UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

A NEW APPROACH ADAPTING NEURAL NETWORK CLASSIFIERS TO SUDDEN CHANGES IN NONSTATIONARY ENVIRONMENTS

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

MASTER OF SCIENCE

By

ALEXANDRA LYNN AMIDON
Norman, Oklahoma
2017

A NEW APPROACH ADAPTING NEURAL NETWORK CLASSIFIERS TO SUDDEN CHANGES IN NONSTATIONARY ENVIRONMENTS

A THESIS APPROVED FOR THE
GALLOGLY COLLEGE OF ENGINEERING

BY

_____
Dr. Charles Nicholson, Chair

_____
Dr. Randa Shehab

_____
Dr. Ziho Kang

To my parents and grandparents,
Who have empowered me to pursue my dreams,
Whatever they may be.

Acknowledgements

I would like to thank and acknowledge my thesis advisor, Dr. Charles Nicholson of the School of Industrial and Systems Engineering at the University of Oklahoma for his support and feedback on this work. I would also like to thank Nerd Kingdom for funding and motivating this research – I hope this work can contribute to the algorithms driving the *The Untitled Game*. I would also like to thank my father-in-law, Mark Amidon, for reading and providing final edits – numerous hidden typos and errors were fixed thanks to your careful eye. Finally, thank you to my husband, Michael Amidon, for being with me every step of the way of this journey.

Table of Contents

List of Tables

List of Figures

Abstract

Business are increasingly analyzing streaming data in real time to achieve business objectives such as monetization or quality control. The predictive algorithms applied to streaming data sources are often trained sequentially by updating the model weights after each new data point arrives. When disruptions or changes in the data generating process occur, the online learning process allows the algorithm to slowly learn the changes; however, there may be a period of time after concept drift during which the predictive algorithm underperforms. This thesis introduces a method that makes online neural network classifiers more resilient to these concept drifts by utilizing data about concept drift to update neural network parameters.

Chapter 1: Introduction

Technology that generates data in a continuous, "streaming" fashion, such as smart phones, internet-of-things devices, networks of sensors, and internet applications and games has proliferated greatly in recent years (Ditzler, et al. 2015). Such sources of streaming data are often mined and analyzed in real time to achieve business objectives such as monetization or quality control.

Predictive algorithms applied to streaming data sources are often trained sequentially ("online") by updating the model weights after each new data point arrives. This allows a model to reflect the characteristics of the most recent data points.

A common assumption, particularly when data does not arrive sequentially, is that the process generating the data does not change; that is, the characteristics of the data are fixed. This assumption is often false, as human habits or patterns often change or processes are disrupted by external factors. These disruptions or changes are known as "concept drift" and change the underlying characteristics of the generated data. Although the online learning process allows the model to eventually follow changes in the data generating process, there may be a period of time after concept drift during which the predictive algorithm underperforms. Aspects of underperformance include the time to recover from drift, the total "systemic impact", and the drop in performance after drift.

In this thesis I demonstrate that information about concept drift can be used to reduce the negative performance impact of concept drift on learning algorithms, such as online neural networks. I develop and test three methods that use this concept drift information to update the neural network parameters. The three proposed methods are

1) reset a given percentage of randomly selected weights to a random value; 2) rescale all weights between the existing value and a random value – the rescaling depends on the characteristics of the concept drift; 3) reset all weights based on a Bayesian-inspired formula. I test these three update methods on four simulated datasets that simulate concept drift at assigned intervals. The results demonstrate that the networks using the rescale methods perform better than an online network with no update method at all. Thus, there is value in the information about concept drift that can be used to aid online learning algorithms. Further, the methods are most effective in cases where concept drift is not small. As an added benefit, all three methods have low computational costs and memory requirements.

This thesis is structured as follows. Chapter 2 contains background on related subjects that motivated this work, including data streams, online learning, concept drift, neural networks, and describes the related work and the contribution of this thesis. In Chapter 3, I present the methodology, which includes the description of the proposed methods, simulated data sets, and experimental design. In Chapter 4, I describe how results are evaluated, the results themselves, and discuss their significance. The work is concluded in Chapter 5.

Chapter 2: Background

Machine learning is an area of applied statistics that emphasizes the use of computers to statistically estimate complicated functions. A machine learning algorithm *learns*,[1] or is *trained* to perform some task if it can use past experience (data) to adjust a set of parameters in order to optimize some measure of performance or cost (Goodfellow, Bengio and Courville 2016; Murata 1998; Engelbrecht 2007). This trained algorithm is a *model* of the task. In general, a predictive model is a function $f: X \rightarrow y$ that maps the input feature space X to a corresponding output target space y (Gama, Zliobaite, et al. 2014). Machine learning can train a predictive model for the *classification* task, where the output target space y is a class label. Other tasks that machine learning can solve include regression, transcription, translation, and anomaly detection (Goodfellow, Bengio and Courville 2016).

This chapter provides a background for key concepts in the areas of data streams, online learning, concept drift, and neural networks. This background is followed by related work and the contribution of this thesis.

Chapter 2.1: Data Streams and Online Learning

A data stream is an ordered sequence of m-dimensional points $X_1, X_2, \ldots, X_n$ that may contain time stamps. The points generally must be accessed in order and can be read only once or a limited number of times in the prescribed sequence (Henzinger, Raghavan and Rajagopalan 1998; Guha, et al. 2003; Webb, Hyde, et al. 2016). This

[1] Another definition of learning: an algorithm learns by optimizing its parameter set with respect to examples of the underlying rule that it is learning (Saad 1998).

sequential-temporal property distinguishes data streams from non-stream data (Bifet, Read, et al. 2013).

The data stream framework is important because sources of streaming data are becoming increasingly common. Further, many types of data can be modeled as data streams, such as data sets that are too large to fit in main memory (Guha, et al. 2003) or data that arrive at sequential points in time. Examples include the growing number of monitoring systems that generate data at high rates, such as systems that track climate variables, physical systems, computer network traffic, or wearable and household devices that are often called the Internet of Things (Balzanella, Rivoli and Verde 2013; Ellis 2014). Other sources of data streams include social media activity and ecommerce websites (Ellis 2014). The possible uses of these streams are extensive.

There are three key challenges commonly associated with data streams analysis: volume, velocity, and concept drift. First, data streams are unbounded: there is no bounded time interval during which the stream produces data and hence no corresponding limit to the volume of data produced (Balzanella, Rivoli and Verde 2013; Mena-Torres and Aguilar-Ruiz 2014). Due to this volume, it may be difficult to store all the data in a database to interact with it as needed (Rajaraman, Ullman and Leskovec 2014) so the data may be discarded or archived and no longer be accessible for processing (Balzanella, Rivoli and Verde 2013). For this reason, many data stream algorithms seek to summarize the data stream so as to store the core signal in a reduced amount of space. Second, the velocity of a data stream, that is, the rate at which data enters the system, may be faster than an analytical method can update in real-time. Third, data streams are often generated by nonstationary processes, that is, processes

whose characteristics may change over time. A change in the data generating process is often called "concept drift", which will be formally defined in Chapter 2.2.1. Accurately representing changing environments requires analytical processes that adapt quickly to new emerging concepts (Balzanella, Rivoli and Verde 2013).

To deal with the constraints of the data stream model, data stream algorithms make two important assumptions. First, data stream algorithms often assume that there is limited space for computation (Guha, et al. 2003). They seek to use each data point only a few times and require a workspace that is smaller than the size of the input (Henzinger, Raghavan and Rajagopalan 1998). Second, data stream algorithms often require decisions to be made before all the data are available (Guha, et al. 2003).

Batch, or "offline" learning is a traditional modeling approach that first trains a model on the entire dataset, possibly in batches, and then applies the model to real-time data or other applications (Ellis 2014). Figure 1 illustrates this general framework. In a typical supervised learning model, a dataset is first preprocessed, a step that may include data transformations and variable selection. Next, the data is partitioned into training and test datasets, typically via a randomized split. A model is then trained using only the training data. The performance and generalizability of the trained model is subsequently evaluated using the set of test data, which the model has not seen before. In some cases, the parameters of the model will be tuned and the model will be retrained on the training data in order to optimize the model performance (Kuhn and Johnson 2015). Other steps, such as cross-validation, may be added to this process.

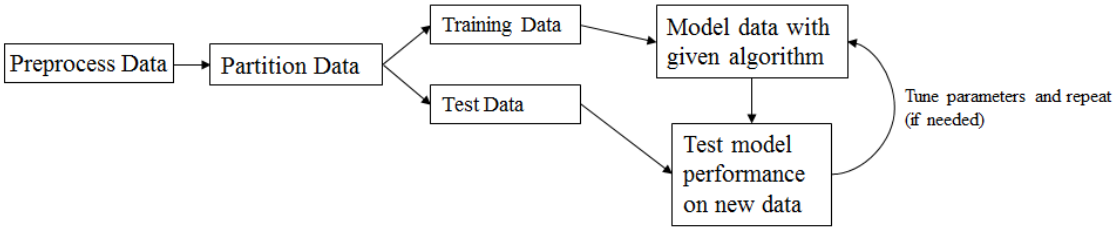

Figure 1: Supervised (batch) Machine Learning Paradigm

Batch learning algorithms generally violate the constraints and assumptions of the data stream model. Batch learning algorithms assume that there are no time constraints in updating a model in response to new information. This is seen in how models are updated with new observations: batch learning systems must rebuild a trained model from scratch using the entire training set, including old observations (Esposito, et al. 2004; Pérez-Sánchez, Fontenla-Romero and Guijarro-Berdiñas, 2016). Further, they frequently make several passes over the data during training, not limiting the number of times a data point is read (Mena-Torres and Aguilar-Ruiz 2014). The model can only be used for prediction once training is completed (Gama, Zliobaite, et al. 2014; Pérez-Sánchez, Fontenla-Romero and Guijarro-Berdiñas, 2016).

Online, or incremental learning algorithms incrementally train or update a model in a sequential manner (Ellis 2014; Gama, Zliobaite, et al. 2014). In terms of the data stream framework, the model is continuously updated when each new data point arrives, alternating between observing new data and modifying model parameters (Ellis 2014) (Pérez-Sánchez, Fontenla-Romero and Guijarro-Berdiñas, 2016; Murata, Kawanabe, et al. 1998). Variations of this online process include training a model incrementally by continuously updating the model when each new observation arrives or retraining the model using the most recent batches of observations (Gama, Zliobaite, et al. 2014). As illustrated by Figure 2, online learning follows a simple procedure: (1) initialize the

model; (2) predict the output y_t using input data x_t; (3) diagnose the model accuracy when the true value of y_t has been received; and, (4) update the model with the new information (Gama, Zliobaite, et al. 2014). In this way, incremental/online techniques can refine or update a model without retraining it from scratch (Esposito, et al. 2004). Online learning algorithms also offer the advantage of low computational cost because all training examples do not need to be stored in memory (Murata 1998). In sum, online learning algorithms meet the constraints of the data stream framework and are well-suited to analyzing and predicting data streams.

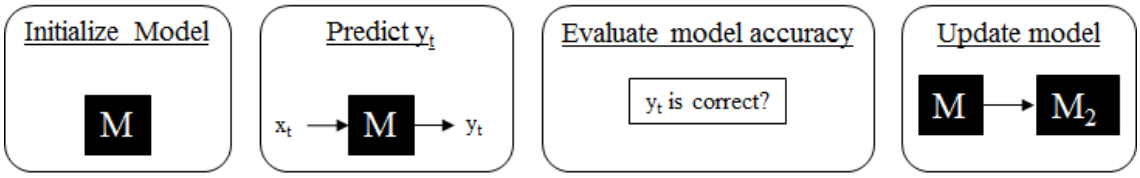

Figure 2: Elements of the Online Machine Learning Paradigm

An implicit assumption of batch learning is that the process generating the data stream is stationary (Ditzler, et al. 2015). That is, there are no changes in the distribution of the data and the expected model output. The model trained at one time point will be equally valid at all future time points. This assumption of stationarity does not hold for many real-world tasks. Many real-world data streams are generated by nonstationary processes that are represented by continuous flow of new information that affects the trained model (Balzanella, Rivoli and Verde 2013; Esposito, et al. 2004). The online learning paradigm, on the other hand, assumes that the information gained at any given moment is incomplete and thus that any learned theory is potentially susceptible of changes (Esposito, et al. 2004). This is particularly true when the

environment is not stationary. There is no expectation that a model trained at one time point is valid at all points in the future. Because online learning algorithms reference each training example only once, newer examples have more influence on the model parameters. This feature functions as a forgetting effect that allows the online learning algorithm to follow gradual changes in the environment (Murata 1998).

Chapter 2.2: Concept Drift

Dynamic and rapidly changing environments, where incremental learning is most suitable, are becoming increasingly common (Balzanella, Rivoli and Verde 2013; Ellis 2014). A common scenario of a changing environment is in industrial applications, where the wear and tear of machines causes data distributions to change gradually over time (Murata, Kawanabe, et al. 1998). The spam (unwanted email) detection problem is an example of a dynamically changing environment. The distinction between the spam and non-spam classes may evolve over time as spammers become more sophisticated or use new tactics. Indeed, spammers actively seek to evade spam filters. Further, as spam is user-specific, user preferences about what constitutes spam may also change over time (Kuncheva 2004). Another rapidly changing environment is user modeling and associated recommendation systems. Such systems are dynamic because the attributes that characterize a user and their interests in products and services are likely to change over time (Ditzler, et al. 2015; Webb, Pazzani and Billsus 2001).

Because the real world is often dynamic and nonstationary, it is reasonable to assume that data representing dynamic real world environments are also nonstationary and unpredictable, changing over time as the real world changes. Machine learning

algorithms should contain mechanisms for detecting and handling the complex, changing phenomena that the models aim to capture (Webb, Hyde, et al. 2016; Gama, Zliobaite, et al. 2014). In terms of the classification task, it follows that if the process generating the data changes over time, the target function to be predicted may also change (Gama, Zliobaite, et al. 2014).

Traditionally, most machine learning algorithms operate in batch mode. The result is one or more static models that represent the state of the environment at the time the model was generated. Such models are insufficient in nonstationary environments because they fail to adequately incorporate the most recent information about the environment. Nonstationary environments however can be handled using online learning algorithms (Webb, Hyde, et al. 2016).

In a nonstationary environment, "concept drift" occurs when the characteristics of a data stream change. Concept drift is often framed in the context of data streams; however, the framework is applicable to any context in which a model may be learned from historical data and applied to future data (Webb, Hyde, et al. 2016).

Chapter 2.2.1: Definition of Concept Drift

The general framework for concept drift is as follows. Let f be the data generating process that produces a sequence of tuples (X_t, y_t) at time t, where X_t is a vector of inputs and y_t is the true class label. Let m be the model representing the data generating process f. The true underlying probability distribution of data produced by f at time t, $P_t(X, y)$, is unknown. The classification problem can be described in terms of Bayesian Decision Theory (Duda, Hart and Stork 2012), where classification decisions

are made in terms of the posterior probabilities of the classes. Given the prior probabilities of the classes $P(y)$ and the class conditional probability density functions $P(X|y)$ the classification decision can be made according to $P(y|X) = \frac{P(X|y)P(y)}{P(X)}$ (Gama, Zliobaite, et al. 2014). This probability, $P(y|X)$, is the concept function that m represents.

The concept at time t is the conditional probability that input data X is assigned to target class y, $P_t(y|X)$ (Webb, Hyde, et al. 2016; Gama, Zliobaite, et al. 2014). In Figure 3 (Gama, Zliobaite, et al. 2014), the delineation between the green and red classes represents the "concept". The goal of a model is to reproduce the concept, enabling it to correctly identify the target class of any input data point.

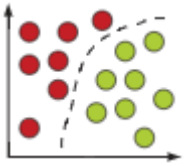

Figure 3: Graphical representation of a concept

Concept "drift" occurs when the description of a concept is disrupted by some change that requires the definition to be revised (Schlimmer and Granger 1986). In terms of modeling, if the data generating function f has changed, the model m describing the function should also change (Martínez-Rego, Pérez-Sánchez, et al. 2011). Mathematically, concept drift occurs between times *t* and u if the probability distributions change: $P_t(y|X) \neq P_u(y|X)$ (Webb, Hyde, et al. 2016).

There are two types of concept drift – "real concept drift" and "virtual concept drift." Real concept drift occurs when $P(y|X)$ changes; it requires model m to be updated in order to maintain accuracy. This change can occur with or without changes

in the distribution of the input data, $P(X)$. Virtual concept drift occurs when distribution of the input data, $P(X)$, changes over time but $P(y|X)$ does not. This type of drift does not affect the model's description of the target concept (Webb, Hyde, et al. 2016; Gama, Zliobaite, et al. 2014). A typical example of real concept drift is the changing interests of a user following an online news stream. If the distribution of the incoming news remains constant but the conditional distribution of the users' preference for "interesting" news documents changes, then the target concept (articles that the user will find interesting) has changed. In the same vein, virtual concept drift would occur if the distribution of types of news documents in the stream changes; users would still have the same preferences, even if the variety of documents available to choose from are different (Gama, Zliobaite, et al. 2014).

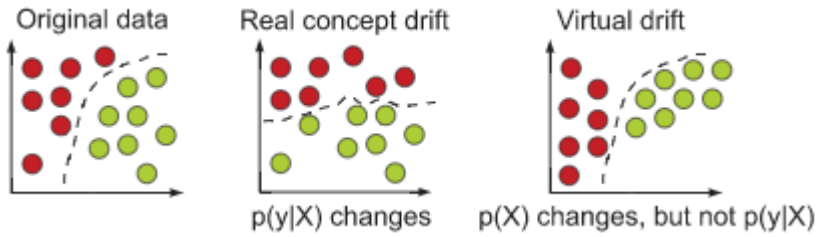

Figure 4: Real and virtual concept drift

Figure 4 (Gama, Zliobaite, et al. 2014) illustrates the difference between real concept drift and virtual concept drift. The *concept* is the distinction between the red and green points. Under real concept drift, the distinction between the red and green points changes. After virtual drift, the distribution of red and green points changes but the previous distinction between classes remains valid. As the illustration suggests, the previous decision model only becomes obsolete under real concept drift (Gama, Zliobaite, et al. 2014). Absent updates to the model after real concept drift, the model

will no longer correctly describe the full target concept space; as a result, the model's ability to correctly classify data under the new concept will decrease.

This thesis will refer to real concept drift as simply "concept drift" or "drift."

Chapter 2.2.2: Quantifying Concept Drift

The magnitude of drift between times t and u can be generally defined as the distance function $D(t, u)$ (Webb, Hyde, et al. 2016). A frequently used metric for drift magnitude is $\frac{\#\, misclassified}{\#\, examples\ after\ drift}$, the percentage of input space that have a different class label when the change (drift) from concept f at time t to g at time u is complete (Kosina, Gama and Sebastiao 2010; Minku, White and Yao 2010; Chen, Koh and Riddle 2015). Note that this metric mainly reflects changes in $P(y)$ and $P(y|X)$. It poorly reflects any changes in $P(X)$ or $P(X|y)$ (Minku, White and Yao 2010). Webb, Hyde, et al. (2016) argues that the appropriate metric for measuring drift may vary by domain; this definition is selected because it is generalizable and easy to apply to many domains.

The severity of drift can vary widely. In the extreme, "severe" drift occurs when all examples are misclassified under the new target concept. Most drift is "intersected" drift, where part of the input space has the same target class in both the old and new concepts (Minku, White and Yao 2010). One could specify a threshold for "major" v "minor" drift, such that if the drift magnitude is less than the threshold, the model simply requires updating; if the drift magnitude is greater than the threshold, the drift is "severe" and the model ought to be abandoned in favor of a new model (Webb, Hyde, et al. 2016).

Cases in which the drift is "severe" can reasonably be assumed to be rare and the causes of such drift are likely apparent without the use of drift detection. As such, this study assumes that all drift encountered is "minor" and only requires updates to the model.

Chapter 2.2.3: Drift Duration

A stream is composed of discrete periods of time during which there are stable concepts, possibly interspersed by periods of instability, where the concept changes. Concept stability can be defined as any interval between time steps *t* and *u* such that $D(t,u) < \theta$, where θ is some minimum threshold for stability (Webb, Hyde, et al. 2016). In this work, unstable concepts that do not account for the transition from one concept to another are assumed to be noise; a goal of an online learning algorithm is to account for such instability.

The speed, or duration, of concept drift is the number of time steps for a new concept to completely replace an old concept (Minku, White and Yao 2010). A drift with a short drift duration is known as an *abrupt drift*. Abrupt drift occurs when a stream generated by concept f is suddenly replaced by a new generating function g. The abrupt drift model assumes that concept drift occurs over discrete periods of time that is bounded before and after by periods without drift (Webb, Hyde, et al. 2016). An example of abrupt drift might be a market crash, where the market value suddenly drops significantly.

Slow drift implies a long drift duration, also known as *incremental*, *continuous*, or *extended* drift. In this case, the change is a steady progression from concept f to

concept g such that at each time step $t + i$, the distance from the old concept f, $D(t, t + i)$, increases and the distance to the new concept g, $D(t + i, u)$, decreases (Webb, Hyde, et al. 2016). When incremental drift occurs, there may be several intermediate concepts between concepts f and g (Minku, White and Yao 2010). An example of incremental drift is a recession, where there are many intermediate concepts between the market peak (concept f) and the bottom of the recession (concept g).

Figure 5 presents an example of incremental drift that occurs over a period of 100 time steps. The functions $v_1(t)$ and $v_2(t)$ model the probability that an example from the old and new concepts, respectively, will be presented at time t. As illustrated, the speed of drift can be represented as the change in $v_2(t)$, the probability that a sample from the new concept g will be presented (Minku, White and Yao 2010).

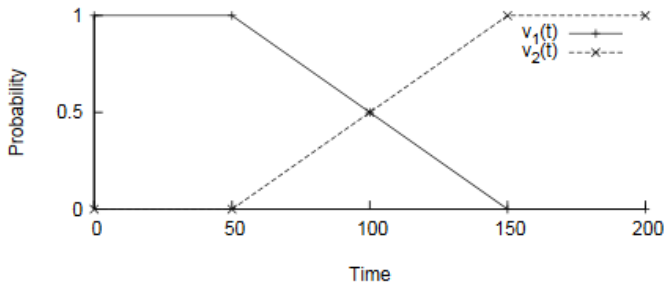

Figure 5: Illustration of incremental drift

Drift recurrence occurs when the "new" concept has previously appeared in the data stream (Webb, Hyde, et al. 2016). Old concepts may reappear in a specific order, such as weather patterns (cyclical drift), or may be unordered, such as the market basket analysis problem (Webb, Hyde, et al. 2016; Minku, White and Yao 2010). In the market basket analysis problem, there is concept drift when a new product is introduced to the market. The data may return to the previous concept if consumers stop purchasing the new product and resume their past purchasing patterns (Minku, White and Yao 2010).

Chapter 2.2.4: Drift Detection

Detecting drift quickly and providing a reasonable measure of drift magnitude is a challenging task. For example, if the label for data X at time t_1 is y_1 but at t_2 the correct label is y_2, does this indicate concept drift or just noisy data? (Bach and Maloof 2010).

Various algorithms have been proposed to address the drift detection problem. Work in drift detection generally aims to efficiently identify the true points of concept drift with accuracy while also minimizing the drift detection time (Chen, Koh and Riddle 2015). A simple approach by Nishida and Yamauchi (2007) uses a statistical test based on prediction errors to detect concept drift in an online classifier. There are other adaptive test statistics for drift detection, as evaluated in Dries and Ruckert (2009). The DDM algorithm detects drift when drift causes the mean classification error to significantly increase; this increase is defined by the difference between the current cumulative mean and standard deviation and the minimum mean and standard deviation (Gama, Medas, et al. 2004; Chen, Koh and Riddle 2015). The EDDM algorithm for drift detection produces results similar to DDM but is designed to work well in the presence of slow, incremental change (Baena-Garcia, et al. 2006). ADWIN2 detects drift by using a Hoeffding bound and an adaptive windowing technique that stores classification errors in an exponential histogram data structure (Bifet and Gavalda 2007; Chen, Koh and Riddle 2015). SEED uses the same Hoeffding bound as ADWIN2 but uses a different data structure to store classification errors and uses a compression algorithm to reduce the number of boundary checks (Huang, et al. 2014; Chen, Koh and

Riddle 2015). The MagSeed algorithm can detect both drift and the drift magnitude (Chen, Koh and Riddle 2015).

The detection of concept drift is outside the scope of this research. This thesis assumes that any application of the proposal makes use of a drift detection algorithm that detects drift quickly and provides an accurate measurement of drift magnitude.

Chapter 2.2.5: Other Considerations

Modeling nonstationary environments requires that the past learned experience be replaced with new information that represents the current concept (Martínez-Rego, Pérez-Sánchez, et al. 2011). To do so, a learning system must balance *stability*, the ability to retain significant knowledge about the environment, and *plasticity*, the ability to update in response to new information by overwriting old concepts with new concepts. A learning system must be both plastic in response to new significant events and stable in response to noisy training inputs. Ideally, the system considers new samples to be more important than old samples to model the current concept. Because both requirements are desirable but in direct conflict, the challenge is known as the *stability-plasticity dilemma* (Grossberg 1987).

Other desirable features that enable models in nonstationary environments to respond quickly to concept drift include the ability to: (i) automatically respond to drift without explicit detection (Widmer and Kubat 1996), (ii) respond to changes in the environment (Schlimmer and Granger 1986), (iii) adjust model in response to new concept (Widmer and Kubat 1996), (iv) distinguish between genuine change in the

underlying function and randomness (Schlimmer and Granger 1986), and (v) use previous learning to handle concepts that reappear (Widmer and Kubat 1996).

Chapter 2.2.6: How I Will Treat Drift

In this work, I assume that all concept drift is abrupt. To start, the proposed method, described in Chapters 2.5 and 3.1, requires that concept drift first be detected and assumes the use of a drift detection algorithm. It is difficult to detect drift in general, much less gradual drift. Few drift-detection methods are capable of detecting extended drift or concept instability. Further, it is impossible for a drift detection algorithm to detect whether a period of incremental drift has finished because the probability distribution of unseen observations is inherently unknown. So even if a method could handle incremental drift or unstable concepts, it would not know to do so.

The assumption of abrupt drift is reasonable within the concept drift framework. Periods of incremental drift can be considered several distinct concepts. Each intermediate concept in a period of drift is a "new" concept, between which there is no stable period. Periods of incremental drift or concept instability can also be thought of as periods with increased noise. Non-trivially, the very purpose of online learning is to update the model in response to small or incremental changes.

Chapter 2.3: Neural Networks

In this section, neural networks are described and defined in detail. Neural networks are a powerful class of nonlinear machine learning algorithms that can model a wide variety of tasks, including regression, classification, speech recognition, and image processing (Saad and Rattray 1998; Engelbrecht 2007). Further, neural networks can be trained online and are therefore a candidate model for handling concept drift in data streams.

Chapter 2.3.1: Neural Network Overview

Artificial neural networks are a diverse class of machine learning algorithms that are inspired by and modeled on the architecture of biological neurons in a brain (Goodfellow, Bengio and Courville 2016; Zou, Han and So 2009). The inspiration is drawn from the idea that the brain operates like a complex, nonlinear, and parallel computer that can perform difficult tasks, such as speech recognition and image processing, faster and more accurately than any computer (Engelbrecht 2007). The brain consists of a large network of interconnected nerve cells ("neurons") that receive and transmit signals from neighboring neurons (Engelbrecht 2007; Zou, Han and So 2009). Each neuron is a simple processing unit that performs a simple task (Jain, et al. 2014; Zou, Han and So 2009). When the total signal that a neuron receives is greater than its given threshold, the neuron emits an electrochemical signal to neighboring neurons, which may in turn also propagate signals to further neurons (Zou, Han and So 2009). A neuron can amplify or reduce the strength of a signal (Engelbrecht 2007). Like the biological network, an artificial neural network (or simply neural network) is a

network of interconnected nodes that represent biological neurons (Zou, Han and So 2009).

Generally, neural networks are directed acyclic graphs that describe how a series of functions, represented by artificial neurons, interact in a connected chain to complete some task (Goodfellow, Bengio and Courville 2016). Neural networks are often modeled as layered networks of artificial neurons; each neuron receives multiple weighted inputs from neurons in other layers (Engelbrecht 2007; Zou, Han and So 2009). If the weighted sum of inputs is greater than the neuron's threshold, then the neuron is activated and it passes the signal through an activation function to neighboring neurons in the network (Zou, Han and So 2009).

A popular neural network architecture or design is the feedforward multilayer perceptron topology, illustrated in Figure 6 (Saad 1998; Jain, et al. 2014; Zou, Han and So 2009). In feedforward networks, input information flows through the network function in a single direction to produce some target output (Goodfellow, Bengio and Courville 2016). In other architectures, such as recurrent neural networks, neurons may have feedback connections to previous layers, through which information may be sent back into nodes in previous layers of the network (Saad and Rattray 1998; Engelbrecht 2007; Goodfellow, Bengio and Courville 2016). In the multilayer perceptron topology, neurons are arranged in layers, which typically include an input layer that contains one neuron per input variable, one or more "hidden" layers, and an output layer that contains one neuron for each possible output (Jain, et al. 2014; Saad 1998). Each layer must contain at least one neuron (Jain, et al. 2014). The number of nodes in each layer is typically set intuitively and adjusted manually after several training iterations (Zou,

Han and So 2009). Figure 6 (Jain, et al. 2014) illustrates a feedforward multilayer perceptron neural network with three input neurons, four hidden neurons, and three output neurons. The circles represent neurons in the network.

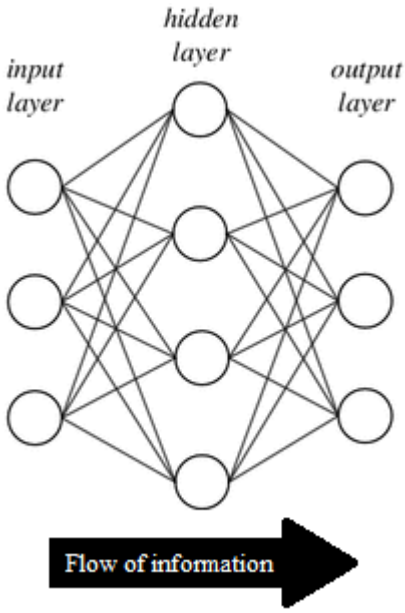

Figure 6: Feedforward Neural Network

Like biological neurons, artificial neurons receive signals from the environment or other neurons, process the input, and, if activated, transmit a signal to all connected neurons. Alone, a single neuron can be used to model linearly separable functions with zero error (Engelbrecht 2007). Figure 7 (Engelbrecht 2007) illustrates how information flows through a neuron.

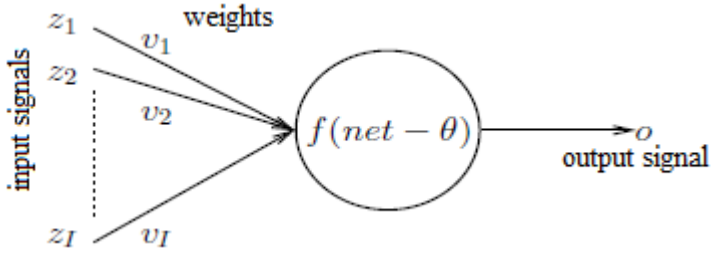

Figure 7: Flow of information through a neuron

A neuron receives a vector of I input signals: $\boldsymbol{z} = (z_1, z_2, \ldots, z_I)$ (Engelbrecht 2007; Zou, Han and So 2009). Each input signal z_i is associated with a weight, v_i, that excites or inhibits the input signal (Engelbrecht 2007). The neuron computes the net input signal as a function of the input and its respective weights:

$$net = \sum_{i=1}^{I} z_i v_i$$

(Engelbrecht 2007; Zou, Han and So 2009). The neuron then applies an activation function f to the net input signal and neuron's bias θ (threshold) to compute the output signal

$$o = f(net - \theta).$$

Alternatively, the threshold θ can also be represented as

$$\theta = z_{I+1} v_{I+1},$$

where $z_{I+1} = -1$. Correspondingly, the net input signal is calculated as

$$net = \sum_{i=1}^{I+1} z_i v_i$$

and output function is denoted as

$$o = f(net)$$

(Engelbrecht 2007). Figure 8 (Engelbrecht 2007) illustrates the flow of information through a neuron with the alternative formulation.

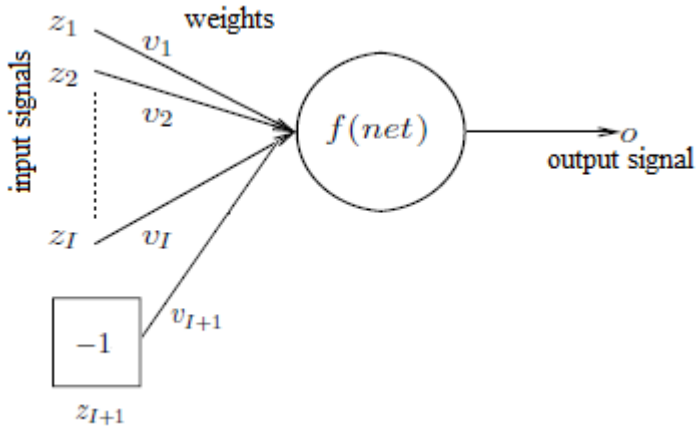

Figure 8: Flow of information through a neuron

The activation function f controls whether the neuron fires and the strength of the signal released. Generally, activation functions are monotonically increasing functions that produce outputs that range from 0 to 1. There are a variety of activation functions, including the sigmoid function, softmax function, linear function, step function, ramp function, hyperbolic tangent function, and the Gaussian function.[2] The sigmoid function is used in this study because of its wide usage. The sigmoid function, depicted in Figure 9, is calculated as

$$f(net - \theta) = \frac{1}{1 + e^{-\lambda(net-\theta)}},$$

where λ controls the steepness (usually, $\lambda = 1$) (Engelbrecht 2007). The sigmoid activation function approaches 0 when $(net - \theta)$ becomes very negative and approaches to 1 when $(net - \theta)$ becomes very positive (Goodfellow, Bengio and Courville 2016).

[2] Only the sigmoid and softmax activation functions are used in this work.

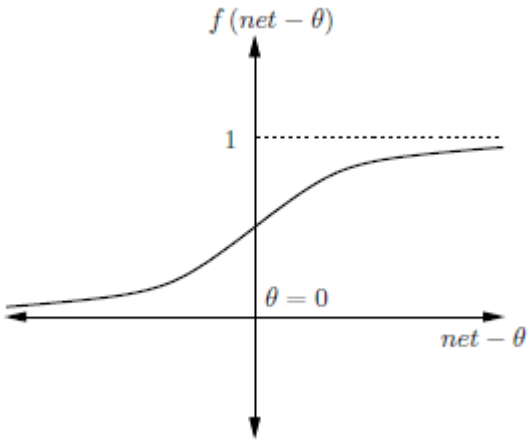

Figure 9: Sigmoid activation function

The softmax function is typically used as a classifier output in order to represent the probability distribution of *n* different classes. Softmax is calculated as

$$softmax(net-\theta)_k = \frac{\exp(net-\theta_k)}{\sum_j \exp(net-\theta_j)}$$

(Goodfellow, Bengio and Courville 2016).

Figure 10 illustrates a feedforward neural network with I neurons in the input layer $\boldsymbol{z} = (z_1, z_2, \dots, z_I)$, J neurons in the single hidden layer $\boldsymbol{y} = (y_1, y_2, \dots, y_j)$, and K neurons in the output layer $\boldsymbol{o} = (o_1, o_2, \dots, o_K)$. The $[-1]$ elements and their respective weights represent the biases of each neuron in the hidden and output layers (Engelbrecht 2007).

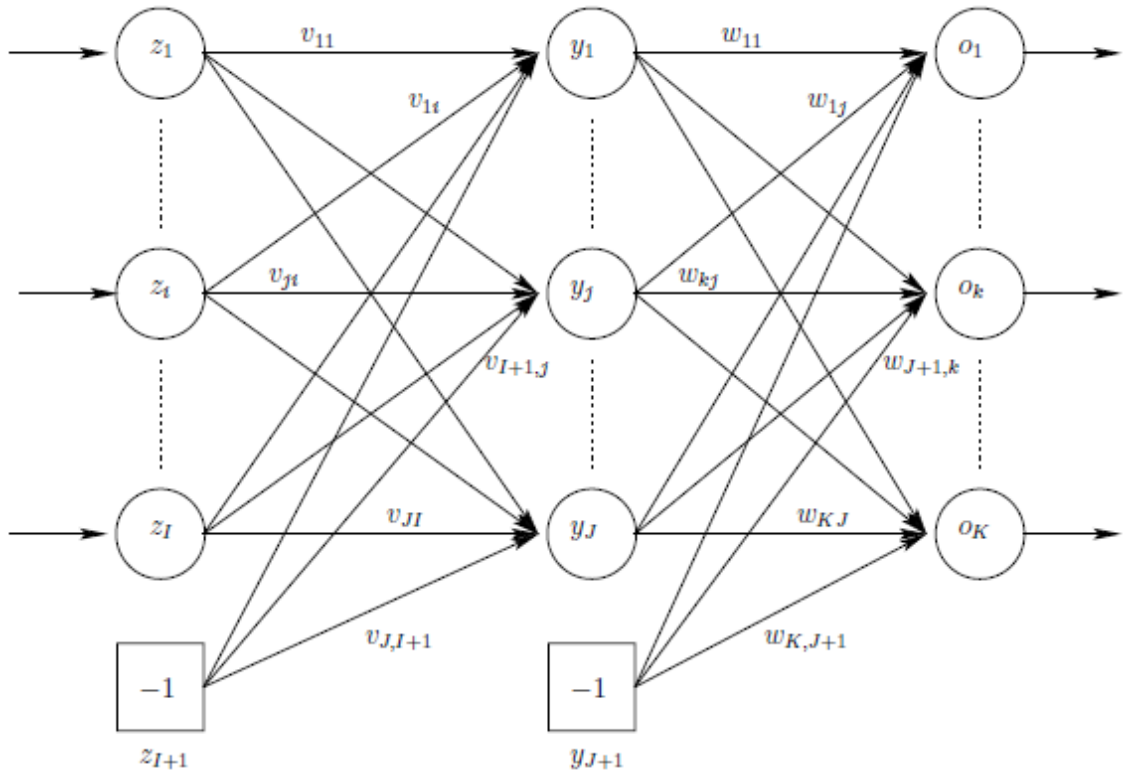

Figure 10: Single-layer feedforward neural network

The output of the network with any input $\boldsymbol{z}_p$ is calculated with a single forward pass through the network, as indicated by the arrows. The value of each output unit o_k for observation p is calculated as

$$o_{k,p} = f_{ok}\left(net_{ok,p}\right)$$

$$= f_{ok}\left(\sum_{j=1}^{J+1} w_{kj} f_{yj}\left(net_{yj,p}\right)\right)$$

$$= f_{ok}\left(\sum_{j=1}^{J+1} w_{kj} f_{yj}\left(\sum_{i=1}^{I+1} v_{ji} z_{i,p}\right)\right)$$

where f_{ok} and f_{yj} are activation functions for output unit o_k and hidden unit y_j; $w_{kj}$is the weight between the output unit o_k and hidden unit y_j; and $z_{i,p}$ is the value of input unit z_i of input pattern $\boldsymbol{z}_p$. The (I + 1)-th input unit and the (J + 1)-th hidden unit are

bias units that represent the threshold values of neurons in the next layer. Note that each activation function f can be a different function. Further, although the information from input neurons can be passed through an activation function, it is typically assumed that input units have linear activation functions (Engelbrecht 2007).

Neural networks are trained by modifying network parameters (weights and biases) to minimize some objective or error function that measures the model's ability to estimate the target function. The error function can be generically defined as $\varepsilon = f_e(\boldsymbol{w})$, where ε is the computed error, f_e is the error function, and $\boldsymbol{w}$ is the vector of weights and biases in the network. Minimizing this error function in neural networks is commonly achieved using *stochastic gradient descent* (Goodfellow, Bengio and Courville 2016; Engelbrecht 2007). Gradient descent works by calculating the gradient of the error function $\frac{\partial \varepsilon}{\partial \boldsymbol{w}} = f_e{}'(\boldsymbol{w})$ in the weight space and moving the vector of weights along the negative gradient. For $w_i \in \boldsymbol{w}$, the weight at time t is updated as $w_i(t) = w_i(t-1) - \eta \frac{\partial \varepsilon}{\partial \boldsymbol{w}}$, where η is the learning rate. The learning rate controls the rate at which vector moves along the negative slope of the gradient (Engelbrecht 2007). These calculations should bring the function implemented by the network is closer to the target function (Saad 1998). Figure 11 (Engelbrecht 2007) illustrates moving a weight vector containing a single weight along a gradient slope that corresponds to the network's error.

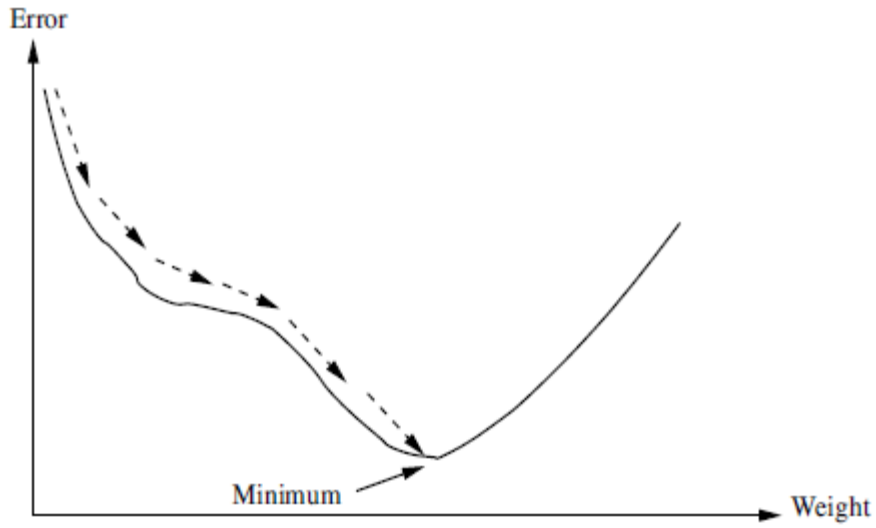

Figure 11: Illustration of gradient descent

Stochastic gradient descent with backpropagation has two phases for each epoch, or training iteration. First, the algorithm makes a *forward pass* by calculating the neural network's output $\boldsymbol{o}_p$ for each training input $\boldsymbol{z}_p$. Second, the error is calculated and the error signal is propagated back from the output layer through the hidden layer(s) of the network to the input layer. The weights are then adjusted as functions of the backpropagated error signal (Engelbrecht 2007).

The sum of squared errors (SSE) is often used as the error function for feedforward networks. For input vector $\boldsymbol{z}_p$, the SSE is calculated as

$$\varepsilon_p = \frac{1}{2}\left(\frac{\sum_{k=1}^{K}\left(t_{k,p} - o_{k,p}\right)^2}{K}\right),$$

where K is the number of output units and $t_{k,p}$ and $o_{k,p}$ are the target and calculated output values of k-th output unit, respectively. The value targets $t_{k,p}$ is given by the training example (Engelbrecht 2007).

The rest of the calculations for gradient descent using the SSE objective function and sigmoid activation function are formulated as follows[3] (the pattern subscript p will be omitted to simplify the notation). The value of the k-th output unit is calculated as

$$o_k = f_{ok,p}(net_{ok}) = \frac{1}{1 + e^{-net_{ok}}}.$$

The value of the j-th hidden unit is calculated as

$$y_j = f_{yj}(net_{yj}) = \frac{1}{1 + e^{-net_{yj}}}.$$

The changes in hidden-to-output weights are computed as

$$\Delta w_{kj,p}(t) = \eta\left(-\frac{\partial \varepsilon}{\partial w_{kj}}\right) = -\eta\delta_{ok}y_j,$$

where the output error to be backpropagated is calculated as

$$\delta_{ok} = -(t_k - o_k)(1 - o_k)o_k.$$

The changes in input-to-hidden weights are computed as

$$\Delta v_{ji,p}(t) = \eta\left(-\frac{\partial \varepsilon}{\partial v_{ji}}\right) = -\eta\delta_{yj}z_i,$$

where the hidden-layer error to be back propagated is calculated as

$$\delta_{yj} = \sum_{k=1}^{K} \delta_{ok}w_{kj}y_j(1 - y_j)$$

(Engelbrecht 2007).

The batch learning framework for backpropagation and stochastic gradient descent accumulates all weight changes and adjusts the weights only after all training patterns have been presented. So, given P_T patterns in the training set, the changes in the hidden-to-output weights are calculated as

[3] See (Engelbrecht 2007) for a complete derivation of these formulae.

$$\Delta w_{kj}(t) = \sum_{p=1}^{P_T} \Delta w_{kj,p}(t)$$

and the changes in the input-to-hidden weights are calculated as

$$\Delta v_{ji}(t) = \sum_{p=1}^{P_T} \Delta v_{ji,p}(t)$$

(Engelbrecht 2007). Thus, the weights at iteration $t + 1$ are adjusted as $w_{kj}(t+1) = w_{kj}(t) + \Delta w_{kj}(t)$ and $v_{ji}(t+1) = v_{ji}(t) + \Delta v_{ji}(t)$ (Jain2014).

Figure 12 (Engelbrecht 2007) outlines a generic implementation of the stochastic gradient descent (SGD) algorithm. The algorithm trains a neural network by iterating over sets of input-output data pairs, seeking to minimize the error function (Jain, et al. 2014). Typical stopping conditions for the SGD include: stop after a given number of epochs; stop when the error of the training set is small enough, that is, it converges to an acceptable level of error; stop when overfitting is observed (e.g. network is memorizing the training data).

Algorithm: Stochastic Gradient Descent Learning (batch)

Initialize weights, η, and the number of epochs $t = 0$;
while *stopping condition(s) not true* **do**
    Let ε_T = 0;
    for *each training pattern p* **do**
        Do the feedforward phase to calculate $y_{j,p}$ $(\forall\, j = 1, \dots, J)$ and
            $o_{k,p}$ $(\forall\, k = 1, \dots, K)$;
        Compute output error signals $\delta_{ok,p}$ and hidden layer error signals $\delta_{yj,p}$;
        Adjust weights w_{kj} and v_{jk} (backpropagation of errors);
        ε_T+= $[\varepsilon_p = \sum_{k=1}^{K}(t_{k,p} - o_{k,p})^2]$;
    end
    $t = t + 1$;
end

Figure 12: SGD Algorithm

Once the neural network has been trained via SGD, the model is applied to the desired application. Neural networks are usually not trained further after the training phase in order to preserve the learning. Further, a network trained in batch mode has no mechanism to update itself in response to new information. To incorporate new training examples, the neural network needs to be retrained with the all new and old data samples (Jain, et al. 2014).

Three key factors that impact the training and performance of neural networks are weight initialization, the learning rate, and network architecture. *Weight initialization* is important because gradient-based training methods like SGD are sensitive to how initial weights are set. If the weights are initialized close to a local minimum, the algorithm will converge quickly; if the weights are initialized on a flat area of the error surface, the algorithm will converge slowly. A common strategy for is to select small, randomized values centered around zero; this removes the bias toward any particular set of weights. The *learning rate* controls the size of the weight adjustments. Accordingly, the speed of the network's convergence is proportional to the learning rate. If the learning rate is very small, only small adjustments are made to the weights each epoch and more learning iterations are required to converge. A small learning rate allows the algorithm to closely follow the gradient path, but this may also cause it to become trapped in a bad local minimum. If the learning rate is very large, large weight adjustments are applied and the algorithm converge quickly; however, the algorithm may also oscillate without reaching minimum because step size is too large. A large learning rate could also lead the algorithm to skip a good local minimum and converge at a bad local minimum. There are numerous strategies for selecting an

appropriate learning rate, including the simple approach of selecting a small value and increasing or decreasing the learning rate manually according to how the network converges. *Network architecture*, or the number of layers and number of neurons in each layer, controls the complexity of functions that the neural network can learn. In general, a simple architecture containing the fewer neurons or layers is preferable to a complex architecture with equivalent performance because on average, the simplest network will generalize best. Further, a network with too many extra neurons may memorize the training patterns and noise in the training data, leading to bad generalization to other data (Engelbrecht 2007).

Chapter 2.3.2: Online Neural Networks

Training a neural network in batch mode, as described in the previous section, is a viable approach to modeling problems in stationary environments. However, this is not the case in nonstationary environments, where a model ought to be updated in response to new information as it appears. Online learning is an efficient, common, and powerful approach for training networks in nonstationary environments (Jain, et al. 2014).

To train a neural network incrementally (online), update the network weights after observing each sample, instead of accumulating and averaging the weight updates at the end of each epoch, (Jain, et al. 2014; Kuncheva 2004). The set of parameters w_t at time t are modified to w_{t+1} by using only the next example (x_{t+1}, y_{t+1}) given by the data generating process (Murata, Kawanabe, et al. 1998). Each iteration of online gradient descent uses a single example to update the parameter set, instead of averaging

the gradient over the complete training set (Bottou 1998). The formulae described in the previous section also apply to SGD in the online context.

It has been shown that online learning is asymptotically as effective as batch learning if the appropriate learning rate η is selected (Murata, Kawanabe, et al. 1998). Although online learning introduces some random noise into the learning, it is acceptable to assume that on average, random noise will not affect the algorithm's behavior (Bottou 1998).

There are a couple challenges in training neural networks. First, the training processes are dependent on the choice of training parameters, which impact the speed and convergence of the algorithm (Saad 1998; Saad and Rattray 1998). While training online with live data, it is difficult to retrain a network with a new set of training parameters. Second, SGD assumes that the error surface is fixed whereas the error surface in the online setting is inherently stochastic (Saad 1998). Third, the order that training examples are presented may introduce some bias to the model (Engelbrecht 2007), as the weights are more strongly influenced by the most recent observations. This may be beneficial, as it enables the network to most closely represent the current concept. Although the influence of old observations cannot be removed directly, reducing this influence incrementally over time allows the network to better handle nonstationary environments, where a concept may drift among a set of concepts (Mena-Torres and Aguilar-Ruiz 2014).

Chapter 2.4: Related Work

There are three general approaches to addressing concept drift: detect-and-retrain, constant updates, and ensembles (Kuncheva and Zliobaite 2009; Martínez-Rego, Pérez-Sánchez, et al. 2011). In the detect-and-retrain approach, when classification performance declines, the model is retrained using the new incoming data that represents the current distribution. A potential drawback of detect and retrain is that it can be computationally expensive to completely retrain a model on a high velocity and high volume data streams. Further, the time required to detect drift may allow the model to perform poorly for a while – this may be unacceptable for tasks with a high cost of failure. The detect-and-retrain approach is rarely mentioned in the context of neural networks because of the high retraining cost. Constant updates are commonly implemented via a moving window or constant updates to the model parameters (Kuncheva and Zliobaite 2009). With a moving window, the window size is critical: if it is too long, then the system is less responsive to changes and if it is too short, then the system is unstable and undertrained (Martínez-Rego, Pérez-Sánchez, et al. 2011). Another drawback of windowing is that it is expensive to retrain the model each time the window moves. The constant updates approach is computationally cheaper; however, it retains old information without considering its value or relevance (Kuncheva and Zliobaite 2009). Ensembles are a popular approach to concept drift that combine several models to obtain a solution. There are many ways to create, update, and manage the ensembles. A potential challenge of the ensemble approach is that it may require creating a new model periodically and must determine how to handle old models when they become irrelevant. Managing a large ensemble of models may make

the system slower in the adaption to fast-changing environments due to the storage cost and computation size (Martínez-Rego, Pérez-Sánchez, et al. 2011).

Chapter 2.4.1: Forgetting via Constant Updates and Instance Weighting

The goal of the constant updates approach to responding to nonstationary environments is to learn new class descriptions and unlearn old knowledge while avoiding the need to explicitly detect drift (Kuncheva 2004). Constant updates address the fact that it is not possible to directly "forget" the influence of old observations on the network weights (Elwell and Polikar 2009). The primary challenge of the approach is to select appropriate rate of forgetting so that it corresponds to the rate and type of change (Kuncheva 2004).

For other machine learning algorithms, a sliding window is a common implementation of forgetting (Kuncheva and Zliobaite 2009). The sliding window approach is limited by the requirement of storing the previous N data points. It is rarely used for neural networks because the retraining phase is computationally intensive (Elwell and Polikar 2009).

In neural networks, instance weighting is used to "forget" the influence of old observations by weighting new observations higher. Unlike the window approach, instance weighting methods do not need to maintain past batch data. Further, instance weighting should lead to an adaptive neural network that balances new knowledge and old knowledge (Pérez-Sánchez, Fontenla-Romero and Guijarro-Berdinas 2010). This approach addresses and incorporates the criticism that the influence of older

observations on network weights cannot be canceled later; the influence can only be reduced over time (Mena-Torres and Aguilar-Ruiz 2014).

Martínez-Rego, Pérez-Sánchez, et al. (2011) introduced an online and incremental one-layer neural network model for non-stationary problems that implements forgetting in the cost function by assigning higher weights to new observations. The objective cost function is weighted by a forgetting function that is constant in stationary environments and monotonically increasing in nonstationary environments. The model can thus adapt dynamically to both stable and dynamic environments. The method can model non-stationary environments without needing to detect changes or maintain irrelevant data. Martínez-Rego, Fontenla-Romero and Alonso-Betanzos (2012) updated the previous work by reducing the computational complexity of their algorithm. The previous algorithm was cumbersome because it needed to determine the weight for each new data sample by solving a system of linear equations and because the weighting of the data samples needed to be periodically reset.

Pérez-Sánchez, Fontenla-Romero and Guijarro-Berdinas (2010) proposed a neural network training scheme that uses a factor to weigh the error committed by each sample in order to "forget" old information. Pérez-Sánchez, Fontenla-Romero, et al. (2013) furthered the previous work by designing an online incremental neural network with adaptive network topology. The algorithm allows the network structure to change, depending on the needs of the learning process. For example, increasing the number of hidden neurons implies changes to both the modified network layer and the next network layer because number of inputs to the next layer grows.

Chapter 2.4.2: Neural Network Ensembles

Neural network ensembles, a combination of several networks, collectively produce classifications by some voting scheme (Hansen and Salamon 1990). Neural network ensembles have been used to improve the generalization of neural network performance. Members of the ensemble may be trained for the same task or concept or may be trained on different tasks or concepts (Akhand, Islam and Murase 2009).

There are many strategies for constructing an ensemble that adapts to changing environments. Results can by produced by simple or dynamically-weighted majority voting schemes (Elwell and Polikar 2009). Maintaining diversity in an ensemble can help reduce the initial drop in accuracy that occurs immediately after drift (Minku, White and Yao 2010). Maintaining old classifiers in an ensemble can allow the ensemble to handle recurrent drifts. When a new concept is encountered, the model simply adds a new classifier to the ensemble. Each classifier in the ensemble then belongs to a different concept (Ramamurthy and Bhatnagar 2007; Elwell and Polikar 2009). Each classifier in an ensemble could be constructed on a different subset of the available data points (Street and Kim 2001). Other techniques include dynamic combiners, updated training data, updating ensemble members, updating training data, structural changes of the ensemble, and adding new features (Kuncheva 2004).

Ensembles of neural networks have also been used to address concept drift. Ghazikhani, Monsefi and Yazdi (Online cost-sensitive neural network classifiers for non-stationary and imbalanced data streams 2013) proposed a cost-sensitive, online neural network ensemble for learning imbalanced classes in nonstationary environments. The method proposed a dynamic weighting method for the ensemble.

Ghazikhani, Monsefi and Yazdi (Ensemble of online neural networks for non-stationary and imbalanced data streams 2013) proposed an online ensemble of cost-sensitive neural network classifiers for non-stationary and imbalanced data streams. The cost function assigned more importance is assigned to errors in the minority class. The new mechanism for weighting classifiers of an online ensemble used the Winnow method in order to handle both concept drift and class imbalance. The ensemble uses a fixed number of classifiers and generates ensemble diversity by using different initial weights. Ghazikhani, Monsefi and Yazdi (2014) furthered their previous work by creating an online neural network model that applies a forgetting function to handle concept drift.

Chapter 2.5: Contribution of This Thesis

If a neural network is trained in time with a data stream, the neural network parameters will follow minor concept drifts and concept instability; the responsiveness of the network depends on the learning rate used in the training algorithm (Kuncheva 2004). Training a neural network on streaming data can therefore enable it to respond immediately to concept instability and extremely gradual drift. Even in the face of larger, faster drifts, a network will eventually learn to classify the new concept correctly. Yet there remains a period when the network's ability to produce correct classifications is reduced.

In this thesis, I demonstrate that if a learning system has detected concept drift, information about the drift can be used to reduce the underperformance of an online

neural network as it learns the new concept. Just as in Bayesian statistics,[4] where a hypothesis is updated in response to new information (Kruschke 2014), an online learning model can be updated in response to information about concept drift. In this context, information about the drift is used to update the model parameters, reducing the negative impact of concept drift on model performance. Because information about drift is different from a new training observation, such updates ought to be applied outside the standard learning process.

There are various attributes of concept drift that can be useful to a learning algorithm, including *magnitude*, *duration*, and *scope*. Magnitude is the distance between the new concept and the old concept; it is formally defined in Chapter 3 as the change in model accuracy after concept drift. Duration is the amount of time, or number of time steps, over which the concept is changing. Scope is the proportion of the domain of X for which $P(y|X)$ changes and affects how much of a model requires updating (Webb, Hyde, et al. 2016). The methods proposed in this thesis only make use of magnitude because the duration is negligible (concept drift is simulated as abrupt) and the selected definition of magnitude is very similar to the definition of scope.

This approach of using information about concept drift does not fall neatly into the categories of drift adaption described in Chapter 2.4. Instead, my thesis proposes that the learning system detect drift and make changes to model parameters such that the model moves away from the old concept. Under this model, weights and biases of a

[4] Bayesian statistics is a branch of statistics that generally applies Bayes' rule: $p(c|r) = \frac{p(r,c)}{p(r)} = \frac{p(r|c)P(c)}{p(r)}$. Bayes' rule states that the probability of c given r (the posterior belief) is the probability that both events occur together (the likelihood times the prior belief), relative to the probability of r (the evidence) (Kruschke2014). In terms of concept drift, the prior represents the existing belief that the current set of parameters is optimal and the likelihood and evidence represents the information from drift.

neural network are updated using the drift magnitude, an approach that has the added benefits of low computational costs and memory requirements. The details of this method and its variations are described in the following chapters.

Chapter 3: Methodology

The methodology is presented in this Chapter. I describe the proposed methods in Chapter 3.1, the simulated data sets with concept drift in Chapter 3.2, and the experiment design in Chapter 3.3.

Chapter 3.1: Description of the Proposed Method

After concept drift occurs, the proposed methods use the drift magnitude in various ways to update the neural network parameters independently of the gradient descent training algorithm. Drift magnitude is measured as the reclassification rate, or the change in the accuracy rate after drift occurs, calculated as

$$drift\ magnitude = \max\left(accuracy_{before}\right) - \ accuracy_{after}\ .$$

Model accuracy is defined as

$$accuracy = \frac{\#\ correct}{\#\ classified},$$

where the accuracy is evaluated on an independent set of test data. In cases were drift leads to an improvement in model accuracy (the magnitude is negative), the proposed methods need not be applied.

Given the calculated drift magnitude after concept drift, the proposed method adjusts the network parameters independently of the training algorithm to adapt to the drift. Three variations of updating are tested, each of which are applied to all network weights and applied to all network weights and biases for a total of six tests. Note that the term ‘node’ can refer to either weight or bias value.

The first update variation is *full reset*, where the drift magnitude dictates the probability that a node will be randomly reset from the trained value to a random value.

Weights are adjusted according the following functions:

$$w_{kj}(t_1) = b_{kj} w_{kj}(t_0) + \left(1 - b_{kj}\right) * rand$$

and

$$v_{ji}(t_1) = b_{ji} v_{ji}(t_0) + \left(1 - b_{ji}\right) * rand,$$

where t_0 and t_1 are represent the moments before and after reset is applied to the network weights trained at time t, respectively, and b_{kj} and b_{ji} are Bernoulli variables with probabilities $P\left(b_{kj} = 1\right) = magnitude$ and $P\left(b_{ji} = 1\right) = magnitude$. For example, if the drift magnitude is 20%, then each node has a 20% probability of being reset to a random value. Nodes are randomly selected because it is difficult or impossible to distinguish which combination of nodes represent the old concept and to what degree. Logically, a larger drift leads to resetting more nodes; thus, if the drift magnitude is 100%, then all nodes are reset and the neural network is effectively retrained from scratch. The "full reset" method is inspired by "dropout", a popular regularization technique that, for each training example, randomly drops nodes in hidden layers with a given probability and trains the remaining nodes with backpropagation (Baldi and Sadowski 2013). Dropout is similar to averaging neural network ensembles (Hinton, et al. 2012).

The second update variation is *scaled reset*, where all nodes are rescaled according to drift magnitude. In scaled reset, the new node values are determined by a sliding scale between the old node value and the new node value:

$$w_{kj}(t_1) = w_{kj}(t_0) * (1 - magnitude) + rand * magnitude$$

and

$$v_{ji}(t_1) = v_{ji}(t_0) * (1 - magnitude) + rand * magnitude.$$

Scaled reset is motivated by the idea that the greater the drift, the further the optimal network node values of the new concept will be from the trained node values representing the previous concept. The drift magnitude measures how close the new concept is to the old concept. Because the optimal node values to represent the new concept are unknown, a random value is used to represent the new concept. If there is a large drift, then the new node value is expected to be less like the old node value, therefore it is scaled closer to the random value. For small drifts, the new node value will be scaled closer to the old node value because less has changed. If the drift magnitude is 100%, scaled reset functions to reset the entire network. This update method tests whether the magnitude indicates how close the old network node is to the optimal value for the new concept.

The third update variation is *Bayesian rescaling*, which applies the Bayesian probability formula to node values. Bayesian-inspired rescaling reapplies Bayes' rule[5] to update the value of a node (Kruschke2014). Interpreting Bayes' prior as the node value under the old concept, the likelihood as the magnitude drift, and the sum of magnitude-weighted node values as the evidence, the following formulas for updating a node value given a drift magnitude were derived as

$$w_{kj}(t_1) = \frac{magnitude * w_{kj}(t_0)}{\sum_{j=0}^{J} magnitude * w_{kj}(t_0)}$$

and

$$v_{ji}(t_1) = \frac{magnitude * v_{ji}(t_0)}{\sum_{i=0}^{I} magnitude * v_{ji}(t_0)}.$$

[5] $p(c|r) = \frac{p(r|c)P(c)}{p(r)}$

Table 1 summarizes the six update methods that will be tested.

Table 1: Summary of network update methods

Method	Nodes updated
Full Reset	Weights and biases
Full Reset	Weights only
Scaled Reset	Weights and biases
Scaled Reset	Weights only
Bayesian Rescaling	Weights and biases
Bayesian Rescaling	Weights only

The three update methods have low computational costs and memory requirements. Computational costs are low because update methods are only applied when drift is detected; other solutions to nonstationary environments require additional computations to predict each new observation, such as dynamically weighting an ensemble to determine model output for each input or calculating how much to weight each new input to the model. Computational costs are also low because the methods are simple to calculate – each constant-time formula is applied once for each parameter and thus runs in $O(w)$ time, where w is the number of weights (and biases) to update. Memory requirements are low because only information about the current concept requires storage.

Chapter 3.2: Simulated Datasets

With real datasets, it is difficult to know exactly when concept drift begins or ends, the type of drift present, or whether drift truly occurred. So, to analyze the strengths and weaknesses of the described methods and to control the timing and severity of drift, I perform tests on simulated datasets. This way, it will be known for which situations the strategy will be useful (Minku, White and Yao 2010).

The described update methods will be tested on four simulated datasets, each labeled according to a different function. For each function, the x, y, and z parameters will be randomly generated; the other parameters of the functions will be set and readjusted when the simulated drift occurs.

The *SineV* function randomly generates x and y variables in the range -10 to 10. The label is assigned as *0* if $y \leq a\ \sin(bx + c) + d$; otherwise the label is *1*. The function takes three static parameters: $a = b = 1$ and $c = 0$. To simulate concept drift, the parameter d changes to a new value every 500 observations, assuming the following sequence of values: $-2, 1, -5, 4, -8, 7$ (Minku, White and Yao 2010). As shown in the scatter plot, changes in d shifts the class division (the concept) vertically.

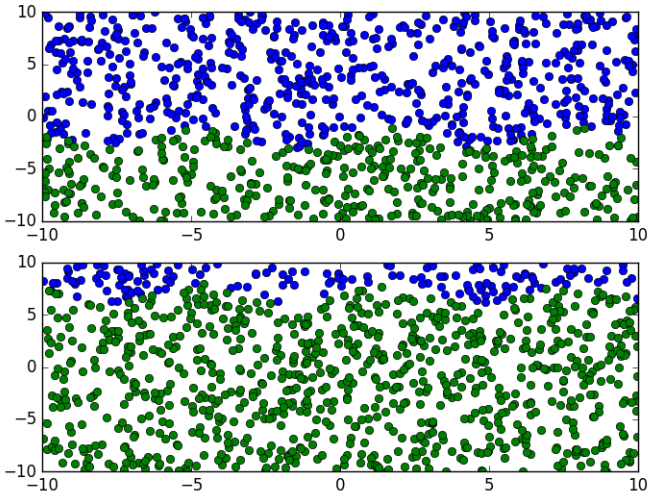

Figure 13: SineV concept, before and after drift

The *plane* function randomly generates x, y, and z variables. The label is assigned as *0* if $y \leq -a_0 + a_1 * x_1 + a_2 * x_2$; otherwise the label is *1*. The fixed parameters a_1 and a_2 default to the value 0.1. To simulate concept drift, the parameter a_0 changes to a new value every 500 observations, taking the following sequence of values: -2.0, -2.7, -1, -3.2, -0.7, -4.4 (Minku, White and Yao 2010). As shown in Figure 12, changes in a_0 rotates the three-dimensional plane (the concept) towards the origin of the x-y axis.

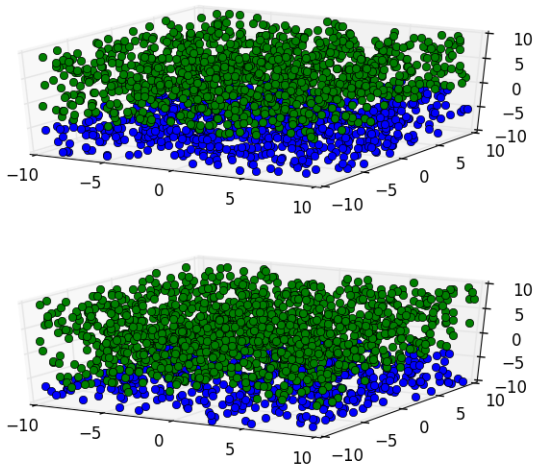

Figure 14: plane function, before and after drift

The *plane2d* rotating hyperplane function randomly generates x and y variables. The two classes are delineated by a straight line that is centered at the origin. The label is assigned as class *0* if $y \cos\theta + x \sin\theta > 0$, where $\theta = \frac{k*\pi}{180}$, otherwise the label is *1*. There is 10% noise in class assignments (Narasimhamurthy and Kuncheva 2007). When concept drift occurs, the line rotates k degrees around the origin. In the concept drift simulations, k takes on the following sequence of values: $0, 10, 25, 45, 70$. The number of degrees rotated increases after each drift to test the impact of varying degrees of drift.

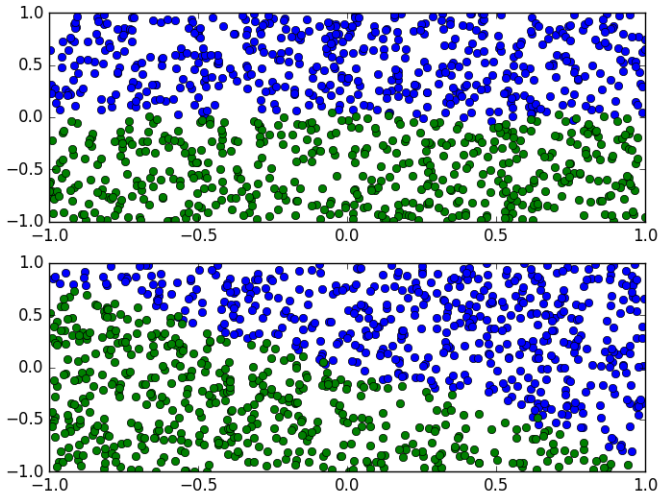

Figure 15: plane2d function before and after drift

The *four Gaussian components* function samples x and y values from a 2-dimensional mixture of four equiprobable Gaussian clusters. The means of the clusters are centered at $(10, 10)$, $(-10, 10)$, $(-10, -10)$, $(10, -10)$. Each point is assigned to one of two classes, depending on the current concept. During concept 1, if the y in (x, y) is less than 0, then the point is in class 1, otherwise it is in class 2. During concept 2, if the x in (x, y) is less than 0, then the point is in class 1, otherwise it is in class 2 (Narasimhamurthy and Kuncheva 2007).

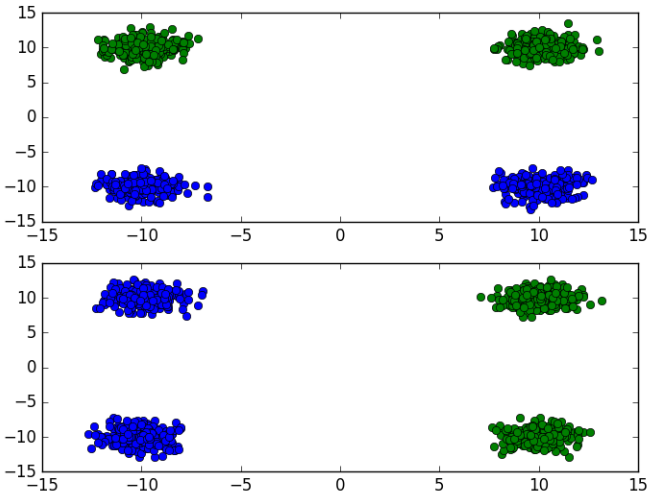

Figure 16: Four Gaussian Components function, before and after drift

The simulated datasets are implemented via a data generation class that returns a new single randomly generated data point at a time and assigns a label according to the current concept. After generating a given number of observations, the data generator labels the data points according to the subsequent concept until all pre-defined concepts have appeared.

Chapter 3.3: Experimental Design

All data simulations and experiments are implemented in Python. The neural network model is implemented by the pylearn2 package, a machine learning research library designed with flexibility to facilitate machine learning research (Goodfellow, et al. 2013). Pylearn2 was selected because it is extendable and allows the neural network parameters to be updated outside of the assigned learning function.

For each simulated dataset, the tests generate a baseline neural network with no external adjustments after drift and six networks that are modified after drift by the six

update functions described in Chapter 3.1. The seven neural networks are initialized with identical parameters, learning rates, and network structures for each dataset; the hidden layer weights are initialized in the range -0.1 to 0.1 and the biases are initialized to 1. The hidden layers uses a sigmoid activation function and the output layer uses a softmax activation function. Table 2 summarizes the network structures selected for each dataset. The networks for each dataset are initialized and trained with the same settings and data so that any differences in performance are attributable to the update methods alone.

Table 2: Neural network parameters

Dataset	**Learning Rate**	**# Input Nodes**	**# Hidden Nodes**	**# Output Nodes**
SineV	0.05	2	3	2
Plane	0.05	3	3	2
Plane2d	0.075	2	2	2
Four Gaussian Components	0.05	2	4	2

The seven neural networks are sequentially trained with gradient descent on the same sequence of labeled training data. After the networks are trained on each new observation, the accuracy is evaluated on the 1,000 test data points sampled from the current concept. Model accuracies are evaluated on test data that was not used as training inputs in order to test the generalized performances of the models.

When concept drift occurs, the first observation of the new concept is tested on the neural networks that model the old concepts because a network cannot be adjusted for concept drift until after it occurs. Then, 1,000 new test data points then are drawn from the new concept. This new test data is used to test the performances of the neural

networks under the new concept. The change in model performance (accuracy) is then used to calculate the drift magnitude. Once the drift has been detected and measured, the designated update method is applied to each network.

As noted by Engelbrecht (2007), any study evaluating the performance of neural networks ought to be based on several simulations, with each simulation initialized with different random initial weights and different training and test datasets. Engelbrecht (2007), further noted that at least 30 independent simulations ought to be run in order for the central limit theorem's[6] normality assumption to hold. Thus, to show that the results are not due to a fortuitous generation of random numbers, each experiment is tested 30 times with different seeds passed into every random number generator; the results are averaged and reported in aggregate.[7]

[6] The central limit theorem can be defined as follows: "the probability distribution governing the variable [x] approaches a Normal distribution as the number of observations (simulations) tends to infinity" (Engelbrecht 2007).
[7] The initial neural network parameters are not varied across simulations.

Chapter 4: Results and Discussion

In this chapter, the results are described and evaluated. The results are presented in Chapter 4.1; subsection 4.1.1 describes the evaluation metrics and subsections 4.1.2-5 present the results for each simulated dataset. In Chapter 4.2, the results are analyzed and discussed.

Chapter 4.1: Results

As explained in Chapter 3.3, the accuracy of each neural network is assessed after receiving each new observation. The results thus appear as a time series of accuracy rates. In order to improve the ability of the metrics described in Chapter 4.1.1 to measure the model recoveries, I first smooth out the volatility using a simple moving average over a window of 15 observations.[8] The primary benefit of removing the noise is to improve detection of model stability.

Model stability is a critical component of the metrics described in the following section. Qualitatively, a model is considered *stable* during an interval if the model accuracy is relatively constant throughout the interval. A model has *recovered* from concept drift if the accuracy rate is stable and at a level similar to the stable accuracy rate before drift – there are cases where a model may not regain the previous level of accuracy or may exceed the previous level of accuracy. Formally, a model is *stable* during the interval $[t, u]$ if $\max(accuracy_{[t,u]}) - \min(accuracy_{[t,u]}) < \theta$, where θ is the stability threshold. The model has *recovered* and is stable in the long run if

[8] The window includes 7 observations before the given observation, the given observation, and 7 observations after the given observation.

$\max(accuracy_T) - accuracy_t < \theta_l$, where T is any arbitrary point in the future and θ_l is the threshold for long term stability.[9]

Chapter 4.1.1: Metrics for Evaluating Results

This thesis adopts the resilience framework to evaluate the ability of the aforementioned update methods to aid in model recovery from concept drift. A formal definition of resilience is given by Vugrin, et al. (2010):

> Given the occurrence of a particular disruptive event (or set of events), the resilience of a system to that event (or events) is the ability to efficiently reduce both the magnitude and duration of the deviation from targeted system performance levels.

In this application, concept drift is the disruptive event, the system is the neural network model and its assigned update method, and the targeted system performance level is the long-run stable accuracy rate. Vugrin, et al. (2010) noted that their approach for measuring system resilience is not specific to any model or domain; it simply requires time series data that measure a system's output and recovery efforts. Because an inherent component of system resilience is recovery, the framework for evaluating recovery is directly applicable to this study.

The resilience of a system is measured by the systemic impact, recovery duration, and the recovery effort. *Systemic impact* measures the effect on system performance, and is calculated as the difference between targeted and actual system performance levels following the disruptive event (concept drift). The *recovery duration* is the duration of time between system disruption and system recovery, $duration = u - t_0$. The *total recovery effort* is the amount of resources consumed

[9] In this application, T is the last point of the current concept before the next drift occurs.

during the recovery process following disruption (Vugrin, et al. 2010). Because the recovery effort of all tested methods is the same (one external update to the network nodes), this metric will not be used.

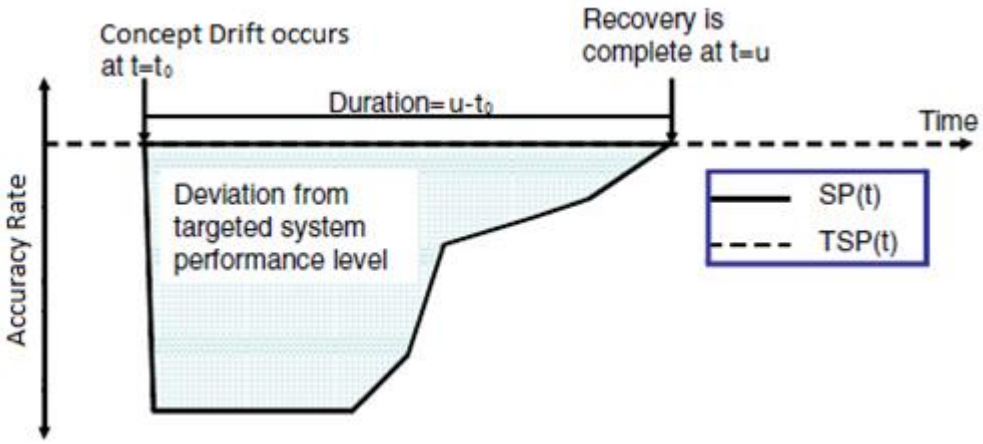

Figure 17: Resilience illustration

Figure 17 (Vugrin, et al. 2010) illustrates the measurement of system resilience in a nonstationary environment. Given concept drift at time $t = t_0$, the systemic impact is the total deviation of the actual system performance (SP) from the targeted system performance (TSP) level. The duration is the number of time steps in which the system performance is less than the targeted system performance level. The model has recovered when $SP(u) = TSP(u)$ (Vugrin, et al. 2010).

In evaluating the results of this study, the recovery duration and systemic impact will be used, in addition to the worst accuracy after drift and the categorical evaluation of recovery to pre-concept drift levels. *Recovery duration*, as described before, is the number of time steps between concept drift and model recovery, or the amount of time the neural network is underperforming. The *systemic impact* is measured as $SI = \int_{t_0}^{u}[TSP(t) - SP(t)]dt$, where TSP is the target system performance and SP is the

actual system performance. It is the area between the target system performance and the actual system performance, illustrated in Figure 17. A good update method will reduce the systemic impact of concept drift on the neural network. The *worst accuracy after drift*, or largest deviation from target system performance is included because the update method should reduce the post-drift drop in accuracy, not increase it. Because accuracy is a proxy for closeness to the target concept: the update method should aid in bringing the model *closer* to the target concept, not further away. Given $accuracy_f$, the model accuracy under the previous concept, and $accuracy_g$, the model accuracy under the new concept after the model has stabilized, the *recovery performance* of a neural network is evaluated as follows. If the concept is stable and $abs(accuracy_f - accuracy_g) = \theta$, the performance is *recover*. If the concept is stable and $accuracy_f > accuracy_g$, the recovery performance is *underperform*. Otherwise, the concept is stable and $accuracy_f < accuracy_g$; the recovery performance is *overperform*.

Compared to the baseline neural network, an update method that adds robustness to the network will minimize recovery time, minimize systemic impact, and maximize worst accuracy after drift, and will either recover to or overperform the model accuracy under the old concept.

Chapter 4.1.2: Results – SineV

The averaged accuracy rates of the SineV experiments are plotted in Figure 18. The vertical axis represents the accuracy rate (0-100%) and the horizontal axis represents the time units (new observations are introduced to the system each time step). The five vertical lines indicate the five points in time when concept drift occurred. The

chart plots seven lines that represent accuracy rates of the baseline neural network with no update after concept drift and the six neural networks augmented by the aforementioned update methods.

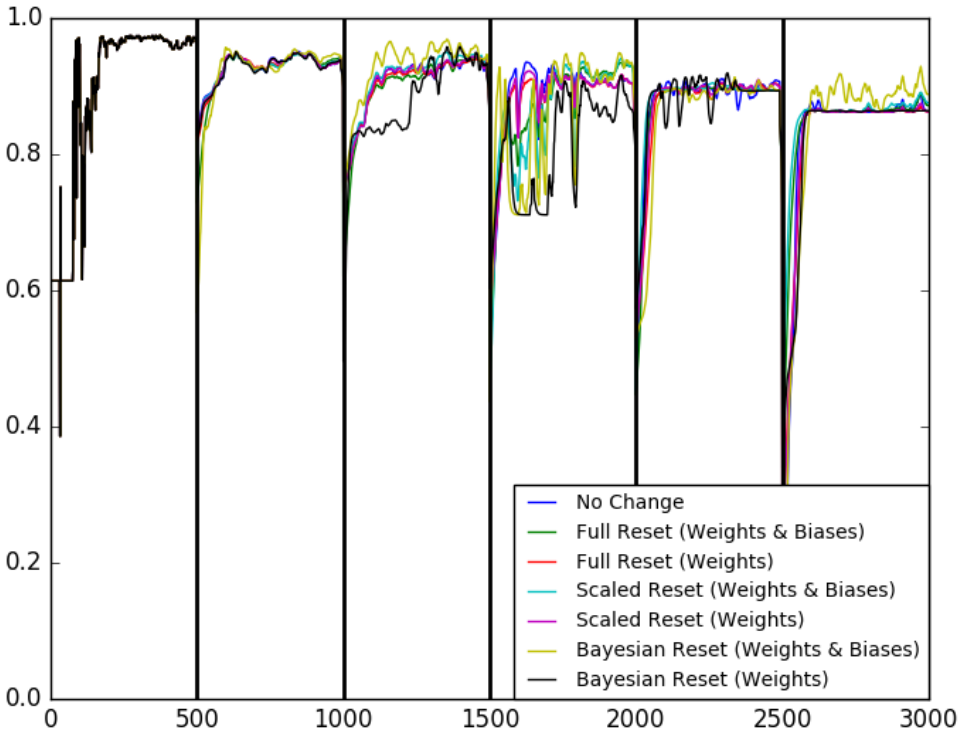

Figure 18: SineV results

Before the first concept drift occurs (time 0 through 499), all the neural networks have identical accuracy rates because they are initialized identically and are trained using the same observations. After each concept drift occurs, all networks experience some drop in accuracy and generally recover to some "stable" or constant level. After the first concept drift at time 500, all networks recover at similar rates. The network with Bayesian rescaling applied to the weights took longer to recover than the other networks after the second drift at observation 1000. Several networks took a long

time to recover after the third drift at time 1500; the variance in accuracy rates was high before converging. After the fourth drift at time 2000 and fifth drift at time 2500, most networks converged to a stable accuracy rate quickly. Interestingly, none of the networks recovered to the accuracy rate achieved prior to the first drift.

The results in terms of the resilience metrics described in the previous section are presented in Table 3. The first column lists the metric used by the given table section: worst accuracy rate after drift, recovery duration, systemic impact, and recovery performance. The neural networks that recover the best have high worst accuracy rates, small recovery duration, and small systemic impact. The second column, Label, indicates which instance of concept drift the row is describing; "Drift 1," for example, refers to the first vertical line at time 500 in Figure 18. The third column, Drift Magnitude, lists the average measured concept drift magnitude for the given drift instance. The remaining columns list the measured results for each tested method for the given metric; "WB" represents "weights and biases" and "W" indicates "weights" only. The first row, for example, lists the drift label, the drift magnitude, and the worst accuracy rates of each update method after the first drift (time 500).

Table 3: SineV results

Metric	*Label*	*Drift Magnitude*	*No change*	*Full (WB)*	*Full (W)*	*Scaled (WB)*	*Scaled (W)*	*Bayes (WB)*	*Bayes (W)*
Worst Accuracy	Drift 1	14%	84%	73%	82%	83%	84%	57%	83%
	Drift 2	21%	76%	62%	74%	75%	75%	74%	50%
	Drift 3	36%	58%	53%	58%	47%	58%	44%	60%
	Drift 4	46%	50%	45%	53%	45%	53%	53%	57%
	Drift 5	78%	17%	34%	25%	39%	24%	14%	22%
Recovery Duration	Drift 1	14%	98	95	96	96	97	94	98
	Drift 2	21%	105	104	105	105	106	146	344
	Drift 3	36%	117	220	115	253	117	399	479
	Drift 4	46%	66	60	71	48	64	84	47
	Drift 5	78%	130	74	77	60	76	237	80
Systemic Impact	Drift 1	14%	4.80	6.74	5.13	4.57	4.90	10.47	5.13
	Drift 2	21%	7.47	9.14	7.51	8.02	8.13	11.38	33.62
	Drift 3	36%	13.55	21.27	11.64	27.23	12.83	29.14	28.59
	Drift 4	46%	9.89	11.00	12.09	7.16	11.05	17.46	8.14
	Drift 5	78%	24.32	12.07	21.38	8.42	21.17	32.00	23.01
Recovery Performance	Drift 1	14%	recover	under	recover	recover	recover	recover	recover
	Drift 2	21%	Under	under	under	under	under	recover	recover
	Drift 3	36%	recover	recover	under	recover	recover	recover	under
	Drift 4	46%	recover	under	recover	under	recover	under	recover
	Drift 5	78%	Under	under	under	under	under	recover	under

Table 4 shows the difference between performance of networks with the given update methods and the baseline (No Change) neural network for each of the three numerical metrics. For example, the network applying the full reset on both weights and biases ("Full (WB)") method had a worst accuracy rate 11% lower than the baseline after drift 1 and 17% higher worst accuracy rate after drift 5. Further, full reset on weights and biases recovered 3 and 1 time steps faster than the baseline after the first

and second drifts, respectively, but 103 time steps slower after the third drift. Full reset on weights and biases had a 1.94 higher systemic impact than the baseline network after drift 1. Both scaled reset methods performed better than the baseline in all three metrics after drift 5 – the worst accuracy rate was higher (22% and 7% higher), the recovery duration was faster (70 and 54 time steps better), and the systemic impact was better (15.91 and 3.16 lower).

Table 4: SineV results relative to baseline

Metric	*Label*	*Full (WB)*	*Full (W)*	*Scaled (WB)*	*Scaled (W)*	*Bayes (WB)*	*Bayes (W)*
Worst Accuracy	Drift 1	-11%	-2%	-1%	0%	-27%	-1%
	Drift 2	-14%	-1%	-1%	0%	-2%	-26%
	Drift 3	-6%	-1%	-12%	0%	-15%	2%
	Drift 4	-5%	3%	-5%	3%	3%	7%
	Drift 5	17%	8%	22%	7%	-3%	5%
Recovery Duration	Drift 1	3	2	2	1	4	0
	Drift 2	1	0	0	-1	-41	-239
	Drift 3	-103	2	-136	0	-282	-362
	Drift 4	6	-5	18	2	-18	19
	Drift 5	56	53	70	54	-107	50
Systemic Impact	Drift 1	-1.94	-0.32	0.23	-0.10	-5.67	-0.33
	Drift 2	-1.67	-0.03	-0.55	-0.66	-3.90	-26.14
	Drift 3	-7.71	1.92	-13.67	0.72	-15.59	-15.04
	Drift 4	-1.11	-2.20	2.73	-1.16	-7.57	1.74
	Drift 5	12.26	2.94	15.91	3.16	-7.68	1.31

Chapter 4.1.3: Results – plane

The mean accuracy rates of the *plane* experiments are plotted in Figure 19. The chart has the same properties as Figure 18, which displayed the SineV results.

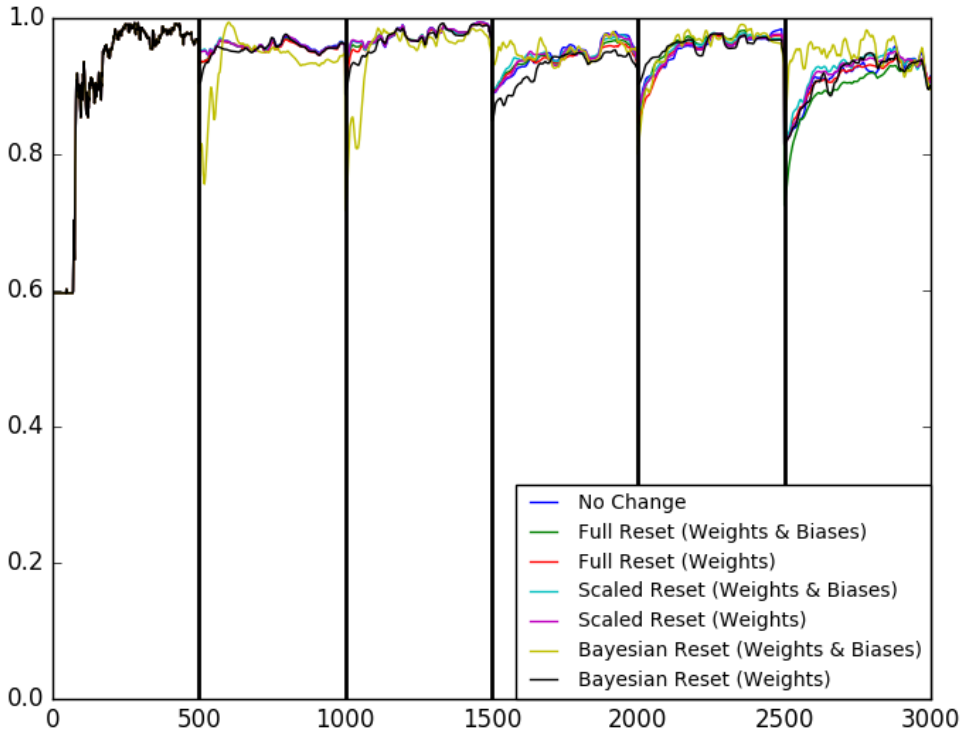

Figure 19: Plane results

As in the SineV simulations (Figure 18), all the neural networks modeling the plane dataset have identical accuracy rates before the first concept drift occurs at time 500. After each concept drift occurs, most networks experience some drop in accuracy and generally recover to some "stable" or constant level. After the first drift (time 500), the baseline neural network and the networks with scaled reset all maintained accuracy rates consistent with those prior to the drift. As shown in Table 5, these networks have a recovery duration of 1 after the first drift. The other tested methods experience some

drop in accuracy after the first concept drift. After the second drift at time 1000, the baseline neural network and the networks with scaled reset experienced a small drop in accuracy and quickly recover to the previous level of accuracy after only several time steps. Networks using the other four update methods had greater drops in accuracy and took longer to recover. After the third and fourth drifts, all networks displayed some drop in accuracy after concept drift and required more than a few time steps to recover. Bayesian rescaling for weights and biases made the network most resilient to the third drift and Bayesian rescaling for weights made the network most resilient to the fourth drift. Aside from Bayesian rescaling for weights and biases, most networks experienced the steepest drop in accuracy and worst systemic impact after the fifth drift at time 2500.

The results in terms of the resilience metrics described in the Chapter 4.1.1 are presented in Table 5. The table is structured like Table 4 in Chapter 4.1.2. The baseline model was unaffected by the first drift of magnitude of 4% – it required a single time step to recover fully to the previous accuracy level and experienced zero systemic impact. The impact of concept drift on the baseline neural network was similarly negligible after the second concept drift. Only the scaled reset methods showed similar performance after the first drift.

Table 5: Plane results

Metric	*Label*	*Drift Magnitude*	*No change*	*Full (WB)*	*Full (W)*	*Scaled (WB)*	*Scaled (W)*	*Bayes (WB)*	*Bayes (W)*
Worst Accuracy	Drift 1	4%	95%	93%	93%	95%	95%	76%	90%
	Drift 2	3%	94%	93%	92%	94%	94%	72%	89%
	Drift 3	11%	89%	89%	90%	90%	89%	93%	85%
	Drift 4	10%	86%	84%	85%	86%	87%	80%	90%
	Drift 5	17%	82%	73%	81%	82%	82%	85%	81%
Recovery Duration	Drift 1	4%	1	59	59	1	1	131	48
	Drift 2	3%	16	16	21	16	16	105	56
	Drift 3	11%	154	63	82	67	121	85	96
	Drift 4	10%	129	110	127	116	127	180	30
	Drift 5	17%	96	101	96	96	96	98	195
Systemic Impact	Drift 1	4%	0.00	0.93	0.94	0.00	0.00	8.36	0.95
	Drift 2	3%	0.20	0.23	0.42	0.20	0.20	10.75	1.02
	Drift 3	11%	3.35	0.94	1.69	1.38	3.09	-0.60	2.17
	Drift 4	10%	4.17	4.44	5.73	3.99	4.34	7.60	0.52
	Drift 5	17%	5.40	5.42	4.01	4.97	5.11	1.88	9.47
Recovery Performance	Drift 1	4%	recover	recover	recover	recover	recover	recover	recover
	Drift 2	3%	recover	recover	recover	recover	recover	over	recover
	Drift 3	11%	under	under	under	under	under	under	under
	Drift 4	10%	recover	recover	recover	recover	recover	recover	recover
	Drift 5	17%	under	under	under	under	under	recover	under

Table 6 shows the improvement of the neural networks with each update method over the baseline neural network for each metric. After the concept drifts in the plane simulation, the worst accuracy rates of most neural networks were very similar to that of the baseline network. The networks with scaled weights and biases recovered in the same number of time steps as the baseline after drifts 1, 2, and 5 where the recovery durations were short; the network with full reset on weights and biases also recovered in

the same number of time steps after the second drift and the network with full reset on weights recovered in the same number of time steps after the fifth drift.

Table 6: Plane results relative to baseline

Metric	*Label*	*Full (WB)*	*Full (W)*	*Scaled (WB)*	*Scaled (W)*	*Bayes (WB)*	*Bayes (W)*
Worst Accuracy	Drift 1	-1%	-1%	0%	0%	-19%	-4%
	Drift 2	-1%	-3%	0%	0%	-23%	-5%
	Drift 3	0%	0%	1%	0%	4%	-4%
	Drift 4	-3%	-2%	0%	1%	-6%	4%
	Drift 5	-9%	0%	0%	0%	4%	0%
Recovery Duration	Drift 1	-58	-58	0	0	-130	-47
	Drift 2	0	-5	0	0	-89	-40
	Drift 3	91	72	87	33	69	58
	Drift 4	19	2	13	2	-51	99
	Drift 5	-5	0	0	0	-2	-99
Systemic Impact	Drift 1	-0.93	-0.94	0.00	0.00	-8.36	-0.95
	Drift 2	-0.03	-0.22	0.00	0.01	-10.55	-0.82
	Drift 3	2.40	1.66	1.96	0.26	3.94	1.18
	Drift 4	-0.27	-1.56	0.17	-0.18	-3.43	3.65
	Drift 5	-0.01	1.39	0.43	0.29	3.52	-4.07

Chapter 4.1.4: Results – plane2d

The mean accuracy rates of the *plane2d* experiments are plotted in Figure 20. The figure has the same properties as the charts showing the SineV and plane results, with the slight deviation that there are only four drifts.

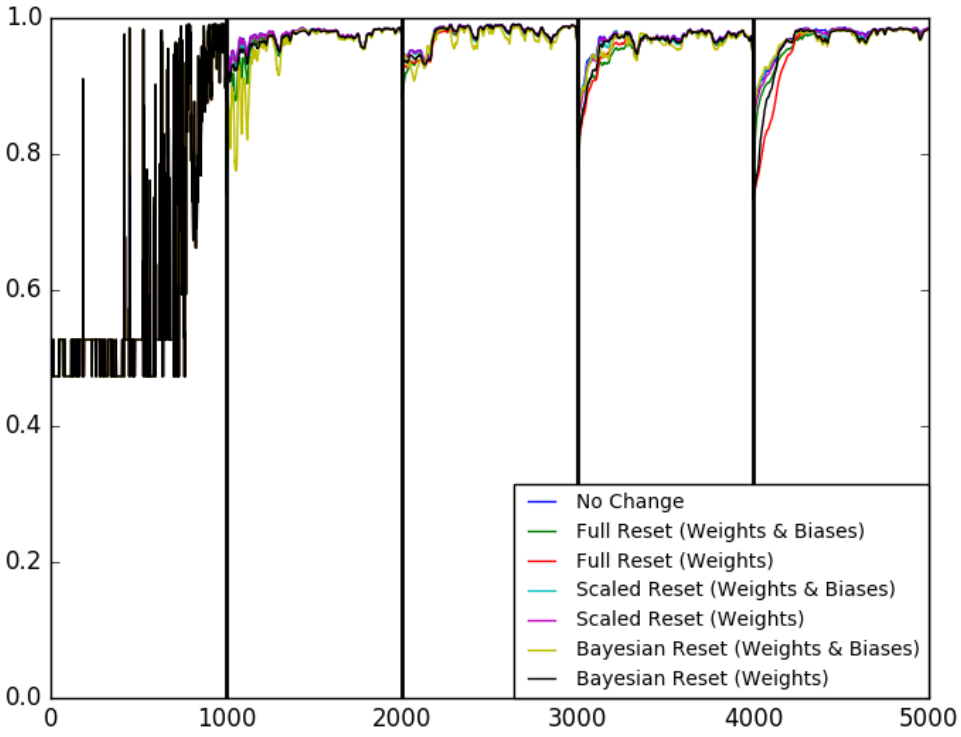

Figure 20: Plane2d results

Prior to the first concept drift, the neural networks converge after around 900 time steps. After convergence, the neural networks show stable performance after each recovery from concept drift. After the first drift at time 1000, some networks, such as Bayesian rescaling and full reset on weights and biases display volatile performance before converging. Responding to the second drift at time 2000, all networks perform stably at a reduced level for approximately 100 time steps before returning to the

previous accuracy level. After drifts three and four at times 3000 and 4000, respectively, all networks show a drop in accuracy and a steady recovery to the previous performance level at varying rates. An interesting property of the plane2d tests is that the accuracy rates are highly correlated, even after drift: all the networks converge to the same accuracy rates after each concept drift.

The results in terms of the resilience metrics are presented in Table 7. As illustrated in Figure 20, the recovery measures are fairly similar across neural networks.

Table 7: Plane2d results

Metric	*Label*	*Drift Magnitude*	*No change*	*Full (WB)*	*Full (W)*	*Scaled (WB)*	*Scaled (W)*	*Bayes (WB)*	*Bayes (W)*
Worst Accuracy	Drift 1	11%	92%	74%	92%	91%	92%	78%	91%
	Drift 2	5%	93%	88%	92%	93%	93%	91%	93%
	Drift 3	13%	87%	78%	82%	87%	85%	86%	82%
	Drift 4	13%	87%	78%	73%	87%	84%	87%	73%
Recovery Duration	Drift 1	11%	125	137	128	129	125	186	129
	Drift 2	5%	172	171	173	171	172	169	172
	Drift 3	13%	111	198	170	113	116	116	121
	Drift 4	13%	147	177	228	140	146	138	150
Systemic Impact	Drift 1	11%	0.44	7.76	1.32	1.12	0.38	12.48	2.22
	Drift 2	5%	3.14	4.86	4.06	3.05	3.21	4.20	3.78
	Drift 3	13%	3.32	8.72	8.50	3.67	4.46	3.99	7.07
	Drift 4	13%	5.33	8.75	21.67	4.83	6.28	4.61	14.20
Recovery Performance	Drift 1	11%	under	under	under	under	under	under	under
	Drift 2	5%	recover	recover	under	recover	recover	recover	recover
	Drift 3	13%	under	under	under	under	under	under	under
	Drift 4	13%	recover	recover	recover	recover	recover	recover	recover

Like Tables 6 and 7, Table 8 shows the difference between the performances of the neural network with the given update method relative to the baseline neural network. As illustrated in Figure 20, the differences in how the neural networks recovered from concept drift were very similar; this correlation is apparent in Table 8, where most differences between networks with update methods and the baseline are very small. The networks with full reset generally produced lower worst accuracy rates, recovered slower, and accrued a higher systemic impact than the baseline neural network.

Table 8: Plane2d results relative to baseline

Metric	*Label*	*Full (WB)*	*Full (W)*	*Scaled (WB)*	*Scaled (W)*	*Bayes (WB)*	*Bayes (W)*
Worst Accuracy	Drift 1	-18%	-1%	-1%	0%	-15%	-2%
	Drift 2	-5%	-1%	0%	0%	-3%	-1%
	Drift 3	-9%	-5%	0%	-1%	-1%	-4%
	Drift 4	-9%	-14%	0%	-2%	0%	-13%
Recovery Duration	Drift 1	-12	-3	-4	0	-61	-4
	Drift 2	1	-1	1	0	3	0
	Drift 3	-87	-59	-2	-5	-5	-10
	Drift 4	-30	-81	7	1	9	-3
Systemic Impact	Drift 1	-7.31	-0.88	-0.68	0.06	-12.04	-1.78
	Drift 2	-1.72	-0.92	0.09	-0.07	-1.07	-0.64
	Drift 3	-5.40	-5.18	-0.35	-1.14	-0.67	-3.75
	Drift 4	-3.42	-16.34	0.50	-0.95	0.72	-8.88

Chapter 4.1.5: Results – four Gaussian components

The mean accuracy rates of the *four Gaussian component* experiments are plotted in Figure 21. As illustrated, only one drift was applied to this simulation. Prior to drift, all networks achieved and maintained 100% accuracy. After the single concept

drift at time 500, all networks eventually return to the 100% accuracy level. Both Bayesian update methods aid the network in recovering from the drift in only a couple time steps. The network that applied full reset on weights and biases recovered the slowest and took many training observations to return to the 100% accuracy level.

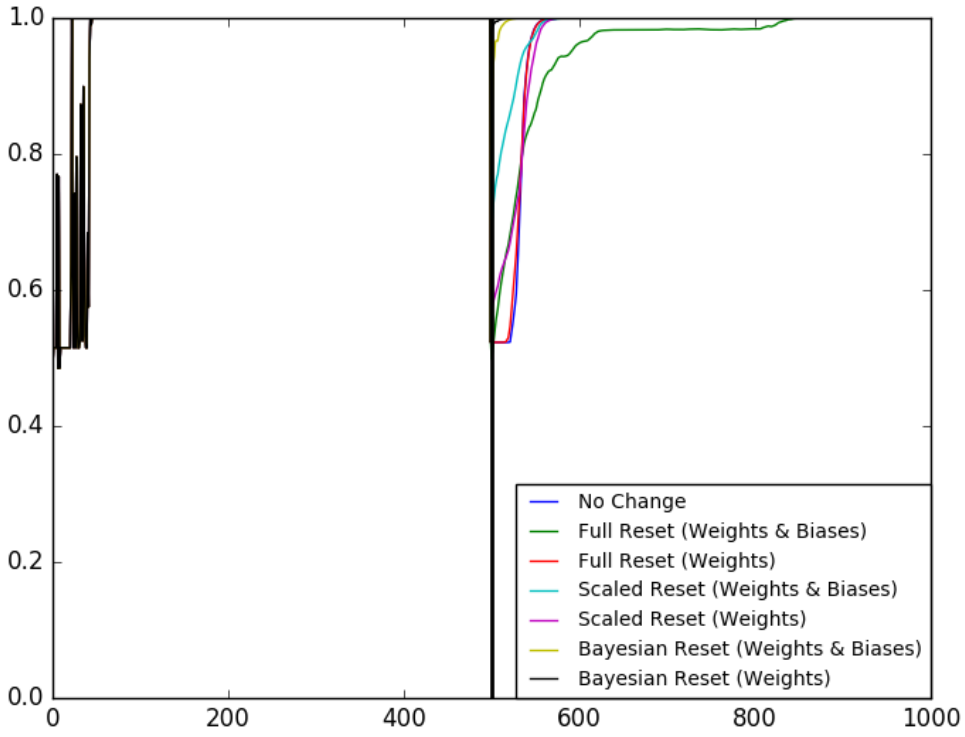

Figure 21: Four Gaussian components results

The results in terms of the resilience metrics described in the Chapter 4.1.1 are presented in Table 9. Unlike the previous tables, there is no column listing the drift label because only one drift occurred for this simulation. Although Figure 21 shows the recovery duration for full reset on weights and biases to be greater than 300 time steps, the table records the recovery duration as only 115 time steps because the accuracy of the neural network is stable (constant) for a long time – nearly 200 time steps – and

meets the definition for long-term stability in Chapter 4.1.1. The table records the Recovery Performance of this update method as "under" because the network underperformed the target accuracy rate of 100% at the time it was assessed to be stable. As illustrated in Figure 21, the networks with Bayesian rescaling have good worst accuracies, very short recovery durations, and negligible systemic impacts.

Table 9: Four Gaussian components results

Metric	*Drift Magnitude*	*No change*	*Full (WB)*	*Full (W)*	*Scaled (WB)*	*Scaled (W)*	*Bayes (WB)*	*Bayes (W)*
Worst Accuracy	48%	52%	50%	52%	69%	57%	84%	97%
Recovery Duration	48%	52	115	52	56	60	14	2
Systemic Impact	48%	15.93	15.67	15.37	6.02	13.27	0.51	0.02
Recovery Performance	48%	recover	under	recover	recover	recover	recover	recover

Table 10 shows the difference between the performance of networks with each update method and the baseline neural network for each metric. Although networks using the full reset on weights and biases and scaled reset had slightly longer recovery durations than the baseline network, these networks still had a better systemic impact.

Table 10: Four Gaussian components results relative to baseline

Metric	*Full (WB)*	*Full (W)*	*Scaled (WB)*	*Scaled (W)*	*Bayes (WB)*	*Bayes (W)*
Worst Accuracy	-2%	0%	16%	5%	31%	45%
Recovery Duration	-63	0	-4	-8	38	50
Systemic Impact	0.26	0.56	9.91	2.66	15.42	15.91

Chapter 4.2: Discussion

An ideal update method will enable a neural network to be more *resilient* to concept drift than a baseline network that makes no attempt to respond to drift beyond the defined online learning process. Such an update method will aid the neural network in maximizing the worst accuracy after drift, minimizing the recovery duration, and minimizing the systemic impact. The ideal method should also aid the method in recovering to the pre-drift performance levels. In addition, in cases where the update method is shown to cause a neural network to be less resilient than the baseline, the decrease in resilience will be negligible. A *negligible* decrease in performance is defined as some negative, near-zero difference between the updated network and the baseline. An update method allows a network to perform *well* relative to the baseline if the recovery performance is better than the baseline, matches the baseline performance, or showed only a negligible decrease in performance relative to the baseline.

Chapter 4.2.1: Evaluation of Update Methods

In addition to Tables 3 through 10, Table 11 averages the differences between the performance of each six update methods and the baseline for each metric. Aggregating across all tests for the given method, the table displays the average differences between worst accuracy rates, average differences in recovery duration, and average differences in systemic impact.

Table 11: Average Difference from the Baseline

	Full (WB)	***Full (W)***	***Scaled (WB)***	***Scaled (W)***	***Bayes (WB)***	***Bayes (W)***
Avg Worst Accuracy	-5%	-1%	0%	2%	-6%	-1%
Avg Recovery Duration	-3.67	-5.40	7.93	9.73	-43.07	-31.80
Avg Systemic Impact	-1.14	-1.38	0.47	0.66	-4.40	-2.62

On average, the neural networks to which the full reset methods were applied performed worse than the baseline in the three quantitative metrics. Specifically, the average worst accuracy was 1 to 5% lower, the average recovery duration was 3 to 5 time steps longer, and the systemic impact was 1.14 to 1.38 greater. For concept drifts of magnitude less than 40%, the worst accuracy rate of the networks with full reset were less than or equal to the baseline neural network. In around half of the simulated concept drifts, the networks with full reset recovered at a rate greater than or equal to the baseline neural network. In just one third of simulated concept drifts, the networks with full reset had a lower systemic impact than the baseline neural network.

Table 11 shows that on average, the scaled reset methods match or outperform the baseline in the three selected metrics. Specifically, the average worst accuracy rate was 0 to 2% higher, the recovery duration was 7 to 9 time steps faster, and the average systemic impact was 0.47 to 0.66 lower. For concept drifts of magnitude less than 40%, the neural networks with scaled reset produce worst accuracy rates similar to those of the baseline neural networks. After concept drifts of magnitude greater than 40%, the worst accuracy rates for the networks with scaled reset are greater than or equal to those of the baseline neural networks. In general, neural networks with scaled reset have similar or faster recovery durations than the baseline neural networks. When a network with scaled reset recovers more slowly than the baseline, the difference is only a few time steps. The notable exception is recovery duration of the network with scaled reset for weights and biases after Drift 3 of the SineV simulation – the updated network required nearly 300 more time steps to recover than the baseline. With regard to systemic impact, neural networks with scaled reset have an equivalent or lower

systemic impact than the baseline in over 60% of concept drifts. With the exception of drift 3 of the SineV simulation, when the systemic impact of the network with scaled reset is less than the baseline, the differences are small.

On average, the neural networks with Bayesian rescaling all recovered slower and produced worse systemic impact the baseline neural networks. The average worst accuracy was 1 to 6% lower, the average recovery duration was 31 to 43 time steps longer, and the systemic impact was 2.62 to 4.40 greater. Neural networks with Bayesian rescaling recovered much better than the baseline neural network in some cases and much worse in other cases. Networks with the Bayesian rescaling methods performed very well after the concept drift in the four Gaussian components dataset: worst accuracy rates were 84-97%, compared to the baseline of 52%; recovery durations were short, taking only 2 – 14 time steps compared to the 52 time steps taken by the baseline; systemic impacts were 0.51 or less, compared to the 15.93 systemic impact of the baseline. On the other hand, the networks with Bayesian rescaling performed very poorly relative to the baseline on other tests. For example, Bayesian rescaling required over 250 time steps to recover than the baseline after drift 3 the SineV simulation (see table 4) and over 47 to 130 more time steps to recover than the baseline after drift 1 of the plane simulation. Accordingly, the systemic impacts of the networks with these slow recoveries is large relative to the baseline.

It is clear from this evaluation that only the networks with scaled reset meet the desired criteria of performing *well* relative to the baseline in most tests and metrics. The neural networks with full reset do not consistently perform better than the baseline in the three quantitative metrics. Despite performing well in some cases, networks with

Bayesian rescaling performed poorly relative to the baseline after several concept drifts, violating the requirement that any decreases in performance are negligible.

Chapter 4.2.2: Evaluation by Drift Magnitude

When concept drift was small, none of the tested update methods enabled the neural networks to perform better than the baseline. When the concept drift magnitude is less than 10%, the networks with an update method either match or perform worse than the performance of the baseline network in all metrics. After drifts of this small magnitude, networks with scaled reset have similar worst accuracy, recovery duration, and systemic impact as the baseline network because the network weights were changed by only a small amount. The full reset methods, which completely reset the value of one or more nodes, likely removed too much information. The Bayesian rescaling methods likely moved the node weights too far from both the original and new concepts.

When concept drift was large, the tested methods enabled the neural networks to perform better than the baseline in most cases. After larger drifts of magnitude greater than 40%, nearly all update methods enabled its neural network to recover faster than the baseline or have a smaller systemic impact than the baseline. This is explained by the fact that update methods move the weight values further from the old concept (and theoretically closer to the new target concept) faster than the assigned network learning rates allow.

This suggests it is best to apply the selected update method in cases where concept drift is above some threshold. The tests indicate that such a threshold might be around 40%.

Chapter 5: Conclusion

In this thesis, I introduced the idea that information about concept drift can be applied to assist online learning algorithms in recovering from that concept drift. I proposed and tested three methods that applied data about concept drift to online neural networks: full reset, scaled reset, and Bayesian-inspired rescaling. These three methods were tested on four simulated datasets that generated concept drift at assigned intervals.

The results show that a neural network applying scaled reset after drift performs better than an online neural network with no drift adaption. It follows that there is value in the information about concept drift that can be used to aid neural networks, and online learning algorithms in general. Further, the benefits of scaled reset on neural network resilience to concept drift were negligible when drift magnitude was small but significant when drift magnitude was large. This suggests that the scaled reset method ought to be applied when drift is not small, or is above some threshold.

In conclusion, information about drift can be used to assist a learning algorithm in better recovering from drift – in time-sensitive applications, any reduction in a model's underperformance is valuable. As an added benefit, this can be accomplished cheaply in terms of computational costs and memory requirements.

There are several areas in which this work could be extended. The proposed scaled reset method could be tested in real-world environments. The framework could be applied to other online learning algorithms, such as support vector machines. The proposed update methods could be further refined or additional information about concept drift could be incorporated into the methods.

References

Akhand, M. A. H., Md. Monirul Islam, and K. Murase. 2009. "Progressive interactive training: A sequential neural network ensemble learning method." *Neurocomputing* 73: 260-273. doi:http://dx.doi.org/10.1016/j.neucom.2009.09.001.

Bach, Stephen, and Mark Maloof. 2010. "A bayesian approach to concept drift." *Advances in neural information processing systems.* 127-135.

Baena-Garcia, Manuel, Jose Campo-Avila, Raul Fidalgo, Albert Bifet, R. Gavalda, and R. Morales-Bueno. 2006. "Early drift detection method." *Fourth international workshop on knowledge discovery from data streams.* 77-86.

Baldi, Pierre, and Peter J. Sadowski. 2013. "Understanding Dropout." In *Advances in Neural Information Processing Systems 26*, edited by C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani and K. Q. Weinberger, 2814-2822. Curran Associates, Inc. http://papers.nips.cc/paper/4878-understanding-dropout.pdf.

Balzanella, Antonio, Lidia Rivoli, and Rosanna Verde. 2013. "Data stream summarization by histograms clustering." In *Statistical Models for Data Analysis*, 27-35. Springer.

Bifet, Albert, and Ricard Gavalda. 2007. "Learning from time-changing data with adaptive windowing." *Proceedings of the 2007 SIAM International Conference on Data Mining.* 443-448.

Bifet, Albert, Jesse Read, Indre Zliobaite, Bernhard Pfahringer, and Geoff Holmes. 2013. "Pitfalls in Benchmarking Data Stream Classification and How to Avoid Them." In *Machine Learning and Knowledge Discovery in Databases:*

European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part I, edited by Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen and Filip {\v{Z}}elezn{\'y}, 465-479. Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-642-40988-2_30.

Bottou, Leon. 1998. "Online learning and stochastic approximations." *On-line learning in neural networks* (Cambridge Univ Pr) 17: 142.

Chen, Kylie, Yun Sing Koh, and Patricia Riddle. 2015. "Tracking Drift Severity in Data Streams." In *AI 2015: Advances in Artificial Intelligence: 28th Australasian Joint Conference, Canberra, ACT, Australia, November 30 -- December 4, 2015, Proceedings*, edited by Bernhard Pfahringer and Jochen Renz, 96-108. Cham: Springer International Publishing. doi:10.1007/978-3-319-26350-2_9.

Ditzler, Gregory, Manuel Roveri, Cesare Alippi, and Robi Polikar. 2015. "Learning in nonstationary environments: A survey." *IEEE Computational Intelligence Magazine* (IEEE) 10: 12-25.

Dries, Anton, and Ulrich Ruckert. 2009. "Adaptive concept drift detection." *Statistical Analysis and Data Mining* (Wiley Online Library) 2: 311-327.

Duda, Richard O., Peter E. Hart, and David G. Stork. 2012. *Pattern classification.* John Wiley & Sons.

Ellis, Byron. 2014. *Real-Time Analytics: Techniques to Analyze and Visualize Streaming Data.* 1st. Wiley Publishing.

Elwell, Ryan, and Robi Polikar. 2009. "Incremental Learning in Nonstationary Environments with Controlled Forgetting." *Proceedings of the 2009*

International Joint Conference on Neural Networks. Piscataway, NJ: IEEE Press. 1388-1395. http://dl.acm.org/citation.cfm?id=1704175.1704377.

Engelbrecht, Andries P. 2007. *Computational intelligence: an introduction.* John Wiley & Sons.

Esposito, F., S. Ferilli, N. Fanizzi, T. M. A. Basile, and N. Di Mauro. 2004. "Incremental Learning and Concept Drift in INTHELEX." *Intell. Data Anal.* (IOS Press) 8: 213-237. http://dl.acm.org/citation.cfm?id=1293831.1293833.

Gama, Joao, Indre Zliobaite, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. "A Survey on Concept Drift Adaptation." *ACM Comput. Surv.* (ACM) 46: 44:1--44:37. doi:10.1145/2523813.

Gama, Joao, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. 2004. "Learning with drift detection." *Brazilian Symposium on Artificial Intelligence.* 286-295.

Ghazikhani, Adel, Reza Monsefi, and Hadi Sadoghi Yazdi. 2013. "Ensemble of online neural networks for non-stationary and imbalanced data streams ." *Neurocomputing* 122: 535-544. doi:http://dx.doi.org/10.1016/j.neucom.2013.05.003.

Ghazikhani, Adel, Reza Monsefi, and Hadi Sadoghi Yazdi. 2013. "Online cost-sensitive neural network classifiers for non-stationary and imbalanced data streams." *Neural Computing and Applications* 23 (5): 1283-1295. doi:10.1007/s00521-012-1071-6.

Ghazikhani, Adel, Reza Monsefi, and Hadi Sadoghi Yazdi. 2014. "Online neural network model for non-stationary and imbalanced data stream classification." *International Journal of Machine Learning and Cybernetics* (Springer) 5: 51-62.

Goodfellow, Ian J., David Warde-Farley, Pascal Lamblin, Vincent Dumoulin, Mehdi Mirza, Razvan Pascanu, James Bergstra, Fr{\'{e}}d{\'{e}}ric Bastien, and Yoshua Bengio. 2013. "Pylearn2: a machine learning research library." *arXiv preprint arXiv:1308.4214.* http://arxiv.org/abs/1308.4214.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning.* MIT Press.

Grossberg, Stephen. 1987. "Competitive learning: From interactive activation to adaptive resonance." *Cognitive science* (Wiley Online Library) 11: 23-63.

Guha, Sudipto, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. 2003. "Clustering data streams: Theory and practice." *IEEE transactions on knowledge and data engineering* (IEEE) 15: 515-528.

Hansen, L. K., and P. Salamon. 1990. "Neural Network Ensembles." *IEEE Trans. Pattern Anal. Mach. Intell.* (IEEE Computer Society) 12: 993-1001. doi:10.1109/34.58871.

Henzinger, M. Rauch, Prabhakar Raghavan, and Sridhar Rajagopalan. 1998. "Computing on data streams." Tech. rep., Technical Note 1998-011, Digital Systems Research Center, Palo Alto, CA.

Hinton, Geoffrey E., Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. 2012. "Improving neural networks by preventing co-adaptation of feature detectors." *arXiv preprint arXiv:1207.0580.*

Huang, David Tse Jung, Yun Sing Koh, Gillian Dobbie, and Russel Pears. 2014. "Detecting volatility shift in data streams." *Data Mining (ICDM), 2014 IEEE International Conference on.* 863-868.

Jain, Lakhmi C., Manjeevan Seera, Chee Peng Lim, and P. Balasubramaniam. 2014. "A review of online learning in supervised neural networks." *Neural Computing and Applications* (Springer) 25: 491-509.

Kosina, Petr, Joao Gama, and Raquel Sebastiao. 2010. "Drift Severity Metric." *Proceedings of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence.* Amsterdam, The Netherlands, The Netherlands: IOS Press. 1119-1120. http://dl.acm.org/citation.cfm?id=1860967.1861234.

Kruschke, John. 2014. *Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan.* Academic Press.

Kuhn, Max, and Kjell Johnson. 2015. *Applied Predictive Modeling.* New York: Springer.

Kuncheva, Ludmila I. 2004. "Classifier ensembles for changing environments." *International Workshop on Multiple Classifier Systems.* 1-15.

Kuncheva, Ludmila I., and Indre Zliobaite. 2009. "On the Window Size for Classification in Changing Environments." *Intell. Data Anal.* (IOS Press) 13: 861-872. http://dl.acm.org/citation.cfm?id=1662648.1662650.

Martínez-Rego, David, Beatriz Pérez-Sánchez, Oscar Fontenla-Romero, and Amparo Alonso-Betanzos. 2011. "A robust incremental learning method for non-stationary environments ." *Neurocomputing* 74: 1800-1808. doi:http://dx.doi.org/10.1016/j.neucom.2010.06.037.

Martínez-Rego, David, Oscar Fontenla-Romero, and Amparo Alonso-Betanzos. 2012. "Nonlinear single layer neural network training algorithm for incremental,

nonstationary and distributed learning scenarios." *Pattern Recognition* 45: 4536-4546. doi:http://dx.doi.org/10.1016/j.patcog.2012.05.009.

Mena-Torres, Dayrelis, and Jesús S. Aguilar-Ruiz. 2014. "A similarity-based approach for data stream classification ." *Expert Systems with Applications* 41: 4224-4234. doi:http://dx.doi.org/10.1016/j.eswa.2013.12.041.

Minku, Leandro L., Allan P. White, and Xin Yao. 2010. "The Impact of Diversity on Online Ensemble Learning in the Presence of Concept Drift." *IEEE Trans. on Knowl. and Data Eng.* (IEEE Educational Activities Department) 22: 730-742. doi:10.1109/TKDE.2009.156.

Murata, Noboru. 1998. "A statistical study of on-line learning." *Online Learning and Neural Networks. Cambridge University Press, Cambridge, UK* 63-92.

Murata, Noboru, Motoaki Kawanabe, Andreas Ziehe, Klaus-Robert Muller, and Shun-ichi Amari. 1998. "On-line learning in changing environments with applications in supervised and unsupervised learning." *Online Learning and Neural Networks. Cambridge University Press, Cambridge, UK* 93-110.

Narasimhamurthy, Anand, and Ludmila I. Kuncheva. 2007. "A Framework for Generating Data to Simulate Changing Environments." *Proceedings of the 25th Conference on Proceedings of the 25th IASTED International Multi-Conference: Artificial Intelligence and Applications.* Anaheim, CA: ACTA Press. 384-389. http://dl.acm.org/citation.cfm?id=1295303.1295369.

Nishida, Kyosuke, and Koichiro Yamauchi. 2007. "Detecting concept drift using statistical testing." *International conference on discovery science.* 264-269.

Pérez-Sánchez, Beatriz, Oscar Fontenla-Romero, and Bertha Guijarro-Berdiñas. 2016. "A review of adaptive online learning for artificial neural networks." *Artificial Intelligence Review* (Springer) 1-16.

Pérez-Sánchez, Beatriz, Oscar Fontenla-Romero, and Bertha Guijarro-Berdinas. 2010. "An incremental learning method for neural networks in adaptive environments." *Neural Networks (IJCNN), The 2010 International Joint Conference on.* 1-8.

Pérez-Sánchez, Beatriz, Oscar Fontenla-Romero, Bertha Guijarro-Berdiñas, and David Martínez-Rego. 2013. "An online learning algorithm for adaptable topologies of neural networks." *Expert Systems with Applications* 40: 7294-7304. doi:http://dx.doi.org/10.1016/j.eswa.2013.06.066.

Rajaraman, Anand, Jeffrey Ullman, and Jure Leskovec. 2014. *Mining of Massive Datasets.* New York, NY, USA: Cambridge University Press.

Ramamurthy, Sasthakumar, and Raj Bhatnagar. 2007. "Tracking recurrent concept drift in streaming data using ensemble classifiers." *Machine Learning and Applications, 2007. ICMLA 2007. Sixth International Conference on.* 404-409.

Saad, David, ed. 1998. *On-line Learning in Neural Networks.* New York, NY, USA: Cambridge University Press.

Saad, David, and Magnus Rattray. 1998. "Optimal on-line learning in multilayer neural networks." *Online Learning in Neural Networks* 135-164.

Schlimmer, Jeffrey C., and Richard H. Granger. 1986. "Incremental learning from noisy data." *Machine Learning* 1: 317-354. doi:10.1007/BF00116895.

Street, W. Nick, and YongSeog Kim. 2001. "A Streaming Ensemble Algorithm (SEA) for Large-scale Classification." *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* New York, NY, USA: ACM. 377-382. doi:10.1145/502512.502568.

Vugrin, Eric D., Drake E. Warren, Mark A. Ehlen, and R. Chris Camphouse. 2010. "A Framework for Assessing the Resilience of Infrastructure and Economic Systems." In *Sustainable and Resilient Critical Infrastructure Systems: Simulation, Modeling, and Intelligent Engineering*, edited by Kasthurirangan Gopalakrishnan and Srinivas Peeta, 77-116. Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-642-11405-2_3.

Webb, Geoffrey I., Michael J. Pazzani, and Daniel Billsus. 2001. "Machine Learning for User Modeling." *User Modeling and User-Adapted Interaction* 11: 19-29. doi:10.1023/A:1011117102175.

Webb, Geoffrey I., Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean. 2016. "Characterizing concept drift." *Data Mining and Knowledge Discovery* 30: 964-994. doi:10.1007/s10618-015-0448-4.

Widmer, Gerhard, and Miroslav Kubat. 1996. "Learning in the presence of concept drift and hidden contexts." *Machine Learning* 23: 69-101. doi:10.1007/BF00116900.

Zou, Jinming, Yi Han, and Sung-Sau So. 2009. "Overview of artificial neural networks." *Artificial Neural Networks: Methods and Applications* (Springer) 14-22.