15} & w_{19,16} & \dots & w_{19,21} \\
w_{20,1} & \dots & w_{20,5} & w_{20,6} & \dots & w_{20,15} & w_{20,16} & \dots & w_{20,21} \\
\vdots & \dots & \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots \\
w_{28,1} & \dots & w_{28,5} & w_{28,6} & \dots & w_{28,15} & w_{28,16} & \dots & w_{28,21}
\end{bmatrix}
\odot
\begin{bmatrix}
1_{1,1} & \dots & 1_{1,5} & 0_{1,6} & \dots & 0_{1,15} & 0_{1,16} & \dots & 0_{1,21} \\
\vdots & \dots & \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots \\
1_{7,1} & \dots & 1_{7,5} & 0_{7,6} & \dots & 0_{7,15} & 0_{7,16} & \dots & 0_{7,21} \\
0_{8,1} & \dots & 0_{8,5} & 1_{8,6} & \dots & 1_{8,15} & 0_{8,16} & \dots & 0_{8,21} \\
\vdots & \dots & \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots \\
0_{19,1} & \dots & 0_{19,5} & 1_{19,6} & \dots & 1_{19,15} & 0_{19,16} & \dots & 0_{19,21} \\
0_{20,1} & \dots & 0_{20,5} & 0_{20,6} & \dots & 0_{20,15} & 1_{20,16} & \dots & 1_{20,21} \\
\vdots & \dots & \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots \\
0_{28,1} & \dots & 0_{28,5} & 0_{28,6} & \dots & 0_{28,15} & 1_{28,16} & \dots & 1_{28,21}
\end{bmatrix}
$$

$$
=
\begin{bmatrix}
w_{1,1} & \dots & w_{1,5} & 0_{1,6} & \dots & 0_{1,15} & 0_{1,16} & \dots & 0_{1,21} \\
\vdots & \dots & \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots \\
w_{7,1} & \dots & w_{7,5} & 0_{7,6} & \dots & 0_{7,15} & 0_{7,16} & \dots & 0_{7,21} \\
0_{8,1} & \dots & 0_{8,5} & w_{8,6} & \dots & w_{8,15} & 0_{8,16} & \dots & 0_{8,21} \\
\vdots & \dots & \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots \\
0_{19,1} & \dots & 0_{19,5} & w_{19,6} & \dots & w_{19,15} & 0_{19,16} & \dots & 0_{19,21} \\
0_{20,1} & \dots & 0_{20,5} & 0_{20,6} & \dots & 0_{20,15} & w_{20,16} & \dots & w_{20,21} \\
\vdots & \dots & \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots \\
0_{28,1} & \dots & 0_{28,5} & 0_{28,6} & \dots & 0_{28,15} & w_{28,16} & \dots & w_{28,21}
\end{bmatrix}
$$

Hadamard Product of Weight Matrix and Binary Mask Matrix for Module 2 (28 to 22)

$$
\begin{bmatrix}
w_{1,1} & \dots & w_{1,5} & w_{1,6} & \dots & w_{1,15} & w_{1,16} & \dots & w_{1,22} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{7,1} & \dots & w_{7,5} & w_{7,6} & \dots & w_{7,15} & w_{7,16} & \dots & w_{7,22} \\
w_{8,1} & \dots & w_{8,5} & w_{8,6} & \dots & w_{8,15} & w_{8,16} & \dots & w_{8,22} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{19,1} & \dots & w_{19,5} & w_{19,6} & \dots & w_{19,15} & w_{19,16} & \dots & w_{19,22} \\
w_{20,1} & \dots & w_{20,5} & w_{20,6} & \dots & w_{20,15} & w_{20,16} & \dots & w_{20,22} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{28,1} & \dots & w_{28,5} & w_{28,6} & \dots & w_{28,15} & w_{28,16} & \dots & w_{28,22}
\end{bmatrix}
\odot
\begin{bmatrix}
1_{1,1} & \dots & 1_{1,5} & 0_{1,6} & \dots & 0_{1,15} & 0_{1,16} & \dots & 0_{1,22} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
1_{7,1} & \dots & 1_{7,5} & 0_{7,6} & \dots & 0_{7,15} & 0_{7,16} & \dots & 0_{7,22} \\
0_{8,1} & \dots & 0_{8,5} & 1_{8,6} & \dots & 1_{8,15} & 0_{8,16} & \dots & 0_{8,22} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{19,1} & \dots & 0_{19,5} & 1_{19,6} & \dots & 1_{19,15} & 0_{19,16} & \dots & 0_{19,22} \\
0_{20,1} & \dots & 0_{20,5} & 0_{20,6} & \dots & 0_{20,15} & 1_{20,16} & \dots & 1_{20,22} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{28,1} & \dots & 0_{28,5} & 0_{28,6} & \dots & 0_{28,15} & 1_{28,16} & \dots & 1_{28,22}
\end{bmatrix}
$$

$$
=
\begin{bmatrix}
w_{1,1} & \dots & w_{1,5} & 0_{1,6} & \dots & 0_{1,15} & 0_{1,16} & \dots & 0_{1,22} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{7,1} & \dots & w_{7,5} & 0_{7,6} & \dots & 0_{7,15} & 0_{7,16} & \dots & 0_{7,22} \\
0_{8,1} & \dots & 0_{8,5} & w_{8,6} & \dots & w_{8,15} & 0_{8,16} & \dots & 0_{8,22} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{19,1} & \dots & 0_{19,5} & w_{19,6} & \dots & w_{19,15} & 0_{19,16} & \dots & 0_{19,22} \\
0_{20,1} & \dots & 0_{20,5} & 0_{20,6} & \dots & 0_{20,15} & w_{20,16} & \dots & w_{20,22} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{28,1} & \dots & 0_{28,5} & 0_{28,6} & \dots & 0_{28,15} & w_{28,16} & \dots & w_{28,22}
\end{bmatrix}
$$

Hadamard Product of Weight Matrix and Binary Mask Matrix for Module 3 (28 to 20)

$$
\begin{bmatrix}
w_{1,1} & \cdots & w_{1,6} & w_{1,7} & \cdots & w_{1,17} & w_{1,14} & \cdots & w_{1,20} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{9,1} & \cdots & w_{9,6} & w_{9,9} & \cdots & w_{9,17} & w_{9,14} & \cdots & w_{9,20} \\
w_{10,1} & \cdots & w_{10,6} & w_{10,9} & \cdots & w_{10,17} & w_{10,14} & \cdots & w_{10,20} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{19,1} & \cdots & w_{19,6} & w_{19,9} & \cdots & w_{19,17} & w_{19,14} & \cdots & w_{19,20} \\
w_{20,1} & \cdots & w_{20,6} & w_{20,9} & \cdots & w_{20,17} & w_{20,14} & \cdots & w_{20,20} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{28,1} & \cdots & w_{28,6} & w_{28,9} & \cdots & w_{28,17} & w_{28,14} & \cdots & w_{28,20}
\end{bmatrix}
\odot
\begin{bmatrix}
1_{1,1} & \cdots & 1_{1,6} & 0_{1,7} & \cdots & 0_{1,13} & 0_{1,14} & \cdots & 0_{1,20} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
1_{9,1} & \cdots & 1_{9,6} & 0_{9,7} & \cdots & 0_{9,13} & 0_{9,14} & \cdots & 0_{9,20} \\
0_{10,1} & \cdots & 0_{10,6} & 1_{10,7} & \cdots & 1_{10,13} & 0_{10,14} & \cdots & 0_{10,20} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{19,1} & \cdots & 0_{19,6} & 1_{19,7} & \cdots & 1_{19,13} & 0_{19,14} & \cdots & 0_{19,20} \\
0_{20,1} & \cdots & 0_{20,6} & 0_{20,7} & \cdots & 0_{20,13} & 1_{20,14} & \cdots & 1_{20,20} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{28,1} & \cdots & 0_{28,6} & 0_{28,7} & \cdots & 0_{28,13} & 1_{28,14} & \cdots & 1_{28,20}
\end{bmatrix}
$$

$$
=
\begin{bmatrix}
w_{1,1} & \cdots & w_{1,6} & 0_{1,7} & \cdots & 0_{1,13} & 0_{1,14} & \cdots & 0_{1,20} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{9,1} & \cdots & w_{9,6} & 0_{9,7} & \cdots & 0_{9,13} & 0_{9,14} & \cdots & 0_{9,20} \\
0_{10,1} & \cdots & 0_{10,6} & w_{10,7} & \cdots & w_{10,13} & 0_{10,14} & \cdots & 0_{10,20} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{19,1} & \cdots & 0_{19,6} & w_{19,7} & \cdots & w_{19,13} & 0_{19,14} & \cdots & 0_{19,20} \\
0_{20,1} & \cdots & 0_{20,6} & 0_{20,7} & \cdots & 0_{20,13} & w_{20,14} & \cdots & w_{20,20} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{28,1} & \cdots & 0_{28,6} & 0_{28,7} & \cdots & 0_{28,13} & w_{28,14} & \cdots & w_{28,20}
\end{bmatrix}
$$

Hadamard Product of Weight Matrix and Binary Mask Matrix for Module 3 (28 to 21)

$$
\begin{bmatrix}
w_{1,1} & \cdots & w_{1,7} & w_{1,8} & \cdots & w_{1,15} & w_{1,16} & \cdots & w_{1,21} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{9,1} & \cdots & w_{9,7} & w_{9,8} & \cdots & w_{9,15} & w_{9,16} & \cdots & w_{9,21} \\
w_{10,1} & \cdots & w_{10,7} & w_{10,8} & \cdots & w_{10,15} & w_{10,16} & \cdots & w_{10,21} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{19,1} & \cdots & w_{19,7} & w_{19,8} & \cdots & w_{19,15} & w_{19,16} & \cdots & w_{19,21} \\
w_{20,1} & \cdots & w_{20,7} & w_{20,8} & \cdots & w_{20,15} & w_{20,16} & \cdots & w_{20,21} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{28,1} & \cdots & w_{28,7} & w_{28,8} & \cdots & w_{28,15} & w_{28,16} & \cdots & w_{28,21}
\end{bmatrix}
\odot
\begin{bmatrix}
1_{1,1} & \cdots & 1_{1,7} & 0_{1,8} & \cdots & 0_{1,15} & 0_{1,16} & \cdots & 0_{1,21} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
1_{9,1} & \cdots & 1_{9,7} & 0_{9,8} & \cdots & 0_{9,15} & 0_{9,16} & \cdots & 0_{9,21} \\
0_{10,1} & \cdots & 0_{10,7} & 1_{10,8} & \cdots & 1_{10,15} & 0_{10,16} & \cdots & 0_{10,21} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{19,1} & \cdots & 0_{19,7} & 1_{19,8} & \cdots & 1_{19,15} & 0_{19,16} & \cdots & 0_{19,21} \\
0_{20,1} & \cdots & 0_{20,7} & 0_{20,8} & \cdots & 0_{20,15} & 1_{20,16} & \cdots & 1_{20,21} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{28,1} & \cdots & 0_{28,7} & 0_{28,8} & \cdots & 0_{28,15} & 0_{28,16} & \cdots & 1_{28,21}
\end{bmatrix}
$$

$$
=
\begin{bmatrix}
w_{1,1} & \cdots & w_{1,7} & 0_{1,8} & \cdots & 0_{1,15} & 0_{1,16} & \cdots & 0_{1,21} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{9,1} & \cdots & w_{9,7} & 0_{9,8} & \cdots & 0_{9,15} & 0_{9,16} & \cdots & 0_{9,21} \\
0_{10,1} & \cdots & 0_{10,7} & w_{10,8} & \cdots & w_{10,15} & 0_{10,16} & \cdots & 0_{10,21} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{19,1} & \cdots & 0_{19,7} & w_{19,8} & \cdots & w_{19,15} & 0_{19,16} & \cdots & 0_{19,21} \\
0_{20,1} & \cdots & 0_{20,7} & 0_{20,8} & \cdots & 0_{20,15} & w_{20,16} & \cdots & w_{20,21} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{28,1} & \cdots & 0_{28,7} & 0_{28,8} & \cdots & 0_{28,15} & w_{28,16} & \cdots & w_{28,21}
\end{bmatrix}
$$

Hadamard Product of Weight Matrix and Binary Mask Matrix for Module 3 (28 to 22)

$$
\begin{bmatrix}
w_{1,1} & \dots & w_{1,7} & w_{1,8} & \dots & w_{1,15} & w_{1,16} & \dots & w_{1,22} \\
\vdots & \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \vdots \\
w_{9,1} & \dots & w_{9,7} & w_{9,8} & \dots & w_{9,15} & w_{9,16} & \dots & w_{9,21} \\
w_{10,1} & \dots & w_{10,7} & w_{10,8} & \dots & w_{10,15} & w_{10,16} & \dots & w_{10,22} \\
\vdots & \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \vdots \\
w_{19,1} & \dots & w_{19,7} & w_{19,8} & \dots & w_{19,15} & w_{19,16} & \dots & w_{19,22} \\
w_{20,1} & \dots & w_{20,7} & w_{20,8} & \dots & w_{20,15} & w_{20,16} & \dots & w_{20,22} \\
\vdots & \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \vdots \\
w_{28,1} & \dots & w_{28,7} & w_{28,8} & \dots & w_{28,15} & w_{28,16} & \dots & w_{28,22}
\end{bmatrix}
\odot
\begin{bmatrix}
1_{1,1} & \dots & 1_{1,7} & 0_{1,8} & \dots & 0_{1,15} & 0_{1,16} & \dots & 0_{1,22} \\
\vdots & \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \vdots \\
1_{9,1} & \dots & 1_{9,7} & 0_{9,8} & \dots & 0_{9,15} & 0_{9,16} & \dots & 0_{9,22} \\
0_{10,1} & \dots & 0_{10,7} & 1_{10,8} & \dots & 1_{10,15} & 0_{10,16} & \dots & 0_{10,22} \\
\vdots & \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \vdots \\
0_{19,1} & \dots & 0_{19,7} & 1_{19,8} & \dots & 1_{19,15} & 0_{19,16} & \dots & 0_{19,22} \\
0_{20,1} & \dots & 0_{20,7} & 0_{20,8} & \dots & 0_{20,15} & 1_{20,16} & \dots & 1_{20,22} \\
\vdots & \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \vdots \\
0_{28,1} & \dots & 0_{28,7} & 0_{28,8} & \dots & 0_{28,15} & 1_{28,16} & \dots & 1_{28,22}
\end{bmatrix}
$$

$$
=
\begin{bmatrix}
w_{1,1} & \dots & w_{1,7} & 0_{1,8} & \dots & 0_{1,15} & 0_{1,16} & \dots & 0_{1,22} \\
\vdots & \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \vdots \\
w_{9,1} & \dots & w_{9,7} & 0_{9,8} & \dots & 0_{9,15} & 0_{9,16} & \dots & 0_{9,22} \\
0_{10,1} & \dots & 0_{10,7} & w_{10,8} & \dots & w_{10,15} & 0_{10,16} & \dots & 0_{10,22} \\
\vdots & \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \vdots \\
0_{19,1} & \dots & 0_{19,7} & w_{19,8} & \dots & w_{19,15} & 0_{19,16} & \dots & 0_{19,22} \\
0_{20,1} & \dots & 0_{20,7} & 0_{20,8} & \dots & 0_{20,15} & w_{20,16} & \dots & w_{20,22} \\
\vdots & \vdots & \vdots & \vdots & \dots & \vdots & \vdots & \vdots & \vdots \\
0_{28,1} & \dots & 0_{28,7} & 0_{28,8} & \dots & 0_{28,15} & w_{28,16} & \dots & w_{28,22}
\end{bmatrix}
$$

Hadamard Product of Weight Matrix and Binary Mask Matrix for Module 4 (31 to 20)

$$
\begin{bmatrix}
w_{1,1} & \dots & w_{1,6} & w_{1,7} & \dots & w_{1,13} & w_{1,14} & \dots & w_{1,20} \\
\vdots & \dots & \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots \\
w_{10,1} & \dots & w_{10,6} & w_{10,7} & \dots & w_{10,13} & w_{10,14} & \dots & w_{10,20} \\
w_{11,1} & \dots & w_{11,6} & w_{11,7} & \dots & w_{11,13} & w_{11,14} & \dots & w_{11,20} \\
\vdots & \dots & \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots \\
w_{22,1} & \dots & w_{22,6} & w_{22,7} & \dots & w_{22,13} & w_{22,14} & \dots & w_{22,20} \\
w_{23,1} & \dots & w_{23,6} & w_{23,7} & \dots & w_{23,13} & w_{23,14} & \dots & w_{23,20} \\
\vdots & \dots & \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots \\
w_{31,1} & \dots & w_{31,6} & w_{31,7} & \dots & w_{31,13} & w_{31,14} & \dots & w_{31,20}
\end{bmatrix}
\odot
\begin{bmatrix}
1_{1,1} & \dots & 1_{1,6} & 0_{1,7} & \dots & 0_{1,13} & 0_{1,14} & \dots & 0_{1,20} \\
\vdots & \dots & \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots \\
1_{10,1} & \dots & 1_{10,6} & 0_{10,7} & \dots & 0_{10,13} & 0_{10,14} & \dots & 0_{10,20} \\
0_{11,1} & \dots & 0_{11,6} & 1_{11,7} & \dots & 1_{11,13} & 0_{11,14} & \dots & 0_{11,20} \\
\vdots & \dots & \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots \\
0_{22,1} & \dots & 0_{22,6} & 1_{22,7} & \dots & 1_{22,13} & 0_{22,14} & \dots & 0_{22,20} \\
0_{23,1} & \dots & 0_{23,6} & 0_{23,7} & \dots & 0_{23,13} & 1_{23,14} & \dots & 1_{23,20} \\
\vdots & \dots & \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots \\
0_{31,1} & \dots & 0_{31,6} & 0_{31,7} & \dots & 0_{31,13} & 1_{31,14} & \dots & 1_{31,20}
\end{bmatrix}
$$

$$
=
\begin{bmatrix}
w_{1,1} & \dots & w_{1,6} & 0_{1,7} & \dots & 0_{1,13} & 0_{1,14} & \dots & 0_{1,20} \\
\vdots & \dots & \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots \\
w_{10,1} & \dots & w_{10,6} & 0_{10,7} & \dots & 0_{10,13} & 0_{10,14} & \dots & 0_{10,20} \\
0_{11,1} & \dots & 0_{11,6} & w_{11,7} & \dots & w_{11,13} & 0_{11,14} & \dots & 0_{11,20} \\
\vdots & \dots & \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots \\
0_{22,1} & \dots & 0_{22,6} & w_{22,7} & \dots & w_{22,13} & 0_{22,14} & \dots & 0_{22,20} \\
0_{23,1} & \dots & 0_{23,6} & 0_{23,7} & \dots & 0_{23,13} & w_{23,14} & \dots & w_{23,20} \\
\vdots & \dots & \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots \\
0_{31,1} & \dots & 0_{31,6} & 0_{31,7} & \dots & 0_{31,13} & w_{31,14} & \dots & w_{31,20}
\end{bmatrix}
$$

Hadamard Product of Weight Matrix and Binary Mask Matrix for Module 4 (31 to 21)

$$
\begin{bmatrix}
w_{1,1} & \dots & w_{1,7} & w_{1,8} & \dots & w_{1,15} & w_{1,16} & \dots & w_{1,21} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{10,1} & \dots & w_{10,7} & w_{10,8} & \dots & w_{10,15} & w_{10,16} & \dots & w_{10,21} \\
w_{11,1} & \dots & w_{11,7} & w_{11,8} & \dots & w_{11,15} & w_{11,16} & \dots & w_{11,21} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{22,1} & \dots & w_{22,7} & w_{22,8} & \dots & w_{22,15} & w_{22,16} & \dots & w_{22,21} \\
w_{23,1} & \dots & w_{23,7} & w_{23,8} & \dots & w_{23,15} & w_{23,16} & \dots & w_{23,21} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{31,1} & \dots & w_{31,7} & w_{31,8} & \dots & w_{31,15} & w_{31,16} & \dots & w_{31,21}
\end{bmatrix}
\odot
\begin{bmatrix}
1_{1,1} & \dots & 1_{1,7} & 0_{1,8} & \dots & 0_{1,15} & 0_{1,16} & \dots & 0_{1,21} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
1_{10,1} & \dots & 1_{10,7} & 0_{10,8} & \dots & 0_{10,15} & 0_{10,16} & \dots & 0_{10,21} \\
0_{11,1} & \dots & 0_{11,7} & 1_{11,8} & \dots & 1_{11,15} & 0_{11,16} & \dots & 0_{11,21} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{22,1} & \dots & 0_{22,7} & 1_{22,8} & \dots & 1_{22,15} & 0_{22,16} & \dots & 0_{22,21} \\
0_{23,1} & \dots & 0_{23,7} & 0_{23,8} & \dots & 0_{23,15} & 1_{23,16} & \dots & 1_{23,21} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{31,1} & \dots & 0_{31,7} & 0_{31,8} & \dots & 0_{31,15} & 1_{31,16} & \dots & 1_{31,21}
\end{bmatrix}
$$

$$
=
\begin{bmatrix}
w_{1,1} & \dots & w_{1,7} & 0_{1,8} & \dots & 0_{1,15} & 0_{1,16} & \dots & 0_{1,21} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{10,1} & \dots & w_{10,7} & 0_{10,8} & \dots & 0_{10,15} & 0_{10,16} & \dots & 0_{10,21} \\
0_{11,1} & \dots & 0_{11,7} & w_{11,8} & \dots & w_{11,15} & 0_{11,16} & \dots & 0_{11,21} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{22,1} & \dots & 0_{22,7} & w_{22,8} & \dots & w_{22,15} & 0_{22,16} & \dots & 0_{22,21} \\
0_{23,1} & \dots & 0_{23,7} & 0_{23,8} & \dots & 0_{23,15} & w_{23,16} & \dots & w_{23,21} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{31,1} & \dots & 0_{31,7} & 0_{31,8} & \dots & 0_{31,15} & w_{31,16} & \dots & w_{31,21}
\end{bmatrix}
$$

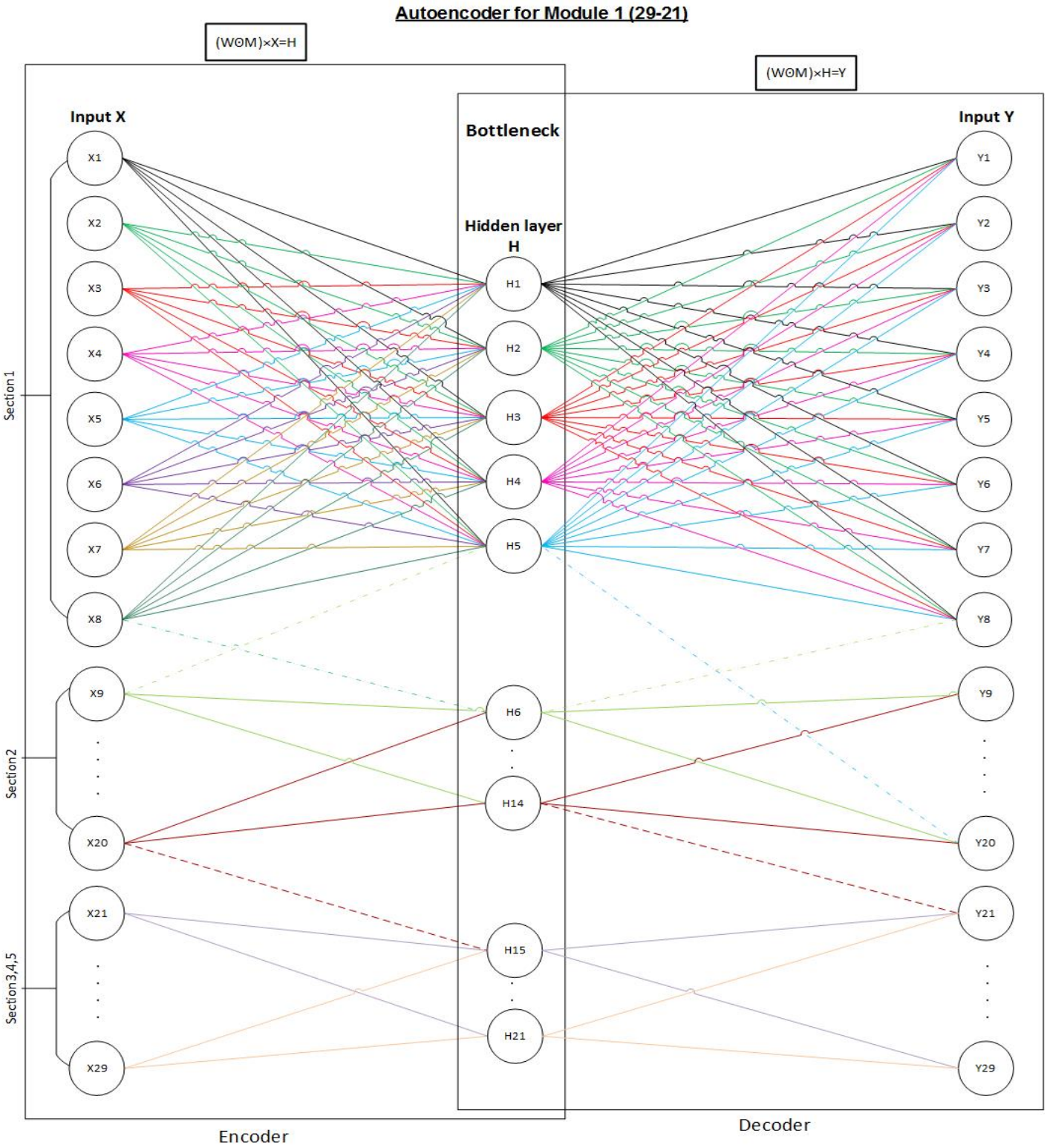
Hadamard Product of Weight Matrix and Binary Mask Matrix for Module 4 (31 to 22)

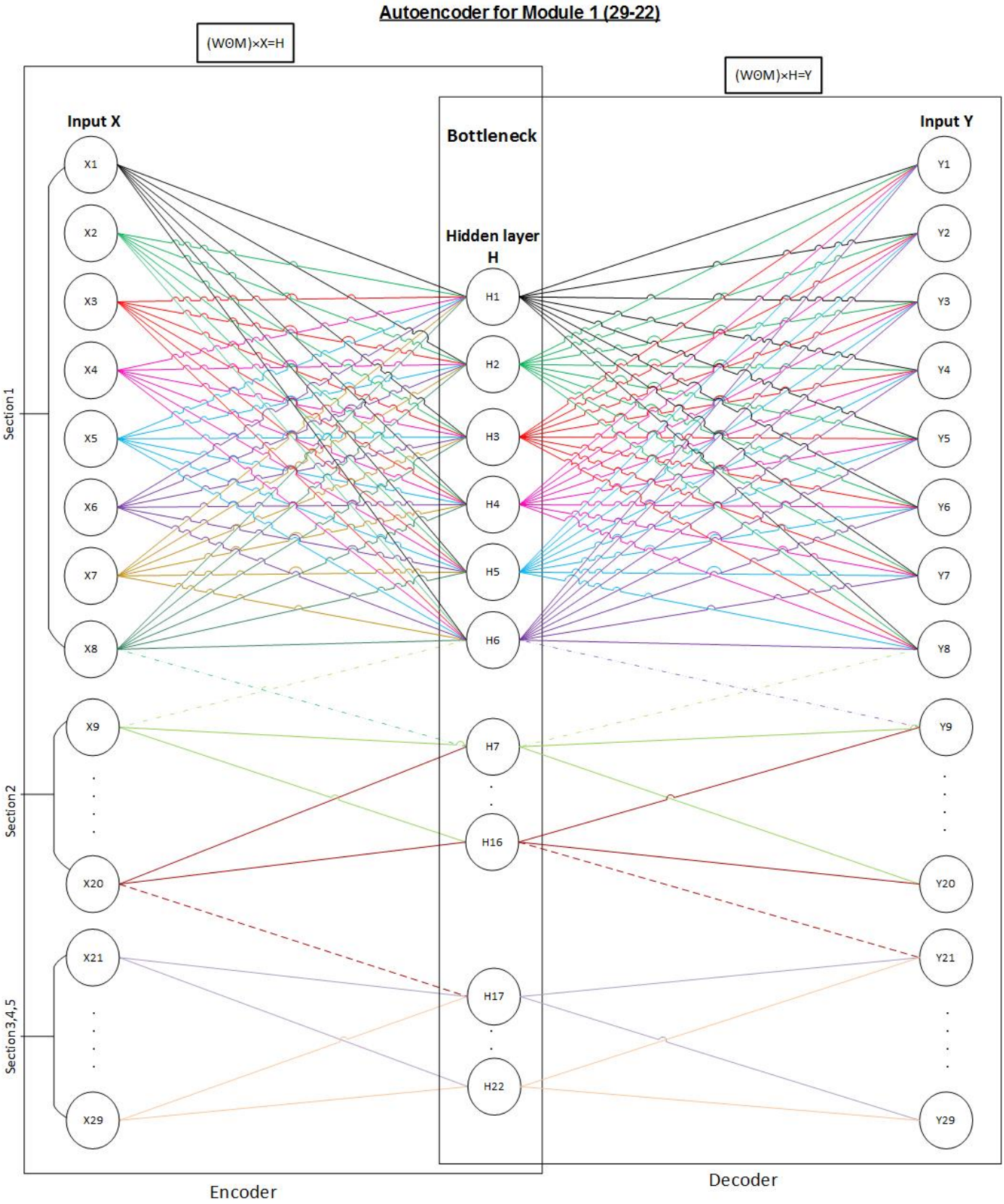
$$
\begin{bmatrix}
w_{1,1} & \dots & w_{1,7} & w_{1,8} & \dots & w_{1,16} & w_{1,17} & \dots & w_{1,22} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{10,1} & \dots & w_{10,7} & w_{10,8} & \dots & w_{10,16} & w_{10,17} & \dots & w_{10,22} \\
w_{11,1} & \dots & w_{11,7} & w_{11,8} & \dots & w_{11,16} & w_{11,17} & \dots & w_{11,22} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{22,1} & \dots & w_{22,7} & w_{22,8} & \dots & w_{22,16} & w_{22,17} & \dots & w_{22,22} \\
w_{23,1} & \dots & w_{23,7} & w_{23,8} & \dots & w_{23,16} & w_{23,17} & \dots & w_{23,22} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{31,1} & \dots & w_{31,7} & w_{31,8} & \dots & w_{31,16} & w_{31,17} & \dots & w_{31,22}
\end{bmatrix}
\odot
\begin{bmatrix}
1_{1,1} & \dots & 1_{1,7} & 0_{1,8} & \dots & 0_{1,16} & 0_{1,17} & \dots & 0_{1,22} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
1_{10,1} & \dots & 1_{10,7} & 0_{10,8} & \dots & 0_{10,16} & 0_{10,17} & \dots & 0_{10,22} \\
0_{11,1} & \dots & 0_{11,7} & 1_{11,8} & \dots & 1_{11,16} & 0_{11,17} & \dots & 0_{11,22} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{22,1} & \dots & 0_{22,7} & 1_{22,8} & \dots & 1_{22,16} & 0_{22,17} & \dots & 0_{22,22} \\
0_{23,1} & \dots & 0_{23,7} & 0_{23,8} & \dots & 0_{23,16} & 1_{23,17} & \dots & 1_{23,22} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{31,1} & \dots & 0_{31,7} & 0_{31,8} & \dots & 0_{31,16} & 1_{31,17} & \dots & 1_{31,22}
\end{bmatrix}
$$

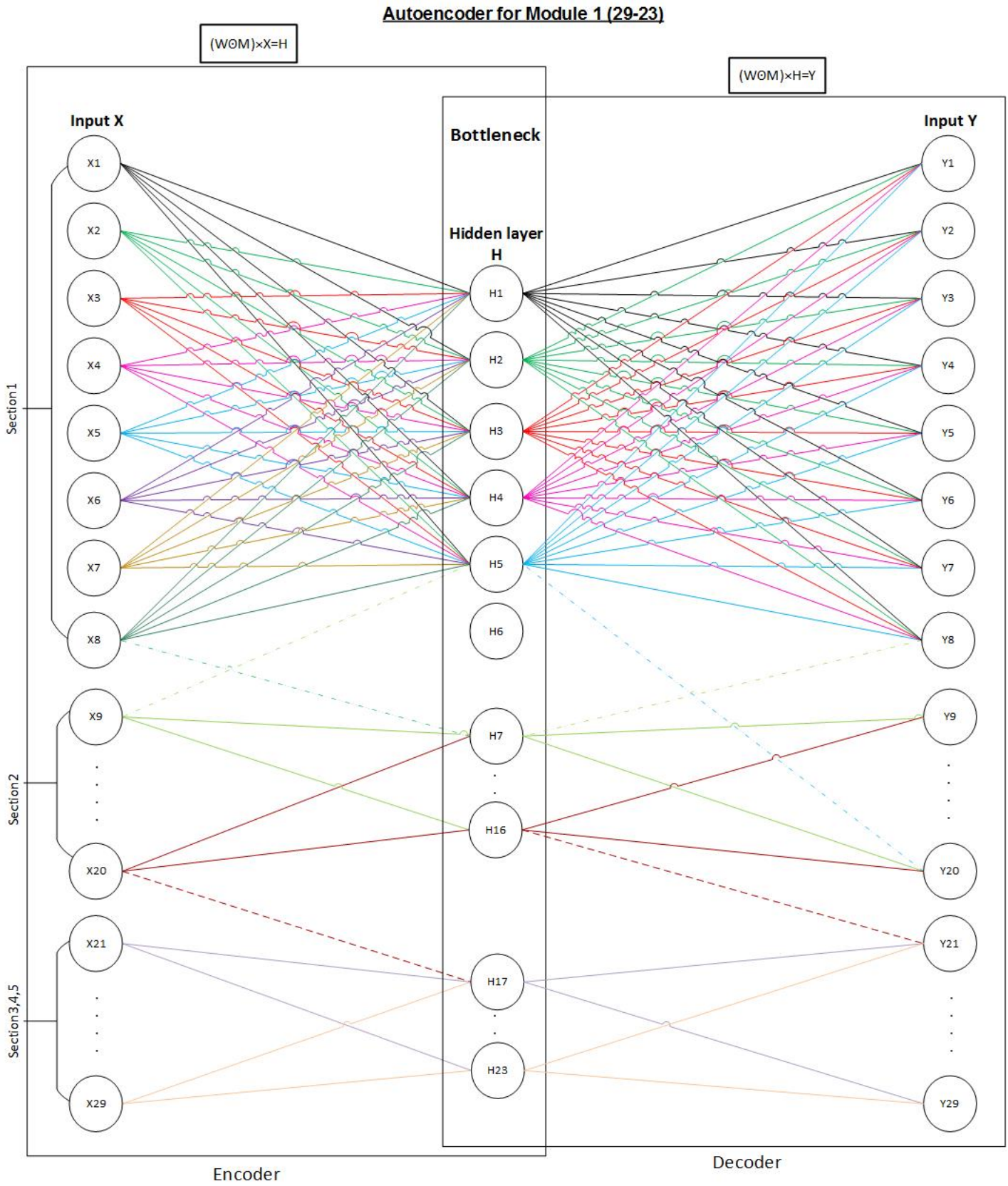
$$
=
\begin{bmatrix}
w_{1,1} & \dots & w_{1,7} & 0_{1,8} & \dots & 0_{1,16} & 0_{1,17} & \dots & 0_{1,22} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
w_{10,1} & \dots & w_{10,7} & 0_{10,8} & \dots & 0_{10,16} & 0_{10,17} & \dots & 0_{10,22} \\
0_{11,1} & \dots & 0_{11,7} & w_{11,8} & \dots & w_{11,16} & 0_{11,17} & \dots & 0_{11,22} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{22,1} & \dots & 0_{22,7} & w_{22,8} & \dots & w_{22,16} & 0_{22,17} & \dots & 0_{22,22} \\
0_{23,1} & \dots & 0_{23,7} & 0_{23,8} & \dots & 0_{23,16} & w_{23,17} & \dots & w_{23,22} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0_{31,1} & \dots & 0_{31,7} & 0_{31,8} & \dots & 0_{31,16} & w_{31,17} & \dots & w_{31,22}
\end{bmatrix}
$$

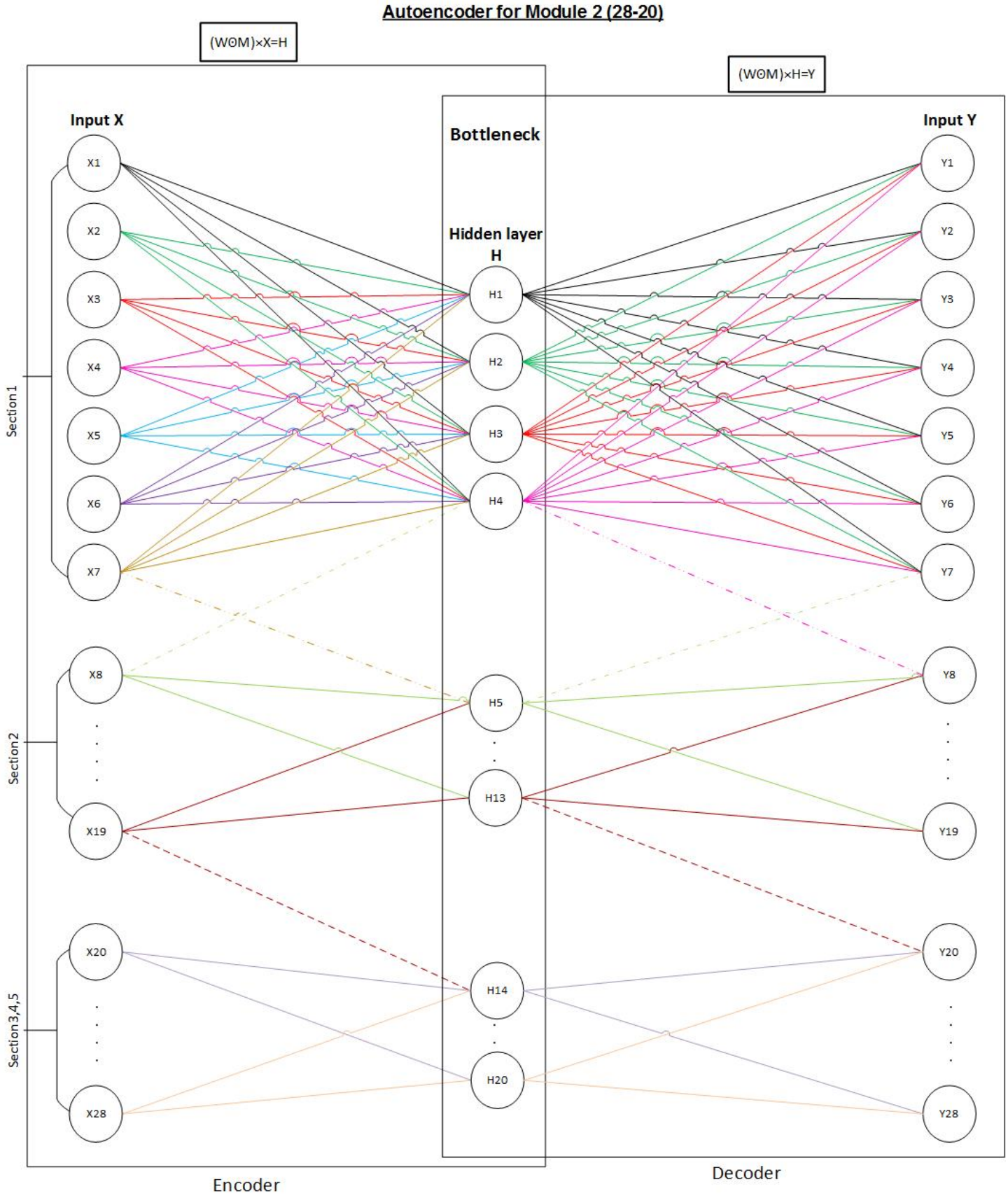
APPENDIX C

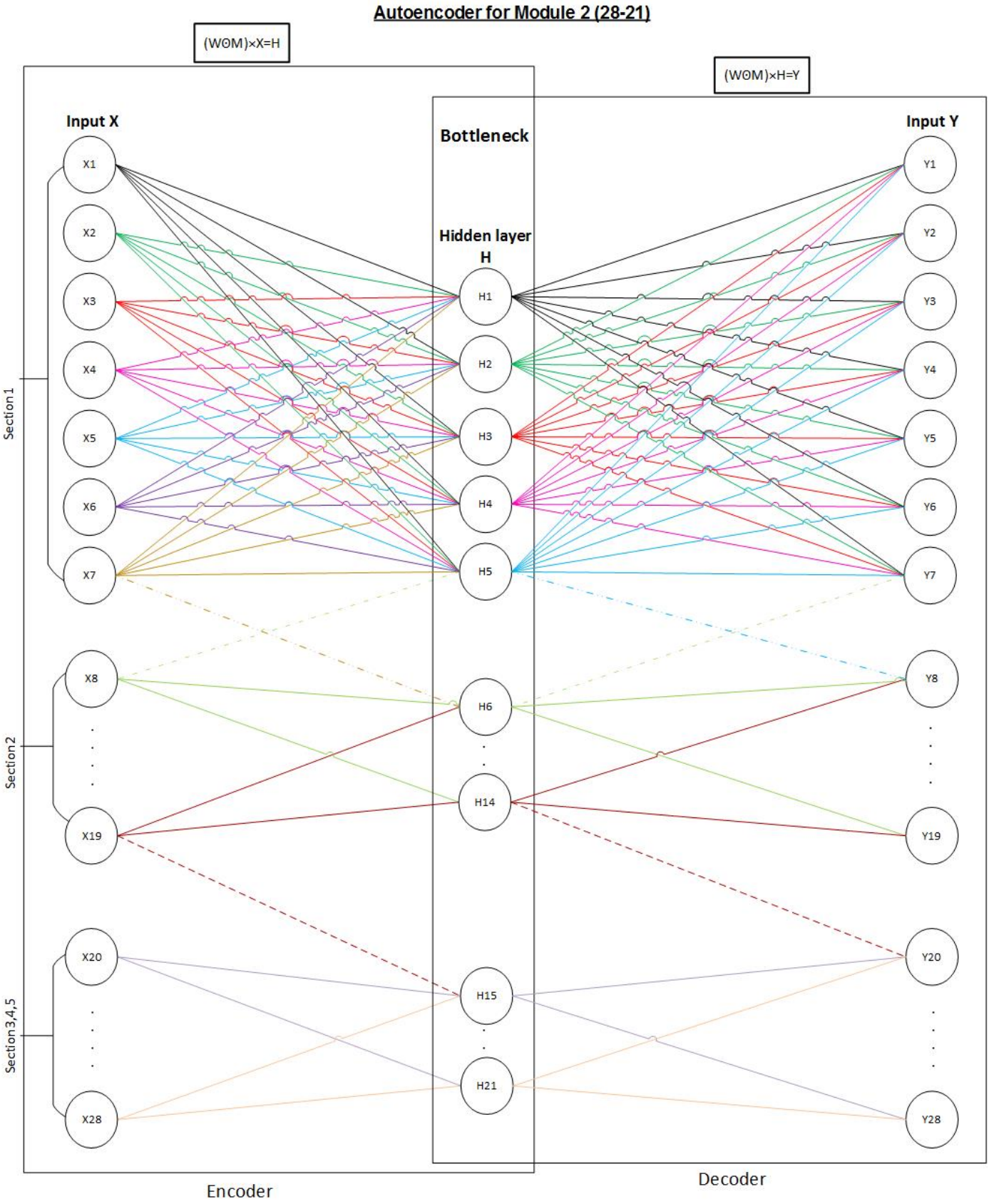
AUTOENCODER DIAGRAMS

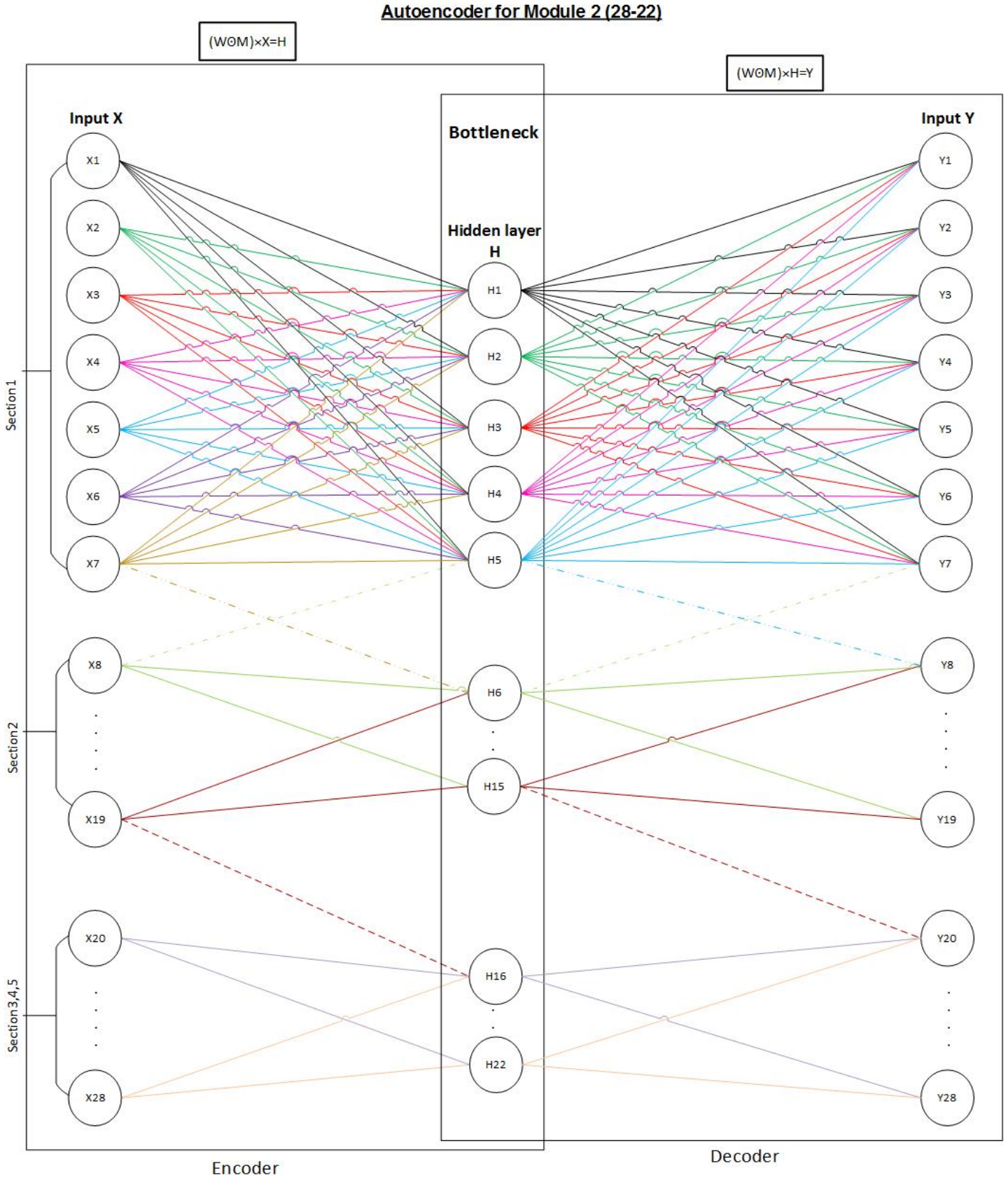

Autoencoder for Module 1 (29-21)
(WΘM)×X=H
(WΘM)×H=Y
Input X
Bottleneck
Hidden layer H
Input Y
X1
X2
X3
X4
X5
X6
X7
X8
X9
X20
X21
X29
H1
H2
H3
H4
H5
H6
H14
H15
H21
Y1
Y2
Y3
Y4
Y5
Y6
Y7
Y8
Y9
Y20
Y21
Y29
Section1
Section2
Section3,4,5
Encoder
Decoder

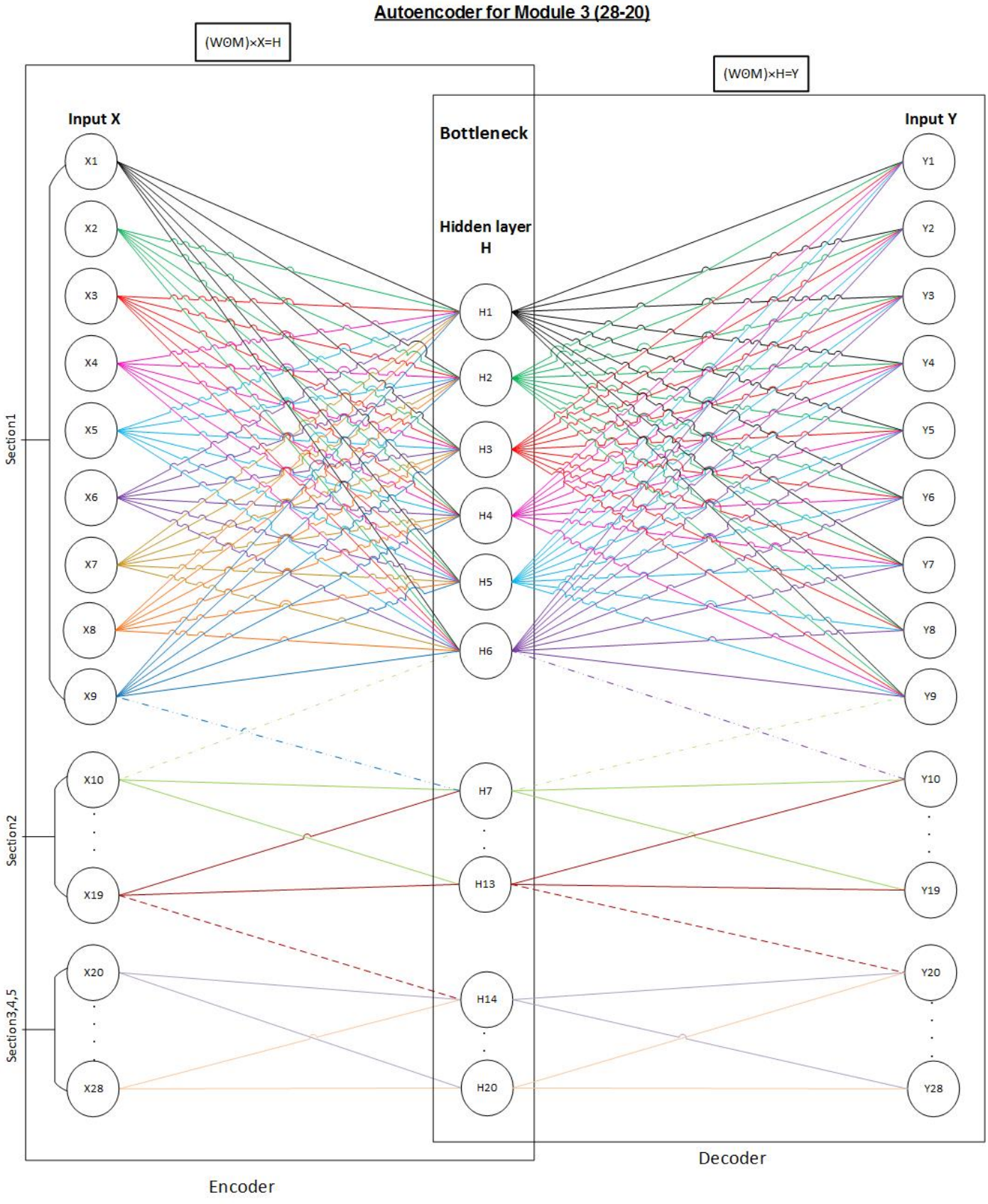

Autoencoder for Module 1 (29-22)
(W⊙M)×X=H
(W⊙M)×H=Y
Input X
Bottleneck
Hidden layer
H
Input Y
X1
X2
X3
X4
X5
X6
X7
X8
X9
X20
X21
X29
H1
H2
H3
H4
H5
H6
H7
H16
H17
H22
Y1
Y2
Y3
Y4
Y5
Y6
Y7
Y8
Y9
Y20
Y21
Y29
Section1
Section2
Section3,4,5
Encoder
Decoder

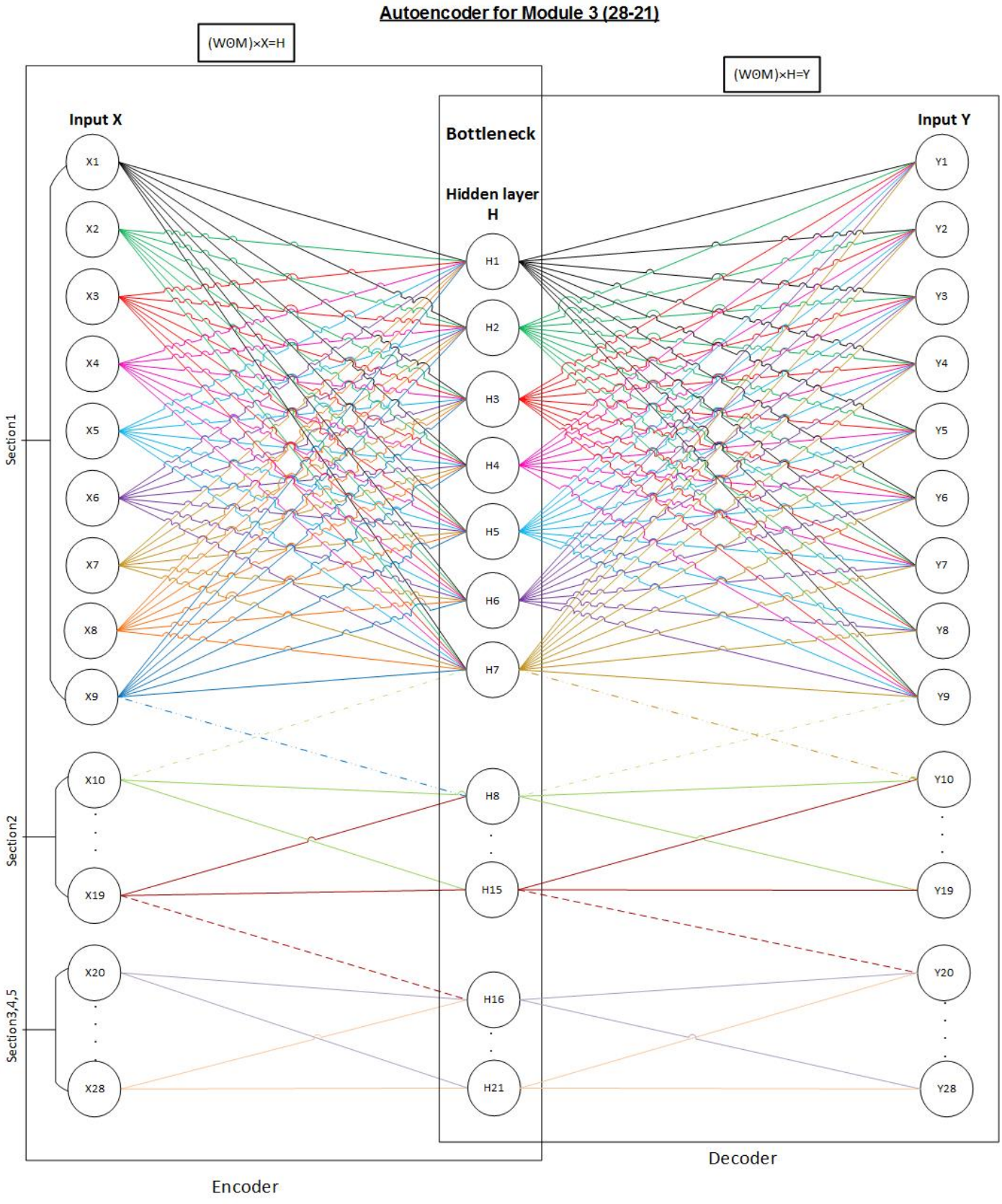

Autoencoder for Module 1 (29-23)
(W⊙M)×X=H
(W⊙M)×H=Y
Input X
Bottleneck
Hidden layer
H
Input Y
X1
X2
X3
X4
X5
X6
X7
X8
H1
H2
H3
H4
H5
H6
Y1
Y2
Y3
Y4
Y5
Y6
Y7
Y8
X9
X20
X21
X29
H7
H16
H17
H23
Y9
Y20
Y21
Y29
Section 1
Section 2
Section 3,4,5
Encoder
Decoder

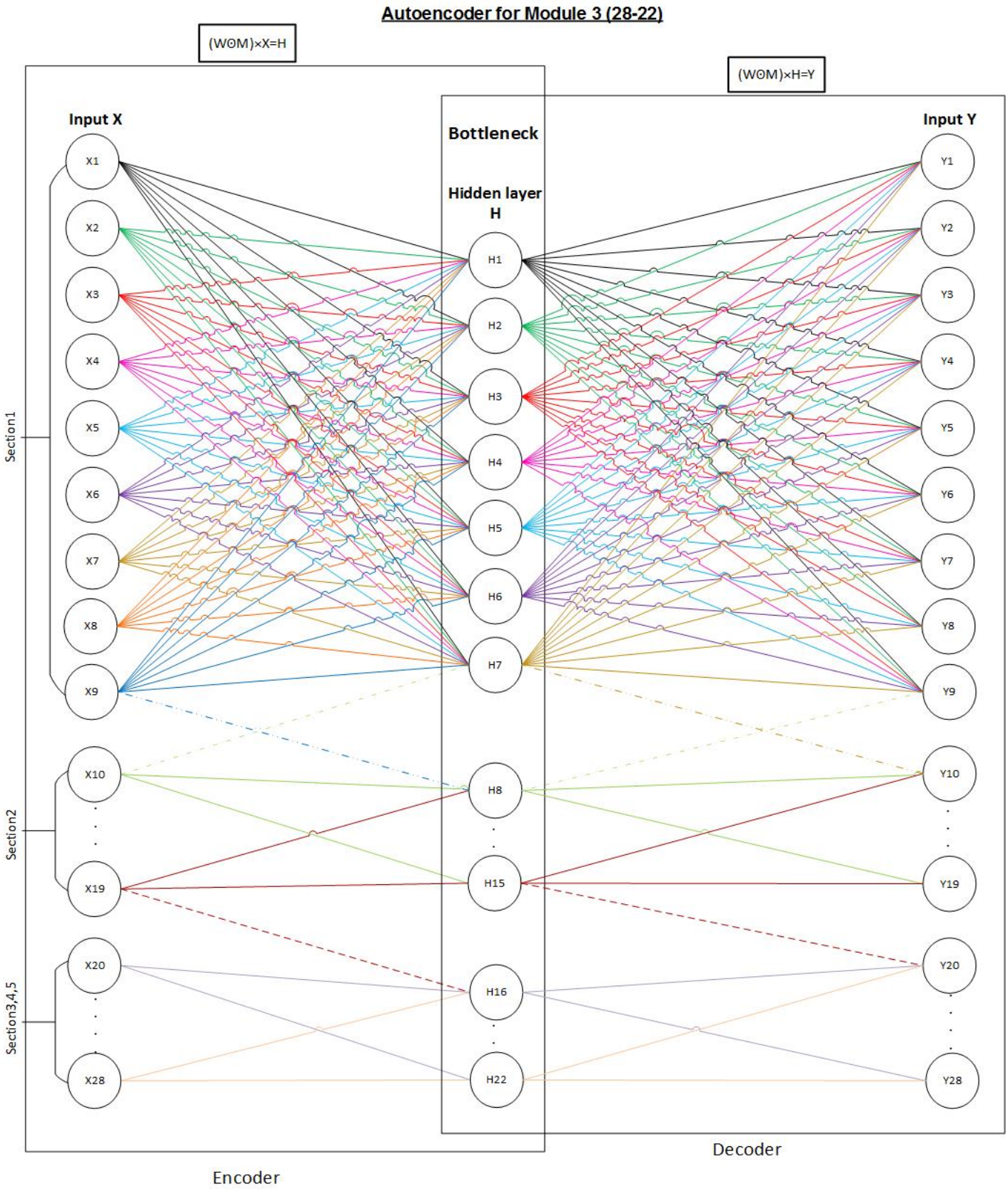

Autoencoder for Module 2 (28-20)
(WΘM)×X=H
(WΘM)×H=Y
Input X
Bottleneck
Hidden layer H
Input Y
X1
X2
X3
X4
X5
X6
X7
H1
H2
H3
H4
Y1
Y2
Y3
Y4
Y5
Y6
Y7
X8
X19
H5
H13
Y8
Y19
X20
X28
H14
H20
Y20
Y28
Section1
Section2
Section3,4,5
Encoder
Decoder

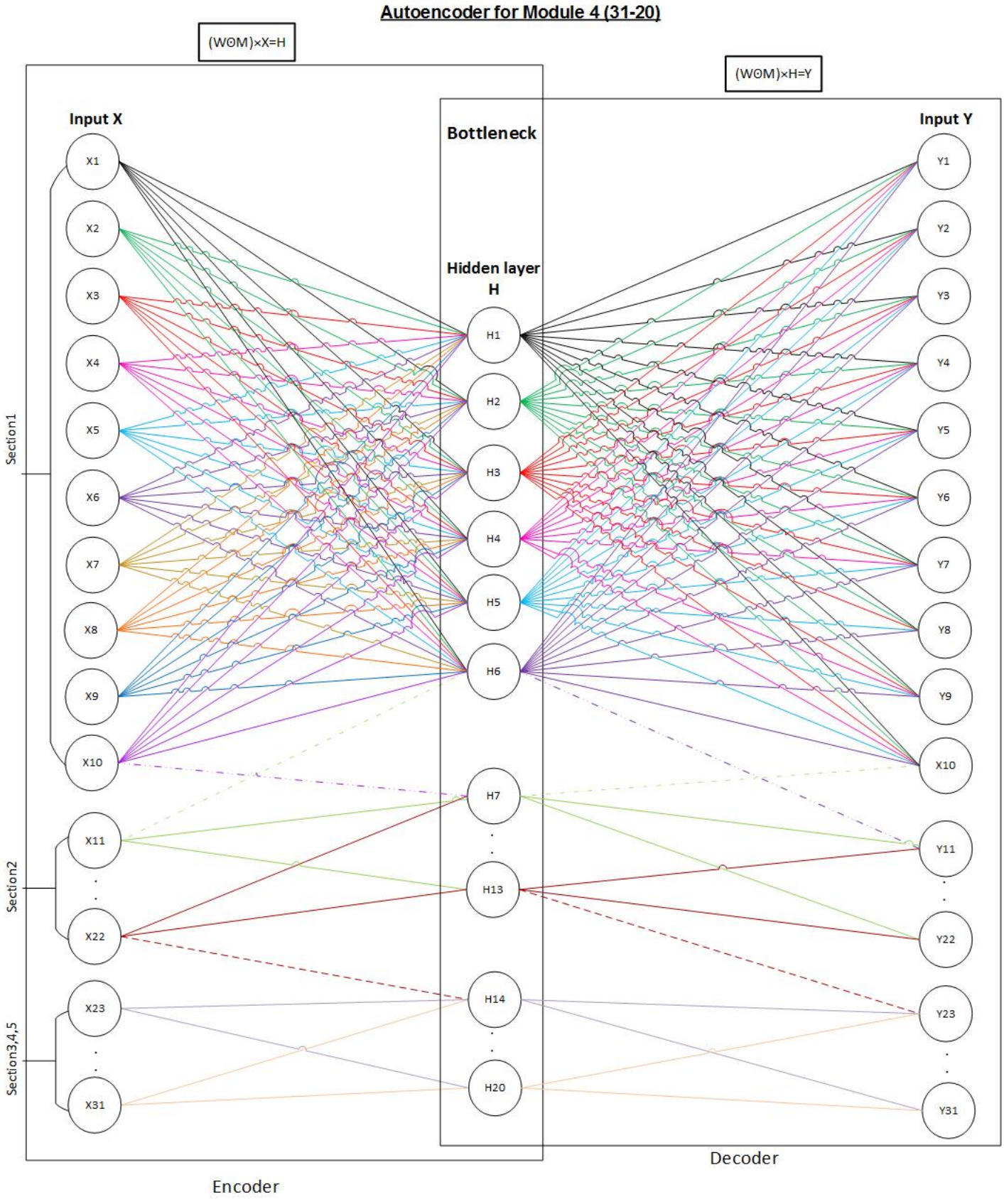

Autoencoder for Module 2 (28-21)
(WΘM)×X=H
(WΘM)×H=Y
Input X
Bottleneck
Hidden layer
H
Input Y
X1
X2
X3
X4
X5
X6
X7
H1
H2
H3
H4
H5
Y1
Y2
Y3
Y4
Y5
Y6
Y7
X8
X19
H6
H14
Y8
Y19
X20
X28
H15
H21
Y20
Y28
Section1
Section2
Section3,4,5
Encoder
Decoder

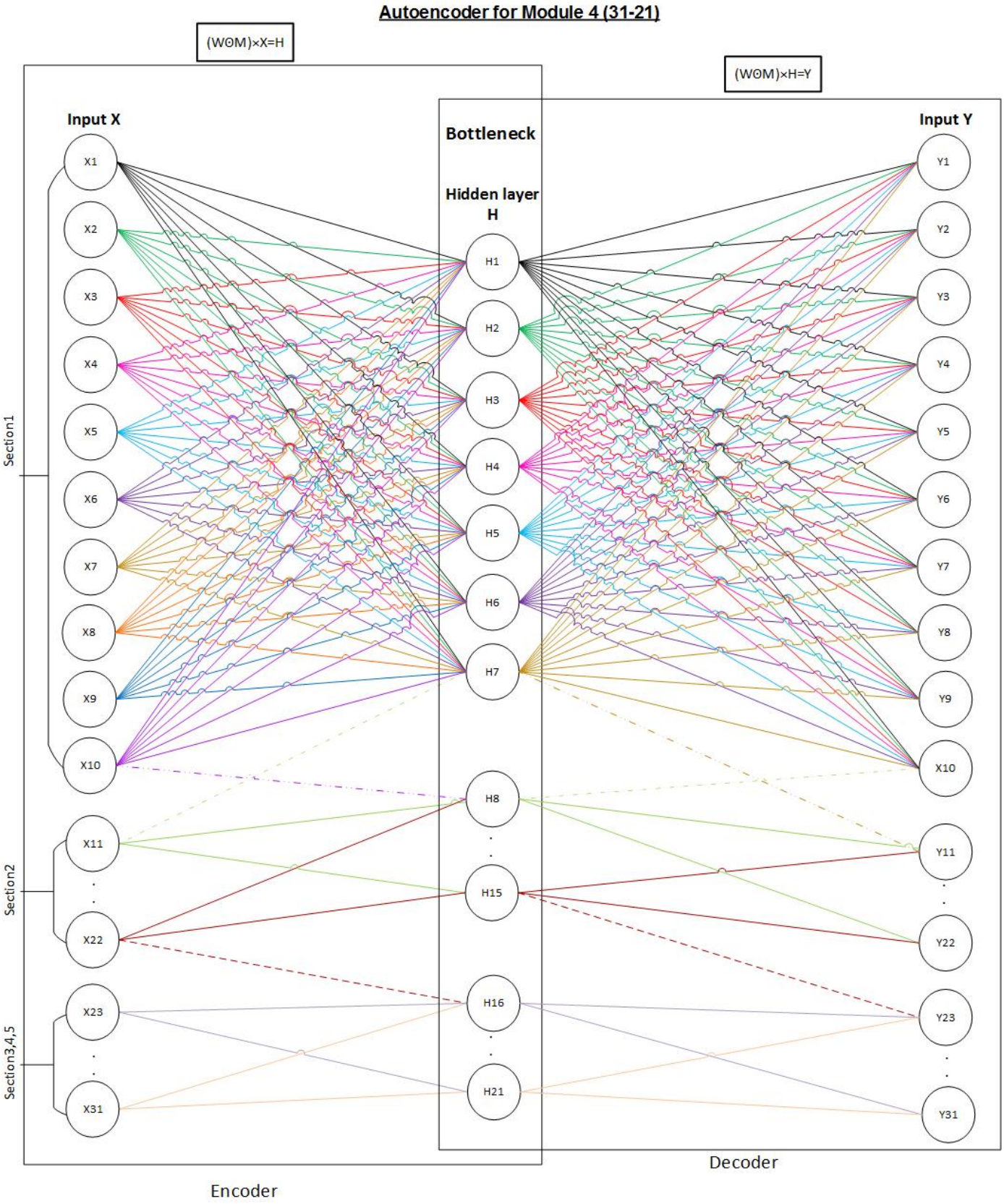

Autoencoder for Module 2 (28-22)
(WΘM)×X=H
(WΘM)×H=Y
Input X
Bottleneck
Hidden layer
H
Input Y
X1
X2
X3
X4
X5
X6
X7
H1
H2
H3
H4
H5
Y1
Y2
Y3
Y4
Y5
Y6
Y7
Section1
X8
X19
H6
H15
Y8
Y19
Section2
X20
X28
H16
H22
Y20
Y28
Section3,4,5
Encoder
Decoder

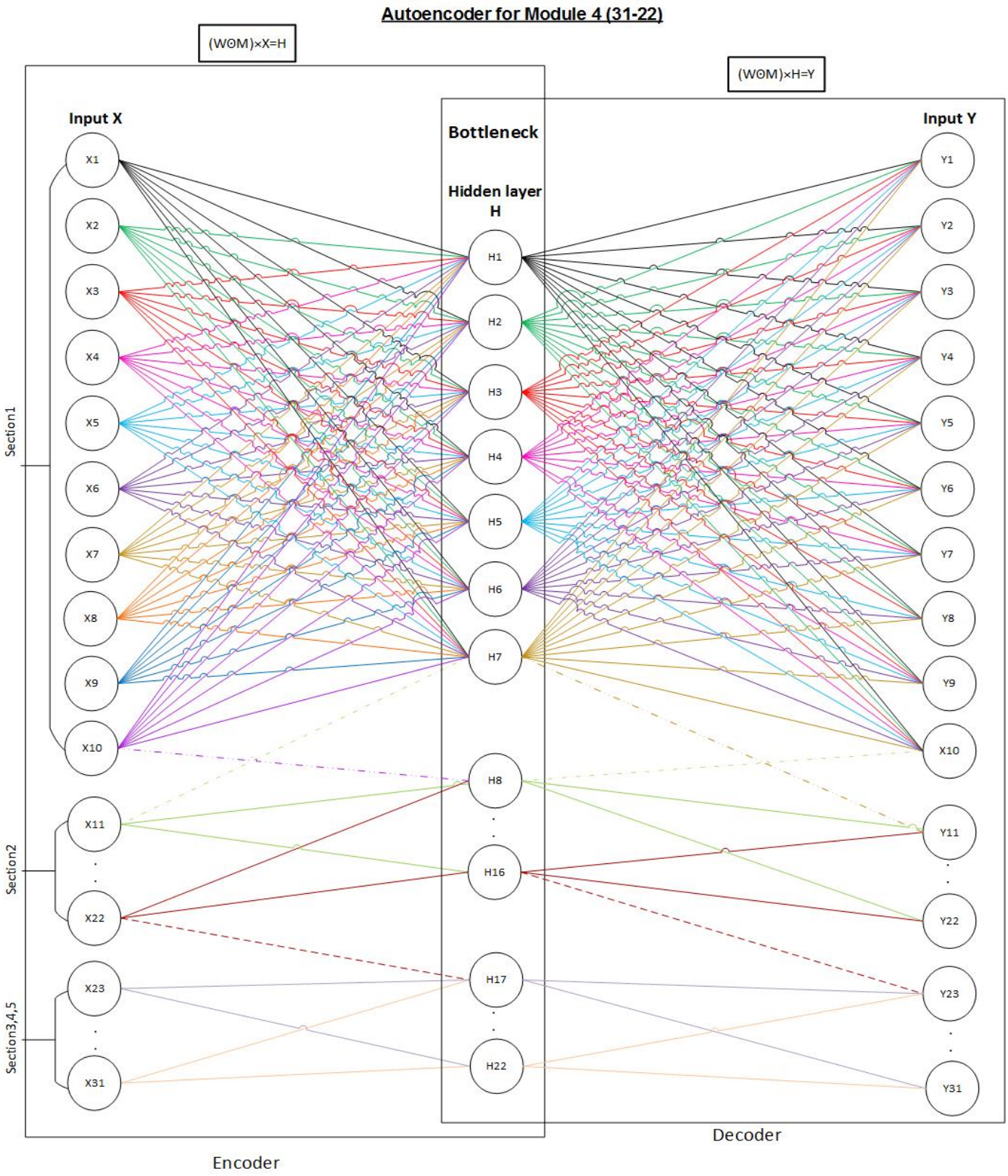

Autoencoder for Module 3 (28-20)
(W⊙M)×X=H
(W⊙M)×H=Y
Input X
Bottleneck
Hidden layer
H
Input Y
X1
X2
X3
X4
X5
X6
X7
X8
X9
X10
X19
X20
X28
H1
H2
H3
H4
H5
H6
H7
H13
H14
H20
Y1
Y2
Y3
Y4
Y5
Y6
Y7
Y8
Y9
Y10
Y19
Y20
Y28
Section1
Section2
Section3,4,5
Encoder
Decoder


Autoencoder for Module 3 (28-21)
(W⊙M)×X=H
(W⊙M)×H=Y
Input X
Bottleneck
Hidden layer
H
Input Y
X1
X2
X3
X4
X5
X6
X7
X8
X9
X10
X19
X20
X28
H1
H2
H3
H4
H5
H6
H7
H8
H15
H16
H21
Y1
Y2
Y3
Y4
Y5
Y6
Y7
Y8
Y9
Y10
Y19
Y20
Y28
Section1
Section2
Section3,4,5
Encoder
Decoder


Autoencoder for Module 3 (28-22)
(W0M)×X=H
(W0M)×H=Y
Input X
Bottleneck
Hidden layer
H
Input Y
X1
X2
X3
X4
X5
X6
X7
X8
X9
X10
X19
X20
X28
H1
H2
H3
H4
H5
H6
H7
H8
H15
H16
H22
Y1
Y2
Y3
Y4
Y5
Y6
Y7
Y8
Y9
Y10
Y19
Y20
Y28
Section1
Section2
Section3,4,5
Encoder
Decoder


Autoencoder for Module 4 (31-20)
(W⊙M)×X=H
(W⊙M)×H=Y
Input X
Bottleneck
Hidden layer
H
Input Y
X1
X2
X3
X4
X5
X6
X7
X8
X9
X10
X11
X22
X23
X31
H1
H2
H3
H4
H5
H6
H7
H13
H14
H20
Y1
Y2
Y3
Y4
Y5
Y6
Y7
Y8
Y9
X10
Y11
Y22
Y23
Y31
Section1
Section2
Section3,4,5
Encoder
Decoder


Autoencoder for Module 4 (31-21)
(WΘM)×X=H
(WΘM)×H=Y
Input X
Bottleneck
Hidden layer
H
Input Y
X1
X2
X3
X4
X5
X6
X7
X8
X9
X10
X11
X22
X23
X31
H1
H2
H3
H4
H5
H6
H7
H8
H15
H16
H21
Y1
Y2
Y3
Y4
Y5
Y6
Y7
Y8
Y9
X10
Y11
Y22
Y23
Y31
Section1
Section2
Section3,4,5
Encoder
Decoder

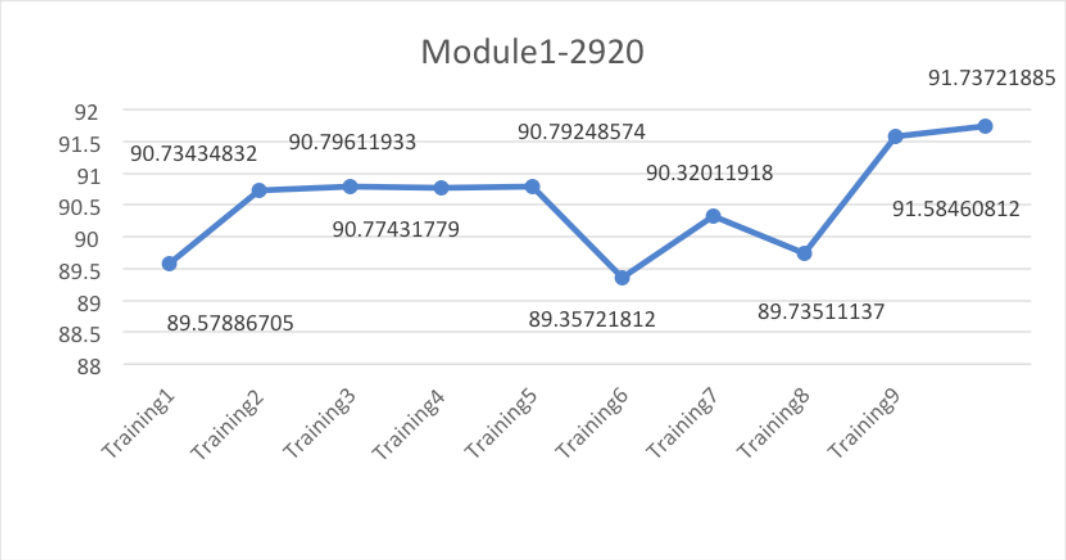

Autoencoder for Module 4 (31-22)
(WΘM)×X=H
(WΘM)×H=Y
Input X
Bottleneck
Hidden layer H
Input Y
X1
X2
X3
X4
X5
X6
X7
X8
X9
X10
X11
X22
X23
X31
H1
H2
H3
H4
H5
H6
H7
H8
H16
H17
H22
Y1
Y2
Y3
Y4
Y5
Y6
Y7
Y8
Y9
X10
Y11
Y22
Y23
Y31
Section1
Section2
Section3,4,5
Encoder
Decoder

APPENDIX D

RECONSTRUCTION ACCURACIES

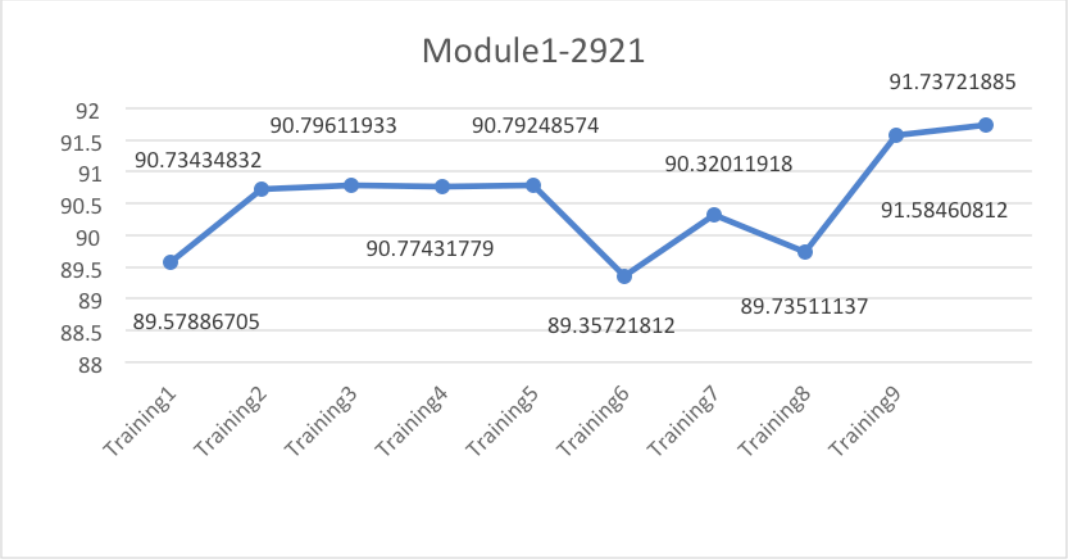
Module 1 with Hidden Layer of Size 20 Training Iterations and their Respective Reconstruction Accuracies

Module 1-2920 total cell 27521				
Training #	Wrong cell found	Percentage of error	Correct cell found	Percentage of correct
Training 1	3161	11.4857745	24360	88.5142255
Training 2	2756	10.014171	24765	89.985829
Training 3	2753	10.00327023	24768	89.99672977
Training 4	2760	10.02870535	24761	89.97129465
Training 5	3295	11.97267541	24226	88.02732459
Training 6	2755	10.01053741	24766	89.98946259
Training 7	2795	10.15588096	24726	89.84411904
Training 8	2810	10.2103848	24711	89.7896152
Training 9	3423	12.43777479	24098	87.56222521
Training 10	3023	10.98433923	24498	89.01566077
Total	29531	107.3035137	245679	892.6964863
Average	2953.1	10.73035137	24567.9	89.26964863



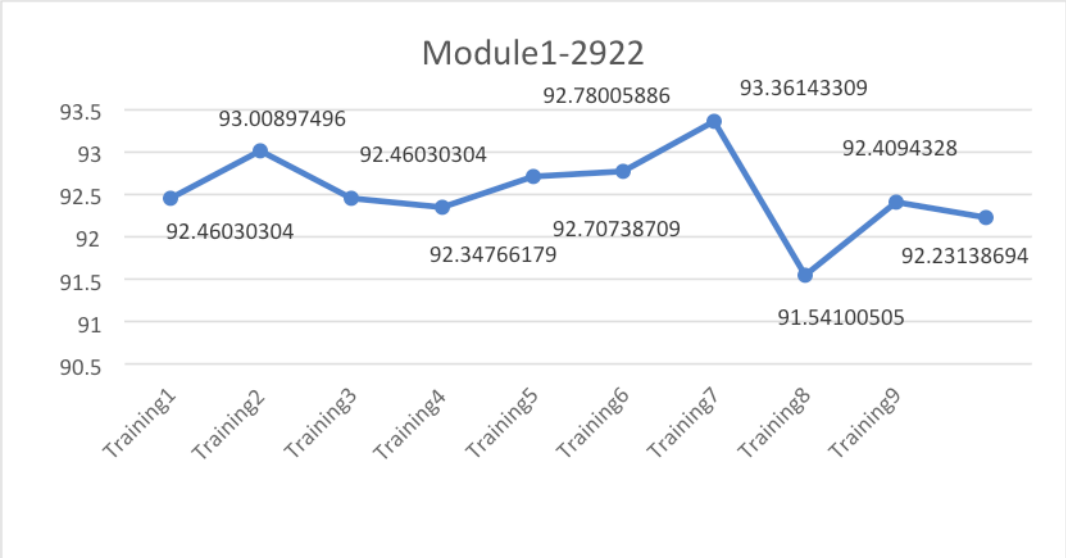
Module 1 with Hidden Layer of Size 21 Training Iterations and their Respective Reconstruction Accuracies

Module 1-2921 total cell 27521				
Training #	Wrong cell found	Percentage of error	Correct cell found	Percentage of correct
Training 1	2868	10.42113295	24653	89.57886705
Training 2	2550	9.265651684	24971	90.73434832
Training 3	2533	9.203880673	24988	90.79611933
Training 4	2539	9.225682206	24982	90.77431779
Training 5	2534	9.207514262	24987	90.79248574
Training 6	2929	10.64278188	24592	89.35721812
Training 7	2664	9.679880818	24857	90.32011918
Training 8	2825	10.26488863	24696	89.73511137
Training 9	2316	8.415391883	25205	91.58460812
Training 10	2274	8.262781149	25247	91.73721885
Total	26032	94.58958613	249178	905.4104139
Average	2603.2	9.458958613	24917.8	90.54104139



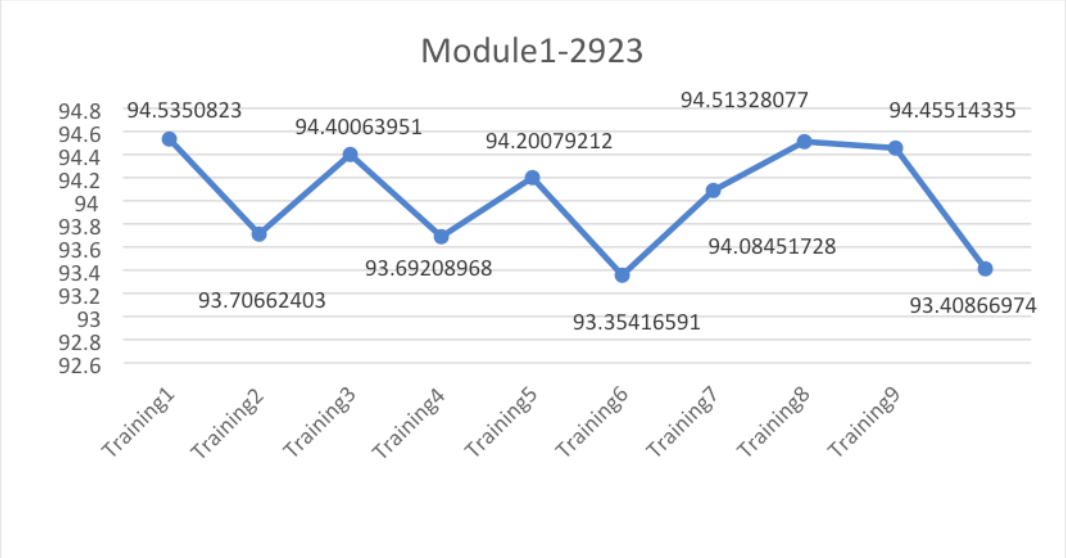
Module 1 with Hidden Layer of Size 22 Training Iterations and their Respective Reconstruction Accuracies

Module 1-2922 total cell 27521				
Training #	Wrong cell found	Percentage of error	Correct cell found	Percentage of correct
Training 1	2075	7.539696959	25446	92.46030304
Training 2	1924	6.991025035	25597	93.00897496
Training 3	2075	7.539696959	25446	92.46030304
Training 4	2106	7.652338214	25415	92.34766179
Training 5	2007	7.292612914	25514	92.70738709
Training 6	1987	7.219941136	25534	92.78005886
Training 7	1827	6.638566913	25694	93.36143309
Training 8	2328	8.458994949	25193	91.54100505
Training 9	2089	7.590567203	25432	92.4094328
Training 10	2138	7.768613059	25383	92.23138694
Total	20556	74.69205334	254654	925.3079467
Average	2055.6	7.469205334	25465.4	92.53079467



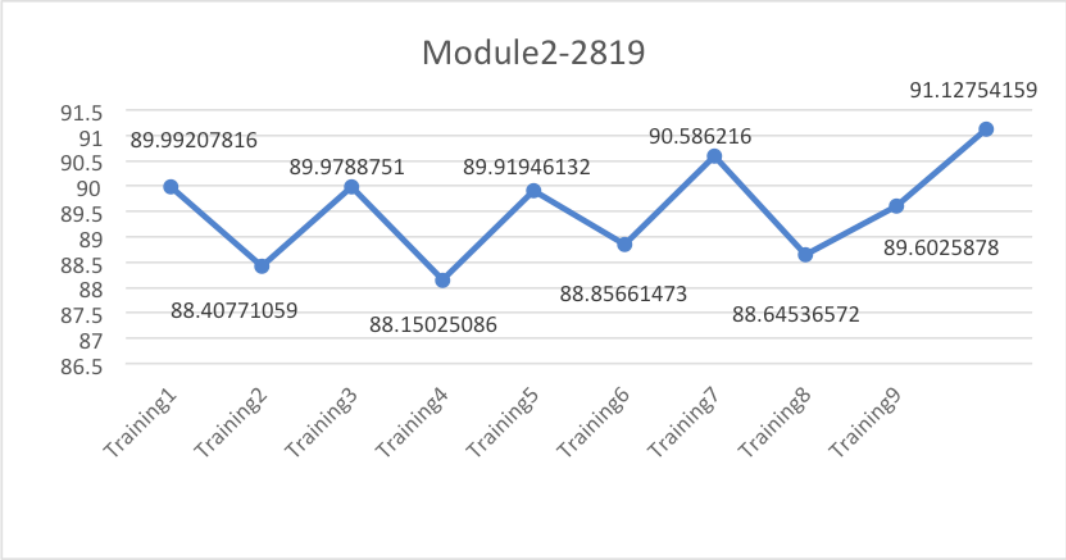
Module 1 with Hidden Layer of Size 23 Training Iterations and their Respective Reconstruction Accuracies

Module 1-2923 total cell 27521				
Training #	Wrong cell found	Percentage of error	Correct cell found	Percentage of correct
Training 1	1504	5.464917699	26017	94.5350823
Training 2	1732	6.293375967	25789	93.70662403
Training 3	1541	5.599360488	25980	94.40063951
Training 4	1736	6.307910323	25785	93.69208968
Training 5	1596	5.799207878	25925	94.20079212
Training 6	1829	6.64583409	25692	93.35416591
Training 7	1628	5.915482722	25893	94.08451728
Training 8	1510	5.486719233	26011	94.51328077
Training 9	1526	5.544856655	25995	94.45514335
Training 10	1814	6.591330257	25707	93.40866974
Total	16416	59.64899531	258794	940.3510047
Average	1641.6	5.964899531	25879.4	94.03510047



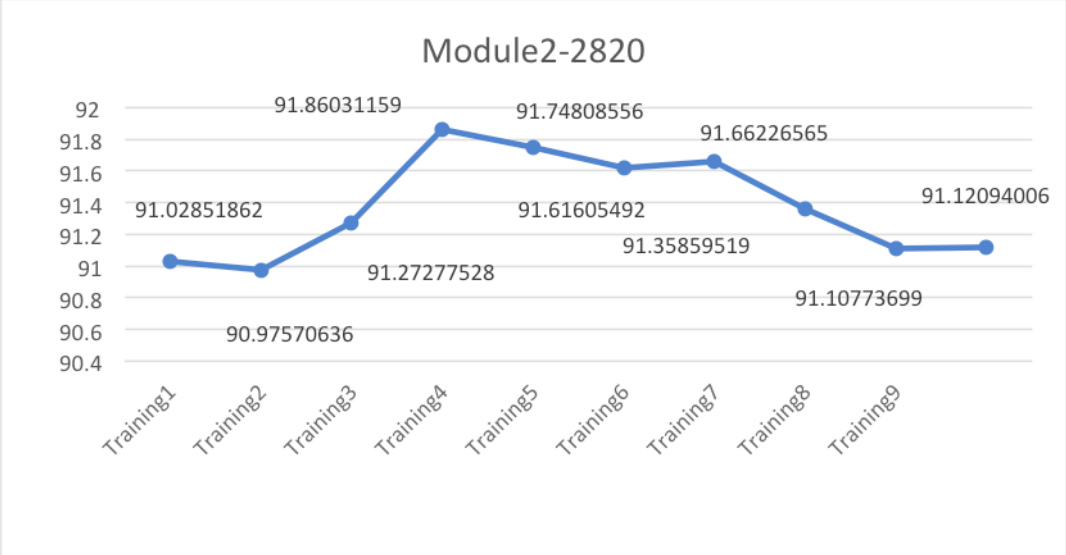
Module 2 with Hidden Layer of Size 19 Training Iterations and their Respective Reconstruction Accuracies

Module 2-2819 total cell 15148				
Training #	Wrong cell found	Percentage of error	Correct cell found	Percentage of correct
Training 1	1516	10.00792184	13632	89.99207816
Training 2	1756	11.59228941	13392	88.40771059
Training 3	1518	10.0211249	13630	89.9788751
Training 4	1795	11.84974914	13353	88.15025086
Training 5	1527	10.08053868	13621	89.91946132
Training 6	1688	11.14338527	13460	88.85661473
Training 7	1426	9.413783998	13722	90.586216
Training 8	1720	11.35463428	13428	88.64536572
Training 9	1575	10.3974122	13573	89.6025878
Training 10	1344	8.87245841	13804	91.12754159
Total	15865	104.7332981	135615	895.2667019
Average	1586.5	10.47332981	13561.5	89.52667019



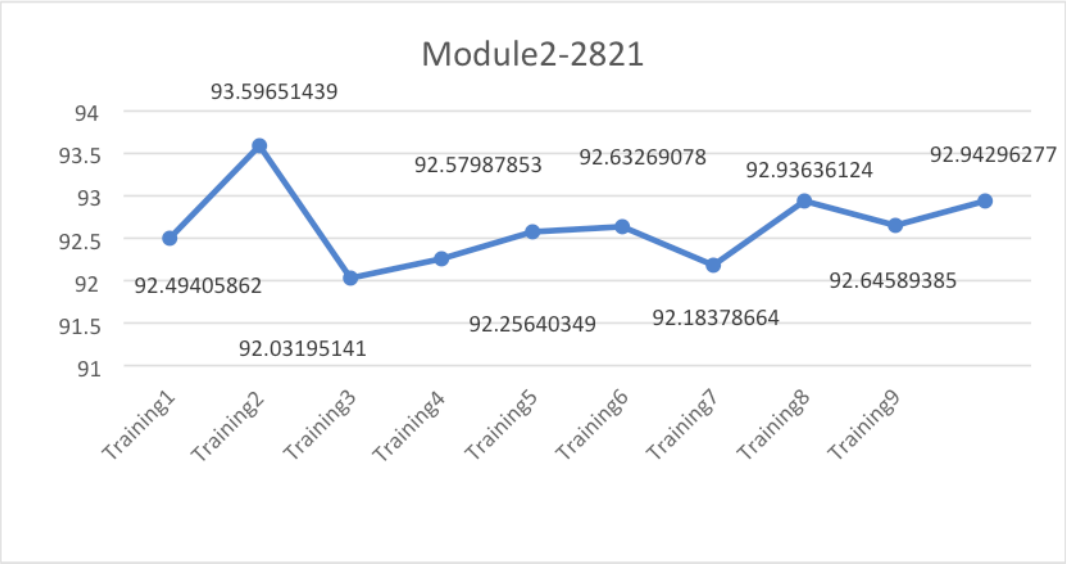
Module 2 with Hidden Layer of Size 20 Training Iterations and their Respective Reconstruction Accuracies

Module 2-2820 total cell 15148				
Training #	Wrong cell found	Percentage of error	Correct cell found	Percentage of correct
Training 1	1359	8.971481384	13789	91.02851862
Training 2	1367	9.024293636	13781	90.97570636
Training 3	1322	8.727224716	13826	91.27277528
Training 4	1233	8.139688408	13915	91.86031159
Training 5	1250	8.251914444	13898	91.74808556
Training 6	1270	8.383945075	13878	91.61605492
Training 7	1263	8.337734354	13885	91.66226565
Training 8	1309	8.641404806	13839	91.35859519
Training 9	1347	8.892263005	13801	91.10773699
Training 10	1345	8.879059942	13803	91.12094006
Total	13065	86.24900977	138415	913.7509902
Average	1306.5	8.624900977	13841.5	91.37509902



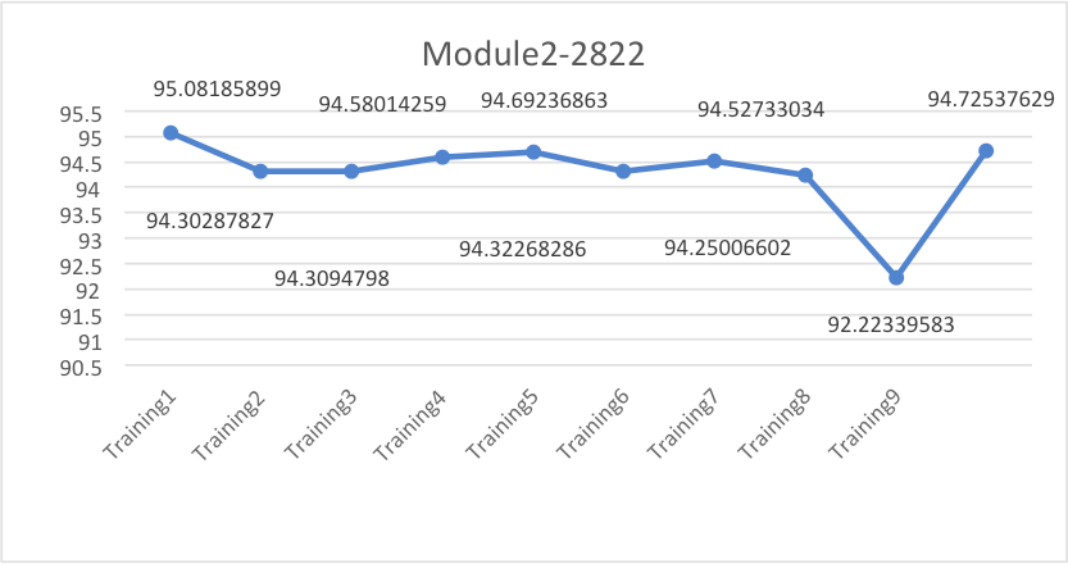
Module 2 with Hidden Layer of Size 21 Training Iterations and their Respective Reconstruction Accuracies

Module 2-2821 total cell 15148				
Training #	Wrong cell found	Percentage of error	Correct cell found	Percentage of correct
Training 1	1137	7.505941378	14011	92.49405862
Training 2	970	6.403485609	14178	93.59651439
Training 3	1207	7.968048587	13941	92.03195141
Training 4	1173	7.743596514	13975	92.25640349
Training 5	1124	7.420121468	14024	92.57987853
Training 6	1116	7.367309216	14032	92.63269078
Training 7	1184	7.816213361	13964	92.18378664
Training 8	1070	7.063638764	14078	92.93636124
Training 9	1114	7.354106153	14034	92.64589385
Training 10	1069	7.057037233	14079	92.94296277
Total	11164	73.69949828	140316	926.3005017
Average	1116.4	7.369949828	14031.6	92.63005017



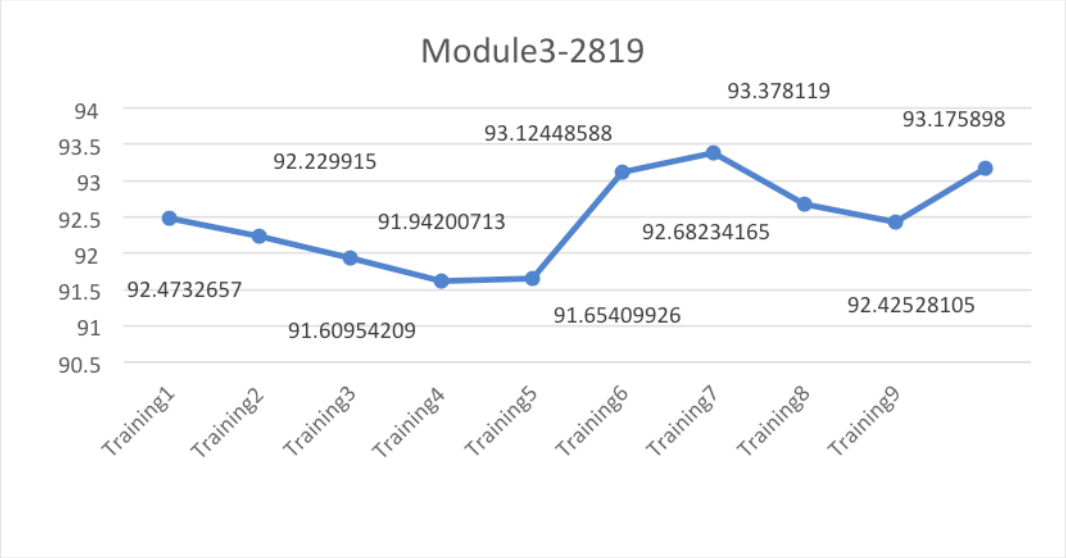
Module 2 with Hidden Layer of Size 22 Training Iterations and their Respective Reconstruction Accuracies

Module 2-2822 total cell 15148				
Training #	Wrong cell found	Percentage of error	Correct cell found	Percentage of correct
Training 1	745	4.918141009	14403	95.08185899
Training 2	863	5.697121732	14285	94.30287827
Training 3	862	5.690520201	14286	94.3094798
Training 4	821	5.419857407	14327	94.58014259
Training 5	804	5.30763137	14344	94.69236863
Training 6	860	5.677317138	14288	94.32268286
Training 7	829	5.472669659	14319	94.52733034
Training 8	871	5.749933985	14277	94.25006602
Training 9	1178	7.776604172	13970	92.22339583
Training 10	799	5.274623713	14349	94.72537629
Total	7887	52.06627938	128445	847.9337206
Average	788.7	5.206627938	12844.5	84.79337206



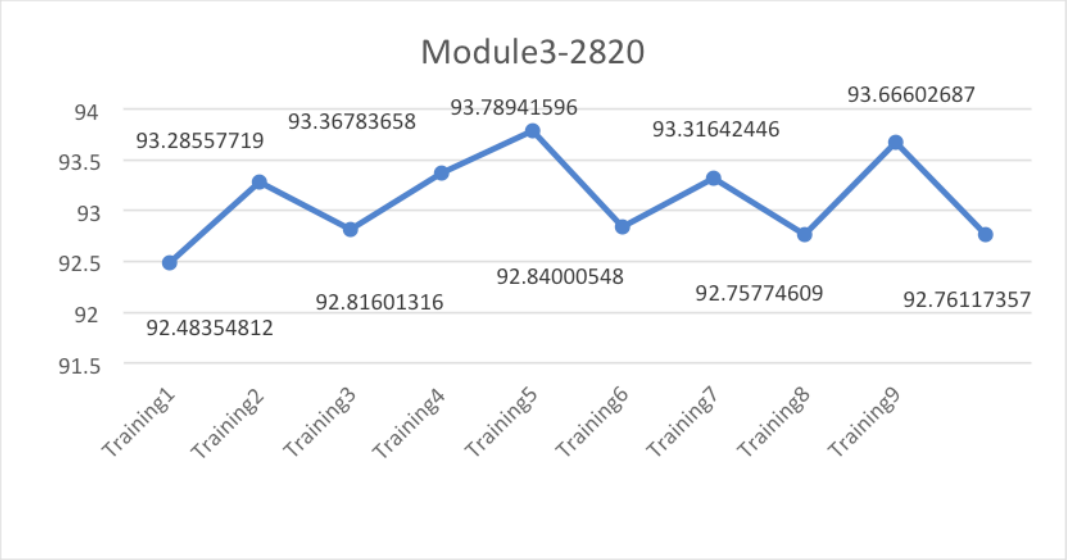
Module 3 with Hidden Layer of Size 19 Training Iterations and their Respective Reconstruction Accuracies

Module 3-2919 total cell 29176				
Training #	Wrong cell found	Percentage of error	Correct cell found	Percentage of correct
Training 1	2196	7.526734302	26980	92.4732657
Training 2	2267	7.770085001	26909	92.229915
Training 3	2351	8.057992871	26825	91.94200713
Training 4	2448	8.390457911	26728	91.60954209
Training 5	2435	8.34590074	26741	91.65409926
Training 6	2006	6.875514121	27170	93.12448588
Training 7	1932	6.621880998	27244	93.378119
Training 8	2135	7.317658349	27041	92.68234165
Training 9	2210	7.574718947	26966	92.42528105
Training 10	1991	6.824102002	27185	93.175898
Total	21971	75.30504524	269789	924.6949548
Average	2197.1	7.530504524	26978.9	92.46949548



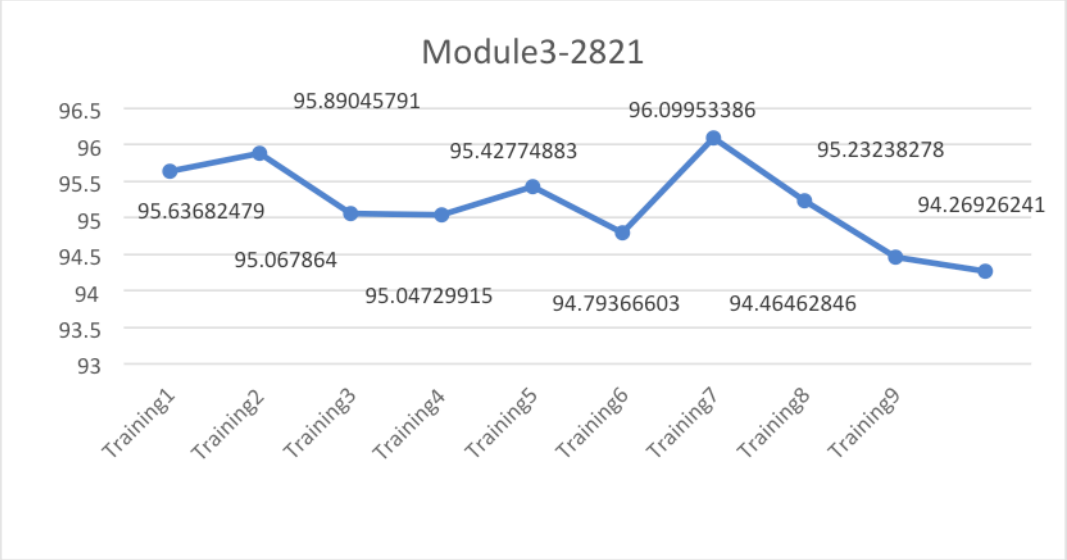
Module 3 with Hidden Layer of Size 20 Training Iterations and their Respective Reconstruction Accuracies

Module 3-2920 total cell 29176				
Training #	Wrong cell found	Percentage of error	Correct cell found	Percentage of correct
Training 1	2193	7.516451878	26983	92.48354812
Training 2	1959	6.714422813	27217	93.28557719
Training 3	2096	7.183986838	27080	92.81601316
Training 4	1935	6.632163422	27241	93.36783658
Training 5	1812	6.210584042	27364	93.78941596
Training 6	2089	7.159994516	27087	92.84000548
Training 7	1950	6.683575542	27226	93.31642446
Training 8	2113	7.242253907	27063	92.75774609
Training 9	1848	6.333973129	27328	93.66602687
Training 10	2112	7.238826433	27064	92.76117357
Total	20107	68.91623252	271653	931.0837675
Average	2010.7	6.891623252	27165.3	93.10837675



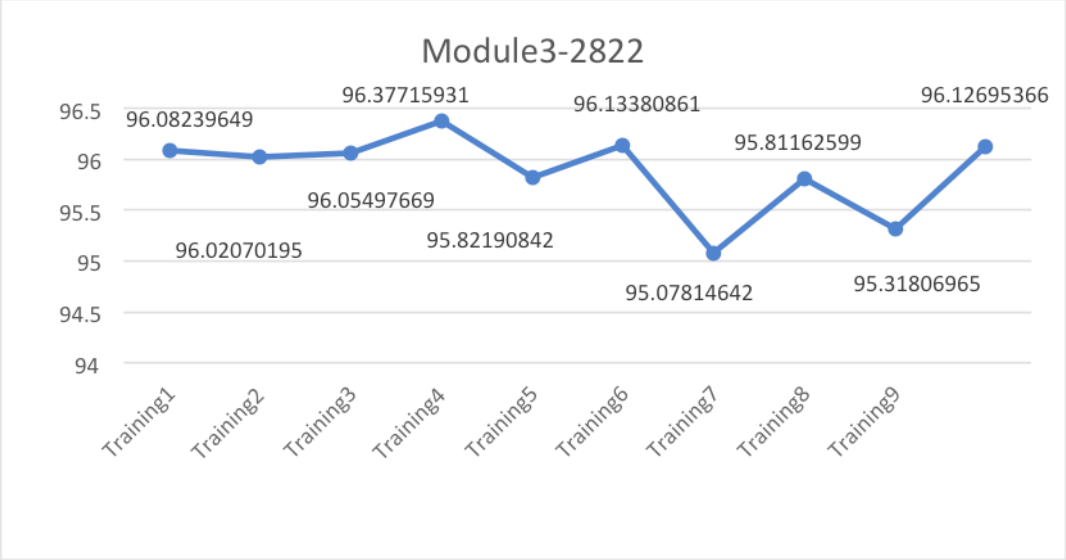
Module 3 with Hidden Layer of Size 21 Training Iterations and their Respective Reconstruction Accuracies

Module 3-2921 total cell 29176				
Training #	Wrong cell found	Percentage of error	Correct cell found	Percentage of correct
Training 1	1273	4.363175213	27903	95.63682479
Training 2	1199	4.109542089	27977	95.89045791
Training 3	1439	4.932136002	27737	95.067864
Training 4	1445	4.95270085	27731	95.04729915
Training 5	1334	4.572251165	27842	95.42774883
Training 6	1519	5.206333973	27657	94.79366603
Training 7	1138	3.900466137	28038	96.09953386
Training 8	1391	4.76761722	27785	95.23238278
Training 9	1615	5.535371538	27561	94.46462846
Training 10	1672	5.730737593	27504	94.26926241
Total	14025	48.07033178	277735	951.9296682
Average	1402.5	4.807033178	27773.5	95.19296682



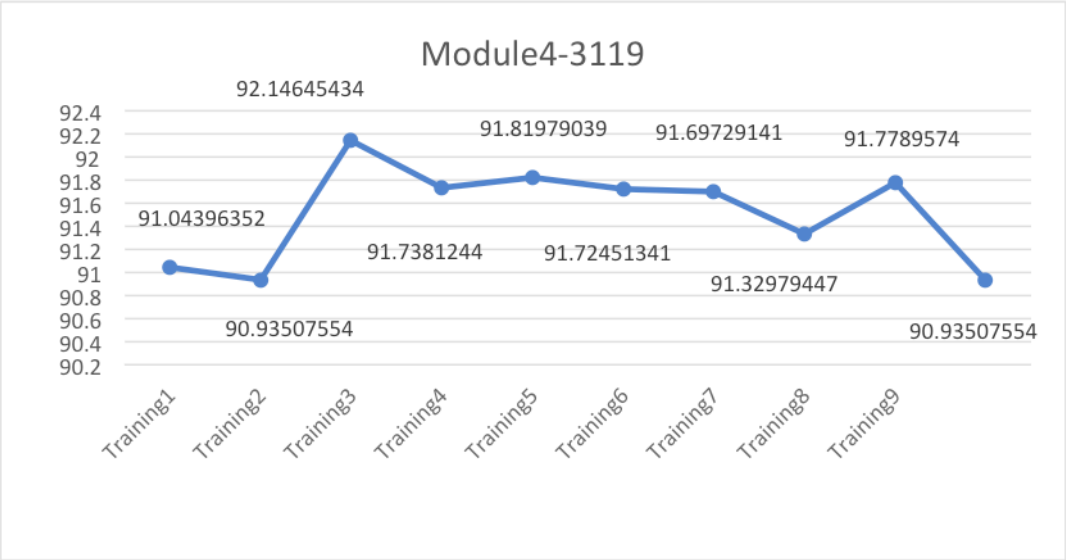
Module 3 with Hidden Layer of Size 22 Training Iterations and their Respective Reconstruction Accuracies

Module 3-2922 total cell 29176				
Training #	Wrong cell found	Percentage of error	Correct cell found	Percentage of correct
Training 1	1143	3.91760351	28033	96.08239649
Training 2	1161	3.979298053	28015	96.02070195
Training 3	1151	3.945023307	28025	96.05497669
Training 4	1057	3.622840691	28119	96.37715931
Training 5	1219	4.178091582	27957	95.82190842
Training 6	1128	3.86619139	28048	96.13380861
Training 7	1436	4.921853578	27740	95.07814642
Training 8	1222	4.188374006	27954	95.81162599
Training 9	1366	4.681930354	27810	95.31806965
Training 10	1130	3.873046339	28046	96.12695366
Total	12013	41.17425281	279747	958.8257472
Average	1201.3	4.117425281	27974.7	95.88257472



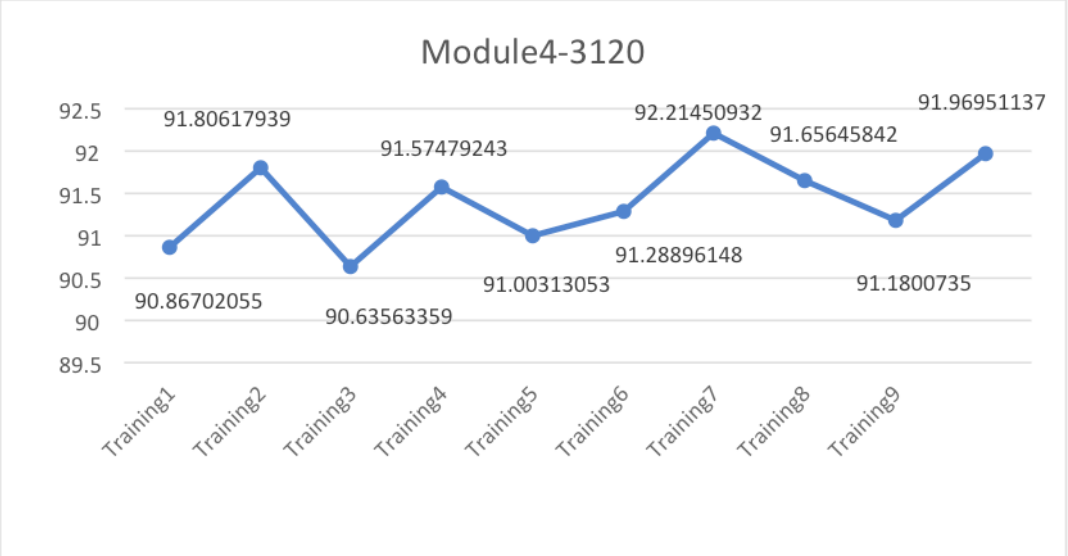
Module 4 with Hidden Layer of Size 19 Training Iterations and their Respective Reconstruction Accuracies

Module 4-3119 total cell 7347				
Training #	Wrong cell found	Percentage of error	Correct cell found	Percentage of correct
Training 1	658	8.956036477	6689	91.04396352
Training 2	666	9.064924459	6681	90.93507554
Training 3	577	7.853545665	6770	92.14645434
Training 4	607	8.261875595	6740	91.7381244
Training 5	601	8.180209609	6746	91.81979039
Training 6	608	8.275486593	6739	91.72451341
Training 7	610	8.302708589	6737	91.69729141
Training 8	637	8.670205526	6710	91.32979447
Training 9	604	8.221042602	6743	91.7789574
Training 10	666	9.064924459	6681	90.93507554
Total	6234	84.85095958	67236	915.1490404
Average	623.4	8.485095958	6723.6	91.51490404



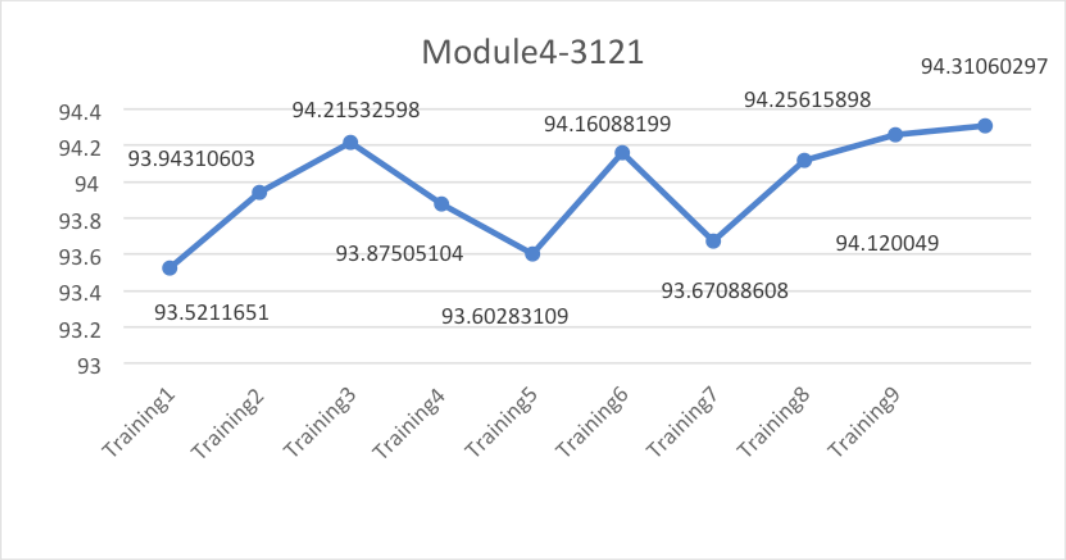
Module 4 with Hidden Layer of Size 20 Training Iterations and their Respective Reconstruction Accuracies

Module 4-3120 total cell 7347				
Training #	Wrong cell found	Percentage of error	Correct cell found	Percentage of correct
Training 1	671	9.132979447	6676	90.86702055
Training 2	602	8.193820607	6745	91.80617939
Training 3	688	9.364366408	6659	90.63563359
Training 4	619	8.425207568	6728	91.57479243
Training 5	661	8.996869471	6686	91.00313053
Training 6	640	8.711038519	6707	91.28896148
Training 7	572	7.785490676	6775	92.21450932
Training 8	613	8.343541582	6734	91.65645842
Training 9	648	8.819926501	6699	91.1800735
Training 10	590	8.030488635	6757	91.96951137
Total	6304	85.80372941	67166	914.1962706
Average	630.4	8.580372941	6716.6	91.41962706



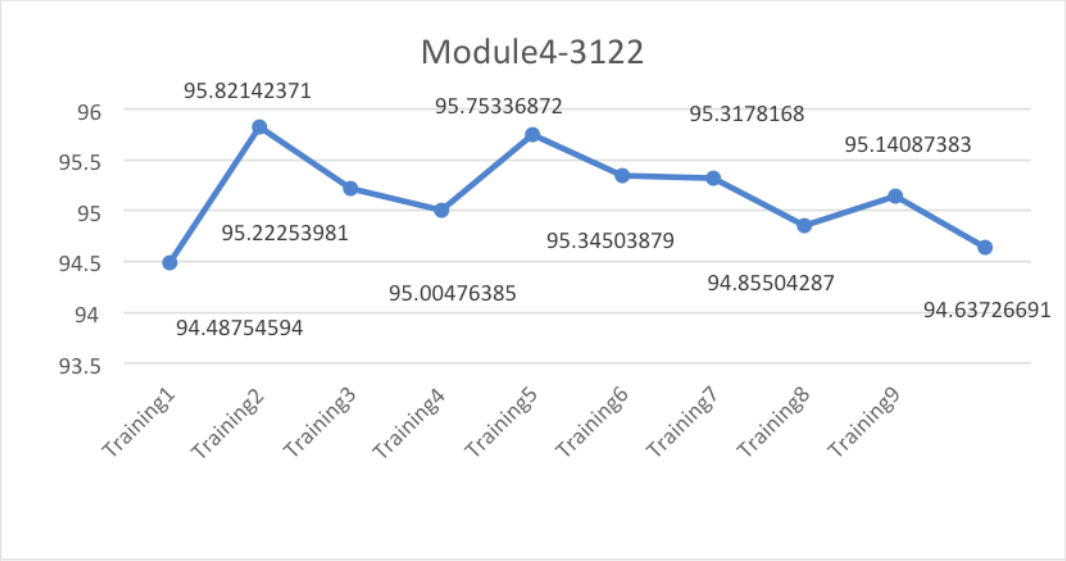
Module 4 with Hidden Layer of Size 21 Training Iterations and their Respective Reconstruction Accuracies

Module 4-3121 total cell 7347				
Training #	Wrong cell found	Percentage of error	Correct cell found	Percentage of correct
Training 1	476	6.478834899	6871	93.5211651
Training 2	445	6.05689397	6902	93.94310603
Training 3	425	5.784674017	6922	94.21532598
Training 4	450	6.124948959	6897	93.87505104
Training 5	470	6.397168912	6877	93.60283109
Training 6	429	5.839118007	6918	94.16088199
Training 7	465	6.329113924	6882	93.67088608
Training 8	432	5.879951	6915	94.120049
Training 9	422	5.743841024	6925	94.25615898
Training 10	418	5.689397033	6929	94.31060297
Total	4432	60.32394174	69038	939.6760583
Average	443.2	6.032394174	6903.8	93.96760583



Module 4 with Hidden Layer of Size 22 Training Iterations and their Respective Reconstruction Accuracies

Module 4-3122 total cell 7347				
Training #	Wrong cell found	Percentage of error	Correct cell found	Percentage of correct
Training 1	405	5.512454063	6942	94.48754594
Training 2	307	4.17857629	7040	95.82142371
Training 3	351	4.777460188	6996	95.22253981
Training 4	367	4.995236151	6980	95.00476385
Training 5	312	4.246631278	7035	95.75336872
Training 6	342	4.654961209	7005	95.34503879
Training 7	344	4.682183204	7003	95.3178168
Training 8	378	5.144957125	6969	94.85504287
Training 9	357	4.859126174	6990	95.14087383
Training 10	394	5.362733088	6953	94.63726691
Total	3557	48.41431877	69913	951.5856812
Average	355.7	4.841431877	6991.3	95.15856812



APPENDIX E

MODULES WEIGHTS

Module1 29-20

												overall
OLANG	0.2479	STEREO	0.3488	UOTHR	0.2608	UOTHR	0.26	UOTHR	0.3677	UOTHR		1.2057
UOTHR	0.1876	OLANG	0.1824	INTON	0.1614	STEREO	0.1899	IECHO	0.253	OLANG		0.9042
INTON	0.1301	UOTHR	0.1296	OLANG	0.1583	OLANG	0.1827	OLANG	0.1328	IECHO		0.7815
IECHO	0.1167	IECHO	0.124	IECHO	0.1446	IECHO	0.1433	POINT	0.0837	STEREO		0.708
POINT	0.1031	INTON	0.0927	POINT	0.1162	POINT	0.0864	FVOC	0.0521	INTON		0.4531
STEREO	0.0965	POINT	0.0606	FVOC	0.0597	FVOC	0.0695	INTON	0.045	POINT		0.4501
FVOC	0.0609	FVOC	0.0406	GEST	0.0551	GEST	0.0442	GEST	0.0367	FVOC		0.2829
GEST	0.0571	GEST	0.0213	STEREO	0.0438	INTON	0.0239	STEREO	0.029	GEST		0.2144

Module1 29-21

												overall
INTON	0.1992	UOTHR	0.3105	STEREO	0.1607	OLANG	0.1893	UOTHR	0.3144	UOTHR		1.1095
OLANG	0.1661	POINT	0.1545	UOTHR	0.1575	UOTHR	0.1693	OLANG	0.1445	OLANG		0.7942
IECHO	0.1661	OLANG	0.1493	OLANG	0.145	IECHO	0.1509	POINT	0.1374	INTON		0.6829
UOTHR	0.1576	INTON	0.1384	INTON	0.1406	STEREO	0.1362	STEREO	0.1184	POINT		0.657
STEREO	0.1331	STEREO	0.0896	POINT	0.1327	POINT	0.1196	INTON	0.0989	STEREO		0.6381
POINT	0.1129	FVOC	0.0599	IECHO	0.131	INTON	0.106	IECHO	0.0793	IECHO		0.5857
FVOC	0.0385	IECHO	0.0585	FVOC	0.0895	FVOC	0.0986	FVOC	0.0663	FVOC		0.3529
GEST	0.0266	GEST	0.0393	GEST	0.0431	GEST	0.0301	GEST	0.0408	GEST		0.1798

Module1 29-22

												overall
STEREO	0.2638	INTON	0.2089	STEREO	0.2111	OLANG	0.2234	UOTHR	0.2211	UOTHR	0.2256	UOTHR 1.1017
IECHO	0.1674	UOTHR	0.2022	UOTHR	0.201	POINT	0.1761	OLANG	0.218	IECHO	0.181	OLANG 1.0473
UOTHR	0.1297	IECHO	0.1769	OLANG	0.1965	IECHO	0.1476	IECHO	0.1526	OLANG	0.1636	IECHO 0.9327
OLANG	0.1191	OLANG	0.1266	POINT	0.1247	INTON	0.1296	STEREO	0.1457	INTON	0.1573	STEREO 0.8645
INTON	0.119	POINT	0.1053	IECHO	0.1073	UOTHR	0.1221	INTON	0.1117	POINT	0.0939	INTON 0.8101
POINT	0.0982	STEREO	0.099	INTON	0.0836	STEREO	0.0857	POINT	0.0785	FVOC	0.0695	POINT 0.6768
FVOC	0.0615	GEST	0.047	FVOC	0.0487	GEST	0.0651	FVOC	0.0391	STEREO	0.0592	FVOC 0.3034
GEST	0.0413	FVOC	0.0342	GEST	0.0271	FVOC	0.0505	GEST	0.0332	GEST	0.0499	GEST 0.2635

Module1 29-23

												overall
STEREO	0.2176	OLANG	0.2773	UOTHR	0.2321	IECHO	0.2134	STEREO	0.2139	OLANG	0.2026	OLANG 1.125
OLANG	0.2113	INTON	0.1996	IECHO	0.2089	STEREO	0.2092	OLANG	0.1928	IECHO	0.178	IECHO 0.9749
INTON	0.1412	UOTHR	0.1768	POINT	0.1438	OLANG	0.156	UOTHR	0.1566	UOTHR	0.155	STEREO 0.9026
IECHO	0.1266	IECHO	0.1667	INTON	0.1426	FVOC	0.1173	POINT	0.1509	INTON	0.1507	UOTHR 0.8845
POINT	0.1058	POINT	0.073	STEREO	0.0885	POINT	0.0998	FVOC	0.0936	STEREO	0.1246	INTON 0.8018
UOTHR	0.088	FVOC	0.0498	FVOC	0.0854	INTON	0.0899	IECHO	0.0812	FVOC	0.095	POINT 0.6576
FVOC	0.072	STEREO	0.0488	OLANG	0.0851	UOTHR	0.076	INTON	0.0778	POINT	0.0843	FVOC 0.5131
GEST	0.0376	GEST	0.0081	GEST	0.0137	GEST	0.0383	GEST	0.0331	GEST	0.0096	GEST 0.1404

Module1 29-20

																		overall
REQ	0.128	SHRNJ	0.1216	SHRNJ	0.1265	GZSOV	0.1448	RJNT	0.1318	RNAME	0.1256	GZSOV	0.1334	GZSOV	0.1445	RJNT	0.1119	GZSOV	0.9869
GZSOV	0.1045	SSMLE	0.1181	SSMLE	0.1096	GIVE	0.1387	SSMLE	0.1194	SHRNJ	0.1206	SSMLE	0.1126	SHRNJ	0.1249	GZSOV	0.1095	SSMLE	0.9465
UEYE	0.1	RJNT	0.1164	REQ	0.1081	UEYE	0.1014	GIVE	0.1137	GZSOV	0.1165	REQ	0.1117	SSMLE	0.1025	SHRNJ	0.0999	SHRNJ	0.9457
SSMLE	0.0958	QSOV	0.1063	RNAME	0.1009	SHOW	0.0917	RNAME	0.1073	REQ	0.1076	SHRNJ	0.103	RNAME	0.1005	REQ	0.0999	RJNT	0.8943
SHOW	0.0932	GIVE	0.0873	GIVE	0.0951	SSMLE	0.0899	SHRNJ	0.0964	SSMLE	0.1073	RNAME	0.1003	RJNT	0.0973	RNAME	0.0946	REQ	0.8784
RJNT	0.0902	RNAME	0.0791	GZSOV	0.0929	RJNT	0.0836	REQ	0.0957	GIVE	0.0903	RJNT	0.0965	REQ	0.0862	SSMLE	0.0911	RNAME	0.8686
GIVE	0.0883	REQ	0.0756	RJNT	0.0776	RNAME	0.0789	GZSOV	0.0794	RJNT	0.0888	UEYE	0.0953	UEYE	0.0839	GIVE	0.0911	GIVE	0.8331
RNAME	0.0813	UEYE	0.0724	UEYE	0.0766	SHRNJ	0.0789	UEYE	0.0678	UEYE	0.0865	GIVE	0.068	SHOW	0.0615	SHOW	0.0907	UEYE	0.7634
SHRNJ	0.074	SHOW	0.0691	SHOW	0.0574	REQ	0.0657	SHOW	0.0598	SHOW	0.0569	SHOW	0.065	GIVE	0.0606	UEYE	0.0794	SHOW	0.6453
FACEO	0.0535	FACEO	0.0623	QSOV	0.0542	SIJNT	0.0574	SIJNT	0.0585	FACEO	0.0444	QSOV	0.0513	QSOV	0.0585	SIJNT	0.0595	QSOV	0.4584
QSOV	0.0485	GZSOV	0.0615	FACEO	0.0514	QSOV	0.043	QSOV	0.0399	QSOV	0.0361	FACEO	0.032	FACEO	0.0472	FACEO	0.0519	FACEO	0.3989
SIJNT	0.0427	SIJNT	0.0302	SIJNT	0.0497	FACEO	0.0258	FACEO	0.0302	SIJNT	0.0193	SIJNT	0.031	SIJNT	0.0323	QSOV	0.0206	SIJNT	0.3805

Module1 29-21

																		overall
FACEO	0.1541	GIVE	0.1284	RJNT	0.1323	GZSOV	0.1532	GIVE	0.1389	SSMLE	0.1263	GIVE	0.1303	SHRNJ	0.1415	GIVE	0.1248	GIVE	0.9637
SSMLE	0.112	RNAME	0.1196	SHRNJ	0.1206	SHRNJ	0.1312	RJNT	0.1214	SHRNJ	0.1171	SSMLE	0.1293	REQ	0.1225	SSMLE	0.1227	SSMLE	0.9504
REQ	0.1079	SSMLE	0.1052	GIVE	0.1204	REQ	0.1181	GZSOV	0.1034	GZSOV	0.1079	SHRNJ	0.0992	SSMLE	0.0987	REQ	0.1061	RJNT	0.9473
RJNT	0.1051	RJNT	0.1009	REQ	0.1189	GIVE	0.1163	SHRNJ	0.0974	RJNT	0.106	UEYE	0.0956	RJNT	0.0963	SHRNJ	0.1044	SHRNJ	0.9454
UEYE	0.1025	GZSOV	0.0873	RNAME	0.0871	RJNT	0.1073	SSMLE	0.0941	QSOV	0.1002	REQ	0.0921	UEYE	0.0895	RJNT	0.0949	REQ	0.899
RNAME	0.0876	SHRNJ	0.084	UEYE	0.0842	SSMLE	0.1026	REQ	0.0922	RNAME	0.0875	RJNT	0.0831	GZSOV	0.0862	RNAME	0.0941	GZSOV	0.8307
SHOW	0.0717	REQ	0.0773	SHOW	0.0723	RNAME	0.0811	RNAME	0.0912	GIVE	0.0861	RNAME	0.0787	RNAME	0.0808	UEYE	0.0901	RNAME	0.8077
GZSOV	0.0712	UEYE	0.0735	GZSOV	0.0698	UEYE	0.0553	UEYE	0.0657	UEYE	0.0741	SHOW	0.0747	SHOW	0.0779	GZSOV	0.0774	UEYE	0.7306
GIVE	0.0513	QSOV	0.0712	SSMLE	0.0595	FACEO	0.0445	FACEO	0.0634	REQ	0.064	GZSOV	0.0743	GIVE	0.0673	SHOW	0.0638	SHOW	0.5466
SHRNJ	0.05	SHOW	0.0536	QSOV	0.0561	QSOV	0.0336	QSOV	0.0595	SHOW	0.0606	QSOV	0.0668	FACEO	0.0517	QSOV	0.0551	QSOV	0.5419
QSOV	0.05	FACEO	0.0511	SIJNT	0.0497	SHOW	0.0326	SHOW	0.0394	SIJNT	0.0351	SIJNT	0.0488	QSOV	0.0493	FACEO	0.0429	FACEO	0.4986
SIJNT	0.0366	SIJNT	0.048	FACEO	0.029	SIJNT	0.0243	SIJNT	0.0335	FACEO	0.0349	FACEO	0.0271	SIJNT	0.0382	SIJNT	0.0238	SIJNT	0.3381

Module1 29-22

																			overall
UEYE	0.1277	SHRNJ	0.1227	SSMLE	0.1688	GZSOV	0.1325	GZSOV	0.1054	GZSOV	0.1188	RNAME	0.1493	UEYE	0.1369	SSMLE	0.124	GZSOV	0.147	GZSOV	1.0178
GIVE	0.1149	GZSOV	0.1226	REQ	0.1211	GIVE	0.1037	REQ	0.1034	UEYE	0.1067	RJNT	0.112	REQ	0.133	GIVE	0.1164	SHRNJ	0.1048	SHRNJ	0.9687
QSOV	0.0948	RJNT	0.1106	RJNT	0.113	UEYE	0.0988	QSOV	0.0976	RJNT	0.0997	SHRNJ	0.1053	RNAME	0.0971	GZSOV	0.1083	RJNT	0.0954	UEYE	0.9669
RJNT	0.0928	RNAME	0.1094	SHRNJ	0.1052	QSOV	0.0962	UEYE	0.0961	SSMLE	0.0943	SSMLE	0.1018	SSMLE	0.0969	SHRNJ	0.1063	SSMLE	0.0922	SSMLE	0.9581
REQ	0.0925	REQ	0.1022	GZSOV	0.0904	SHRNJ	0.0934	GIVE	0.0947	RNAME	0.0917	UEYE	0.0865	SHOW	0.0916	FACEO	0.0963	GIVE	0.0879	RJNT	0.9398
SHRNJ	0.0891	UEYE	0.0825	RNAME	0.0765	SIJNT	0.0787	RJNT	0.0922	QSOV	0.0897	SIJNT	0.081	SHRNJ	0.0829	UEYE	0.0868	UEYE	0.0806	REQ	0.897
SSMLE	0.0827	QSOV	0.0722	FACEO	0.0673	RNAME	0.0778	SHRNJ	0.0847	REQ	0.0792	GIVE	0.0712	RJNT	0.0786	RJNT	0.0857	FACEO	0.0769	RNAME	0.8887
RNAME	0.0811	SSMLE	0.071	UEYE	0.0644	FACEO	0.0756	RNAME	0.0841	SHOW	0.0788	REQ	0.0702	SIJNT	0.0735	REQ	0.0699	QSOV	0.0753	GIVE	0.8228
GZSOV	0.0733	GIVE	0.0652	SIJNT	0.0624	REQ	0.069	FACEO	0.0687	SHRNJ	0.0742	FACEO	0.07	GIVE	0.0601	SIJNT	0.0602	RNAME	0.0691	QSOV	0.7304
SIJNT	0.0717	SIJNT	0.0563	GIVE	0.0542	SSMLE	0.0672	SIJNT	0.0612	SIJNT	0.0616	GZSOV	0.0681	QSOV	0.0572	RNAME	0.0526	SIJNT	0.067	SIJNT	0.6737
SHOW	0.0523	FACEO	0.044	QSOV	0.0476	RJNT	0.0596	SSMLE	0.0591	GIVE	0.0545	QSOV	0.0529	GZSOV	0.0514	QSOV	0.0469	REQ	0.0565	FACEO	0.6174
FACEO	0.0271	SHOW	0.0413	SHOW	0.0291	SHOW	0.0474	SHOW	0.0527	FACEO	0.0508	SHOW	0.0317	FACEO	0.0408	SHOW	0.0467	SHOW	0.0472	SHOW	0.5187

Module1 29-23

																			overall
REQ	0.1218	GIVE	0.1327	GZSOV	0.1126	GIVE	0.1107	SHRNJ	0.1385	RJNT	0.1247	FACEO	0.1239	SIJNT	0.1178	REQ	0.122	SHRNJ	0.1248	GIVE	1.039
RNAME	0.1204	REQ	0.1051	REQ	0.1125	SHRNJ	0.098	RJNT	0.119	SSMLE	0.118	UEYE	0.1184	SHRNJ	0.1082	RJNT	0.1178	GIVE	0.1166	REQ	0.9754
SSMLE	0.1096	RJNT	0.105	GIVE	0.1048	RNAME	0.0933	GIVE	0.1175	GZSOV	0.1018	QSOV	0.0971	RJNT	0.1019	SHRNJ	0.1154	UEYE	0.108	SHRNJ	0.9701
QSOV	0.1063	RNAME	0.101	UEYE	0.0978	SSMLE	0.0916	REQ	0.1103	SHRNJ	0.0986	GIVE	0.0934	GIVE	0.0998	GIVE	0.1007	REQ	0.0967	UEYE	0.9112
UEYE	0.0926	UEYE	0.0996	RJNT	0.0871	FACEO	0.0903	SSMLE	0.1076	REQ	0.0894	GZSOV	0.092	SSMLE	0.0911	UEYE	0.0895	SIJNT	0.0896	RJNT	0.898
GIVE	0.0845	GZSOV	0.0954	SHRNJ	0.0865	GZSOV	0.0854	GZSOV	0.1042	SIJNT	0.0831	RNAME	0.0846	RNAME	0.0866	RNAME	0.0879	QSOV	0.0822	GZSOV	0.8413
SIJNT	0.0792	QSOV	0.0693	SIJNT	0.0822	UEYE	0.0844	UEYE	0.0561	UEYE	0.0794	SHRNJ	0.0815	UEYE	0.0853	SHOW	0.0792	SSMLE	0.0809	RNAME	0.8347
SHOW	0.0751	SIJNT	0.0625	FACEO	0.0743	QSOV	0.0741	QSOV	0.0537	GIVE	0.0783	SIJNT	0.0762	REQ	0.0815	GZSOV	0.0675	RNAME	0.0751	SSMLE	0.8211
RJNT	0.0625	SHOW	0.0621	SSMLE	0.0682	REQ	0.0733	FACEO	0.0516	RNAME	0.0765	SHOW	0.0636	FACEO	0.0749	SIJNT	0.0595	RJNT	0.072	SIJNT	0.7697
SHRNJ	0.058	SHRNJ	0.0604	RNAME	0.0618	SHOW	0.0732	SIJNT	0.0516	FACEO	0.0617	REQ	0.063	GZSOV	0.073	FACEO	0.0562	SHOW	0.0712	QSOV	0.6797
FACEO	0.0477	FACEO	0.058	QSOV	0.0593	SIJNT	0.068	RNAME	0.0475	QSOV	0.0455	SSMLE	0.0559	SHOW	0.0428	QSOV	0.055	GZSOV	0.0671	FACEO	0.6543
GZSOV	0.0422	SSMLE	0.0488	SHOW	0.0528	RJNT	0.0578	SHOW	0.0424	SHOW	0.0431	RJNT	0.0502	QSOV	0.0372	SSMLE	0.0493	FACEO	0.0158	SHOW	0.6055

Module1 29-20

																overall
URBEH	0.1887	AGG	0.1828	IMGCR	0.1596	URBEH	0.1915	AGG	0.1912	AGG	0.2258	AGG	1.1101			
AGG	0.1844	URBEH	0.1697	AGG	0.1593	IMGCR	0.1804	URBEH	0.1732	IMGCR	0.2011	URBEH	0.9743			
IMGCR	0.1658	OMAN	0.167	URBEH	0.1567	AGG	0.1667	ACTVE	0.1488	FPLAY	0.1775	IMGCR	0.9487			
ACTVE	0.1393	ACTVE	0.132	FPLAY	0.1562	ACTVE	0.1505	FPLAY	0.1459	ACTVE	0.1215	FPLAY	0.8307			
FPLAY	0.1379	USENS	0.1248	OMAN	0.1461	OMAN	0.1448	IMGCR	0.1232	OMAN	0.1087	ACTVE	0.825			
OMAN	0.116	IMGCR	0.1187	ACTVE	0.1329	FPLAY	0.1084	USENS	0.119	URBEH	0.0944	OMAN	0.7813			
USENS	0.0679	FPLAY	0.1049	USENS	0.0892	USENS	0.0577	OMAN	0.0987	USENS	0.0711	USENS	0.5297			
ANXTY	1E-05	ANXTY	1E-05	SELFINJ	1E-05	ANXTY	2E-05	SELFINJ	1E-05	ANXTY	2E-05	ANXTY	8E-05			
SELFINJ	8E-06	SELFINJ	1E-05	ANXTY	1E-05	SELFINJ	1E-05	ANXTY	9E-06	SELFINJ	8E-06	SELFINJ	6E-05			

Module1 29-21

																overall
URBEH	0.1707	URBEH	0.23	IMGCR	0.1676	IMGCR	0.1795	IMGCR	0.1865	IMGCR	0.1696	IMGCR	0.1825	IMGCR	1.1947	
ACTVE	0.1661	USENS	0.1554	ACTVE	0.1647	FPLAY	0.1488	AGG	0.1672	URBEH	0.1675	OMAN	0.1771	URBEH	1.0675	
IMGCR	0.1592	IMGCR	0.1499	URBEH	0.1615	URBEH	0.1311	OMAN	0.1359	AGG	0.1424	ACTVE	0.1569	ACTVE	0.9575	
FPLAY	0.1353	ACTVE	0.142	OMAN	0.1411	ACTVE	0.115	FPLAY	0.1256	USENS	0.1401	FPLAY	0.1391	AGG	0.8515	
OMAN	0.1262	AGG	0.1383	AGG	0.1217	USENS	0.1147	URBEH	0.1148	OMAN	0.1141	USENS	0.13	FPLAY	0.8398	
USENS	0.1132	FPLAY	0.087	USENS	0.111	AGG	0.1114	ACTVE	0.1093	FPLAY	0.1052	URBEH	0.0919	USENS	0.8392	
AGG	0.0791	ANXTY	0.0545	FPLAY	0.0987	ANXTY	0.1052	ANXTY	0.0858	ACTVE	0.1036	AGG	0.0913	OMAN	0.8315	
ANXTY	0.0502	OMAN	0.0427	ANXTY	0.0338	OMAN	0.0944	USENS	0.0749	ANXTY	0.0575	ANXTY	0.0311	ANXTY	0.4181	
SELFINJ	1E-05	SELFINJ	1E-05	SELFINJ	1E-05	SELFINJ	2E-05	SELFINJ	2E-05	SELFINJ	2E-05	SELFINJ	2E-05	SELFINJ	0.0001	

Module1 29-22

																overall
AGG	0.2091	AGG	0.1876	AGG	0.1759	AGG	0.2166	AGG	0.2379	OMAN	0.1849	AGG	1.1359			
OMAN	0.1525	URBEH	0.165	IMGCR	0.1696	OMAN	0.2014	USENS	0.1548	ACTVE	0.1802	OMAN	0.9208			
URBEH	0.1483	OMAN	0.1584	ACTVE	0.1573	ACTVE	0.149	ACTVE	0.1411	IMGCR	0.1459	ACTVE	0.9134			
ACTVE	0.1287	ACTVE	0.1571	URBEH	0.1437	URBEH	0.1386	IMGCR	0.1301	URBEH	0.1358	URBEH	0.8209			
IMGCR	0.1236	USENS	0.1486	FPLAY	0.1244	USENS	0.0978	OMAN	0.1158	FPLAY	0.1276	IMGCR	0.7485			
USENS	0.1109	IMGCR	0.0972	OMAN	0.1079	IMGCR	0.0822	FPLAY	0.0906	AGG	0.1089	USENS	0.6829			
FPLAY	0.0728	FPLAY	0.0572	USENS	0.0784	FPLAY	0.0785	URBEH	0.0894	USENS	0.0923	FPLAY	0.5512			
SELFINJ	0.0336	SELFINJ	0.0161	ANXTY	0.0369	SELFINJ	0.0277	SELFINJ	0.0359	ANXTY	0.0193	SELFINJ	0.1242			
ANXTY	0.0205	ANXTY	0.0128	SELFINJ	0.0059	ANXTY	0.0082	ANXTY	0.0044	SELFINJ	0.0051	ANXTY	0.1022			

Module1 29-23

																overall
ACTVE	0.187	OMAN	0.172	AGG	0.1585	IMGCR	0.2181	FPLAY	0.1664	IMGCR	0.155	IMGCR	0.1626	IMGCR	1.0898	
AGG	0.1436	IMGCR	0.154	IMGCR	0.1576	URBEH	0.126	ACTVE	0.1615	AGG	0.1492	URBEH	0.1544	URBEH	0.9867	
URBEH	0.141	ACTVE	0.1487	URBEH	0.1455	FPLAY	0.1192	URBEH	0.1455	ACTVE	0.1368	AGG	0.1515	ACTVE	0.9451	
IMGCR	0.1276	URBEH	0.1485	OMAN	0.1182	AGG	0.1188	AGG	0.1275	URBEH	0.1257	FPLAY	0.1436	AGG	0.9406	
OMAN	0.12	SELFINJ	0.097	SELFINJ	0.1156	OMAN	0.1164	IMGCR	0.1149	USENS	0.1129	OMAN	0.1319	OMAN	0.8714	
FPLAY	0.0975	FPLAY	0.0945	ACTVE	0.1097	ACTVE	0.1048	OMAN	0.1018	OMAN	0.1111	ACTVE	0.0966	FPLAY	0.8286	
SELFINJ	0.0883	AGG	0.0915	FPLAY	0.1015	USENS	0.0896	USENS	0.0958	FPLAY	0.1059	USENS	0.0863	USENS	0.6327	
USENS	0.0754	USENS	0.0794	USENS	0.0933	SELFINJ	0.0546	SELFINJ	0.0654	SELFINJ	0.0706	SELFINJ	0.0422	SELFINJ	0.5337	
ANXTY	0.0195	ANXTY	0.0143	ANXTY	1E-05	ANXTY	0.0525	ANXTY	0.0212	ANXTY	0.0329	ANXTY	0.0309	ANXTY	0.1714	

module 2 28-19

									over all
OLANG	0.26693	DGEST	0.18154	POINT	0.28455	STEREO	0.2288	POINT	0.70866
STEREO	0.1697	STEREO	0.18148	IECHO	0.15652	CONVS	0.1752	STEREO	0.70027
IECHO	0.16198	POINT	0.17685	DGEST	0.13928	POINT	0.14814	CONVS	0.59775
CONVS	0.12436	CONVS	0.16798	OLANG	0.13863	DGEST	0.13398	OLANG	0.59536
DGEST	0.11674	IECHO	0.14254	CONVS	0.13021	OLANG	0.12531	DGEST	0.57154
POINT	0.09912	SPABN	0.08512	STEREO	0.12029	SPABN	0.09609	IECHO	0.55351
SPABN	0.06118	OLANG	0.06449	SPABN	0.03052	IECHO	0.09248	SPABN	0.2729

module 2 28-20

									over all
DGEST	0.25719	IECHO	0.33006	IECHO	0.2221	OLANG	0.23391	IECHO	0.95077
IECHO	0.20365	DGEST	0.15835	DGEST	0.21884	IECHO	0.19497	DGEST	0.79302
CONVS	0.14741	OLANG	0.14433	SPABN	0.12821	DGEST	0.15864	OLANG	0.56953
STEREO	0.13204	CONVS	0.12605	POINT	0.12319	SPABN	0.11812	CONVS	0.49149
SPABN	0.11967	POINT	0.1011	CONVS	0.10546	CONVS	0.11257	SPABN	0.44443
OLANG	0.08627	SPABN	0.07844	OLANG	0.10502	STEREO	0.11031	STEREO	0.40123
POINT	0.05376	STEREO	0.06168	STEREO	0.09719	POINT	0.07148	POINT	0.34953

module 2 28-21

											overall
DGEST	0.22343	STEREO	0.20953	STEREO	0.2044	OLANG	0.40017	STEREO	0.26409	STEREO	1.03255
STEREO	0.21093	IECHO	0.20823	DGEST	0.19104	STEREO	0.1436	DGEST	0.18285	OLANG	0.94254
OLANG	0.16234	DGEST	0.19557	OLANG	0.16991	IECHO	0.13835	SPABN	0.12545	DGEST	0.87718
IECHO	0.13286	SPABN	0.11847	IECHO	0.16287	POINT	0.09423	OLANG	0.11423	IECHO	0.74368
CONVS	0.1269	POINT	0.11405	CONVS	0.11525	SPABN	0.08663	CONVS	0.11173	SPABN	0.49191
SPABN	0.08951	OLANG	0.09589	POINT	0.08468	DGEST	0.08429	IECHO	0.10137	CONVS	0.46487
POINT	0.05404	CONVS	0.05826	SPABN	0.07185	CONVS	0.05273	POINT	0.10028	POINT	0.44728

module 2 28-22

											overall
STEREO	0.19805	STEREO	0.2047	STEREO	0.2442	STEREO	0.22556	DGEST	0.23634	STEREO	1.01262
IECHO	0.16409	SPABN	0.17616	IECHO	0.18195	IECHO	0.17407	SPABN	0.19404	IECHO	0.88276
POINT	0.16358	IECHO	0.16908	DGEST	0.17636	SPABN	0.16297	IECHO	0.19357	DGEST	0.85432
DGEST	0.14873	DGEST	0.16441	SPABN	0.13051	OLANG	0.14596	OLANG	0.14272	SPABN	0.79391
OLANG	0.14164	OLANG	0.14801	OLANG	0.11685	DGEST	0.12849	STEREO	0.14011	OLANG	0.69517
SPABN	0.13023	POINT	0.09064	POINT	0.10929	POINT	0.1197	POINT	0.07564	POINT	0.55884
CONVS	0.05368	CONVS	0.04701	CONVS	0.04085	CONVS	0.04326	CONVS	0.01757	CONVS	0.20237

Module 2 Section 2 Weights of Hidden Layer Nodes Ranked

module 2 28-19

																				over all	
OQRAP	0.13452	SIJNT	0.13101	SIJNT	0.12808	SHOW	0.151	SHRNJ	0.12458	SHRNJ	0.13264	SHRNJ	0.14395	QSOV	0.15805	UEYE	0.11432	SHRNJ	1.01094		
SHRNJ	0.10826	OQRAP	0.12234	SHOW	0.10854	UEYE	0.14143	SIJNT	0.12315	UEYE	0.11289	QSOV	0.13304	ASOV	0.11902	SHRNJ	0.11331	ASOV	0.92179		
SHOW	0.10425	RNAME	0.11782	ASOV	0.10817	SIJNT	0.1145	ASOV	0.11798	ASOV	0.1042	OQRAP	0.1067	OQRAP	0.11617	ASOV	0.11113	OQRAP	0.9019		
ASOV	0.1012	SHRNJ	0.11116	SHRNJ	0.10782	OQRAP	0.10316	QSOV	0.11258	RNAME	0.10343	ASOV	0.10431	UEYE	0.09071	ARSOC	0.10612	SIJNT	0.89856		
SIJNT	0.09725	ASOV	0.10386	RNAME	0.10455	QSOV	0.10281	QSRES	0.09447	SIJNT	0.09873	FACEO	0.09527	SIJNT	0.08758	SHOW	0.10561	SHOW	0.885		
FACEO	0.09067	UEYE	0.09538	UEYE	0.09905	RNAME	0.09281	SHOW	0.09028	OQRAP	0.0888	ARSOC	0.08998	FACEO	0.08683	RNAME	0.09852	UEYE	0.87737		
UEYE	0.08721	SHOW	0.09403	QSOV	0.08623	SHRNJ	0.09059	RNAME	0.0809	FACEO	0.08755	RNAME	0.08069	SHRNJ	0.07862	OQRAP	0.09248	QSOV	0.84698		
QSOV	0.08009	QSOV	0.07098	ARSOC	0.08459	ARSOC	0.08027	ARSOC	0.06867	SHOW	0.08646	UEYE	0.07834	RNAME	0.07589	SIJNT	0.07384	RNAME	0.82933		
RNAME	0.07472	FACEO	0.05991	OQRAP	0.0731	ASOV	0.0519	FACEO	0.06469	ARSOC	0.07629	SHOW	0.07565	SHOW	0.06918	QSRES	0.0696	ARSOC	0.67553		
ARSOC	0.06543	ARSOC	0.04772	QSRES	0.0659	FACEO	0.05177	OQRAP	0.06464	QSRES	0.06177	QSRES	0.04764	QSRES	0.06148	FACEO	0.0591	FACEO	0.62971		
QSRES	0.05638	QSRES	0.04576	FACEO	0.03394	QSRES	0.01974	UEYE	0.05805	QSOV	0.04723	SIJNT	0.04442	ARSOC	0.05646	QSOV	0.05596	QSRES	0.52274		
RJNT	1.4E-05	RJNT	2.4E-05	RJNT	3.1E-05	RJNT	2.1E-05	RJNT	1.7E-05	RJNT	1.6E-05	RJNT	9E-06	RJNT	1.4E-05	RJNT	1.2E-05	RJNT	0.00016		

module 2 28-20

																				over all	
SIJNT	0.1335	RNAME	0.18319	SIJNT	0.17678	SHOW	0.12393	UEYE	0.14082	QSOV	0.13019	RNAME	0.15787	RNAME	0.14664	RNAME	0.12876	RNAME	1.0114		
SHRNJ	0.1218	SIJNT	0.13667	SHRNJ	0.12136	OQRAP	0.11081	FACEO	0.1308	UEYE	0.11613	UEYE	0.11817	ARSOC	0.10915	ARSOC	0.11666	SIJNT	0.98519		
RNAME	0.10357	SHOW	0.12272	QSOV	0.12015	SHRNJ	0.10307	SHOW	0.12277	SHOW	0.09427	FACEO	0.1017	SHOW	0.10524	QSOV	0.11566	QSOV	0.94012		
QSOV	0.10238	ARSOC	0.09405	SHOW	0.10776	QSOV	0.10004	QSOV	0.11917	OQRAP	0.09394	SHRNJ	0.09735	SHRNJ	0.09998	SIJNT	0.09899	SHRNJ	0.88499		
UEYE	0.09838	ASOV	0.08257	UEYE	0.10647	SIJNT	0.09681	ARSOC	0.10122	FACEO	0.09145	SIJNT	0.0959	OQRAP	0.09132	FACEO	0.09207	SHOW	0.88499		
SHOW	0.09705	QSOV	0.07926	ARSOC	0.08894	ARSOC	0.0876	SIJNT	0.09865	RNAME	0.09008	QSOV	0.0918	UEYE	0.08712	SHRNJ	0.08953	UEYE	0.87802		
FACEO	0.08884	OQRAP	0.07446	RNAME	0.07367	ASOV	0.08596	SHRNJ	0.09364	SHRNJ	0.08898	OQRAP	0.09073	QSOV	0.08147	ASOV	0.07885	ARSOC	0.81238		
OQRAP	0.0785	SHRNJ	0.06927	ASOV	0.06685	UEYE	0.08076	ASOV	0.06116	SIJNT	0.08769	ARSOC	0.08888	FACEO	0.07745	UEYE	0.07411	FACEO	0.76373		
ASOV	0.07015	FACEO	0.0632	FACEO	0.05882	QSRES	0.07624	RNAME	0.05225	ASOV	0.07556	QSRES	0.06495	QSRES	0.07649	QSRES	0.07129	OQRAP	0.69014		
ARSOC	0.05471	UEYE	0.05606	QSRES	0.04033	RNAME	0.07538	OQRAP	0.04058	ARSOC	0.07117	SHOW	0.04809	ASOV	0.06492	OQRAP	0.07093	ASOV	0.63055		
QSRES	0.05112	QSRES	0.03853	OQRAP	0.03887	FACEO	0.05939	QSRES	0.03891	QSRES	0.06052	ASOV	0.04454	SIJNT	0.0602	SHOW	0.06316	QSRES	0.51839		
RJNT	1E-05	RJNT	1.3E-05	RJNT	1.6E-05	RJNT	1.4E-05	RJNT	1.1E-05	RJNT	1.5E-05	RJNT	1.1E-05	RJNT	1.2E-05	RJNT	6.96E-06	RJNT	0.00011		

module 2 28-21

																				overall	
FACEO	0.12633	RNAME	0.13325	ASOV	0.13681	FACEO	0.12024	ASOV	0.1531	RNAME	0.12018	SIJNT	0.14698	SIJNT	0.12149	SHRNJ	0.12083	SIJNT	0.12389	ASOV	1.04
QSOV	0.1037	SHRNJ	0.12058	QSOV	0.1312	QSOV	0.10394	SIJNT	0.11088	ASOV	0.11199	RNAME	0.11518	SHRNJ	0.11255	OQRAP	0.11607	SHOW	0.12216	RNAME	1.03163
OQRAP	0.10061	FACEO	0.09983	UEYE	0.12067	RNAME	0.10351	SHRNJ	0.10599	FACEO	0.11185	OQRAP	0.0947	ASOV	0.10593	ARSOC	0.10318	RNAME	0.12059	SHRNJ	1.00631
RNAME	0.10056	ASOV	0.0935	ARSOC	0.10018	ARSOC	0.09535	ARSOC	0.10383	SHRNJ	0.10216	SHOW	0.09402	SHOW	0.10292	FACEO	0.09928	FACEO	0.11921	FACEO	0.98112
SHRNJ	0.09434	SHOW	0.0891	SHRNJ	0.09077	SHOW	0.09532	RNAME	0.09644	SIJNT	0.09688	ASOV	0.09144	OQRAP	0.10286	QSOV	0.093	OQRAP	0.11643	SIJNT	0.9403
ASOV	0.09392	ARSOC	0.08612	SHOW	0.09038	ASOV	0.09015	OQRAP	0.08097	QSOV	0.09657	QSRES	0.08543	UEYE	0.09255	UEYE	0.08452	SHRNJ	0.09587	SHOW	0.90805
SIJNT	0.08582	UEYE	0.08384	OQRAP	0.08895	SIJNT	0.08534	FACEO	0.08031	SHOW	0.08061	QSOV	0.08492	RNAME	0.09164	ASOV	0.08323	UEYE	0.08769	OQRAP	0.90331
UEYE	0.08357	QSOV	0.0826	RNAME	0.0748	SHRNJ	0.08285	SHOW	0.07846	QSRES	0.07608	SHRNJ	0.08038	FACEO	0.08708	SHOW	0.08211	ASOV	0.07994	QSOV	0.87881
SHOW	0.07298	OQRAP	0.06622	FACEO	0.07439	UEYE	0.0725	UEYE	0.07437	ARSOC	0.06828	UEYE	0.07061	QSOV	0.07043	RNAME	0.0755	QSOV	0.05861	UEYE	0.82914
ARSOC	0.06479	SIJNT	0.066	SIJNT	0.04522	OQRAP	0.07018	QSOV	0.05384	OQRAP	0.06632	ARSOC	0.06495	ARSOC	0.07029	QSRES	0.06981	ARSOC	0.0433	ARSOC	0.80029
QSRES	0.05067	QSRES	0.06254	QSRES	0.04458	QSRES	0.05898	QSRES	0.05191	UEYE	0.05882	FACEO	0.0626	QSRES	0.03692	SIJNT	0.0578	QSRES	0.02091	QSRES	0.55783
RJNT	0.02271	RJNT	0.01643	RJNT	0.00204	RJNT	0.02165	RJNT	0.0099	RJNT	0.01027	RJNT	0.00879	RJNT	0.00534	RJNT	0.01469	RJNT	0.0114	RJNT	0.12322

module 2 28-22

																				overall	
RNAME	0.14917	OQRAP	0.16714	SHRNJ	0.11912	QSOV	0.12234	ARSOC	0.12259	SIJNT	0.12512	SIJNT	0.12492	UEYE	0.12764	SHRNJ	0.13292	FACEO	0.10527	OQRAP	0.97653
QSRES	0.1141	QSRES	0.15602	RNAME	0.10223	QSRES	0.1048	RNAME	0.11521	UEYE	0.11458	OQRAP	0.11489	SIJNT	0.11459	SIJNT	0.12424	SHOW	0.10258	UEYE	0.97634
ARSOC	0.11347	RNAME	0.11284	QSRES	0.09983	ARSOC	0.10193	SIJNT	0.11358	SHRNJ	0.10987	SHOW	0.11092	ASOV	0.1046	ASOV	0.10518	QSOV	0.10237	SHRNJ	0.95272
UEYE	0.0963	SHOW	0.11084	QSOV	0.09873	SHRNJ	0.0972	QSRES	0.11301	QSOV	0.1021	QSOV	0.09894	QSOV	0.10247	OQRAP	0.10093	SIJNT	0.10016	RNAME	0.9486
ASOV	0.09288	UEYE	0.10084	ARSOC	0.09686	FACEO	0.09716	SHRNJ	0.10101	OQRAP	0.1005	UEYE	0.09488	RNAME	0.09695	ARSOC	0.09991	UEYE	0.09394	SIJNT	0.94273
QSOV	0.0907	SHRNJ	0.09217	UEYE	0.09363	ASOV	0.09693	ASOV	0.0958	QSRES	0.08892	FACEO	0.08988	FACEO	0.09029	SHOW	0.09237	ASOV	0.09347	QSRES	0.94204
SIJNT	0.08802	FACEO	0.07133	OQRAP	0.09312	OQRAP	0.0877	SHOW	0.09425	SHOW	0.08307	SHRNJ	0.08863	OQRAP	0.08624	UEYE	0.09223	RNAME	0.08762	ARSOC	0.92108
OQRAP	0.06713	ARSOC	0.05786	FACEO	0.08599	UEYE	0.08232	UEYE	0.07998	ARSOC	0.08149	ARSOC	0.08828	SHRNJ	0.07996	QSRES	0.08732	ARSOC	0.08402	ASOV	0.86976
FACEO	0.06629	ASOV	0.05613	ASOV	0.0773	RNAME	0.0803	OQRAP	0.07516	RNAME	0.07081	ASOV	0.08027	ARSOC	0.07467	RNAME	0.05967	OQRAP	0.08371	QSOV	0.86338
SHOW	0.0614	QSOV	0.0392	SHOW	0.0744	SHOW	0.07158	QSOV	0.04754	ASOV	0.0672	RNAME	0.07378	QSRES	0.06794	QSOV	0.05898	QSRES	0.0755	SHOW	0.85605
SHRNJ	0.06051	SIJNT	0.0356	SIJNT	0.05878	SIJNT	0.05773	FACEO	0.04185	FACEO	0.05633	QSRES	0.03459	SHOW	0.05463	FACEO	0.04624	SHRNJ	0.07134	FACEO	0.75063
RJNT	1.1E-05	RJNT	1.5E-05	RJNT	9.9E-06	RJNT	1.5E-05	RJNT	1.7E-05	RJNT	1.4E-05	RJNT	1.8E-05	RJNT	9.2E-06	RJNT	1.8E-05	RJNT	2.1E-05	RJNT	0.00015

Module 2 Section 3 Weights of Hidden Layer Nodes Ranked

module 2 28-19

													overalll	
ACTVE	0.23806	OMAN	0.29256	ACTVE	0.32424	USENS	0.19124	USENS	0.22159	URBEH	0.24499	ACTVE	1.17547	
IMGCR	0.17593	FPLAY	0.20153	URBEH	0.16091	IMGCR	0.17559	FPLAY	0.15994	ACTVE	0.23452	OMAN	0.9145	
FPLAY	0.14801	AGG	0.13761	IMGCR	0.1589	URBEH	0.15109	ACTVE	0.15815	OMAN	0.13616	URBEH	0.88358	
USENS	0.14046	ACTVE	0.12032	AGG	0.10457	FPLAY	0.13886	OMAN	0.13687	IMGCR	0.13314	USENS	0.86841	
OMAN	0.13724	USENS	0.10459	OMAN	0.088	OMAN	0.12367	URBEH	0.1256	USENS	0.12664	IMGCR	0.79713	
URBEH	0.13121	IMGCR	0.07361	USENS	0.08389	AGG	0.11935	AGG	0.11786	AGG	0.0954	FPLAY	0.75693	
AGG	0.02907	URBEH	0.06977	FPLAY	0.07947	ACTVE	0.10017	IMGCR	0.07997	FPLAY	0.02913	AGG	0.60386	
SELFINJ	1.2E-05	ANXTY	1.1E-05	SELFINJ	1.2E-05	SELFINJ	1.7E-05	SELFINJ	8E-06	SELFINJ	1.1E-05	SELFINJ	6.9E-05	
ANXTY	1E-05	SELFINJ	8.6E-06	ANXTY	6.5E-06	ANXTY	5.3E-06	ANXTY	3.8E-06	ANXTY	9.2E-06	ANXTY	4.6E-05	

module 2 28-20

														over all	
URBEH	0.1786	IMGCR	0.21245	USENS	0.2638	ACTVE	0.17691	FPLAY	0.21033	FPLAY	0.19247	URBEH	0.19254	URBEH	1.05876
OMAN	0.15511	FPLAY	0.16463	AGG	0.14595	AGG	0.17173	ACTVE	0.17401	ACTVE	0.1864	OMAN	0.17983	FPLAY	1.03582
ACTVE	0.14238	OMAN	0.15793	URBEH	0.1375	URBEH	0.16294	URBEH	0.14797	IMGCR	0.16868	AGG	0.17633	ACTVE	1.01785
FPLAY	0.12763	URBEH	0.15354	OMAN	0.12214	IMGCR	0.15151	AGG	0.1317	USENS	0.16546	ACTVE	0.13563	IMGCR	0.99018
IMGCR	0.11394	AGG	0.11519	ACTVE	0.1185	FPLAY	0.14485	IMGCR	0.12171	OMAN	0.097	IMGCR	0.13194	USENS	0.96517
USENS	0.10179	USENS	0.11016	FPLAY	0.10019	USENS	0.11799	USENS	0.12096	URBEH	0.08568	FPLAY	0.09571	AGG	0.91038
AGG	0.09881	ACTVE	0.08403	IMGCR	0.08994	OMAN	0.07319	OMAN	0.08321	AGG	0.07066	USENS	0.08501	OMAN	0.86842
ANXTY	0.08171	ANXTY	0.00205	ANXTY	0.02196	ANXTY	0.00086	ANXTY	0.01008	ANXTY	0.03363	ANXTY	0.00298	ANXTY	0.15326
SELFINJ	2E-05	SELFINJ	2.5E-05	SELFINJ	2.4E-05	SELFINJ	1.9E-05	SELFINJ	2E-05	SELFINJ	2.1E-05	SELFINJ	2.7E-05	SELFINJ	0.00016

module 2 28-21

													overall	
ACTVE	0.18128	OMAN	0.22152	OMAN	0.21064	OMAN	0.25086	FPLAY	0.26612	AGG	0.34032	ACTVE	1.02288	
IMGCR	0.18002	ACTVE	0.20015	ACTVE	0.18706	USENS	0.17306	ACTVE	0.23113	FPLAY	0.14164	OMAN	1.01714	
FPLAY	0.16194	USENS	0.19791	FPLAY	0.15402	IMGCR	0.16032	USENS	0.15573	URBEH	0.13209	FPLAY	0.89258	
USENS	0.15663	URBEH	0.13755	IMGCR	0.13896	FPLAY	0.12951	OMAN	0.15005	ACTVE	0.10507	USENS	0.87426	
AGG	0.15175	AGG	0.13269	USENS	0.13501	ACTVE	0.1182	AGG	0.07696	IMGCR	0.09001	AGG	0.85231	
OMAN	0.09825	IMGCR	0.0694	URBEH	0.11435	AGG	0.09276	IMGCR	0.07281	OMAN	0.08582	IMGCR	0.71152	
URBEH	0.07011	FPLAY	0.03935	AGG	0.05783	URBEH	0.07524	URBEH	0.03408	USENS	0.05592	URBEH	0.56342	
SELFINJ	1.5E-05	ANXTY	0.00141	ANXTY	0.00211	SELFINJ	3.6E-05	ANXTY	0.01309	ANXTY	0.04911	ANXTY	0.06573	
ANXTY	1.2E-05	SELFINJ	2.5E-05	SELFINJ	1.9E-05	ANXTY	2.9E-06	SELFINJ	3.7E-05	SELFINJ	1.6E-05	SELFINJ	0.00015	

module 2 28-22

													overall	
URBEH	0.17552	URBEH	0.1796	AGG	0.15353	ACTVE	0.33538	USENS	0.1991	IMGCR	0.22317	AGG	0.24926	ACTVE	1.11616
IMGCR	0.16868	USENS	0.1703	OMAN	0.14892	URBEH	0.18408	FPLAY	0.18652	USENS	0.14927	OMAN	0.16168	URBEH	1.03871
FPLAY	0.1501	ACTVE	0.1589	URBEH	0.14892	IMGCR	0.12087	OMAN	0.18399	URBEH	0.14664	IMGCR	0.14416	IMGCR	1.01711
AGG	0.14503	IMGCR	0.15084	FPLAY	0.13766	USENS	0.11228	ACTVE	0.17111	FPLAY	0.13676	USENS	0.11329	USENS	1.00175
OMAN	0.13903	AGG	0.14146	USENS	0.13316	OMAN	0.10077	URBEH	0.1027	ACTVE	0.12882	FPLAY	0.10504	AGG	0.94042
USENS	0.12435	FPLAY	0.12138	ACTVE	0.13148	AGG	0.08945	IMGCR	0.07795	OMAN	0.11122	URBEH	0.10125	OMAN	0.92056
ACTVE	0.09206	OMAN	0.07495	IMGCR	0.13144	FPLAY	0.03748	AGG	0.07037	AGG	0.09133	ACTVE	0.09841	FPLAY	0.87494
ANXTY	0.00523	ANXTY	0.00257	ANXTY	0.01486	ANXTY	0.01965	ANXTY	0.00825	ANXTY	0.01276	ANXTY	0.02689	ANXTY	0.09021
SELFINJ	2E-05	SELFINJ	1.2E-05	SELFINJ	2.9E-05	SELFINJ	2.4E-05	SELFINJ	1.7E-05	SELFINJ	1.9E-05	SELFINJ	2E-05	SELFINJ	0.00014

Module 3 Section 1 Weights of Hidden Layer Nodes Ranked

module3 28-19

													overall
OLANG	0.201382	OLANG	0.233947	DGEST	0.22672	DGEST	0.22225	DGEST	0.205114	REPRT	0.255963	DGEST	1.057394
SPABN	0.158177	AINFO	0.219867	REPRT	0.182618	STEREO	0.201028	REPRT	0.192699	AINFO	0.15323	AINFO	0.991035
AINFO	0.157499	DGEST	0.184342	STEREO	0.143593	AINFO	0.185103	OLANG	0.18741	OINFO	0.142775	REPRT	0.922371
OINFO	0.155951	REPRT	0.107989	AINFO	0.134637	SPABN	0.129213	OINFO	0.158779	SPABN	0.11592	OLANG	0.891673
DGEST	0.113664	STEREO	0.095749	OINFO	0.119354	REPRT	0.084306	AINFO	0.140699	DGEST	0.105304	STEREO	0.72349
STEREO	0.104838	SPABN	0.07497	OLANG	0.09932	OINFO	0.081216	STEREO	0.074819	STEREO	0.103464	OINFO	0.715556
REPRT	0.098796	OINFO	0.05748	SPABN	0.09241	OLANG	0.069684	SPABN	0.029041	OLANG	0.09993	SPABN	0.599731
CONVS	0.009685	CONVS	0.025645	CONVS	0.001339	CONVS	0.027195	CONVS	0.011431	CONVS	0.023406	CONVS	0.098702
IECHO	7.88E-06	IECHO	9.03E-06	IECHO	8.91E-06	IECHO	5.48E-06	IECHO	7.25E-06	IECHO	8.78E-06	IECHO	4.73E-05

module3 28-20

													overall
OLANG	0.196975	OLANG	0.207845	AINFO	0.212256	DGEST	0.20262	AINFO	0.250411	OLANG	0.230895	OLANG	1.156706
SPABN	0.165265	DGEST	0.17159	SPABN	0.186454	OLANG	0.201589	SPABN	0.169933	AINFO	0.156203	AINFO	1.007326
OINFO	0.161621	STEREO	0.162834	OLANG	0.168643	OINFO	0.172367	OLANG	0.150759	OINFO	0.145561	SPABN	0.914717
AINFO	0.139226	SPABN	0.14639	STEREO	0.143572	SPABN	0.125402	STEREO	0.14968	STEREO	0.128547	DGEST	0.820164
DGEST	0.135666	AINFO	0.144236	DGEST	0.131765	STEREO	0.117213	OINFO	0.144073	DGEST	0.123473	STEREO	0.80144
STEREO	0.099595	OINFO	0.103361	REPRT	0.084401	AINFO	0.104995	REPRT	0.063748	SPABN	0.121273	OINFO	0.790873
REPRT	0.081173	REPRT	0.054036	OINFO	0.06389	REPRT	0.071027	DGEST	0.05505	REPRT	0.076691	REPRT	0.431078
CONVS	0.02047	CONVS	0.009701	CONVS	0.009011	CONVS	0.004784	CONVS	0.016339	CONVS	0.01735	CONVS	0.077655
IECHO	8.81E-06	IECHO	6.6E-06	IECHO	6.79E-06	IECHO	4.29E-06	IECHO	6.17E-06	IECHO	7.98E-06	IECHO	4.06E-05

module3 28-21

															overall
AINFO	0.17167	DGEST	0.182581	AINFO	0.177786	OINFO	0.197288	OLANG	0.240285	AINFO	0.166775	DGEST	0.173689	AINFO	1.028975
CONVS	0.137612	AINFO	0.166318	OLANG	0.157097	STEREO	0.190348	DGEST	0.14676	SPABN	0.146013	OLANG	0.144673	OLANG	0.965112
DGEST	0.132863	CONVS	0.127919	OINFO	0.149309	DGEST	0.114296	CONVS	0.11765	OINFO	0.136115	AINFO	0.134248	DGEST	0.947804
OINFO	0.127522	OINFO	0.119465	STEREO	0.141156	AINFO	0.111395	AINFO	0.100784	DGEST	0.125135	SPABN	0.133915	OINFO	0.915363
STEREO	0.11868	OLANG	0.102092	SPABN	0.104445	OLANG	0.103162	STEREO	0.096524	STEREO	0.115381	STEREO	0.120673	STEREO	0.878432
OLANG	0.117802	STEREO	0.09567	CONVS	0.095227	SPABN	0.096451	OINFO	0.092497	REPRT	0.115296	OINFO	0.093168	SPABN	0.76805
SPABN	0.108956	REPRT	0.088407	REPRT	0.091792	CONVS	0.088823	SPABN	0.090262	OLANG	0.100001	CONVS	0.076113	CONVS	0.72067
REPRT	0.06017	SPABN	0.088008	DGEST	0.072479	REPRT	0.084846	REPRT	0.077466	CONVS	0.077325	REPRT	0.074826	REPRT	0.592804
IECHO	0.024726	IECHO	0.029539	IECHO	0.010709	IECHO	0.01339	IECHO	0.037772	IECHO	0.01796	IECHO	0.048696	IECHO	0.182791

module3 28-22

															overall
DGEST	0.1796	REPRT	0.163387	IECHO	0.162485	STEREO	0.195121	REPRT	0.171494	OLANG	0.220773	STEREO	0.201656	OLANG	1.134204
OLANG	0.17624	AINFO	0.162855	OLANG	0.147656	OLANG	0.16588	STEREO	0.170718	DGEST	0.142962	REPRT	0.152375	STEREO	1.034956
REPRT	0.146105	DGEST	0.143655	STEREO	0.131411	DGEST	0.153592	AINFO	0.159912	REPRT	0.137176	OLANG	0.143695	DGEST	0.992407
STEREO	0.144744	OLANG	0.138939	AINFO	0.126356	SPABN	0.133634	OLANG	0.141021	AINFO	0.122307	DGEST	0.130187	REPRT	0.950557
AINFO	0.129168	SPABN	0.12524	SPABN	0.115476	OINFO	0.108177	DGEST	0.132836	SPABN	0.112489	SPABN	0.125547	AINFO	0.881222
SPABN	0.117731	OINFO	0.100913	DGEST	0.109575	AINFO	0.082775	SPABN	0.115898	STEREO	0.11013	OINFO	0.10169	SPABN	0.846017
CONVS	0.048941	CONVS	0.083168	REPRT	0.102055	REPRT	0.077965	CONVS	0.053527	OINFO	0.091436	AINFO	0.09785	OINFO	0.556849
OINFO	0.043047	STEREO	0.081175	OINFO	0.058599	CONVS	0.067291	OINFO	0.052985	CONVS	0.061937	CONVS	0.036396	CONVS	0.397645
IECHO	0.014423	IECHO	0.000669	CONVS	0.046386	IECHO	0.015565	IECHO	0.001608	IECHO	0.00079	IECHO	0.010603	IECHO	0.206143

module3 28-19																						overall	
FACEO	0.198466	LLNVC	0.183529	EMPTH	0.146341	INSIG	0.178204	INSIG	0.143785	EMPTH	0.150748	FACEO	0.168941	EMPTH	0.957368								
ARSOC	0.154608	INSIG	0.156326	SHRNJ	0.131537	LLNVC	0.144034	EMPTH	0.141983	INSIG	0.13025	QSOV	0.155537	INSIG	0.936019								
SHRNJ	0.134325	EMPTH	0.14052	INSIG	0.127572	ARSOC	0.143245	SHRNJ	0.138463	OQRAP	0.12764	EMPTH	0.128757	LLNVC	0.823743								
EMPTH	0.122143	OQRAP	0.118054	OQRAP	0.115313	EMPTH	0.126876	FACEO	0.128192	QSOV	0.11739	LLNVC	0.103361	FACEO	0.784724								
LLNVC	0.10813	QSRES	0.103166	LLNVC	0.112151	SHRNJ	0.121299	ARSOC	0.09859	ARSOC	0.113877	OQRAP	0.101984	SHRNJ	0.78106								
INSIG	0.106693	SHRNJ	0.075108	QSOV	0.10434	OQRAP	0.088328	QSOV	0.093738	QSRES	0.106426	INSIG	0.09319	ARSOC	0.761125								
OQRAP	0.091625	QSOV	0.075089	QSRES	0.101108	FACEO	0.070502	OQRAP	0.089513	SHRNJ	0.089619	SHRNJ	0.090709	OQRAP	0.732458								
QSOV	0.048776	FACEO	0.074467	ARSOC	0.095597	QSRES	0.067275	LLNVC	0.086653	LLNVC	0.085886	ARSOC	0.081494	QSOV	0.655085								
QSRES	0.03521	ARSOC	0.073714	FACEO	0.066015	QSOV	0.060216	QSRES	0.079071	FACEO	0.078141	QSRES	0.076004	QSRES	0.568261								
UEYE	2.35E-05	UEYE	2.84E-05	UEYE	2.61E-05	UEYE	2.05E-05	UEYE	1.21E-05	UEYE	2.26E-05	UEYE	2.31E-05	UEYE	0.000156								
module3 28-20																						overall	
EMPTH	0.172511	OQRAP	0.145209	ARSOC	0.156782	LLNVC	0.140377	EMPTH	0.16863	ARSOC	0.195335	ARSOC	0.142698	ARSOC	1.022295								
ARSOC	0.161526	QSRES	0.14516	FACEO	0.130311	EMPTH	0.139901	OQRAP	0.146983	QSRES	0.136195	EMPTH	0.135925	EMPTH	0.926727								
QSRES	0.133808	EMPTH	0.123075	INSIG	0.120166	INSIG	0.125717	ARSOC	0.13458	LLNVC	0.119384	QSOV	0.133927	OQRAP	0.868516								
SHRNJ	0.113125	QSOV	0.119375	LLNVC	0.116645	OQRAP	0.116911	LLNVC	0.11385	OQRAP	0.118084	OQRAP	0.121751	QSRES	0.791516								
OQRAP	0.10792	ARSOC	0.115781	OQRAP	0.111658	ARSOC	0.115592	SHRNJ	0.113187	SHRNJ	0.117647	SHRNJ	0.114828	LLNVC	0.730988								
INSIG	0.105015	SHRNJ	0.106723	QSOV	0.107604	QSRES	0.105177	QSOV	0.110945	FACEO	0.094241	LLNVC	0.106371	SHRNJ	0.701621								
QSOV	0.085286	INSIG	0.090804	QSRES	0.098619	QSOV	0.104156	QSRES	0.092043	EMPTH	0.093431	INSIG	0.084729	INSIG	0.698217								
LLNVC	0.067472	FACEO	0.085972	EMPTH	0.093253	FACEO	0.08197	INSIG	0.080122	INSIG	0.091664	QSRES	0.080515	QSOV	0.695287								
FACEO	0.053313	LLNVC	0.067889	SHRNJ	0.065941	SHRNJ	0.07017	FACEO	0.039638	QSOV	0.033994	FACEO	0.079233	FACEO	0.564678								
UEYE	2.24E-05	UEYE	1.19E-05	UEYE	2.03E-05	UEYE	2.98E-05	UEYE	2.14E-05	UEYE	2.61E-05	UEYE	2.41E-05	UEYE	0.000156								
module3 28-21																						overall	
LLNVC	0.132665	EMPTH	0.166952	ARSOC	0.143956	FACEO	0.184362	OQRAP	0.148329	OQRAP	0.188799	OQRAP	0.146557	EMPTH	0.157522	EMPTH	1.019922						
INSIG	0.130287	LLNVC	0.160415	INSIG	0.137124	QSRES	0.160094	ARSOC	0.139689	QSRES	0.180848	ARSOC	0.140599	ARSOC	0.133017	ARSOC	1.016983						
QSOV	0.119278	INSIG	0.156386	SHRNJ	0.130908	EMPTH	0.122492	INSIG	0.12323	FACEO	0.120394	INSIG	0.134587	FACEO	0.107458	INSIG	0.976547						
EMPTH	0.117025	ARSOC	0.140876	QSRES	0.123231	INSIG	0.113959	QSRES	0.119386	ARSOC	0.12009	EMPTH	0.12458	QSRES	0.107034	OQRAP	0.956506						
SHRNJ	0.115647	SHRNJ	0.119726	EMPTH	0.121978	SHRNJ	0.10011	SHRNJ	0.106789	EMPTH	0.114615	FACEO	0.121531	LLNVC	0.104213	QSRES	0.9296						
OQRAP	0.103519	OQRAP	0.082569	OQRAP	0.09886	ARSOC	0.098285	QSOV	0.10625	INSIG	0.077613	LLNVC	0.104431	OQRAP	0.103648	FACEO	0.85081						
ARSOC	0.10047	FACEO	0.066038	QSOV	0.092514	OQRAP	0.084223	EMPTH	0.094757	LLNVC	0.074377	QSRES	0.102469	INSIG	0.103362	LLNVC	0.802546						
QSRES	0.092037	QSOV	0.062509	LLNVC	0.077138	LLNVC	0.075463	FACEO	0.08771	QSOV	0.073772	QSOV	0.090807	SHRNJ	0.097067	SHRNJ	0.754142						
FACEO	0.089045	QSRES	0.044501	FACEO	0.074272	QSOV	0.060992	LLNVC	0.073845	SHRNJ	0.049479	SHRNJ	0.034416	QSOV	0.086654	QSOV	0.692776						
UEYE	2.74E-05	UEYE	2.75E-05	UEYE	1.91E-05	UEYE	2.07E-05	UEYE	1.44E-05	UEYE	1.23E-05	UEYE	2.29E-05	UEYE	2.55E-05	UEYE	0.00017						
module3 28-22																						overall	
INSIG	0.146332	OQRAP	0.165849	EMPTH	0.161527	EMPTH	0.164471	ARSOC	0.169344	SHRNJ	0.14862	OQRAP	0.151047	INSIG	0.157475	INSIG	1.10128						
ARSOC	0.144777	INSIG	0.134168	ARSOC	0.151865	ARSOC	0.151413	INSIG	0.160554	INSIG	0.128786	ARSOC	0.14696	SHRNJ	0.151795	ARSOC	1.099668						
SHRNJ	0.139871	ARSOC	0.12975	LLNVC	0.145571	INSIG	0.125085	EMPTH	0.142336	ARSOC	0.11561	QSOV	0.135314	QSRES	0.141756	EMPTH	0.957591						
QSOV	0.128321	QSRES	0.108736	QSRES	0.125972	QSOV	0.115889	QSRES	0.109138	FACEO	0.10991	INSIG	0.134708	FACEO	0.113466	SHRNJ	0.933942						
EMPTH	0.12738	FACEO	0.100184	INSIG	0.114173	LLNVC	0.101268	LLNVC	0.107268	QSOV	0.10472	SHRNJ	0.104507	LLNVC	0.097501	QSRES	0.859552						
LLNVC	0.105747	QSOV	0.099741	SHRNJ	0.103178	QSRES	0.098308	SHRNJ	0.09427	QSRES	0.102947	EMPTH	0.097687	QSOV	0.09252	QSOV	0.82777						
QSRES	0.093865	SHRNJ	0.097345	OQRAP	0.090279	SHRNJ	0.094354	QSOV	0.089094	LLNVC	0.101438	FACEO	0.080717	ARSOC	0.089949	LLNVC	0.796301						
FACEO	0.076008	EMPTH	0.096903	QSOV	0.06217	FACEO	0.082827	OQRAP	0.066655	EMPTH	0.101121	QSRES	0.078831	OQRAP	0.089324	OQRAP	0.753959						
OQRAP	0.037642	LLNVC	0.067295	FACEO	0.045238	OQRAP	0.066358	FACEO	0.061295	OQRAP	0.086804	LLNVC	0.070213	EMPTH	0.066167	FACEO	0.669645						
UEYE	5.58E-05	UEYE	3.01E-05	UEYE	2.81E-05	UEYE	2.7E-05	UEYE	4.52E-05	UEYE	4.21E-05	UEYE	1.62E-05	UEYE	4.67E-05	UEYE	0.000291						

Module 3 Section 3 Weights of Hidden Layer Nodes Ranked

module3 28-19																						
																					overall	
ACTVE	0.345297	RITL	0.435049	RITL	0.322665	OMAN	0.305562	ACTVE	0.341085	OMAN	0.335456	RITL	1.170622									
IMGCR	0.145572	OMAN	0.145275	TOPIC	0.221197	IMGCR	0.17669	TOPIC	0.212132	USENS	0.200667	OMAN	1.078963									
USENS	0.129441	IMGCR	0.133272	USENS	0.152363	USENS	0.174635	OMAN	0.137528	TOPIC	0.128344	ACTVE	0.971971									
AGG	0.113471	USENS	0.12496	IMGCR	0.126754	ACTVE	0.10818	RITL	0.119243	RITL	0.109914	USENS	0.89698									
TOPIC	0.103369	TOPIC	0.100665	OMAN	0.078165	RITL	0.097906	USENS	0.114915	IMGCR	0.104797	TOPIC	0.839488									
RITL	0.085846	ACTVE	0.047249	ACTVE	0.072387	TOPIC	0.07378	IMGCR	0.05577	AGG	0.063027	IMGCR	0.742855									
OMAN	0.076978	AGG	0.01352	AGG	0.020698	AGG	0.044019	AGG	0.015326	ACTVE	0.057771	AGG	0.270061									
ANXTY	1.35E-05	SELFINJ	6.23E-06	ANXTY	0.005762	ANXTY	0.01921	ANXTY	0.003986	ANXTY	1.25E-05	ANXTY	0.028988									
SELFINJ	1.31E-05	ANXTY	3.84E-06	SELFINJ	8.76E-06	SELFINJ	1.75E-05	SELFINJ	1.58E-05	SELFINJ	1.19E-05	SELFINJ	7.33E-05									
module3 28-20																						
																					overall	
AGG	0.187682	RITL	0.299944	RITL	0.221042	USENS	0.190714	TOPIC	0.237578	ACTVE	0.197616	OMAN	0.226294	RITL	1.09521							
IMGCR	0.161651	OMAN	0.191434	ACTVE	0.165002	TOPIC	0.157741	ACTVE	0.18876	ANXTY	0.137378	ACTVE	0.210777	ACTVE	1.056091							
RITL	0.14342	USENS	0.149793	TOPIC	0.146924	OMAN	0.1524	ANXTY	0.183164	TOPIC	0.13387	ANXTY	0.165727	TOPIC	0.995869							
USENS	0.12529	TOPIC	0.118352	OMAN	0.142841	RITL	0.145154	IMGCR	0.140644	USENS	0.125881	USENS	0.123597	OMAN	0.972217							
ACTVE	0.120939	IMGCR	0.117126	IMGCR	0.112402	IMGCR	0.124409	USENS	0.098011	RITL	0.119749	TOPIC	0.091845	USENS	0.889653							
TOPIC	0.109559	ACTVE	0.057865	ANXTY	0.09473	ACTVE	0.115132	RITL	0.09701	OMAN	0.109602	IMGCR	0.082704	IMGCR	0.830584							
OMAN	0.101466	ANXTY	0.035292	USENS	0.076367	ANXTY	0.1111	OMAN	0.048179	IMGCR	0.091647	RITL	0.068891	ANXTY	0.771743							
ANXTY	0.044353	SELFINJ	0.029996	SELFINJ	0.020613	AGG	0.002908	AGG	0.006583	SELFINJ	0.078263	SELFINJ	0.026939	AGG	0.226668							
SELFINJ	0.005641	AGG	0.000197	AGG	0.020078	SELFINJ	0.000444	SELFINJ	7.08E-05	AGG	0.005995	AGG	0.003226	SELFINJ	0.161967							
module3 28-21																						
																					overall	
ACTVE	0.304095	RITL	0.285705	TOPIC	0.214116	TOPIC	0.18108	OMAN	0.317929	TOPIC	0.251874	RITL	1.207318									
OMAN	0.200779	USENS	0.211737	USENS	0.182361	USENS	0.159403	RITL	0.270238	USENS	0.209226	TOPIC	1.072866									
RITL	0.164105	TOPIC	0.177126	ACTVE	0.16275	OMAN	0.155274	USENS	0.185284	RITL	0.176893	USENS	1.061802									
TOPIC	0.147201	IMGCR	0.13447	RITL	0.159683	RITL	0.150694	TOPIC	0.101469	IMGCR	0.153073	OMAN	1.027877									
USENS	0.113791	OMAN	0.086198	OMAN	0.114729	IMGCR	0.148796	IMGCR	0.092354	OMAN	0.152967	ACTVE	0.732766									
IMGCR	0.063219	ACTVE	0.069843	ANXTY	0.091137	ACTVE	0.121772	ACTVE	0.032657	ACTVE	0.041649	IMGCR	0.65997									
ANXTY	0.006796	ANXTY	0.033648	IMGCR	0.068057	ANXTY	0.046761	AGG	2.8E-05	ANXTY	0.014297	ANXTY	0.19266									
SELFINJ	9.71E-06	AGG	0.001267	AGG	0.007157	AGG	0.036215	ANXTY	2.1E-05	AGG	1.17E-05	AGG	0.044682									
AGG	3.39E-06	SELFINJ	5.96E-06	SELFINJ	9.45E-06	SELFINJ	5.78E-06	SELFINJ	1.91E-05	SELFINJ	9.31E-06	SELFINJ	5.93E-05									
module3 28-22																						
																					overall	
ACTVE	0.206584	RITL	0.243704	ACTVE	0.270924	TOPIC	0.244223	AGG	0.227339	OMAN	0.243693	TOPIC	0.217024	TOPIC	1.101953							
TOPIC	0.193022	USENS	0.228357	IMGCR	0.178987	RITL	0.192483	OMAN	0.174842	ACTVE	0.184393	IMGCR	0.195865	ACTVE	1.049586							
IMGCR	0.13675	ACTVE	0.142584	USENS	0.127579	AGG	0.143646	USENS	0.149483	TOPIC	0.133449	USENS	0.139765	RITL	0.963679							
RITL	0.130948	OMAN	0.113209	OMAN	0.103154	USENS	0.131056	TOPIC	0.136222	RITL	0.126133	AGG	0.130857	OMAN	0.95904							
OMAN	0.12004	TOPIC	0.095875	AGG	0.084458	OMAN	0.128511	ACTVE	0.103139	USENS	0.117437	RITL	0.095411	USENS	0.950968							
AGG	0.095766	IMGCR	0.069615	TOPIC	0.082137	IMGCR	0.09403	RITL	0.095776	AGG	0.07507	ACTVE	0.080668	IMGCR	0.813335							
ANXTY	0.057892	AGG	0.055065	RITL	0.079224	ACTVE	0.061294	IMGCR	0.083398	ANXTY	0.064656	OMAN	0.075591	AGG	0.8122							
USENS	0.057291	ANXTY	0.04918	ANXTY	0.072128	ANXTY	0.004151	ANXTY	0.029556	IMGCR	0.05469	ANXTY	0.054822	ANXTY	0.332384							
SELFINJ	0.001708	SELFINJ	0.002411	SELFINJ	0.001409	SELFINJ	0.000606	SELFINJ	0.000244	SELFINJ	0.00048	SELFINJ	0.009998	SELFINJ	0.016856							

Module 4 Section 1 Weights of Hidden Layer Nodes Ranked

Module4 31-19

													overall
AINFO	0.2297507	AINFO	0.1923882	EGEST	0.2355262	AINFO	0.1865528	DGEST	0.248546	REPRT	0.1773935	STEREO	1.0414364
STEREO	0.1790846	DGEST	0.1807076	STEREO	0.189171	STEREO	0.1612652	STEREO	0.1745232	STEREO	0.1712442	EGEST	1.0268337
EGEST	0.1503326	EGEST	0.1699917	AINFO	0.1763351	EGEST	0.1386637	EGEST	0.1736931	EGEST	0.1586264	AINFO	0.9772069
REPRT	0.1084407	STEREO	0.1661482	REPRT	0.1250222	SPABN	0.1063817	AINFO	0.1072303	OINFO	0.1393649	DGEST	0.7697173
CONVS	0.1025832	OINFO	0.0794444	CONVS	0.1127311	CONVS	0.1045552	OINFO	0.0987817	DGEST	0.0935395	REPRT	0.6810886
OINFO	0.099918	REPRT	0.0742111	DGEST	0.0685313	REPRT	0.1044894	REPRT	0.0915317	CONVS	0.0883473	OINFO	0.5676273
DGEST	0.083677	SPABN	0.0701446	OINFO	0.0467873	OINFO	0.1033311	SPABN	0.0705228	SPABN	0.086486	CONVS	0.5102512
SPABN	0.0461762	CONVS	0.066912	SPABN	0.0458605	DGEST	0.0947159	CONVS	0.0351225	AINFO	0.0849498	SPABN	0.4255719
IECHO	2.772E-05	IECHO	3.712E-05	IECHO	2.522E-05	IECHO	3.5E-05	IECHO	4.153E-05	IECHO	3.904E-05	IECHO	0.0002056
OLANG	9.403E-06	OLANG	1.508E-05	OLANG	1.006E-05	OLANG	1.001E-05	OLANG	7.229E-06	OLANG	9.289E-06	OLANG	6.107E-05

Module4 31-20

													overall
REPRT	0.2161495	EGEST	0.2614778	STEREO	0.2203494	REPRT	0.2581238	AINFO	0.2119417	AINFO	0.2090849	AINFO	1.0433556
OINFO	0.1678316	AINFO	0.2055481	AINFO	0.1872468	STEREO	0.1777797	OINFO	0.1657792	STEREO	0.1882451	EGEST	1.0272648
EGEST	0.1655575	STEREO	0.1670807	EGEST	0.1806703	EGEST	0.1590605	REPRT	0.1577027	EGEST	0.1500349	STEREO	1.0039829
SPABN	0.1087307	DGEST	0.1107949	REPRT	0.172293	AINFO	0.1373008	STEREO	0.1459763	OINFO	0.1271606	REPRT	0.9637213
STEREO	0.1045517	OINFO	0.1041714	OINFO	0.0961663	OINFO	0.1327707	CONVS	0.1262451	CONVS	0.1247903	OINFO	0.7938798
DGEST	0.0987078	REPRT	0.072802	DGEST	0.0667476	SPABN	0.0599398	EGEST	0.1104638	REPRT	0.0866503	DGEST	0.3996454
AINFO	0.0922333	SPABN	0.0595377	SPABN	0.0474919	CONVS	0.0386607	SPABN	0.0517125	DGEST	0.0569929	SPABN	0.3843696
CONVS	0.0462036	CONVS	0.0184728	CONVS	0.0289163	DGEST	0.0363059	DGEST	0.0300964	SPABN	0.056957	CONVS	0.3832887
IECHO	2.084E-05	IECHO	9.535E-05	IECHO	8.236E-05	IECHO	5.027E-05	IECHO	5.258E-05	IECHO	4.942E-05	IECHO	0.0003508
OLANG	1.337E-05	OLANG	1.938E-05	OLANG	3.614E-05	OLANG	7.839E-06	OLANG	2.974E-05	OLANG	3.446E-05	OLANG	0.0001409

Module4 31-21

													overall
EGEST	0.15758	AINFO	0.2190935	OINFO	0.2012927	CONVS	0.2201565	EGEST	0.1821063	AINFO	0.1998138	STEREO	0.168114	STEREO	1.0325627
STEREO	0.1541517	EGEST	0.1741799	AINFO	0.1729685	DGEST	0.1740546	DGEST	0.1493773	EGEST	0.1616301	CONVS	0.1588144	AINFO	1.0259329
DGEST	0.153669	STEREO	0.1347098	STEREO	0.1535778	STEREO	0.1568517	STEREO	0.1446443	SPABN	0.1314728	EGEST	0.1445078	EGEST	0.9902743
REPRT	0.1530547	SPABN	0.1282015	SPABN	0.142265	AINFO	0.1137017	OINFO	0.1328566	OINFO	0.128776	AINFO	0.1291029	CONVS	0.8711227
SPABN	0.1083013	REPRT	0.1107064	CONVS	0.10553	REPRT	0.095519	REPRT	0.1224229	STEREO	0.1205133	SPABN	0.1126514	SPABN	0.8037316
CONVS	0.1077025	CONVS	0.1045641	EGEST	0.1000015	SPABN	0.0858695	SPABN	0.0949701	REPRT	0.1199846	DGEST	0.1050899	OINFO	0.8027082
AINFO	0.0992235	OINFO	0.0980543	REPRT	0.0754618	OINFO	0.0835449	AINFO	0.0920291	CONVS	0.092803	OINFO	0.0919002	REPRT	0.7669283
OINFO	0.0662835	DGEST	0.030455	DGEST	0.0488702	EGEST	0.0702688	CONVS	0.0815522	DGEST	0.0449811	REPRT	0.0897789	DGEST	0.706497
IECHO	2.182E-05	IECHO	2.587E-05	IECHO	2.338E-05	IECHO	2.182E-05	IECHO	3.229E-05	IECHO	1.439E-05	IECHO	2.86E-05	IECHO	0.0001682
OLANG	1.204E-05	OLANG	9.544E-06	OLANG	9.159E-06	OLANG	1.149E-05	OLANG	9.058E-06	OLANG	1.103E-05	OLANG	1.188E-05	OLANG	7.421E-05

Module4 31-22

													overall
STEREO	0.2108373	EGEST	0.1621878	DGEST	0.1900874	EGEST	0.189823	DGEST	0.1893132	OINFO	0.1743366	STEREO	0.1838364	STEREO	1.144481
DGEST	0.1619517	AINFO	0.1596103	EGEST	0.1633663	REPRT	0.1755624	STEREO	0.1801496	REPRT	0.16727	REPRT	0.1633007	DGEST	0.9947851
AINFO	0.1615612	STEREO	0.1586714	AINFO	0.142207	STEREO	0.131957	AINFO	0.1301465	AINFO	0.1484654	DGEST	0.1496469	REPRT	0.9666453
OINFO	0.118467	REPRT	0.1323872	STEREO	0.1394445	AINFO	0.1214228	CONVS	0.1153946	STEREO	0.1395848	CONVS	0.115785	AINFO	0.9605183
REPRT	0.0998192	DGEST	0.1305801	CONVS	0.1296375	DGEST	0.1040661	REPRT	0.1133882	SPABN	0.1201574	EGEST	0.1074377	EGEST	0.8831272
CONVS	0.0970067	OINFO	0.1062902	REPRT	0.1144238	CONVS	0.1027471	OINFO	0.1040577	CONVS	0.0994642	AINFO	0.0971051	OINFO	0.7365292
EGEST	0.0861979	SPABN	0.0899261	SPABN	0.0780065	OINFO	0.093676	EGEST	0.0925611	EGEST	0.0815534	OINFO	0.0969014	CONVS	0.7203519
SPABN	0.064101	CONVS	0.0603168	OINFO	0.0428003	SPABN	0.0807214	SPABN	0.0744682	DGEST	0.0691397	SPABN	0.0859508	SPABN	0.5933315
IECHO	5.111E-05	IECHO	1.973E-05	OLANG	1.4E-05	IECHO	1.72E-05	IECHO	2.022E-05	IECHO	2.205E-05	IECHO	2.837E-05	IECHO	0.0001714
OLANG	6.958E-06	OLANG	1.02E-05	IECHO	1.269E-05	OLANG	7.031E-06	OLANG	6.763E-06	OLANG	6.492E-06	OLANG	7.566E-06	OLANG	5.9E-05

Module 4 Section 2 Weights of Hidden Layer Nodes Ranked

Module4 31-19

																					overall	
EMPTH	0.1528493	FACEO	0.1563426	RESP	0.1678412	OQRAP	0.1581921	FACEO	0.1402426	CAFF	0.1491274	LLNVC	0.1968172	EMPTH	0.8547313							
CAFF	0.1483383	LLNVC	0.1372513	CAFF	0.138366	INSIG	0.12174	RESP	0.1396313	EMPTH	0.1179232	EMPTH	0.1450615	RESP	0.8451745							
OQRAP	0.1280676	EMPTH	0.1341283	SEI	0.1264781	SEI	0.1150582	SEI	0.1135202	FACEO	0.1165457	OQRAP	0.1352448	OQRAP	0.8321057							
RESP	0.1151746	RESP	0.1288454	OQRAP	0.1045303	EMPTH	0.1148852	CAFF	0.1119721	SEI	0.1105337	SEI	0.1075972	LLNVC	0.8138742							
FACEO	0.1078798	OQRAP	0.1174246	EMPTH	0.102785	CAFF	0.1017485	LLNVC	0.1078329	OQRAP	0.1089047	RESP	0.1031681	CAFF	0.7602484							
LLNVC	0.0890278	SEI	0.1005535	LLNVC	0.0941224	RESP	0.0990603	EMPTH	0.0870987	LLNVC	0.1054092	FACEO	0.0698379	SEI	0.7563319							
SEI	0.082591	INSIG	0.0641611	INSIG	0.0895384	LLNVC	0.0834134	OQRAP	0.0797416	RESP	0.0914536	ARSOC	0.0696383	FACEO	0.7407451							
INSIG	0.0759478	CAFF	0.0627666	FACEO	0.0718503	FACEO	0.0780463	ARSOC	0.0741807	QSRES	0.0688857	INSIG	0.0647023	INSIG	0.553504							
QSRES	0.0481723	QSRES	0.0515561	ARSOC	0.0530509	QSRES	0.0592249	INSIG	0.0723883	INSIG	0.065026	QSRES	0.0487409	ARSOC	0.3976327							
ARSOC	0.0479844	ARSOC	0.0334647	QSRES	0.0438996	ARSOC	0.0573778	QSRES	0.0573489	ARSOC	0.0619358	CAFF	0.0479295	QSRES	0.3778285							
QSOV	0.0037734	QSOV	0.0132478	QSOV	0.0073246	QSOV	0.0111364	QSOV	0.0160061	QSOV	0.0041735	QSOV	0.0110229	QSOV	0.0666848							
UEYE	0.0001936	UEYE	0.0002578	UEYE	0.0002132	UEYE	0.0001169	UEYE	3.667E-05	UEYE	8.15E-05	UEYE	0.0002393	UEYE	0.001139							

Module4 31-20

																					overall	
INSIG	0.1343736	OQRAP	0.1526663	LLNVC	0.1594397	LLNVC	0.1610886	FACEO	0.1445497	FACEO	0.1535148	SEI	0.1395978	SEI	0.8738891							
SEI	0.1268889	INSIG	0.1433401	SEI	0.1393888	RESP	0.1296148	EMPTH	0.1375367	SEI	0.1436142	OQRAP	0.1320904	OQRAP	0.853294							
LLNVC	0.1209608	EMPTH	0.1046975	OQRAP	0.1243182	SEI	0.1257232	CAFF	0.1290749	EMPTH	0.1188429	FACEO	0.1223896	LLNVC	0.8154559							
OQRAP	0.1209367	FACEO	0.1045304	RESP	0.1213847	CAFF	0.1181268	LLNVC	0.1261159	OQRAP	0.1129175	RESP	0.1089201	FACEO	0.8047647							
CAFF	0.1205446	QSOV	0.1041065	INSIG	0.1109071	OQRAP	0.1132564	INSIG	0.1197846	INSIG	0.1078898	INSIG	0.1080183	INSIG	0.7942029							
EMPTH	0.0870924	SEI	0.0990379	CAFF	0.1035798	FACEO	0.1044999	SEI	0.0996382	LLNVC	0.0914061	LLNVC	0.0877693	CAFF	0.7042684							
RESP	0.0837201	CAFF	0.0839243	FACEO	0.1010238	EMPTH	0.0744785	OQRAP	0.0971084	RESP	0.0863012	CAFF	0.0724772	RESP	0.6716137							
FACEO	0.0742565	RESP	0.075633	EMPTH	0.0790499	INSIG	0.0698894	RESP	0.0660397	QSOV	0.0767585	QSRES	0.0663904	EMPTH	0.6647909							
QSRES	0.0599531	LLNVC	0.0686754	QSOV	0.0436454	QSRES	0.0609945	QSRES	0.0398399	CAFF	0.0765407	EMPTH	0.0630929	QSOV	0.3863583							
QSOV	0.0554713	QSRES	0.0462058	QSRES	0.0137625	QSOV	0.0310771	QSOV	0.0302246	ARSOC	0.0160082	ARSOC	0.053514	QSRES	0.3028833							
ARSOC	0.0151163	ARSOC	0.0159224	ARSOC	0.0027035	ARSOC	0.0104671	ARSOC	0.0095868	QSRES	0.015737	QSOV	0.045075	ARSOC	0.1233183							
UEYE	0.0006857	UEYE	0.0012603	UEYE	0.0007965	UEYE	0.0007837	UEYE	0.0005004	UEYE	0.000469	UEYE	0.0006649	UEYE	0.0051606							

Module4 31-21

																					overall	
RESP	0.1122439	LLNVC	0.1469906	SEI	0.13622	QSRES	0.1315087	SEI	0.1276033	SEI	0.1663025	SEI	0.1243625	RESP	0.1607484	SEI	0.980281					
SEI	0.1120552	EMPTH	0.1243064	QSOV	0.1101371	OQRAP	0.1264306	OQRAP	0.1192764	QSOV	0.1417424	RESP	0.1210219	LLNVC	0.1241553	RESP	0.8676858					
QSOV	0.1055998	RESP	0.1123496	RESP	0.1083365	QSOV	0.1252218	LLNVC	0.1136384	RESP	0.1185312	LLNVC	0.1105376	OQRAP	0.101614	QSOV	0.8521725					
LLNVC	0.1042845	CAFF	0.1106834	EMPTH	0.1078472	SEI	0.1141717	QSOV	0.1115491	EMPTH	0.1117143	OQRAP	0.1043301	SEI	0.0992924	LLNVC	0.8120788					
CAFF	0.1017831	SEI	0.1002734	CAFF	0.1039992	EMPTH	0.1071399	ARSOC	0.1019131	CAFF	0.0851525	QSRES	0.0949794	CAFF	0.0984539	OQRAP	0.7785777					
OQRAP	0.0964735	INSIG	0.0894009	FACEO	0.093549	ARSOC	0.0959216	CAFF	0.0971152	QSRES	0.0756315	QSOV	0.089067	QSOV	0.0824907	CAFF	0.7359024					
QSRES	0.0928015	QSOV	0.0863646	LLNVC	0.0923259	RESP	0.0853333	EMPTH	0.0912351	OQRAP	0.0705327	FACEO	0.0812298	INSIG	0.0779939	EMPTH	0.7349515					
ARSOC	0.0867313	OQRAP	0.0842541	OQRAP	0.0756663	CAFF	0.0835723	INSIG	0.0809587	INSIG	0.0697097	INSIG	0.0776976	ARSOC	0.0739824	QSRES	0.6285068					
EMPTH	0.0680356	ARSOC	0.0551366	QSRES	0.0646106	LLNVC	0.053088	QSRES	0.0808467	LLNVC	0.0670585	ARSOC	0.0766659	FACEO	0.0715286	ARSOC	0.6214848					
FACEO	0.0602098	FACEO	0.0520295	ARSOC	0.0645753	FACEO	0.04803	RESP	0.049121	ARSOC	0.0665586	EMPTH	0.0649358	EMPTH	0.0597373	INSIG	0.5277332					
INSIG	0.0597438	QSRES	0.0381545	INSIG	0.0427128	INSIG	0.0295158	FACEO	0.026676	FACEO	0.0270434	CAFF	0.0551429	QSRES	0.0499739	FACEO	0.4602961					
UEYE	3.788E-05	UEYE	5.657E-05	UEYE	2.012E-05	UEYE	6.632E-05	UEYE	6.691E-05	UEYE	2.278E-05	UEYE	2.957E-05	UEYE	2.922E-05	UEYE	0.0003294					

Module4 31-22

																					overall	
LLNVC	0.1711885	FACEO	0.1409945	SEI	0.1253605	RESP	0.1314929	SEI	0.1247954	RESP	0.1386274	CAFF	0.1402509	OQRAP	0.1329267	SEI	0.1104449	RESP	0.9589826			
RESP	0.1363708	EMPTH	0.1226113	EMPTH	0.1118081	SEI	0.1176715	CAFF	0.1180701	INSIG	0.1187075	ARSOC	0.1219155	LLNVC	0.1160767	CAFF	0.1089601	CAFF	0.9565551			
CAFF	0.0967733	CAFF	0.1140195	RESP	0.109153	QSRES	0.1172983	RESP	0.1165982	QSOV	0.0952823	EMPTH	0.1094317	CAFF	0.1042775	FACEO	0.1082504	SEI	0.8861181			
SEI	0.0878518	INSIG	0.0971655	QSOV	0.1028946	ARSOC	0.1106505	OQRAP	0.0935295	CAFF	0.0939372	OQRAP	0.1052462	INSIG	0.0983623	QSOV	0.0931208	OQRAP	0.8529463			
EMPTH	0.0826362	ARSOC	0.0950988	FACEO	0.1027797	OQRAP	0.1103645	EMPTH	0.0932283	FACEO	0.091066	FACEO	0.0992471	SEI	0.0971092	RESP	0.0921562	LLNVC	0.8512037			
ARSOC	0.081295	OQRAP	0.0893593	LLNVC	0.1022458	INSIG	0.1061403	LLNVC	0.0917212	ARSOC	0.0887015	INSIG	0.0870642	FACEO	0.0959445	OQRAP	0.0916003	FACEO	0.840444			
INSIG	0.0778785	SEI	0.0847919	CAFF	0.0995991	CAFF	0.0806673	QSRES	0.0850386	OQRAP	0.0857834	LLNVC	0.0866569	EMPTH	0.0835237	QSRES	0.0882311	EMPTH	0.8121906			
FACEO	0.0750504	RESP	0.0840031	OQRAP	0.0792515	LLNVC	0.0698622	INSIG	0.0831658	QSRES	0.0846611	RESP	0.0742217	QSRES	0.0809618	ARSOC	0.0876059	INSIG	0.7936862			
QSRES	0.0676926	QSOV	0.0763921	INSIG	0.0674928	EMPTH	0.066527	FACEO	0.0793387	SEI	0.0741786	QSOV	0.0699784	RESP	0.0763593	LLNVC	0.0862154	ARSOC	0.7616983			
OQRAP	0.0648848	LLNVC	0.0649794	ARSOC	0.0589735	FACEO	0.0477727	ARSOC	0.0681076	EMPTH	0.0667612	SEI	0.0639142	QSOV	0.0650672	EMPTH	0.0756631	QSOV	0.6489708			
QSOV	0.0583332	QSRES	0.0305585	QSRES	0.0404176	QSOV	0.0415169	QSOV	0.0463852	LLNVC	0.0622576	QSRES	0.0419275	ARSOC	0.0493499	INSIG	0.0577092	QSRES	0.636787			
UEYE	4.491E-05	UEYE	2.604E-05	UEYE	2.384E-05	UEYE	3.587E-05	UEYE	2.115E-05	UEYE	3.616E-05	UEYE	0.0001458	UEYE	4.105E-05	UEYE	4.249E-05	UEYE	0.0004173			

Module 4 Section 3 Weights of Hidden Layer Nodes Ranked

Module4 31-19																
																overall
SELFINJ	0.2834081	SELFINJ	0.438736	SELFINJ	0.4211943	OMAN	0.2408709	SELFINJ	0.179364	SELFINJ	0.4442385	SELFINJ	1.8899426
RITL	0.1464073	ANXTY	0.1826181	TOPIC	0.1375974	IMGCR	0.2244698	OMAN	0.1746003	OMAN	0.1330907	ANXTY	0.7654468
ANXTY	0.1005635	TOPIC	0.101333	AGG	0.1211242	ANXTY	0.1862258	TOPIC	0.16381	RITL	0.1239196	OMAN	0.7540116
IMGCR	0.0963462	RITL	0.0850327	ANXTY	0.0696538	SELFINJ	0.1230018	AGG	0.1202768	ANXTY	0.1218808	TOPIC	0.6388695
TOPIC	0.0946899	OMAN	0.0657677	ACTVE	0.0615452	USENS	0.0879991	ANXTY	0.1045047	TOPIC	0.0640261	IMGCR	0.56097
OMAN	0.0860745	IMGCR	0.0371972	IMGCR	0.0590606	TOPIC	0.0774131	IMGCR	0.0968851	IMGCR	0.0470111	RITL	0.4889844
USENS	0.0758526	USENS	0.0358731	RITL	0.0576474	RITL	0.0522668	USENS	0.0841542	ACTVE	0.0351082	AGG	0.3477311
ACTVE	0.0635433	AGG	0.0268391	OMAN	0.0536075	AGG	0.0077394	ACTVE	0.0526942	AGG	0.0186372	USENS	0.3145365
AGG	0.0531145	ACTVE	0.0266031	USENS	0.0185696	ACTVE	1.334E-05	RITL	0.0237106	USENS	0.0120879	ACTVE	0.2395073

Module4 31-20																
																overall
SELFINJ	0.3074028	SELFINJ	0.3954847	SELFINJ	0.5436292	SELFINJ	0.229584	SELFINJ	0.5452995	SELFINJ	0.4614715	SELFINJ	2.4828717
OMAN	0.141894	OMAN	0.1502384	USENS	0.1169944	TOPIC	0.2092253	OMAN	0.1072666	USENS	0.1683111	USENS	0.7314222
USENS	0.1365758	USENS	0.1342487	RITL	0.0873498	OMAN	0.1491155	RITL	0.0997546	ANXTY	0.0983558	OMAN	0.6782585
ANXTY	0.1232733	ANXTY	0.0930886	TOPIC	0.0858807	USENS	0.1371945	ANXTY	0.0588815	OMAN	0.0652819	TOPIC	0.5141064
RITL	0.1055953	TOPIC	0.0766632	OMAN	0.0644622	IMGCR	0.1085669	IMGCR	0.0452407	RITL	0.0633901	ANXTY	0.4635996
AGG	0.0742288	RITL	0.0521719	ANXTY	0.0575265	ACTVE	0.0649821	AGG	0.0446007	TOPIC	0.0626938	RITL	0.4302572
IMGCR	0.0734482	IMGCR	0.0374048	IMGCR	0.0271887	AGG	0.0468622	TOPIC	0.0428143	IMGCR	0.0487015	IMGCR	0.3405509
TOPIC	0.0368291	AGG	0.0332034	AGG	0.0161934	ANXTY	0.032474	USENS	0.0380978	AGG	0.0175706	AGG	0.2326592
ACTVE	0.0007527	ACTVE	0.0274963	ACTVE	0.0007751	RITL	0.0219956	ACTVE	0.0180444	ACTVE	0.0142237	ACTVE	0.1262743

Module4 31-21																
																overall
SELFINJ	0.3221787	OMAN	0.3266598	SELFINJ	0.3234867	SELFINJ	0.4204799	OMAN	0.2942364	TOPIC	0.2694989	SELFINJ	1.284811
TOPIC	0.1263446	TOPIC	0.2150514	USENS	0.2205873	ANXTY	0.1628786	TOPIC	0.1419005	IMGCR	0.2058422	OMAN	1.0263573
RITL	0.1257156	RITL	0.1032339	RITL	0.185223	OMAN	0.0864005	USENS	0.1327349	OMAN	1.61E-01	TOPIC	0.8550035
ANXTY	0.1184199	USENS	0.0971518	ANXTY	0.1014927	USENS	0.0848624	RITL	0.13047	USENS	0.1229494	USENS	0.7523281
OMAN	0.1032038	ANXTY	0.0807439	TOPIC	0.0684607	RITL	0.0840736	ANXTY	0.1284701	ANXTY	0.1084499	ANXTY	0.7004552
USENS	0.0940422	IMGCR	0.0745043	OMAN	0.0544984	ACTVE	0.0798863	SELFINJ	0.0821138	SELFINJ	0.0886872	RITL	0.6632305
IMGCR	0.0832468	SELFINJ	0.0478647	IMGCR	0.0426991	TOPIC	0.0337474	IMGCR	0.0510829	RITL	0.0345144	IMGCR	0.4846526
ACTVE	0.0169636	ACTVE	0.0309272	AGG	0.0026083	IMGCR	0.0272774	ACTVE	0.0319604	ACTVE	0.0080095	ACTVE	0.1686906
AGG	0.0098848	AGG	0.0238629	ACTVE	0.0009436	AGG	0.0203941	AGG	0.007031	AGG	0.0006901	AGG	0.0644711

Module4 31-22																
																overall
SELFINJ	0.3835633	SELFINJ	0.2414337	SELFINJ	0.3680713	SELFINJ	0.5271743	SELFINJ	0.2441979	OMAN	0.1784219	SELFINJ	1.9368048
RITL	0.2322877	TOPIC	0.2125475	OMAN	0.140928	RITL	0.1177442	RITL	0.1749732	SELFINJ	0.1723644	TOPIC	0.7592361
TOPIC	0.1533431	ANXTY	0.1581507	ANXTY	0.1095802	ANXTY	0.0977654	OMAN	0.1605791	TOPIC	0.1586547	OMAN	0.757716
ANXTY	0.0941404	IMGCR	0.1381138	TOPIC	0.099203	IMGCR	0.0931978	TOPIC	0.1095215	USENS	0.1547817	RITL	0.701191
OMAN	0.0736038	OMAN	0.1228268	RITL	0.0798848	OMAN	0.0813564	IMGCR	0.1032638	ANXTY	0.1054509	ANXTY	0.6642912
USENS	0.025166	USENS	0.0419588	IMGCR	0.0700385	ACTVE	0.042969	ANXTY	0.0992037	IMGCR	0.1054233	IMGCR	0.5276166
IMGCR	0.0175795	RITL	0.0393446	ACTVE	0.0672381	TOPIC	0.0259663	USENS	0.074452	ACTVE	0.0661257	USENS	0.3636462
ACTVE	0.0143959	AGG	0.0291111	USENS	0.064392	AGG	0.0109311	ACTVE	0.0335183	RITL	0.0569565	ACTVE	0.2407601
AGG	0.0059203	ACTVE	0.0165131	AGG	0.000664	USENS	0.0028956	AGG	0.0002904	AGG	0.001821	AGG	0.0487379

APPENDIX F

MODEL CODE

Building module 1 Autoencoder with Hidden layer of size 20

```
#import all dependencies
import pandas
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import torch
from torch.autograd import Variable
from torch import nnfrom torch.optim import lr_scheduler
from torch.nn import Parameter
import numpy as np
import math
#Create mask 29-20 matrix
mask2 = np.matrix('1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1
1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1
1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
1 1 1 1 1 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0; 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 1 1 1 1
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0; 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 1 1
1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0 0; 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1; 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1')
#create transposed mask
mask1 = mask2.transpose()
torch_mask1 = torch.from_numpy(mask1)    #convert mask to torch tensor
torchmask1 = torch_mask1.type(torch.FloatTensor)    #match tensor type with
weights
torchmask1 = torchmask1.cuda()    #convert to GPU CUDA tensor
mask1 = torch.autograd.Variable(torchmask1, requires_grad=True)  #wrap in
variable to add to model
torch_mask2 = torch.from_numpy(mask2)    #convert mask to torch tensor
torchmask2 = torch_mask2.type(torch.FloatTensor)    #match tensor type with
weights
torchmask2 = torchmask2.cuda()    #convert to GPU CUDA tensor
mask2 = torch.autograd.Variable(torchmask2, requires_grad=True)    #wrap in
variable to add to model
#create special masked layer for encoder by altering feedforward layer's forward
pass
class MaskedLayer1(nn.Module):
   def __init__(self, in_features, out_features, bias=True):
```

```
    super(MaskedLayer1, self).__init__()
    self.in_features = in_features
    self.out_features = out_features
    self.weight = Parameter(torch.Tensor(in_features, out_features))
    if bias:
        self.bias = Parameter(torch.Tensor(out_features))
    else:
        self.register_parameter('bias', None)
    self.reset_parameters()
def reset_parameters(self):
    stdv = 1. / math.sqrt(self.weight.size(1))
    self.weight.data.uniform_(-stdv, stdv)
    if self.bias is not None:
        self.bias.data.uniform_(-stdv, stdv)
def forward(self, input):
    return input.mm(self.weight*mask1)
def __repr__(self):
    return self.__class__.__name__ + ' ('
        + str(self.in_features) + ' -> '
        + str(self.out_features) + ')'
```

```
#create special masked layer for decoder by altering feedforward layer's forward
pass
class MaskedLayer2(nn.Module):
    def __init__(self, in_features, out_features, bias=True):
        super(MaskedLayer2, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()
    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)

    def forward(self, input):
        return input.mm(self.weight*mask2)
    def __repr__(self):
```

```
        return self.__class__.__name__ + ' ('
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'
dtype = torch.cuda.FloatTensor
N, D_in, H1, D_out = 10, 29, 20, 29    #batch size, dimensions of network
#randomly initialize weights with matching tensor type to mask
w1 = Variable(torch.randn(D_in, H1).type(dtype), requires_grad=True)
w2 = Variable(torch.randn(H1, D_out).type(dtype), requires_grad=True)
#import data from csv with pandas, strip values from dataframe into array
dataframe = pandas.read_csv("ADOSModule 1.csv", header=0)
dataset = dataframe.values
#model configuration, dimensions, and hyperparameters
activation = torch.nn.SELU()
model = torch.nn.Sequential(MaskedLayer1(29, 20), #incoder
                            activation,
                            MaskedLayer2(20,29))#decoder
model = model.cuda()
criterion = torch.nn.L1Loss(size_average=True)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

#create arrays to store loss measurements for training and test sets
trainingloss =[]
```

```
trainingloss = np.array(trainingloss, dtype =np.float64)
testloss = []
trainingloss = np.array(testloss, dtype =np.float64)
#10 folds for cross validation, data set is split 90/10 training test, placed into
tensor on GPU
for i in range(10):
    train, test = train_test_split(dataset, test_size=0.1)
    Module 1train = torch.from_numpy(train)
    Module 1test = torch.from_numpy(test)
    Module 1train = Module 1train.type(torch.FloatTensor)
    Module 1train = Module 1train.cuda()
    Module 1test = Module 1test.type(torch.FloatTensor)
    Module 1test = Module 1test.cuda()
    x = torch.autograd.Variable(Module 1train)     #input
    y = torch.autograd.Variable(Module 1train, requires_grad=False)     #target,
same as input
    testset = torch.autograd.Variable(Module 1test, requires_grad = False)     #test
set
    #100,000 epochs per validation fold, model minimizes difference between
target and prediction, training and test losses added to respective arrays every
epoch
        for t in range(100000):
```

```
        y_pred = model(x)
        loss = criterion(y_pred, y)
        print(i, t, loss.data[0])
        trainingloss = np.append(trainingloss, loss.data[0])
        optimizer.zero_grad()       loss.backward()
        optimizer.step()
    testing = model(testset)
    val_loss = criterion(testing, testset)
    print(val_loss.data[0])
    testloss = np.append(testloss,val_loss.data[0])
#remove encoder weights from model by iterating through model.parameters and
removing data; break loop after encoder weights removed
#CPU tensor to numpy array to pandas dataframe to comma-separated value file
for param in model.parameters():
    cpuweights = param.data.cpu()
    npweights = cpuweights.numpy()
    df = pandas.DataFrame(npweights)
    df.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module 1/training1/Module
1weights.csv")
    break
#take reconstruction (90% of data), move it to CPU tensor, then to numpy array,
then to pandas dataframe, then to csv with headers listed
```

```
reconstruction = y_pred.data.cpu()
reconstruction = y_pred.data.numpy()
reconstructionset = pandas.DataFrame(reconstruction, columns=
['OLANG','FVOC','INTON','IECHO','STEREO','UOTHR','POINT','GEST','UEYE','S
SMLE','FACEO','GZSOV','SHRNJ','RNAME','REQ','GIVE','SHOW','SIJNT','RJNT'
,'QSOV','FPLAY','IMGCR','USENS','OMAN','SELFINJ','URBEH','ACTVE','AGG','A
NXTY' ])
reconstructionset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
1/training1/Module 1_reconstruction.csv")
#target goes through same pipeline as reconstruction
target = y.data.cpu().numpy()
targetset = pandas.DataFrame(target, columns=
['OLANG','FVOC','INTON','IECHO','STEREO','UOTHR','POINT','GEST','UEYE','S
SMLE','FACEO','GZSOV','SHRNJ','RNAME','REQ','GIVE','SHOW','SIJNT','RJNT'
,'QSOV','FPLAY','IMGCR','USENS','OMAN','SELFINJ','URBEH','ACTVE','AGG','A
NXTY' ])
targetset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
1/training1/Module 1_y.csv")
#testing reconstruction
test = testset.data.cpu().numpy()
testset = pandas.DataFrame(test, columns=
['OLANG','FVOC','INTON','IECHO','STEREO','UOTHR','POINT','GEST','UEYE','S
```

```
SMLE','FACEO','GZSOV','SHRNJ','RNAME','REQ','GIVE','SHOW','SIJNT','RJNT'
,'QSOV','FPLAY','IMGCR','USENS','OMAN','SELFINJ','URBEH','ACTVE','AGG','A
NXTY' ])
testset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
1/training1/Module 1_testset.csv")
#testing target
tests = testing.data.cpu().numpy()
testingset = pandas.DataFrame(tests, columns=
['OLANG','FVOC','INTON','IECHO','STEREO','UOTHR','POINT','GEST','UEYE','S
SMLE','FACEO','GZSOV','SHRNJ','RNAME','REQ','GIVE','SHOW','SIJNT','RJNT'
,'QSOV','FPLAY','IMGCR','USENS','OMAN','SELFINJ','URBEH','ACTVE','AGG','A
NXTY' ])
testingset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
1/training1/Module 1_testeing.csv")
```

Building module 1 Autoencoder with Hidden layer of size 21

```
#import all dependencies
import pandas
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import torch
from torch.autograd import Variable
from torch import nnfrom torch.optim import lr_scheduler
from torch.nn import Parameter
import numpy as np
import math
#Create mask 29-21 matrix
mask2 = np.matrix('1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1
1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1
1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
1 1 1 1 1 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0; 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 1 1 1 1
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0; 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 1 1
1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0 0 0; 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1; 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1; 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1')
#create transposed mask
mask1 = mask2.transpose()
torch_mask1 = torch.from_numpy(mask1)    #convert mask to torch tensor
torchmask1 = torch_mask1.type(torch.FloatTensor)    #match tensor type with
weights
torchmask1 = torchmask1.cuda()    #convert to GPU CUDA tensor
mask1 = torch.autograd.Variable(torchmask1, requires_grad=True)  #wrap in
variable to add to model
torch_mask2 = torch.from_numpy(mask2)    #convert mask to torch tensor
torchmask2 = torch_mask2.type(torch.FloatTensor)    #match tensor type with
weights
torchmask2 = torchmask2.cuda()    #convert to GPU CUDA tensor
mask2 = torch.autograd.Variable(torchmask2, requires_grad=True)    #wrap in
variable to add to model
#create special masked layer for encoder by altering feedforward layer's forward
pass
class MaskedLayer1(nn.Module):
```

```
def __init__(self, in_features, out_features, bias=True):
    super(MaskedLayer1, self).__init__()
    self.in_features = in_features
    self.out_features = out_features
    self.weight = Parameter(torch.Tensor(in_features, out_features))
    if bias:
        self.bias = Parameter(torch.Tensor(out_features))
    else:
        self.register_parameter('bias', None)
    self.reset_parameters()
def reset_parameters(self):
    stdv = 1. / math.sqrt(self.weight.size(1))
    self.weight.data.uniform_(-stdv, stdv)
    if self.bias is not None:
        self.bias.data.uniform_(-stdv, stdv)
def forward(self, input):
    return input.mm(self.weight*mask1)
def __repr__(self):
    return self.__class__.__name__ + ' ('
        + str(self.in_features) + ' -> '
        + str(self.out_features) + ')'
```

```
#create special masked layer for decoder by altering feedforward layer's forward
pass
class MaskedLayer2(nn.Module):
    def __init__(self, in_features, out_features, bias=True):
        super(MaskedLayer2, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()
    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)
    def forward(self, input):
        return input.mm(self.weight*mask2)
    def __repr__(self):
```

```
        return self.__class__.__name__ + ' ('
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'
dtype = torch.cuda.FloatTensor
N, D_in, H1, D_out = 10, 29, 21, 29    #batch size, dimensions of network
#randomly initialize weights with matching tensor type to mask
w1 = Variable(torch.randn(D_in, H1).type(dtype), requires_grad=True)
w2 = Variable(torch.randn(H1, D_out).type(dtype), requires_grad=True)
#import data from csv with pandas, strip values from dataframe into array
dataframe = pandas.read_csv("ADOSModule 1.csv", header=0)
dataset = dataframe.values
#model configuration, dimensions, and hyperparameters
activation = torch.nn.SELU()
model = torch.nn.Sequential(MaskedLayer1(29, 21), #incoder
                            activation,
                            MaskedLayer2(21,29))#decoder
model = model.cuda()
criterion = torch.nn.L1Loss(size_average=True)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
#create arrays to store loss measurements for training and test sets
trainingloss =[]
trainingloss = np.array(trainingloss, dtype =np.float64)
```

```
testloss = []
trainingloss = np.array(testloss, dtype =np.float64)
#10 folds for cross validation, data set is split 90/10 training test, placed into
tensor on GPU
for i in range(10):
    train, test = train_test_split(dataset, test_size=0.1)
    Module 1train = torch.from_numpy(train)
    Module 1test = torch.from_numpy(test)
    Module 1train = Module 1train.type(torch.FloatTensor)
    Module 1train = Module 1train.cuda()
    Module 1test = Module 1test.type(torch.FloatTensor)
    Module 1test = Module 1test.cuda()
    x = torch.autograd.Variable(Module 1train)     #input
    y = torch.autograd.Variable(Module 1train, requires_grad=False)     #target,
same as input
    testset = torch.autograd.Variable(Module 1test, requires_grad = False)     #test
set
    #100,000 epochs per validation fold, model minimizes difference between
target and prediction, training and test losses added to respective arrays every
epoch
        for t in range(100000):
        y_pred = model(x)
```

```
        loss = criterion(y_pred, y)
        print(i, t, loss.data[0])
        trainingloss = np.append(trainingloss, loss.data[0])
        optimizer.zero_grad()       loss.backward()
        optimizer.step()
    testing = model(testset)
    val_loss = criterion(testing, testset)
    print(val_loss.data[0])
    testloss = np.append(testloss,val_loss.data[0])
#remove encoder weights from model by iterating through model.parameters and
removing data; break loop after encoder weights removed
#CPU tensor to numpy array to pandas dataframe to comma-separated value file
for param in model.parameters():
    cpuweights = param.data.cpu()
    npweights = cpuweights.numpy()
    df = pandas.DataFrame(npweights)
    df.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module 1/training1/Module
1weights.csv")
    break
#take reconstruction (90% of data), move it to CPU tensor, then to numpy array,
then to pandas dataframe, then to csv with headers listed
reconstruction = y_pred.data.cpu()
```

```
reconstruction = y_pred.data.numpy()
reconstructionset = pandas.DataFrame(reconstruction, columns=
['OLANG','FVOC','INTON','IECHO','STEREO','UOTHR','POINT','GEST','UEYE','S
SMLE','FACEO','GZSOV','SHRNJ','RNAME','REQ','GIVE','SHOW','SIJNT','RJNT'
,'QSOV','FPLAY','IMGCR','USENS','OMAN','SELFINJ','URBEH','ACTVE','AGG','A
NXTY' ])
reconstructionset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
1/training1/Module 1_reconstruction.csv")
#target goes through same pipeline as reconstruction
target = y.data.cpu().numpy()
targetset = pandas.DataFrame(target, columns=
['OLANG','FVOC','INTON','IECHO','STEREO','UOTHR','POINT','GEST','UEYE','S
SMLE','FACEO','GZSOV','SHRNJ','RNAME','REQ','GIVE','SHOW','SIJNT','RJNT'
,'QSOV','FPLAY','IMGCR','USENS','OMAN','SELFINJ','URBEH','ACTVE','AGG','A
NXTY' ])
targetset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
1/training1/Module 1_y.csv")
#testing reconstruction
test = testset.data.cpu().numpy()
testset = pandas.DataFrame(test, columns=
['OLANG','FVOC','INTON','IECHO','STEREO','UOTHR','POINT','GEST','UEYE','S
SMLE','FACEO','GZSOV','SHRNJ','RNAME','REQ','GIVE','SHOW','SIJNT','RJNT'
```

```
,'QSOV','FPLAY','IMGCR','USENS','OMAN','SELFINJ','URBEH','ACTVE','AGG','A
NXTY' ])
testset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
1/training1/Module 1_testset.csv")
#testing target
tests = testing.data.cpu().numpy()
testingset = pandas.DataFrame(tests, columns=
['OLANG','FVOC','INTON','IECHO','STEREO','UOTHR','POINT','GEST','UEYE','S
SMLE','FACEO','GZSOV','SHRNJ','RNAME','REQ','GIVE','SHOW','SIJNT','RJNT'
,'QSOV','FPLAY','IMGCR','USENS','OMAN','SELFINJ','URBEH','ACTVE','AGG','A
NXTY' ])
testingset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
1/training1/Module 1_testeing.csv")
```

Building module 1 Autoencoder with Hidden layer of size 22

```
#import all dependencies
import pandas
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import torch
from torch.autograd import Variable
from torch import nnfrom torch.optim import lr_scheduler
from torch.nn import Parameter
import numpy as np
import math
#Create mask 29-22 matrix
mask2 = np.matrix('1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1
1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1
1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0; 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 1 1 1 1
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0; 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 1 1
1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0 0 0; 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0
```

```
1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1; 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 1')
#create transposed mask
mask1 = mask2.transpose()
torch_mask1 = torch.from_numpy(mask1)    #convert mask to torch tensor
torchmask1 = torch_mask1.type(torch.FloatTensor)    #match tensor type with
weights
torchmask1 = torchmask1.cuda()    #convert to GPU CUDA tensor
mask1 = torch.autograd.Variable(torchmask1, requires_grad=True)  #wrap in
variable to add to model
torch_mask2 = torch.from_numpy(mask2)    #convert mask to torch tensor
torchmask2 = torch_mask2.type(torch.FloatTensor)    #match tensor type with
weights
torchmask2 = torchmask2.cuda()    #convert to GPU CUDA tensor
mask2 = torch.autograd.Variable(torchmask2, requires_grad=True)    #wrap in
variable to add to model
#create special masked layer for encoder by altering feedforward layer's forward
pass
```

```
class MaskedLayer1(nn.Module):
    def __init__(self, in_features, out_features, bias=True):
        super(MaskedLayer1, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()
    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)
    def forward(self, input):
        return input.mm(self.weight*mask1)
    def __repr__(self):
        return self.__class__.__name__ + ' ('
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'
```

```
#create special masked layer for decoder by altering feedforward layer's forward
pass
class MaskedLayer2(nn.Module):
    def __init__(self, in_features, out_features, bias=True):
        super(MaskedLayer2, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()
    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)
    def forward(self, input):
        return input.mm(self.weight*mask2)
    def __repr__(self):
        return self.__class__.__name__ + ' ('
```

```
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'
dtype = torch.cuda.FloatTensor
N, D_in, H1, D_out = 10, 29, 22, 29    #batch size, dimensions of network
#randomly initialize weights with matching tensor type to mask
w1 = Variable(torch.randn(D_in, H1).type(dtype), requires_grad=True)
w2 = Variable(torch.randn(H1, D_out).type(dtype), requires_grad=True)
#import data from csv with pandas, strip values from dataframe into array
dataframe = pandas.read_csv("ADOSModule 1.csv", header=0)
dataset = dataframe.values
#model configuration, dimensions, and hyperparameters
activation = torch.nn.SELU()
model = torch.nn.Sequential(MaskedLayer1(29, 22), #incoder
                            activation,
                            MaskedLayer2(22,29))#decoder
model = model.cuda()
criterion = torch.nn.L1Loss(size_average=True)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
#create arrays to store loss measurements for training and test sets
trainingloss =[]
trainingloss = np.array(trainingloss, dtype =np.float64)
testloss = []
```

```
trainingloss = np.array(testloss, dtype =np.float64)
#10 folds for cross validation, data set is split 90/10 training test, placed into
tensor on GPU
for i in range(10):
    train, test = train_test_split(dataset, test_size=0.1)
    Module 1train = torch.from_numpy(train)
    Module 1test = torch.from_numpy(test)
    Module 1train = Module 1train.type(torch.FloatTensor)
    Module 1train = Module 1train.cuda()
    Module 1test = Module 1test.type(torch.FloatTensor)
    Module 1test = Module 1test.cuda()
    x = torch.autograd.Variable(Module 1train)     #input
    y = torch.autograd.Variable(Module 1train, requires_grad=False)     #target,
same as input
    testset = torch.autograd.Variable(Module 1test, requires_grad = False)     #test
set
    #100,000 epochs per validation fold, model minimizes difference between
target and prediction, training and test losses added to respective arrays every
epoch
        for t in range(100000):
        y_pred = model(x)
        loss = criterion(y_pred, y)
```

```
        print(i, t, loss.data[0])
        trainingloss = np.append(trainingloss, loss.data[0])
        optimizer.zero_grad()       loss.backward()
        optimizer.step()
    testing = model(testset)
    val_loss = criterion(testing, testset)
    print(val_loss.data[0])
    testloss = np.append(testloss,val_loss.data[0])
#remove encoder weights from model by iterating through model.parameters and
removing data; break loop after encoder weights removed
#CPU tensor to numpy array to pandas dataframe to comma-separated value file
for param in model.parameters():
    cpuweights = param.data.cpu()
    npweights = cpuweights.numpy()
    df = pandas.DataFrame(npweights)
    df.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module 1/training1/Module
1weights.csv")
    break
#take reconstruction (90% of data), move it to CPU tensor, then to numpy array,
then to pandas dataframe, then to csv with headers listed
reconstruction = y_pred.data.cpu()
reconstruction = y_pred.data.numpy()
```

```
reconstructionset = pandas.DataFrame(reconstruction, columns=
['OLANG','FVOC','INTON','IECHO','STEREO','UOTHR','POINT','GEST','UEYE','S
SMLE','FACEO','GZSOV','SHRNJ','RNAME','REQ','GIVE','SHOW','SIJNT','RJNT'
,'QSOV','FPLAY','IMGCR','USENS','OMAN','SELFINJ','URBEH','ACTVE','AGG','A
NXTY' ])
reconstructionset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
1/training1/Module 1_reconstruction.csv")
#target goes through same pipeline as reconstruction
target = y.data.cpu().numpy()
targetset = pandas.DataFrame(target, columns=
['OLANG','FVOC','INTON','IECHO','STEREO','UOTHR','POINT','GEST','UEYE','S
SMLE','FACEO','GZSOV','SHRNJ','RNAME','REQ','GIVE','SHOW','SIJNT','RJNT'
,'QSOV','FPLAY','IMGCR','USENS','OMAN','SELFINJ','URBEH','ACTVE','AGG','A
NXTY' ])
targetset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
1/training1/Module 1_y.csv")
#testing reconstruction
test = testset.data.cpu().numpy()
testset = pandas.DataFrame(test, columns=
['OLANG','FVOC','INTON','IECHO','STEREO','UOTHR','POINT','GEST','UEYE','S
SMLE','FACEO','GZSOV','SHRNJ','RNAME','REQ','GIVE','SHOW','SIJNT','RJNT'
,'QSOV','FPLAY','IMGCR','USENS','OMAN','SELFINJ','URBEH','ACTVE','AGG','A
```

```
NXTY' ])
testset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
1/training1/Module 1_testset.csv")
#testing target
tests = testing.data.cpu().numpy()
testingset = pandas.DataFrame(tests, columns=
['OLANG','FVOC','INTON','IECHO','STEREO','UOTHR','POINT','GEST','UEYE','S
SMLE','FACEO','GZSOV','SHRNJ','RNAME','REQ','GIVE','SHOW','SIJNT','RJNT'
,'QSOV','FPLAY','IMGCR','USENS','OMAN','SELFINJ','URBEH','ACTVE','AGG','A
NXTY' ])
testingset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
1/training1/Module 1_testeing.csv")
```

Building module 1 Autoencoder with Hidden layer of size 23

```
#import all dependencies
import pandas
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import torch
from torch.autograd import Variable
from torch import nnfrom torch.optim import lr_scheduler
from torch.nn import Parameter
import numpy as np
import math
#Create mask 29-23 matrix
mask2 = np.matrix('1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1
1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1
1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0; 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 1 1 1 1
1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0; 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 1 1
1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0 0; 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0
```

```
1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1; 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1;0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1')
#create transposed mask
mask1 = mask2.transpose()
torch_mask1 = torch.from_numpy(mask1)    #convert mask to torch tensor
torchmask1 = torch_mask1.type(torch.FloatTensor)    #match tensor type with
weights
torchmask1 = torchmask1.cuda()    #convert to GPU CUDA tensor
mask1 = torch.autograd.Variable(torchmask1, requires_grad=True)  #wrap in
variable to add to model
torch_mask2 = torch.from_numpy(mask2)    #convert mask to torch tensor
torchmask2 = torch_mask2.type(torch.FloatTensor)    #match tensor type with
weights
torchmask2 = torchmask2.cuda()    #convert to GPU CUDA tensor
mask2 = torch.autograd.Variable(torchmask2, requires_grad=True)    #wrap in
variable to add to model
#create special masked layer for encoder by altering feedforward layer's forward
pass
```

```
class MaskedLayer1(nn.Module):
    def __init__(self, in_features, out_features, bias=True):
        super(MaskedLayer1, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()
    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)
    def forward(self, input):
        return input.mm(self.weight*mask1)
    def __repr__(self):
        return self.__class__.__name__ + ' ('
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'
```

```
#create special masked layer for decoder by altering feedforward layer's forward
pass
class MaskedLayer2(nn.Module):
    def __init__(self, in_features, out_features, bias=True):
        super(MaskedLayer2, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()
    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)
    def forward(self, input):
        return input.mm(self.weight*mask2)
    def __repr__(self):
        return self.__class__.__name__ + ' ('
```

```
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'
dtype = torch.cuda.FloatTensor
N, D_in, H1, D_out = 10, 29, 23, 29    #batch size, dimensions of network
#randomly initialize weights with matching tensor type to mask
w1 = Variable(torch.randn(D_in, H1).type(dtype), requires_grad=True)
w2 = Variable(torch.randn(H1, D_out).type(dtype), requires_grad=True)
#import data from csv with pandas, strip values from dataframe into array
dataframe = pandas.read_csv("ADOSModule 1.csv", header=0)
dataset = dataframe.values
#model configuration, dimensions, and hyperparameters
activation = torch.nn.SELU()
model = torch.nn.Sequential(MaskedLayer1(29, 23), #incoder
                            activation,
                            MaskedLayer2(23,29))#decoder
model = model.cuda()
criterion = torch.nn.L1Loss(size_average=True)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
#create arrays to store loss measurements for training and test sets
trainingloss =[]
trainingloss = np.array(trainingloss, dtype =np.float64)
testloss = []
```

```
trainingloss = np.array(testloss, dtype =np.float64)
#10 folds for cross validation, data set is split 90/10 training test, placed into
tensor on GPU
for i in range(10):
    train, test = train_test_split(dataset, test_size=0.1)
    Module 1train = torch.from_numpy(train)
    Module 1test = torch.from_numpy(test)
    Module 1train = Module 1train.type(torch.FloatTensor)
    Module 1train = Module 1train.cuda()
    Module 1test = Module 1test.type(torch.FloatTensor)
    Module 1test = Module 1test.cuda()
    x = torch.autograd.Variable(Module 1train)     #input
    y = torch.autograd.Variable(Module 1train, requires_grad=False)     #target,
same as input
    testset = torch.autograd.Variable(Module 1test, requires_grad = False)     #test
set
    #100,000 epochs per validation fold, model minimizes difference between
target and prediction, training and test losses added to respective arrays every
epoch
        for t in range(100000):
        y_pred = model(x)
        loss = criterion(y_pred, y)
```

```
        print(i, t, loss.data[0])
        trainingloss = np.append(trainingloss, loss.data[0])
        optimizer.zero_grad()       loss.backward()
        optimizer.step()
    testing = model(testset)
    val_loss = criterion(testing, testset)
    print(val_loss.data[0])
    testloss = np.append(testloss,val_loss.data[0])
#remove encoder weights from model by iterating through model.parameters and
removing data; break loop after encoder weights removed
#CPU tensor to numpy array to pandas dataframe to comma-separated value file
for param in model.parameters():
    cpuweights = param.data.cpu()
    npweights = cpuweights.numpy()
    df = pandas.DataFrame(npweights)
    df.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module 1/training1/Module
1weights.csv")
    break
#take reconstruction (90% of data), move it to CPU tensor, then to numpy array,
then to pandas dataframe, then to csv with headers listed
reconstruction = y_pred.data.cpu()
reconstruction = y_pred.data.numpy()
```

```
reconstructionset = pandas.DataFrame(reconstruction, columns=
['OLANG','FVOC','INTON','IECHO','STEREO','UOTHR','POINT','GEST','UEYE','S
SMLE','FACEO','GZSOV','SHRNJ','RNAME','REQ','GIVE','SHOW','SIJNT','RJNT'
,'QSOV','FPLAY','IMGCR','USENS','OMAN','SELFINJ','URBEH','ACTVE','AGG','A
NXTY' ])
reconstructionset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
1/training1/Module 1_reconstruction.csv")
#target goes through same pipeline as reconstruction
target = y.data.cpu().numpy()
targetset = pandas.DataFrame(target, columns=
['OLANG','FVOC','INTON','IECHO','STEREO','UOTHR','POINT','GEST','UEYE','S
SMLE','FACEO','GZSOV','SHRNJ','RNAME','REQ','GIVE','SHOW','SIJNT','RJNT'
,'QSOV','FPLAY','IMGCR','USENS','OMAN','SELFINJ','URBEH','ACTVE','AGG','A
NXTY' ])
targetset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
1/training1/Module 1_y.csv")
#testing reconstruction
test = testset.data.cpu().numpy()
testset = pandas.DataFrame(test, columns=
['OLANG','FVOC','INTON','IECHO','STEREO','UOTHR','POINT','GEST','UEYE','S
SMLE','FACEO','GZSOV','SHRNJ','RNAME','REQ','GIVE','SHOW','SIJNT','RJNT'
,'QSOV','FPLAY','IMGCR','USENS','OMAN','SELFINJ','URBEH','ACTVE','AGG','A
```

```
NXTY' ])
testset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
1/training1/Module 1_testset.csv")
#testing target
tests = testing.data.cpu().numpy()
testingset = pandas.DataFrame(tests, columns=
['OLANG','FVOC','INTON','IECHO','STEREO','UOTHR','POINT','GEST','UEYE','S
SMLE','FACEO','GZSOV','SHRNJ','RNAME','REQ','GIVE','SHOW','SIJNT','RJNT'
,'QSOV','FPLAY','IMGCR','USENS','OMAN','SELFINJ','URBEH','ACTVE','AGG','A
NXTY' ])
testingset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
1/training1/Module 1_testeing.csv")
```

Building module 2 Autoencoder with Hidden layer of size 19

```
#import all dependencies

import pandas

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

import torch

from torch.autograd import Variable

from torch import nnfrom torch.optim import lr_scheduler

from torch.nn import Parameter

import numpy as np

import math

#Create mask 28-19 matrix

mask2 = np.matrix('1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;  1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0;  1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;  1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0;  1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;  1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;  1
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;  0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0; 0 0 0 0 1
1 1 1 1 1 1 1 1 0 0 0 0 0 0; 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0; 0 0 0 0 1 1 1 1 1
1 1 1 1 0 0 0 0 0 0; 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0; 0 0 0 0 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0; 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0; 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0
0 0; 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0; 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0; 0 0
0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0; 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0;0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0
```

```
0 0 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1;
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1')
#create transposed mask
mask1 = mask2.transpose()
torch_mask1 = torch.from_numpy(mask1)    #convert mask to torch tensor
torchmask1 = torch_mask1.type(torch.FloatTensor)    #match tensor type with
weights
torchmask1 = torchmask1.cuda()    #convert to GPU CUDA tensor
mask1 = torch.autograd.Variable(torchmask1, requires_grad=True)  #wrap in
variable to add to model
torch_mask2 = torch.from_numpy(mask2)    #convert mask to torch tensor
torchmask2 = torch_mask2.type(torch.FloatTensor)    #match tensor type with
weights
torchmask2 = torchmask2.cuda()    #convert to GPU CUDA tensor
mask2 = torch.autograd.Variable(torchmask2, requires_grad=True)    #wrap in
variable to add to model
#create special masked layer for encoder by altering feedforward layer's forward
pass
class MaskedLayer1(nn.Module):
   def __init__(self, in_features, out_features, bias=True):
      super(MaskedLayer1, self).__init__()
```

```
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()
    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)
    def forward(self, input):
        return input.mm(self.weight*mask1)
    def __repr__(self):
        return self.__class__.__name__ + ' ('
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'
#create special masked layer for decoder by altering feedforward layer's forward
pass
class MaskedLayer2(nn.Module):
```

```
def __init__(self, in_features, out_features, bias=True):
    super(MaskedLayer2, self).__init__()
    self.in_features = in_features
    self.out_features = out_features
    self.weight = Parameter(torch.Tensor(in_features, out_features))
    if bias:
        self.bias = Parameter(torch.Tensor(out_features))
    else:
        self.register_parameter('bias', None)
    self.reset_parameters()
def reset_parameters(self):
    stdv = 1. / math.sqrt(self.weight.size(1))
    self.weight.data.uniform_(-stdv, stdv)
    if self.bias is not None:
        self.bias.data.uniform_(-stdv, stdv)
def forward(self, input):
    return input.mm(self.weight*mask2)
def __repr__(self):
    return self.__class__.__name__ + ' ('
        + str(self.in_features) + ' -> '
        + str(self.out_features) + ')'
```

```
dtype = torch.cuda.FloatTensor
N, D_in, H1, D_out = 10, 28, 19, 28    #batch size, dimensions of network
#randomly initialize weights with matching tensor type to mask
w1 = Variable(torch.randn(D_in, H1).type(dtype), requires_grad=True)
w2 = Variable(torch.randn(H1, D_out).type(dtype), requires_grad=True)
#import data from csv with pandas, strip values from dataframe into array
dataframe = pandas.read_csv("ADOSModule 2.csv", header=0)
dataset = dataframe.values
#model configuration, dimensions, and hyperparameters
activation = torch.nn.SELU()
model = torch.nn.Sequential(MaskedLayer1(28, 19), #incoder
                            activation,
                            MaskedLayer2(19,28))#decoder
model = model.cuda()
criterion = torch.nn.L1Loss(size_average=True)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
#create arrays to store loss measurements for training and test sets
trainingloss =[]
trainingloss = np.array(trainingloss, dtype =np.float64)
testloss = []
trainingloss = np.array(testloss, dtype =np.float64)
```

```
#10 folds for cross validation, data set is split 90/10 training test, placed into
tensor on GPU
for i in range(10):
    train, test = train_test_split(dataset, test_size=0.1)
    Module 2train = torch.from_numpy(train)
    Module 2test = torch.from_numpy(test)
    Module 2train = Module 2train.type(torch.FloatTensor)
    Module 2train = Module 2train.cuda()
    Module 2test = Module 2test.type(torch.FloatTensor)
    Module 2test = Module 2test.cuda()
    x = torch.autograd.Variable(Module 2train)     #input
    y = torch.autograd.Variable(Module 2train, requires_grad=False)     #target,
same as input
    testset = torch.autograd.Variable(Module 2test, requires_grad = False)     #test
set
    #100,000 epochs per validation fold, model minimizes difference between
target and prediction, training and test losses added to respective arrays every
epoch
        for t in range(100000):
        y_pred = model(x)
        loss = criterion(y_pred, y)
        print(i, t, loss.data[0])
```

```
        trainingloss = np.append(trainingloss, loss.data[0])
        optimizer.zero_grad()       loss.backward()
        optimizer.step()
    testing = model(testset)
    val_loss = criterion(testing, testset)
    print(val_loss.data[0])
    testloss = np.append(testloss,val_loss.data[0])
#remove encoder weights from model by iterating through model.parameters and removing data; break loop after encoder weights removed
#CPU tensor to numpy array to pandas dataframe to comma-separated value file
for param in model.parameters():
    cpuweights = param.data.cpu()
    npweights = cpuweights.numpy()
    df = pandas.DataFrame(npweights)
    df.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module 2/training1/Module 2weights.csv")
    break
#take reconstruction (90% of data), move it to CPU tensor, then to numpy array, then to pandas dataframe, then to csv with headers listed
reconstruction = y_pred.data.cpu()
reconstruction = y_pred.data.numpy()
reconstructionset = pandas.DataFrame(reconstruction, columns=
```

```
['OLANG','SPABN','IECHO','STEREO','CONVS','POINT','DGEST','UEYE','FACE
O', 'SHRNJ', 'RNAME', 'SHOW', 'SIJNT', 'RJNT', 'QSOV', 'ASOV', 'QSRES',
'ARSOC', 'OQRAP', 'FPLAY', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'URBEH',
'ACTVE', 'AGG', 'ANXTY' ])
reconstructionset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
2/training1/Module 2_reconstruction.csv")
#target goes through same pipeline as reconstruction
target = y.data.cpu().numpy()
targetset = pandas.DataFrame(target, columns=
['OLANG','SPABN','IECHO','STEREO','CONVS','POINT','DGEST','UEYE','FACE
O', 'SHRNJ', 'RNAME', 'SHOW', 'SIJNT', 'RJNT', 'QSOV', 'ASOV', 'QSRES',
'ARSOC', 'OQRAP', 'FPLAY', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'URBEH',
'ACTVE', 'AGG', 'ANXTY'])
targetset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
2/training1/Module 2_y.csv")
#testing reconstruction
test = testset.data.cpu().numpy()
testset = pandas.DataFrame(test, columns=
['OLANG','SPABN','IECHO','STEREO','CONVS','POINT','DGEST','UEYE','FACE
O', 'SHRNJ', 'RNAME', 'SHOW', 'SIJNT', 'RJNT', 'QSOV', 'ASOV', 'QSRES',
'ARSOC', 'OQRAP', 'FPLAY', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'URBEH',
'ACTVE', 'AGG', 'ANXTY'])
```

```
testset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
2/training1/Module 2_testset.csv")
#testing target
tests = testing.data.cpu().numpy()
testingset = pandas.DataFrame(tests, columns=
['OLANG','SPABN','IECHO','STEREO','CONVS','POINT','DGEST','UEYE','FACE
O', 'SHRNJ', 'RNAME', 'SHOW', 'SIJNT', 'RJNT', 'QSOV', 'ASOV', 'QSRES',
'ARSOC', 'OQRAP', 'FPLAY', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'URBEH',
'ACTVE', 'AGG', 'ANXTY'])
testingset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
2/training1/Module 2_testeing.csv")
```

Building module 2 Autoencoder with Hidden layer of size 20

```
#import all dependencies
import pandas
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import torch
from torch.autograd import Variable
from torch import nnfrom torch.optim import lr_scheduler
from torch.nn import Parameter
import numpy as np
import math
#Create mask 28-20 matrix
mask2 = np.matrix('1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;  1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0;1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;  1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 ;  1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ;  1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 ;  1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ;0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0; 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0; 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0
0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0 0 0
0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0 0 0 0 1
1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0 0 0 0 1 1 1
1 1 1 1 1 1 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0
```

```
0 0 0 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 1
1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1')
#create transposed mask
mask1 = mask2.transpose()
torch_mask1 = torch.from_numpy(mask1)    #convert mask to torch tensor
torchmask1 = torch_mask1.type(torch.FloatTensor)    #match tensor type with
weights
torchmask1 = torchmask1.cuda()    #convert to GPU CUDA tensor
mask1 = torch.autograd.Variable(torchmask1, requires_grad=True)  #wrap in
variable to add to model
torch_mask2 = torch.from_numpy(mask2)    #convert mask to torch tensor
torchmask2 = torch_mask2.type(torch.FloatTensor)    #match tensor type with
weights
torchmask2 = torchmask2.cuda()    #convert to GPU CUDA tensor
mask2 = torch.autograd.Variable(torchmask2, requires_grad=True)    #wrap in
variable to add to model
#create special masked layer for encoder by altering feedforward layer's forward
pass
class MaskedLayer1(nn.Module):
   def __init__(self, in_features, out_features, bias=True):
```

```
        super(MaskedLayer1, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()
    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)
    def forward(self, input):
        return input.mm(self.weight*mask1)
    def __repr__(self):
        return self.__class__.__name__ + ' ('
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'
#create special masked layer for decoder by altering feedforward layer's forward
pass
```

```
class MaskedLayer2(nn.Module):
    def __init__(self, in_features, out_features, bias=True):
        super(MaskedLayer2, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()
    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)
    def forward(self, input):
        return input.mm(self.weight*mask2)
    def __repr__(self):
        return self.__class__.__name__ + ' ('
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'
```

```
dtype = torch.cuda.FloatTensor
N, D_in, H1, D_out = 10, 28, 20, 28    #batch size, dimensions of network
#randomly initialize weights with matching tensor type to mask
w1 = Variable(torch.randn(D_in, H1).type(dtype), requires_grad=True)
w2 = Variable(torch.randn(H1, D_out).type(dtype), requires_grad=True)
#import data from csv with pandas, strip values from dataframe into array
dataframe = pandas.read_csv("ADOSModule 2.csv", header=0)
dataset = dataframe.values
#model configuration, dimensions, and hyperparameters
activation = torch.nn.SELU()
model = torch.nn.Sequential(MaskedLayer1(28, 20), #incoder
                            activation,
                            MaskedLayer2(19,20))#decoder
model = model.cuda()
criterion = torch.nn.L1Loss(size_average=True)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
#create arrays to store loss measurements for training and test sets
trainingloss =[]
trainingloss = np.array(trainingloss, dtype =np.float64)
testloss = []
trainingloss = np.array(testloss, dtype =np.float64)
```

```
#10 folds for cross validation, data set is split 90/10 training test, placed into
tensor on GPU
for i in range(10):
    train, test = train_test_split(dataset, test_size=0.1)
    Module 2train = torch.from_numpy(train)
    Module 2test = torch.from_numpy(test)
    Module 2train = Module 2train.type(torch.FloatTensor)
    Module 2train = Module 2train.cuda()
    Module 2test = Module 2test.type(torch.FloatTensor)
    Module 2test = Module 2test.cuda()
    x = torch.autograd.Variable(Module 2train)     #input
    y = torch.autograd.Variable(Module 2train, requires_grad=False)     #target,
same as input
    testset = torch.autograd.Variable(Module 2test, requires_grad = False)     #test
set
    #100,000 epochs per validation fold, model minimizes difference between
target and prediction, training and test losses added to respective arrays every
epoch
        for t in range(100000):
        y_pred = model(x)
        loss = criterion(y_pred, y)
```

```
        print(i, t, loss.data[0])
        trainingloss = np.append(trainingloss, loss.data[0])
        optimizer.zero_grad()       loss.backward()
        optimizer.step()
    testing = model(testset)
    val_loss = criterion(testing, testset)
    print(val_loss.data[0])
    testloss = np.append(testloss,val_loss.data[0])
#remove encoder weights from model by iterating through model.parameters and
removing data; break loop after encoder weights removed
#CPU tensor to numpy array to pandas dataframe to comma-separated value file
for param in model.parameters():
    cpuweights = param.data.cpu()
    npweights = cpuweights.numpy()
    df = pandas.DataFrame(npweights)
    df.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module 2/training1/Module
2weights.csv")
    break
#take reconstruction (90% of data), move it to CPU tensor, then to numpy array,
then to pandas dataframe, then to csv with headers listed
reconstruction = y_pred.data.cpu()
reconstruction = y_pred.data.numpy()
```

```
reconstructionset = pandas.DataFrame(reconstruction, columns=
['OLANG','SPABN','IECHO','STEREO','CONVS','POINT','DGEST','UEYE','FACE
O', 'SHRNJ', 'RNAME', 'SHOW', 'SIJNT', 'RJNT', 'QSOV', 'ASOV', 'QSRES',
'ARSOC', 'OQRAP', 'FPLAY', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'URBEH',
'ACTVE', 'AGG', 'ANXTY' ])
reconstructionset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
2/training1/Module 2_reconstruction.csv")
#target goes through same pipeline as reconstruction
target = y.data.cpu().numpy()
targetset = pandas.DataFrame(target, columns=
['OLANG','SPABN','IECHO','STEREO','CONVS','POINT','DGEST','UEYE','FACE
O', 'SHRNJ', 'RNAME', 'SHOW', 'SIJNT', 'RJNT', 'QSOV', 'ASOV', 'QSRES',
'ARSOC', 'OQRAP', 'FPLAY', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'URBEH',
'ACTVE', 'AGG', 'ANXTY'])
targetset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
2/training1/Module 2_y.csv")
#testing reconstruction
test = testset.data.cpu().numpy()
testset = pandas.DataFrame(test, columns=
['OLANG','SPABN','IECHO','STEREO','CONVS','POINT','DGEST','UEYE','FACE
O', 'SHRNJ', 'RNAME', 'SHOW', 'SIJNT', 'RJNT', 'QSOV', 'ASOV', 'QSRES',
'ARSOC', 'OQRAP', 'FPLAY', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'URBEH',
```

```
'ACTVE', 'AGG', 'ANXTY'])
testset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
2/training1/Module 2_testset.csv")
#testing target
tests = testing.data.cpu().numpy()
testingset = pandas.DataFrame(tests, columns=
['OLANG','SPABN','IECHO','STEREO','CONVS','POINT','DGEST','UEYE','FACE
O', 'SHRNJ', 'RNAME', 'SHOW', 'SIJNT', 'RJNT', 'QSOV', 'ASOV', 'QSRES',
'ARSOC', 'OQRAP', 'FPLAY', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'URBEH',
'ACTVE', 'AGG', 'ANXTY'])
testingset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
2/training1/Module 2_testeing.csv")
```

Building module 2 Autoencoder with Hidden layer of size 21

```
#import all dependencies

import pandas

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

import torch

from torch.autograd import Variable

from torch import nnfrom torch.optim import lr_scheduler

from torch.nn import Parameter

import numpy as np

import math


#Create mask 28-21 matrix

mask2 = np.matrix('1 1 1 1 1  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ;  1 1 1 1 1 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 ; 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;  1 1 1 1 1 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0;  1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;  1 1 1 1 1 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0;  1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 1 1 1 1 1

1 1 1 1 1 0 0 0 0 0 0; 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0; 0 0 0 0 0 1 1 1 1 1

1 1 1 1 1 0 0 0 0 0 0; 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0; 0 0 0 0 0 1 1 1 1 1

1 1 1 1 1 0 0 0 0 0 0; 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0; 0 0 0 0 0 1 1 1 1 1

1 1 1 1 1 0 0 0 0 0 0; 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0; 0 0 0 0 0 1 1 1 1 1
```

```
1 1 1 1 1 0 0 0 0 0 0; 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0; 0 0 0 0 0 1 1 1 1 1
1 1 1 1 1 0 0 0 0 0 0; 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1')

#create transposed mask
mask1 = mask2.transpose()

torch_mask1 = torch.from_numpy(mask1)     #convert mask to torch tensor
torchmask1 = torch_mask1.type(torch.FloatTensor)     #match tensor type with
weights
torchmask1 = torchmask1.cuda()     #convert to GPU CUDA tensor
mask1 = torch.autograd.Variable(torchmask1, requires_grad=True)  #wrap in
variable to add to model

torch_mask2 = torch.from_numpy(mask2)     #convert mask to torch tensor
torchmask2 = torch_mask2.type(torch.FloatTensor)     #match tensor type with
weights
torchmask2 = torchmask2.cuda()     #convert to GPU CUDA tensor
```

```
mask2 = torch.autograd.Variable(torchmask2, requires_grad=True)     #wrap in
variable to add to model

#create special masked layer for encoder by altering feedforward layer's forward
pass
class MaskedLayer1(nn.Module):
    def __init__(self, in_features, out_features, bias=True):
        super(MaskedLayer1, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()

    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)
```

```
    def forward(self, input):
        return input.mm(self.weight*mask1)

    def __repr__(self):
        return self.__class__.__name__ + ' ('
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'


#create special masked layer for decoder by altering feedforward layer's forward
pass
class MaskedLayer2(nn.Module):
    def __init__(self, in_features, out_features, bias=True):
        super(MaskedLayer2, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
            self.register_parameter('bias', None)
```

```
        self.reset_parameters()


    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)


    def forward(self, input):
        return input.mm(self.weight*mask2)
    def __repr__(self):
        return self.__class__.__name__ + ' ('
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'


dtype = torch.cuda.FloatTensor
N, D_in, H1, D_out = 10, 28, 21, 28    #batch size, dimensions of network


#randomly initialize weights with matching tensor type to mask
w1 = Variable(torch.randn(D_in, H1).type(dtype), requires_grad=True)
w2 = Variable(torch.randn(H1, D_out).type(dtype), requires_grad=True)
```

```
#import data from csv with pandas, strip values from dataframe into array
dataframe = pandas.read_csv("ADOSModule 2.csv", header=0)
dataset = dataframe.values

#model configuration, dimensions, and hyperparameters
activation = torch.nn.SELU()
model = torch.nn.Sequential(MaskedLayer1(28, 21), #incoder
                            activation,
                            MaskedLayer2(21,28))#decoder
model = model.cuda()
criterion = torch.nn.L1Loss(size_average=True)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

#create arrays to store loss measurements for training and test sets
trainingloss =[]
trainingloss = np.array(trainingloss, dtype =np.float64)
testloss = []
trainingloss = np.array(testloss, dtype =np.float64)

#10 folds for cross validation, data set is split 90/10 training test, placed into
tensor on GPU
for i in range(10):
```

```
    train, test = train_test_split(dataset, test_size=0.1)
    Module 2train = torch.from_numpy(train)
    Module 2test = torch.from_numpy(test)
    Module 2train = Module 2train.type(torch.FloatTensor)
    Module 2train = Module 2train.cuda()
    Module 2test = Module 2test.type(torch.FloatTensor)
    Module 2test = Module 2test.cuda()
    x = torch.autograd.Variable(Module 2train)     #input
    y = torch.autograd.Variable(Module 2train, requires_grad=False)     #target,
same as input
    testset = torch.autograd.Variable(Module 2test, requires_grad = False)     #test
set
    #100,000 epochs per validation fold, model minimizes difference between
target and prediction, training and test losses added to respective arrays every
epoch
        for t in range(100000):
        y_pred = model(x)
        loss = criterion(y_pred, y)
        print(i, t, loss.data[0])
        trainingloss = np.append(trainingloss, loss.data[0])
        optimizer.zero_grad()        loss.backward()
        optimizer.step()
```

```
    testing = model(testset)
    val_loss = criterion(testing, testset)
    print(val_loss.data[0])
    testloss = np.append(testloss,val_loss.data[0])

#remove encoder weights from model by iterating through model.parameters and
removing data; break loop after encoder weights removed
#CPU tensor to numpy array to pandas dataframe to comma-separated value file
for param in model.parameters():
    cpuweights = param.data.cpu()
    npweights = cpuweights.numpy()
    df = pandas.DataFrame(npweights)
    df.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module 2/training1/Module
2weights.csv")
    break

#take reconstruction (90% of data), move it to CPU tensor, then to numpy array,
then to pandas dataframe, then to csv with headers listed
reconstruction = y_pred.data.cpu()
reconstruction = y_pred.data.numpy()
reconstructionset = pandas.DataFrame(reconstruction, columns=
['OLANG','SPABN','IECHO','STEREO','CONVS','POINT','DGEST','UEYE','FACE
```

```
O', 'SHRNJ', 'RNAME', 'SHOW', 'SIJNT', 'RJNT', 'QSOV', 'ASOV', 'QSRES',
'ARSOC', 'OQRAP', 'FPLAY', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'URBEH',
'ACTVE', 'AGG', 'ANXTY' ])

reconstructionset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
2/training1/Module 2_reconstruction.csv")

#target goes through same pipeline as reconstruction
target = y.data.cpu().numpy()
targetset = pandas.DataFrame(target, columns=
['OLANG','SPABN','IECHO','STEREO','CONVS','POINT','DGEST','UEYE','FACE
O', 'SHRNJ', 'RNAME', 'SHOW', 'SIJNT', 'RJNT', 'QSOV', 'ASOV', 'QSRES',
'ARSOC', 'OQRAP', 'FPLAY', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'URBEH',
'ACTVE', 'AGG', 'ANXTY'])
targetset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
2/training1/Module 2_y.csv")

#testing reconstruction
test = testset.data.cpu().numpy()
testset = pandas.DataFrame(test, columns=
['OLANG','SPABN','IECHO','STEREO','CONVS','POINT','DGEST','UEYE','FACE
O', 'SHRNJ', 'RNAME', 'SHOW', 'SIJNT', 'RJNT', 'QSOV', 'ASOV', 'QSRES',
```

```
'ARSOC', 'OQRAP', 'FPLAY', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'URBEH',
'ACTVE', 'AGG', 'ANXTY'])
testset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
2/training1/Module 2_testset.csv")

#testing target
tests = testing.data.cpu().numpy()
testingset = pandas.DataFrame(tests, columns=
['OLANG','SPABN','IECHO','STEREO','CONVS','POINT','DGEST','UEYE','FACE
O', 'SHRNJ', 'RNAME', 'SHOW', 'SIJNT', 'RJNT', 'QSOV', 'ASOV', 'QSRES',
'ARSOC', 'OQRAP', 'FPLAY', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'URBEH',
'ACTVE', 'AGG', 'ANXTY'])
testingset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
2/training1/Module 2_testeing.csv")
```

Building module 2 Autoencoder with Hidden layer of size 22

```
#import all dependencies
import pandas
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import torch
from torch.autograd import Variable
from torch import nnfrom torch.optim import lr_scheduler
from torch.nn import Parameter
import numpy as np
import math


#Create mask 28-22 matrix
mask2 = np.matrix('1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;  1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0;1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;  1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 ;  1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ;  1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ;  1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ;0 0 0
0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0
0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0; 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0; 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0
```

0 0 0 0 0; 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0; 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0 0 0 0 0 1 1 1 1 1 1 1 1
1 1 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1; 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1')

#create transposed mask

mask1 = mask2.transpose()

torch_mask1 = torch.from_numpy(mask1) #convert mask to torch tensor

torchmask1 = torch_mask1.type(torch.FloatTensor) #match tensor type with weights

torchmask1 = torchmask1.cuda() #convert to GPU CUDA tensor

mask1 = torch.autograd.Variable(torchmask1, requires_grad=True) #wrap in variable to add to model

torch_mask2 = torch.from_numpy(mask2) #convert mask to torch tensor

torchmask2 = torch_mask2.type(torch.FloatTensor) #match tensor type with weights

```
torchmask2 = torchmask2.cuda()    #convert to GPU CUDA tensor
mask2 = torch.autograd.Variable(torchmask2, requires_grad=True)    #wrap in
variable to add to model

#create special masked layer for encoder by altering feedforward layer's forward
pass
class MaskedLayer1(nn.Module):
    def __init__(self, in_features, out_features, bias=True):
        super(MaskedLayer1, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()

    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
```

```
            self.bias.data.uniform_(-stdv, stdv)


    def forward(self, input):
        return input.mm(self.weight*mask1)


    def __repr__(self):
        return self.__class__.__name__ + ' ('
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'



#create special masked layer for decoder by altering feedforward layer's forward
pass
class MaskedLayer2(nn.Module):
    def __init__(self, in_features, out_features, bias=True):
        super(MaskedLayer2, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
```

```
            self.register_parameter('bias', None)
        self.reset_parameters()

    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)

    def forward(self, input):
        return input.mm(self.weight*mask2)
    def __repr__(self):
        return self.__class__.__name__ + ' ('
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'

dtype = torch.cuda.FloatTensor
N, D_in, H1, D_out = 10, 28, 22, 28    #batch size, dimensions of network

#randomly initialize weights with matching tensor type to mask
w1 = Variable(torch.randn(D_in, H1).type(dtype), requires_grad=True)
w2 = Variable(torch.randn(H1, D_out).type(dtype), requires_grad=True)
```

```
#import data from csv with pandas, strip values from dataframe into array
dataframe = pandas.read_csv("ADOSModule 2.csv", header=0)
dataset = dataframe.values

#model configuration, dimensions, and hyperparameters
activation = torch.nn.SELU()
model = torch.nn.Sequential(MaskedLayer1(28, 22), #incoder
                            activation,
                            MaskedLayer2(22,28))#decoder
model = model.cuda()
criterion = torch.nn.L1Loss(size_average=True)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

#create arrays to store loss measurements for training and test sets
trainingloss =[]
trainingloss = np.array(trainingloss, dtype =np.float64)
testloss = []
trainingloss = np.array(testloss, dtype =np.float64)

#10 folds for cross validation, data set is split 90/10 training test, placed into
tensor on GPU
```

```
for i in range(10):
    train, test = train_test_split(dataset, test_size=0.1)
    Module 2train = torch.from_numpy(train)
    Module 2test = torch.from_numpy(test)
    Module 2train = Module 2train.type(torch.FloatTensor)
    Module 2train = Module 2train.cuda()
    Module 2test = Module 2test.type(torch.FloatTensor)
    Module 2test = Module 2test.cuda()
    x = torch.autograd.Variable(Module 2train)     #input
    y = torch.autograd.Variable(Module 2train, requires_grad=False)     #target,
same as input
    testset = torch.autograd.Variable(Module 2test, requires_grad = False)     #test
set
    #100,000 epochs per validation fold, model minimizes difference between
target and prediction, training and test losses added to respective arrays every
epoch
        for t in range(100000):
        y_pred = model(x)
        loss = criterion(y_pred, y)
        print(i, t, loss.data[0])
        trainingloss = np.append(trainingloss, loss.data[0])
        optimizer.zero_grad()        loss.backward()
```

```
        optimizer.step()
    testing = model(testset)
    val_loss = criterion(testing, testset)
    print(val_loss.data[0])
    testloss = np.append(testloss,val_loss.data[0])

#remove encoder weights from model by iterating through model.parameters and
removing data; break loop after encoder weights removed
#CPU tensor to numpy array to pandas dataframe to comma-separated value file
for param in model.parameters():
    cpuweights = param.data.cpu()
    npweights = cpuweights.numpy()
    df = pandas.DataFrame(npweights)
    df.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module 2/training1/Module
2weights.csv")
    break

#take reconstruction (90% of data), move it to CPU tensor, then to numpy array,
then to pandas dataframe, then to csv with headers listed
reconstruction = y_pred.data.cpu()
reconstruction = y_pred.data.numpy()
reconstructionset = pandas.DataFrame(reconstruction, columns=
```

```
['OLANG','SPABN','IECHO','STEREO','CONVS','POINT','DGEST','UEYE','FACE
O', 'SHRNJ', 'RNAME', 'SHOW', 'SIJNT', 'RJNT', 'QSOV', 'ASOV', 'QSRES',
'ARSOC', 'OQRAP', 'FPLAY', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'URBEH',
'ACTVE', 'AGG', 'ANXTY' ])

reconstructionset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
2/training1/Module 2_reconstruction.csv")

#target goes through same pipeline as reconstruction
target = y.data.cpu().numpy()
targetset = pandas.DataFrame(target, columns=
['OLANG','SPABN','IECHO','STEREO','CONVS','POINT','DGEST','UEYE','FACE
O', 'SHRNJ', 'RNAME', 'SHOW', 'SIJNT', 'RJNT', 'QSOV', 'ASOV', 'QSRES',
'ARSOC', 'OQRAP', 'FPLAY', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'URBEH',
'ACTVE', 'AGG', 'ANXTY'])
targetset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
2/training1/Module 2_y.csv")

#testing reconstruction
test = testset.data.cpu().numpy()
testset = pandas.DataFrame(test, columns=
['OLANG','SPABN','IECHO','STEREO','CONVS','POINT','DGEST','UEYE','FACE
```

```
O', 'SHRNJ', 'RNAME', 'SHOW', 'SIJNT', 'RJNT', 'QSOV', 'ASOV', 'QSRES',
'ARSOC', 'OQRAP', 'FPLAY', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'URBEH',
'ACTVE', 'AGG', 'ANXTY'])
testset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
2/training1/Module 2_testset.csv")

#testing target
tests = testing.data.cpu().numpy()
testingset = pandas.DataFrame(tests, columns=
['OLANG','SPABN','IECHO','STEREO','CONVS','POINT','DGEST','UEYE','FACE
O', 'SHRNJ', 'RNAME', 'SHOW', 'SIJNT', 'RJNT', 'QSOV', 'ASOV', 'QSRES',
'ARSOC', 'OQRAP', 'FPLAY', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'URBEH',
'ACTVE', 'AGG', 'ANXTY'])
testingset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
2/training1/Module 2_testeing.csv")
```

Building module 3 Autoencoder with Hidden layer of size 19

```
#import all dependencies
import pandas
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import torch
from torch.autograd import Variable
from torch import nnfrom torch.optim import lr_scheduler
from torch.nn import Parameter
import numpy as np
import math


#Create mask 28-19 matrix
mask2 = np.matrix('1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;  1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0;  1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0;  1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0;  0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0; 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 ; 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0; 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0; 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0; 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0; 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0; 0 0 0
```

```
0 0 0 1 1 1 1 1 1 0 0 0 0 0 0; 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0; 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1')

#create transposed mask
mask1 = mask2.transpose()

torch_mask1 = torch.from_numpy(mask1)     #convert mask to torch tensor
torchmask1 = torch_mask1.type(torch.FloatTensor)     #match tensor type with
weights
torchmask1 = torchmask1.cuda()     #convert to GPU CUDA tensor
mask1 = torch.autograd.Variable(torchmask1, requires_grad=True)  #wrap in
variable to add to model

torch_mask2 = torch.from_numpy(mask2)     #convert mask to torch tensor
torchmask2 = torch_mask2.type(torch.FloatTensor)     #match tensor type with
weights
torchmask2 = torchmask2.cuda()     #convert to GPU CUDA tensor
mask2 = torch.autograd.Variable(torchmask2, requires_grad=True)     #wrap in
variable to add to model
```

```
#create special masked layer for encoder by altering feedforward layer's forward
pass
class MaskedLayer1(nn.Module):
    def __init__(self, in_features, out_features, bias=True):
        super(MaskedLayer1, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()

    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)

    def forward(self, input):
```

```
        return input.mm(self.weight*mask1)


    def __repr__(self):
        return self.__class__.__name__ + ' ('
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'



#create special masked layer for decoder by altering feedforward layer's forward
pass
class MaskedLayer2(nn.Module):
    def __init__(self, in_features, out_features, bias=True):
        super(MaskedLayer2, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()
```

```
    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)

    def forward(self, input):
        return input.mm(self.weight*mask2)
    def __repr__(self):
        return self.__class__.__name__ + ' ('
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'

dtype = torch.cuda.FloatTensor
N, D_in, H1, D_out = 10, 28, 19, 28    #batch size, dimensions of network

#randomly initialize weights with matching tensor type to mask
w1 = Variable(torch.randn(D_in, H1).type(dtype), requires_grad=True)
w2 = Variable(torch.randn(H1, D_out).type(dtype), requires_grad=True)

#import data from csv with pandas, strip values from dataframe into array
dataframe = pandas.read_csv("ADOSModule 3.csv", header=0)
```

```
dataset = dataframe.values

#model configuration, dimensions, and hyperparameters
activation = torch.nn.SELU()
model = torch.nn.Sequential(MaskedLayer1(28, 19), #incoder
                            activation,
                            MaskedLayer2(19,28))#decoder
model = model.cuda()
criterion = torch.nn.L1Loss(size_average=True)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

#create arrays to store loss measurements for training and test sets
trainingloss =[]
trainingloss = np.array(trainingloss, dtype =np.float64)
testloss = []
trainingloss = np.array(testloss, dtype =np.float64)

#10 folds for cross validation, data set is split 90/10 training test, placed into
tensor on GPU
for i in range(10):
    train, test = train_test_split(dataset, test_size=0.1)
    Module 3train = torch.from_numpy(train)
```

```
    Module 3test = torch.from_numpy(test)
    Module 3train = Module 3train.type(torch.FloatTensor)
    Module 2train = Module 3train.cuda()
    Module 3test = Module 3test.type(torch.FloatTensor)
    Module 3test = Module 3test.cuda()
    x = torch.autograd.Variable(Module 3train)     #input
    y = torch.autograd.Variable(Module 3train, requires_grad=False)     #target,
same as input
    testset = torch.autograd.Variable(Module 3test, requires_grad = False)     #test
set
    #100,000 epochs per validation fold, model minimizes difference between
target and prediction, training and test losses added to respective arrays every
epoch
        for t in range(100000):
        y_pred = model(x)
        loss = criterion(y_pred, y)
        print(i, t, loss.data[0])
        trainingloss = np.append(trainingloss, loss.data[0])
        optimizer.zero_grad()        loss.backward()
        optimizer.step()
    testing = model(testset)
    val_loss = criterion(testing, testset)
```

```
    print(val_loss.data[0])
    testloss = np.append(testloss,val_loss.data[0])

#remove encoder weights from model by iterating through model.parameters and
removing data; break loop after encoder weights removed
#CPU tensor to numpy array to pandas dataframe to comma-separated value file
for param in model.parameters():
    cpuweights = param.data.cpu()
    npweights = cpuweights.numpy()
    df = pandas.DataFrame(npweights)
    df.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module 3/training1/Module
3weights.csv")
    break

#take reconstruction (90% of data), move it to CPU tensor, then to numpy array,
then to pandas dataframe, then to csv with headers listed
reconstruction = y_pred.data.cpu()
reconstruction = y_pred.data.numpy()
reconstructionset = pandas.DataFrame(reconstruction, columns=
['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CONVS','DGES
T','UEYE','FACEO','LLNVC','SHRNJ', 'EMPTH', 'INSIG', 'QSOV', 'QSRES',
'ARSOC', 'OQRAP', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'TOPIC','RITL',
```

```
'ACTVE', 'AGG', 'ANXTY' ])
reconstructionset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
3/training1/Module 3_reconstruction.csv")

#target goes through same pipeline as reconstruction
target = y.data.cpu().numpy()
targetset = pandas.DataFrame(target, columns=
['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CONVS','DGES
T','UEYE','FACEO','LLNVC','SHRNJ', 'EMPTH', 'INSIG', 'QSOV', 'QSRES',
'ARSOC', 'OQRAP', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'TOPIC','RITL',
'ACTVE', 'AGG', 'ANXTY' ])
targetset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
3/training1/Module 3_y.csv")

#testing reconstruction
test = testset.data.cpu().numpy()
testset = pandas.DataFrame(test, columns=
['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CONVS','DGES
T','UEYE','FACEO','LLNVC','SHRNJ', 'EMPTH', 'INSIG', 'QSOV', 'QSRES',
'ARSOC', 'OQRAP', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'TOPIC','RITL',
'ACTVE', 'AGG', 'ANXTY' ])
testset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
```

```
3/training1/Module 3_testset.csv")


#testing target
tests = testing.data.cpu().numpy()
testingset = pandas.DataFrame(tests, columns=
['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CONVS','DGES
T','UEYE','FACEO','LLNVC','SHRNJ', 'EMPTH', 'INSIG', 'QSOV', 'QSRES',
'ARSOC', 'OQRAP', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'TOPIC','RITL',
'ACTVE', 'AGG', 'ANXTY' ])
testingset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
3/training1/Module 3_testeing.csv")
```

Building module 3 Autoencoder with Hidden layer of size 20

```
#import all dependencies

import pandas

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

import torch

from torch.autograd import Variable

from torch import nnfrom torch.optim import lr_scheduler

from torch.nn import Parameter

import numpy as np

import math


#Create mask 28-20 matrix

mask2 = np.matrix('1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0; 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0; 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0;  1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0;  1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0;  1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0;  0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0
0; 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0
0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0 0 0
0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0 0 0 0 0
0 1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0 0 0 0 0 0 1
```

```
1 1 1 1 1 1 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 1 1
1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1')

#create transposed mask
mask1 = mask2.transpose()

torch_mask1 = torch.from_numpy(mask1)    #convert mask to torch tensor
torchmask1 = torch_mask1.type(torch.FloatTensor)    #match tensor type with
weights
torchmask1 = torchmask1.cuda()    #convert to GPU CUDA tensor
mask1 = torch.autograd.Variable(torchmask1, requires_grad=True)  #wrap in
variable to add to model

torch_mask2 = torch.from_numpy(mask2)    #convert mask to torch tensor
torchmask2 = torch_mask2.type(torch.FloatTensor)    #match tensor type with
weights
torchmask2 = torchmask2.cuda()    #convert to GPU CUDA tensor
mask2 = torch.autograd.Variable(torchmask2, requires_grad=True)    #wrap in
variable to add to model
```

```
#create special masked layer for encoder by altering feedforward layer's forward
pass
class MaskedLayer1(nn.Module):
    def __init__(self, in_features, out_features, bias=True):
        super(MaskedLayer1, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()

    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)

    def forward(self, input):
```

```
        return input.mm(self.weight*mask1)


    def __repr__(self):
        return self.__class__.__name__ + ' ('
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'



#create special masked layer for decoder by altering feedforward layer's forward
pass
class MaskedLayer2(nn.Module):
    def __init__(self, in_features, out_features, bias=True):
        super(MaskedLayer2, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()
```

```
    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)

    def forward(self, input):
        return input.mm(self.weight*mask2)
    def __repr__(self):
        return self.__class__.__name__ + ' ('
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'

dtype = torch.cuda.FloatTensor
N, D_in, H1, D_out = 10, 28, 20, 28    #batch size, dimensions of network

#randomly initialize weights with matching tensor type to mask
w1 = Variable(torch.randn(D_in, H1).type(dtype), requires_grad=True)
w2 = Variable(torch.randn(H1, D_out).type(dtype), requires_grad=True)

#import data from csv with pandas, strip values from dataframe into array
dataframe = pandas.read_csv("ADOSModule 3.csv", header=0)
```

```
dataset = dataframe.values

#model configuration, dimensions, and hyperparameters
activation = torch.nn.SELU()
model = torch.nn.Sequential(MaskedLayer1(28, 20), #incoder
                            activation,
                            MaskedLayer2(20,28))#decoder
model = model.cuda()
criterion = torch.nn.L1Loss(size_average=True)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

#create arrays to store loss measurements for training and test sets
trainingloss =[]
trainingloss = np.array(trainingloss, dtype =np.float64)
testloss = []
trainingloss = np.array(testloss, dtype =np.float64)

#10 folds for cross validation, data set is split 90/10 training test, placed into
tensor on GPU
for i in range(10):
    train, test = train_test_split(dataset, test_size=0.1)
    Module 3train = torch.from_numpy(train)
```

```
    Module 3test = torch.from_numpy(test)
    Module 3train = Module 3train.type(torch.FloatTensor)
    Module 2train = Module 3train.cuda()
    Module 3test = Module 3test.type(torch.FloatTensor)
    Module 3test = Module 3test.cuda()
    x = torch.autograd.Variable(Module 3train)     #input
    y = torch.autograd.Variable(Module 3train, requires_grad=False)     #target,
same as input
    testset = torch.autograd.Variable(Module 3test, requires_grad = False)     #test
set
    #100,000 epochs per validation fold, model minimizes difference between
target and prediction, training and test losses added to respective arrays every
epoch
        for t in range(100000):
        y_pred = model(x)
        loss = criterion(y_pred, y)
        print(i, t, loss.data[0])
        trainingloss = np.append(trainingloss, loss.data[0])
        optimizer.zero_grad()        loss.backward()
        optimizer.step()
    testing = model(testset)
    val_loss = criterion(testing, testset)
```

```
    print(val_loss.data[0])
    testloss = np.append(testloss,val_loss.data[0])

#remove encoder weights from model by iterating through model.parameters and
removing data; break loop after encoder weights removed
#CPU tensor to numpy array to pandas dataframe to comma-separated value file
for param in model.parameters():
    cpuweights = param.data.cpu()
    npweights = cpuweights.numpy()
    df = pandas.DataFrame(npweights)
    df.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module 3/training1/Module
3weights.csv")
    break

#take reconstruction (90% of data), move it to CPU tensor, then to numpy array,
then to pandas dataframe, then to csv with headers listed
reconstruction = y_pred.data.cpu()
reconstruction = y_pred.data.numpy()
reconstructionset = pandas.DataFrame(reconstruction, columns=
['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CONVS','DGES
T','UEYE','FACEO','LLNVC','SHRNJ', 'EMPTH', 'INSIG', 'QSOV', 'QSRES',
'ARSOC', 'OQRAP', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'TOPIC','RITL',
```

```
'ACTVE', 'AGG', 'ANXTY' ])
reconstructionset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
3/training1/Module 3_reconstruction.csv")

#target goes through same pipeline as reconstruction
target = y.data.cpu().numpy()
targetset = pandas.DataFrame(target, columns=
['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CONVS','DGES
T','UEYE','FACEO','LLNVC','SHRNJ', 'EMPTH', 'INSIG', 'QSOV', 'QSRES',
'ARSOC', 'OQRAP', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'TOPIC','RITL',
'ACTVE', 'AGG', 'ANXTY' ])
targetset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
3/training1/Module 3_y.csv")

#testing reconstruction
test = testset.data.cpu().numpy()
testset = pandas.DataFrame(test, columns=
['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CONVS','DGES
T','UEYE','FACEO','LLNVC','SHRNJ', 'EMPTH', 'INSIG', 'QSOV', 'QSRES',
'ARSOC', 'OQRAP', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'TOPIC','RITL',
'ACTVE', 'AGG', 'ANXTY' ])
testset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
```

```
3/training1/Module 3_testset.csv")


#testing target
tests = testing.data.cpu().numpy()
testingset = pandas.DataFrame(tests, columns=
['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CONVS','DGES
T','UEYE','FACEO','LLNVC','SHRNJ', 'EMPTH', 'INSIG', 'QSOV', 'QSRES',
'ARSOC', 'OQRAP', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'TOPIC','RITL',
'ACTVE', 'AGG', 'ANXTY' ])
testingset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
3/training1/Module 3_testeing.csv")
```

Building module 3 Autoencoder with Hidden layer of size 21

```
#import all dependencies

import pandas

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

import torch

from torch.autograd import Variable

from torch import nnfrom torch.optim import lr_scheduler

from torch.nn import Parameter

import numpy as np

import math


#Create mask 28-21 matrix

mask2 = np.matrix('1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 1 0 0 0 0

0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 1 0 0 0 0

0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 1 0 0 0 0

0 0 0 0 0 0 0 0 0 0;  1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0;  1 1 1 1 1 1 1 0 0 0 0

0 0 0 0 0 0 0 0 0 0;  1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0;  0 0 0 0 0 0 0 1 1 1 1

1 1 1 1 0 0 0 0 0 0; 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0; 0 0 0 0 0 0 0 1 1 1 1

1 1 1 1 0 0 0 0 0 0; 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0; 0 0 0 0 0 0 0 1 1 1 1

1 1 1 1 0 0 0 0 0 0; 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0; 0 0 0 0 0 0 0 1 1 1 1

1 1 1 1 0 0 0 0 0 0; 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0; 0 0 0 0 0 0 0 1 1 1 1
```

```
1 1 1 1 0 0 0 0 0 0; 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 ')

#create transposed mask
mask1 = mask2.transpose()

torch_mask1 = torch.from_numpy(mask1)     #convert mask to torch tensor
torchmask1 = torch_mask1.type(torch.FloatTensor)     #match tensor type with
weights
torchmask1 = torchmask1.cuda()     #convert to GPU CUDA tensor
mask1 = torch.autograd.Variable(torchmask1, requires_grad=True)  #wrap in
variable to add to model

torch_mask2 = torch.from_numpy(mask2)     #convert mask to torch tensor
torchmask2 = torch_mask2.type(torch.FloatTensor)     #match tensor type with
weights
torchmask2 = torchmask2.cuda()     #convert to GPU CUDA tensor
mask2 = torch.autograd.Variable(torchmask2, requires_grad=True)     #wrap in
```

```
variable to add to model


#create special masked layer for encoder by altering feedforward layer's forward
pass
class MaskedLayer1(nn.Module):
    def __init__(self, in_features, out_features, bias=True):
        super(MaskedLayer1, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()


    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)
```

```
    def forward(self, input):
        return input.mm(self.weight*mask1)


    def __repr__(self):
        return self.__class__.__name__ + ' ('
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'



#create special masked layer for decoder by altering feedforward layer's forward
pass
class MaskedLayer2(nn.Module):
    def __init__(self, in_features, out_features, bias=True):
        super(MaskedLayer2, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()
```

```
    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)

    def forward(self, input):
        return input.mm(self.weight*mask2)
    def __repr__(self):
        return self.__class__.__name__ + ' ('
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'

dtype = torch.cuda.FloatTensor
N, D_in, H1, D_out = 10, 28, 21, 28    #batch size, dimensions of network

#randomly initialize weights with matching tensor type to mask
w1 = Variable(torch.randn(D_in, H1).type(dtype), requires_grad=True)
w2 = Variable(torch.randn(H1, D_out).type(dtype), requires_grad=True)

#import data from csv with pandas, strip values from dataframe into array
```

```
dataframe = pandas.read_csv("ADOSModule 3.csv", header=0)
dataset = dataframe.values

#model configuration, dimensions, and hyperparameters
activation = torch.nn.SELU()
model = torch.nn.Sequential(MaskedLayer1(28, 21), #incoder
                            activation,
                            MaskedLayer2(21,28))#decoder
model = model.cuda()
criterion = torch.nn.L1Loss(size_average=True)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

#create arrays to store loss measurements for training and test sets
trainingloss =[]
trainingloss = np.array(trainingloss, dtype =np.float64)
testloss = []
trainingloss = np.array(testloss, dtype =np.float64)

#10 folds for cross validation, data set is split 90/10 training test, placed into
tensor on GPU
for i in range(10):
    train, test = train_test_split(dataset, test_size=0.1)
```

```
    Module 3train = torch.from_numpy(train)
    Module 3test = torch.from_numpy(test)
    Module 3train = Module 3train.type(torch.FloatTensor)
    Module 2train = Module 3train.cuda()
    Module 3test = Module 3test.type(torch.FloatTensor)
    Module 3test = Module 3test.cuda()
    x = torch.autograd.Variable(Module 3train)     #input
    y = torch.autograd.Variable(Module 3train, requires_grad=False)     #target,
same as input
    testset = torch.autograd.Variable(Module 3test, requires_grad = False)     #test
set
    #100,000 epochs per validation fold, model minimizes difference between
target and prediction, training and test losses added to respective arrays every
epoch
        for t in range(100000):
        y_pred = model(x)
        loss = criterion(y_pred, y)
        print(i, t, loss.data[0])
        trainingloss = np.append(trainingloss, loss.data[0])
        optimizer.zero_grad()       loss.backward()
        optimizer.step()
    testing = model(testset)
```

```
    val_loss = criterion(testing, testset)
    print(val_loss.data[0])
    testloss = np.append(testloss,val_loss.data[0])

#remove encoder weights from model by iterating through model.parameters and
removing data; break loop after encoder weights removed
#CPU tensor to numpy array to pandas dataframe to comma-separated value file
for param in model.parameters():
    cpuweights = param.data.cpu()
    npweights = cpuweights.numpy()
    df = pandas.DataFrame(npweights)
    df.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module 3/training1/Module
3weights.csv")
    break

#take reconstruction (90% of data), move it to CPU tensor, then to numpy array,
then to pandas dataframe, then to csv with headers listed
reconstruction = y_pred.data.cpu()
reconstruction = y_pred.data.numpy()
reconstructionset = pandas.DataFrame(reconstruction, columns=
['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CONVS','DGES
T','UEYE','FACEO','LLNVC','SHRNJ', 'EMPTH', 'INSIG', 'QSOV', 'QSRES',
```

```
'ARSOC', 'OQRAP', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'TOPIC','RITL',
'ACTVE', 'AGG', 'ANXTY' ])
reconstructionset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
3/training1/Module 3_reconstruction.csv")

#target goes through same pipeline as reconstruction
target = y.data.cpu().numpy()
targetset = pandas.DataFrame(target, columns=
['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CONVS','DGES
T','UEYE','FACEO','LLNVC','SHRNJ', 'EMPTH', 'INSIG', 'QSOV', 'QSRES',
'ARSOC', 'OQRAP', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'TOPIC','RITL',
'ACTVE', 'AGG', 'ANXTY' ])
targetset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
3/training1/Module 3_y.csv")

#testing reconstruction
test = testset.data.cpu().numpy()
testset = pandas.DataFrame(test, columns=
['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CONVS','DGES
T','UEYE','FACEO','LLNVC','SHRNJ', 'EMPTH', 'INSIG', 'QSOV', 'QSRES',
'ARSOC', 'OQRAP', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'TOPIC','RITL',
'ACTVE', 'AGG', 'ANXTY' ])
```

```
testset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
3/training1/Module 3_testset.csv")


#testing target
tests = testing.data.cpu().numpy()
testingset = pandas.DataFrame(tests, columns=
['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CONVS','DGES
T','UEYE','FACEO','LLNVC','SHRNJ', 'EMPTH', 'INSIG', 'QSOV', 'QSRES',
'ARSOC', 'OQRAP', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'TOPIC','RITL',
'ACTVE', 'AGG', 'ANXTY' ])
testingset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
3/training1/Module 3_testeing.csv")
```

Building module 3 Autoencoder with Hidden layer of size 22

```
#import all dependencies

import pandas

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

import torch

from torch.autograd import Variable

from torch import nnfrom torch.optim import lr_scheduler

from torch.nn import Parameter

import numpy as np

import math


#Create mask 28-22 matrix

mask2 = np.matrix('1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0;  1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0;  1 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0;  1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0;  0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 ; 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0; 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0
```

```
0 0 0 0 0; 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1; 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 ')

#create transposed mask
mask1 = mask2.transpose()

torch_mask1 = torch.from_numpy(mask1)    #convert mask to torch tensor
torchmask1 = torch_mask1.type(torch.FloatTensor)    #match tensor type with
weights
torchmask1 = torchmask1.cuda()    #convert to GPU CUDA tensor
mask1 = torch.autograd.Variable(torchmask1, requires_grad=True)  #wrap in
variable to add to model

torch_mask2 = torch.from_numpy(mask2)    #convert mask to torch tensor
torchmask2 = torch_mask2.type(torch.FloatTensor)    #match tensor type with
weights
torchmask2 = torchmask2.cuda()    #convert to GPU CUDA tensor
mask2 = torch.autograd.Variable(torchmask2, requires_grad=True)    #wrap in
```

```
variable to add to model


#create special masked layer for encoder by altering feedforward layer's forward
pass
class MaskedLayer1(nn.Module):
    def __init__(self, in_features, out_features, bias=True):
        super(MaskedLayer1, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()


    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)
```

```
    def forward(self, input):
        return input.mm(self.weight*mask1)


    def __repr__(self):
        return self.__class__.__name__ + ' ('
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'



#create special masked layer for decoder by altering feedforward layer's forward
pass
class MaskedLayer2(nn.Module):
    def __init__(self, in_features, out_features, bias=True):
        super(MaskedLayer2, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()
```

```
    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)

    def forward(self, input):
        return input.mm(self.weight*mask2)
    def __repr__(self):
        return self.__class__.__name__ + ' ('
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'

dtype = torch.cuda.FloatTensor
N, D_in, H1, D_out = 10, 28, 22, 28    #batch size, dimensions of network

#randomly initialize weights with matching tensor type to mask
w1 = Variable(torch.randn(D_in, H1).type(dtype), requires_grad=True)
w2 = Variable(torch.randn(H1, D_out).type(dtype), requires_grad=True)

#import data from csv with pandas, strip values from dataframe into array
```

```
dataframe = pandas.read_csv("ADOSModule 3.csv", header=0)
dataset = dataframe.values

#model configuration, dimensions, and hyperparameters
activation = torch.nn.SELU()
model = torch.nn.Sequential(MaskedLayer1(28, 22), #incoder
                            activation,
                            MaskedLayer2(22,28))#decoder
model = model.cuda()
criterion = torch.nn.L1Loss(size_average=True)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

#create arrays to store loss measurements for training and test sets
trainingloss =[]
trainingloss = np.array(trainingloss, dtype =np.float64)
testloss = []
trainingloss = np.array(testloss, dtype =np.float64)

#10 folds for cross validation, data set is split 90/10 training test, placed into
tensor on GPU
for i in range(10):
    train, test = train_test_split(dataset, test_size=0.1)
```

```
    Module 3train = torch.from_numpy(train)
    Module 3test = torch.from_numpy(test)
    Module 3train = Module 3train.type(torch.FloatTensor)
    Module 2train = Module 3train.cuda()
    Module 3test = Module 3test.type(torch.FloatTensor)
    Module 3test = Module 3test.cuda()
    x = torch.autograd.Variable(Module 3train)     #input
    y = torch.autograd.Variable(Module 3train, requires_grad=False)     #target,
same as input
    testset = torch.autograd.Variable(Module 3test, requires_grad = False)     #test
set
    #100,000 epochs per validation fold, model minimizes difference between
target and prediction, training and test losses added to respective arrays every
epoch
        for t in range(100000):
        y_pred = model(x)
        loss = criterion(y_pred, y)
        print(i, t, loss.data[0])
        trainingloss = np.append(trainingloss, loss.data[0])
        optimizer.zero_grad()       loss.backward()
        optimizer.step()
    testing = model(testset)
```

```
    val_loss = criterion(testing, testset)
    print(val_loss.data[0])
    testloss = np.append(testloss,val_loss.data[0])

#remove encoder weights from model by iterating through model.parameters and
removing data; break loop after encoder weights removed
#CPU tensor to numpy array to pandas dataframe to comma-separated value file
for param in model.parameters():
    cpuweights = param.data.cpu()
    npweights = cpuweights.numpy()
    df = pandas.DataFrame(npweights)
    df.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module 3/training1/Module
3weights.csv")
    break

#take reconstruction (90% of data), move it to CPU tensor, then to numpy array,
then to pandas dataframe, then to csv with headers listed
reconstruction = y_pred.data.cpu()
reconstruction = y_pred.data.numpy()
reconstructionset = pandas.DataFrame(reconstruction, columns=
['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CONVS','DGES
T','UEYE','FACEO','LLNVC','SHRNJ', 'EMPTH', 'INSIG', 'QSOV', 'QSRES',
```

```
'ARSOC', 'OQRAP', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'TOPIC','RITL',
'ACTVE', 'AGG', 'ANXTY' ])
reconstructionset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
3/training1/Module 3_reconstruction.csv")

#target goes through same pipeline as reconstruction
target = y.data.cpu().numpy()
targetset = pandas.DataFrame(target, columns=
['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CONVS','DGES
T','UEYE','FACEO','LLNVC','SHRNJ', 'EMPTH', 'INSIG', 'QSOV', 'QSRES',
'ARSOC', 'OQRAP', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'TOPIC','RITL',
'ACTVE', 'AGG', 'ANXTY' ])
targetset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
3/training1/Module 3_y.csv")

#testing reconstruction
test = testset.data.cpu().numpy()
testset = pandas.DataFrame(test, columns=
['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CONVS','DGES
T','UEYE','FACEO','LLNVC','SHRNJ', 'EMPTH', 'INSIG', 'QSOV', 'QSRES',
'ARSOC', 'OQRAP', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'TOPIC','RITL',
'ACTVE', 'AGG', 'ANXTY' ])
```

```
testset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
3/training1/Module 3_testset.csv")


#testing target
tests = testing.data.cpu().numpy()
testingset = pandas.DataFrame(tests, columns=
['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CONVS','DGES
T','UEYE','FACEO','LLNVC','SHRNJ', 'EMPTH', 'INSIG', 'QSOV', 'QSRES',
'ARSOC', 'OQRAP', 'IMGCR', 'USENS', 'OMAN', 'SELFINJ', 'TOPIC','RITL',
'ACTVE', 'AGG', 'ANXTY' ])
testingset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
3/training1/Module 3_testeing.csv")
```

Building module 4 Autoencoder with Hidden layer of size 19

```
#import all dependencies

import pandas

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

import torch

from torch.autograd import Variable

from torch import nnfrom torch.optim import lr_scheduler

from torch.nn import Parameter

import numpy as np

import math


#Create mask 31-19 matrix

mask2 = np.matrix('1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;1 1
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0; 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1
1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
1 1 1 1 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0; 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0; 0 0
```

```
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1; 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
1')

#create transposed mask
mask1 = mask2.transpose()

torch_mask1 = torch.from_numpy(mask1)    #convert mask to torch tensor
torchmask1 = torch_mask1.type(torch.FloatTensor)    #match tensor type with
weights
torchmask1 = torchmask1.cuda()    #convert to GPU CUDA tensor
mask1 = torch.autograd.Variable(torchmask1, requires_grad=True)  #wrap in
variable to add to model

torch_mask2 = torch.from_numpy(mask2)    #convert mask to torch tensor
torchmask2 = torch_mask2.type(torch.FloatTensor)    #match tensor type with
weights
torchmask2 = torchmask2.cuda()    #convert to GPU CUDA tensor
mask2 = torch.autograd.Variable(torchmask2, requires_grad=True)    #wrap in
```

```
variable to add to model


#create special masked layer for encoder by altering feedforward layer's forward
pass
class MaskedLayer1(nn.Module):
    def __init__(self, in_features, out_features, bias=True):
        super(MaskedLayer1, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()


    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)
```

```
    def forward(self, input):
        return input.mm(self.weight*mask1)


    def __repr__(self):
        return self.__class__.__name__ + ' ('
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'



#create special masked layer for decoder by altering feedforward layer's forward
pass
class MaskedLayer2(nn.Module):
    def __init__(self, in_features, out_features, bias=True):
        super(MaskedLayer2, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()
```

```
    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)

    def forward(self, input):
        return input.mm(self.weight*mask2)
    def __repr__(self):
        return self.__class__.__name__ + ' ('
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'

dtype = torch.cuda.FloatTensor
N, D_in, H1, D_out = 10, 31, 19, 31    #batch size, dimensions of network

#randomly initialize weights with matching tensor type to mask
w1 = Variable(torch.randn(D_in, H1).type(dtype), requires_grad=True)
w2 = Variable(torch.randn(H1, D_out).type(dtype), requires_grad=True)

#import data from csv with pandas, strip values from dataframe into array
```

```
dataframe = pandas.read_csv("ADOSModule 4.csv", header=0)
dataset = dataframe.values

#model configuration, dimensions, and hyperparameters
activation = torch.nn.SELU()
model = torch.nn.Sequential(MaskedLayer1(31, 19), #incoder
                            activation,
                            MaskedLayer2(19,31))#decoder
model = model.cuda()
criterion = torch.nn.L1Loss(size_average=True)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

#create arrays to store loss measurements for training and test sets
trainingloss =[]
trainingloss = np.array(trainingloss, dtype =np.float64)
testloss = []
trainingloss = np.array(testloss, dtype =np.float64)

#10 folds for cross validation, data set is split 90/10 training test, placed into
tensor on GPU
for i in range(10):
    train, test = train_test_split(dataset, test_size=0.1)
```

```
    Module 4train = torch.from_numpy(train)
    Module 4test = torch.from_numpy(test)
    Module 4train = Module 4train.type(torch.FloatTensor)
    Module 4train = Module 4train.cuda()
    Module 4test = Module 4test.type(torch.FloatTensor)
    Module 4test = Module 4test.cuda()
    x = torch.autograd.Variable(Module 4train)     #input
    y = torch.autograd.Variable(Module 4train, requires_grad=False)     #target,
same as input
    testset = torch.autograd.Variable(Module 4test, requires_grad = False)     #test
set
    #100,000 epochs per validation fold, model minimizes difference between
target and prediction, training and test losses added to respective arrays every
epoch
        for t in range(100000):
        y_pred = model(x)
        loss = criterion(y_pred, y)
        print(i, t, loss.data[0])
        trainingloss = np.append(trainingloss, loss.data[0])
        optimizer.zero_grad()       loss.backward()
        optimizer.step()
    testing = model(testset)
```

```
    val_loss = criterion(testing, testset)
    print(val_loss.data[0])
    testloss = np.append(testloss,val_loss.data[0])

#remove encoder weights from model by iterating through model.parameters and
removing data; break loop after encoder weights removed
#CPU tensor to numpy array to pandas dataframe to comma-separated value file
for param in model.parameters():
    cpuweights = param.data.cpu()
    npweights = cpuweights.numpy()
    df = pandas.DataFrame(npweights)
    df.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module 4/training1/Module
4weights.csv")
    break

#take reconstruction (90% of data), move it to CPU tensor, then to numpy array,
then to pandas dataframe, then to csv with headers listed
reconstruction = y_pred.data.cpu()
reconstruction = y_pred.data.numpy()
reconstructionset = pandas.DataFrame(reconstruction, columns=
['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CONVS','DGES
T','EGEST','UEYE','FACEO','LLNVC','SEI','CAFF','EMPTH','INSIG','RESP','QSO
```

```
V','QSRES','ARSOC','OQRAP','IMGCR','USENS','OMAN','SELFINJ','TOPIC','RIT
L','ACTVE','AGG','ANXTY'])
reconstructionset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
4/training1/Module 4_reconstruction.csv")

#target goes through same pipeline as reconstruction
target = y.data.cpu().numpy()
targetset = pandas.DataFrame(target, columns=
['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CONVS','DGES
T','EGEST','UEYE','FACEO','LLNVC','SEI','CAFF','EMPTH','INSIG','RESP','QSO
V','QSRES','ARSOC','OQRAP','IMGCR','USENS','OMAN','SELFINJ','TOPIC','RIT
L','ACTVE','AGG','ANXTY'])
targetset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
4/training1/Module 4_y.csv")

#testing reconstruction
test = testset.data.cpu().numpy()
testset = pandas.DataFrame(test, columns=
['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CONVS','DGES
T','EGEST','UEYE','FACEO','LLNVC','SEI','CAFF','EMPTH','INSIG','RESP','QSO
V','QSRES','ARSOC','OQRAP','IMGCR','USENS','OMAN','SELFINJ','TOPIC','RIT
L','ACTVE','AGG','ANXTY'])
```

```
testset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
4/training1/Module 4_testset.csv")

#testing target
tests = testing.data.cpu().numpy()
testingset = pandas.DataFrame(tests,
columns=['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CON
VS','DGEST','EGEST','UEYE','FACEO','LLNVC','SEI','CAFF','EMPTH','INSIG','RE
SP','QSOV','QSRES','ARSOC','OQRAP','IMGCR','USENS','OMAN','SELFINJ','TO
PIC','RITL','ACTVE','AGG','ANXTY'])
testingset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
4/training1/Module 4_testeing.csv")
```

Building module 4 Autoencoder with Hidden layer of size 20

```
#import all dependencies

import pandas

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

import torch

from torch.autograd import Variable

from torch import nnfrom torch.optim import lr_scheduler

from torch.nn import Parameter

import numpy as np

import math


#Create mask 31-20 matrix

mask2 = np.matrix('1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1
1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0; 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1
1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
1 1 1 1 1 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0; 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
1 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0; 0
```

```
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0;  0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1; 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1
1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1')

#create transposed mask
mask1 = mask2.transpose()

torch_mask1 = torch.from_numpy(mask1)    #convert mask to torch tensor
torchmask1 = torch_mask1.type(torch.FloatTensor)    #match tensor type with
weights
torchmask1 = torchmask1.cuda()    #convert to GPU CUDA tensor
mask1 = torch.autograd.Variable(torchmask1, requires_grad=True)  #wrap in
variable to add to model

torch_mask2 = torch.from_numpy(mask2)    #convert mask to torch tensor
torchmask2 = torch_mask2.type(torch.FloatTensor)    #match tensor type with
weights
torchmask2 = torchmask2.cuda()    #convert to GPU CUDA tensor
mask2 = torch.autograd.Variable(torchmask2, requires_grad=True)    #wrap in
```

```
variable to add to model


#create special masked layer for encoder by altering feedforward layer's forward
pass
class MaskedLayer1(nn.Module):
    def __init__(self, in_features, out_features, bias=True):
        super(MaskedLayer1, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()


    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)
```

```
    def forward(self, input):
        return input.mm(self.weight*mask1)


    def __repr__(self):
        return self.__class__.__name__ + ' ('
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'



#create special masked layer for decoder by altering feedforward layer's forward
pass
class MaskedLayer2(nn.Module):
    def __init__(self, in_features, out_features, bias=True):
        super(MaskedLayer2, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()
```

```
    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)

    def forward(self, input):
        return input.mm(self.weight*mask2)
    def __repr__(self):
        return self.__class__.__name__ + ' ('
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'

dtype = torch.cuda.FloatTensor
N, D_in, H1, D_out = 10, 31, 20, 31    #batch size, dimensions of network

#randomly initialize weights with matching tensor type to mask
w1 = Variable(torch.randn(D_in, H1).type(dtype), requires_grad=True)
w2 = Variable(torch.randn(H1, D_out).type(dtype), requires_grad=True)

#import data from csv with pandas, strip values from dataframe into array
```

```
dataframe = pandas.read_csv("ADOSModule 4.csv", header=0)
dataset = dataframe.values

#model configuration, dimensions, and hyperparameters
activation = torch.nn.SELU()
model = torch.nn.Sequential(MaskedLayer1(31, 20), #incoder
                            activation,
                            MaskedLayer2(20,31))#decoder
model = model.cuda()
criterion = torch.nn.L1Loss(size_average=True)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

#create arrays to store loss measurements for training and test sets
trainingloss =[]
trainingloss = np.array(trainingloss, dtype =np.float64)
testloss = []
trainingloss = np.array(testloss, dtype =np.float64)

#10 folds for cross validation, data set is split 90/10 training test, placed into
tensor on GPU
for i in range(10):
    train, test = train_test_split(dataset, test_size=0.1)
```

```
    Module 4train = torch.from_numpy(train)
    Module 4test = torch.from_numpy(test)
    Module 4train = Module 4train.type(torch.FloatTensor)
    Module 4train = Module 4train.cuda()
    Module 4test = Module 4test.type(torch.FloatTensor)
    Module 4test = Module 4test.cuda()
    x = torch.autograd.Variable(Module 4train)     #input
    y = torch.autograd.Variable(Module 4train, requires_grad=False)     #target,
same as input
    testset = torch.autograd.Variable(Module 4test, requires_grad = False)     #test
set
    #100,000 epochs per validation fold, model minimizes difference between
target and prediction, training and test losses added to respective arrays every
epoch
        for t in range(100000):
        y_pred = model(x)
        loss = criterion(y_pred, y)
        print(i, t, loss.data[0])
        trainingloss = np.append(trainingloss, loss.data[0])
        optimizer.zero_grad()       loss.backward()
        optimizer.step()
    testing = model(testset)
```

```
    val_loss = criterion(testing, testset)
    print(val_loss.data[0])
    testloss = np.append(testloss,val_loss.data[0])

#remove encoder weights from model by iterating through model.parameters and
removing data; break loop after encoder weights removed
#CPU tensor to numpy array to pandas dataframe to comma-separated value file
for param in model.parameters():
    cpuweights = param.data.cpu()
    npweights = cpuweights.numpy()
    df = pandas.DataFrame(npweights)
    df.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module 4/training1/Module
4weights.csv")
    break

#take reconstruction (90% of data), move it to CPU tensor, then to numpy array,
then to pandas dataframe, then to csv with headers listed
reconstruction = y_pred.data.cpu()
reconstruction = y_pred.data.numpy()
reconstructionset = pandas.DataFrame(reconstruction, columns=
['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CONVS','DGES
T','EGEST','UEYE','FACEO','LLNVC','SEI','CAFF','EMPTH','INSIG','RESP','QSO
```

```
V','QSRES','ARSOC','OQRAP','IMGCR','USENS','OMAN','SELFINJ','TOPIC','RIT
L','ACTVE','AGG','ANXTY'])
reconstructionset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
4/training1/Module 4_reconstruction.csv")

#target goes through same pipeline as reconstruction
target = y.data.cpu().numpy()
targetset = pandas.DataFrame(target, columns=
['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CONVS','DGES
T','EGEST','UEYE','FACEO','LLNVC','SEI','CAFF','EMPTH','INSIG','RESP','QSO
V','QSRES','ARSOC','OQRAP','IMGCR','USENS','OMAN','SELFINJ','TOPIC','RIT
L','ACTVE','AGG','ANXTY'])
targetset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
4/training1/Module 4_y.csv")

#testing reconstruction
test = testset.data.cpu().numpy()
testset = pandas.DataFrame(test, columns=
['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CONVS','DGES
T','EGEST','UEYE','FACEO','LLNVC','SEI','CAFF','EMPTH','INSIG','RESP','QSO
V','QSRES','ARSOC','OQRAP','IMGCR','USENS','OMAN','SELFINJ','TOPIC','RIT
L','ACTVE','AGG','ANXTY'])
```

```
testset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
4/training1/Module 4_testset.csv")

#testing target
tests = testing.data.cpu().numpy()
testingset = pandas.DataFrame(tests,
columns=['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CON
VS','DGEST','EGEST','UEYE','FACEO','LLNVC','SEI','CAFF','EMPTH','INSIG','RE
SP','QSOV','QSRES','ARSOC','OQRAP','IMGCR','USENS','OMAN','SELFINJ','TO
PIC','RITL','ACTVE','AGG','ANXTY'])
testingset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
4/training1/Module 4_testeing.csv")
```

Building module 4 Autoencoder with Hidden layer of size 21

```
#import all dependencies

import pandas

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

import torch

from torch.autograd import Variable

from torch import nnfrom torch.optim import lr_scheduler

from torch.nn import Parameter

import numpy as np

import math


#Create mask 31-20 matrix

mask2 = np.matrix('1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1
1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0; 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1
1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0; 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
1 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0; 0
```

0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1; 0 0 0 0 0 1 1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1')

```
#create transposed mask
mask1 = mask2.transpose()

torch_mask1 = torch.from_numpy(mask1)    #convert mask to torch tensor
torchmask1 = torch_mask1.type(torch.FloatTensor)    #match tensor type with
weights
torchmask1 = torchmask1.cuda()    #convert to GPU CUDA tensor
mask1 = torch.autograd.Variable(torchmask1, requires_grad=True)  #wrap in
variable to add to model

torch_mask2 = torch.from_numpy(mask2)    #convert mask to torch tensor
torchmask2 = torch_mask2.type(torch.FloatTensor)    #match tensor type with
weights
torchmask2 = torchmask2.cuda()    #convert to GPU CUDA tensor
```

```
mask2 = torch.autograd.Variable(torchmask2, requires_grad=True)     #wrap in
variable to add to model

#create special masked layer for encoder by altering feedforward layer's forward
pass
class MaskedLayer1(nn.Module):
    def __init__(self, in_features, out_features, bias=True):
        super(MaskedLayer1, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()

    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)
```

```
    def forward(self, input):
        return input.mm(self.weight*mask1)

    def __repr__(self):
        return self.__class__.__name__ + ' ('
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'


#create special masked layer for decoder by altering feedforward layer's forward
pass
class MaskedLayer2(nn.Module):
    def __init__(self, in_features, out_features, bias=True):
        super(MaskedLayer2, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
            self.register_parameter('bias', None)
```

```
        self.reset_parameters()


    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)


    def forward(self, input):
        return input.mm(self.weight*mask2)
    def __repr__(self):
        return self.__class__.__name__ + ' ('
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'


dtype = torch.cuda.FloatTensor
N, D_in, H1, D_out = 10, 31, 21, 31    #batch size, dimensions of network


#randomly initialize weights with matching tensor type to mask
w1 = Variable(torch.randn(D_in, H1).type(dtype), requires_grad=True)
w2 = Variable(torch.randn(H1, D_out).type(dtype), requires_grad=True)
```

```
#import data from csv with pandas, strip values from dataframe into array
dataframe = pandas.read_csv("ADOSModule 4.csv", header=0)
dataset = dataframe.values

#model configuration, dimensions, and hyperparameters
activation = torch.nn.SELU()
model = torch.nn.Sequential(MaskedLayer1(31, 21), #incoder
                            activation,
                            MaskedLayer2(21,31))#decoder
model = model.cuda()
criterion = torch.nn.L1Loss(size_average=True)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

#create arrays to store loss measurements for training and test sets
trainingloss =[]
trainingloss = np.array(trainingloss, dtype =np.float64)
testloss = []
trainingloss = np.array(testloss, dtype =np.float64)

#10 folds for cross validation, data set is split 90/10 training test, placed into
tensor on GPU
for i in range(10):
```

```
    train, test = train_test_split(dataset, test_size=0.1)
    Module 4train = torch.from_numpy(train)
    Module 4test = torch.from_numpy(test)
    Module 4train = Module 4train.type(torch.FloatTensor)
    Module 4train = Module 4train.cuda()
    Module 4test = Module 4test.type(torch.FloatTensor)
    Module 4test = Module 4test.cuda()
    x = torch.autograd.Variable(Module 4train)     #input
    y = torch.autograd.Variable(Module 4train, requires_grad=False)     #target,
same as input
    testset = torch.autograd.Variable(Module 4test, requires_grad = False)     #test
set
    #100,000 epochs per validation fold, model minimizes difference between
target and prediction, training and test losses added to respective arrays every
epoch
        for t in range(100000):
        y_pred = model(x)
        loss = criterion(y_pred, y)
        print(i, t, loss.data[0])
        trainingloss = np.append(trainingloss, loss.data[0])
        optimizer.zero_grad()        loss.backward()
        optimizer.step()
```

```
    testing = model(testset)
    val_loss = criterion(testing, testset)
    print(val_loss.data[0])
    testloss = np.append(testloss,val_loss.data[0])

#remove encoder weights from model by iterating through model.parameters and
removing data; break loop after encoder weights removed
#CPU tensor to numpy array to pandas dataframe to comma-separated value file
for param in model.parameters():
    cpuweights = param.data.cpu()
    npweights = cpuweights.numpy()
    df = pandas.DataFrame(npweights)
    df.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module 4/training1/Module
4weights.csv")
    break

#take reconstruction (90% of data), move it to CPU tensor, then to numpy array,
then to pandas dataframe, then to csv with headers listed
reconstruction = y_pred.data.cpu()
reconstruction = y_pred.data.numpy()
reconstructionset = pandas.DataFrame(reconstruction, columns=
['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CONVS','DGES
```

```
T','EGEST','UEYE','FACEO','LLNVC','SEI','CAFF','EMPTH','INSIG','RESP','QSO
V','QSRES','ARSOC','OQRAP','IMGCR','USENS','OMAN','SELFINJ','TOPIC','RIT
L','ACTVE','AGG','ANXTY'])
reconstructionset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
4/training1/Module 4_reconstruction.csv")

#target goes through same pipeline as reconstruction
target = y.data.cpu().numpy()
targetset = pandas.DataFrame(target, columns=
['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CONVS','DGES
T','EGEST','UEYE','FACEO','LLNVC','SEI','CAFF','EMPTH','INSIG','RESP','QSO
V','QSRES','ARSOC','OQRAP','IMGCR','USENS','OMAN','SELFINJ','TOPIC','RIT
L','ACTVE','AGG','ANXTY'])
targetset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
4/training1/Module 4_y.csv")

#testing reconstruction
test = testset.data.cpu().numpy()
testset = pandas.DataFrame(test, columns=
['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CONVS','DGES
T','EGEST','UEYE','FACEO','LLNVC','SEI','CAFF','EMPTH','INSIG','RESP','QSO
V','QSRES','ARSOC','OQRAP','IMGCR','USENS','OMAN','SELFINJ','TOPIC','RIT
```

```
L','ACTVE','AGG','ANXTY'])
testset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
4/training1/Module 4_testset.csv")

#testing target
tests = testing.data.cpu().numpy()
testingset = pandas.DataFrame(tests,
columns=['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CON
VS','DGEST','EGEST','UEYE','FACEO','LLNVC','SEI','CAFF','EMPTH','INSIG','RE
SP','QSOV','QSRES','ARSOC','OQRAP','IMGCR','USENS','OMAN','SELFINJ','TO
PIC','RITL','ACTVE','AGG','ANXTY'])
testingset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
4/training1/Module 4_testeing.csv")
```

Building module 4 Autoencoder with Hidden layer of size 22

```
#import all dependencies

import pandas

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

import torch

from torch.autograd import Variable

from torch import nnfrom torch.optim import lr_scheduler

from torch.nn import Parameter

import numpy as np

import math


#Create mask 31-22 matrix

mask2 = np.matrix('1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1
1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0; 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1
1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0; 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
1 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0; 0
```

```
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 0 1 1 1
1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0; 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1; 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 1')

#create transposed mask
mask1 = mask2.transpose()

torch_mask1 = torch.from_numpy(mask1)     #convert mask to torch tensor
torchmask1 = torch_mask1.type(torch.FloatTensor)     #match tensor type with
weights
torchmask1 = torchmask1.cuda()     #convert to GPU CUDA tensor
mask1 = torch.autograd.Variable(torchmask1, requires_grad=True)  #wrap in
variable to add to model

torch_mask2 = torch.from_numpy(mask2)     #convert mask to torch tensor
torchmask2 = torch_mask2.type(torch.FloatTensor)     #match tensor type with
weights
```

```
torchmask2 = torchmask2.cuda()    #convert to GPU CUDA tensor
mask2 = torch.autograd.Variable(torchmask2, requires_grad=True)    #wrap in
variable to add to model

#create special masked layer for encoder by altering feedforward layer's forward
pass
class MaskedLayer1(nn.Module):
    def __init__(self, in_features, out_features, bias=True):
        super(MaskedLayer1, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()

    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
```

```
            self.bias.data.uniform_(-stdv, stdv)


    def forward(self, input):
        return input.mm(self.weight*mask1)


    def __repr__(self):
        return self.__class__.__name__ + ' ('
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'



#create special masked layer for decoder by altering feedforward layer's forward
pass
class MaskedLayer2(nn.Module):
    def __init__(self, in_features, out_features, bias=True):
        super(MaskedLayer2, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.Tensor(out_features))
        else:
```

```
            self.register_parameter('bias', None)
        self.reset_parameters()

    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)

    def forward(self, input):
        return input.mm(self.weight*mask2)
    def __repr__(self):
        return self.__class__.__name__ + ' ('
            + str(self.in_features) + ' -> '
            + str(self.out_features) + ')'

dtype = torch.cuda.FloatTensor
N, D_in, H1, D_out = 10, 31, 22, 31    #batch size, dimensions of network

#randomly initialize weights with matching tensor type to mask
w1 = Variable(torch.randn(D_in, H1).type(dtype), requires_grad=True)
w2 = Variable(torch.randn(H1, D_out).type(dtype), requires_grad=True)
```

```
#import data from csv with pandas, strip values from dataframe into array
dataframe = pandas.read_csv("ADOSModule 4.csv", header=0)
dataset = dataframe.values

#model configuration, dimensions, and hyperparameters
activation = torch.nn.SELU()
model = torch.nn.Sequential(MaskedLayer1(31, 22), #incoder
                    activation,
                    MaskedLayer2(22,31))#decoder
model = model.cuda()
criterion = torch.nn.L1Loss(size_average=True)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

#create arrays to store loss measurements for training and test sets
trainingloss =[]
trainingloss = np.array(trainingloss, dtype =np.float64)
testloss = []
trainingloss = np.array(testloss, dtype =np.float64)

#10 folds for cross validation, data set is split 90/10 training test, placed into
tensor on GPU
```

```
for i in range(10):
    train, test = train_test_split(dataset, test_size=0.1)
    Module 4train = torch.from_numpy(train)
    Module 4test = torch.from_numpy(test)
    Module 4train = Module 4train.type(torch.FloatTensor)
    Module 4train = Module 4train.cuda()
    Module 4test = Module 4test.type(torch.FloatTensor)
    Module 4test = Module 4test.cuda()
    x = torch.autograd.Variable(Module 4train)     #input
    y = torch.autograd.Variable(Module 4train, requires_grad=False)     #target,
same as input
    testset = torch.autograd.Variable(Module 4test, requires_grad = False)     #test
set
    #100,000 epochs per validation fold, model minimizes difference between
target and prediction, training and test losses added to respective arrays every
epoch
        for t in range(100000):
        y_pred = model(x)
        loss = criterion(y_pred, y)
        print(i, t, loss.data[0])
        trainingloss = np.append(trainingloss, loss.data[0])
        optimizer.zero_grad()        loss.backward()
```

```
        optimizer.step()
    testing = model(testset)
    val_loss = criterion(testing, testset)
    print(val_loss.data[0])
    testloss = np.append(testloss,val_loss.data[0])

#remove encoder weights from model by iterating through model.parameters and
removing data; break loop after encoder weights removed
#CPU tensor to numpy array to pandas dataframe to comma-separated value file
for param in model.parameters():
    cpuweights = param.data.cpu()
    npweights = cpuweights.numpy()
    df = pandas.DataFrame(npweights)
    df.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module 4/training1/Module
4weights.csv")
    break

#take reconstruction (90% of data), move it to CPU tensor, then to numpy array,
then to pandas dataframe, then to csv with headers listed
reconstruction = y_pred.data.cpu()
reconstruction = y_pred.data.numpy()
reconstructionset = pandas.DataFrame(reconstruction, columns=
```

```
['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CONVS','DGES
T','EGEST','UEYE','FACEO','LLNVC','SEI','CAFF','EMPTH','INSIG','RESP','QSO
V','QSRES','ARSOC','OQRAP','IMGCR','USENS','OMAN','SELFINJ','TOPIC','RIT
L','ACTVE','AGG','ANXTY'])
reconstructionset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
4/training1/Module 4_reconstruction.csv")

#target goes through same pipeline as reconstruction
target = y.data.cpu().numpy()
targetset = pandas.DataFrame(target, columns=
['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CONVS','DGES
T','EGEST','UEYE','FACEO','LLNVC','SEI','CAFF','EMPTH','INSIG','RESP','QSO
V','QSRES','ARSOC','OQRAP','IMGCR','USENS','OMAN','SELFINJ','TOPIC','RIT
L','ACTVE','AGG','ANXTY'])
targetset.to_csv("/Users/sara/Desktop/AGRE/Autoencoder Module
4/training1/Module 4_y.csv")

#testing reconstruction
test = testset.data.cpu().numpy()
testset = pandas.DataFrame(test, columns=
['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CONVS','DGES
T','EGEST','UEYE','FACEO','LLNVC','SEI','CAFF','EMPTH','INSIG','RESP','QSO
```

```
V','QSRES','ARSOC','OQRAP','IMGCR','USENS','OMAN','SELFINJ','TOPIC','RIT
L','ACTVE','AGG','ANXTY'])
testset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
4/training1/Module 4_testset.csv")

#testing target
tests = testing.data.cpu().numpy()
testingset = pandas.DataFrame(tests,
columns=['OLANG','SPABN','IECHO','STEREO','OINFO','AINFO','REPRT','CON
VS','DGEST','EGEST','UEYE','FACEO','LLNVC','SEI','CAFF','EMPTH','INSIG','RE
SP','QSOV','QSRES','ARSOC','OQRAP','IMGCR','USENS','OMAN','SELFINJ','TO
PIC','RITL','ACTVE','AGG','ANXTY'])
testingset.to_csv("/home/sara/Desktop/AGRE/Autoencoder Module
4/training1/Module 4_testeing.csv")
```

REFERENCES

[1] N. Vijayakumar and M. Judy, "Autism spectrum disorders: Integration of the genome, transcriptome and the environment", *Journal of the Neurological Sciences*, vol. 364, pp. 167-176, 2016.

[2] "World Autism Awareness Day", Ministry of Health Portal Kingdom of Saudi Arabia, 2015. [Online]. Available: http://www.moh.gov.sa/en/HealthAwareness/HealthDay/2015/Pages/HealthDay-2015-04-02.aspx.

[3] B. Zablotsky, L. Black, M. Maenner, L. Schieve and S. Blumberg, "Estimated Prevalence of Autism and Other Developmental Disabilities Following Questionnaire Changes in the 2014 National Health Interview Survey", *National Health Statistics Reports*, 2015.

[4] D. Christensen, J. Baio, K. Braun, D. Bilder, J. Charles, J. Constantino, J. Daniels, M. Durkin, R. Fitzgerald, M. Kurzius-Spencer, L. Lee, S. Pettygrove, C. Robinson, E. Schulz, C. Wells, M. Wingate, W. Zahorodny and M. Yeargin-Allsopp, "Prevalence and Characteristics of Autism Spectrum Disorder Among Children Aged 8 Years — Autism and Developmental Disabilities Monitoring Network, 11 Sites, United States, 2012", *MMWR. Surveillance Summaries*, vol. 65, no. 3, pp. 1-23, 2016.

[5] M. Samms-Vaughan, M. Rahbar, A. Dickerson, K. Loveland, M. Hessabi, D. Pearson, J. Bressler, S. Shakespeare-Pellington, M. Grove, C. Coore-Desai, J. Reece and E. Boerwinkle, "The diagnosis of autism and autism spectrum

disorder in low- and middle-income countries: Experience from Jamaica", *Autism*, vol. 21, no. 5, pp. 564-572, 2017.

[6] N. Akshoomoff, C. Corsello and H. Schmidt, "The Role of the Autism Diagnostic Observation Schedule in the Assessment of Autism Spectrum Disorders in School and Community Settings", *The California School Psychologist*, vol. 11, no. 1, pp. 7-19, 2006.

[7] E. Braaten and G. Felopulos, *Straight talk about Psychology testing for kids*. New York: The Guilford Press, 2012, pp. 52-53.

[8] I. Goodfellow, Y. Bengio and A. Courville, *Deep learning*. MIT Press, 2016, pp. 164-223.

[9] *Maths Learning Service: Revision Matrices*. 2007.

[10] M. Labs, "Secret Sauce behind the beauty of Deep Learning: Beginners guide to Activation Functions", *Towards Data Science*, 2017. [Online]. Available: https://towardsdatascience.com/secret-sauce-behind-the-beauty-of-deep-learning-beginners-guide-to-activation-functions-a8e23a57d046.

[11]] I. Goodfellow, Y. Bengio and A. Courville, *Deep learning*. MIT Press, 2016, pp. 1-26.

[12] A. Walia, "Types of Optimization Algorithms used in Neural Networks and Ways to Optimize Gradient Descent", *Towards Data Science*, 2017. [Online]. Available: https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f.

[13] "First Order Optimization Methods", A Bit less Wrong, 2016.

[14] S. Ruder, "An overview of gradient descent optimization algorithms", 2016.

[15] Q. Yue and C. Ma, "Deep Learning for Hyperspectral Data Classification through Exponential Momentum Deep Convolution Neural Networks", *Journal of Sensors*, vol. 2016, pp. 1-8, 2016.

[16] C. Olah, "Calculus on Computational Graphs: Backpropagation", colah's blog, 2015.

[17] I. Goodfellow, Y. Bengio and A. Courville, *Deep learning*. MIT Press, 2016, pp. 29-50.

[18] I. Kamp-Becker, M. Ghahreman, M. Heinzel-Gutenbrunner, M. Peters, H. Remschmidt and K. Becker, "Evaluation of the revised algorithm of Autism Diagnostic Observation Schedule (ADOS) in the diagnostic investigation of high-functioning children and adolescents with autism spectrum disorders", *Autism*, vol. 17, no. 1, pp. 87-102, 2013.

[19] C. Lord, M. Rutter, P. DiLavore and S. Risi, Autism diagnostic observation schedule, second edition (ADOS-2). Los Angeles, CA: WPS, 2012.

[20] "Overview-Autism Genetic Resource Exchange", *Agre.autismspeaks.org*. [Online]. Available: http://agre.autismspeaks.org/site/c.lwLZKnN1LtH/b.5002149/k.E3CE/Overview.htm.

[21] "NumPy — NumPy", Numpy.org. [Online]. Available: http://www.numpy.org/.

[22] "Python Data Analysis Library — pandas: Python Data Analysis Library", *Pandas.pydata.org*. [Online]. Available: https://pandas.pydata.org.

[23] "scikit-learn: machine learning in Python — scikit-learn 0.19.1 documentation", *Scikit-learn.org*. [Online]. Available: http://scikit-learn.org/stable/.

[24] "Matplotlib: Python plotting — Matplotlib 2.1.2 documentation", *Matplotlib.org*. [Online]. Available: https://matplotlib.org.

[25] "The Jupyter Notebook — Jupyter Notebook 5.4.0 documentation", *Jupyter-notebook.readthedocs.io*. [Online]. Available: https://jupyter-notebook.readthedocs.io/en/stable/notebook.html.

[26] "PyTorch | About", *Pytorch.org*. [Online]. Available: http://pytorch.org/about/.

[27] "PyTorch documentation — PyTorch master documentation", *Pytorch.org*. [Online]. Available: http://pytorch.org/docs/master/index.html.

[28] "CUDA Zone", *NVIDIA Developer*. [Online]. Available: https://developer.nvidia.com/cuda-zone.

[29] I. Goodfellow, Y. Bengio and A. Courville, *Deep learning*. MIT Press, 2016, pp. 271-325.

[30] Chervinskii, *Schematic picture of an autoencoder architecture*. 2015.

[31] "Autoencoder", *Maciek*, 2018.

[32] O. Tadevosyan-leyfer, M. Dowd, R. Mankoski, B. Winklosky, S. Putnam, L. McGrath, H. Tager-flusberg and S. Folstein, "A Principal Components Analysis of the Autism Diagnostic Interview-Revised", *Journal of the American Academy of Child & Adolescent Psychiatry*, vol. 42, no. 7, pp. 864-872, 2003.

[33] "torch.nn — PyTorch master documentation", *Pytorch.org*, 2018. [Online]. Available: http://pytorch.org/docs/master/nn.html#l1loss.

[34] A. Sharma V, "Understanding Activation Functions in Neural Networks", Medium, 2017. [Online]. Available: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0.

[35] "Deep Learning with Keras", *O'Reilly | Safari*, 2018. [Online]. Available: https://www.safaribooksonline.com/library/view/deep-learning-with/9781787128422/8e232ec5-c2b8-48ee-b4ea-294d1e344533.xhtml.

[36] D. Clevert, T. Unterthiner and S. Hochreiter, *FAST AND ACCURATE DEEP NETWORK LEARNING BY EXPONENTIAL LINEAR UNITS (ELUS)*. Published as a conference paper at ICLR 2016, 2016.

[37] H. Patel, "SELU vs RELU activation in simple NLP models", *Hardik Patel*, 2017. [Online]. Available: https://www.hardikp.com/2017/07/24/SELU-vs-RELU/.

[38] Hao Zheng, Zhanlei Yang, Wenju Liu, Jizhong Liang and Yanpeng Li, "Improving deep neural networks using softplus units", *2015 International Joint Conference on Neural Networks (IJCNN)*, 2015.

[39] L. Bottou, *Stochastic Gradient Descent Tricks*. Microsoft Research, 2012.

[40] *elite data science, overfitting. 2017*

[41] Srivastava, Nitish, et al., *Dropout: a simple way to prevent neural networks from overfitting*. 2014.

[42] G. Roffo, "Ranking to Learn and Learning to Rank: On the Role of Ranking in Pattern Recognition Applications", PhD, University of Glasgow, 2018.

[43] K. Markham, "scikit-learn video #7: Optimizing your model with cross-validation", *The Official Blog of Kaggle.com*, 2018.

[44] "Challenging Behavior in Autism: Self-Injury | Interactive Autism Network", *Iancommunity.org*, 2018. [Online]. Available: https://iancommunity.org/aic/challenging-behavior-autism-self-injury.