Slow Learning with Backprop on Nonstationary Tasks

Abstract

Backprop has been greatly successful at training artificial neural networks (ANNs) for a single data distribution. However, we show that backprop's ability to adapt ANNs for an abruptly changing data distribution can be quick for early data distributions but then become poor to the point where using a fixed random representation ANN would have led to much lower estimation errors or only slightly higher estimation errors. We attribute backprop's extremely poor performance to backprop's sensitivity to its initialization phase, which initializes features with small, random weights. To modify backprop for tasks where ANNs must be continually updated, we use *continual backprop*: a version of backprop that regularly replaces some of the ANN's features with new features that have small, random weights. This simple and cheap modification led to large performance gains and showed that injecting small, random weights throughout the task can be beneficial for backprop's ability to continually adapt.

1 Introduction

Recently, there have been many papers that have focused on modifying backprop for continual supervised learning tasks. A continual supervised learning task is a supervised learning task where the learner can expect to experience arbitrarily many data distributions but does not have access to past data. To be successful on continual supervised learning tasks, the learner should be able to retain useful knowledge from the past and quickly adapt to new situations. These tasks are considered *nonstationary*, since the data distribution changes. McCloskey and McCohen (1989) pointed out that backprop can experience *catastrophic interference*, a sudden and large performance decrease on past distributions, and many papers, including an influx of recent papers, have been dedicated to avoid backprop's catastrophic interference by modifying backprop (Fahlman and Lebiere, 1989; French, 1999; Aljundi et al., 2019; Kirkpatrick et al., 2017; Yoon et al., 2018; Golkar et al., 2019; Rajasegaran et al., 2019; Rolnick et al., 2019).

However, a more overlooked observation about backprop is that its ability to adapt artificial neural networks (ANNs) to new distributions can decrease when it is learning from a sequence of data distributions. Sutton (2014) presented the unpublished results of Mahmood that used a nonstationary MNIST task where the data distribution changes periodically and abruptly. They found the classification error at the end of each data distribution increased over distributions. A similar demonstration was done by Martínez-Rego et al. (2011) who showed that 1-layer ANNs that update with a form of backprop can have a slower learning speed on later sample distributions compared to the learning speed on earlier sample distributions. More generally, Fahlman (1988) noted that backprop can slow in learning from new samples as the derivative of the activation functions go to zero.

Backprop's slower learning on new sample distributions has yet to be established as problematic, so we contribute a clear demonstration of the surprisingly poor learning performance of backprop on new data distributions. To ensure transparency in the demonstration, we used a generic nonstationary supervised learning task called the *generic continual feature discovery* (GCFD) task where the data distribution changes abruptly and periodically. The task is online, meaning the learner receives one sample at a time. Independent of the choice of tanh, logistic, ReLU, or SoftSign activation function, backprop learned ANNs that yielded low estimation error on early data distributions. However, depending on the choice of activation function, the ANNs learned by backprop for later data distributions yielded higher or only slightly lower estimation errors than the estimation errors of an ANN with a fixed random representation. This high estimation error after learning with backprop showed backprop's poor ability to continually learn useful ANN weights for new data distributions.

We attribute the extremely poor learning ability of backprop on new data distributions to backprop's sensitivity to its initialization phase. Backprop consists of two phases: initialization of the ANN's representation with small, random numbers, and using stochastic gradient descent with data. It has been established that initializing ANNs properly is crucial to fast learning with backprop (Thimm and Fiesler, 1997; Glorot and Bengio, 2010; He et al., 2015). However, Schwarz et al. (2018) noted that backprop should reinitialize the ANN's weights when it encounters a different data distribution, otherwise backprop can be slow to learn from that distribution. This means, after learning on data distributions, backprop can lose properties from initialization that would have improved its learning ability. We find a similar result with out experiments.

To preserve the benefits of the initialization phase when learning from new sample distributions, other works have proposed modifications for backprop that introduce small, random weights during learning. Like Schwarz et al. (2018), many works assume the learner is informed of when the data distribution changes, and they choose to add multiple new features to learn new data distributions (Yoon et al., 2018, Rusu et al., 2016). Special knowledge of a distribution shift was not required from backprop initially. Cascade correlation networks increase the size of the ANN one feature at a time and use stochastic gradient descent to only update the newest feature (Fahlman and Lebiere, 1989; Fu et al., 1996). In a continual learning task, using an unbounded and increasing amount of memory can be problematic since the task is arbitrarily long. Also, backprop did not require an unbounded amount of memory to begin with.

Zhou et al. (2012) presents an algorithm to learn the optimal size of the ANN by adding and removing features, but requires an unspecified amount of memory. Evci et al. (2019) showed that generating random weights while removing low magnitude weights can eventually lead to a performant sparse network. However, they were motivated to reduce the memory usage and computation of a fully connected network rather than improve the learning speed of backprop on new sample distributions.

To avoid the extremely poor learning ability of backprop on the GCFD task while maintaining the cheap computation and fixed memory usage of backprop, we propose the use of *continual backprop*, which regularly injects small, random weights into the ANN alongside backprop. We call it "continual backprop" because it augments backprop to be suitable for learning continually in nonstationary supervised learning tasks without modifying backprop too much. By regularly replacing some features in the ANN with new features that have small, random weights throughout the learning process, we can potentially improve backprop's performance on new data distributions since these introduced features are known to be useful for learning new distributions. To replace features in a disciplined manner, we use *generate-and-test* algorithms. At a high level, generate-and-test algorithms replace the least useful parts of a learnable function with potentially more useful parts, and have been used on various applications including for learning Boolean functions (Kaelbling, 1990), rule-based classifiers (Booker et al., 1989), and ANN representations (Mahmood and Sutton, 2013). For ANN representations, generate-and-test algorithms replace the least useful features. Mahmood and Sutton (2013) demonstrated that using generate-and-test algorithms alongside backprop can lead to faster learning than with backprop alone on a stationary supervised learning task. However, they did not establish results on nonstationary supervised learning tasks. Dohare (2020) presented results showing that continual backprop is quicker to adapt ANNs to a gradually changing input distribution compared to backprop.

2 Background

An online supervised learning task provides a stream of examples to the learner. Each example contains an input $\mathbf{i} \in \mathbb{R}^n$, where $n \in \mathbb{N}$ is the size of the input, and a target t. Since we are focused on regression tasks, t is a scalar. The target is calculated by a *target function* that is given the input $\mathbf{i}$. The target function can be a human noting the targets or a defined function, and it can change during the task.

For each example, the learner is first provided the example's input. The learner then returns an estimate of the example's target. The error between the learner's estimate and the example's target is recorded for the results, and then the learner is provided the example's target to adapt its estimates for future examples.

The learners we focus on are two-layer artificial neural networks (ANNs). These ANNs have a *representation* that consists of $m \in \mathbb{N}$ features. A feature is a function that summarizes the input signals into a scalar. The representation takes in an example's input $\mathbf{i} \in \mathbb{R}^n$ and outputs the representation's summary $\mathbf{f} \in \mathbb{R}^m$. The k^{th} feature's output, $f_k \in \mathbb{R}$, is the k^{th} entry of $\mathbf{f}$ and is calculated with the expression $f_k = \sigma\left(\mathbf{i} \cdot \mathbf{w}_k\right)$ where σ is a scalar to scalar function known as the *activation function*, and $\mathbf{w}_k \in \mathbb{R}^n$ is the k^{th} feature's input weights.

The final estimate e of the ANN given an example's input $\mathbf{i}$ is calculated using the output layer's weight vector $\mathbf{w^o} \in \mathbb{R}^m$, representation's summary $\mathbf{f}$, and the expression $e = \mathbf{w^o} \cdot \mathbf{f}$.

To use backprop to learn the ANN weights, the features are initialized with small, random weights. Then, to update with each example's input $\mathbf{i}$ and target t, backprop uses the stochastic gradient descent updates:

$$\mathbf{w}_k \leftarrow \mathbf{w}_k + \alpha_r \frac{\partial (t-e)^2}{\partial \mathbf{w}_k} \text{for the } k^{th} \text{ feature, and}$$
$$\mathbf{w^o} \leftarrow \mathbf{w^o} + 2\alpha_o \frac{\partial (t-e)^2}{\partial \mathbf{w^o}}$$

where $\alpha_r \in \mathbb{R}^+$ is the representation's step size and the $\alpha_o \in \mathbb{R}^+$ is the output layer's step size.

Backprop has been found to be sensitive to its initialization phase. Initializing features with random weights ensures the independently updated features do not remain uniform, but also provides a level of feature diversity that can speed up learning with backprop. Feature diversity includes number of features and how different the features are from one another. Increasing these two aspects of feature diversity have been found to increase the learning speed of backprop (Nguyen and Widrow, 1990; Denoeux and Lengellé, 1993). Initializing features with small weights has also been found to improve the learning speed of backprop for ANNs with logistic activation functions (Thimm and Fiesler, 1997) and is key for specialized initialization schemes that speed up learning (Glorot and Bengio, 2010; He et al., 2015).

3 The Generic Continual Feature Discovery Task

To test algorithms for their ability to continually adapt ANNs for new data distributions, we use the *Generic Continual Feature Discovery* (GCFD) task. The GCFD task is a nonstationary online supervised learning task that changes the data distribution every $\lambda \in \mathbb{N}$ examples. Each example is composed of an input $\mathbf{i}$ that is a randomly sampled binary vector and a target $t = f(\mathbf{i})$ where f is the target function.

The target function f is a randomly initialized 2-layer ANN with all weights equiprobably -1 or 1. The features in the target function are called *target features*. The activation function of the target features output 1 or 0, and are known as linear threshold functions. When the input into the activation function is greater than or equal to a set threshold, the activation function outputs 1. Otherwise, the activation function outputs 0. The threshold β_k of the k^{th} feature's activation function is determined by the expression $\beta_k =$

(# of -1's in $\mathbf{w}_k$) $+ \nu m$. Each feature that uses the above 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 at least ν proportion of the -1 and 1 pattern of the LTU's input weight vector corresponds to the 0 and 1 pattern of the input vector.

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 learning task can become too simple. In our experiments we use size-7 inputs and 2 target features in the target function. With a size-7 input, the probability a target feature outputs one with a random binary input vector is $\sim$0.226 (see Appendix for calculation). With two target features, this makes the probability at least one LTU outputs one for a random binary input vector approximately $1 - (1 - 0.227)^2 = 0.40$.

We divide the GCFD task into *subtasks*. A subtask of the GCFD task is an interval of $\lambda \in \mathbb{N}$ examples where the data distribution is unchanged. The learner is never explicitly notified when the subtask switches 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 data distribution.

The task challenges the learner to quickly learn from each new data distribution. With two target features and size-7 inputs, a given target feature is expected to repeat every 64 subtasks (see Appendix for calculation), which makes the target functions diverse. 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 example in the subtask interval, the learner can ideally learn to account for each of the examples 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 the end of each subtask.

4 Backprop on the GCFD Task

In this section we present the results of our experiment that uses the generic continual feature discovery (GCFD) task to evaluate backprop's ability to adapt ANNs for each data distribution in a sequence. We used various activation functions with the ANNs to test how the choice of activation function affects the learning performance of backprop.

4.1 Task

In this GCFD task, the target function has two target features, and the inputs are of size 7. Each subtask is 15,000 examples long, which provides the learner $15,000/128 \approx$117 of each of the 128 input configurations in expectation per subtask. The learner must learn to estimate targets for a total of 4,000 subtasks. This long length gives a sample of a learner's performance on a task with arbitrarily many data distribution changes.

4.2 Learners

The learners were 2-layer artificial neural networks (ANNs) updated with backprop. Backprop initialized the features with a zero-centered normal distribution with variance 0.01. We found the performance of backprop to be insensitive to various small variances. Each ANN had either 15 or 40 features. Each ANN had logistic, tanh, ReLU, or SoftSign activation functions. Every backprop trained ANN was compared to a version of it that had a fixed representation i.e. $\alpha_r = 0$.

To select the representation step size α_r and the output layer step size α_o for backprop, we evaluated the step size choices on a shorter GCFD task. The shorter GCFD task has 30 subtasks (450,000 examples) 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 examples 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 for a learner that needs to continually learn, so the goal was not to optimize the step size for the specific $4{,}000^{th}$ subtask. The step-size parameter search was over the following possibilities: $\alpha_o \in \{0.00625/N, 0.0125/N, 0.025/N, 0.05/N, 0.1/N, 0.2/N, 0.4/N, 0.8/N, 1.6/N, 3.2/N\}$, where N is the number of features, and $\alpha_r \in \{0.0, 0.0015625, 0.003125, 0.00625, 0.0125, 0.025, 0.05, 0.1, 0.2, 0.4, 0.8, 1.6, 3.2\}$.

4.3 Evaluation Metrics

To measure backprop's effectiveness in learning from new data distributions, we looked at the error at the end of each subtask and the mean squared errors during subtasks 100, 1000, 4000. The *subtask final error* (SFE) is the average error of the last 1,000 examples of the subtask. The SFE gives a measure of how effectively the learners learned weights for a subtask.

We wished to monitor how two properties set by the initialization phase of backprop changed during the task: the activation function derivatives, and the number of features that are useful for learning with.

We monitored how the activation function derivatives changed by measuring the product of the average activation function derivative (which changes) and the representation's step size (which is constant) at the beginning of each subtask. It is difficult to directly compare the activation function derivatives of different types of activation functions, since the representation step size scales up the derivative. Together, both scale up how much the weights are changed by stochastic gradient descent updates.

We monitored how many features were no longer useful for learning by measuring the proportion of *calcified* features in the ANN. A feature is considered calcified when it is slow to update. Specifically, we define a feature as calcified when the product of the activation function derivative and representation step size is less than 0.0001 when averaged across the input space.

4.4 Results

We refer to the ANNs that update with backprop 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 backprop fell between the extremes of the search ranges (Appendix). For each squashing activation function: SoftSign, logistic, and tanh, the 40-feature ANN had a larger representation step size than the representation step size of the 15-feature ANN.

For each ANN, the rate the mean squared errors decreased differed across subtasks. With the exception of the SoftSign ANNs, there were noticeably higher errors throughout the 1000^{th} and 4000^{th} subtasks compared to those on the earlier subtasks (Figure 1). For all ANNs, the mean squared error decrease on the $1{,}000^{th}$ subtask was similar to the mean squared error decrease on $4{,}000^{th}$ subtask.

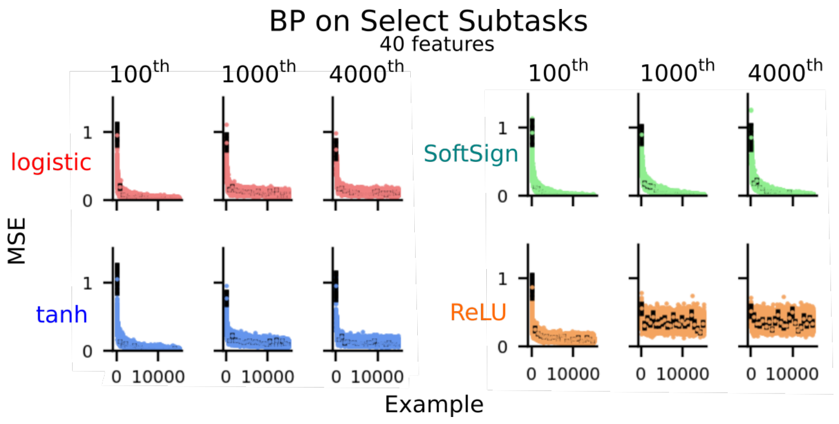

Figure 1: These plots show the mean squared errors from subtasks 100, 1,000, and 4,000 for size-40 ANNs. Each black bar represents one standard error determined by 50 independent runs, and they are plotted every 500 subtasks. There were higher MSEs on later subtasks. A similar trend occurred with the size-15 ANN results.

The fixed representation ANNs had SFEs around 0.15 or greater for each subtask (Figure 2).

The SFEs of the ReLU ANNs greatly increased over subtasks (Figure 2). On early subtasks, the ReLU ANNs had much lower SFEs than those from the fixed representation ANNs. However, after a few hundred subtasks, the ReLU ANNs had much higher SFEs than those from the fixed representation ANNs.

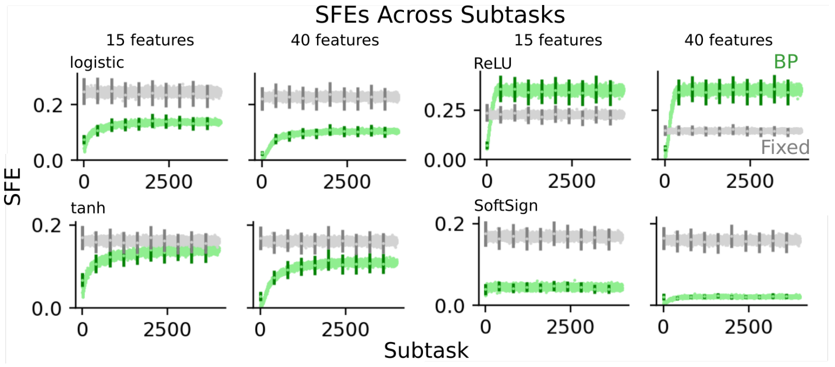

Figure 2: The dark-colored bars represent one standard error over 50 independent runs and are shown every 400 subtasks. The SFEs increased over subtasks with backprop.

The SFEs of the ReLU ANNs were approximately between 0.3 and 0.4 for all later subtasks.

On earlier subtasks, the SFEs of the logistic and tanh ANNs were significantly lower than the SFEs of the fixed representation ANNs (Figure 2). Then, the SFEs increased significantly within a few hundred subtasks to become slightly lower than the SFEs of the fixed representation ANNs.

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 (Figure 2).

The average product of the activation function derivative and the representation step size at the beginning of subtasks steeply decreased with all ANNs (Figure 3). The average product went to zero with ReLU activation functions, which indicates all the ReLU features were outputting zero, and only the output bias weight was being updated. The average product decreased by multiple orders of magnitude with the logistic and tanh activation functions, and the average product decreased the least with the SoftSign activation function. Like the SFEs, the average product at the beginning of subtasks changed mostly during early subtasks.

The proportion of calcified features at the beginning of subtasks increased for all ANNs except for the SoftSign ANNs (Figure 4). The SoftSign ANNs had no calcified features at the beginning of each subtask. The ReLU ANNs had only calcified features at the beginning of all later subtasks. The tanh and logistic activation function ANNs had an increase in calcified features at the beginning of early subtasks, but the increase was far less dramatic than the increase with the ReLU ANNs.

4.5 Experiment Conclusions

Based on the increase in SFEs over subtasks, we say backprop learned worse ANN weights on later subtasks compared to those from earlier subtasks. On

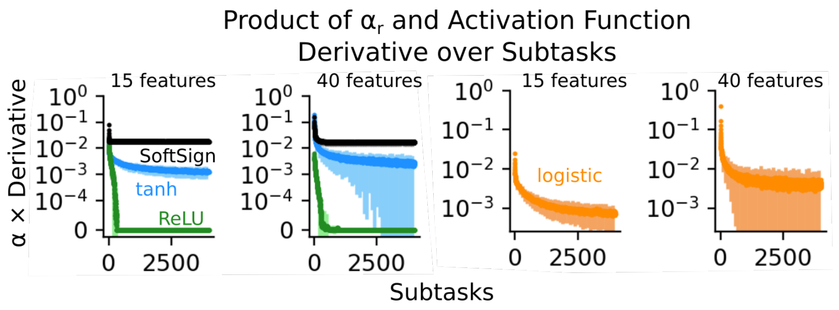

Figure 3: The two left plots are log scaled for all points above 10^{-4} and are linearly scaled for all points below. The light-colored bars represent one standard error over 50 independent runs, and they are plotted every 100 subtasks. The average product decreased rapidly over subtasks.

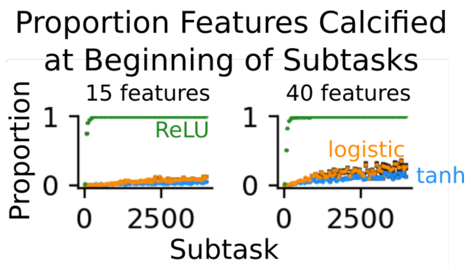

Figure 4: These points are plotted every 50th subtask. The black bars represent one standard error determined by 50 independent runs. The SoftSign ANNs (not shown) had zero calcified features at the beginning of each subtask.

later subtasks, backprop formed a representation that only output zero for the ReLU ANNs, which led to worse estimation performance than the ANNs with a fixed random representation. With logistic and tanh ANNs, backprop learned weights that had better performance than the fixed random representation ANNs but not by much. Finally, backprop learned the best performing ANNs with SoftSign ANNs as they had much lower SFEs on the task than the fixed random representation ANNs.

The decrease in the average activation function derivative at the beginning of subtasks and the increase in calcified features at the beginning of subtasks appeared to affect the learning ability of backprop. The initialization phase of backprop initialized the average activation function derivatives to some relatively high level and began all ANNs with no calcified features. But, at the beginning of later subtasks these properties were less present. The SoftSign ANNs had the least change in these properties at the beginning of subtasks and performed the best in terms of learning each data distribution quickly. The ReLU ANNs had the most dramatic loss of these properties at the beginning of subtasks and also had the poorest learning performance. The tanh and logistic

ANNs were middling in how much the properties were preserved and middling in terms of their learning performance.

Overall, backprop can have extremely poor learning performance when the data distribution changes, but its performance depends on the choice of the activation function. When learning later distributions, backprop did not appear to preserve the properties from its initialization phase that could have improved its learning performance. Introducing properties from the initialization phase of backprop throughout the task could help remedy the poor learning performance on later subtasks, and it is this solution we explore in the next experiment.

5 Continual Backprop on the GCFD Task

Knowing the poor learning performance of backprop on later subtasks of the GCFD task, we sought out an algorithm that can reliably adapt an ANN for all subtasks. Since backprop's initialization phase, which sets all features with small, random weights, is crucial for learning from a distribution quickly, we chose to evaluate versions of backprop that regularly introduce features with small, random weights. These versions of backprop are known as continual backprop algorithms.

5.1 Continual Backprop

Like backprop, a continual backprop algorithm initializes the features with small, random weights, and then uses stochastic gradient descent to update the ANN weights with examples. Unlike backprop, after each stochastic gradient descent update, a generate-and-test update occurs. The generate-and-test update is focused on replacing the least useful features with a potentially more useful feature.

There are parameters that must be set for the generate-and-test updates. 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 examples.

The generator of our generate-and-test updates generates features with weights sampled from a zero-centered normal distribution with a variance of 0.01. The variance of 0.01 is the same used to initialize the random feature weights of an ANN with backprop.

We try two different generate-and-test updates for continual backprop: *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.

Both generate-and-test updates 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.

5.2 Task

The task is the same GCFD task as the previous experiment. The target function has two target features, and the example inputs are of size 7. Each subtask is 15,000 examples long, and there are a total of 4,000 subtasks.

5.3 Learners

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 experiment.

We compared five different algorithms: backprop; continual backprop that uses partial random replace (BP+PRR); continual backprop that uses random generate and test (BP+RGT); partial random replace (PRR); and random generate and test (RGT). The learning algorithms that include backprop updates used the tuned step-size parameters from the previous experiment.

5.4 Evaluation Metrics

Our evaluation metrics were the same as the previous experiment and are motivated by the same reasons. For each combination of learning algorithm and ANN, the average error of the last 1000 samples of each of the 4,000 subtasks was measured. We call the average error of the last 1000 samples of a subtask the *subtask final error* (SFE). We noted the performance on the $4,000^{th}$ subtask, the average product of activation function derivative and representation step size at the beginning of subtasks, and the proportion of calcified features at the beginning of subtasks.

5.5 Hypotheses

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 backprop, because of the above mentioned potential interaction between backprop and the generate-and-test algorithms. We expected

BP+RGT to perform better than BP+PRR since BP+PRR has a trivial heuristic for judging a feature's utility and may discard useful features more frequently.

5.6 Results

The mean squared errors were mostly lower on the last subtask when BP+PRR or BP+RGT was used instead of backprop (Figure 5). For a given ANN, 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 backprop. However, with the SoftSign ANNs, BP+PRR resulted in higher mean squared errors compared to those from when backprop was used instead.

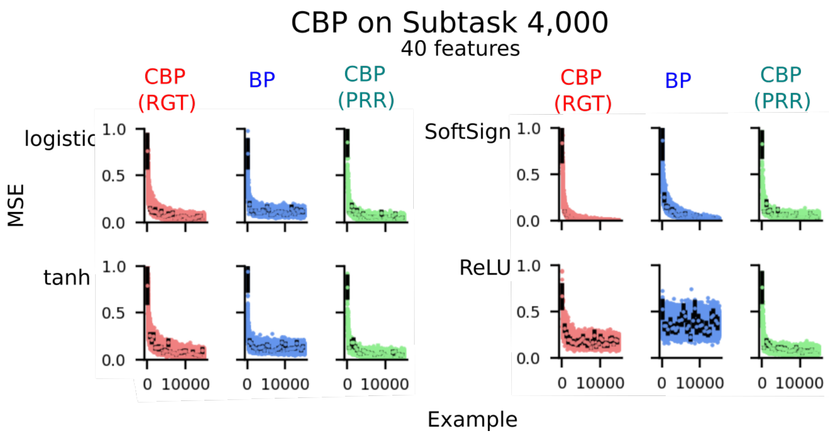

Figure 5: The mean squared errors on the last subtask for each size-40 ANN is shown here. The black bars represent one standard error determined by 50 independent runs, and they are plotted every 500 subtasks. ANNs trained with BP+RGT or BP+PRR tended to have lower error than backprop in most cases.

For almost all ANNs, the SFEs on later substasks were higher with backprop than with a continual backprop algorithm (Figure 6). 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 backprop. The SFEs of ReLU ANNs that learned with a continual backprop algorithm do not significantly exceed the 0.15 threshold that marked the performance of a fixed random representation. The logistic and tanh ANNs when learning with a continual backprop algorithm had much lower SFEs than those when learning with backprop.

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). This result was surprising as BP+PRR has a simpler tester than BP+RGT's. With the other ANNs, BP+RGT had lower SFEs than the SFEs when trained with BP+PRR.

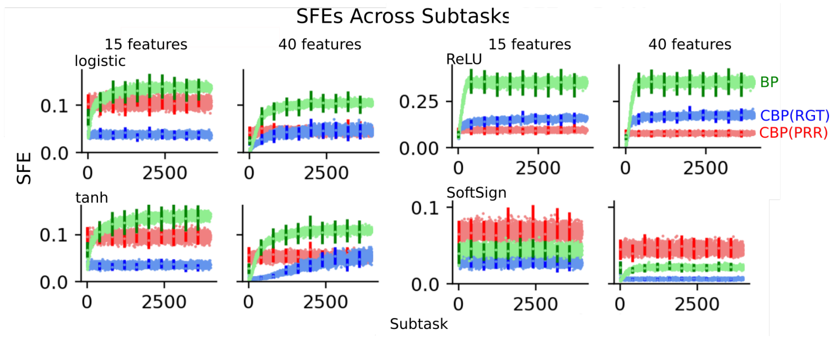

Figure 6: The dark bars represent one standard error determined by 50 independent runs, and they are plotted every 400 subtasks.

For a given learning algorithm and activation function, the SFEs on later subtasks were typically smaller or similar with more features (Figure 6). There were three exceptions to this trend: the tanh ANNs that updated with BP+RGT, the logistic ANNs that updated with BP+RGT, and the ReLU ANNs that updated with 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 backprop or BP+RGT (Figure 6). With all activation functions, BP+PRR typically had stable SFEs less than or around 0.1. However, ANNs that used backprop or BP+RGT had lower SFEs on early subtasks compared to their SFEs on later subtasks and had SFEs that exceeded 0.1 for some ANNs. 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 products of activation function derivative and representation step size at the beginning of subtasks (Figure 7) than those from when backprop was used. BP+PRR consistently had the highest average products at the beginning of subtasks.

ANNs that used BP+PRR or BP+RGT typically had fewer calcified features at the beginning of subtasks than when they used backprop (Figure 8). SoftSign ANNs had zero calcified features at the beginning of subtasks regardless of the learning algorithm.

The average product of activation function derivative and representation step size at the beginning of subtasks, and the proportion of calcified features at the beginning of subtasks did not predict whether a learning algorithm had lower or higher SFEs than another learning algorithm. The ranking of algorithms from highest average product at the beginning of subtasks to lowest was: BP+PRR, BP+RGT, and backprop (Figure 7). The ranking of algorithms from lowest proportion of calcified features to highest was the same (Figure 8). However, when it comes to ranking learning algorithms based on their SFEs, the algo-

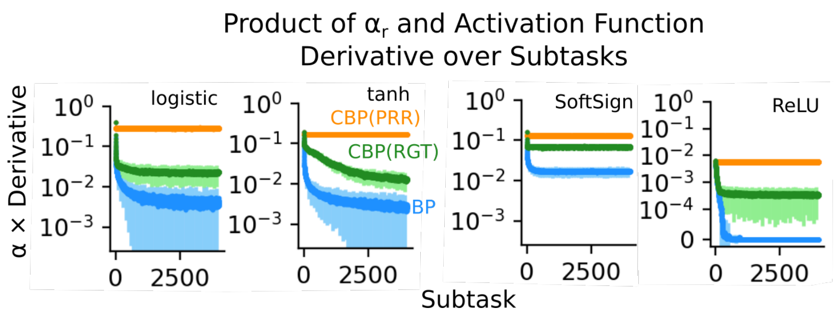

Figure 7: These results are for the 40-feature ANNs. The rightmost plot is log-scaled above 10^{-4} and linear-scaled below. The light-colored bars represent one standard error determined by 50 independent runs and are plotted every 100 subtasks.

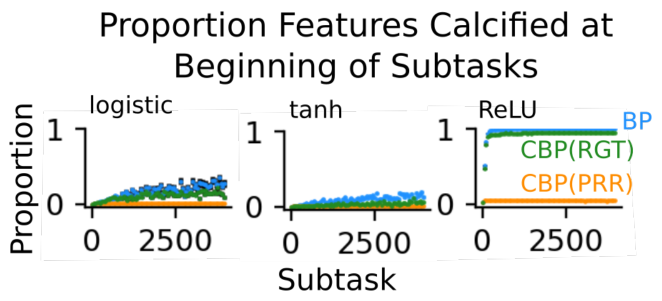

Figure 8: These points are plotted every 50th subtask for 40-features ANNs. The black bars represent one standard error determined by 50 independent runs. The SoftSign ANNs (not shown) always had zero calcified features at the beginning of subtasks.

rithms 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 9).

5.7 Experiment Conclusions

For most ANNs, updating with a continual backprop algorithm adapted ANNs better for each data distribution than when they were updated with backprop, as evidenced by the lower SFEs with a continual backprop algorithm compared to those with backprop. With SoftSign ANNs, using BP+RGT led to near zero SFEs on later subtasks, which was lower than the SFEs with backprop, but using BP+PRR led to SFEs greater than those with backprop. For all other ANNs, either continual backprop algorithm greatly improved the estimation

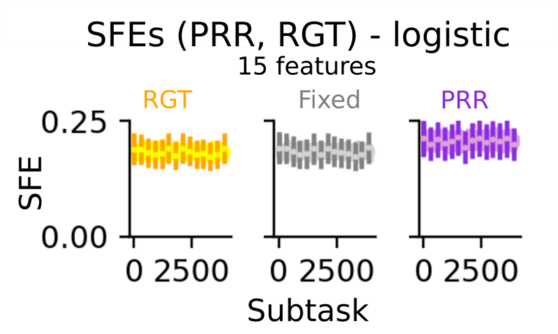

Figure 9: The dark-colored bars represent one standard error determined by 50 runs, and they are plotted every 100 subtasks. For all ANNs (many not shown), RGT and PRR had SFEs similar to or higher than those of the fixed representation.

performance of the learned ANNs from when they were updated with backprop. The ReLU ANNs do not have catastrophically poor learning performance with continual backprop algorithms, although the ReLU ANNs that learned with BP+RGT performed similar to a fixed random representation. This poor performance makes sense, since BP+RGT's tester was not suited for ReLU activation functions.

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

The continual backprop algorithms were more successful than backprop at preserving the properties set by the initialization phase: the activation function derivative at the beginning of subtasks and the proportion of calcified features at the beginning of subtasks. BP+PRR led to the best preservation with near zero calcified features at the beginning of subtasks and activation function derivatives that began close to the point they were initially set to. However, how an algorithm maintained the activation function derivatives and the proportion of calcified features may come with poorer ANN adaptation as it did for the SoftSign ANNs that used BP+PRR instead of backprop. These properties can be beneficial to discovering useful features if introduced carefully.

Overall, backprop's learning performance on subtasks was improved with generate-and-test algorithms. The generate-and-test algorithms we tested updated infrequently, were computationally cheap, and were untuned. Yet, the performance gains of the continual backprop algorithms were great.

6 Discussion

In this paper, we clearly demonstrated the weakness of backprop when adapting an ANN on a sequence of different data distributions. Over many data distributions, the performance of the ANNs learned with backprop can perform worse or only slightly better than a fixed random representation ANN. Using a Soft-Sign activation function enabled backprop to adapt the ANN quicker than with other activation functions, while using ReLU activation functions led backprop to form an extremely poor representation that outputted a zero vector for all inputs.

We sought to remedy this weakness of backprop by combining it with a generate-and-test algorithm to form continual backprop. Although each generate-and-test algorithm on their own performed similar or worse compared to a fixed random representation ANN, the continual backprop algorithms demonstrated the ability to quickly adapt ANNs on later data distributions. More work needs to be done to establish reliable generators and testers, but continual backprop algorithms can be greatly beneficial.

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.

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

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*.

Dohare, S. (2020). *The Interplay of Search and Gradient Descent in Semi-stationary Learning Problems*. Master's thesis. University of Alberta.

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

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

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

Glorot, X., Bengio, Y. (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

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.

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

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.

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

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.

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.

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.

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.

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

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.

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

Zhou, G., Sohn, K., Lee, H. (2012). Online Incremental Feature Learning with Denoising Autoencoders. In *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics 22*, pp. 1453–1461. PMLR.

7 Appendix

7.1 GDFC Task Calculations

A calculation can be done to determine how often a target feature will show up on subsequent subtasks. Let i be the size of the binary input vector, and let t be the number of target features. The number of unique target features with -1 and 1 weights is 2^i and the number of unique target functions is $(2^i)^t = 2^{ti}$. $(\frac{2^i-1}{2^i})^t$ 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.

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 Bernoulli process: $X_1, X_2, X_3, ...,$ where X_s, $s \in \mathbb{N}$, is a random variable equal to one if the target feature reappears on the s^{th} subtask after $(P(X_s = 1) = p)$. X_s is equal to zero otherwise $(P(X_s = 0) = 1-p)$. For such a Bernoulli process, the expected earliest s when $X_s = 1$ is $i = 1/p$. So, for our experiments, where $p \sim 0.016$, target features are expected to reappear every ~ 64 subtasks.

7.2 Optimized Backprop Parameters

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