8-2013

Mitigation of Catastrophic Interference in Neural Networks and Ensembles using a Fixed Expansion Layer

Robert Austin Coop
University of Tennessee - Knoxville, rcoop@utk.edu

To the Graduate Council:

I am submitting herewith a dissertation written by Robert Austin Coop entitled "Mitigation of Catastrophic Interference in Neural Networks and Ensembles using a Fixed Expansion Layer." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Engineering.

Itamar Arel, Major Professor

We have read this dissertation and recommend its acceptance:

Gregory Peterson, Jeremy Holleman, Joe Wilck

Accepted for the Council:
Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Mitigation of Catastrophic Interference in Neural Networks and Ensembles using a Fixed Expansion Layer

A Dissertation

Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Robert Austin Coop

August 2013

Dedication

So many have inspired me and kept me going throughout this long journey, and I will never be able to personally acknowledge and thank all of those without whom I would never have achieved so much or progressed so far. I will forever cherish all of my advisers, teachers, friends, family, and colleagues; I could never have done this without their enduring love and support, which has played a critical role in giving me the desire, conviction, and strength required to complete this journey.

Of all these positive influences, there is one person in particular that stands out as having played a critical role in my development; she provided a monumental influence through dedicating herself to her position, demonstrating genuine caring and insight, and generally performing at a level which was drastically above and beyond expectation, even when it would have been so easy (and socially acceptable) to do otherwise.

So, this dissertation is dedicated to Ms. Pamela Harrison, my physics teacher and mentor at Beech Senior High School in Sumner County, Tennessee. You nurtured my love for science, helped me to discover that problems can be solved in ways other than those found in textbooks, and ignited a passion for learning within me that has yet to be extinguished. You have done so much and inspired so many; this dedication is realistically only a small drop in the bucket of all the recognition and honors that you deserve, but you should know that your passion for teaching has given me the world. Thank you for all that you have done; your work has inspired and had an impact far greater than that for which you have been given credit.

Acknowledgments

Though only my name appears on the cover of this dissertation, a great many people have contributed to its production. I owe my gratitude to all those people who have made this dissertation possible and because of whom my graduate experience has been one that I will cherish forever.

First and foremost, I would like to thank my adviser Dr. Itamar Arel for introducing me to the world of machine learning, listening to my grandiose plans and schemes, and letting me aim for the stars (and in doing so learn of my own limitations). Your passion for the field inspired me to continue into graduate school, and I could not have done so without your advice and support. In a very real sense, I could not have done any of this if it was not for you.

I want to thank all of those who helped support me during graduate school. Thanks to Dr. Richard Bennett, Dr. Ortal Arel, and Professor Will Schleter for giving me the opportunity to gain valuable insight and teaching experience by teaching students within the Engineering Fundamentals program. I owe much to Dr. James Nutaro at the Oak Ridge National Laboratory for sponsoring my work there through the HERE and ASTRO programs; you gave me an opportunity to work within your world, to propose and investigate research topics, and to learn about the national laboratory environment while providing funding for my educational program at the same time. Finally, I would like to acknowledge the support that I have been given by those at the the RTSync Corporation; Dr. Doohwan Kim and Dr. Bernard Zeigler have been exceedingly kind in providing me employment which allows me to telecommute at the same time that I finish my academic studies.

Finally, I must be sure to acknowledge all those that have given me the personal support without which I never would have had the strength to continue. I must give thanks to my wife Amanda, who has always been a constant source of inspiration, support, and is still the hardest working person that I know; to my family, for being there to talk about all the work that needed to be completed and also to talk, when needed, about anything and everything except for that work. Also, I owe a debt of gratitude to my lab mates Derek Rose, Steven Young, Scott Livingston, Benjamin Martin, Nicole Pennington, and all of the others, for always being a sounding board, a colleague, or a distraction, depending on whatever was needed at the time.

Abstract

Catastrophic forgetting (also known in the literature as catastrophic interference) is the phenomenon by which learning systems exhibit a severe exponential loss of learned information when exposed to relatively small amounts of new training data. This loss of information is not caused by constraints due to the lack of resources available to the learning system, but rather is caused by representational overlap within the learning system and by side-effects of the training methods used. Catastrophic forgetting in auto-associative pattern recognition is a well-studied attribute of most parameterized supervised learning systems. A variation of this phenomenon, in the context of feedforward neural networks, arises when non-stationary inputs lead to loss of previously learned mappings. The majority of the schemes proposed in the literature for mitigating catastrophic forgetting are not data-driven, but rather rely on storage of prior representations of the learning system. We introduce the Fixed Expansion Layer (FEL) feedforward neural network that embeds an expansion layer which sparsely encodes the information contained within the hidden layer, in order to help mitigate forgetting of prior learned representations. The fixed expansion layer approach is generally applicable to feedforward neural networks, as demonstrated by the application of the FEL technique to a recurrent neural network algorithm built on top of a standard feedforward neural network. Additionally, we investigate a novel framework for training ensembles of FEL networks, based on exploiting an information-theoretic measure of diversity between FEL learners, to further control undesired plasticity. The proposed methodology is demonstrated on a several tasks, clearly emphasizing its advantages over existing techniques. The architecture proposed can be applied to address a range of computational intelligence tasks, including classification problems, regression problems and system control.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The problem of catastrophic interference (also referred to as catastrophic forgetting [20]) in artificial neural networks has been studied for over two decades by researchers from many disciplines, including machine learning, cognitive science, and psychology [36, 44]. Many real-world applications, such as financial time series analysis and climate prediction, involve data streams that are either strictly non-stationary or can only be considered piecewise stationary. It has been shown that in mammals, long durations of time between observations of stationary patterns can lead to an excessive tendency to form associations between sensory inputs and desired outputs (abnormal potentiation) often at the expense of weakening existing associations [32, 22]. In parameterized supervised learning systems (including connectionist architectures such as neural networks), catastrophic interference is the process by which a network 'forgets' learned patterns upon being presented with new patterns for a sufficiently long period of time. In such non-stationary settings, one can expect a neural network to have an inherent 'learning capacity' determined by the number of weights and neurons it contains. When this capacity is reached, learning new information will gradually interfere with a network's ability to recall prior representations.

However, catastrophic interference is commonly not a result of a network reaching its learning capacity. Instead, once the network has been trained on new patterns, or is no longer being adequately presented with inputs drawn from a prior observation distribution, drastic information loss occurs at exponential rates. New information catastrophically degrades learned representations even though sufficient learning capacity remains. Such scenarios are commonly encountered when non-stationary input streams are presented to the network or in circumstances where continuous learning is desired.

Many approaches to diminishing the impact of catastrophic interference have been proposed in the literature, with varying levels of success [3, 4, 18, 26, 27, 33, 45]. The vast majority of the schemes proposed do not pertain to online learning, but are rather based on batch-learning processes (with a particular emphasis on the study of auto-associative pattern recognition). Moreover, most techniques require extensive memory resources, as they continually store representations of prior network configurations as means of refreshing knowledge of older representations. This work introduces a novel approach for mitigating catastrophic forgetting by augmenting multilayer perceptron (MLP) networks with an additional sparsely-encoded hidden layer specifically designed to retain prior learned mapping of inputs to outputs. Learning occurs incrementally and minimal storage requirements are imposed. Furthermore, this work examines other methods for the mitigation of catastrophic interference in neural network systems (including the introduction of the target variation trace, a method which controls the learning of new information based on the novelty of the network error) as well as methods that can control the effects of catastrophic interference in ensembles of neural networks.

This manuscript is divided into six chapters. Chapter 1, this chapter, introduces the problem, and Chapter 2 provides a detailed background of the concepts and examines other approaches found in the literature. The Fixed Expansion Layer network is introduced in Chapter 3, which examines the problem of catastrophic interference within a single learning system. Chapter 4 examines catastrophic interference within ensembles of learning systems. Experimental results are presented in Chapter 5, with a summary of contributions and discussion of future work presented in Chapter 6.

Chapter 2

Background and Literature Review

2.1 Artificial Neural Networks

Artificial neural networks (ANNs) are composed of individual units, analogous to neurons. Each unit processes input and produces output independently of other neurons in the network, with the network's output being determined by the output values of its individual neurons. The first significant implementation of an ANN was the 'perceptron' [47].

The Perceptron

The perceptron is the most fundamental ANN algorithm. Each neuron is connected to network inputs, outputs, and other neurons via 'synapses'. Synapses have an associated weight value, and the output of a neuron is some function of the weighted sum of its input values. Formally, each neuron performs the following calculation

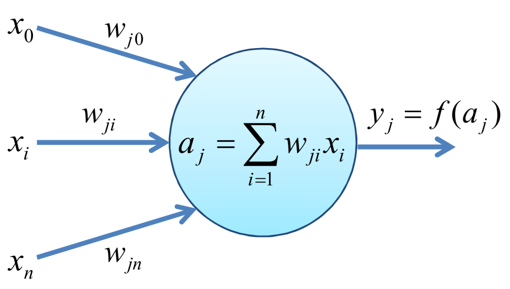


Figure 2.1: A Single Perceptron Unit

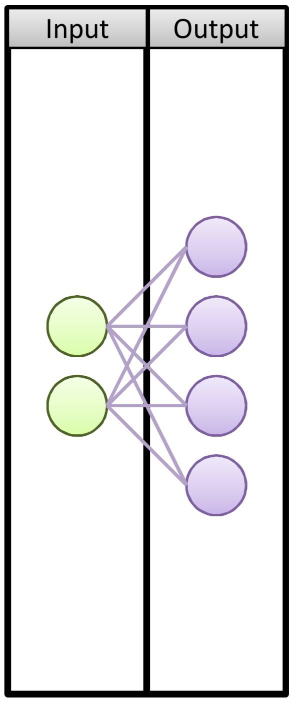


Figure 2.2: Perceptron Network

$$\hat{y}_j = f\left(\sum_{i=1}^{n} w_{ji} x_i\right) \tag{2.1}$$

where $\hat{y}_j$ is the output of the j-th neuron (the network's approximation of the desired output, which is represented as y_j), w_{ji} is the weight that neuron j assigns to the i-th synapse, x_i is the value of the i-th input (the value transmitted along the i-th input synapse), and $f(\cdot)$ is the activation function (which can be the identity function if the output is just the value of the weighted sum or a threshold function). A perceptron neuron is depicted in Figure 2.1.

A perceptron network uses two layers of neurons. The neurons in the first layer are fully connected to the second layer by a number of synapses (as shown in Figure 2.2). Inputs are fed to the first layer, signal is transmitted to the second layer, and the network's output is composed of the values of the neurons in the second layer. During training, the network's output is compared with the desired output. The difference between the two is the network error and is used to train the network. Learning in these networks takes place in the synaptic junctions between the two layers; by adjusting the synaptic weights (the values used for w_{ji}), the network is able to learn mappings between inputs to the network and the desired outputs.

Perceptron networks perform *supervised learning*, where training data consists of pairs of network inputs and also the desired output for each input. The output cells of the network compare the desired output with the actual output in order to calculate an error signal used for training. Training is performed using the Widrow-Hoff rule [53]. Intuitively, if an output cell's value is too high, then it makes sense to decrease the synaptic weight values for synapses connected to input cells with positive values (and increase the weight for synapses connected to input cells with negative values). Similarly, if the output value is too low, then the synapses connected to input cells with positive values should be increased. Formally, the weight update rule is calculated as

$$w_{ji}^{(t+1)} = w_{ji}^{(t)} + \eta(y_j - \hat{y}_j)x_i \tag{2.2}$$

where $w_{ji}^{(t+1)}$ is the weight assigned to the synapse between the j-th and the i-th input at time $t+1$ (with $w_{ji}^{(t)}$ representing this value at time t), y_j is the desired target value of the j-th output, $\hat{y}_j$ is the actual value of the j-th network output, and x_i is the value of the i-th input. The value of η is a small constant which acts as a learning step-size; the step-size controls the rate of weight changes within the network.

This early neural network is able to learn associations between inputs and outputs, but suffers from a key drawback: the output must be a linear transformation of the input. Essentially, the perceptron network's operation is equivalent to linear regression and to discriminant analysis [37, 48].

The Multilayer Perceptron

The multilayer perceptron (also known as the feedforward neural network or the error backpropagation network) solves the problem of learning associations which are not linear transformations. This is accomplished by adding a third 'hidden' layer between the input and output layers (as depicted in Figure 2.3). Formally, the network's operation uses the following equations

$$h_j = f\left(\sum_i w_{ji}x_i\right) \tag{2.3}$$

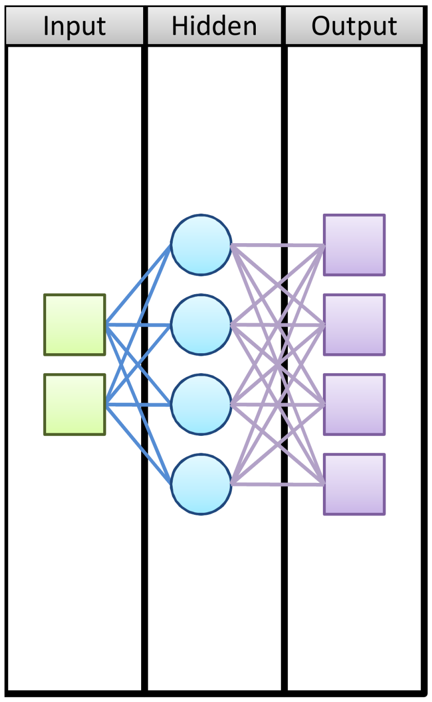


Figure 2.3: Multilayer Perceptron

$$\hat{y}_k = g\left(\sum_j z_{kj} h_j\right) \tag{2.4}$$

where h_j is the value of the j-th hidden neuron, $f(\cdot)$ is the activation function for the hidden layer, w_{ji} is the weight that hidden neuron j assigns to the i-th input, x_i is the value of the i-th input, $\hat{y}_k$ is the value of the k-th output neuron, $g(\cdot)$ is the activation function used for the output layer, and z_{kj} is the weight that the k-th output neuron assigns to the j-th hidden neuron.

By using a nonlinear activation function $f(\cdot)$ for the hidden layer, the network is able to learn nonlinear input-output mappings, however the training algorithm must be changed to account for the new layer. Multilayer perceptrons can be trained using gradient descent (also known as error backpropagation), which is explained in detail in Section 2.3.

2.2 Recurrent Neural Networks

The standard multilayer perceptron neural network does not have the ability to approximate temporally-dependent mappings between its input space and its output space. To address

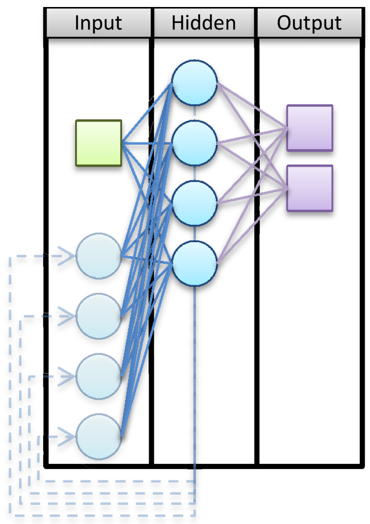


Figure 2.4: Elman Recurrent Network (Simple Recurrent Network)

this, Elman's Simple Recurrent Network (SRN) [15] utilizes 'context' neurons which form a representation of the current state. Following each feedforward process, the values of the hidden neurons are stored into the context neurons (as depicted in Figure 2.4). These context neurons are fed to the network, in addition to the external inputs, during the subsequent time step.

Formally, consider a multilayer perceptron which has N input neurons $[x_1, x_2, \ldots, x_N]$ and M hidden neurons $[h_1, h_2, \ldots, h_M]$. To transform this MLP into a recurrent network, context neurons are added to the input layer. The M new context neurons $[x'_1, x'_2, \ldots, x'_M]$ will store the prior value of the hidden layer neurons. Equation 2.3 shows the hidden layer calculation for a MLP; the new hidden layer calculation for this recurrent network becomes

$$h_j^{(t)} = f\left(\sum_{i=1}^{N} w_{ji} x_i + \sum_{i=1}^{M} w'_{ji} x'_i\right) \tag{2.5}$$

where $h_j^{(t)}$ is the value of the j-th hidden neuron at time t and w'_{ji} is the weight that the j-th hidden neuron assigns to the i-th context neuron. The values of the context neurons x'_i are given by

$$x'_j = h_j^{(t-1)} \qquad j \in [1, M] \tag{2.6}$$

where $h_j^{(t-1)}$ is the value of the j-th hidden neuron from the previous input at time $t - 1$.

Using these memory signals the network is able to capture temporal dependencies. However, retaining information about long-term temporal dependencies is a difficult task; SRNs, and most gradient-based recurrent neural network models, have difficulty with capturing long sequences, or functions with long-term temporal dependencies, due to the ‘vanishing gradient’ phenomenon [6]. Additionally, the non-stationary nature of real-world data sequences typically fed to recurrent networks renders the task of accurately modeling processes difficult.

2.3 Gradient-based Network Training

Training for the most simple neural network, the perceptron, is accomplished using the Widrow-Hoff rule defined in Equation 2.2, but this equation is not suitable for adjusting the weights of multiple layers simultaneously. For MLPs, RNNs, and other multilayer networks, error backpropagation is often used to perform weight updates during supervised learning. Error backpropagation (or variants of the method) is the most popular training algorithm for networks with at least three layers and has been discovered and independently rediscovered by many authors (e.g. [11, 29, 40, 49, 52]).

The intuition behind error backpropagation is fairly simple. Under supervised training, the weights of linear neurons in the output layer of a network can be adjusted according to the standard Widrow-Hoff learning rule; the error from the output layer is then propagated backward from the output layer to the hidden layer using those same connections and weights. An error signal for the hidden layer neurons can be calculated as a weighted average of the error signal in the output layer neurons (with the weighting determined by the weights between the output and hidden neurons). Once an error signal can be determined for the neurons in the hidden layer, the Widrow-Hoff rule can be applied to adjust the weights between the hidden and input layer neurons.

Consider a multilayer network with activations given by

$$h_j = f\left(\sum_i w_{ji} x_i\right) \tag{2.7}$$

$$\hat{y}_k = g\left(\sum_j z_{kj} h_j\right) \tag{2.8}$$

where h_j is the value of the j-th hidden neuron, $f(\cdot)$ is the activation function for the hidden layer, w_{ji} is the weight that hidden neuron j assigns to the i-th input, x_i is the value of the i-th input, $\hat{y}_k$ is the value of the k-th output neuron, $g(\cdot)$ is the potentially non-linear activation function used for the output layer, and z_{kj} is the weight that the k-th output neuron assigns to the j-th hidden neuron.

The total error for the k-th output neuron is calculated as

$$e_k = (y_k - \hat{y}_k) \tag{2.9}$$

where y_k is the k-th target output, and we define an error signal for this neuron using the equation

$$\delta_{output,k} = g'\left(\sum_j z_{kj} h_j\right) \times e_k \tag{2.10}$$

where $g'(\cdot)$ is the derivative of the activation function for the output layer.

This error signal combines the neuron error with the activation of the neuron. Using this error signal in conjunction with the Widrow-Hoff rule, we obtain the delta update rule for the output layer weights:

$$z_{kj}^{(t+1)} = z_{kj}^{(t)} + \eta \delta_{output,k} h_j \tag{2.11}$$

where $z_{kj}^{(t+1)}$ is the value of the weight that the k-th output neuron assigns to the j-th hidden neuron at time $t+1$ (with $z_{kj}^{(t)}$ representing this value at time t) and η is a step-size constant.

The error signal for the hidden layer cannot be directly computed, due to the fact that there is no defined 'target' value defined for this layer. The error signal is estimated as a function of the output error, the weights between the output and hidden layers, and the activation of the hidden layer neurons. For the j-th hidden neuron, this estimation is given by

$$\delta_{hidden,j} = f'\left(\sum_i w_{ji} x_i\right) \times \sum_k \left(z_{kj} \delta_{output,k}\right) \tag{2.12}$$

where $f'(\cdot)$ is the derivative of the activation function for the hidden layer. Given the estimated error signal, we apply the Widrow-Hoff rule to obtain the update equation

$$w_{ji}^{(t+1)} = w_{ji}^{(t)} + \eta \delta_{hidden,j} x_i \tag{2.13}$$

where $w_{ji}^{(t+1)}$ is the value of the weight that the j-th hidden neuron assigns to the i-th input at time $t+1$ (with $w_{ji}^{(t)}$ representing this value at time t).

Using the iterative update rules given by Equations 2.11 and 2.13, all of the weights of the network can be adjusted in response to each training example presented to the network. Error backpropagation has been shown to converge toward a local minimum of the mean square of the error for the output layer (provided that η is chosen to be appropriately small) [1]. Precisely, if the total error of the network in response to the t-th training sample is defined as

$$E^{(t)} = \frac{1}{2}\sum_{k}(y_k - \hat{y}_k)^2 \tag{2.14}$$

then the given update rules will cause the network weights to change such that $E^{(t)}$ will converge to a local minimum as $t \to \infty$.

This convergence property of error backpropagation can be proven by showing that error backpropagation is an implementation of gradient descent, a well-known procedure in numerical analysis [12, 17, 43, 50]. Equations 2.11 and 2.13 are equivalent to

$$w_{ji}^{(t+1)} = w_{ji}^{(t)} - \eta\frac{\partial E^{(t)}}{\partial w_{ji}} \tag{2.15}$$

and

$$z_{kj}^{(t+1)} = z_{kj}^{(t)} - \eta\frac{\partial E^{(t)}}{\partial z_{kj}} \tag{2.16}$$

where $\frac{\partial E^{(t)}}{\partial w_{ji}}$ is the partial derivative of the error at time t with respect to the weight w_{ji} and $\frac{\partial E^{(t)}}{\partial z_{kj}}$ is the partial derivative of the error at time t with respect to the weight z_{kj}. A full proof of this equivalence can be found in [1].

When considering catastrophic interference, two features of error backpropagation are very significant. Firstly, network weights are updated at each time step in response to each training sample; this means that the influence that any one training sample has on the network decreases as the total number of training samples grows. Secondly, error backpropagation is only guaranteed to converge to a *local* minimum of the error function; there is no guarantee that the global minimum will be located or that a small adjustment in the network weights will not result in a large increase in network error.

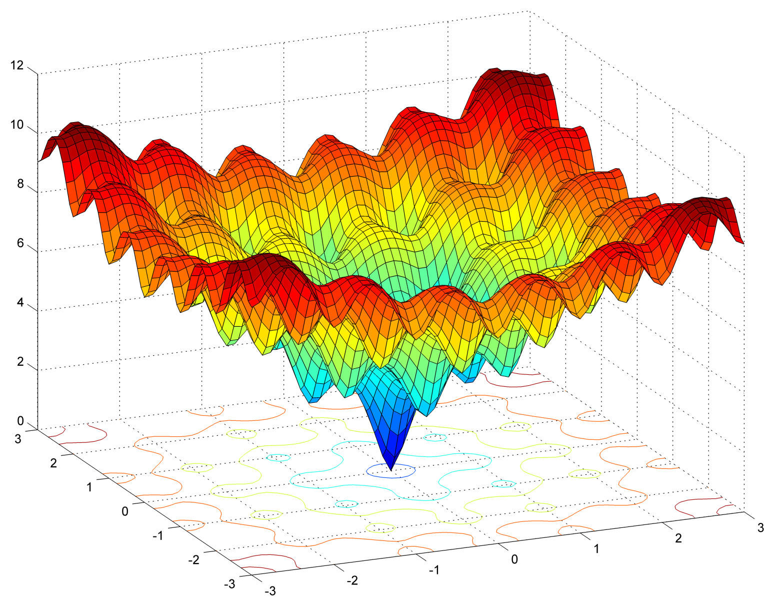


Figure 2.5: The Ackley Problem

To illustrate the significance of these drawbacks, consider a classic minimization problem known as the Ackley problem (introduced as a two-dimensional problem in [2], and generalized to n dimensions in [5]). The Ackley problem defines the multidimensional function

$$F(\vec{x}) = -20 \cdot \exp\left(-0.2\sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n} \cdot \sum_{i=1}^{n} \cos(2\pi x_i)\right) + 20 + \exp(1) \tag{2.17}$$

which is shown in Figure 2.5 (for a two-dimensional $\vec{x}$).

If the surface of the error function for a neural network is similar to that of the Ackley problem, then the error backpropagation is likely to converge to a local minima and not to the global minima. Additionally, if the network is able to converge to the global minima, it is not likely to stay there; the continuous weight updates will ‘bump’ the network out of the global minima and cause the network to converge back to a local minima.

2.4 Online Learning and Non-stationary Environments

Machine learning systems can either be trained using unsupervised learning (wherein training takes place without any *a priori* definitions of 'right' or 'wrong') or supervised learning (which trains the system using training data which has been previously annotated with correct answers). Additionally, both of these types of systems can be trained in two different ways, depending on how many presentations of training data are allowed.

When performing *batch training*, the training data is collected before the learning system is to be trained. The training data is then presented to the learning system until a desired degree of proficiency is demonstrated. The entire collection of training data is collected beforehand and stored; this allows training to perform multiple passes through the training data as needed. Batch training is typically more effective, as the learning system benefits from being able to examine all available training data during the learning process. Typically, learning systems which use batch training are only trained a single time. Afterwords, the systems are put into a sort of 'production mode', and learning no longer takes place.

In contrast to batch training, *online learning* methodologies are those which do not allow repeated presentations of training data. Each sample from the set of training data is presented a single time, and there can be no reuse of data once all training data has been observed. The advantage of this methodology is that no persistent storage is required for the training data; the system is given each sample of training data as it becomes available, and then the sample is discarded. This has the disadvantage of not allowing the system to repeatedly train over the same dataset.

The process of online learning is further complicated when it occurs within non-stationary environments (also referred to as 'concept drift'). The environment is said to be non-stationary when the process by which inputs are generated changes over time.

2.5 Catastrophic Forgetting in Neural Networks

The problem of catastrophic interference (also referred to as catastrophic forgetting) in neural networks has been studied for over two decades by researchers from many disciplines such as machine learning, cognitive science, and psychology [36, 44]. In neural networks (and other connectionist architectures), catastrophic interference is the process by which a network 'forgets' learned patterns upon being presented with new patterns. One can expect a neural network to have an inherent 'learning capacity' determined by the number of weights and neurons it contains; when this capacity is reached, learning new information will gradually interfere with a network's ability to recall existing information.

However, catastrophic interference is not caused by a network having reached its learning capacity. Instead, once the network has been trained on new patterns, or is no longer being adequately presented with inputs drawn from its entire observation space, drastic information loss occurs; new information catastrophically interferes with the previously learned model even though the theoretical learning capacity has not been met. Such scenarios are commonly encountered when non-stationary inputs are presented to the network.

2.6 Conventional Mitigation of Catastrophic Forgetting

Many approaches to reducing the effect of catastrophic interference have been proposed with varying levels of success. Notably, mainstream exploration of the problem of catastrophic interference has been within the domain of auto-associative pattern learning, which has not addressed problems inherent with more general function approximation [38]. The most common schemes proposed in the literature can coarsely be categorized as rehearsal methods, psuedorehearsal methods, dual methods, and activation sharpening.

2.6.1 Rehearsal Methods

Rehearsal methods were among the first approaches aimed at addressing the problem of catastrophic interference; two such methods are the rehearsal buffer model [44] and sweep rehearsal [45]. Each method attempts to retain information about previously learned patterns by maintaining a buffer of previously observed patterns; these buffered patterns are then periodically used for training during the learning of subsequent patterns. Such early methods mitigated the effect of catastrophic interference somewhat, but required persistent storage of learned patterns and introduced challenges with respect to the correct balance between new patterns and buffered ones.

The Rehearsal Buffer Model

The rehearsal buffer model operates as follows: assuming there are M patterns to be learned, one creates a rehearsal buffer containing a small subset of m patterns (e.g. to start, the 1st pattern through the m^{th} pattern, with typical values of m being around 4). The neural network is then trained over each pattern in the buffer; each pattern in the buffer is presented in sequence, and the entire buffer is looped over N times. After N loops through the buffer, the first item in the buffer is removed and the next pattern to be learned is added to the buffer. So, the buffer would first contain items in the range $[1, m]$, training loops over these items N times, and the buffer is updated to contain the items in the range $[2, m + 1]$. This process continues until all M patterns have been learned.

Sweep Rehearsal

Sweep rehearsal is similar to the rehearsal buffer model, but uses a “dynamic” training buffer. Given M patterns and a buffer size of m, one must first learn at least $(m - 1)$ patterns before the dynamic buffer comes into play. Once some number of patterns has been learned (with a pattern considered ‘learned’ by being trained until the estimation error is below a given criterion), the process of learning a new pattern proceeds as follows: let the new pattern to be learned be pattern x_i. The training buffer is created by combining pattern x_i with $(m-1)$ previously learned patterns (selected at random), and the network is trained by presenting each pattern in the buffer once. However, where the rehearsal buffer model would sweep through this same buffer N times, sweep rehearsal builds a new buffer containing x_i and $(m - 1)$ randomly selected previously learned patterns for each epoch (where an epoch is one presentation of each pattern in the buffer). A new buffer is created and used for training until pattern x_i is considered learned (at which point we repeat the process for x_{i+1}, etc.). In practice, sweep rehearsal shows better performance than the rehearsal buffer model [45].

2.6.2 Pseudorehearsal Methods

Whereas rehearsal methods attempt to retain learned information by storing and rehearsing previously learned patterns, pseudorehearsal methods attempt to latch on to learned information without the requirement of pattern storage [46]. Instead of using previously learned patterns for rehearsal, pseudopatterns consisting of random input values are generated periodically during training. The pseudopattern is fed into the network and the network's output is recorded. After some number of subsequent training iterations, a previously generated pseudopattern is selected for pseudorehearsal. The pseudopattern is fed into the network, and the previously recorded output is used as a training target.

Pseudorehearsal can use the rehearsal buffer model or sweep rehearsal as the base learning mechanism. However, random pseudopatterns are used instead of actual patterns that have previously been learned. To use pseudorehearsal with the sweep rehearsal approach, we would construct our sweep buffer by selecting the n-th pattern to be learned and generating $(m-1)$ pseudopatterns (as opposed to randomly selecting $(m-1)$ previously learned patterns) before each epoch. During each epoch, the network is trained over the sweep buffer once. We then construct another sweep buffer containing the n-th pattern and newly generated pseudopatterns. This process repeats until the n-th pattern is learned sufficiently (then we would repeat using the $(n+1)$-th pattern, etc.).

Consider the problem of learning binary patterns with an auto-associative neural network. The training data is given as input-output pairs (x_i, y_i), where x_i is the i-th input and y_i is the desired i-th output.The function to be approximated is $f(x_i) = y_i$. Note that, for the auto-associative problem, $x_i = y_i$ and $f(\cdot)$ is the identity function. The function approximation performed by the auto-associative neural network $h(x_i) = \hat{y}_i$, where $\hat{y}_i$ is the output of the neural network (and we desire $h(x_i) = \hat{y}_i \approx y_i$).

We generate a pseudopattern p_i by randomly setting each bit to 0 or 1 with equal probability. We compute the output of the network over this pseudopattern and store the result as q_i (i.e. we compute $h(p_i) = q_i$). For pseudopatterns, instead of training in order to achieve the goal $q_i \approx p_i$, we use the value of q_i as the actual training target. So, our sweep buffer would contain our pattern to be learned, (x_i, y_i) as well as a number of pseudopatterns (p_i, q_i).

These pseudopatterns serve as approximate snapshots of the network's internal state at some time during the training process. As training proceeds, the network's internal state is being adjusted in order to recognize the currently viewed patterns. When pseudorehearsal is performed, the network's internal state is essentially being re-adjusted in order to be more like the snapshot of its prior internal state. This adjustment causes the network to be more likely to retain prior information, thus combating the catastrophic interference effects. However, the process of generating pseudopatterns and periodically retraining over these pseudopatterns increases the storage and computational requirements of the system. Moreover, analysis suggests that in some networks the effectiveness of pseudorehearsal is reduced when used with low-dimensional input or input patterns that are nearly (or completely) orthogonal [18].

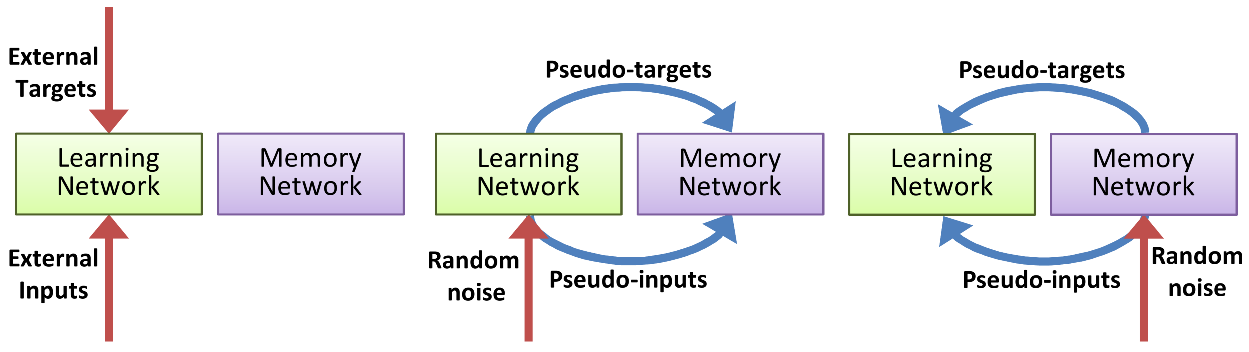
2.6.3 Dual Methods

Dual methods address catastrophic interference by means of attempting to separate that which is being learned from that which has already been learned; these methods are characterized by the explicit representation of short-term and long-term memory. Dual weight methods [27, 33] maintain two sets of weights for a single network architecture, while dual network methods [21, 4, 26] utilize entirely separate neural networks. Both approaches use one of their resources (a set of weights or a network) for storing long-term, slowly changing information and use the other resource for storing short-term, quickly changing information.

Dual network methods operate by using one network (the 'learning' network) for processing input and learning the input-output function to be approximated, while another network (the 'memory' network) performs long-term storage of learned information. During training, the learning network alternates between periods of normal training and periods of knowledge transfer. For the normal training, the learning network is fed training information and trained in the same way as any multilayer perceptron feedforward neural network would be used [3].

Knowledge transfer is performed using pseudopatterns [3]. A pseudopattern is generated for one of the networks (the 'source' network) by creating pseudo-inputs consisting of random noise and feeding these into the source network; the source network's output is recorded as the pseudo-targets. The same pseudo-inputs are presented to the other network (the 'destination' network), and the pseudo-targets from the source network are used as a training target for the destination network.

During training, knowledge is transferred from each network to the other. Knowledge is transferred from the learning network to the memory network in order to store information about



(a) Learning new information **(b)** Storing of new information **(c)** Refreshing learned information

Figure 2.6: Operation of a dual network system

newly learned patterns, while knowledge is transferred from the memory network to the learning network in order to retain information about previously learned patterns [3]. Operation of a dual network system is depicted in Figure 2.6.

2.6.4 Activation Sharpening

Activation sharpening is inspired by the belief that catastrophic forgetting is a consequence of the overlap of pattern representations within the neural network and can be addressed by reducing such overlap [19]. The goal of activation sharpening is to gradually develop semi-distributed representations of patterns in the hidden layer of the network by causing neurons within the hidden layer to 'latch' onto specific regions of the input space. This approach modifies this traditional feedforward process; the input pattern is fed forward and the activation of nodes in the hidden layer is 'sharpened' by increasing one or more of the hidden nodes with the largest activation values and decreasing the activation values of all the other hidden nodes. The difference between the original and sharpened activation values is immediately backpropagated to the input-hidden weights (as if it was an error signal) in order to train the network to produce a sharpened activation in the future. After this occurs, the input is fed forward and the error backpropagated as usual.

More formally, consider a network with the hidden layer values given by

$$h_j = f\left(\sum_i w_{ji}x_i\right) \tag{2.18}$$

where h_j is the value of the j-th hidden neuron, $f(\cdot)$ is the activation function for the hidden layer, w_{ji} is the weight that hidden neuron j assigns to the i-th input and x_i is the value of the i-th input.

A typical neural network performs one feedforward and backpropagation operation for each training sample. A typical feedforward iteration for the network would be performed by setting the input layer values, calculating the hidden layer values, then calculating the output values. After the feedforward, error backpropagation would take place (as described in Section 2.3).

A network with a single hidden layer using activation sharpening performs one and a half feedforward and backpropagation operations for each training sample. The first set of operations are performed in order to calculate the hidden layer activations and perform activation sharpening. This feedforward operation is performed by setting the input layer values and then calculating the hidden layer values; output layer values are not yet calculated.

Then, some number of the most active hidden layer neurons are selected for sharpening. Assuming that activation values for the hidden layer neurons are between 0 and 1, new activation values for all hidden neurons are calculated using the formula

$$\begin{cases} h'_j = h_j + \alpha(1 - h_j) & \text{if } h_j \in S \\ h'_j = h_j - \alpha h_j & \text{if } h_j \notin S \end{cases} \tag{2.19}$$

where h'_j is the new activation value, α is a sharpening factor between 0 and 1, and S is the set of neurons selected for activation.

After the new activation values have been calculated, a backpropagation operation is performed. The difference between the old activation values and the new activation values are treated as an error signal, and the hidden neuron weights are updated according to the equation

$$w'_{ji} = w_{ji} + \eta(h'_j - h_j)x_i \tag{2.20}$$

where w_{ji} is the value of the weight that the j-th hidden neuron assigns to the i-th input, w'_{ji} is the updated weight, x_i is the value of the i-th input, and η is the network step size.

After this ‘half’ feedforward and backpropagation are completed, a standard feedforward and backpropagation is performed (using the new weight values). The goal of this procedure is to evolve representations with a few highly activated hidden neurons instead of many active neurons with average activation levels; this has the effect of reducing the average amount of activation overlap among representations[19].

2.7 Ensemble Diversification Strategies

Ensembles of neural networks have been shown to be more effective than individual neural networks. Originally, neural network ensembles were constructed using a two-stage design process; individual networks are first generated and trained, and then they are combined into ensembles. These networks were usually trained independently.

The mean squared error (MSE) of an ensemble of neural networks can be decomposed into bias, variance, and covariance terms [51]. Ensembles aim to reduce the variance term; the variance of the ensemble will be lower than the average variance of its components. This can be shown by considering the output of an ensemble $\mathcal{H}$ in response to input x_i given by a weighted linear combination of the ensemble member outputs as

$$\mathcal{H}(x_i) = \sum_{j=1}^{N} w_j h_j(x_i) \tag{2.21}$$

where w_j is the weight given to ensemble member j such that $0 \leq w_j \leq 1$ and $\sum_j w_j = 1$, and $h_j(x_i)$ is the output of ensemble member j. Using the property that, for uncorrelated random variables X and Y, Var$[\alpha X + \beta Y] = \alpha^2Var[X] + \beta^2Var[Y]$ we can show that the variance of $\mathcal{H}$ is given by

$$\text{Var}[\mathcal{H}] = \text{Var}\left[\sum_{j=1}^{N} w_j h_j\right] = \sum_{j=1}^{N} \left(w_j^2 \text{Var}[h_j]\right) \tag{2.22}$$

Since $w_j \leq 1$, we can see that Var$[\mathcal{H}] < \sum_{j=1}^{N} (\text{Var}[h_j])$ (under the assumption that there is more than one network in the ensemble, i.e. $N > 1$). However, this is under the assumption that the members of $\mathcal{H}$ are uncorrelated. If there is any correlation between members, then we have to add a covariance term to the calculation of Var$[\mathcal{H}]$; correlation among members of an ensemble directly reduce the maximum accuracy that the ensemble can achieve, and any method that can increase diversity of the members in an ensemble has the potential to increase the ensemble's accuracy.

In the case that ensemble members are neural networks, the function calculated by h_j is fully specified by the weights contained in the network. In a sense, we can consider h_j to be occupying a point in a hypothesis space, with the coordinates being specified by the network weights. During the training process, h_j follows a trajectory through hypothesis space; the initialization of the network weights determine the starting point and each training iteration moves h_j along a trajectory. The ensemble $\mathcal{H}$ can therefore be defined as a collection of points within this hypothesis space. If $\mathcal{H}$ is to have diversity among its members, we would like ensemble members to occupy sufficiently different points within this hypothesis space. Under this world view, we can categorize diversification strategies into three categories depending on how the strategy alters the hypothesis trajectory of an ensemble member. Diversification strategies can alter the *starting point in hypothesis space*, the *set of accessible hypotheses*, or the *trajectory through hypothesis space*. [10]

2.7.1 Starting Point in Hypothesis Space

Random Weight Initialization

Initializing the weights of each neural network randomly is one of the most common methods for creating diversity; the hope is that starting each network in a different location within hypothesis space will increase the probability that the networks will continue to follow unique trajectories within the space. This is generally accepted as the least effective diversification strategy, and is typically only used as a benchmark for comparison (e.g. [39]). It has been empirically shown that, after network type, training set structure, and number of hidden neurons, randomized weight initialization is the least effective means of generating diversity. [54, 41]

2.7.2 Set of Accessible Hypotheses

The most common way of altering the set of hypotheses accessible by an ensemble member is to alter the set of training data by presenting a different subset of the training data to each ensemble member.

***K*-fold Cross-Validation**

This strategy separates the training samples into k disjoint subsets. k new partially overlapping training sets are then created for each ensemble member by leaving one of the k subsets out and training over the others. [28]

Bagging

Bagging re-samples the original training set to randomly create training sets for each ensemble member. Given a set of N training samples, each ensemble member is trained using a subset of the training samples generated by randomly drawing (with replacement) N samples from the training data. Each member is still trained using N samples, but some samples are repeated and some omitted from each member's specific training set. [7]

Boosting

This strategy is similar to bagging, however training sets are explicitly created. Ensemble members are trained in series; when selecting the training set for the j-th ensemble member, weighted random sampling is used to draw the samples from the training set. The weighting is determined from the performance data obtained when training the previous ensemble members, and specified such that samples that were difficult for existing ensemble members to classify are more likely to be presented to the new member. [23] The popular AdaBoost algorithm [24] falls under this category.

2.7.3 Trajectory Through Hypothesis Space

These diversification strategies alter the way that ensemble members traverse the hypothesis space. The most popular method of altering the hypothesis space trajectory is modifying the cost function by adding a penalty term. The normal error function for an ensemble member is

$$e_j(x_t) = \frac{1}{2}(y_t - \hat{y}_t^{(j)})^2 \tag{2.23}$$

When we add a penalty term, this becomes

$$e_j(x_t) = \frac{1}{2}(y_t - \hat{y}_t^{(j)})^2 + \lambda R \tag{2.24}$$

where R is the penalty term, and λ is a weighting factor that controls the tradeoff between the normal error and the error due to incurred penalty.

Negative Correlation (NC) Learning

Negative correlation (NC) learning applies a penalty term in order to negatively correlate each network's error with the errors of the rest of the ensemble [35]. It relies on the intuition that diversity between neural networks can be enforced by augmenting their cost function with an explicit diversity term. NCL employs the standard back-propagation algorithm to train the individual neural networks in parallel, and then each network's error function is penalized in order to negatively correlate the errors of the learners as means of guaranteeing diversity.

Consider a training dataset D, defined as

$$D = \{(x_1, y_1), \ldots, (x_t, y_t), \ldots, (x_N, y_N)\} \tag{2.25}$$

where $x_t \in \mathbb{R}^{D_X}$ is the t-th training input with D_X dimensions, $y_t \in \mathbb{R}^{D_Y}$ is the t-th desired output with D_Y dimensions, and the dataset has a total of N samples.

We take the output of the ensemble to be the simple weighted average of it's members, defining

$$\mathcal{H}(x_t) = \frac{1}{M}\sum_{j=1}^{M}(h_j(x_t)) = \hat{y}_t \tag{2.26}$$

where we use $\mathcal{H}(\cdot)$ to represent the function computed by the entire ensemble and $\hat{y}_t$ to represent the output of the ensemble in response to the input x_i (i.e. $\mathcal{H}(x_t) = \hat{y}_t$). When referring to individual members of an ensemble, $h_j(\cdot)$ represents the function computed by the j-th ensemble member, $h_j(x_t)$ is the j-th ensemble member's output in response to the t-th input, and $\hat{y}_t^{(j)}$ is the member's output (i.e. $h_j(x_t) = \hat{y}_t^{(j)}$).

So, an alternative expression of the ensemble calculations can be given by

$$h_j(x_t) = \hat{y}_t^{(j)} \tag{2.27}$$

$$\mathcal{H}(x_t) = \frac{1}{M}\sum_{j=1}^{M} \hat{y}_t^{(j)} \tag{2.28}$$

Normally, if each member was being trained in order to minimize the standard mean squared error equation, the error for member j would be given by

$$e_j(x_t) = \frac{1}{2}(y_t - \hat{y}_t^{(j)})^2 \tag{2.29}$$

However, under NCL, we add a penalty term to obtain the error function

$$e_j(x_t) = \frac{1}{2}(y_t - \hat{y}_t^{(j)})^2 + \lambda p_j(x_t) \tag{2.30}$$

where $p_j(x_t)$ is the penalty for the j-th ensemble member in response to input x_t and λ is a weighting term balancing the normal mean squared error term with the new penalty term.

The penalty term for each network is given by

$$p_j(x_i) = (h_j(x_i) - \mathcal{H}(x_i)) \sum_{k \neq j} (h_k(x_i) - \mathcal{H}(x_i)) \tag{2.31}$$

which, as shown in the literature (e.g. [9, 35]), causes the errors between ensemble members to become negatively correlated. NC has been shown to consistently outperform simple ensemble systems. In addition to empirical studies, NC has been shown [9] to produce desirable theoretical properties grounded in statistical theory. This penalty retains the ability to train each of the networks both simultaneously and incrementally.

2.8 Weighting Contributions from Ensemble Members

The method used to combine the individual outputs h_j into the ensemble output $\mathcal{H}$ has a significant effect on ensemble accuracy [51]. If the members of $\mathcal{H}$ have very low covariance, then it can be expected that each member will have different areas in which it is most accurate. Simple methods of weighting the member outputs (e.g. setting all $w_j = \frac{1}{N}$) cannot be expected to perform as well in input regions where the members of $\mathcal{H}$ have significantly different outputs.

There are several approaches to combining outputs from ensemble members into a single coherent output for the ensemble. Some of the most common approaches are explored here. In this section, we let our set of ensemble members be $\mathcal{H} = \{h_j \mid 1 \leq j \leq M\}$, where M is the number of members in the ensemble. We use $\mathcal{H}(\cdot)$ to represent the function computed by the entire ensemble and $\hat{y}_i$ to represent the output of the ensemble in response to the input x_i (i.e. $\mathcal{H}(x_i) = \hat{y}_i$). When referring to individual members of an ensemble, $h_j(\cdot)$ represents the function computed by the j-th ensemble member, $h_j(x_i)$ is the j-th ensemble member's output in response to the i-th input, and $\hat{y}_i^{(j)}$ is the member's output (i.e. $h_j(x_i) = \hat{y}_i^{(j)}$).

2.8.1 Plurality Voting

Plurality voting [25] is one of the simplest methods of weighting ensemble members in a classification context. For classification, the inputs x_i are samples and the outputs y_i are class labels. Under plurality voting, the ensemble choice is defined to be the class for which the most ensemble members voted. Formally, let $\hat{y}_{ik}^{(j)}$ be the value that member h_j estimates for the probability that $x_i \in$ class k. The set of members that believe x_i is a member of class k is defined as

$$S_k = \left\{ h_j \mid \hat{y}_{ik}^{(j)} = \max_{k'} \hat{y}_{ik'}^{(j)} \right\} \tag{2.32}$$

Thus, S_k is the set of all ensemble members that gave x_i the highest probability of being in set k. The ensemble's classification output is then

$$\mathcal{H}(x_i) = \arg\max_k |S_k| \tag{2.33}$$

where $|S_k|$ denotes the size of set S_k. $\mathcal{H}(x_i)$ is then the class label for which the most ensemble members voted.

2.8.2 Basic Ensemble Method (BEM)

The Basic Ensemble Method (BEM) [42] takes the simple average of all the ensemble members; this can be used in either a classification or a regression scenario. The ensemble output under BEM is given by

$$\mathcal{H}(x_i) = \frac{1}{M}\sum_{j=1}^{M}(h_j(x_i)) \tag{2.34}$$

If the correct function value to be estimated is given by $f(x_i)$, in the BEM we can define the misfit function for each member as

$$m_j(x_i) = f(x_i) - h_j(x_i) \tag{2.35}$$

which allows us restate the network output as

$$\mathcal{H}(x_i) = f(x_i) - \frac{1}{M}\sum_{j=1}^{M}(m_j(x_i)) \tag{2.36}$$

Interestingly, it can be shown ([42]) that as long as $m_j(x_i)$ are mutually independent with zero mean, the error in estimating $f(x_i)$ can be made arbitrarily small by increasing the population size of $\mathcal{H}$. This result helps to highlight the need for diversity amongst the members of an ensemble; the mutual independence in $m_j(\cdot)$ is directly affected by the amount of diversity amongst the members of $\mathcal{H}$.

2.8.3 Linear Combinations

Linear combinations of the ensemble members' output is a more generalized form of the BEM that is applicable to regression problems. The ensemble output is calculated as

$$\mathcal{H}(x_i) = \sum_{j=1}^{M} w_j h_j(x_i) \tag{2.37}$$

where w_j is a weight assigned to each ensemble member such that $0 \le w_j \le 1$ and $\sum_{j=1}^{M} w_j = 1$.

There are many techniques to determining the values of w_j. In [42], the BEM is extended to the Generalized Ensemble Method (GEM) by the addition of a weighting term. It is further shown that the optimal weights (optimal in the sense that the mean squared error is minimized over the training set) is

$$w_j = \frac{\sum_i C_{ji}^{-1}}{\sum_k \sum_i C_{ki}^{-1}} \tag{2.38}$$

where C is the covariance matrix of the misfit functions, such that $C_{ij} = E[m_i(x)m_j(x)]$ (where $E[\cdot]$ is the expected value). The optimal mean square error under this weighting is

$$MSE = \left[\sum_{i,j} C_{ij}^{-1}\right]^{-1} \tag{2.39}$$

This result depends on two assumptions: the rows and columns of C are linearly independent and we have a reasonable estimate of C. In the case where we have nearly duplicate members in the population $\mathcal{H}$, we will have nearly linearly dependent rows and columns in C, making the inversion calculation unstable and our estimate of C^{-1} unreliable. This reinforces the need for a sufficient level of diversity amongst the members of the ensemble.

Chapter 3

Mitigation of Catastrophic Interference within a Neural Network

3.1 The Fixed Expansion Layer Feedforward Neural Network

The motivation behind the fixed expansion layer (FEL) neural network, first introduced in [13], is similar to the motivation for activation sharpening: reducing the overlap of pattern representations within the network. The FEL network addresses the problem of representational overlap by exploiting an augmented MLP architecture which includes the addition of an *expansion* hidden layer to the network, as depicted in Figure 3.1. The weights for this layer are fixed during network initialization and remain unchanged during subsequent network operations. As a result, the FEL framework inherently supports an incremental, online learning process and exploits sparse encoding to latch onto previously learned input/output mappings.

During the feedforward phase, the FEL neurons are triggered in order to present a consistent sparse representation of the input pattern to the output layer, as illustrated in Figure 3.2. The sparsity of the triggered FEL neurons protects the input-to-hidden layer weights which were not used in the sparse coding from the backpropagated error signal (see Figure 3.3), thus preventing the network weights from changing drastically when exposed to new information, which mitigates the effects of catastrophic interference. Sparsity thus serves as means of latching on to older information by selectively gating weight update signal propagation through the network.

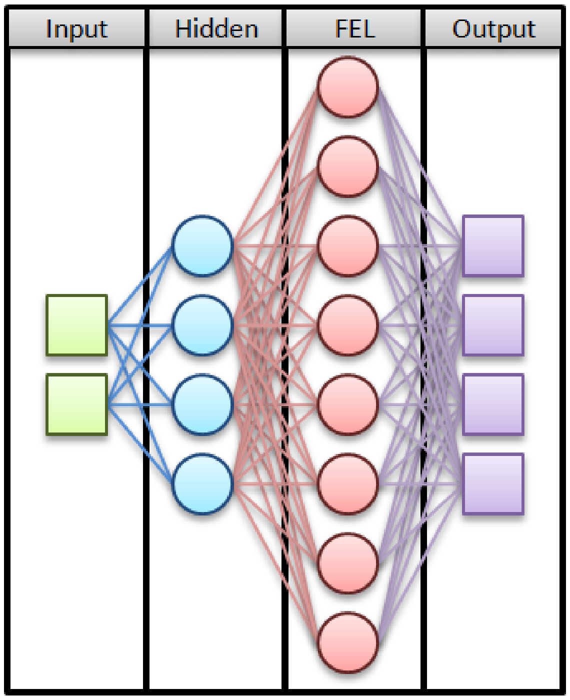


Figure 3.1: Architecture of the FEL neural network

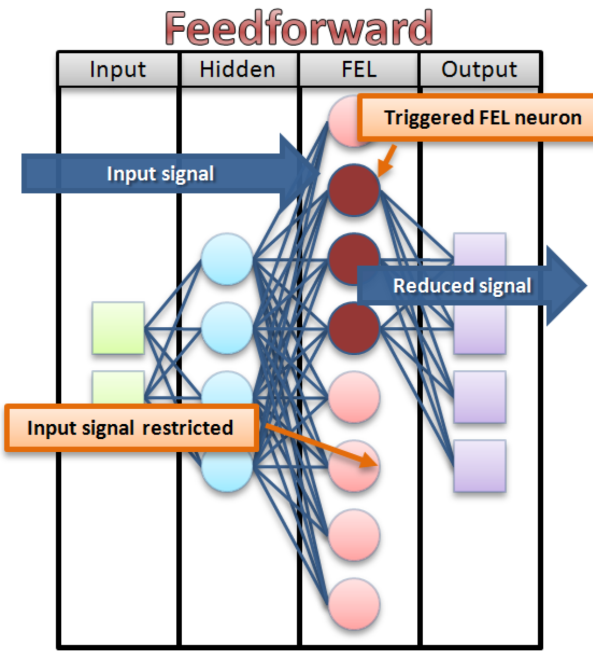


Figure 3.2: FEL neural network operation - feedforward

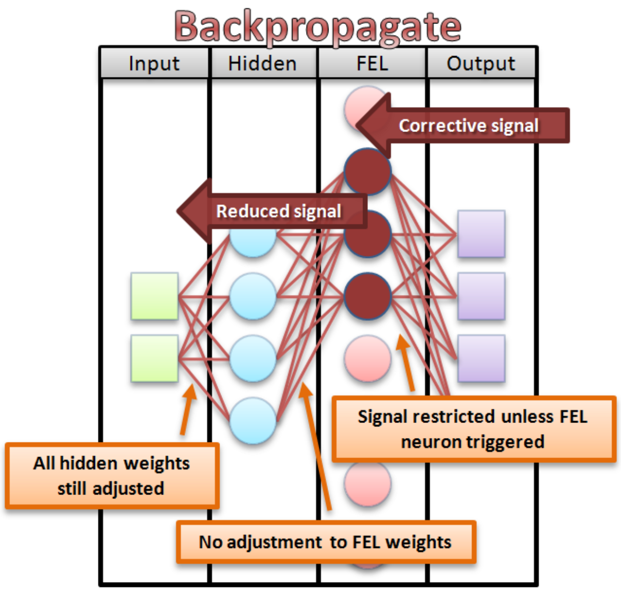

Backpropagate
Input
Hidden
FEL
Output
Corrective signal
Reduced signal
Signal restricted unless FEL neuron triggered
All hidden weights still adjusted
No adjustment to FEL weights

Figure 3.3: FEL neural network operation - backpropagation

Consistency of representations within the FEL neurons is critical to the accuracy of the network; the network cannot retain information if the dense signal from the hidden layer is not sparsely encoded in a consistent fashion. This requirement is the main factor to consider in deciding how the activation values of the FEL neurons are to be calculated.

Two approaches to sparse representational consistency have been investigated; the first approach, the parameterized FEL network, creates sparsity by using fixed weights that are not fully connected, directly evaluating the full fixed layer signal, and then sparsifying the signal heuristically. Building off of the lessons learned from the parameterized approach, the second technique, the feature-sign FEL network, uses techniques that has greater theoretical foundations within the sparse-coding literature and that directly forms a sparse representation of the hidden layer signal.

3.2 Parameterized FEL Network

One approach to the FEL activation problem is to perform a traditional feedforward operation (from the input layer to the hidden layer) and then to heuristically sparsify the FEL activation values; this is the approach investigated in [13]. Using the fixed weights and the activation values of the hidden layer, activation values are calculated for all FEL neurons. Once these values have been calculated, we heuristically determine which FEL neurons contain the most significant information about the hidden layer and zero out the values of the other FEL neurons.

3.2.1 Weight Initialization

The weights between the hidden layer and the FEL are set when the neural network is created and are not updated during the learning process. These FEL weights must facilitate expansion of the signal contained in the activations of the hidden layer neurons into a more sparse representation of that signal in the FEL neurons. To achieve this, each FEL neuron is only connected to a portion of the neurons in the hidden layer. If each hidden neuron were fully connected to each FEL neuron (as in traditional feedforward neural network weighting), each FEL neuron's activation would be a function of the full hidden layer signal; the FEL layer's signal would be just as dense as that of the hidden layer. We therefore select only a portion of the hidden layer neurons to contribute to each FEL neuron.

To initialize the FEL weights, we first determine the number of hidden layer neurons that contribute to each FEL neuron's activation (N_C) and the number of hidden layer neurons that will inhibit the activation of each FEL neuron (N_H). For each FEL neuron, we randomly select N_C contributory neurons and assign them a contributory weight (v_C) of 1. We then select N_H inhibitory neurons and assign them an inhibitory weight value (v_H) of $-\frac{v_C}{N_H}$. The selection of contributory and inhibitory neurons is performed such that each neuron in the hidden layer will contribute and inhibit the same number of FEL neurons.

3.2.2 Neuron Triggering

During training, only a small portion of the FEL neurons (the 'triggered neurons') have nonzero activation values. Any hidden layer neuron connected to a triggered FEL neuron will receive a corrective training signal and consequently update all of its input layer weights (depicted in Figure 3.3); if all FEL neurons were triggered, then all hidden neurons would receive a training signal, and all input-hidden weights would be updated during each training iteration. We trigger some number (N_+) of neurons that have the largest activation value as well as some number (N_-) of neurons that have the smallest activation values. Intuitively, this can be thought of as selecting the neurons that strongly 'agree' or strongly 'disagree' with the hidden layer signal; the strength of the agreement or disagreement is a result of the contributory or inhibitory weights between the hidden layer neurons and the FEL neurons. These triggered neurons then have their activation value set to a constant value (v_+ or v_-), and all other FEL neurons have their activation value set to 0.

By setting the activation value of the triggered FEL neurons to specific values (as opposed to using their actual activation values), we are effectively limiting the information that can be sent between the hidden layer neurons and the output layer neurons. In effect, we are dividing the learning process into two parts; the hidden layer weights are adjusted in order to create the sparse representation that will be most informative to the output layer, and the output layer weights are adjusted in order to interpret the sparse FEL signal into an accurate representation of the desired output.

3.2.3 Examination of the Parameterized Approach

The fixed expansion layer feedforward neural network is an effective approach to combating catastrophic interference and learning under non-stationary training data [13]. Using the FEL, we are able to obtain higher accuracy than other current approaches to this problem. There are minimal computational requirements added due to this approach; the only additional processing required is the extra feedforward layer. The FEL feedforward neural network is able to retain more previously learned information about the training data, even when the training data is non-stationary in nature.

However, there are some aspects of the parameterized approach that are troublesome. In the above approach, there are eight constants ($N_C, N_H, v_c, v_h, N_A, N_D, v_p, v_n$) which must be manually specified; this can lead to the requirement of empirically tuning the parameters for any given problem. Ideally, we want as few 'magic constants' as possible and would prefer a more data-driven approach to FEL operation which might be able to make more use of the resources available in the fixed expansion layer.

Such a data-driven approach would ideally address the need for a special initialization of the fixed weights, the issue of which FEL neurons to trigger, and the value to assign to each triggered neuron. The Feature-Sign FEL network was developed in order to address these concerns and to incorporate advances in sparse coding theory into the algorithm.

3.3 Feature-Sign FEL Network

Sparse coding can be used as an improved method for determining the FEL activation neurons and their values. In such framing, the activations of the hidden layer denote the dense signal, the activations of the expansion layer represent the sparse signal, and the FEL weights act as the basis for the transformation. We seek to minimize the L_1 norm of the fixed expansion layer neurons while at the same time retaining key information from the hidden layer neurons. This is equivalent to the following optimization formulation:

$$\min_x ||y - Ax^2|| + \gamma ||x||_1, \tag{3.1}$$

where y is the activation of the hidden layer, A is the FEL weight matrix, x is the FEL activation (over which the minimization is performed), and γ is a penalty constant which acts to balance the desire for a sparse FEL activation and retention of information from the hidden layer.

This problem can be efficiently solved using the feature-sign search algorithm introduced in [31]. Equation (3.1) can equivalently be written as

$$\min_x ||y - Ax^2|| + \gamma \sum_{i=1}^{N} |x_i|, \tag{3.2}$$

where N is the number of elements in x. If we know the signs of the elements in x, then we can replace each of the terms $|x_i|$ with either x_i (if $x_i > 0$), $-x_i$ (if $x_i < 0$), or 0 (if $x_i = 0$). Considering only nonzero coefficients, this reduces (3.2) to a standard, unconstrained quadratic optimization problem (QP), which can be solved analytically and efficiently. The feature-sign search algorithm operates by maintaining an *active set* of potentially non-zero coefficients in x and the sign (positive or negative) of these values; all other members of x are assumed to be zero. Given an active set, we can analytically solve (3.2); in addition, given the corresponding minimization solution, an improved active set can be obtained (if it exists).

Table 3.1: The Feature-Sign Search Algorithm

```
**Feature-sign search algorithm**
1: Initialize $x := \vec{0}$, $\theta := \vec{0}$, and $active\ set := \{\}$, where $\theta_i \in \{-1, 0, 1\}$ denotes sign($x_i$).
2: From zero coefficients of $x$, select $i = \arg\max_i \left|\frac{\partial ||y-Ax||^2}{\partial x_i}\right|$.
       Activate $x_i$ (add $i$ to the $active\ set$) only if it locally improves the objective, namely:
       If $\left|\frac{\partial ||y-Ax||^2}{\partial x_i}\right| > \gamma$, then set $\theta_i := -1$, $active\ set := \{i\} \cup active\ set$ .
       If $\left|\frac{\partial ||y-Ax||^2}{\partial x_i}\right| < -\gamma$, then set $\theta_i := 1$, $active\ set := \{i\} \cup active\ set$ .
3: Feature-sign step:
       Let $\hat{A}$ be a submatrix of $A$ that contains only the columns corresponding to the $active\ set$.
       Let $\hat{x}$ and $\hat{\theta}$ be subvectors of $x$ and $\theta$ corresponding to the $active\ set$.
       Compute the analytical solution to the resulting unconstrained QP (minimize$_{\hat{x}}||y - \hat{A}\hat{x}||^2 + \gamma\hat{\theta}^T$):
       $\hat{x}_{new} := (\hat{A}^T\hat{A})^{-1}(\hat{A}^T y - \gamma\hat{\theta}/2)$,
       Perform a discrete line search on the closed line segment from $\hat{x}$ to $\hat{x}_{new}$:
       Check the objective value at $\hat{x}_{new}$ and all points where any coefficient changes sign.
       Update $\hat{x}$ (and the corresponding entries in $x$) to the point with the lowest objective value.
       Remove zero coefficients of $\hat{x}$ from the $active\ set$ and update $\theta :=$ sign($x$).
4: Check the optimality conditions:
       (a) Optimality condition for nonzero coefficients: $\frac{\partial ||y-Ax||^2}{\partial x_i} + \gamma\text{sign}(x_j) = 0, \ \forall x_j \neq 0$
       If condition (a) is not satisfied, go to Step 3 (without any new activation); else check condition (b).
       (b) Optimality condition for zero coefficients: $\frac{\partial ||y-Ax||^2}{\partial x_i} \leq \gamma, \ \forall x_j = 0$
       If condition (b) is not satisfied, go to Step 2; otherwise return $x$ as the solution.
```

Each such step reduces the objective function. It has been shown that the feature-sign search algorithm converges to a global optimum of the optimization problem (3.2) in a finite number of steps [31]. The algorithm is shown in Table 3.1.

The framing of the fixed expansion layer network within the sparse coding domain and the application of the feature-sign search algorithm lead to significant advantages. By applying this approach, we are able to eliminate eight constants ($N_C, N_H, v_c, v_h, N_A, N_D, v_p, v_n$) and replace these with a single constant (γ). Beyond offering a more firm theoretical grounding to the FEL approach, this improvement results in significantly higher accuracy over the original approach, as discussed in Section 5.4.

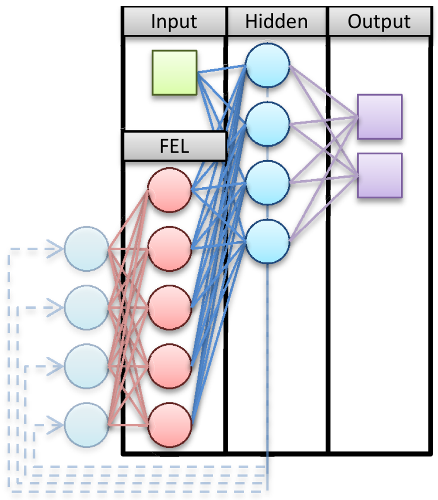


Figure 3.4: FEL Elman recurrent neural network

3.4 Fixed Expansion Layers in Elman Recurrent Neural Networks

Applying the FEL technique to Elman's Simple Recurrent Network (SRN) [14] (shown in Figure 2.4) allows the system to capture temporal dependencies. In this recurrent FEL (rFEL) network, the expansion layer now acts to prevent convergence of the weights between the context neurons and the hidden layer neurons. This is done by gating the context signal using an expansion layer, as depicted in Figure 3.4.

Elman recurrent neural networks ([15], structure depicted in figure 2.4) are widely used to approximate functions that have temporal dependencies. However, as an extension of regular neural networks, Elman networks also suffer from the problem of catastrophic forgetting.

Consider the scenario where we are trying to approximate $f : \mathbb{X} \times \mathbb{S} \rightarrow \mathbb{Y} \times \mathbb{S}$ with an Elman recurrent neural network (RNN). In other words, we have that $f(x_i, s_t) = (y_i, s_{t+1})$, where $x_i \in \mathbb{X}$ is the input, $s_t \in \mathbb{S}$ is the current state, $y_i \in \mathbb{Y}$ is the output, and $s_{t+1} \in \mathbb{S}$ is the new state. This is the typical scenario for approximation by a RNN. Suppose there is a subset of the input space $A \subset \mathbb{X}$ such that $\forall x_i \in A\ \exists y_i \in \mathbb{Y}\ \forall s \in \mathbb{S}\ f(x_i, s_t) = (y_i, s_{t+1})\ s_t = s_{t+1}$. Simply put, there is some region of the input space (A) where $f(x_i, s)$ has no actual dependence on the state history. If many samples from A occur sequentially, then the weights from the recurrent layer to the hidden layer will converge toward 0, thus disregarding the state history. When we once again observe any $x_i \notin A$, the network will no longer have the information from those weights required to adequately approximate f.

This effect is mitigated by isolating sparse state signals contained within dense hidden layer signal by inserting a FEL between the hidden layer and the recurrent portion of the input layer (see figure 3.4). This FEL will act as a gate on the state signal; in the above example, some FEL neurons would latch on inputs $x_i \in A$. The weights from those FEL neurons to the input layer will converge to 0, and the information stored in the other weights will not be lost.

3.5 Target Variation Trace

An alternative method of mitigating catastrophic forgetting is to vary the rate of learning in order to maximize the learning of new information and minimize repetition of already learned information. In order to do this, we compute some metric to quantify whether or not new information has been presented and use this metric to proportionately control the learning rate of the network. We refer to this approach as the 'target variation trace'.

We can compare this approach with others used for the mitigation of catastrophic interference: activation sharpening seeks to maximize the distribution of information representation within the hidden layer by encouraging individual hidden neurons to latch on to specific areas of the input space; the fixed expansion layer network seeks to minimize representational overlap by learning a sparse representation of the dense hidden layer signal and using this sparse transformation for information storage; by using the target variation trace, the network is encouraged to rapidly explore regions of the error space that have not been observed recently while slowing the rate of network convergence when in regions of the error space that have previously been observed.

Consider a standard gradient descent neural network with error and weight updates defined by

$$E(x_t) = \frac{1}{2} \sum_j \left[e_j(x_t)^2 \right] \tag{3.3}$$

$$\Delta W_t \propto -\eta \nabla_W E \tag{3.4}$$

where x_t is the t-th input vector, $E(x_t)$ is the total network error for the t-th input, $e_j(x_t)$ is the error for the j-th neuron of the output layer in response to the t-th input, W is the matrix of network weights, ΔW_t is the change in weights in response to the t-th input, η is a step-size constant, and $\nabla_W E$ is the gradient of the total network error $E(x_t)$ with respect to the weights W.

The target variation trace approach moderates the rate of weight change depending on the change in error signal. Formally, the network uses

$$\Delta W_t \propto -\gamma(t)\nabla_W E \tag{3.5}$$

$$\gamma(t) \propto \eta \times d\left(E(x_t), E(x_{t-1})\right) \tag{3.6}$$

where $\gamma(t)$ is the new step size for time t, η is the original network step size, and $d(\cdot,\cdot)$ is a distance metric measuring the change in error from time $(t-1)$ to time t.

Intuitively, this means that the network will make small weight adjustments when the same type of approximation error is made between time $(t-1)$ and t. If the network is fed constant values for the input pair (x, y), then $\gamma(t)$ will approach 0, and the network weights will stop being adjusted. Therefore, we can prevent the network weight convergence which ordinarily cause catastrophic forgetting.

In order to gracefully handle multidimensional error signals, we use the cosine distance metric defined as

$$d\left(E(x_t), E(x_{t-1})\right) \equiv \frac{E(x_t) \bullet E(x_{t-1})}{\|E(x_t)\|\|E(x_{t-1})\|} \tag{3.7}$$

where $\bullet$ denotes the dot product operator and $\|\cdot\|$ denotes the L_2 norm.

Chapter 4

Mitigation of Catastrophic Interference within Ensembles of Neural Networks

When examining the results from experiments with a single FEL network, it was observed that, when FEL networks suffer from some degree of forgetting, they tend to do so in different regions. It is possible that this property is exploitable by ensembles of FEL networks; this chapter examines the FEL network within an ensemble setting and presents several techniques for ensuring diversity within members of an ensemble.

4.1 Diversification using Output Distribution

As discussed in Section 2.7, diversification of members within an ensemble is critical to the performance of the ensemble as a whole.

Using explicit weight estimation by ensemble members provides a useful mechanism for weighting member contribution, but it should also be useful for increasing diversity within an ensemble. When we estimate the error, we are effectively estimating the variance of the output. Taking this idea further, we can envision an ensemble member not trying to estimate just the correct output, but estimating a probability distribution defined by (μ, σ^2) in which the correct output is likely to fall.

This extension is useful for determining the corrective signal, but it also allows for more information-theoretical methods of increasing diversity within the ensemble. Given a collection of (μ_j, σ_j^2) outputs, we can calculate the Jensen-Shannon divergence [34] between members j and k as

$$JSD(P \parallel Q) = \frac{1}{2} D_{\mathrm{KL}}(P \parallel M) + \frac{1}{2} D_{\mathrm{KL}}(Q \parallel M) \tag{4.1}$$

$$D_{\mathrm{KL}}(P \| Q) = \sum_i P(i) \ln \frac{P(i)}{Q(i)} \tag{4.2}$$

where P is the normal distribution $\mathscr{N}(\mu_j, \sigma_j^2)$, Q is the normal distribution $\mathscr{N}(\mu_k, \sigma_k^2)$, and M is a distribution defined by $M = \frac{1}{2}(P + Q)$. $D(P \parallel M)$ is the Kullback–Leibler divergence between P and M. Further, the quantity $\sqrt{JSD(P \parallel Q)}$ is a metric [16] which can serve as a metric indicating the distance between two ensemble members. This formulation allows for a theoretically sound method of diversifying ensemble members.

To apply this technique, we add a penalty term to each network's error function in order to increase the JSD between the network's output and the ensemble's output. This gives us the new error function

$$e_j(x_i) = \frac{1}{2} \sum_{k=1}^{N} \left(y_{ik} - \hat{y}_{ik}^{(j)} \right)^2 + \lambda \sqrt{JSD(h_j \parallel \mathcal{H})} \tag{4.3}$$

where λ is a weighting term and $JSD(h_j \parallel \mathcal{H})$ is the Jensen-Shannon divergence between the j-th ensemble member's output and the combined output of the entire ensemble.

4.2 Explicit Error Estimation by Ensemble Members

Ensemble weighting can also be accomplished by explicitly estimating member error for ensemble members (in this case we're using FEL networks). For example, in approximating a N-dimensional function, we also require each ensemble member to estimate its own error, so

$$h_j(x_i) = \begin{bmatrix} \hat{y}_{i1}^{(j)} \\ \vdots \\ \hat{y}_{iN}^{(j)} \\ \hat{e}_j(x_i) \end{bmatrix} \tag{4.4}$$

Table 4.1: Accuracy comparison between a single FEL network and various ensembles of FEL networks

	Accuracy
Single FEL network	0.800
Basic Ensemble Method	0.815
Estimated MSE weighting	0.829
Individual output error estimate weighting	0.827
Estimated max absolute error weighting (separate network)	0.830
Estimated max absolute error weighting (same network)	**0.863**

where h_j is trained with the goals that $\hat{y}_i^{(j)} \approx \hat{y}_i$ and also $\hat{e}_j(x_i) \approx e_j(x_i)$, where $e_j(x_i) = \frac{1}{2}\sum_{k=1}^{N}\left(y_{ik} - \hat{y}_{ik}^{(j)}\right)^2$ (or alternatively the L_1 norm can be used, where $e_j(x_i) = \frac{1}{2}\sum_{k=1}^{N}\left|y_{ik} - \hat{y}_{ik}^{(j)}\right|$)
.

Using this estimate, we can set w_j, the weight assigned to the j-th ensemble member, proportional to the estimated error. So, we have $w_j \propto \hat{e}_j(x_i)^{-1}$, which will lower the overall error in $\mathcal{H}$ as long as our estimation of $e_j(x_i)$ is reasonable. Analysis of the results indicate high correlation (correlation coefficient of 0.45) between the error estimates and the actual errors. Several methods of using this estimate for weighting have been tried, and the results are summarized in table 4.1. The best results have been obtained when requiring each ensemble member to estimate its maximum absolute error. If we define the error of the individual output for the k-th dimension as $e_{jk}(x_i)$, the maximum absolute error is given by

$$\max_{1\leq k\leq N} |e_{jk}(x_i)| \tag{4.5}$$

4.3 Backpropagation Through a Consolidation Neural Network

We propose to further explore the impact of explicitly using a neural network to consolidate the outputs of each ensemble member into an accurate ensemble output. Given outputs h_j, the consolidation network $\mathcal{H}$ will compute $f'(h_1, \dots, h_j, \dots, h_M) = \hat{y}_i$. For a N-dimensional function, the ensemble error is given by

$$E(x_i) = \frac{1}{2}\sum_{k=1}^{N}(y_{ik} - \hat{y}_{ik})^2 \tag{4.6}$$

This error is then backpropagated through the consolidation neural network. The traditional term for the member error (using no penalty terms) is

$$e_j(x_i) = \frac{1}{2}\sum_{k=1}^{N}\left(y_{ik} - \hat{y}_{ik}^{(j)}\right)^2 \tag{4.7}$$

For ensembles of neural networks, we adjust member networks by changing the weights attached to the output nodes. If $\hat{y}_{ik}^{(j)}$ is the output node for the k-th dimension of the j-th member, we adjust the output weights W by a factor proportional to

$$\frac{\partial e_j(x_i)}{\partial \hat{y}_{ik}^{(j)}} = y_{ik} - \hat{y}_{ik}^{(j)} \tag{4.8}$$

The novelty of this work comes from modifying the member error function. We can backpropagate through $\mathcal{H}$ in order to determine the amount of the ensemble error attributable to each ensemble member. More importantly, we can calculate the portion of the ensemble error attributable to each output node of h_j as

$$\frac{\partial E(x_i)}{\partial \hat{y}_{ik}^{(j)}} = \frac{\partial E(x_i)}{\partial \mathcal{H}}\frac{\partial \mathcal{H}}{\partial h_j}\frac{\partial h_j}{\partial \hat{y}_{ik}^{(j)}} \tag{4.9}$$

We can use this knowledge in the weight adjustment such that

$$\Delta W_k^{(j)} \propto \frac{\partial e_j(x_i)}{\partial \hat{y}_{ik}^{(j)}} + \rho\frac{\partial E(x_i)}{\partial \hat{y}_{ik}^{(j)}} \tag{4.10}$$

where ρ is a weighting term balancing the ensemble member's error in estimating f against the amount of error in $\mathcal{H}$ as a whole which is caused by h_j. The intuition is that, by minimizing both estimation error and the error contributed to the ensemble, we can appropriately weight each ensemble member along with encouraging diversity among the ensemble members.

4.4 Ensemble Diversification in FELNN Ensembles

FEL neural networks introduce an additional layer with several unique properties. In ensembles of FEL networks, this property can be exploited in order to both increase diversity in the ensemble and increase utilization of the FEL layers within ensemble members.

Each FEL network within an ensemble is initialized with different values for the fixed weights, which are used as the basis for the sparse coding; as a result, each ensemble member performs the sparse coding process differently. For any given input, each ensemble member sparsely code the hidden neuron activations will different efficiency levels. For example, one ensemble member might have a sparse coding requiring the activation of 5 FEL neurons, while another ensemble member might only have 2 FEL neurons activated.

Using this property, we can define the *coding efficiency* for parameterized FEL network ensemble members as

$$c_i(x_t) \equiv \sum_{r \in V_+} |r| + \sum_{s \in V_-} |s| \tag{4.11}$$

where $c_i(\cdot)$ is the coding efficiency for the i-th ensemble member and x_t is the input at time t. The values of $r \in V_+$ are the values of the N_+ FEL neurons with the largest activation values, and the values of $s \in V_-$ are the values of the N_- FEL neurons with the smallest activation values; these values are taken before the FEL neurons are triggered (i.e. before the values are set to v_+or v_- using the procedure specified in Section 3.2).

This value is used for both weighting and diversity purposes. The weight of the i-th ensemble member (for the input x_t) is calculated as

$$w_i(x_t) = \frac{c_i(x_t)}{\sum_{j=1}^{M} c_j(x_t)} \tag{4.12}$$

and the overall output of the ensemble is given by

$$\mathcal{H}(x_t) = \sum_{i=1}^{M} w_i(x_t) h_i(x_t) \tag{4.13}$$

where $h_i(\cdot)$ is the output of the i-th ensemble member and M is the total number of ensemble members.

Diversity is increased by only training the two ensemble members with the greatest and least coding efficiency values. For each training sample, only the two members with the maximum and minimum values of c_i are trained using backpropagation. The intuition behind this is that divergence can be increased by only modifying the most efficient and least efficient networks.

Chapter 5

Experimental Results

5.1 Auto-Associative Binary Pattern Reconstruction

5.1.1 Task Description

For the first task, we compare FEL network performance to two of the commonly utilized approaches for mitigation of catastrophic forgetting. We also show the performance of a standard multilayer perceptron (MLP) feedforward neural network for reference. Exploration of the problem of catastrophic interference has traditionally been within the domain of auto-associative pattern learning [38]; this task compares the ability of a network to retain previously learned information after learning new information.

The auto-associative binary pattern reconstruction test is performed as follows. First, the network is trained over a set of 20 patterns, where each pattern consists of 32 binary values. The goal of the network is to recreate the input provided; the network has 32 real-valued outputs that do not get rounded or thresholded. The network is trained over all 20 patterns until the mean squared error for the set is less than 0.06. Once the network has learned the 20 base items, we present a new pattern to the network. The network is trained once using this intervening item, and we then measure the mean squared error over the original set of base items in order to determine how much of the original information was retained.

Table 5.1: Pattern reconstruction accuracy for the traditional catastrophic forgetting task.

Intervening Items	Standard MLP	Pseudo-rehearsal	Activation sharpening	**FEL NN**
1	0.930	0.934	0.926	**0.941**
3	0.922	0.931	0.917	**0.936**
5	0.918	0.930	0.914	**0.932**
7	0.914	0.927	0.909	**0.929**
10	0.914	0.926	0.911	**0.926**

5.1.2 Networks Tested

We evaluated the FEL neural network against a standard multilayer perceptron (MLP) feedforward neural network, a network using activation sharpening, and a network using pseudorehearsal. All networks have 32 input neurons, 16 hidden layer neurons, and 32 output neurons. For activation sharpening, the two hidden layer neurons with the largest values were sharpened by a factor of $\alpha = 0.001$. Pseudorehearsal was performed by generating 32 pseudopatterns after original training over the binary patterns. Every time an interleaving pattern is learned, a random pseudopattern is selected and presented for training.

In the FEL neural network 128 neurons were used in the sparse (fixed) layer. Each FEL node received inputs from half of the hidden layer nodes (i.e. $N_C = 4$, $N_H = 4$), with excitatory weights of $v_C = 1$ and inhibitory weights of $v_H = -0.25$. For the neuron triggering, the $N_C = 4$ neurons with the largest activation value and the $N_H = 1$ neuron with the smallest activation value were used. The positive trigger value was $v_p = 0.5$ and the negative trigger value was $v_n = -1$.

5.1.3 Simulation Results

Each network was used in 100 independent test runs, and the results were used in order to determine the mean accuracy, standard deviation, and the 95% confidence interval for the mean accuracy.

A plot of these values is shown in Figure 5.1, with some detailed results presented in Table 5.1. The FEL network performed significantly better than the standard MLP network and the network using activation sharpening and was slightly better than the network using pseudorehearsal.

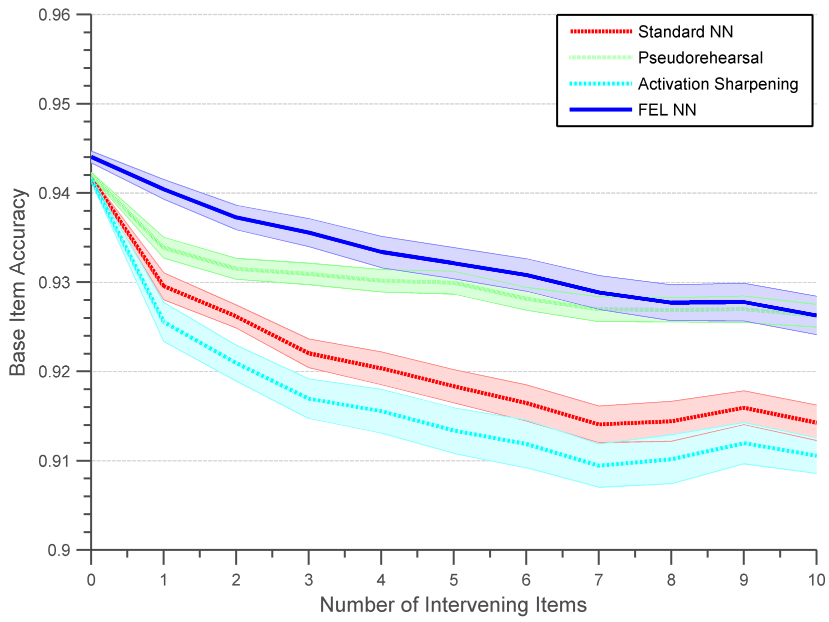


Figure 5.1: Pattern reconstruction accuracy for the traditional catastrophic forgetting task. Each line represents the mean classification accuracy, with the shaded proportion representing the 95% confidence interval.

These results show that the FEL network performs well in the domain typically used for testing catastrophic forgetting.

5.2 Single Learner Non-stationary Gaussian Distribution Classification Task

We study the addition of the FEL on a classification task, whereby clusters of observations with non-stationary properties are used as training input. Inputs consist of data from four clusters of two dimensional points, where each cluster has a predefined mean and standard deviation and samples for that cluster are drawn from a Gaussian distribution. The process involves 50,000 training iterations followed by 1,000 testing iterations. The neural network is given the point's x and y coordinates as a two dimensional input, and produces a four dimensional output representing the class index. This problem is trivial when we train over samples drawn from each distribution for the entire training period. All algorithms evaluated are able to achieve 100% accuracy under such conditions.

However, in order to study each algorithm's ability to resist catastrophic interference, we do not train over all four clusters during the training process. Instead, we present samples from all four clusters during the first portion of training, followed by presenting samples from only two of the clusters (i.e. the 'primary' clusters) during the rest of training. During this second portion of training, no samples from the other two clusters (i.e. the 'restricted' clusters) are presented. Testing is still performed over both the primary and the restricted clusters with the goal being to determine whether or not the network is able to retain information about the restricted clusters upon being presented with many samples from only the primary clusters. The proportion of training that only references the primary clusters (i.e. the 'non-stationary percentage') was adjusted in order to measure how well each algorithm performed under varying amounts of interference.

We tested the fixed layer feedforward neural network against a standard multilayer perceptron (MLP) feedforward neural network, a network using activation sharpening, a network using pseudorehearsal, and a network using the target variation trace method. All networks use 2 input neurons, 16 hidden layer neurons, and 4 output neurons. For activation sharpening, the 2 hidden layer neurons with the largest values were sharpened by a factor of $\alpha = 0.001$. Pseudorehearsal was performed by generating a new pseudopattern every 1000 training iterations. Every 100 training iterations, a random pseudopattern is selected and presented for training.

For the FEL neural network, a FEL of 128 neurons was used. Each FEL received input from half of the hidden layer ($N_C = 4$, $N_H = 4$), with contributory weights of $v_C = 1$ and inhibitory weights of $v_H = -\frac{1}{4}$. For the neuron triggering, the $N_+ = 4$ neurons with the largest activation value and the $N_- = 1$ neuron with the smallest activation value were used. The positive trigger value was $v_+ = \frac{1}{2}$ and the negative trigger value was $v_- = -1$.

5.2.1 Simulation Results

For each value of the non-stationary percentage, 100 independent test runs were performed for each algorithm, and the results were averaged in order to determine the mean accuracy, standard deviation, and the 95% confidence interval ($\alpha = 0.05$) for the mean accuracy.

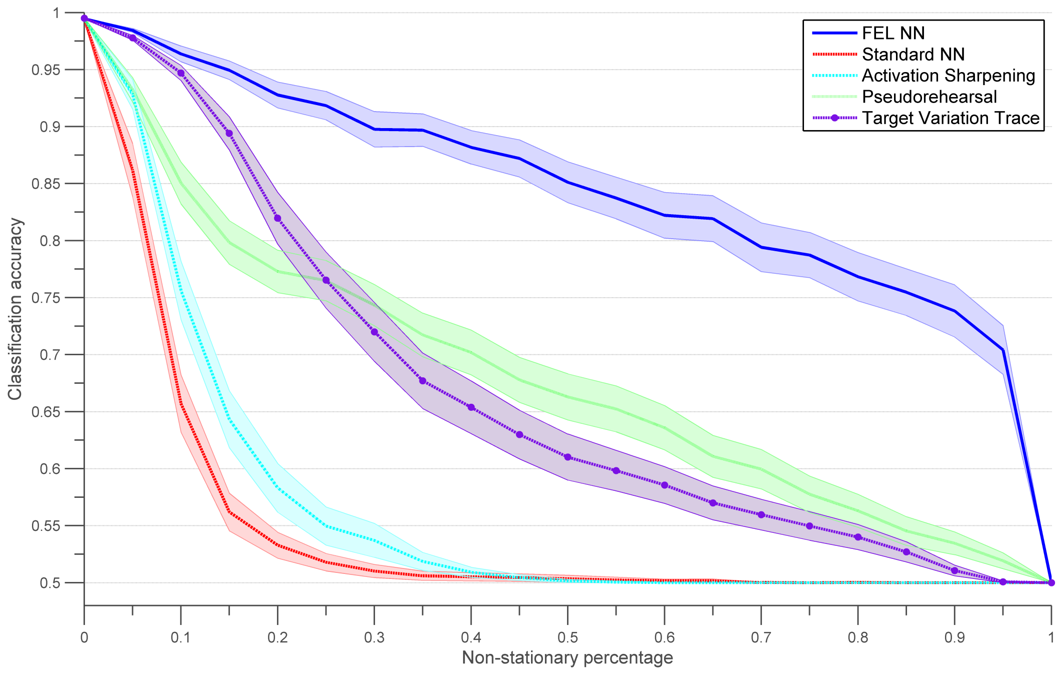


Figure 5.2: Single learner non-stationary Gaussian distribution classification accuracy. Each line represents the mean classification accuracy, with the shaded proportion representing the 95% confidence interval. 'Non-stationary percentage' refers to the percent of training that was performed using only samples from the primary clusters.

Table 5.2: Classification rates for various catastrophic interference mitigation schemes compared to the proposed method

Non-stationary percentage	Standard MLP	Pseudo-rehearsal	Activation sharpening	Target Variation Trace	**FEL NN**
0.00	1.00	1.00	1.00	1.00	**1.00**
0.25	0.52	0.76	0.55	0.74	**0.92**
0.50	0.51	0.66	0.50	0.59	**0.85**
0.75	0.50	0.58	0.50	0.55	**0.79**
1.00	0.50	0.50	0.50	0.50	**0.50**

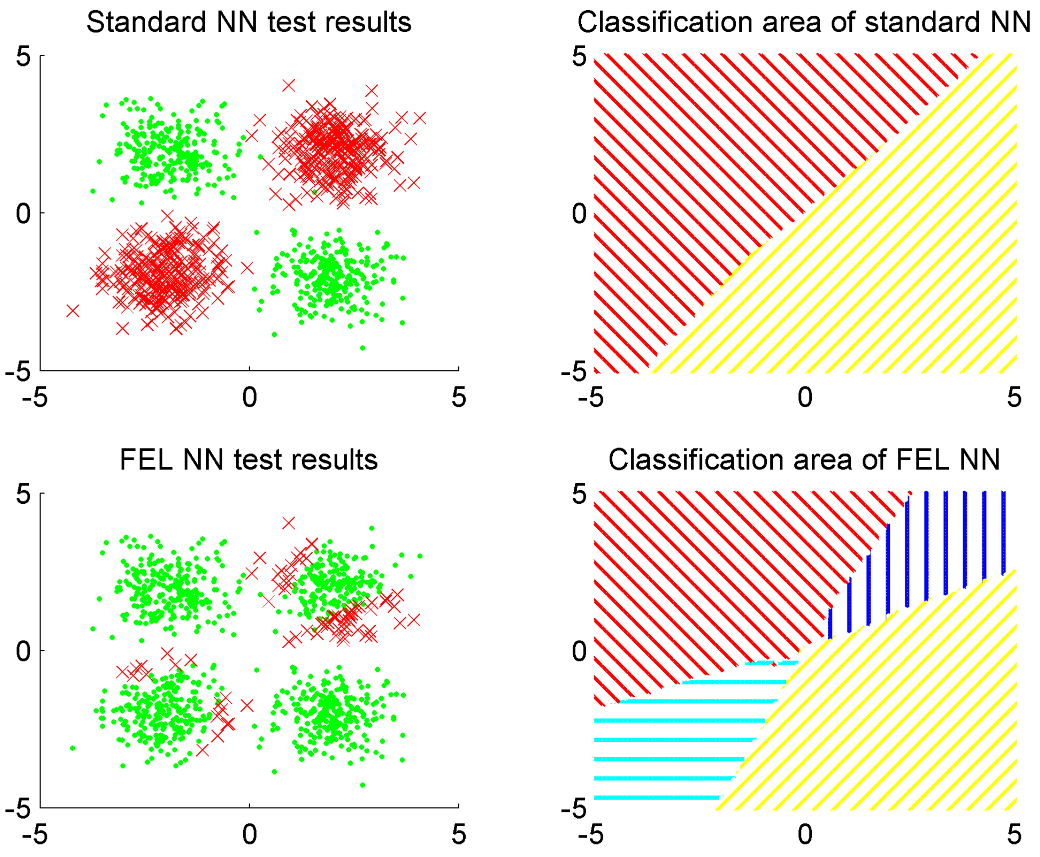


Figure 5.3: Single learner non-stationary Gaussian distribution classification region detail, comparing a standard multilayer perceptron (on top) with a FEL network (below). The primary clusters are centered at (-2,2) and (2,-2). The left shows the results of the test set, with green circles representing correct classifications and red X's representing incorrect classifications. The right displays the entire input space and the classification that each network would make for a point in that region. Note that the standard MLP loses the classification region for both of the restricted clusters, while the FEL network only loses the borders of the restricted clusters.

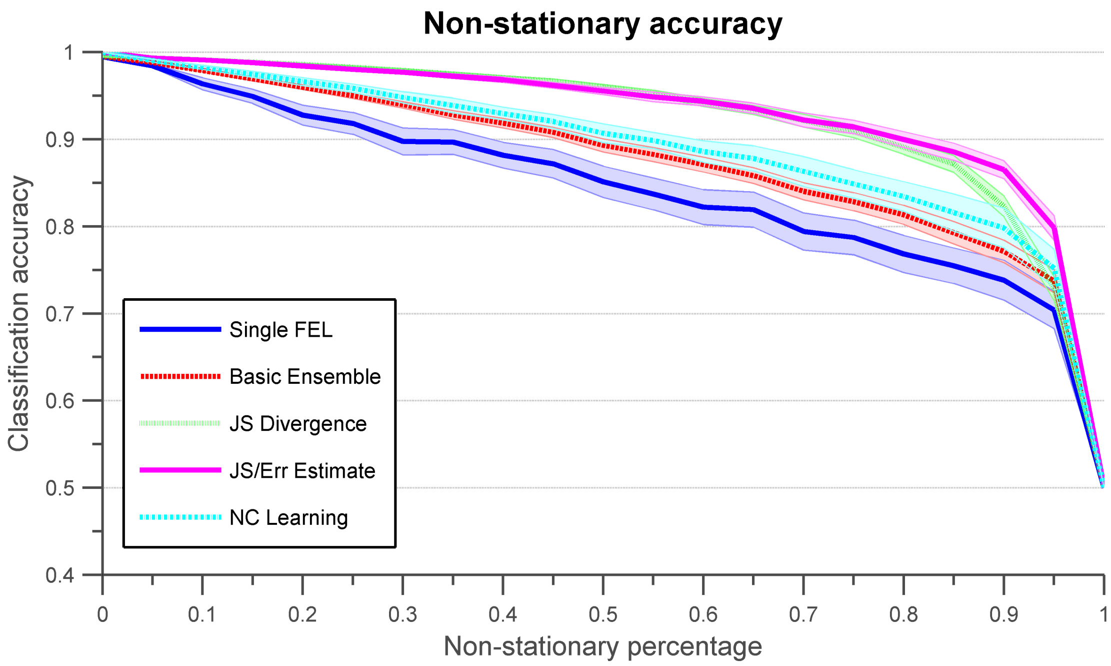
A plot of each network's accuracy is shown in Figure 5.2, with some detailed values presented in table 5.2. For all non-stationary percentages, the FEL shows the highest classification accuracy. Furthermore, the accuracy drops off at a roughly linear rate as the non-stationary percentage increases; the exponential decay in accuracy shown by the standard MLP is characteristic of catastrophic interference. Figure 5.3 provides more detail for one test run (with a non-stationary percentage of 75%). The standard MLP loses all ability to classify samples from the restricted clusters, while the FEL network only misclassified samples that lie near the edges of the restricted clusters.

5.3 Ensemble Learning Non-stationary Gaussian Distribution Task

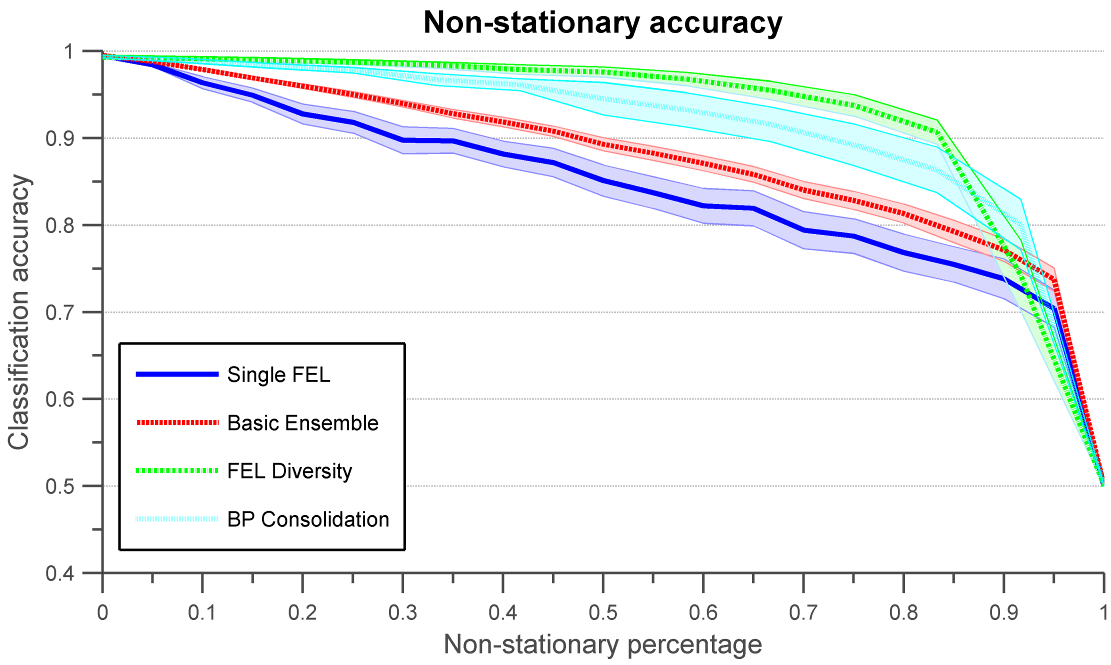
To compare ensemble techniques, we evaluated various schemes for composing multiple FEL neural networks into an ensemble. Each ensemble is composed of 7 FEL neural networks, using the same parameters as in the single learner case. We compared Jensen-Shannon divergence as a diversification term with and without weighting the learners proportionally to their estimated error. Furthermore, both a basic ensemble which uses the mean value of member outputs and an ensemble trained using NCL were considered. The negative correlation penalty term used was $\gamma = 0.5$ (as in [8]), and the Jensen-Shannon divergence term is weighted using a convex with the MSE term, where $\gamma = 0.8$. The temperature parameter in the estimated error weighting was $\tau = 0.1$.

For each value of the non-stationary percentage, 100 independent test runs were performed for each algorithm, and the results were used in order to determine the mean accuracy, standard deviation, and the 95% confidence interval for the mean accuracy.

A plot of the accuracy obtained by using an ensemble of FEL neural networks is illustrated in Figure 5.4, with some detailed values presented in Table 5.3. Results from the ensemble technique comparison show that significant accuracy gains can by made by using the FEL neural network in an ensemble setting. All ensemble techniques perform significantly better than a single learner, with the most significant accuracy differences occurring at higher levels of non-stationarity.



(a) JSD/NC detail



(b) FEL/BPC detail

Figure 5.4: Ensemble learning non-stationary Gaussian distribution classification accuracy in ensembles of FEL networks applied to a basic clustering task . Each line represents the mean classification accuracy, with the shaded proportion representing the 95% confidence interval. 'Non-stationary percentage' refers to the percent of total training iterations that were performed using only samples drawn from the primary clusters.

Table 5.3: Classification accuracy for various ensemble techniques

Non-stationary percentage	Basic Ensemble	Negative Correlation Learning	Jensen-Shannon Divergence	JSD with Error	Back-propagation Consolida-tion	FEL Metric Diversity
0.00	1.00	1.00	1.00	1.00	1.00	1.00
0.25	0.95	0.96	0.98	0.98	0.98	**0.99**
0.50	0.89	0.91	0.96	0.96	0.95	**0.98**
0.75	0.83	0.85	0.91	0.91	0.89	**0.94**
0.90	0.77	0.80	0.82	**0.87**	0.82	0.79
1.00	0.50	0.50	0.50	0.50	0.50	0.50

To examine the possible performance of error estimated weighting, we investigated the correlation between the estimated error and the actual error. The results are shown in Figure 5.5. It can be seen that the correlation level remains significantly high across the entire range of non-stationarity values and, correspondingly, the Jensen-Shannon divergence ensemble using estimated error weighting performs better than other approaches at higher levels of non-stationarity.

5.4 Non-stationary MNIST Classification Task

The previous tasks illustrate the viability of the FEL network when considering the traditional domain over which catastrophic interference is measured and when considering a simple Gaussian classification problem. This task shows that the FEL network is applicable to more complex tasks and that the FEL network's accuracy is significantly improved by the addition of the feature-sign search algorithm.

This task involves the classification of digits taken from the MNIST database of handwritten digits [30]. Digits 1, 2, 3, and 4 are used for this task. The original MNIST digits are 32x32 pixel grayscale images, with each image consisting of 1024 pixels taking on integer values from 0 (white) to 255 (black). We preprocess the MNIST data by shifting the pixel values to be a real number between 0 and 1, centering the data, and using principal component analysis (PCA) to reduce the 1024-dimensional data to 128 dimensions. This dimensionality reduction captures approximately 94% of the variance of the original data. By taking the Moore-Penrose

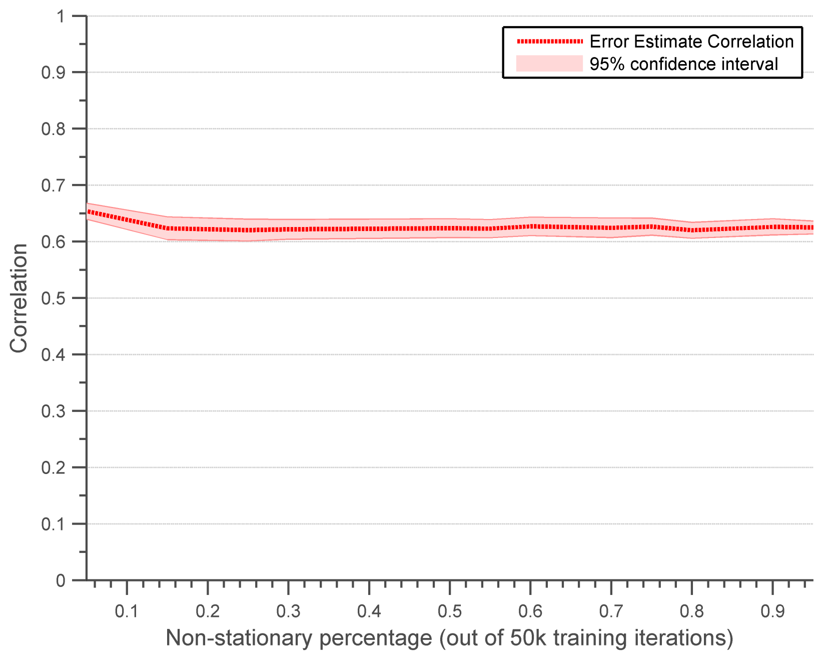


Figure 5.5: Error estimated weighting: Correlation coefficient between the estimated error and the actual error (used for error weighting).

pseudoinverse of the principal component matrix, we can reconstruct the reduced data in order to get a visual representation of the amount of information present in the reduced dataset. This representation is shown in Figure 5.6.

In order to use the MNIST data to study the mitigation of catastrophic interference, we use a similar approach to that used in the Gaussian classification task. The training period is divided into two phases. During the first phase of training, we present training examples of all 4 digits. During the second phase of training, we only present training examples of 2 of the digits (digits 1 and 2). The 'non-stationary percentage' represents the proportion of training time spent in the second phase (where only 2 of the digits were used for training).

We evaluated the performance of a standard MLP network, a FEL network, and a FEL network using the feature-sign search algorithm (FEL-FS). Each network has 128 inputs for the digit and 4 outputs representing the classification of that digit.

For both FEL networks, the number of hidden neurons was increased to 64, and the number of FEL neurons was increased to 512. We also increase the number of hidden neurons in the MLP

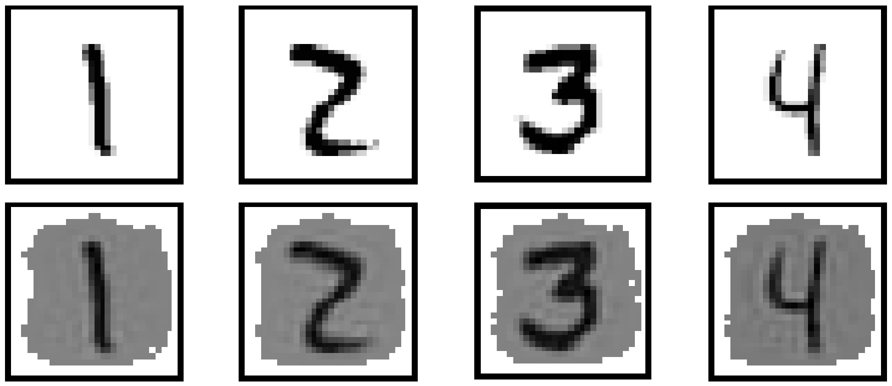


Figure 5.6: Original MNIST handwritten digits (top) and the reconstructed PCA reduced digits (bottom)

network to 78. This number of hidden neurons was chosen in order to give the MLP network the same amount of learning resources as the FEL networks; each FEL network is able to change the weights between the 128 input neurons and the 64 hidden neurons as well as the weights between the 512 fixed expansion layer neurons and the 4 output neurons, giving the FEL networks 10,240 weights with which to store information. The MLP network is able to change the weights between the 128 input neurons and the 78 hidden neurons as well as the weights between the 78 hidden neurons and the 4 output neurons; this gives the MLP network 10,296 weights with which to store information.

The parameters for the MLP network and the FEL network are the same as those used in the single-learner Gaussian classification task. For the FEL-FS network, we use $\gamma = 3.5$. Additionally, we do not apply the FEL weight initialization technique to the FEL-FS network (leaving the input-hidden weights randomly initialized as in the MLP network).

For each value of the non-stationary percentage, 100 independent test runs were performed for each algorithm, and the results used in order to determine the mean accuracy, standard deviation, and the 95% confidence interval for the mean accuracy.

A plot of the accuracy of the networks tested is shown in Figure 5.7, with some detailed values being presented in Table 5.4. These results clearly show that both the FEL and the FEL-FS networks perform significantly better than the standard MLP network on the more complex MNIST classification task, while the FEL-FS network consistently outperforms the basic FEL

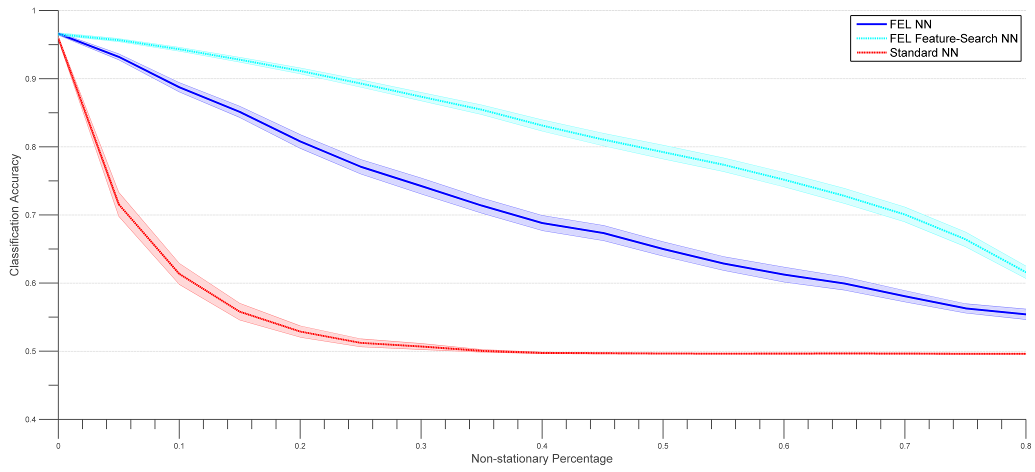


Figure 5.7: Classification accuracy for the MNIST classification task. Each line represents the mean classification accuracy, with the shaded proportion representing the 95% confidence interval. 'Non-stationary percentage' refers to the percent of total training iterations that were performed using only two of the four possible digits.

Table 5.4: Classification accuracy for the MNIST classification task

Non-stationary percentage	Standard MLP Network	FEL Network	**FEL Feature-Sign Network**
0.00	0.9595	0.9661	**0.9654**
0.25	0.5123	0.7708	**0.8930**
0.50	0.4967	0.6501	**0.7923**
0.75	0.4963	0.5629	**0.6643**
0.90	0.4965	0.5227	**0.4989**
1.00	0.4966	0.4964	**0.4944**

network. Additionally, we can see that the performance of the FEL-FS network has a more linear degradation profile compared to the exponential degradation of the MLP and FEL networks.

5.5 Auto-associative binary sequence reconstruction

Auto-associative binary sequence reconstruction can be viewed as a sequence-based version of a traditional catastrophic forgetting task. Mainstream exploration of the problem of catastrophic interference has commonly been restricted to the domain of auto-associative pattern learning [38];

this task compares the ability of a network to retain previously captured sequences of observations after demonstrating the learning of new sequences. In particular, the capacity of networks to retain information about sequences of patterns is evaluated, rather than that of a single presentation for each pattern.

The auto-associative binary sequence reconstruction test was performed as follows. A number of binary pattern sequences were randomly generated, where each sequence consisted of two binary patterns (composed of the values -1 and 1). The objective is to learn to reconstruct the first part of the sequence *after* the complete sequence is observed. This test was repeated for sequences using patterns consisting of 16, 32, and 64 dimensions in order to measure each algorithm's ability to scale to more challenging settings. First, the network is trained over a batch of 20 randomly generated sequences. The network is repeatedly trained over this batch until the mean squared error for the training set reaches an acceptable level. For the 16-dimensional and 32-dimensional tests, a threshold of 0.06 was used; for the 64-dimensional test, this threshold was increased to 0.075 due to the fact that not all networks were able to reach the 0.06 error level. Once the network has learned the 20 base sequences, a new sequence is presented to the network. The network is trained using this intervening sequence, and then the mean squared error over the original set of base items is measured in order to determine how much of the original information was retained.

This experiment was conducted using the rFEL network, a network using pseudorehearsal, a network using activation sharpening, a system using dual networks, and a standard Simple Recurrent Network (SRN). For the pseudorehearsal, activation sharpening, and dual networks we use SRNs as the base network. All networks used 16 hidden neurons, a step size of 2^{-5}, and were trained with standard stochastic gradient descent. After the pseudorehearsal network was successfully trained, 32 pseudopatterns were generated; a pseudopattern was used for training after the introduction of each new sequence. During training of the dual network system, 20 pseudopatterns were created with the learning network and used to transfer information to the memory network. Once a new sequence was introduced, the learning network was trained using 3 pseudopatterns generated by the memory network. For activation sharpening, 8 neurons were sharpened with a sharpening factor of 0.1, and the rFEL network used a sparse expansion layer with 128 neurons and $\gamma = 2.5$.

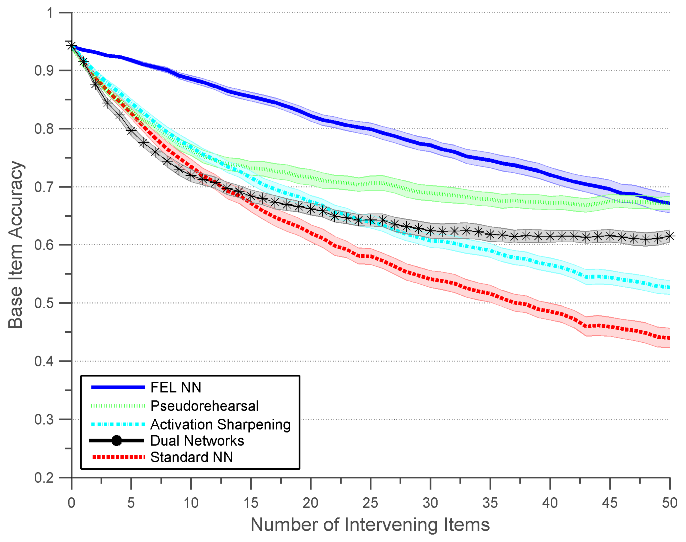


Figure 5.8: Binary sequence reconstruction: 16-dimensional pattern

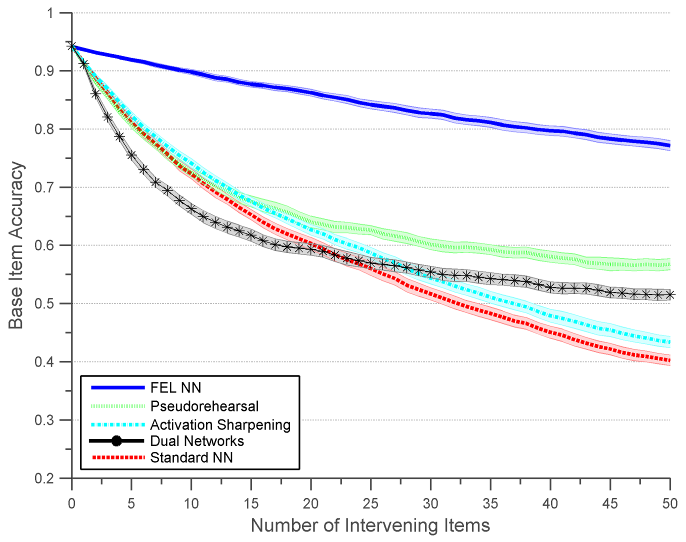


Figure 5.9: Binary sequence reconstruction: 32-dimensional pattern

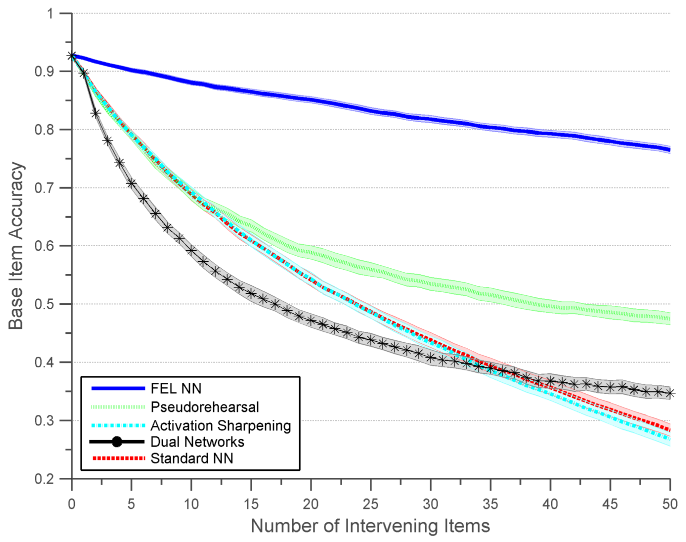


Figure 5.10: Binary sequence reconstruction: 64-dimensional pattern

Table 5.5: Binary sequence reconstruction: 32-dimensional pattern detail. Entries denote the mean accuracy of the network over 50 independent test runs.

# of New Items	Simple Recurrent Network	Pseudo-rehearsal	Activation Sharpening	Dual Networks	**Recurrent FEL**
10	0.7228	0.7253	0.7411	0.6626	**0.8981**
20	0.6031	0.6402	0.6267	0.5929	**0.8623**
30	0.5166	0.6016	0.5440	0.5542	**0.8264**
40	0.4507	0.5805	0.4791	0.5276	**0.7971**
50	0.4024	0.5675	0.4338	0.5151	**0.7714**

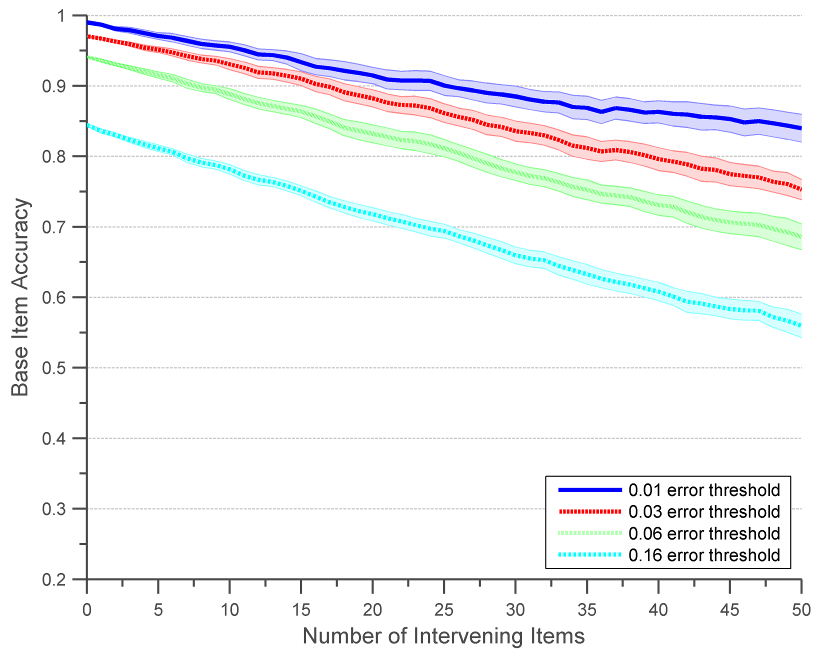


Figure 5.11: FEL reconstruction accuracy for 16-dimensional pattern using different initial error thresholds

For each setting, 50 independent runs were conducted from which the mean accuracy, standard deviation, and the 95% confidence interval for the mean accuracy were derived. These values are shown in Figures 5.8, 5.9, and 5.10; detailed results for the 32-dimensional test are illustrated in Table 5.5. As can be observed from these graphs, a pseudo-linear degradation is exhibited by the recurrent FEL network, while performance of other schemes follows a sharper degradation profile.

Sensitivity of FEL accuracy to initial training period

An interesting phenomenon can be observed by comparing Figure 5.8 with Figure 5.10; the FEL network retains more information and has a higher accuracy when tested on a sequence composed of 64-dimensional patterns than when tested on a sequence composed of 16-dimensional patterns. One would expect that the 64-dimensional test would prove more difficult, given the same network configurations and parameters. Accordingly, the performance of all other methods decreased as expected between the two tests.

Upon further exploration, this behavior was discovered to be caused by the length of the initial training period. All networks used a hidden layer consisting of 16 neurons. For the non-FEL networks, this resulted in the network input being 32 dimensions; 16 for the pattern input and 16 for the context signal (the prior values of the hidden layer neurons). For the FEL network, the total network input was 144 dimensions; 16 for the pattern input and 128 for the context signal, which had been sparsely coded by the FEL. When training with the original 16-dimensional sequences, all networks easily reached the required initial 0.06 error threshold for the training set before being presented with new patterns and re-tested. Furthermore, all networks reached this threshold with fewer training iterations than was required to reach the 0.075 threshold used for the 64-dimensional task.

For the non-FEL methods, there was no significant difference in performance caused by the shorter initial training time. However, due to the larger input dimensionality, the FEL network's performance was hindered by the shortened initial training period used during the 16-dimensional task; the 16 hidden neurons had not yet found optimal weights to assign to the 144 input/context neurons. The FEL network was able to reach the 0.06 threshold before this occurred. Figure 5.11 presents a comparison of FEL network performance on the 16-dimensional task when different initial error thresholds are used, and there are significant differences in performance caused by the longer initial training times required for the lower error thresholds.

5.6 Non-stationary MNIST sequence classification task

The previous experiment demonstrated the viability of the recurrent FEL network within the domain of auto-associative memory of temporal sequences. The next test case was devised in order to evaluate the ability of the rFEL network to classify sequences of complex visual patterns, namely involving the classification of images taken from the MNIST database of handwritten digits [30]. The digits 1, 2, 3, and 4 were selected for this task. The goal of this task is to correctly classify sequences consisting of two of the four digits. Consequently, four different sequences were considered: $(1, 3)$, $(1, 4)$, $(2, 3)$, $(2, 4)$.

The original MNIST digits are 32x32 pixel grayscale images, with each image consisting of 1024 pixels taking on integer values from 0 (white) to 255 (black). The MNIST data was

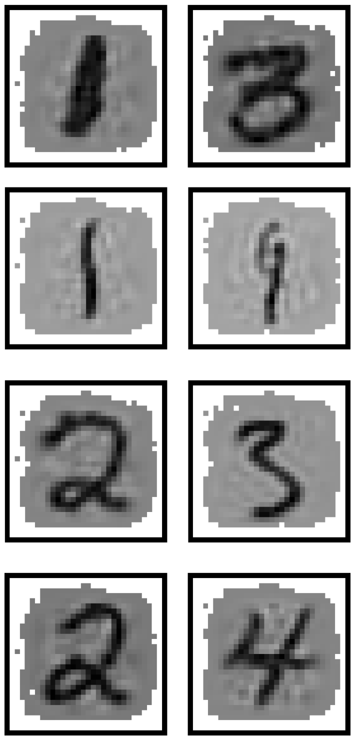


Figure 5.12: Sample MNIST sequences. Each row is a sample of a sequence used in the classification task, with the two digits being presented sequentially.

preprocessed by shifting the pixel values to be a real number between 0 and 1, centering the data, and using principal component analysis (PCA) to reduce the 1024-dimensional data to 128 dimensions. This dimensionality reduction captured approximately 94% of the variance of the original data. By taking the Moore-Penrose pseudoinverse of the principal component matrix, we can reconstruct the reduced data in order to get a visual representation of the amount of information present in the reduced dataset. Samples of the reconstructed sequences are depicted in Figure 5.12.

When the sample distribution does not change during the training period, all algorithms evaluated were able to achieve over 90% accuracy. In order to employ the MNIST data to study the mitigation of catastrophic interference, the training sequences were presented such that the networks were exposed to nonstationary inputs.

In order to study each algorithm's ability to mitigate catastrophic interference in the presence of non-stationarity, the sample distribution was varied over the training period such that samples were drawn from all classes of sequences during the initial phase of the training process, followed by a duration of time in which samples were drawn only from two of the classes (i.e. 'primary' classes). During this latter phase of the training, no samples from the other two classes (i.e. 'restricted' classes) were presented. Testing was performed over both the primary and the restricted classes with the goal being to determine the degree to which the network was capable of retaining information about the restricted classes after being presented with multiple observations drawn only from the primary classes. The proportion of training iterations that pertained to the primary classes (i.e. the 'non-stationary percentage') was varied in order to measure how well each algorithm performed under different levels of interference.

For this task, a total of 8,000 training samples were presented. During the first phase of the training process, samples were drawn randomly from all classes with equal probability, while at the second phase of training, samples were drawn only from the two primary classes. The restricted sequences were $(1, 3),\ (1, 4)$, and the primary sequences were $(2, 3),\ (2, 4)$.

The same algorithms used in the previous section were applied in this experiment. All networks hosted 64 hidden neurons, used a step size of 2^{-5} and were trained using standard stochastic gradient descent. For the pseudorehearsal algorithm, pseudopatterns were created and occasionally used during the training process; following every 400 training iterations, 50 new pseudopatterns were generated and added to the collection of pseudopatterns, and 25 random pseudopatterns were drawn from the collection and used for training. The dual network system was trained in a similar fashion. Following every 400 training samples, 50 pseudopatterns were created and used to transfer information into the memory network, then 100 pseudopatterns were created and used to transfer information to the learning network. For activation sharpening, 32 neurons were sharpened with a sharpening factor of 0.001, and the rFEL network used a sparse expansion layer with 512 neurons and $\gamma = 0.4$; the lower γ value reflects the need for more contextual information than in the previous task.

Each network was used in 35 independent runs; the results were averaged and are shown in Figure 5.13. The 95% confidence intervals were calculated, but only the rFEL network's interval is shown in the plot in the interest of avoiding clutter. The rFEL network performed statistically significantly better than the other algorithms, but the difference in performance between the other algorithms was not statistically significant. The MNIST sequence classification task was more challenging for the other algorithms than originally anticipated, with performance dropping rapidly after only a minor degree of non-stationarity was introduced. For example, using a non-stationary

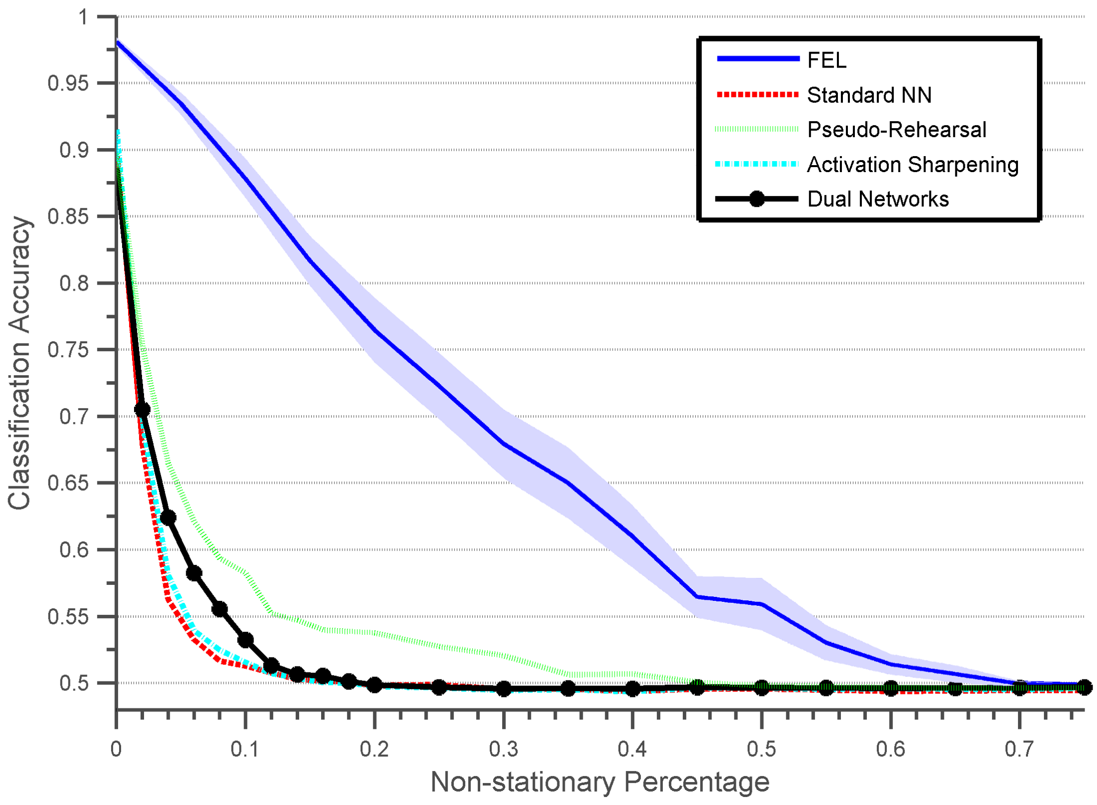


Figure 5.13: MNIST sequence classification accuracy

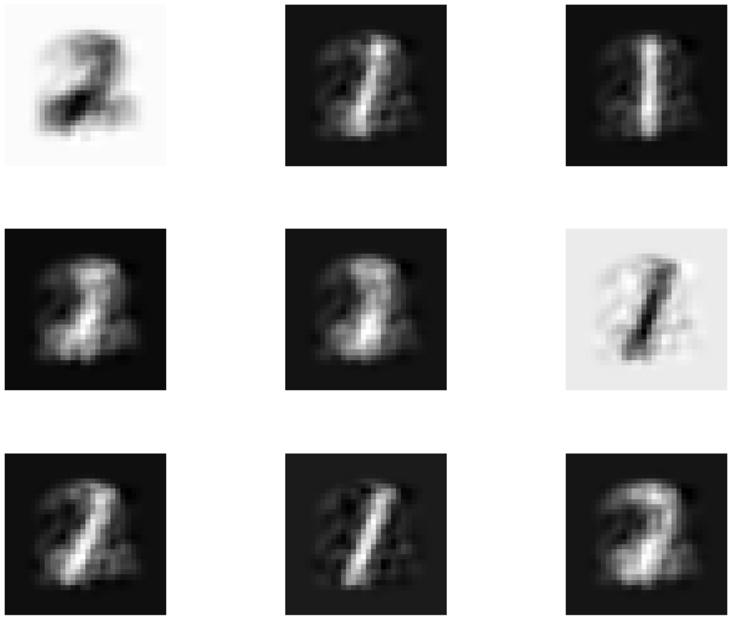


Figure 5.14: Recurrent FEL neuron affinity

level of 0.06, 7,520 training samples from all sequences were presented, with the reduced classes being excluded from only the last 480 training samples. Under these conditions, the Simple Recursive Network's classification accuracy dropped from its baseline accuracy of 0.8962 to 0.5324, while the rFEL network's accuracy dropped from 0.9811 to 0.9347 over the same interval.

The motivation behind the FEL network is that the fixed layer neurons can form a meaningful sparse representation of the hidden layer signal. As such, it was expected that different FEL neurons would consistently react to different features within the input signal. To measure the FEL neuron affinity, the following data was collected. An 'affinity vector' was stored for each FEL neuron. For each test sample, the input signal was multiplied by the activation value of each FEL neuron and added to that neuron's affinity vector; if the FEL neuron was not activated, no change was made to its affinity vector. Once the evaluation process completed, the affinity vectors represented a weighted sum of input values to which the FEL neuron responded. By reconstructing a 32x32 pixel grayscale image from these weighted values (using the procedure described earlier), we can visually observe the input to which each FEL neuron responded. A collection of these affinity vectors is presented in Figure 5.14. It can be seen that the FEL neurons consistently reacted in response to specific patterns; one can observe varying levels of positive and negative responses to the digits 1 and 2, and in multiple cases the same neuron shows a positive response to one digit and a negative response to the other digit. This provides interesting insight into the actual representations captured by the sparse encoding of the FEL layer, further supporting the overall significant results observed in mitigating catastrophic interference.

Chapter 6

Summary and Future Work

6.1 Summary of Contributions

This work introduced the fixed expansion layer neural network, which is uniquely designed to mitigate forgetting effects in parameterized supervised-learning systems. This is achieved by exploiting sparse encoding for latching long-term representations. Learning is inherently achieved in an incremental, online manner, with modest requirements for additional memory and computational resources. Moreover, the feature-sign search algorithm can be used to significantly improve FEL accuracy. When embedded in an ensemble of learners, the FEL exhibits significantly higher accuracy in the presence of non-stationary inputs, without the need for tuning any application-specific parameters.

The following significant contributions were presented in this manuscript:

- The fixed expansion layer neural network was introduced as a method of mitigating the effects of catastrophic interference, compared with several popular techniques found in the literature, and shown to perform more effectively than the other methods.

- The FEL network was enhanced using the feature-sign algorithm for sparse coding. The resulting feature-sign FEL network was shown be more effective than the original FEL network.

- A fixed expansion layer was incorporated into a recurrent neural network, with the resulting recurrent FEL network able to successfully mitigate the effects of catastrophic interference within the state-dependent domain.
- The target variation trace was introduced as an alternative means for combating catastrophic forgetting; this method was shown to be more effective than some alternatives, but not as effective as the FEL network.
- The Jensen-Shannon divergence was applied to achieve diversity amongst members of an ensemble learning system, with this application resulting in increased accuracy of the ensemble.
- An improved method of computing the weighted contribution of each member to the ensemble's output was presented wherein each ensemble member computes an estimate of it's expected error. This approach was shown to improve accuracy when combined with the above JSD diversity mechanism.
- Divergence and weighting of ensemble members were accomplished by using a neural network to consolidate member outputs. Error was backpropagated through the consolidation network in order to determine the training signal to be used for each ensemble member.
- A divergence and weighting strategy for ensembles of FEL networks was presented, wherein the sparse coding efficiency of each member is calculated and used for output weighting and for determining which ensemble members to train.

6.2 Publications

The following manuscripts were published as a result of this work:

- Robert Coop and Itamar Arel. Mitigation of catastrophic interference in neural networks using a fixed expansion layer. In *Circuits and Systems (MWSCAS), 2012 IEEE 55th International Midwest Symposium on*, pages 726 –729, August 2012.

- Robert Coop, Aaron Mishtal, and Itamar Arel. Ensemble Learning in Fixed Expansion Layer Networks for Mitigating Catastrophic Forgetting. *Neural Networks and Learning Systems, IEEE Transactions on*, accepted pending minor revisions, 2013.

- Robert Coop and Itamar Arel. Mitigation of Catastrophic Forgetting in Recurrent Neural Networks using a Fixed Expansion Layer. In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, accepted for publication, 2013.

6.3 Future work

Using a fixed expansion layer has been shown to be an effective means of mitigating the effects of catastrophic interference in neural networks, but there are several aspects of this approach that could be improved upon by future work.

Of primary concern is the computational complexity of the approach; the original parameterized FEL network operates very efficiently, but requires specification of many constant values. The feature-sign algorithm adds theoretical grounding, removes these constant values, and greatly increases accuracy; however, using the feature-sign algorithm for minimization of Equation 3.1 increases the computational complexity of the FEL network. A more efficient data-driven mechanism for sparsely encoding the hidden layer signal into the expansion layer would reduce this complexity.

Further exploration of the algorithm parameters would prove beneficial. In particular, the FEL weight matrix is currently composed of a number of excitatory and inhibitory weights (for the parameterized network) or randomly initialized (for the feature-sign network). The feature-sign search algorithm can find an optimal set of coefficients for a dense signal and a fixed coding basis; there are many approaches to optimizing the basis used for sparse coding (including some from the original feature-sign paper in [31]). Future work might explore the possible ways that the basis might be further leveraged to improve accuracy.

This work has presented the FEL network and its application to several auto-associative and classification tasks. This represents only a small sample of the tasks for which the FEL network might prove effective. Non-stationary environments have been a major focus of this work, but the mitigation of catastrophic interference provided by the FEL network is likely effective for other environments. Of particular interest would be environments where the inputs come from multiple regimes (or modes of operation); each regime may have a stationary distribution of input samples, but there may be several different distributions that can be used. The improved memory of the fixed expansion layer should allow the network to retain information about many different modes of operation, even if a regime has not been observed for quite some time. There are numerous real-world scenarios that fit into this scheme and are of particular interest, such as financial data streams and recurring weather patterns.

The ensemble diversity methods presented were shown to be effective. Further work in this area could be applied to the FEL specific mechanisms for diversity. Specifically, application of FEL specific diversity and weighting techniques have not yet been extensively tested on the feature-sign FEL network.

The FEL scheme can be broadly studied in deep learning neural networks, which is a growing area of research. This can greatly improve deep machine learning schemes in enriching their feature representations.

Bibliography

[1] Herve Abdi. A neural network primer. *Journal of Biological Systems*, 2(03):247–281, 1994. 10

[2] D. H. Ackley. *A connectionist machine for genetic hillclimbing*. Kluwer, Boston, 1987. 11

[3] B. Ans and S. Rousset. Avoiding catastrophic forgetting by coupling two reverberating neural networks. *Comptes Rendus de l'Académie des Sciences-Series III-Sciences de la Vie*, 320(12):989–997, 1997. 2, 16, 17

[4] B. Ans, S. Rousset, R.M. French, and S. Musca. Self-refreshing memory in artificial neural networks: learning temporal sequences without catastrophic forgetting. *Connection Science*, 16(2):71–99, 2004. 2, 16

[5] T. Bäck. *Evolutionary algorithms in theory and practice*. Oxford University Press, 1996. 11

[6] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166, 1994. 8

[7] L. Breiman. Stacked regressions. *Machine learning*, 24(1):49–64, 1996. 21

[8] G. Brown and J. Wyatt. Negative correlation learning and the ambiguity family of ensemble methods. *Multiple Classifier Systems*, pages 161–161, 2003. 49

[9] G. Brown and J. Wyatt. The use of the ambiguity decomposition in neural network ensemble learning methods. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, volume 20, page 67, 2003. 23

[10] G. Brown, J. Wyatt, R. Harris, and X. Yao. Diversity creation methods: a survey and categorisation. *Information Fusion*, 6(1):5–20, 2005. 20

[11] AE Bryson and Yu-Chi Ho. Applied optimal control. 1969. *Blaisdell, Waltham, Mass*. 8

[12] Philippe G Ciarlet. *Introduction to numerical linear algebra and optimisation*. Cambridge University Press, 1989. 10

[13] R. Coop and I. Arel. Mitigation of catastrophic interference in neural networks using a fixed expansion layer. In *Circuits and Systems (MWSCAS), 2012 IEEE 55th International Midwest Symposium on*, pages 726 –729, August 2012. 27, 29, 31

[14] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990. 34

[15] J.L. Elman. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7(2):195–225, 1991. 7, 34

[16] D.M. Endres and J.E. Schindelin. A new metric for probability distributions. *Information Theory, IEEE Transactions on*, 49(7):1858–1860, 2003. 38

[17] Roger Fletcher. Practical methods of optimization. 1987. 10

[18] M. Frean and A. Robins. Catastrophic forgetting in simple networks: an analysis of the pseudorehearsal solution. *Network: Computation in Neural Systems*, 10(3):227–236, 1999. 2, 16

[19] R.M. French. Using semi-distributed representations to overcome catastrophic forgetting in connectionist networks. *Connection Science*, 4(3/4):365–378, 1992. 17, 19

[20] R.M. French. Catastrophic interference in connectionist networks: Can it be predicted, can it be prevented? *Advances in Neural Information Processing Systems*, pages 1176–1176, 1994. 1

[21] R.M. French. Using pseudo-recurrent connectionist networks to solve the problem of sequential learning. In *Proceedings of the 19th Annual Cognitive Science Society Conference, NJ*, 1997. 16

[22] R.M. French and A. Ferrara. Modeling time perception in rats: Evidence for catastrophic interference in animal learning. In *Proceedings of the 21st Annual Conference of the Cognitive Science Conference*, page 173–178, 1999. 1

[23] Y. Freund. Boosting a weak learning algorithm by majority. *Information and computation*, 121(2):256–285, 1995. 21

[24] Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, page 148–156, 1996. 21

[25] L.K. Hansen and P. Salamon. Neural network ensembles. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 12(10):993–1001, 1990. 24

[26] M. Hattori. Dual-network memory model using a chaotic neural network. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, page 1–5, 2010. 2, 16

[27] G.E. Hinton and D.C. Plaut. Using fast weights to deblur old memories. In *Proceedings of the 9th Annual Conference of the Cognitive Science Society*, page 177–186, 1987. 2, 16

[28] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning. *Advances in neural information processing systems*, page 231–238, 1995. 21

[29] Yann Le Cun. Learning process in an asymmetric threshold network. In *Disordered systems and biological organization*, pages 233–240. Springer, 1986. 8

[30] Yann Lecun and Corinna Cortes. The MNIST database of handwritten digits. 51, 59

[31] H. Lee, A. Battle, R. Raina, and A.Y. Ng. Efficient sparse coding algorithms. *Advances in neural information processing systems*, 19:801, 2007. 32, 33, 67

[32] H. Lejeune, A. Ferrara, F. Simons, and J.H. Wearden. Adjusting to changes in the time of reinforcement: Peak interval transitions in rats. *Journal of Experimental Psychology: Animal Behavior Processes*, 23(2):211, 1997. 1

[33] J.P. Levy and D. Bairaktaris. Connectionist dual-weight architectures. *Language and Cognitive Processes*, 10(3-4):265–283, 1995. 2, 16

[34] J. Lin. Divergence measures based on the shannon entropy. *Information Theory, IEEE Transactions on*, 37(1):145–151, 1991. 38

[35] Y. Liu and X. Yao. Ensemble learning via negative correlation. *Neural Networks*, 12(10):1399–1404, 1999. 22, 23

[36] M. McCloskey and N.J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *The psychology of learning and motivation*, 24:109–165, 1989. 1, 13

[37] M Minsky and S Papert. Perceptrons: An introduction to computational geometry, 1969. 5

[38] O.M. Moe-Helgesen and H. Stranden. Catastophic forgetting in neural networks. 2005. 13, 43, 54

[39] D. Opitz and R. Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198, 1999. 20

[40] DB Parker. Learning-logic (tr-47). *Center for Computational Research in Economics and Management Science. MIT-Press, Cambridge, Mass*, 1985. 8

[41] D. Partridge and W.B. Yates. Engineering multiversion neural-net systems. *Neural Computation*, 8(4):869–893, 1996. 20

[42] M.P. Perrone and L.N. Cooper. When networks disagree: Ensemble methods for hybrid neural networks. Technical report, DTIC Document, 1992. 24, 25

[43] Donald A Pierre. *Optimization theory with applications*. Courier Dover Publications, 1969. 10

[44] R. Ratcliff. Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions. *Psychological Review*, 97(2):285, 1990. 1, 13

[45] A. Robins. Catastrophic forgetting in neural networks: the role of rehearsal mechanisms. In *Artificial Neural Networks and Expert Systems, 1993. Proceedings., First New Zealand International Two-Stream Conference on*, page 65–68, 1993. 2, 13, 14

[46] A. Robins and University of Otago. Artificial Intelligence Laboratory. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146, 1995. 15

[47] F Ronsenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65:386–408, 1958. 3

[48] Frank Rosenblatt. Principles of neurodynamics. 1962. 5

[49] DE Rumelhart, GE Hinton, RJ Williams, DE Rumelhart, and JL McClelland. Learning internal representations by error propagation, parallel distributed processing, 1986. *Cambridge, MA*, pages 318–362, 1986. 8

[50] Gilbert Strang and Kaija Aarikka. *Introduction to applied mathematics*, volume 16. Wellesley-Cambridge Press Wellesley, MA, 1986. 10

[51] N. Ueda and R. Nakano. Generalization error of ensemble estimators. In *Neural Networks, 1996., IEEE International Conference on*, volume 1, page 90–95, 1996. 19, 23

[52] PJ Werbos. Beyond regression: new tools for prediction and analysis in the behavioral sciences. 1974. *Harvard University*, 1974. 8

[53] Bernard Widrow, Marcian E Hoff, et al. Adaptive switching circuits. 1960. 5

[54] W.B. Yates and D. Partridge. Use of methodological diversity to improve neural network generalisation. *Neural Computing & Applications*, 4(2):114–128, 1996. 20

Vita

Robert Coop was born in Nashville, TN, to parents Thomas and Michelle Coop. He has an older sister, Kathy. Robert attended Sumner Academy in Gallatin, TN and later graduated from Beech Senior High School in Hendersonville, TN in May of 2003. He enrolled in the computer science program at the University of Tennessee in August of 2003. In the spring of 2005, Robert withdrew from the university, took a semester off, changed plans, and came back for the summer semester with the intent to major in philosophy. Two semesters later, he had decided once again to major in computer science, and in the spring of 2008 he received a Bachelors of Science from the university, majoring in computer science with a minor in philosophy.

After meeting Dr. Itamar Arel during his last semester of undergraduate work, Robert enrolled in the graduate school at the University of Tennessee for the 2008 fall semester, accepting a graduate teaching assistantship in the Engineering Fundamentals department. He had this assistantship through the Engineering Fundamentals department for two years, then for the Computer Science department for a year until the assistantship was ended in the spring of 2011 due to budget cuts. Robert also worked with Dr. James Nutaro at the Oak Ridge National Laboratory under the Higher Education Research Experiences (HERE) and the Advanced Short-Term Research Opportunity (ASTRO) programs for four years.

On May 23rd, 2009, Robert Coop and Amanda Allen got married in their home town of Goodlettsville, TN. In the summer of 2010, Robert received a Masters of Science in Electrical Engineering, with a concentration in Intelligent Control. Robert Coop graduated with a Doctorate of Philosophy in Computer Engineering with a concentration in Machine Learning in the spring of 2013.