Toward Generate-and-Test Algorithms for Continual Feature Discovery

by

Parash Rahman

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

The backpropagation algorithm is a fundamental algorithm for training modern artificial neural networks (ANNs). However, it is known the backpropagation algorithm performs poorly on changing problems. We demonstrate the backpropagation algorithm can perform poorly on a clear, generic, changing task. The task is online meaning the agent learns from one sample at a time from a stream of samples. The task is nonstationary since the sample distribution regularly changes. We call it the *generic continual feature discovery task* (GCFD), as it is sufficiently difficult that the backpropagation algorithm must regularly discover new features to perform well.

We propose an explanation for the poor performance of the backpropagation algorithm on the GCFD task. The backpropagation algorithm consists of two phases: initializing an ANN with small random weights, and using stochastic gradient descent to update the weights with data. It is known that the initialization step is crucial to the fast discovery of useful features with the backpropagation algorithm, and a typical initialization step sets the weights to small, random numbers. We corroborate that the small, random weight initialization step leads to conditions that speed up the discovery of useful features with the backpropagation algorithm. Then, we show that these conditions are not maintained during the GCFD task. Without the maintenance of these conditions, there is little reason to expect the backpropagation algorithm to quickly discover useful features for new sample distributions.

We demonstrate that the backpropagation algorithm's performance on the

GCFD task can be significantly improved with generate-and-test algorithms. The generate-and-test algorithms replace the least useful features of the ANN with features that have small, random weights. By regularly introducing features with small random weights, we restore conditions the backpropagation algorithm can use to quickly discover useful features for new data distributions.

To my family,

who have kept close despite being far

There is a great deal of pain in life and perhaps the only pain that can be avoided is the pain that comes from trying to avoid pain.

– R. D. Laing, 1927-1989.

Acknowledgements

I am indebted to my supervisors Richard Sutton and Joseph Modayil. Both have shown great patience during my growth, and both have set valuable expectations that I will continue to hone. Their dedication to ideas, clarity, and proper experimentation is inspiring. Each conversation with them was a blast of awareness that reflected their wisdom and care about research.

I also acknowledge the regulars of Rich's Representation Group: Rupam Mahmood, Adam White, Richard Sutton, Amir, Banafshe, Chen, Fernando, Katya, Matt, Shibhansh. The feedback and discussions inside and outside the meetings were motivating and helped shape this thesis.

I am grateful to the many that listened to my thoughts and that have reviewed my writing. Thanks to all my labmates in in the AMII lab and those in the RLAI lab for unforgettable memories.

Finally, thank you to my parents for hearing me out about my work.

Contents

1 Introduction **1**

2 Background **10**
- 2.1 Quick Notation Overview . . . 10
- 2.2 Task Taxonomy . . . 11
- 2.3 Representations and Features . . . 13
- 2.4 The Backpropagation Algorithm . . . 15

3 Properties for Fast Feature Discovery **18**
- 3.0.1 Activation Function Derivatives . . . 19
- 3.0.2 Feature Diversity . . . 21
- 3.1 Initial Feature Diversity: Experiment . . . 23
- 3.2 Initial Feature Diversity: Results . . . 25
- 3.3 Initial Feature Diversity: Conclusions . . . 27
- 3.4 Summary of Properties for Fast Learning . . . 27

4 Evaluation for Continual Feature Discovery **29**
- 4.1 Description of the GCFD Task . . . 31
- 4.2 Evaluation for Continual Feature Discovery . . . 33
- 4.3 Summary . . . 35

5 Continual Feature Discovery with the Backpropagation Algorithm **37**
- 5.1 Experiment Description . . . 39
- 5.2 Results . . . 42
- 5.3 Conclusions . . . 49

6 Continual Feature Discovery with Generate-and-Test Algorithms **53**
- 6.1 Generate-and-Test Algorithms . . . 56
- 6.2 Experiment Description . . . 57
- 6.3 Experiment Results . . . 58
- 6.4 Experiment Conclusions . . . 64

7 Conclusion **69**

Bibliography **72**

List of Tables

3.1 Optimized step sizes. 25
5.1 Optimized step sizes. 43

List of Figures

1.1 A diagram of a 2-layer ANN. 2

2.1 A plot of some activation functions. 14
2.2 These plots show derivatives of various activation functions. . 17

3.1 These plots show results from investigating how an ANN's initial feature diversity affects feature discovery performance. . . 26

5.1 These plots have error curves from subtasks 1, 30, 1,000, and 4,000 for size-15 ANNs. 44
5.2 These plots have error curves from subtasks 1, 30, 1,000, and 4,000 for size-40 ANNs. 45
5.3 The backpropagation algorithm's average errors at the end of subtasks on the generic continual feature discovery task. . . . 46
5.4 These plots show average activation function derivative at the beginning of subtasks with the backpropagation algorithm. . . 47
5.5 These plots show average activation function derivative scaled by the representation step size at the beginning of subtasks with the backpropagation algorithm. 48
5.6 These plots show the proportion of calcified features at the beginning of subtasks with the backpropagation algorithm. . . . 49

6.1 These plots show the learning curves on the last subtask with the backpropagation algorithm and generate-and-test algorithms (15-feature ANNs). 59
6.2 These plots show the learning curves on the last subtask with the backpropagation algorithm and generate-and-test algorithms (40-feature ANNs). 60
6.3 These plots show the errors at the end of subtasks with the backpropagation algorithm and generate-and-test algorithms. . 61
6.4 These plots show the average activation function derivatives at the beginning of each subtask when using the backpropagation algorithm and generate-and-test algorithms. 63
6.5 These plots show the average proportion of calcified features at the beginning of subtasks when using the backpropagation algorithm and generate-and-test algorithms. 65
6.6 These plots show how random generate-and-test and partial random replace perform compared to a fixed-representation ANN. 66

Chapter 1

Introduction

A *representation* provides intelligent agents with a useful summary of the agent's input. A summary of the agent's input can be used by the agent's other functions, which we call *output functions*. Output functions can serve various purposes including estimation, prediction, action selection or classification. A useful representation can summarize the agent's input in a way that allows for accurate calculations by the output functions. Output functions can be considered simpler functions (e.g. linear functions) than the representation (e.g. a complex nonlinear function). The representation does much of the difficult calculation needed for an output function's output.

Representations can be viewed as a composition of smaller functions called *features*. A feature takes in input signals and outputs a scalar summary of these input signals. The input signals for a feature can come from the agent's input, the output of other features, or some other source internal to the agent. The scalar summaries from features are concatenated together or combined in some way to form the representation's summary of the agent's input. A feature does not need to contribute directly to the representation's summary of the agent's input, but can indirectly contribute by feeding its scalar summary into other features. Artificial neural networks (ANNs) (see Figure 1.1 for an example), neatly fall into this described view of representations. ANNs consist of features that together output a vector summary to a simpler output function called the *output layer*.

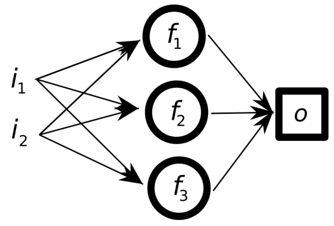

Figure 1.1: A 2-layer artificial neural network with three features ($f_1, f_2,$ and f_3), two input signals ($i_1,$ and i_2), and one output function (o). We focus on discovering useful features for this 2-layer architecture.

Representation learning algorithms can find useful representations that would otherwise be difficult for humans to handcraft. By considering a representation as consisting of features, learning a useful representation is the same as learning useful features. *Generate-and-test algorithms*, which are one of the key algorithm types discussed in this thesis, make the search for useful features explicit. Generate-and-test algorithms generate potentially useful features to replace the least useful features in an ANN (as deemed by a heuristic). In the past, generate-and-test algorithms have been used to learn functions that can be broken up into parts including representations (Blum and Langley, 1997; Guyon and Elisseeff, 2003; Mahmood and Sutton, 2013), Boolean functions (Kaelbling, 1990), and rule-based systems (Booker et al., 1989). The backpropagation algorithm, another key algorithm investigated in this thesis, can also be viewed as a generate-and-test algorithm. A modern usage of the backpropagation algorithm consists of two phases: initializing the ANN features with small, random weights, and using stochastic gradient descent to update the ANN weights with samples. The backpropagation algorithm is a generate-and-test algorithm that initially generates random features, and then regularly replaces features with incrementally modified versions (Mahmood and Sutton, 2013). The generation and testing steps can be seen as two separate steps important for representation learning: discovery of useful new features, and retention of useful current features. We refer to a learning algorithm's ability to discover useful new features as *feature discovery*.

There have been successful artificial neural network (ANN) learning algorithms for *offline supervised learning* tasks, tasks where learning algorithms are provided all data upfront to train a learner with. The backpropagation algorithm is a fundamental algorithm for offline supervised learning tasks, and has been used to train ANNs for the difficult tasks of helicopter flying, image recognition, and playing the game of Go (Ng et al., 2006; Cireşan et al., 2011; Silver et al., 2016). However, an offline supervised learning task is a limited problem formulation for learners that regularly receive samples from a changing sample distribution.

Updating representations regularly to account for changes in the stream of samples can yield better representations. New samples can hold information previously unknown or contradictory to past samples, and new samples are abundant during the operation of the agent (Sutton and Whitehead, 1993). If the new samples are sufficiently different from past samples, the agent may need to discover new features to adapt. The ability for a learning algorithm to discover useful features for new sample distributions is key for adapting to new situations. We call this ability to find useful features for new sample distributions *continual feature discovery*. Continual feature discovery can be seen as a repeated application of feature discovery, the ability to find useful features. In this thesis, we investigate algorithms for their continual feature discovery.

Continual supervised learning tasks are useful for evaluating learning algorithms that must learn new things regularly for good performance. Continual supervised learning tasks are supervised learning tasks that regularly provide the learner with new samples where the samples can come from new sample distributions. The number of times the learner is provided new samples is arbitrarily large, and can be considered potentially infinite. Past samples are not preserved in this task type, but the learner may choose to save copies of the samples in its memory. Since the sample distribution changes in a continual supervised learning task, the task is *nonstationary*.

To make progress on algorithms that scale to arbitrarily long continual supervised learning tasks, we focus on algorithms that use a bounded amount of computation for each new sample and that use a bounded amount of memory. These algorithms are known as *strictly-incremental algorithms* (Sutton and Whitehead, 1993). The backpropagation algorithm, described above, is a strictly-incremental algorithm. To clearly evaluate strictly-incremental algorithms, we use *online* supervised learning tasks, tasks that provide the learning algorithm with one sample at a time. Online tasks evaluate performance on each new sample. Establishing algorithms for online nonstationary supervised learning tasks can aid learners that learn from a large continuous stream of samples (e.g. monitoring systems, reinforcement learning systems, or robots). Our requirement of restricting the learning algorithm to have a bounded memory disqualifies our consideration of algorithms that depend on an unbounded memory to perform well. Fahlman and Lebiere (1989) presented the cascade correlation architecture (CCA) that added new features regularly to learn, but never removed features. The CCA without modification could potentially use an arbitrarily large amount of memory in an arbitrarily long, changing task. Fu et al. (1996) presented a CCA learning algorithm that emphasized a memory limit that kept the memory usage bounded. In a continual supervised learning task, the algorithm's performance after the memory usage has been reached is more interesting since the learner may operate for an arbitrarily long amount of time.

Contrary to our use of online supervised learning tasks, many continual supervised learning tasks used in the literature regularly provide the agent with multiple samples at once (Aljundi et al., 2019), sometimes, providing the learner with extra information about when the sample distribution changes (e.g. Whenever the learner receives a new set of samples, it is assumed the new data comes from a new sample distribution) (Rusu et al., 2016; Kirkpatrick et al., 2017; Shin et al., 2017; Yoon et al., 2018). Hayes et al. (2018) terms these tasks *incremental-batch* tasks. Incremental-batch tasks, like the classical offline supervised learning tasks, evaluate performance on unseen samples to test for generalization. Many algorithms proposed for incremental-batch tasks are not suitable for online supervised learning tasks, since they require extra knowledge about when the sample distribution changes.

Continual feature discovery can be evaluated separately from another subject that is often studied in the continual supervised learning literature: *catastrophic interference*. An ANN training with the backpropagation algorithm is said to experience catastrophic interference when its performance drops suddenly and completely on old sample distributions after learning from a new sample distribution (McCloskey and Cohen, 1989). Since the introduction of the concept of catastrophic interference, many works have proposed methods to mitigate the forgetting in ANNs trained with the backpropagation algorithm (Fahlman and Lebiere, 1989; French, 1999; Aljundi et al., 2017; Kirkpatrick et al., 2017; Yoon et al., 2018; Golkar et al., 2019; Rajasegaran et al., 2019; Rolnick et al., 2019). Methods proposed to reduce catastrophic interference include preventing changes to features that were adapted for past distribution (Fahlman and Lebiere, 1989; Draelos et al., 2017), reducing the amount of changes to features that were adapted for past distribution (McCloskey and Cohen, 1989; Kirkpatrick et al., 2017), and updating features with past samples (Rolnick et al., 2019). However, Kemker et al. (2017) pointed out many methods for reducing catastrophic interference reduced the learning speed on new sample distributions showing how these methods prioritize preserving performance on past sample distributions over performance on new sample distributions. French (1999) noted that the process that focuses on preserving knowledge about past sample distributions can be considered separately from the ability to learn from new sample distributions. We embrace this idea, and restrict our investigation to studying how well a learning algorithm can discover useful features for new sample distributions.

Our first contribution in this thesis is a task that clearly tests a learning algorithm's continual feature discovery: the *generic continual feature discovery* (GCFD) task (Chapter 4). The GCFD task was inspired by other online nonstationary supervised learning tasks that regularly change the sample distribution and that use samples generated from a generic data distribution (Mermillod et al., 2012; Sutton, 2014; Sahoo et al., 2018). One can abundantly generate samples with a generic process, and the sample generating process can be transparently tweaked to change what abilities are tested for. The GCFD task specifically requires regularly discovering useful features for good performance. The *Syn8* task used by Sahoo et al. (2018) was the most similar task found in the literature, but it purposely changes the difficulty of learning new sample distributions in addition to requiring new features to be discovered. On the other hand, the GCFD task only tests how well learning algorithms continually discover useful features, where learning useful features for new sample distributions is approximately as difficult as learning useful features for all other sample distributions.

Our second contribution in this thesis is investigating the backpropagation algorithm's continual feature discovery performance on the GCFD task (Chapter 5). Sutton (2014) presented A. R. Mahmood's results showing the backpropagation algorithm slows down in learning from new sample distributions as the sample distribution changes. We expand on this result by trying various sized ANNs and multiple activation functions to investigate whether there is a slowdown in the discovery of useful feature with various ANN architectures on the GCFD task. Other works that investigate the backpropagation algorithm's performance on new sample distributions bring little attention to this reported fundamental weakness of the backpropagation algorithm. Sahoo et al. (2018) focused on showing that the backpropagation algorithm slowed down when training deep ANNs on new sample distributions and attributed the slowness to the large depth of the ANNs. In contrast, we highlight the backpropagation algorithm's significant feature discovery slowdown with a shallow 2-layer ANN. Also, we found the SoftSign activation function, a lesser used activation function function in research and practice, led to significantly better continual feature discovery performance than the performance with the more commonly considered tanh, logistic and ReLU activation functions.

We investigated how two properties set by the backpropagation algorithm's initialization, the activation function derivatives and the feature diversity, change over the course of learning. In Chapter 3, we review the empirical evidence of others and ourselves that indicates the activation function derivatives and the feature diversity affect the learning speed with the backpropagation algorithm. In Chapter 5, we see how these key properties change when the backpropagation algorithm is used for the GCFD task. We find these properties significantly decrease during the learning, which can help explain the backpropagation algorithm's poor continual feature discovery.

Finding out the backpropagation algorithm slows in discovering useful features for new sample distributions led us to our third contribution: demonstrating that generate-and-test algorithms can be used to quicken the continual feature discovery of the backpropagation algorithm (Chapter 6). A generate-and-test algorithm regularly replaces the least useful features with potentially useful features. In our case, we wished to regularly replace features that were deemed to be least useful (as deemed by a heuristic) with features that contribute to fast feature discovery with the backpropagation algorithm (features with small, random weights). As mentioned earlier, the backpropagation algorithm's feature discovery speed is dependent on its feature weight initialization step that initializes features with small, random weights. By regularly introducing features with small, random weights, we found the backpropagation algorithm discovered useful features quicker when learning from new sample distributions in the GCFD task. This improvement in feature discovery speed for new sample distributions shows that generate-and-test algorithms can benefit the continual feature discovery performance of the backpropagation algorithm.

Past works have suggested algorithms similar to generate-and-test algorithms to adapt other learning algorithms from a stationary task to a nonstationary task. Self-organizing networks are a type of representation that have units called *neurons* that should be initialized with high diversity for fast learning. As a self-organizing network is updated, the diversity amongst the neurons decreases making it slow for the self-organizing map to adapt to new sample distributions (Fritzke, 1997). Fritzke (1997) proposed to repeatedly replace the least useful neurons with random neurons to quickly adapt to new sample distributions. The need for diversity is recognized for evolutionary algorithms. Evolutionary algorithms are slow to change potential solutions called *genes*. Initial diversity among genes can greatly improve the speed in which evolutionary algorithms find a good solution (Mauldin, 1984). Therefore, it is recommended for evolutionary algorithms operating on nonstationary tasks to have an extra step where some genes are killed off to be replaced by fully random genes (Mauldin, 1984; Moriarty, 1997; Cui et al., 2005). An ANN that updates with the backpropagation algorithm is yet another learner that is sensitive to initial conditions of its units (its features). The initial conditions that enable quick learning with the backpropagation algorithm include diversity of the features and high, unstable activation function derivatives. We show it can be effective to use generate-and-test algorithms to maintain these conditions.

Continual feature discovery is a useful ability of representation learning algorithms that expect to learn from new sample distributions. By being able to discover useful features for accurate estimations, the agent can have a representation that adapts to new situations. We used the GCFD task to clearly demonstrate that the backpropagation algorithm can have significantly poor continual feature discovery. We also showed that the generate-and-test algorithms can significantly improve the backpropagation algorithm's continual feature discovery performance. More work will be needed to establish generate-and-test algorithms that reliably improve continual feature discovery performance, but this thesis shows that generate-and-test algorithms hold potential.

Chapter 2

Background

The background material in this chapter will go over the concepts needed for understanding later chapters and the contributions of this thesis. It is recommended to read *Quick Notation Overview* section, but the other parts may be skipped if the reader has a good grasp on the concepts.

First, we discuss some notation conventions used in this thesis. Then, we discuss different types of supervised learning tasks to present a definition of *online nonstationary supervised learning tasks.* We define different parts of the artificial neural network including the representation, the output layer, and the features. We then describe how an artificial neural network can be trained with the backpropagation algorithm for an online nonstationary supervised learning task.

2.1 Quick Notation Overview

A prerequisite for this thesis includes basic linear algebra and knowledge of a gradient and a derivative. There are some random variables and basic probability theory used in the explanations. We use lowercase letters to represent scalars, bold lowercase letters for vectors, and bold uppercase letters for matrices. The $\cdot$ operator is the dot product operator between two vectors.

2.2 Task Taxonomy

In this thesis, we used *supervised learning tasks* to test the learner's learning capabilities. In a supervised learning task, the learner receives samples to learn from. Each sample contains an input and an associated target. The goal of the learner is to accurately estimate a sample's target given the sample's input. The learner can use the error between the true target and its estimate to improve its estimation performance.

Supervised learning tasks require training samples generated with a *target function*, the function that maps a sample's input to the sample's target. If the targets are scalars, the task is a *regression* task. If the targets are a category, the task is a *classification* task. In this thesis, we are focused on regression tasks. How many samples the learner is given at a time, and how often the learner receives samples, can determine the type of supervised learning task.

Online supervised learning tasks provide the learner with one sample at a time. The samples are provided to the learner in a sequence. First, the learner receives the input of a sample; using the sample's input, the learner estimates the sample's target; then, the leaner receives the sample's true target; with information on the true target, the learner can perform a learning update; and the process repeats itself for the next sample. The online supervised learning task requires an input and a target for each sample. It is a task that reflects the demands of systems that receive data and feedback frequently. *Strictly incremental* learning algorithms, learning algorithms have a constant bound on the amount of computation used per sample and the amount of memory used by the learner during its operation (Sutton and Whitehead, 1993), are suited for online supervised learning tasks when the task can be arbitrarily long.

Nonstationary supervised learning tasks are supervised learning tasks where the sample distribution changes during the task. *Stationary* supervised learning tasks are supervised learning tasks where the sample distribution never changes. Nonstationary is a general word that refers to a large class of changes. To narrow down the type of sample distribution changes the learner experiences, the change scheme should be specified. Some nonstationary supervised learning tasks change the task once for demonstrative purposes (Srivastava et al., 2013; Goodfellow et al., 2013). Other nonstationary supervised learning tasks regularly change the distribution throughout the learner's operation. How much the distribution changes is also important to specify. The distribution can change gradually over time, abruptly, or both. Elwell and Polikar (2011) provide a clear review of some nonstationary tasks.

Offline supervised learning tasks are different from their online version. In the offline supervised learning task, the learner learns from a separate set of samples than the set the learner is tested on. Offline supervised learning tasks encourage learners to perform well on unseen data without learning from the data afterwards (Goodfellow, 2016). It is a task that reflects a system that does not receive feedback during crucial operating times. The system is not expected to learn from new samples. Offline supervised learning learning tasks are not relevant to this thesis, as we are interested in learners that receive feedback for every sample and can adapt to new samples.

Recent works have constructed a new type of task that has attributes of both online supervised learning tasks and offline supervised learning tasks. The tasks are known as *online continual learning tasks.* The descriptor "online continual learning" is misleading as the task is not *online* as defined by past *online learning* tasks where the learner receives a single sample at a time (Sutton and Whitehead, 1993; Hoi et al., 2018). In an online continual learning task, the learner receives a sequence of sample sets to train on, and each sample set may be sampled from a different distribution. In between each consecutive pair of training sample sets, the learner is tested for performance on a testing sample set. The testing sample sets can test the learner on sample distributions different from the training sample sets. For example, the learner can be tested for how well it performs on samples from all past sample distributions. The online continual learning task is not relevant to this thesis, because we are interested in tasks where the learner receives a stream of samples, not a stream of sample sets. Also, we are interested in systems that can learn from every sample it receives.

The experiments in this thesis test strictly incremental algorithms on online, nonstationary supervised learning tasks. The specific target function and nonstationary changes were determined by what the learner is being tested for. More details of how and what we test with our task are presented in Chapter 4.

2.3 Representations and Features

On supervised learning tasks, learners can use a *representation* to make accurate estimates. The representation transforms a sample's input into a summary that emphasizes the useful parts for estimation. For example, a system may be able to recognize an image has a cat in it due to the representation emphasizing triangular ears and slit pupils. Traditionally, representations were hand engineered based on expert knowledge or based on successful general properties. Human experts may know how to map each input pattern to its target, but on some tasks, like image recognition, human experts have been unable to handcraft useful representations. As a result, there has been a growing demand for learning representations.

There are many types of representations, but we focus on the representation within an *artificial neural network* (ANN). An ANN's representation consists of a fixed set of features where each feature takes input signals and outputs a scalar summary of the signals. Features in a shallow ANN only take in input from a sample. Features in a deep ANN can also take in output from other features. In this thesis, we focus on shallow artificial neural networks.

In a shallow ANN, the number of features $N \in \mathbb{N}$ is fixed, and the output of the representation is a vector $\mathbf{f} \in \mathbb{R}^N$. The output of the i^{th} feature f_i, which is also the i^{th} entry of $\mathbf{f}$, is calculated by the following expression:

$$f_i = \sigma(\mathbf{input} \cdot \mathbf{w}_i)$$

where $\mathbf{input} \in \mathbb{R}^m$ is the sample's input vector, $\mathbf{w}_i \in \mathbb{R}^m$ is the i^{th} feature's weight vector, and σ is a scalar-to-scalar function called the *activation function*. A feature is considered a semilinear function as there is a linear operation within the activation function that is typically nonlinear. The activation function can be almost any function, but common choices include: tanh, logistic ($\text{logistic}(x) = \frac{1}{1+e^{-x}}$), ReLU ($\text{ReLU}(x) = max(0, x)$), and SoftSign ($\text{SoftSign}(x) = \frac{x}{1+|x|}$). The functions are shown in Figure 2.1. Properly setting each feature's weight vector is crucial for useful representations.

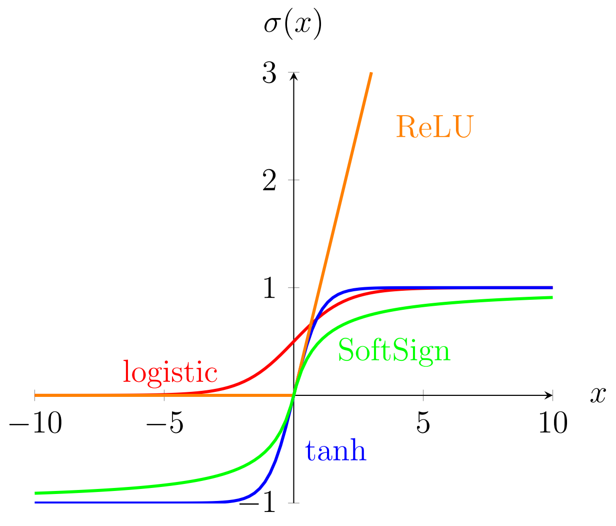

Figure 2.1: A few activation functions are shown.

The *output layer* of an artificial neural network uses the feature output vector $\mathbf{f}$ to make the final estimate. With useful features, the representation's output $\mathbf{f}$ is sufficiently descriptive and a linear function over $\mathbf{f}$ can yield accurate estimates. The output layer is this linear function, and it can be learned or fixed. The estimation output by the learner is retrieved from the feature outputs with the following expression:

$$\text{estimate} = w_0^o + \sum_{i=1}^{N} f_i w_i^o$$

where w_i^o for all $1 \leq i \leq N$ are the *output weights* and w_0^o is the *output bias*. A feature's *output weight* is the w_i^o that multiplies the feature's output f_i.

2.4 The Backpropagation Algorithm

One way to learn the weights of an artificial neural network is with the backpropagation algorithm. In this procedure, there are two phases: initializing the features with small weights and using samples to adjust the weights with *stochastic gradient descent*. First we will define the stochastic gradient descent steps and then review why the initialization step is important.

The algorithm for updating the weights with the gradient for each sample is known as stochastic gradient descent. For each sample, the gradient of the sample's error with respect to the weights is calculated. The gradient reveals how to change the weights to steeply increase the error if the weights were to be modified by an infinitesimally small amount. To approximately decrease the error with the gradient, a small-scaled weight update is made in the opposite direction of the gradient.

For a shallow ANN, and a mean squared error between the estimate and a sample's target, $(\text{target} - \text{estimate})^2$, stochastic gradient descent updates the i^{th} feature's weight vector $\mathbf{w}_i$ with the second update derived from the first:

$$\mathbf{w}_i \leftarrow \mathbf{w}_i + \alpha \frac{\partial (\text{target} - \text{estimate})^2}{\partial \mathbf{w}_i}$$

$$\mathbf{w}_i \leftarrow \mathbf{w}_i + 2\alpha(\text{target} - \text{estimate}) w_i^o \frac{d\sigma(\mathbf{w}_i \cdot \mathbf{input})}{d(\mathbf{w}_i \cdot \mathbf{input})} \mathbf{input}$$

where $\alpha \in \mathbb{R}^+$ is the step-size parameter. To update the output layer weights, stochastic gradient descent uses the following two updates:

$$w_i^o \leftarrow w_i^o + 2\alpha_o(\text{target} - \text{estimate})f_i \text{ for } 1 \leq i \leq N$$

$$w_0^o \leftarrow w_0^o + 2\alpha_o(\text{target} - \text{estimate})$$

where $\alpha_o \in \mathbb{R}^+$ is a step-size parameter that must be chosen.

The performance of stochastic gradient descent is heavily dependent on the step-size parameters. For large step sizes, the weights can change rapidly but cause poor generalization among nearby samples. Small step sizes can make learning slow. It is typical to either sweep over some possible step sizes to determine a decent step-size parameter or use a step-size adaptation algorithm that dynamically changes the step-size parameter with experience. Step-size adaptation algorithms are beyond the scope of this thesis, but are another direction of research that attempts to account for nonstationary tasks (Sutton, 1992; Schraudolph 1999; Veeriah et al., 2017; Jacobsen et al., 2019).

Note that in a 2-layer ANN, the features are updated independently from one another. These independent updates can be problematic if the weights are not initialized properly. If the feature vectors are initialized to be the same vector and the output weights are initialized uniformly, the features will all change in exactly the same way for each sample. By the end of the updates, all the features will be the same, and there was no point in having multiple features to begin with. Beginning with random weights is one way to begin with feature diversity and ensure the independent updates are less likely to result in a uniform feature set (Rumelhart et al., 1986).

Also note that the gradient update of the feature weights is scaled by the activation function's derivative $\frac{\partial \sigma(\mathbf{w}_i \cdot \mathbf{input})}{\partial \mathbf{w}_i \cdot \mathbf{input}}$. For activation functions like the tanh function and the logistic function, the largest derivative is when the function's input is near zero. The derivative exponentially decreases to zero when the magnitude of the function's input is large. To ensure the activation function derivatives begins large and unstable, the initial feature weights begin small for tanh and logistic activation functions (LeCun et al., 1998). Why the activation function derivatives begin high and unstable is discussed in the next chapter. The derivatives of the ReLU, logisitic and tanh activation functions are shown in Figure 2.2.

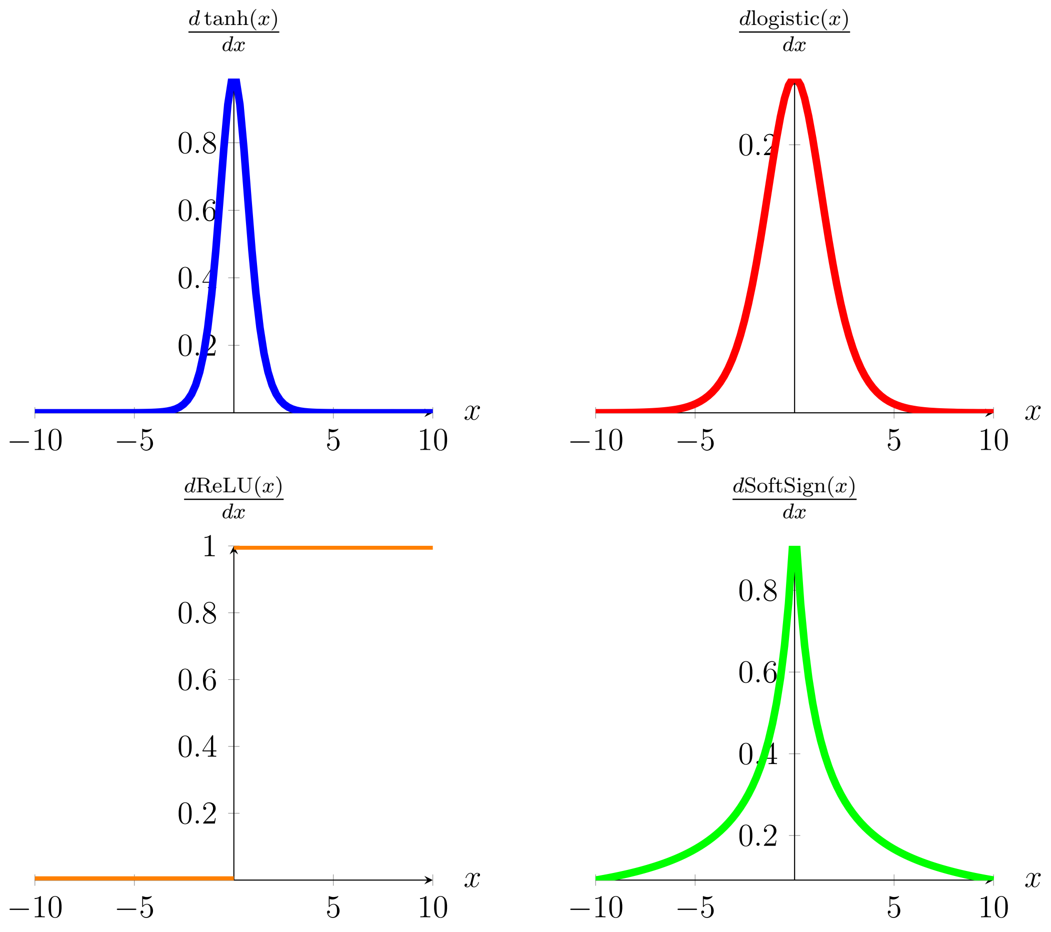

Figure 2.2: The derivatives of the tanh, logistic, ReLU, and SoftSign activation functions.

Chapter 3

Properties for Fast Feature Discovery

In this thesis, we investigate algorithms and their continual feature discovery performance. Algorithms for continual feature discovery should discover useful features for all new sample distributions the learner encounters. One of the algorithms we focus on is the backpropagation algorithm. The backpropagation algorithm consists of two phases: initialize the features with small, random weights, and use data to update the weights with stochastic gradient descent. Two artificial neural network (ANN) properties set by the initialization step of the backpropagation algorithm are: high, unstable activation function derivatives, and diverse features. This chapter is dedicated to investigating whether the ANN properties set by the initialization step affect how fast the backpropagation algorithm discovers useful features for a single sample distribution. These ANN properties will be referred to in Chapter 5, to determine how they change during a task where the learner must estimate targets from a sequence of sample distributions.

The backpropagation algorithm discovers useful features by first generating random features, and then gradually changing the weights of the features according to the estimation error. Features are considered useful when they yield low estimation error. The estimation error at the end of a stationary supervised learning task is a measure for the utility of features discovered with the limited amount of samples. Therefore, the error at the end of the task can also be viewed as a measure for how fast the learner discovered useful features where low error indicates fast discovery.

3.0.1 Activation Function Derivatives

The activation function derivative affects the speed of feature discovery with the backpropagation algorithm, because it scales how much the feature weights change with the stochastic gradient descent update. The activation function derivative is affected by the choice of the activation function and the input into the feature. More information about various activation functions can be found in the background (Chapter 2).

When using the backpropagation algorithm, it is frequently recommended to use activation functions such as tanh, logistic, or ReLU, and to initialize the feature weights with small numbers (Rumelhart et al., 1986; Glorot and Bengio, 2010; He et al., 2015). By using these activation functions and initializing the feature weights small, these works promote initializing ANNs with high, unstable activation function derivatives. When the input is zero, the derivative of a squashing function, like logistic, tanh, or SoftSign, is at its highest. When the input is near zero, the derivative is unstable for the mentioned squashing functions, because the derivative rapidly decreases for nearby points. The instability of the logistic activation function derivatives was empirically shown by Fahlman (1988). He showed the logistic activation function derivatives of many features became close to zero during learning. When the input to a ReLU is near zero, it is an unstable point since the derivative can change from 1 to 0 or from 0 to 1 if the weights are shifted by a small amount. Douglas and Yu (2018) empirically showed and theoretically discussed the tendency of ReLUs to *die*, which means the ReLU features remain with zero activation function derivative for all input.

Empirically, it has been shown that beginning with high, unstable activation function derivatives leads to better performance with the backpropagation algorithm. Thimm and Feisler (1997), demonstrated that large initial weights for logistic features (and therefore small, stable logistic activation function derivatives) led to higher error at the end of learning. He et al. (2015) carefully scaled the initial ReLU feature weights while keeping the weights small to quicken the discovery of useful features. This means the input for the ReLU activation functions would be near zero where the activation function derivatives can be high, but unstable.

There have been proposed solutions to ensure derivatives of activation functions do not become too small during a supervised learning task (Fahlman, 1988; Duch et al., 1997; LeCun, 1998; Ng et al., 2003; Ioffe and Szegdy, 2014). Fahlman (1988) showed simply adding 0.1 to the activation function derivative can speed up learning. For deep ANNs, Ioffe and Szegdy (2014) proposed centering and scaling the inputs of features to keep the activation function output centered around zero, and hence have high, unstable activation function derivatives. However, it is not well established whether strategies that improve feature discovery on stationary supervised learning tasks are useful for nonstationary supervised learning tasks. For example, Ng et al. (2003) proposed decaying the weights of the features to keep squashing activation functions from saturating, but Kirkpatrick et al. (2017) showed weight decay strategies can perform poorly on nonstationary supervised learning tasks. More work needs to be done to understand how to modify the activation function derivatives to reliably improve the speed of feature discovery and the accuracy of the discovered features.

It is worth noting the step sizes of the backpropagation algorithm can also scale the amount the feature weights change due to stochastic gradient descent updates. The step sizes are fixed for the backpropagation algorithm, but there have been proposed step-size adaptation algorithms that change the step sizes during learning (Sutton, 1992; Schraudolph 1999; Veeriah et al., 2017; Jacobsen et al., 2019). In this thesis, though, we perform a search over a set of possible fixed step sizes to determine them for each experiment. Since both the step sizes and the activation function derivatives scale how much a feature's weights change, it would be worth looking into whether one can be absorbed into the other. In our later experiment results (Chapter 5 and 6), we note how the product of a feature's step size and it's activation function derivative change during learning. The product gives a better sense of how fast the feature weights change and therefore how fast the features can be transformed into newly discovered features.

3.0.2 Feature Diversity

Feature diversity is another ANN property that is set at the initialization step and is considered important for fast feature discovery with the backpropagation algorithm. The level of feature diversity in an ANN is determined by the level of difference amongst the features and the number of features. An ANN with many varied features has high feature diversity. An ANN with few similar features has a low amount of feature diversity. However, the kind of feature diversity that is useful for quickly discovering useful features with the backpropagation algorithm cannot include features that change too slowly since these features are not useful for discovering new features quickly. When we talk about feature diversity, we are talking about the diversity of features with relatively high, unstable activation function derivatives across the sample input space, since we discussed earlier how these features can lead to discovering useful features quickly.

One common way to generate diverse features is by randomly generating features. Rumelhart et al. (1986) viewed initializing features with random weights as a way of “breaking symmetry” between features so that the features did not end up uniform after being updated with stochastic gradient descent. Blumenfeld et al. (2020) empirically showed that symmetry breaking did not have to occur at the beginning of learning, but can instead occur throughout learning by injecting random noise. We investigate feature diversity through random initialization as not only a way for breaking symmetry, but also as a way to discover useful features quicker.

Work relating to how the diversity of features affects the backpropagation algorithm’s speed of discovering useful features mainly came in the form of non-random initialization schemes. One non-random initialization scheme initializes features so they activate at different parts of the input space (Nguyen and Widrow, 1990). The scheme worked well with the backpropagation algorithm, and the feature discovery performance improved when more features were used. Another non-random initialization scheme uses random sample inputs and initializes each feature to activate for one of the sample inputs (Denoeux and Lengellé, 1993). This scheme allows the backpropagation algorithm to use information from the data stream to account for different parts of the input space.

Mahmood (2017) conjectured that randomly setting feature weights can spread ANN features across the feature space and make useful features easier to discover for feature search algorithms. We conjecture that the random spread of features can lead to faster feature discovery with the backpropagation algorithm in a similar way to the above-mentioned initialization scheme of Nguyen and Widrow (1990). In the following experiment, we investigated whether the similarity of randomly initialized features affects the quality of discovered features. We also look at whether the number of features affects the speed that useful features are discovered.

3.1 Initial Feature Diversity: Experiment

In this experiment, we investigated the effect of initial feature diversity on the backpropagation algorithm's ability to quickly discover useful features. The backpropagation will use a limited amount of samples and the error of the discovered features will determine the feature discovery performance. The exact scheme used for generating similar random features is described below.

The Task. All tested learners were evaluated on an online stationary supervised learning task. The task provides the learner with a 1000 samples. Each sample contains a vector input and a scalar target. The input vectors of the samples are randomly sampled size-5 binary vectors, and each sample's scalar target is the dot product of a fixed target vector and the sample's input. The fixed target vector is a size-5 vector randomly initialized with -1 and 1 elements. Given a sample's input, the learner estimates the sample's target. The mean squared error between the learner's estimate and the target was noted for each sample.

The Learners. Multiple artificial neural networks (ANNs) were evaluated on the online stationary supervised learning task. To remove the effect of nonlinear activation functions on the feature discovery performance, the features were linear (i.e. the features had an identity activation function). Each ANN had $m \in \{1, 2, 5, 10\}$ features. The feature weights were represented by a matrix $\mathbf{F} \in \mathbb{R}^{m \times 5}$, where 5 is the size of the input vector. The learner also had output weights $\mathbf{o} \in \mathbb{R}^m$ and a bias $o_b \in \mathbb{R}$ that were used to calculate an estimate. For a given sample with input $\mathbf{i} \in \{0, 1\}^5$, the estimate e was calculated with the following expression:

$$e = \mathbf{Fi} \cdot \mathbf{o} + o_b.$$

Every sample, the ANN weights and bias were updated with a stochastic gradient descent update to decrease the squared error. The feature weight update had a step-size parameter $\alpha_r \in \mathbb{R}^+$. The output weights and the output bias were updated with a step-size parameter $\alpha_o \in \mathbb{R}^+$. The parameters were swept over the sets $\alpha_r \in \{0.0, 0.00125, 0.0025, 0.005, 0.01, 0.02, 0.04, 0.08, 0.16, 0.32, 0.64, 1.28, 2.56\}$ and $\alpha_o \in \{0.00125/m, 0.0025/m, 0.005/m, 0.01/m, 0.02/m, 0.04/m, 0.08/m, 0.16/m, 0.32/m, 0.64/m, 1.28/m\}$. The results used the step sizes with the smallest error on the 1000^{th} sample averaged over 100 independent runs.

The features were initialized with a similarity score s where s was drawn from $\{0.0, 0.25, 0.5, 0.75, 1.0\}$. The similarity score controlled the initial feature diversity of the features. A similarity score of zero meant there was a lot of diversity among features, and a similarity score of one meant all feature weight vectors are equal to each another. Each column of $\mathbf{F}$ was randomly generated independently with a multivariate normal distribution with the covariance matrix $\mathbf{C} \in \mathbb{R}^{m \times m}$. The diagonal entries of $\mathbf{C}$ were equal 0.01, and all other entries of $\mathbf{C}$ were equal to $0.01s$. The initial 0.01 variance was chosen, because other small values for the variance did not make a difference in the results. Note, generating features with a similarity score of zero is how it is typically done in practice where feature weights are generated independently from one another. Therefore, in practice, the features are initialized with the most diversity they can have according to our initialization scheme.

For each combination of ANN size m and similarity s, we ran the learner for 1000 steps on the stationary supervised learning task. The results were averaged over 100 independent runs. We expected ANNs with more features to have lower error at the end of 1000 samples and ANNs with higher initial feature diversity to have lower error at the end of 1000 samples.

3.2 Initial Feature Diversity: Results

The parameter tuning with the backpropagation algorithm for different m-sized ANNs and initial similarity scores s resulted in the following step sizes (Table 3.1).

s \ m	1	2	5	10
0.0	$\alpha_o = 0.16$ $\boldsymbol{\alpha_r} = \mathbf{0.00125}$	$\alpha_o = 0.01/2$ $\alpha_r = 0.04$	$\alpha_o = 0.01/5$ $\alpha_r = 0.64$	$\alpha_o = 0.01/10$ $\alpha_r = 1.28$
0.25	-	$\alpha_o = 0.005/2$ $\alpha_r = 0.04$	$\alpha_o = 0.005/5$ $\alpha_r = 0.32$	$\alpha_o = 0.02/10$ $\alpha_r = 0.32$
0.5	-	$\alpha_o = 0.005/2$ $\alpha_r = 0.04$	$\alpha_o = 0.005/5$ $\alpha_r = 0.16$	$\alpha_o = 0.08/10$ $\alpha_r = 0.04$
0.75	-	$\alpha_o = 0.16/2$ $\alpha_r = 0.0025$	$\alpha_o = 0.005/5$ $\alpha_r = 0.16$	$\alpha_o = 0.01/10$ $\alpha_r = 0.16$
1.0	-	$\alpha_o = 0.01/2$ $\alpha_r = 0.02$	$\alpha_o = 0.08/5$ $\alpha_r = 0.01$	$\alpha_o = 0.01/10$ $\alpha_r = 0.16$

Table 3.1: Optimized step sizes.

The step size that is in bold is the only step size that was an extreme of the non-zero values in the search range. However, the error did not change much with some larger step sizes that were not extreme values.

The ANNs with multiple features had a low mean squared error at the end of the task when the initial similarity score was less than one (i.e. when the features did not begin uniform) (Figure 3.1). When the initial similarity score was one (i.e. the features began uniform) the mean squared error at the end of the task was similarly high across the ANN sizes. The learners had the lowest mean squared error at the end when the initial similarity score was zero compared to when the initial similarity score was higher. Generally, the higher the initial similarity score, the higher the error at the end of the task.

With the 2-feature ANN, the initial similarity score was correlated with the error at the end of the task (Figure 3.1). However, with initial similarity

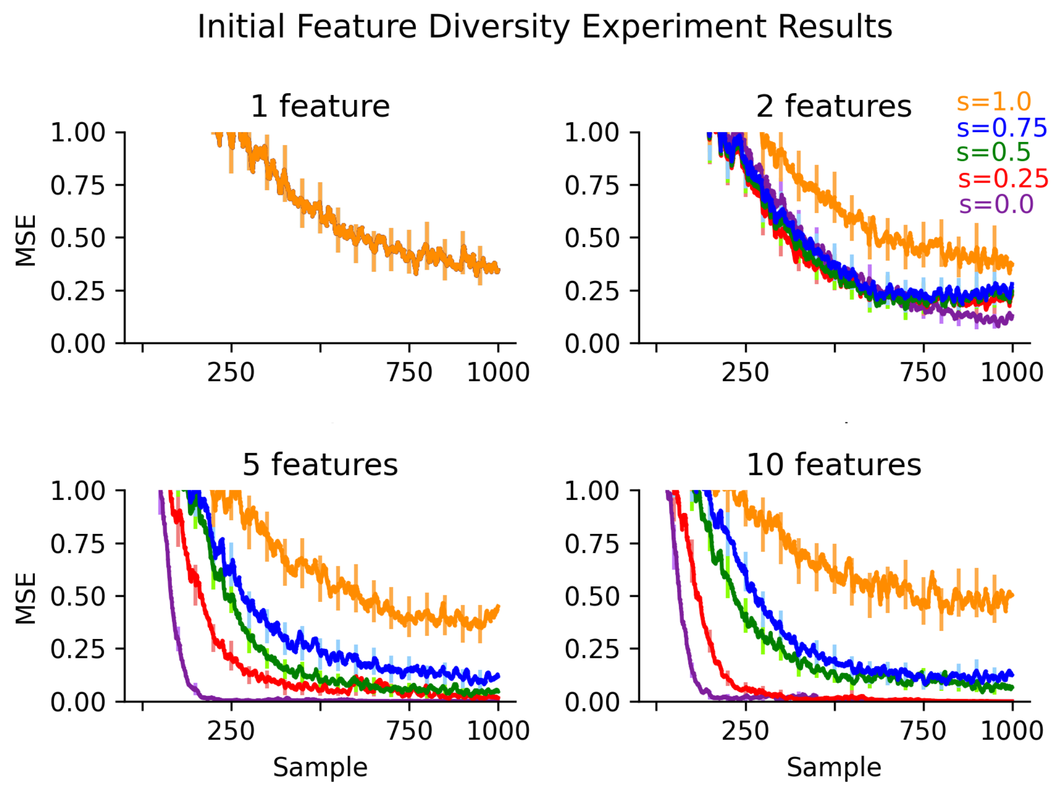

Figure 3.1: The plots show the mean squared errors over 1000 samples for different initial similarity scores s and different ANN sizes. The light-colored bars represent one standard error determined by 100 independent runs (plotted every 100^{th} sample). Since the backpropagation algorithm parameters were optimized for the last step, we focus on the error at the end of the task. ANNs that had more features and had more initial feature diversity tended to have lower error on the 1000^{th} sample.

scores 0.25, 0.5, and 0.75, there was little difference in error at the end. An initial similarity score of 0.0 led to zero error at the end when the ANN had 5 features or 10 features. An initial similarity score 0.25 led to zero error at the end when the ANN had 10 features. When the initial similarity score was 0.5 or 0.75, there was little difference in error from when the ANN had 5 features or 10 features.

3.3 Initial Feature Diversity: Conclusions

From our results, we found ANNs discover a more useful set of features when the ANN has more features. We consider the error on the 1000^{th} step a reflection of the usefulness of features discovered with the limited amount of samples. To discover useful features quickly, it is important to ensure the ANN begins with diverse features. With many features, a little initial feature diversity can significantly improve the usefulness of the discovered features.

This experiment used a small task, but from it we can see the initial feature diversity can affect the feature discovery performance. With an ANN with many features, the diversity among features can be low and the backpropagation algorithm can still discover a useful set of features quickly. However, if there are fewer features available to learn with, the backpropagation algorithm will require more initial diversity to discover useful features quickly. It is also important to note, the kind of initial feature diversity we are considering involves features with small random weights. This is a specific kind of diversity that is abundantly available at the beginning of a task with the backpropagation algorithm, but may not be as available during the course of learning.

3.4 Summary of Properties for Fast Learning

This chapter discussed two initial conditions that are important for the fast discovery of useful features with the backpropagation algorithm: high, unstable activation function derivatives and high feature diversity. High, unstable activation function derivatives have been empirically shown by past work to lead to fast feature discovery with the backpropagation algorithm. We empirically demonstrated the diversity of random features can affect how fast the backpropagation algorithm discovers useful features where more diversity can lead to better performance. This result is relevant to past work with non-random feature initialization strategies that also set a specific kind of feature diversity for faster discovery of useful features.

On a stationary supervised learning task, the backpropagation algorithm does not need to maintain these conditions for decent feature discovery performance. Once the network discovers useful features it does not need to change them, and these factors cease to be relevant. On a nonstationary supervised learning task, it may be important to maintain these conditions for learning from new sample distributions.

In Chapter 5, we revisit these properties for the fast discovery of useful features on the generic continual feature discovery task introduced in the next chapter. These factors will be measured to check whether they are maintained by the backpropagation algorithm throughout the learner's operation and whether the backpropagation algorithm's continual feature discovery performance is poor without them.

Chapter 4

Evaluation for Continual Feature Discovery

This thesis investigates how effective artificial neural network (ANN) learning algorithms are for discovering useful features for new sample distributions. We call the ability of a learner to discover useful features for all new sample distributions continual feature discovery, and continual feature discovery is an important ability for learners that expect to encounter arbitrarily many changes during a task. A contribution of this thesis is the *generic continual feature discovery* (GCFD) task that can be used to evaluate the continual feature discovery performance of an algorithm. This chapter is dedicated to describing the GCFD task. In later chapters, we use the GCFD task to evaluate the continual feature discovery performance of different ANN learning algorithms. Specifically, we test the backpropagation algorithm and generate-and-test algorithms for their continual feature discovery performance in two major experiments.

The GCFD task is an online nonstationary supervised learning task, but it differs from other online nonstationary supervised learning tasks that were used to test ANN learning algorithms. We sought out a generic regression task that tested for continual feature discovery and that could be learned with a fixed-size ANN. One task that is closely related is the GEOFF (GEneric Online

Feature Finding) challenge (Sutton, 2014). In the GEOFF challenge, the target function is a randomly initialized ANN, and the nonstationary aspect of the task comes from the regular change of the output weights of the target function. Since the feature weights of the target function never change, it is not necessary for the learner to continually discover useful features and it suffices to discover useful features once and update the output weights to account for future changes (Veeriah et al., 2017). In our task we also use an ANN target function, but regularly change all the weights so the learner must continually discover useful features. Sahoo et al. (2018) used a similar generic task where the features and number of features of the deep ANN target function change after a fixed interval of samples. The number of features in the target function changes to challenge the learning ANN to use more features to account for the changes. To test for a more basic form of continual feature discovery, we sought a task that can be learned with a fixed-size ANN. Also, the generic task used by Sahoo et al. (2018) was a classification task while we sought out a regression task. Another generic regression task we came across tested 1-layer ANNs (Martínez-Rego et al., 2011). The regression task that tested the 1-layer ANNs were simple, since 1-layer ANNs only use input signals as features and these features are not learned. Since the task does not challenge the learner to discover features, the task is not suitable as a continual feature discovery task.

There are many classification datasets that are used for testing ANNs on nonstationary supervised learning tasks (Hayes et al., 2018). Image classification datasets are commonly used to test learners, because some image classification tasks are difficult to learn without a sophisticated function approximator like an ANN. However, some image classification tasks, like the MNIST dataset and CIFAR10 dataset, are easy to learn with an ANN and provide impressive looking results that may not scale to more complex domains (Kemker et al., 2017). Real-world datasets can provide variety to check if the methods that work on a generic task work with real-world data, but a generic dataset can give more control in what the task tests for. To make the difficulty of our task more explicit and ensure we are evaluating a learner's continual feature discovery performance, we opt for a generic task where the target function's structure is explicit.

4.1 Description of the GCFD Task

The generic continual feature discovery (GCFD) task tests a learner's continual feature discovery, which is the learner's ability to discover useful features for all new sample distributions it encounters. To test for a learner's ability to learn useful features, we use an online nonstationary supervised learning task.

The GCFD task is online, because we are interested in strictly incremental learning algorithms in this thesis, and strictly incremental learning algorithms process one sample at a time. Also, an online task measures the performance on each sample, which provides a useful measure of how quickly the learner is learning from its experience.

The nonstationary aspect of the task is the regular abrupt change in the sample distribution. The sample distribution changes every $\lambda \in \mathbb{N}$ steps. The abrupt change clearly marks when the sample distribution changes for the researcher, although the learner is not explicitly notified of the change.

Each sample contains an input and a scalar target. The input is always a randomly sampled m-sized binary vector. The 2-layer artificial neural network target function itself contains n features known as *target features*. Given the t^{th} sample's input, $\mathbf{input}_t$, the sample's target, $target_t$, is calculated with the following expression:

$$target_t = \sum_{i=1}^{n} (w_i^o \sigma_i(\mathbf{w}_i \cdot \mathbf{input}_t))$$

where $\mathbf{w}_i \in \mathbb{R}^m$ is the input weight vector of the i^{th} target feature, $w_i^o \in \mathbb{R}$ is the i^{th} target feature's output weight, and σ_i is a binary activation function. All the target function weights are randomly initialized as -1 or 1 with equal probability, and

$$\sigma_i(x) = \begin{cases} 0, & x < \beta_i = (\text{\# of -1's in } \mathbf{w}_i) + \nu m \\ 1, & x \geq \beta_i. \end{cases}$$

Each feature that uses the above σ activation function is known as a linear threshold unit (LTU), and $\nu \in [0, 1]$ is a settable parameter (Sutton and Whitehead, 1993). Since the inputs are binary and the network's weights are -1 and 1, the LTU outputs 1 when ν proportion of the -1 and 1 pattern of the LTU's feature weight vector corresponds to the 0 and 1 pattern of the input vector.

To calculate the probability that an LTU outputs 1 given a random binary input vector, we use a Binomial random variable $B(m, 0.5)$ with m trials (i.e. the size of the input and LTU vectors) and 0.5 probability of a trial being successful (i.e. a given entry corresponds). The $P(B(m, 0.5) \geq \lceil \nu m \rceil)$ is the probability an LTU is equal to 1 with a random binary input vector. The ceiling operator is used, because we are working with discrete values and the binomial distribution is discrete. We set ν to 0.6 whenever the GCFD task is used. This is an acceptable choice for our experiments, because we use input binary vectors of size 7 making the probability a target feature outputs one $P(B(7, 0.5) \geq 0.6 * 7) \approx 0.227$. With two target features, this makes the probability at least one LTU outputs one for a random binary input vector approximately $1 - (1 - 0.227)^2 = 0.40$. If the target features output one too infrequently (e.g. the target features only output 0) or too frequently (e.g. the target features only output 1), the task can become trivial.

We divide the GCFD task into *subtasks*. A subtask of the GCFD task is an interval of $\lambda \in \mathbb{N}$ samples where the sample distribution is unchanged.

The learner is never explicitly notified when the subtask switches out or the regularity of the switches. In the literature, it is common to find what we call "subtasks" called "tasks" (Farquhar and Gal, 2018; Diaz-Rodriguez et al., 2018). We deviate from this nomenclature since we consider the GCFD task a single task with a changing sample distribution.

The task challenges the learner to quickly discover useful features for each new sample distribution. At the beginning of each subtask, the learner's estimation error is expected to shoot up as the learner is unprepared for this new subtask. If the interval λ is long enough for the learner to see every sample in the subtask interval, the learner can ideally discover features that account for each of the samples and end the subtask with zero error. This task does not include noise or other forms of irreducible error, so zero error can be achieved by discovering the proper features. If one included noise to the targets, the irreducible error of the noise would be the error threshold that indicates the learner discovered useful features for a subtask. More details about how to evaluate the learner and ensure the learner is being tested for continual feature discovery will be discussed in the next section.

4.2 Evaluation for Continual Feature Discovery

Averaging the error of the last $n \in \mathbb{N}$ samples of a subtask is a measure for how useful the discovered features were for accurate estimations. A high error means the discovered features were less useful for estimation than features that lead to a low estimation error. For more challenging subtasks, the target function can be made more complex and the interval λ can be made shorter.

Another metric for how useful the discovered features are for a subtask is the average error of all steps in the subtask. This metric weights the performance of features that were discovered early in a subtask just as much as the performance of features that were discovered later. This is a valid metric to consider, but we only measure the error at the end of the subtask to focus on how useful the features discovered by the end of the λ interval are.

To judge a feature learning algorithm on its continual feature discovery performance, we look at how consistently the algorithm had low error at the end of each subtask. An algorithm well-suited for continual feature discovery can discover useful features for all sample distributions. If a feature discovery algorithm has high error at the end of all subtasks, it indicates the algorithm may not be suited for continual feature discovery, since it did not find as useful features for the subtasks. The error at the end of early subtasks is more of an indication of how initial conditions set by the algorithm affect early performance. As an algorithm for continual learning can expect to experience arbitrarily many sample distributions, we are more focused on improving the performance on all later subtasks. As subtasks are not expected to vary in difficulty, we aim for continual feature discovery algorithms that consistently perform well on all later subtasks. It is recommended to run learners on the GCFD task for many subtasks to get a better view of the long term performance.

Whether a GCFD task is difficult enough is relative to the capacity of the learner. We wish to avoid a learner that can memorize a representation of all the possible target features and no longer needs to learn new features. At the same time, the learner should have enough memory capacity to be able to represent useful features for each subtask.

To ensure the learner is challenged to learn new features on each new subtask, the size of the sample input $i \in \mathbb{N}$ should be large. The sample input size determines how many unique target features are available for a subtask's target function. Let t be the number of features in the target function. The number of unique target features with -1 and 1 weights is 2^i and the number of unique target functions is $(2^i)^t = 2^{ti}$ where t is the number of target features in the target function. Note, the number of possible subtasks is much higher than the number of target features making it unlikely the learner will encounter the same subtask frequently.

However, the learner may frequently encounter a recently experienced target feature. Keeping the probability a feature appears low disincentivizes memorization for good performance. A calculation can be done to determine how often a target feature will show up on subsequent subtasks. $(\frac{2^i-1}{2^i})^t$ is the probability a feature does not appear in a subtask's target function which makes $1-(\frac{2^i-1}{2^i})^t$ the probability the feature does appear in a subtask's target function. In our experiments, we set $i = 7$ and $t = 2$ making the probability a given feature appears ~ 0.016.

Given the probability a feature appears p, we can calculate the expected number of subtasks before a feature reappears. If we consider the probability of a target feature from the current subtask reappearing in a subsequent subtask to be equal to p, then, in expectation, the target feature will reappear on the $\frac{1}{p}^{th}$ subtask after. This is because we consider the target feature reappearance a Benoulli process: $X_1, X_2, X_3, ...$, where X_i, $i \in \mathbb{N}$, is a random variable equal to one if the target feature reappears on the i^{th} subtask after ($P(X_i = 1) = p$). X_i is equal to zero otherwise ($P(X_i = 0) = 1-p$). For such a Bernoulli process, the expected earliest i when $X_i = 1$ is $i = 1/p$. So, for our experiments, where $p \sim 0.016$, target features are expected to reappear every ~ 64 subtasks.

4.3 Summary

In this chapter we introduced and defined the generic continual feature discovery (GCFD) task that will be used to challenge learning algorithms on their ability to continually discover useful features. We consider continual feature discovery important for learners that can always expect to encounter new experiences. The GCFD task should be challenging enough that the learner is expected to discover new features for each new subtask. However, each subtask should not require more memory capacity than the learner has. We consider a learner excellent for continual feature discovery when the learner has low average estimation error at the end of all later subtasks.

The GCFD task will be used to test the continual feature discovery performance of artificial neural networks (ANNs) learning with the backpropagation algorithm and with generate-and-test algorithms. The results and explanations of these experiments constitute the other main contributions of our investigations into continual feature discovery algorithms.

Chapter 5

Continual Feature Discovery with the Backpropagation Algorithm

In this thesis we are interested in algorithms for continual feature discovery, which is an algorithm's ability to discover useful features for each new sample distribution. This chapter investigates the backpropagation algorithm for continual feature discovery. The generic continual feature discovery (GCFD) task (Chapter 4) was used to evaluate the backpropagation algorithm's ability to discover useful features for all new sample distributions. The artificial neural network (ANN) properties known to affect the feature discovery performance (Chapter 3), the activation function derivatives and feature diversity, were measured to investigate how they change over the course of learning. Demonstrating and explaining the continual feature discovery performance of the backpropagation algorithm is the main focus of this chapter and is one of the contributions of this thesis.

The backpropagation algorithm is known to perform poorly for nonstationary tasks beyond online supervised learning tasks (Choy et al., 2002). The two main limitations are based on two different performance metrics: the estimation accuracy on past sample distributions after learning from new sample distributions (retention), and the estimation accuracy on new sample distributions (integration) (Mahmood, 2017). There is a trade-off between retention and integration with a fixed-memory agent. The trade-off arises because some memory must be dedicated to retention and some must be dedicated to integration. By studying integration and retention separately, the trade-off can be made explicit by the fixed memory allocated for each ability (French, 1999). In this thesis, we focus on integration by looking at how quickly algorithms discover useful features for new sample distributions.

There have been similar works that investigate the learning performance of the backpropagation algorithm on online nonstationary supervised learning tasks. One work investigated the backpropagation algorithm's performance when training deep artificial neural networks (ANNs) on an online nonstationary supervised learning task (Sahoo et al., 2018). They demonstrated that the backpropagation algorithm ends with higher error on new, difficult sample distributions and attributed the performance decrease to some known issues when training deep ANNs. They noted these issues are known to arise when training deep ANNs on stationary supervised learning tasks and these issues were expected to show up on online nonstationary supervised learning tasks. The authors did not discuss any poor performance from training shallow ANNs with the backpropagation algorithm. Our experiments are used to investigate whether shallow ANNs can also have poor performance with the backpropagation algorithm.

Martínez-Rego et al. (2011) demonstrated the poor performance of a 1-layer ANN learning with a modified backpropagation algorithm (1-layer backprop) on online nonstationary supervised learning tasks. For their regression tasks, the 1-layer ANNs were linear and did not have learnable features. The input signals were mapped directly to the estimate. They showed that the ANNs with the 1-layer backprop had higher errors on new sample distributions and increasing errors on gradually changing distributions. The work is somewhat relevant to our research as the 1-layer ANN is like the output layer of our experiment's 2-layer ANNs. However, we emphasize algorithms that discover useful features instead of ones that use fixed features.

We measured two factors known to affect the speed of discovering useful features: the activation function derivatives and the feature diversity. Recall, both high, unstable activation function derivatives and high feature diversity are important for quick discovery of useful features with the backpropagation algorithm (Chapter 3). If these properties are lost during learning, when the agent encounters new sample distributions, the speed at which the agent discovers useful features for new sample distributions may decrease.

Fahlman (1988) found the activation function derivatives can decrease over time with the backpropagation algorithm. We investigate whether a similar decrease occurs on an online nonstationary supervised learning task. Fahlman's (1988) fix for the lowered activation function derivatives was to add 0.1 to the activation function's derivative so it was never below 0.1. To investigate what happens when the activation function derivative does not become low quickly, we tested an activation function that takes more iterations than sigmoid activation functions to near 0.0 known as SoftSign (Elliott, 1993; Glorot and Bengio, 2010).

5.1 Experiment Description

In this experiment, we investigate the backpropagation algorithm's ability to continually discover useful features. Each new sample distribution will provide the learner with a limited amount of samples. To be fit for continual feature discovery, the backpropagation algorithm must discover useful features for each sample distribution.

The Task. To test the backpropagation algorithm for continual feature discovery, we used the generic continual feature discovery (GCFD) task from

Chapter 4. The specific version of the task we used has size-7 inputs, two target features, 15,000 samples per subtask, and 4,000 subtasks (60 million samples). The size-7 input means there are 128 input configurations possible. The learner is provided each input configuration $\sim$117 times in expectation per subtask.

The Learners. The learners were 2-layer artificial neural networks (ANNs) updated with the backpropagation algorithm. The ANN structure and the backpropagation algorithm are outlined in the Background chapter. We tested ANNs of different sizes and activation functions. Each ANN had N features where N was either 15 or 40. Each ANN also had logistic, tanh, ReLU, or SoftSign activation functions. ReLU, logistic, and tanh activation functions are common in research and practice, but the SoftSign is considered less often (Nwankpa et al., 2018). Each combination of the number of features and activation function was tested. Note, the number of features in the learning ANN was large compared to the two target features in the target function giving the agent ample memory resources to represent useful features. To select the representation step size α_r and the output layer step size α_o for the backpropagation algorithm, we chose the best performing step sizes on a shorter GCFD task. The shorter GCFD task has 30 subtasks (450,000 samples) but all other properties the same as the longer GCFD task. For each learner, we used the step sizes that had the lowest average error over the last 1,000 samples and over 20 independent runs on the shorter GCFD task. We used this shorter task to use less computation in the step-size parameter search, while still choosing step sizes that can perform well on some distribution changes. Also, there is no set operation time in a continual supervised learning task, so the goal was not to optimize the step size for the specific 4,000th subtask. The step-size parameter search was over the following possibilities.

- α_o: $\{0.00625/N, 0.0125/N, 0.025/N, 0.05/N, 0.1/N, 0.2/N, 0.4/N, 0.8/N, 1.6/N, 3.2/N\}$
- α_r: $\{0.0, 0.0015625, 0.003125, 0.00625, 0.0125, 0.025, 0.05, 0.1, 0.2, 0.4, 0.8, 1.6, 3.2\}$

We focused on one aspect of feature diversity: the number of features the backpropagation algorithm can update reasonably quick. This measure of diversity is relevant to our challenge of continual feature discovery since the backpropagation algorithm discovers features by changing the weights of the features. Also, we and others have found the backpropagation algorithm is quicker to discover a useful set of features when the ANN has more features (Mahmood and Sutton, 2013; Chapter 4). To be a feature that is reasonably quick to update, the product of the feature's activation function derivative and the representation step size should be reasonably high across the sample input distribution. We call a feature *calcified* if the average mentioned product across the input distribution is less than 0.0001. We consider the number of features that are not calcified to be a measure of diversity. Fewer calcified features means the agent has more features to adjust weights of at a moderate or fast speed. We measured the proportion of calcified features at the beginning of each subtask.

Other metrics of feature diversity such as the correlation or covariance between feature outputs are often discussed to reduce the problem of *overfitting*, poor performance on samples the agent has not learned with (Srivastava et al., 2014; Cogswell, 2016). These metrics of feature diversity are yet to be shown as relevant for feature discovery. During learning, the ANNs weights can enlarge and vary in magnitude. Two features may have low correlation, but also small activation function derivatives and step sizes, making these features slow to learn with. By measuring the proportion of features that are calcified, we are conservatively measuring how many features are slow to learn with and therefore should not be considered the kind of feature diversity that leads to fast discovery of useful features.

For each ANN and its tuned backpropagation algorithm, results were gathered for 50 independent runs on the 4000-subtask GCFD task. Each ANN learning with the backpropagation algorithm was compared to the performance of a fixed representation version that had the representation step size α_r set to zero. The results included the average error on the last 1000 samples of each subtask, mean squared errors for some subtasks (subtasks 1, 30, 100, 1000, and 4000), the average activation function derivative at the beginning of subtasks, and the proportion of calcified features at the beginning of subtasks.

The experiments showed whether the backpropagation algorithm could be used to train the ANNs to have low error at the end of each subtask on the GCFD task. Having low errors would indicate the backpropagation algorithm discovered useful features. We call the average error of the last 1,000 samples of a subtask the *subtask final error* (SFE). The results also showed whether the the backpropagation algorithm maintained high, unstable activation function derivatives and a low proportion of calcified features at the beginning of each subtask. From these results, we determine if the backpropagation algorithm is suitable for continual feature discovery, and if the properties that promote fast feature discovery are maintained throughout learning.

5.2 Results

We refer to the ANNs that update with the backpropagation algorithm by their activation function (e.g. ReLU ANNs) and explicitly note if we are referring to the fixed representation version.

The chosen step sizes used for the backpropagation fell between the extremes of the search ranges. The step sizes that were set for each activation function and size are listed in Table 5.1. Larger and smaller step sizes than the chosen step sizes resulted in higher error.

activation function (15 features)	logistic	tanh	ReLU	SoftSign
α_r	0.2	0.025	0.05	0.1
α_o	0.4/15	0.2/15	0.05/15	0.8/15

activation function (40 features)	logistic	tanh	ReLU	SoftSign
α_r	1.6	0.2	0.0125	0.2
α_o	0.2/40	0.4/40	0.4/40	0.8/40

Table 5.1: Optimized step sizes.

For each ANN, the rate the mean squared errors decreased differed across subtasks. All ANNs except the size-15 ReLU ANN had lower mean squared errors at the end of the 30th subtask, the subtask the step-size parameters were optimized for, compared to those at the end of the first subtask (Figures 5.1 and 5.2). For all ANNs, the mean squared error decrease on the 1,000th subtask was similar to the mean squared error decrease on 4,000th subtask. With the exception of the SoftSign ANNs, there were noticeably higher errors throughout the 1000th and 4000th subtasks compared to the earlier subtasks that had lower errors throughout.

The fixed representation ANNs had SFEs around 0.15 or greater for each subtask (Figure 5.3).

The SFEs of the ReLU ANNs significantly increased over subtasks (Figure 5.3). On early subtasks, the ReLU ANNs had SFEs significantly lower than the fixed representation ANNs'. However, after a few hundred subtasks, the ReLU ANNs had higher SFEs than those with the fixed representation ANNs. The SFEs of the ReLU ANNs were approximately between 0.3 and 0.4 for all later subtasks.

The logistic and tanh ANNs had SFEs that increased over subtasks, but were significantly lower than the SFEs of the ReLU ANNs on later subtasks

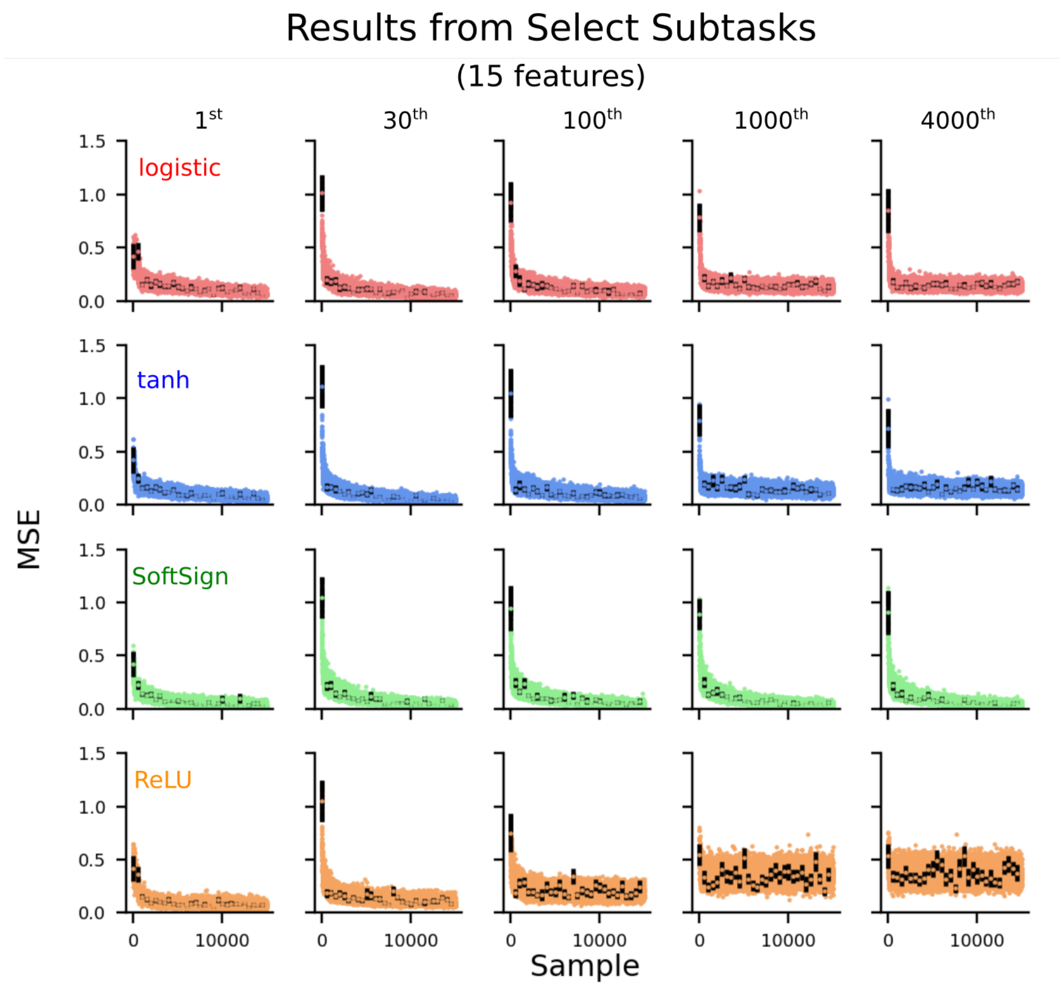

Figure 5.1: These are the average mean squared errors (MSEs) from selected subtasks. The results are for size-15 ANNs, while Figure 5.2 has results for the size-40 ANNs. Each black bar represents one standard error determined by 50 runs, and they are plotted every 500 subtasks. Except with the SoftSign ANN, later subtasks tended to have noticeably higher MSEs across samples compared to the MSEs on earlier subtasks.

(Figure 5.3). The logistic and tanh ANNs' SFEs were similar or slightly lower than the SFEs of the fixed representation ANNs. On earlier subtasks, the SFEs of the logistic and tanh ANNs were significantly lower than the SFEs of the fixed representation ANNs. Then, the SFEs increased significantly within a few hundred subtasks. On later subtasks, the 40-feature logistic and tanh ANNs had lower SFEs than those of their respective 15-feature versions.

The SoftSign ANNs had a less significant increase in the SFEs over subtasks

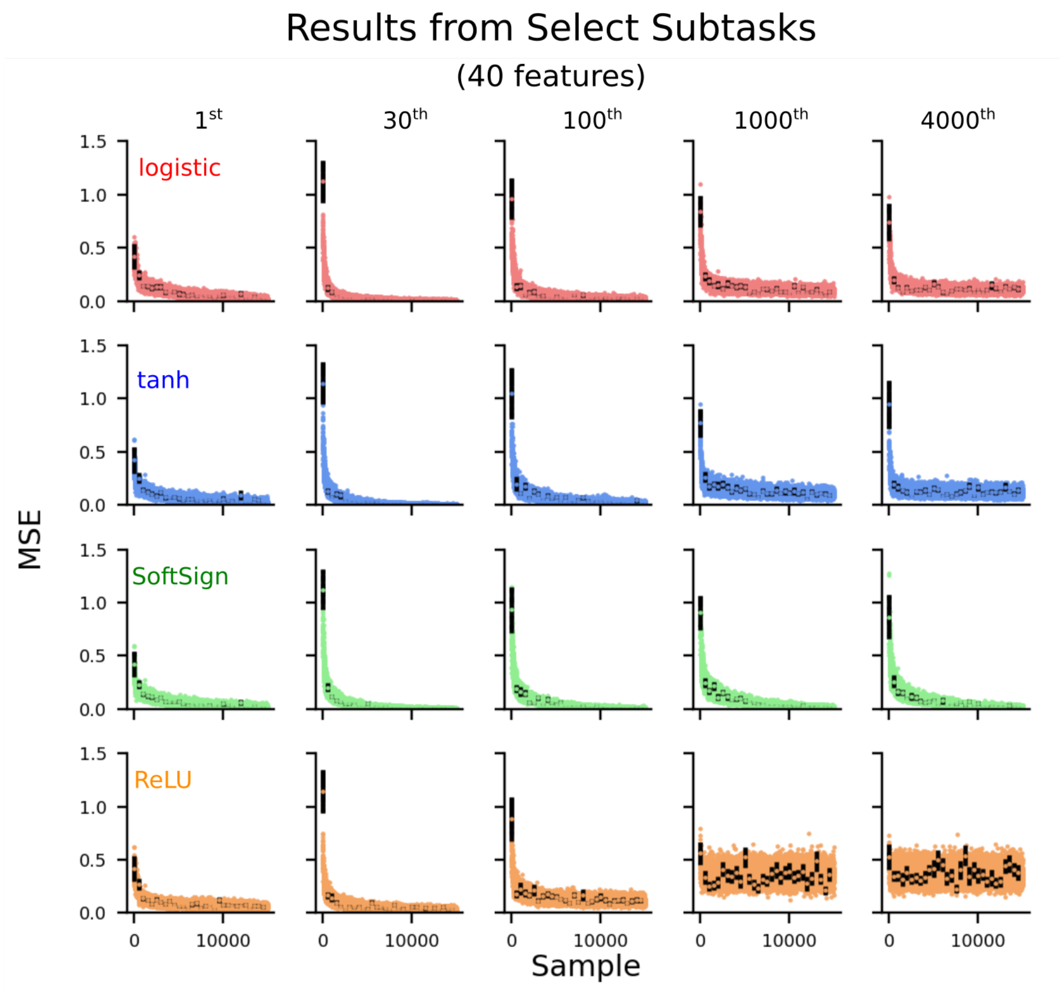

Figure 5.2: These are the average mean squared errors (MSEs) from selected subtasks for size-40 ANNs. Each black bar represents one standard error determined by 50 runs, and they are plotted every 500 subtasks. The results here are similar to the results for size-15 ANNs in Figure 5.1. However, there were noticeably lower errors on early subtasks with the size-40 ANNs than with the size-15 ANNs.

than the other ANNs (Figure 5.3). There was an increase in SFEs over subtasks with the SoftSign ANNs, but the SFEs on later subtasks remained around or lower than 0.05. The 40-feature SoftSign ANN had lower SFEs than the SFEs of the 15-feature SoftSign ANN.

The average activation function derivative at the beginning of subtasks decreased for all ANNs (Figure 5.4). The decrease was most significant with the ReLU ANNs where the average activation function derivative at the beginning

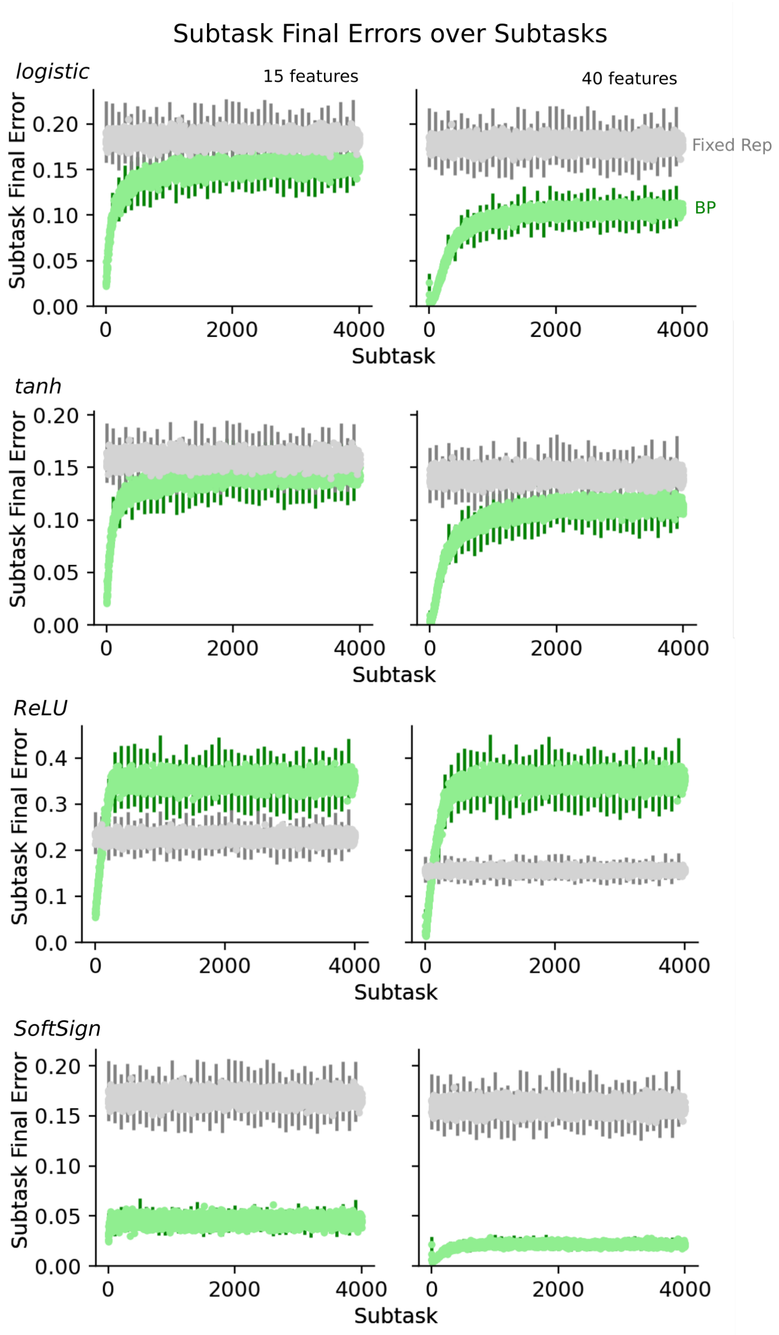

Figure 5.3: The left plots show results for 15-feature ANNs, and the right plots show results for 40-feature ANNs on the GCFD task. The top left of each row shows the activation function for the ANN. The *subtask final error* (SFE) is the average error of the last 1000 samples of a subtask. The dark-colored bars represent one standard error over 50 independent runs and are shown every 100 subtasks. The fixed representation typically had SFEs around or greater than 0.15 for all subtasks. The SFEs increased over subtasks for all ANNs trained with the backpropagation algorithm.

of a subtask became zero for all subtasks after a few hundred subtasks. A zero average activation function derivative means all the ReLU features were outputting zero, and only the output bias weight was being updated. The average activation function derivative at the beginning of the subtasks decreased by an order of magnitude for the the logistic and tanh ANNs. The average activation function derivative at the beginning of subtasks decreased the least for the SoftSign ANNs.

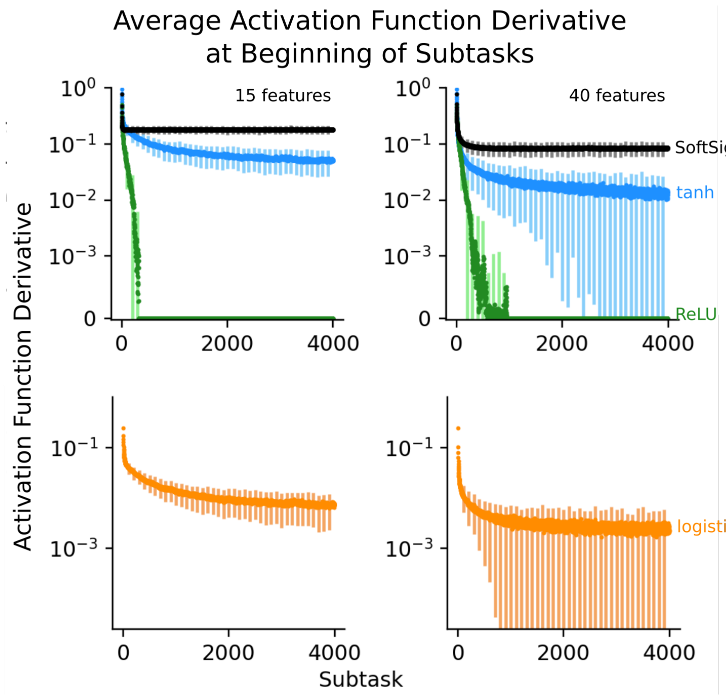

Figure 5.4: These plots show the average activation function derivative at the beginning of subtasks on the GCFD task. The top plot is a semilog plot that is log scaled for all points above 10^{-4} and is linearly scaled for all points below. Each curve represents a different ANN. The light-colored bars represent one standard error over 50 runs, and they are plotted every 100 subtasks. The activation function derivative at the beginning of subtasks decreased rapidly over subtasks, but stabilized after some subtasks. The average ReLU derivative went down to 0 and never recovered.

For the SoftSign, tanh, and logistic ANNs, the average activation function derivative multiplied by the representation step size was similar or higher with 40-feature ANNs than with 15-feature ANNs (Figure 5.5). However, with these activation functions, the average activation function derivative at the beginning of subtasks were higher for 15-feature ANNs than for 40-feature ANNs (Figure 5.4). This showed the higher step sizes picked during the parameter search for the 40-feature ANNs made the product between activation function derivative and the step size more comparable between differently sized ANNs of the same activation function.

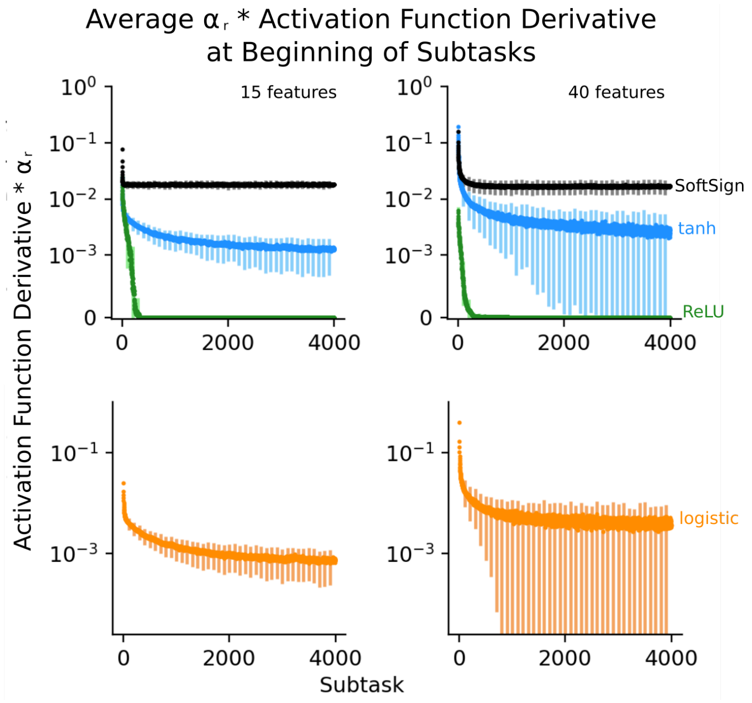

Figure 5.5: These plots show the average activation function derivative multiplied by the representation step size at the beginning of subtasks on the generic continual feature discovery task. The presentation is similar to Figure 5.4. The product of the activation function derivative and the step size was higher or similar for the 40-feature ANNs than their 15-feature versions. The ReLU ANNs are exceptions to this observation.

The proportion of calcified features at the beginning of subtasks increased for all ANNs except for the SoftSign ANNs (Figure 5.6). The SoftSign ANNs had no calcified features at the beginning of each subtask. The ReLU ANNs had only calcified features at beginning of all later subtasks. The tanh and logistic activation function ANNs had an increase in calcified features at the beginning of early subtask, but the increase was far less dramatic than the increase with the ReLU ANNs. Even though the product of the average activation function derivative and step size at the beginning of subtasks were comparable between the differently sized tanh and logistic ANNs, the proportion of calcified features at the beginning of subtasks, which uses the product to determine if a feature is calcified, was higher for the 40-feature versions.

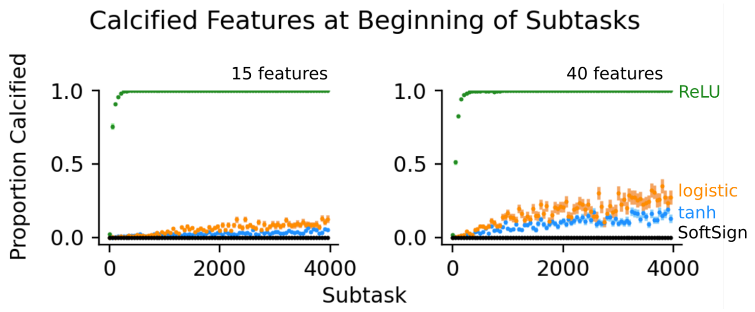

Figure 5.6: These plots show the average proportion of calcified features at the beginning of every 50th subtask for different ANNs. The light-colored bars represent one standard error determined by 50 independent runs. The proportion of calcified features at the beginning of subtasks increased for all the ANNs except for the SoftSign ANNs that had zero calcified features at the beginning of each subtask.

5.3 Conclusions

All the ANNs learning with the backpropagation algorithm were capable of discovering useful feature quickly on early subtasks of the GCFD task, but depending on the activation function, the ANNs experienced different levels of feature discovery slowdown on new sample distributions. The subtask final error on a subtask is a measure of how quickly the ANN discovered useful features for the subtask. The ReLU ANNs had the most severe slowdown; the logistic and tanh ANNs had a significant slowdown, but not as significant as that of the ReLU ANNs; and the SoftSign ANNs had the least amount of slowdown.

From the mean squared error curves, we see the feature discovery stagnated early on in the 1,000th and 4,000th tasks for the logistic, tanh, and ReLU

ANNs. The mean squared error did not decrease much after the initial samples showing how the feature discovery stagnated in finding useful features.

Using the backpropagation algorithm to train ReLU ANNs can lead to no discovery of useful feature for new sample distributions. Since the average activation function derivative was zero at the beginning of later subtasks, the ReLU ANNs could no longer update its features, and all the features were outputting zero. Essentially, the representation consisted of only the bias unit. The subtask final error significantly exceeded the fixed representation ANNs making the backpropagation algorithm a detriment to the feature discovery performance of ReLU ANNs.

For the logistic and tanh ANNs, the speed feature discovery decreased over subtasks. On later subtasks, the subtask final errors of the logistic and tanh ANNs were close to the subtask final errors of the fixed random representations. Having subtask final errors close to those of the fixed representation indicates the discovery of useful features is slow. However, for the logistic and tanh ANNs, there was still some level of useful features discovered on later subtasks.

The SoftSign ANNs had the lowest amount of slowdown when discovering useful features for new sample distributions. There was still an evident slowdown since the subtask final errors increased over subtasks, but the subtask final errors were much lower than those with the fixed representation ANNs. Also, the SoftSign ANN had no calcified features at the beginning of each subtasks meaning it maintained diversity according to our loose measure of diversity. To see whether the feature diversity changes with the SoftSign ANNs may require a tighter measure of diversity.

Early performance can deceive one into thinking an algorithm is suitable for a task that requires continual learning when it may not be. The ANN learning performance on early GCFD subtasks was not reflective of how fast the ANNs learned on later subtasks. All the ANNs learnt quickly on early subtasks, but the decrease in the speed of feature discovery was rapid. Within the first 1000 subtasks, the subtask final errors increased to a higher amount that was sustained over the rest of the subtasks. In a task that requires continual learning, the initial performance is fleeting compared to the performance on later parts of the learner's long operation.

The choice of activation function had a large impact on the backpropagation algorithm's ability to discover useful features for new sample distributions. Activation functions that have sharper drops in the activation function derivative over their domain (ReLU, tanh, and logistic) led to more slowdown than with the SoftSign activation function, which has a shallower change in activation function derivative over its domain. ReLU ANNs had especially large slowdown as they did not recover from all their features being calcified and zeroed out.

Overall, the backpropagation algorithm can have poor continual feature discovery. From our experiments, the performance is more clear with the tanh, logistic, and ReLU ANNs. The SoftSign ANNs had less of a slowdown in discovering useful features for new sample distributions, but still experienced some form of slowdown.

Feature diversity, in terms of number of features the ANN has, did affect the continual feature discovery performance. With logistic, tanh and SoftSign activation functions, ANNs with more features had lower SFEs than those from ANNs with fewer features. Whether the increase in calcified features at the beginning of subtasks further slowed down the discovery of useful features on later subtasks is not clear. However, how fast features can be adjusted (as represented by the product of the average activation function derivative and the step size) at the beginning of subtasks significantly decreased over subtasks with all ANNs. This decrease would greatly affect the performance on later subtasks.

In the next chapter, we seek to improve the backpropagation algorithm's continual feature discovery performance by introducing features with small, random weights during the learner's operation. These introduced features are expected to introduce ANN conditions known to improve feature discovery speed: feature diversity and high, unstable activation function derivatives.

Chapter 6

Continual Feature Discovery with Generate-and-Test Algorithms

In this thesis, we are focused on investigating algorithms for continual feature discovery. After determining the backpropagation algorithm can be poor at discovering useful features for new sample distributions (Chapter 5), in this chapter, we propose the use of generate-and-test algorithms to improve the backpropagation algorithm's continual feature discovery performance. Investigating whether generate-and-test algorithms can improve the backpropagation algorithm's continual feature discovery performance is one of the contributions in this thesis.

Generate-and-test algorithms work by replacing the least useful parts of a function with potentially more useful parts. They have been used for a variety of purposes including for learning Boolean functions (Kaelbling, 1990) and for learning rule-based systems (Booker et al., 1989). For representations, generate-and-test algorithms replace the least useful features of a representation with newly generated features. A feature's utility is relative to the task. For continual feature discovery, useful features contribute to accurate estimates on new sample distributions or expedite the discovery of useful features for new sample distributions.

When it comes to learning on online supervised learning tasks, there have been few strictly incremental generate-and-test algorithms. The strictly incremental generate-and-test algorithms we use are based on those proposed by Mahmood and Sutton (2013). They showed the backpropagation algorithm's performance can be improved with generate-and-test algorithms on an online stationary supervised learning task.

We investigate whether generate-and-test algorithms can improve the backpropagation algorithm's performance on the generic continual feature discovery (GCFD) task described in Chapter 4. The reason we used generate-and-test algorithms with the backpropagation algorithm is because the backpropagation algorithm's feature discovery performance is sensitive to properties of the ANN that are not maintained during learning (Chapter 5). ANNs that update with the backpropagation algorithm and are initialized with both high, unstable activation function derivatives and high feature diversity can discover useful features quickly for a single sample distribution (Chapter 3). To carefully reintroduce distinct features that have high, unstable activation function derivatives, we used generate-and-test algorithms.

Recall, we measure feature diversity based on the proportion of calcified features (Chapter 5). A feature is considered calcified if the average activation function derivative multiplied by the representation step size over the input distribution is lower than 0.0001. A feature that is calcified will slowly transform with the stochastic gradient descent updates of the backpropagation algorithm. More calcified features implies there are fewer features to update quickly to discover new features with. The number of non-calcified features is an upper bound measure for the amount of feature diversity that is useful for quick feature discovery.

There have been efforts to improve the backpropagation algorithm's performance on online supervised learning tasks. Step-size adaptation algorithms assign a step size to each ANN weight and change them during learning (Sutton, 1992; Kingma and Ba, 2015; Jacobsen et al., 2019). Stochastic Meta Descent is an example of a step size adaptation algorithm that can be used to preserve useful features and learn new features (Veeriah et al., 2017).

Other efforts for improving the backpropagation algorithm include changing the architecture of the ANN online. There have been networks that grow one feature at a time (Fahlman and Lebiere, 1990; Fu et al., 1996; Draelos et al., 2017). By freezing the weights of other features, the network ensures old features are not overwritten. Growing a network may improve continual feature discovery as it also introduces new, distinct features that can begin with high, unstable activation function derivatives (Fu et al., 1996). For continual feature discovery, generate-and-test algorithms are more appropriate than ANNs that grow, since generate-and-test algorithms do not increase memory usage. Also, unlike many of these mentioned works, we do not investigate the algorithm's performance on past distributions, because the performance on new distributions is more relevant for continual feature discovery.

Other works have gone in the opposite direction of adding features by removing weights. Pruning methods are used to learn a sparser ANN structure (Pérez-Sánchez, 2018; Evci et al., 2019). One pruning method removes the weights with the least magnitude and randomly makes new connections to learn a sparse ANN structure (Evci et al., 2019). These techniques are motivated by results that show a fully-connected ANN can have many weights that are dispensable and do not contribute to the final performance (Frankle and Carbin, 2019). We hold a similar philosophy to these pruning methods, and we investigate whether there are dispensable features in an ANN during learning that can be replaced with potentially useful features. Unlike generate-and-test algorithms, pruning an ANN structure operates on individual weights, not individual features. Also, our goal is not to reduce the number of connections in the network, but to use the feature capacity effectively.

6.1 Generate-and-Test Algorithms

In this section, we define the generate-and-test algorithms we used in our experiments. The algorithms are based on the generate-and-test algorithms presented by Mahmood and Sutton (2013). Generate-and-test algorithms can update feature weights every sample. They have a replacement-rate parameter ρ that must be set. The replacement rate $\rho \in [0, 1]$ is the proportion of features to replace each update. The features that are to be replaced are those that have the least utility according to the *tester*. The features that replace the least utility features are generated by the *generator*. If $\rho N < 1$, where N is the number of features in the ANN, ρN is the probability of replacing one feature on an update. In our experiments, we set ρ to 0.0001. This ρ was not tuned and was chosen, because it is an infrequent replacement rate. With 15 features, a ρ of 0.0001 means 3 features are expected to be replaced over 2,000 samples.

The generator of our generate-and-test algorithms generates features with weights sampled from a normal distribution with a variance of 0.01. The variance of 0.01 is the same variance we use to initialize the random feature weights of an ANN with the backpropagation algorithm. By regularly injecting features with small, random weights, we conjectured the backpropagation algorithm can quickly adapt these new features for new sample distributions.

The two generate-and-test algorithms we study are *random generate and test* and *partial random replace*. They differ in their testers. Random generate and test determines the utility of a feature by the magnitude of its output weight. This heuristic prefers retaining features with high absolute output weights since these features contribute more magnitude to an estimate when activated (Mahmood and Sutton, 2013). Partial random replace's tester treats all features as equally useful, and hence, it always randomly chooses the features to replace. Partial random replace is a baseline testing algorithm that we used to test random generate and test's tester.

To prevent newly generated features from being replaced before learning an output weight, one can set an age parameter that notes the number of samples a feature must update with before being considered for replacement. As our replacement rate ρ is very infrequent, we set the age parameter to zero.

Both generate-and-test algorithms are computationally cheap. To generate features is cheap, and to find the lowest ρN output weight magnitudes of N features in random generate and test can be done with $O(N)$ computation.

6.2 Experiment Description

We investigated whether the generate-and-test algorithms can improve the continual feature discovery performance of the backpropagation algorithm.

The Task. To evaluate continual feature discovery, we used the same generic continual feature discovery (GCFD) task from the preceding chapter. In this task, the target function has 2 target features, the inputs are size-7 binary vectors, each subtask lasts for 15,000 samples, and there are 4,000 subtasks in sequence.

The Agents. Multiple ANN architectures were examined. Each architecture had either 15 or 40 features, and each had tanh, logistic, ReLU, or SoftSign activation functions. These are the same ANNs tested in the previous chapter.

We compared five different algorithms: the backpropagation algorithm on its own (BP); the backpropagation algorithm with partial random replace (BP+PRR); the backpropagation algorithm with random generate and test (BP+RGT); partial random replace on its own (PRR); and random generate and test on its own (RGT). The learning algorithms that include the backpropagation algorithm used the tuned step-size parameters from the experiment in Chapter 5. The generate and test update is applied after the backpropagation algorithm's update for each sample.

For each combination of learning algorithm and ANN, the average error of the last 1000 samples in each of the 4000 subtasks was measured. We call the average error of the last 1000 samples of a subtask the *subtask final error* (SFE). The SFE is a measure of the usefulness of the features discovered for a sample distribution with a limited number of samples. An algorithm that has a low SFE on all subtasks has excellent continual feature discovery performance on the task, since it can discover useful features for new sample distributions.

We expected the performance of RGT and PRR to be similar to the performance of a fixed random representation, because the generate-and-test algorithms on their own perform infrequent updates. We expected ANNs trained with BP+RGT and BP+PRR to have a lower SFEs on later subtasks compared to those trained with BP, because of the above mentioned potential interaction between the backpropagation algorithm and the generate-and-test algorithms. We expected BP+RGT to perform better than BP+PRR since BP+PRR has no heuristic for judging a feature's utility and may discard useful features more frequently.

6.3 Experiment Results

The mean squared errors were typically lower on the last subtask when BP+PRR or BP+RGT was used instead of BP (Figures 6.1 and 6.2). The mean squared errors over subtasks formed a curve that tended to be thinner and lower when BP+RGT or BP+PRR was used instead of BP for a given ANN. However, with the SoftSign ANNs, BP+PRR resulted in higher mean squared errors compared to those from when BP was used instead.

For almost all ANNs, the SFEs on later substasks were higher with BP

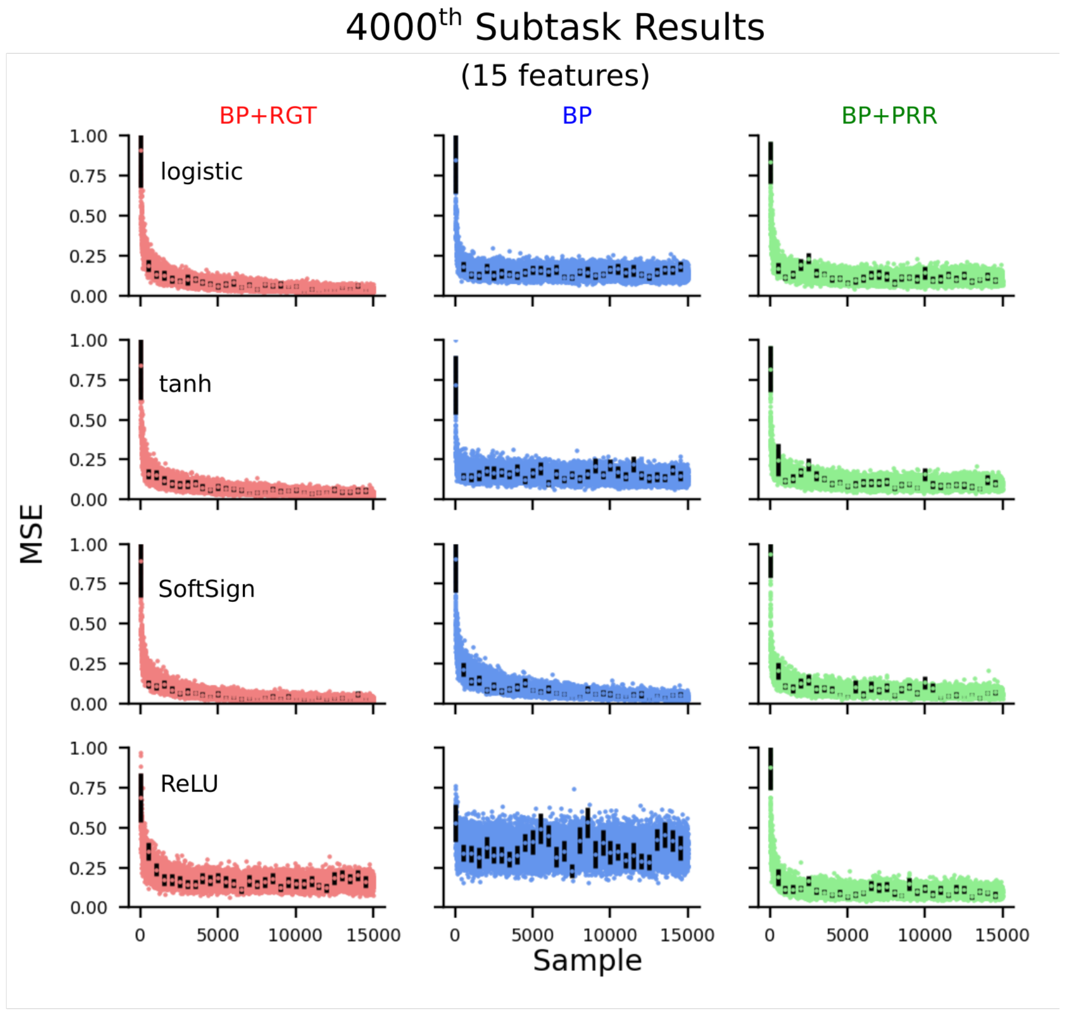

Figure 6.1: The mean squared errors on the last subtask for each size-15 ANN is shown here. The black bars represent one standard error determined by 50 independent runs, and they are plotted every 500 subtasks. ANNs trained with BP+RGT or BP+PRR tended to have lower error than BP in most cases.

than with BP combined with a generate-and-test algorithm (Figure 6.3). The exceptions to this trend involved the SoftSign ANNs. Across all subtasks, the SoftSign ANNs trained with BP+PRR had higher SFEs than the SFEs from when they were trained with BP.

For four ANNs, the SFEs on later subtasks with BP+RGT were higher than or similar to the SFEs on later subtasks with BP+PRR: the 40-feature logistic ANN, the 40-feature tanh ANN, and both ReLU ANNs (Figure 6.3). This result was surprising as BP+PRR has a simpler tester than BP+RGT's.

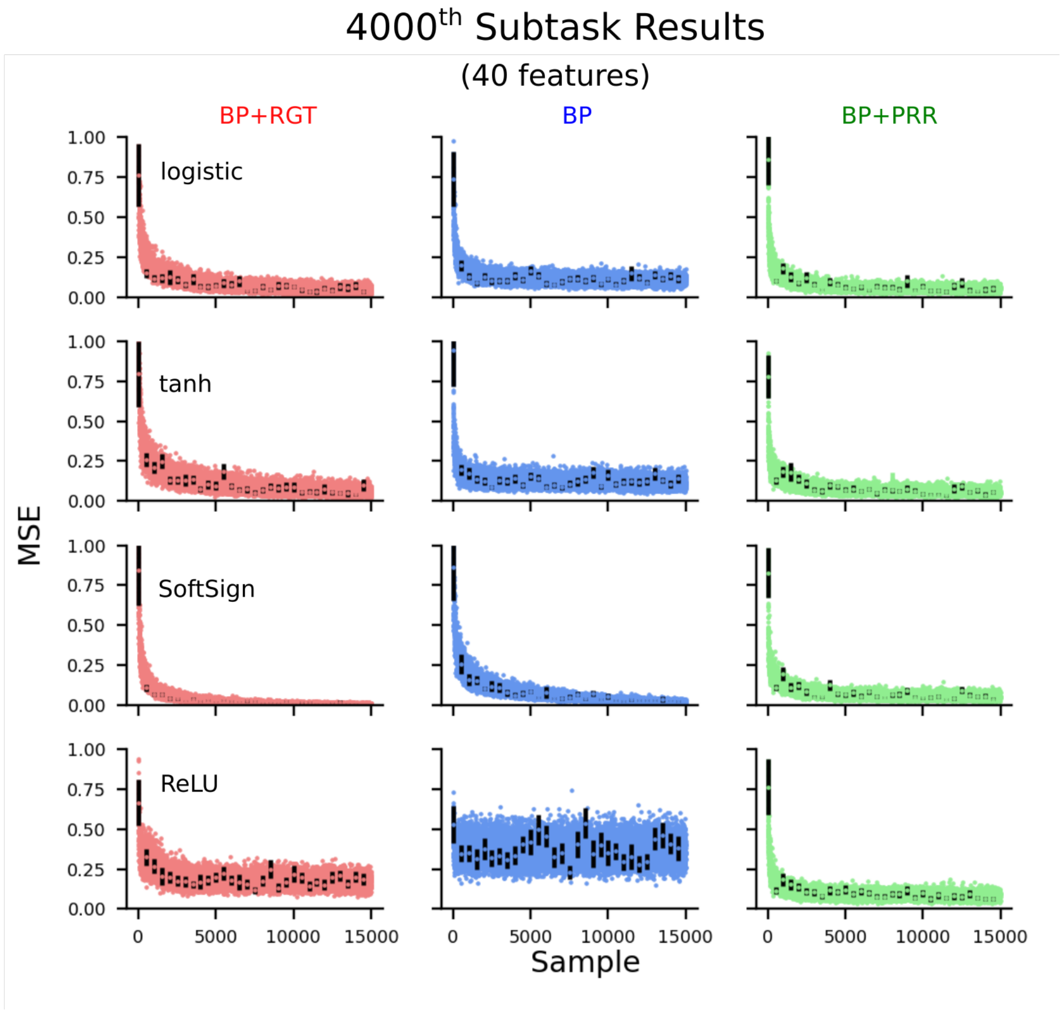

Figure 6.2: The mean squared errors on the last subtask for each size-40 ANN is shown here, and are formatted like Figure 6.1.

With the other ANNs, BP+RGT had lower SFEs than the SFEs when trained with BP+PRR.

On later subtasks, the ReLU ANNs had high SFEs compared to the SFEs from ANNs with other activation functions. The ReLU ANNs trained with BP+RGT had lower SFEs than when trained with BP, but the SFEs were around 0.15, which is close to the SFEs of the fixed-representation ReLU ANN tested in Chapter 5. In contrast, the other ANNs had SFEs around 0.05 or lower when trained with BP+RGT. With BP+PRR, though, the ReLU ANNs had SFEs comparable to the 15-feature ANNs that used BP+PRR (~ 0.1).

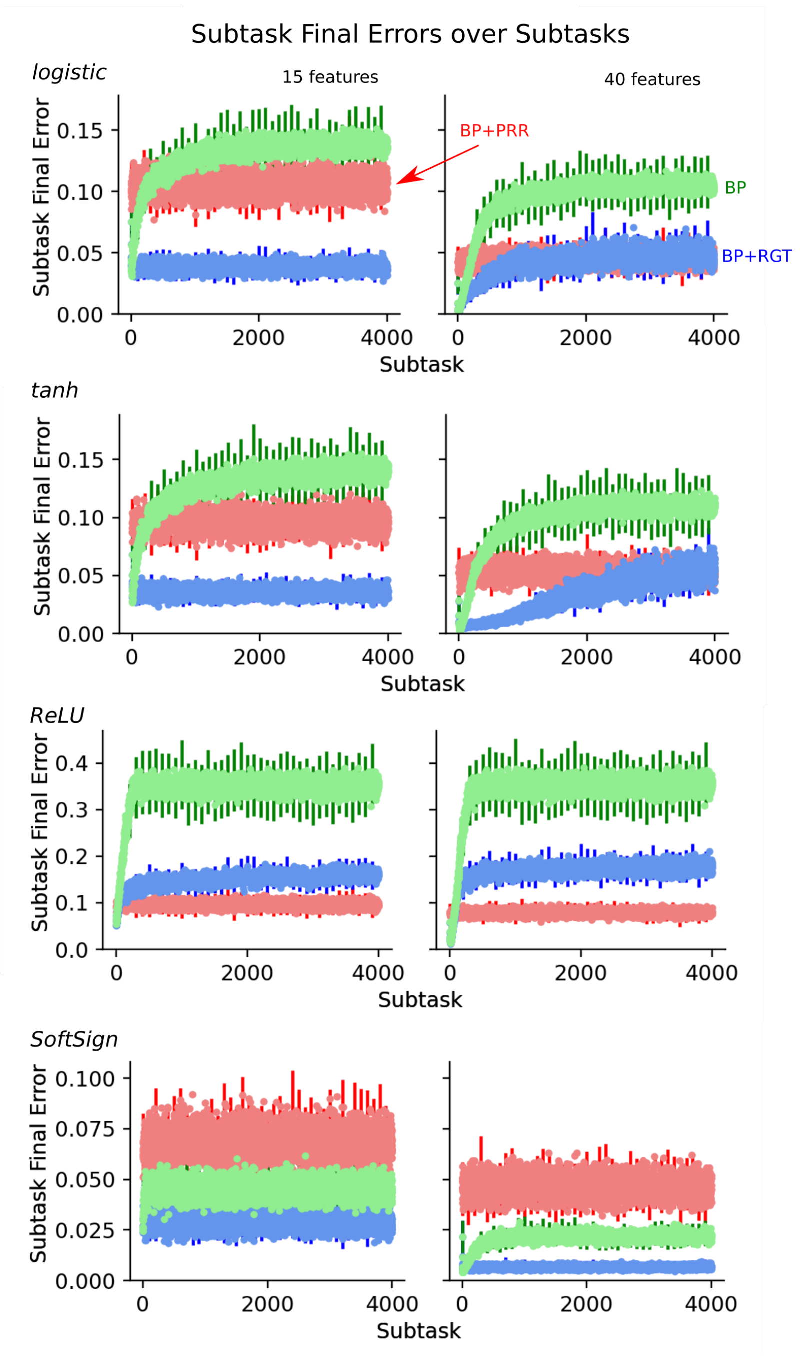

Figure 6.3: These plots illustrate the performance of the different learning algorithms on different ANNs. The dark bars represent one standard error determined by 50 independent runs, and they are plotted every 100 subtasks. An ANN trained with BP combined with a generate-and-test algorithm typically had lower SFEs than those of the ANN trained with BP. The only exception to this observation, is when the SoftSign ANNs were trained with BP+PRR and the SFEs exceeded those from when they trained with BP.

For a given learning algorithm and activation function, the SFEs on later subtasks were typically smaller or similar with more features (Figure 6.3). There were three exceptions to this trend with the tanh, logistic, and ReLU ANNs that used BP+RGT. On later subtasks, their 40-feature versions had higher SFEs than the SFEs of their 15-feature versions.

The ANNs trained with BP+PRR had relatively stable SFEs over subtasks compared to the SFEs of ANNs trained with BP or BP+RGT (Figure 6.3). With all activation functions, BP+PRR typically had SFEs less than or around 0.1. ANNs that used BP or BP+RGT had lower SFEs on early subtasks compared to their SFEs on later subtasks. The increase in SFEs occurred over early subtasks, and the SFEs on later subtasks were relatively stable.

For a given ANN, using either BP+PRR or BP+RGT yielded higher average activation function derivatives at the beginning of subtasks (Figure 6.4) than those from when BP was used. BP+PRR consistently had the highest average activation function derivatives at the beginning of subtasks.

The change in the SFEs over subtasks had a similar pattern to the change in the average activation function derivative at the beginning of subtasks (Figure 6.4). For most ANNs that used BP+RGT, the average activation function derivative at the beginning of subtasks began high but then decreased sharply. The decrease was steep over early subtasks and then flattened out over later subtasks. This change in average activation function derivatives was analogous to how the SFEs when using BP+RGT increased quickly over early subtasks and become more stable over later subtasks. For the 40-feature tanh ANN that used BP+RGT, in particular, the average activation function derivative decreases at a shallower rate, and this shallower pattern resembles the learner's shallower increase in SFEs over subtasks. Both the increase in the SFEs over subtasks and the decrease in the average activation function derivative at the beginning of subtasks were not as steep with BP+RGT as with BP. The stable

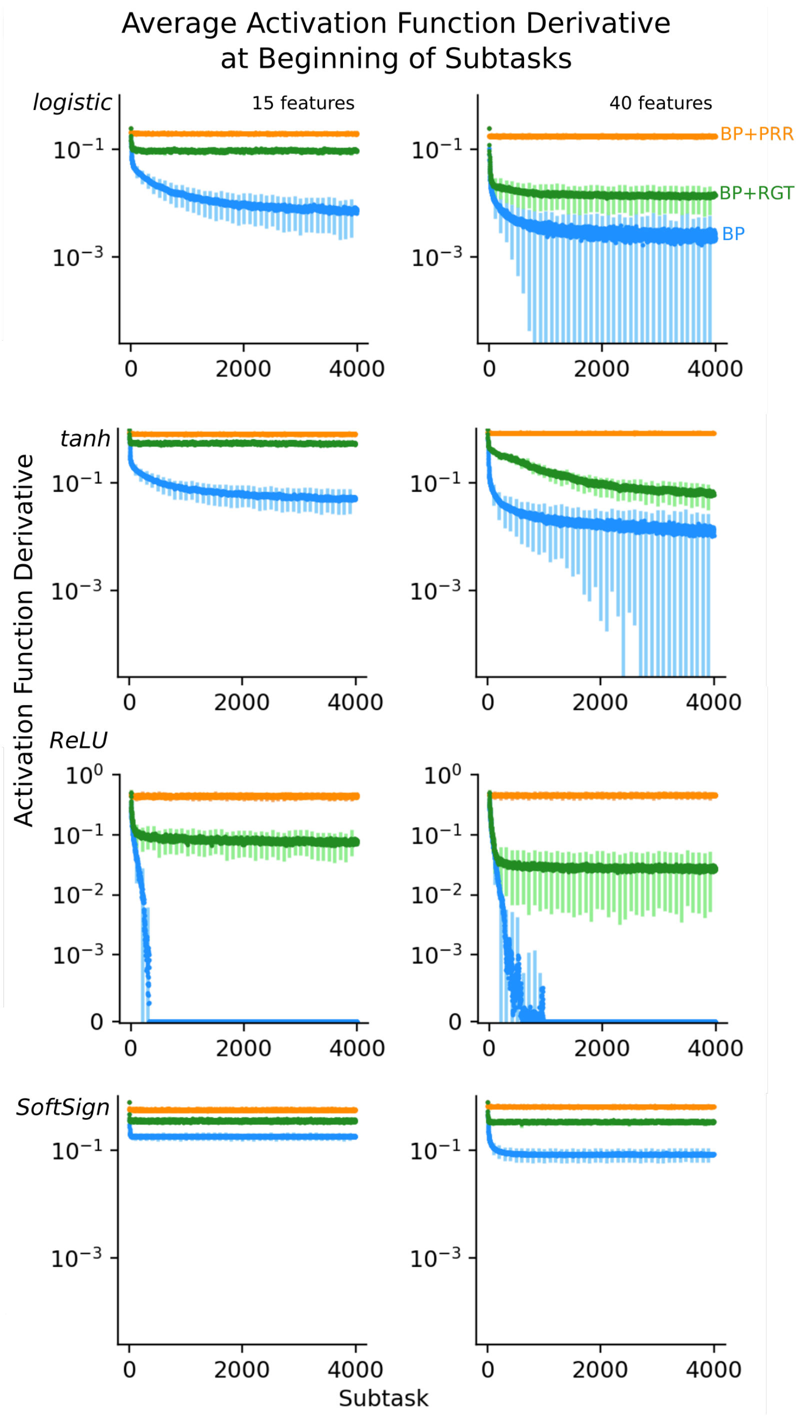

Figure 6.4: Each curve represents an ANN architecture and a learning algorithm. The third row has a semilog plot that is log-scaled above 10^{-4} and linear-scaled below. The light-colored bars represent one standard error determined by 50 independent runs and are plotted every 100 subtasks. ANNs trained with BP and a generate-and-test algorithm had higher average activation function derivatives than when they were trained with BP.

SFEs over subtasks with BP+PRR reflected the relatively uniform activation function derivatives at the beginning of subtasks with BP+PRR.

ANNs that used BP+PRR or BP+RGT typically had fewer calcified features at the beginning of subtasks than when they used BP (Figure 6.5). SoftSign ANNs had zero calcified features at the beginning of subtasks regardless of the learning algorithm. With the 15-feature tanh and logistic ANNs, BP+RGT had almost zero calcified features at the beginning of subtasks unlike when BP was used. With other ANNs, the proportion of calcified features at the beginning of subtasks with BP+RGT was relatively large compared to the near zero proportion of calcified features at the beginning of subtasks that resulted when BP+PRR was used.

The average activation function derivative at the beginning of subtasks and the proportion of calcified features at the beginning of subtasks when a learning algorithm was used, did not predict whether a learning algorithm had lower or higher SFEs than another learning algorithm. The ranking of algorithms from highest average activation function derivatives at the beginning of subtasks to lowest was: BP+PRR, BP+RGT, and BP (Figure 6.4). The ranking of algorithms from highest proportion of calcified features to lowest was: BP, BP+RGT, BP+PRR (Figure 6.5). However, when it comes to ranking learning algorithms based on their SFEs, the algorithms had different rankings depending on the size and the activation function of the ANN.

Finally, we note the ANNs that trained with random generate and test on its own (RGT) and partial random replace on its own (PRR) had SFEs near or higher than those of the fixed-representation ANNs (Figure 6.6).

6.4 Experiment Conclusions

More useful sets of features were discovered on later subtasks with the backpropagation algorithm combined with a generate-and-test algorithm than with

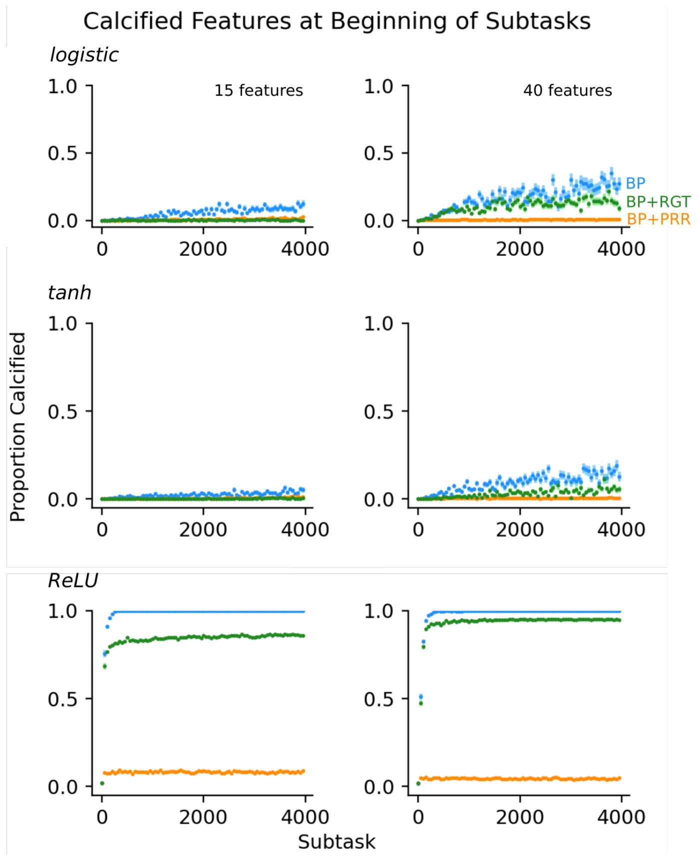

Figure 6.5: These plots show the average proportion of calcified features at the beginning of every 50th subtask. The light-colored bars represent one standard error determined by 50 independent runs. The SoftSign ANNs (not shown) had zero calcified features at the beginning of every subtask with each learning algorithm. Fewer calcified features were present at the beginning of subtasks when BP was combined with a generate-and-test algorithm than when BP was used.

the backpropagation algorithm alone. For all ANNs, the SFEs on later subtasks were lower with BP+RGT than with BP. Also, for most of the ANNs, the SFEs on later subtasks were lower with BP+PRR than with BP. However, when the SoftSign ANNs were trained with BP+PRR the SFEs across subtasks were higher compared to the SFEs when the SoftSign ANNs were trained with BP. Therefore, we still must be careful when choosing a generate-and-test algorithm to improve the feature discovery performance of the backpropagation algorithm, but they can be worth using for the quicker discovery of useful

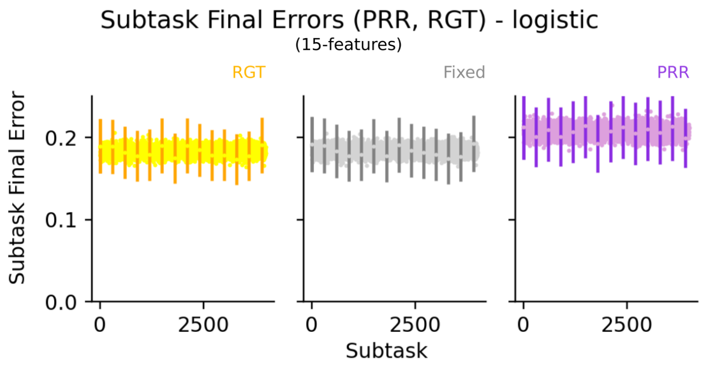

Figure 6.6: These plots illustrate the performance of an ANN updated with RGT *(orange)*, a fixed-representation ANN *(grey)*, and an ANN updated with PRR *(purple)* on the GCFD task. The results are for a size-15 logistic ANN. The dark-colored bars represent one standard error determined by 50 runs, and they are plotted every 100 subtasks. PRR tended to have higher SFEs than the other two, and RGT appeared to have similar SFEs to the fixed-representation ANN's. For other ANNs (not shown), RGT and PRR had similar or higher SFEs than those of the fixed representation.

features.

The interaction between the generate-and-test algorithms and the backpropagation algorithm was the reason for the fast discovery of useful features. On their own, the generate-and-test algorithms performed similarly to a fixed-representation ANN. This implies that when a generate-and-test algorithm was combined with the backpropagation algorithm, the additive effects of the individual algorithms were not as crucial to the faster discovery of useful features as the interaction between the two learning algorithms.

Using the generate-and-test algorithms alongside the backpropagation algorithm led to higher activation function derivatives and fewer calcified features for many of the ANNs. Both having fewer calcified features and having high, unstable activation functions are known to be beneficial to feature discovery performance. However, how an algorithm increases the activation function derivatives and decreases the proportion of calcified features may come with the slower discovery of useful features as it did for the SoftSign ANNs that used BP+PRR instead of BP. These properties can be beneficial to discovering useful features if introduced carefully.

Using a suitable tester can lead to better continual feature discovery performance. For many ANNs, updating with BP+RGT, which discriminates based on a feature's utility, led to lower SFEs than those from updating with BP+PRR, which does not consider a feature's utility. This indicates that BP+RGT can discover more useful features than BP+PRR for many ANNs. However, for the ReLU ANNs, BP+RGT led to higher SFEs than when BP+PRR was used. So, for ReLU ANNs, the BP+RGT may not be appropriate for good continual feature discovery performance. Also, the 40-feature logistic ANN and 40-feature tanh ANN had higher SFEs with BP+RGT than the SFEs from when BP+PRR was used with their 15-feature versions. This indicates that either the backpropagation algorithm parameters were poor for continual feature discovery with the 40-feature versions or the generate and test parameters of BP+RGT algorithm could be tweaked for better performance on the 40-feature ANNs. Accounting for the activation type and size of the network when choosing to use a generate-and-test algorithm can lead to better continual feature discovery performance.

Overall, the discovery of useful features for new sample distributions with the backpropagation algorithm was improved with generate-and-test algorithms. This shows that generate-and-test algorithms can have significant benefits for continual feature discovery performance.

More work with generate-and-test algorithms can potentially yield better continual feature discovery algorithms. The generate-and-test algorithms we tested updated infrequently, were computationally cheap, and were untuned. Yet, the performance gains to the backpropagation algorithm were significant. The partial random replace algorithm was simple, but improved the backpropagation algorithm's discovery of useful features with all ANNs except with the SoftSign ANNs. As mentioned above, finding suitable testers is an important part of the performance improvement from generate-and-test algorithms. BP+PRR had a consistency in performance across the sequence of subtasks that may be desirable for future feature discovery algorithms.

This brings us to the end of the major results of this thesis. In this chapter, we demonstrated how simple and computationally cheap generate-and-test algorithms can improve the backpropagation algorithm's continual feature discovery performance. By showing how effective generate-and-test algorithms can be for continual feature discovery, we hope to have contributed to the research into agents that continually learn.

Chapter 7

Conclusion

Continual feature discovery is an important aspect of continual learning. An agent that continually learns may need to learn from complex sample distributions throughout the it's operation. To empower the agent to learn from these distributions, we can ensure it can discover useful features for all sample distributions it encounters.

In this thesis, we motivated the use of the generate-and-test algorithms for better performance in a continual feature discovery task. First, we introduced the generic continual feature discovery task that tests a learner's ability to discover new features for new sample distributions. Then, we demonstrated that the backpropagation algorithm, an algorithm renowned for its feature discovery on tasks where the sample distribution does not change, can perform poorly on tasks with changing sample distributions. We explained that the poor continual feature discovery performance can be attributed to the backpropagation algorithm's sensitivity to initial conditions that are not maintained throughout the backpropagation algorithm's updates. Then, with two simple generate-and-test algorithms, we showed the backpropagation algorithm can benefit greatly from the synergy between it and generate-and-test algorithms.

The investigated generate-and-test algorithms were not tuned and were relatively unsophisticated yet they contributed a large performance improvement to the backpropagation algorithm's continual feature discovery. The algorithms generated random features, because features with small, random weights are known to be useful for fast learning with the backpropagation algorithm. Potentially, a more useful generator can be proposed to handle various data distributions. The tester of the generate-and-test algorithm could also become more sophisticated and take into account other statistics to improve performance.

More work can be done with the artificial neural network's (ANN's) structure. From our work, it appears the issues with the backpropagation algorithm stemmed from the choice in activation function and how the activation function derivative changed over samples. A better understanding of the activation functions could yield a better understanding of the backpropagation algorithm's limitations in continually discovering useful features. SoftSign, a relatively uncommon choice for an activation function, showed itself as a useful activation function for continual feature discovery with the backpropagation algorithm. The feature discovery with SoftSign activation functions did not slow as much as when the ANN had other activation functions, although, there was still some slowdown.

Further research with other representations can establish better algorithms and principles for the continual discovery of useful features. Continual feature discovery algorithms should eventually account for more sophisticated representations and data streams. In the generic continual feature discovery task, the input distribution never changed, and this assumption does not hold for agents that continually learn in general. Continual feature discovery with other types of features such as recurrent features or convolutional features would help establish the generality of the principles of continual feature discovery algorithms. It would be interesting to investigate whether these other feature types that can be updated with the backpropagation algorithm can also benefit from generate-and-test algorithms for continual feature discovery tasks.

References

Aljundi, R., Belilovsky, E., Tuytelaars, T., Charlin, L., Caccia, M., Lin, M., Page-Caccia, L. (2019). Online Continual Learning with Maximal Interfered Retrieval. In Wallach, H., Larochelle, H., Beygelzimer, A., Alché-Buc, F., Fox, E., Garnett, R., (Eds.), *Advances in Neural Information Processing Systems 32*, pp. 11849–11860. Curran Associates, Inc.

Aljundi, R., Kelchtermans, K., Tuytelaars, T. (2019). Task-Free Continual Learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11254–11263, Long Beach, CA, USA. IEEE, Inc.

Blum, A. L., Langley, P. (1997). Selection of Relevant Features and Examples in Machine Learning. *Artificial Intelligence 97*(1), pp. 245–271. Elsevier.

Blumenfeld, Y., Gilboa, D., Soudry, D. (2020). Beyond Signal Propagation: Is Feature Diversity Necessary in Deep Neural Network Initialization? In *Proceedings of the Thirty-Seventh International Conference on Machine Learning*, pp. 6910–6919. JMLR.

Booker, L. B., Goldberg, D. E., Holland, J. H. (1989). Classifier Systems and Genetic Algorithms. *Artificial Intelligence 40*(1), pp. 235—282. Elsevier.

Choy, M. C., Srinivasan, D., Cheu, R. L. (2002). Hybrid Cooperative Agents with Online Reinforcement Learning for Traffic Control. In *2002 IEEE International Conference on Fuzzy Systems*, pp. 1015–1020. IEEE Inc.

Cireşan, D. C., Meier U., Masci, J., Gambardella, L. M., Schmidhuber, J. (2011). Flexible, High Performance Convolutional Neural Networks for Image Classification. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI-11)*, pp. 1237–1242. AAAI Press.

Cogswell, M., Ahmed, F., Girshick, R. B., Zitnick, C. L., Batra, D. (2016). Reducing Overfitting in Deep Networks by Decorrelating Representations. In *International Conference on Learning Representations (ICLR 2016).*

Cui X., Hardin, C. T., Ragade, R. K., Potok, T. E., Elmaghraby, A. S. (2005). Tracking Non-Stationary Optimal Solution by Particle Swarm Optimizer. *Proceedings of the 6th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Com-*

puting (SNPD 2005), pp. 133–138. IEEE Inc.

Denoeux, T., Lengellé R. (1993). Initializing Back Propagation Networks with Prototypes. *Neural Networks 6*(3), pp. 351–363. Elsevier.

Diaz-Rodriguez, N., Lomonaco, V., Filliat, D., Maltoni, D. (2018). Don't Forget, There is more than Forgetting: New Metrics For Continual Learning. In *Continual Learning Workshop at NeurIPS*.

Douglas, S. C., Yu, J. (2018). Why RELU Units Sometimes Die: Analysis of Single-Unit Error Backpropagation in Neural Networks. In *52nd Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, USA, pp. 864–868.

Draelos, T. J., Miner, N. E., Lamb, C. C., Cox, J. A., Vineyard, C. M., Carlson, K. D., Severa, W. M., James, C. D., Aimone, J. B. (2017). Neurogenesis Deep Learning: Extending Deep Networks to Accommodate New Classes. In *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE Inc.

Duch, W., Adamczak, R., Jankowski, N. (1997). Initialization and Optimization of Multilayered Perceptrons. In *Third Conference on Neural Networks and Their Applications*.

Elliott, D. L. (1993). *A Better Activation Function for Artifical Neural Networks*. ISR Technical Report TR 93-8. University of Maryland.

Elwell, R., Polikar, R. (2011). Incremental Learning of Concept Drift in Nonstationary Environments. *IEEE Transactions on Neural Networks 22*, pp. 1517–1531. IEEE, Inc.

Evci, U., Gale, T., Menick, J., Castro, P. S., Elsen, E. (2019). *Rigging the Lottery: Making All Tickets Winners*. Preprint. arxiv.org/abs/1911.11134

Fahlman, S. E. (1988). *An Empirical Study of Learning Speed in Back-Propagation Networks*. Technical Report CMU-CS-88-162. Carnegie-Mellon University.

Fahlman, S. E., Lebiere, C. (1989). The Cascade-Correlation Learning Architecture. In *Advances in Neural Information Processing Systems II*. Morgan Kaufmann, San Mateo.

Farquhar, S., Gal, Y. (2018). Towards Robust Evaluations of Continual Learning. Preprint. arxiv.org/abs/1805.09733

Frankle, J., Carbin, M. (2019). The Lottery Ticket Hypothesis: Finding

Sparse, Trainable Neural Networks. In *International Conference on Learning Representations (ICLR 2019).*

French, R. (1999). Catastrophic Forgetting in Connectionist Networks. In *Trends in Cognitive Sciences 3*(4), pp. 128–135. Cell Press.

Fritzke, B. (1997). A Self-Organizing Network That Can Follow Non-Stationary Distributions. *ICANN '97: Proceedings of the 7th International Conference on Artificial Neural Networks*, pp. 613–618. Springer Berlin, Heidelberg.

Fu, L., Hsu, H., Principe, J. (1996). Incremental Backpropagation Learning Networks. *IEEE Transactions on Neural Networks 7*, pp. 757–761.

Glorot, X., Yoshua, B. (2010). Understanding the Difficulty of Training Deep Feedforward Neural Networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 249–256. PMLR.

Golkar, S., Kagan, M., and Cho, K. (2019). Continual Learning via Neural Pruning. Preprint. arxiv.org/abs/1903.04476

Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A., Bengio, Y. (2013). An Empirical Investigation of Catastrophic Forgetting in Gradient-Based Neural Networks. In *Proceedings of International Conference on Learning Representations (ICLR 2014).*

Goodfellow, I., Bengio, Y., Courville, A. (2016). *Deep Learning.* MIT Press.

Guyon, I., Elisseeff, A. (2003). An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research 3*, pp. 1157–1182.

Hayes, T. L., Kemker, R., Cahill, N. D., Kanan, C. (2018). New Metrics and Experimental Paradigms for Continual Learning. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 2112–2113. IEEE Inc.

He, K., Zhang X., Ren, S., Sun, J. (2015). Delving Deep Into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034. IEEE Inc.

Hoi, S. C., Sahoo, D., Lu, J., Zhao, P. (2018). Online Learning: A Comprehensive Survey. Preprint. arxiv.org/abs/1802.02871

Ioffe, S., Szegedy, C. (2014). Batch Normalization: Accelerating Deep Network

Training by Reducing Internal Covariate Shift. Preprint. arxiv.org/abs/1502.03167v3

Jacobsen, A., Schlegel, M., Linke, C., Degris, T., White, A., White, M. (2019). Meta-Descent for Online, Continual Prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence 33*, pp. 3943–3950. AAAI Press.

Kaelbling, L. P. (1990). *Learning in Embedded Systems*. Dissertation. Stanford University.

Kemker R., Abitino A., McClure M., Kanan, C. (2017). Measuring Catastrophic Forgetting in Neural Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence 32*, pp. 3390–3398. AAAI Press.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., Hadsell, R. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences 114*(13), pp. 3521–3526. National Academy of Sciences.

Kingma, D. P., Ba, J. (2015). Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations*, San Diego, CA, USA.

LeCun, Y., Bottou L., Orr, G. B., Klauss-Robert, M. (1998). Efficient Back-Prop. In *Neural networks: Tricks of the trade*, pp. 9–48. Springer Berlin, Heidelberg.

Mahmood, A. R., Sutton, R. S. (2013). Representation Search through Generate and Test. In *AAAI Workshop: Learning Rich Representations from Low-Level Sensors.*

Mahmood, A. R. (2017). *Incremental Off-policy Reinforcement Learning Algorithms*. PhD thesis. University of Alberta.

Martínez-Rego, D., Pérez-Sánchez, B., Fontenla-Romero, O., Alonso-Betanzos, A. (2011). A Robust Incremental Learning Method for Non-Stationary Environments. *Neurocomputing 74*(11), pp. 1800–1808. Elsevier.

Mauldin, M. L. (1984). Maintaining Diversity in Genetic Search. *Proceedings of the Fourth National Conference on Artificial Intelligence*, pp. 247–250. AAAI Press.

McCloskey, M., Cohen, N. J. (1989). Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. In G. H. Bower (Eds.), *Psychology of Learning and Motivation 24*, pp. 109–165. Academic Press.

Mermillod, M., Bonin, P., Méot, A., Ferrand, L., Paindavoine, M. (2012). Computational Evidence That Frequency Trajectory Theory Does Not Oppose But Emerges From Age-of-Acquisition Theory. In *Cognitive Science 36*, pp. 1499–1531.

Moriarty, D. E. (1997). *Symbiotic Evolution of Neural Networks in Sequential Decision Tasks.* Dissertation. University of Texas at Austin.

Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., Liang, E. (2006). Autonomous Inverted Helicopter Flight via Reinforcement Learning. In *Experimental Robotics IX*, pp. 363–372. Springer Berlin, Heidelberg.

Ng, S. C., Cheung, C. C., Leung, S. H., Luk A. (2003). Fast Convergence for Backpropagation Network with Magnified Gradient Function. In *Proceedings of the International Joint Conference on Neural Networks, 2003*, pp. 1903–1908.

Nguyen, D., Widrow, B. (1990). Improving the Learning Speed of 2-layer Neural Networks by Choosing Initial Values of the Adaptive Weights. In *1990 International Joint Conference on Neural Networks (IJCNN).* IEEE, Inc.

Nwankpa, C., Ijomah, W., Gachagan, A., Marshall, S. (2018). Activation Functions: Comparison of Trends in Practice and Research for Deep Learning. Preprint. arxiv.org/abs/1811.03378

Pérez-Sánchez, B., Fontenla-Romero, O., Guijarro-Berdiñas, B. (2018). A Review of Adaptive Online Learning for Artificial Neural Networks. *Artificial Intelligence Review 49*, pp. 281–299.

Rajasegaran, J., Hayat, M., Khan, S. H., Khan, F. S., Shao, L. (2019). Random Path Selection for Continual Learning. In *Proceedings of Advances in Neural Information Processing Systems 32*, pp. 12669–12679. Curran Associates, Inc.

Rolnick, D., Ahuja, A., Schwarz, J., Lillicrap, T. P., Wayne, G. (2019). Experience Replay for Continual Learning. In *Proceedings of Advances in Neural Information Processing Systems 32*, pp. 350–360. Curran Associates, Inc.

Rumelhart, D. E., Hinton G. E., Williams, R. J. (1986). Learning Representations by Back-propagating Errors. In *Nature 323*, pp. 533–536.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., Hadsell, R. (2016). Progressive Neural Networks. Preprint. arxiv.org/abs/1606.04671

Sahoo, D., Pham, Q., Lu, J., Hoi, S. C. H. (2018). Online Deep Learning: Learning Deep Neural Networks on the Fly. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)*, pp. 2660–2666. AAAI Press.

Schraudolph, N. N. (1999). Local Gain Adaptation in Stochastic Gradient Descent. In *9th International Conference on Artificial Neural Networks: ICANN '99*, pp. 569–574.

Shin, H., and Lee, J. K., Kim, J., Kim, J. (2017). Continual Learning with Deep Generative Replay. In *Advances in Neural Information Processing Systems 30*, pp. 2990–2999. Curran Associates, Inc.

Srivastava, R. K., Masci, J., Kazerounian, S., Gomez, F., Schmidhuber, J. (2013). Compete to Compute. *Advances in Neural Information Processing Systems 26*, pp. 2310–2318. Curran Associates, Inc.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research 15*(56), pp. 1929–1958. JMLR.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe1, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D. (2016). Mastering the Game of Go with Deep Neural Networks and Tree Search. In *Nature 529*, pp. 484–489.

Sutton, R. S. (1992). Adapting Bias by Gradient Descent: An Incremental Version of Delta-Bar-Delta. In *Proceeding of Tenth National Conference on Artificial Intelligence AAAI-92*, pp. 171–176. AAAI Press.

Sutton, R. S., Whitehead, S. D. (1993). Online Learning with Random Representations. In *Proceedings of the Tenth International Conference on Machine Learning*, pp. 314–321. Morgan Kaufmann.

Sutton, R. S. (2014). *Myths of Representation Learning.* Conference presentation. ICLR 2014, Banff, Canada. iclr.cc/archive/2014/

Thimm, G., Fiesler, E. (1997). High-Order and Multilayer Perceptron Initialization. *IEEE Transactions on Neural Networks, 8*(2), pp. 349–359. IEEE, Inc.

Veeriah, V., Zhang, S., Sutton, R. S. (2017). Crossprop: Learning Representations by Stochastic Meta-Gradient Descent in Neural Networks. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD).*

Yoon, J., Yang, E., Lee, J., Hwang, S. J. (2018). Lifelong Learning with Dynamically Expandable Networks. In *8th International Conference on Learning Representations.*