COSC490 Project: Pseudorehearsal

Robbie Cook

October 2018

1 Introduction

In artificial neural networks, catastrophic forgetting (also known as catastrophic interference) is where, when a new item is learned, there is a catastrophic detrimental effect on the previous learning. That is, old learning is forgotten when new learning is introduced [1]. There have been various solutions introduced to avoid catastrophic forgetting involving retaining memory by making some connections resistant to new learning [2] [3] [4]. Hidden layer modification with orthagonalisation has been also proposed [5] as a solution to catastrophic forgetting. It is likely that adding hidden layers makes this solution too complex and arbitrary to implement in Deep Neural Networks (DNNs).

Rehearsal as a technique was first introduced by Roger Ratcliff in 1990 [6]. Rehearsal and pseudorehearsal are methods which allows for synaptic plasticity whilst still minimising catastrophic forgetting. Using rehearsal to avoid the problem of catastrophic forgetting involves relearning old information when new information is learned [6]. The rehearsal method of avoiding catastrophic forgetting has been refined using sweep rehearsal [7], and pseudorehearsal [7].

This report addresses three topics. First, a review of literature relating to pseudorehearsal as it was originally developed in a basic multilayer perceptron (MLP). Second, a review of literature on the current uses of pseudorehearsal in modern DNNs, for instance in reinforcement learning and image recognition – two popular applications of deep networks. Third, an empirical investigation between these endpoints, studying the properties of pseudorehearsal as layers are added to a basic MLP to make it deeper and the impact of some deep methods in this context.

2 Literature Review

Pseudorehearsal is a biologically-inspired [8] rehearsal technique which doesn't require any knowledge/storage of previously learned items to mitigate catastrophic forgetting. In the original implementation of pseudorehearsal [7], a base population of items is learned by a network, and each intervening item is trained in a buffer with randomly generated pseudoitems (consisting of random input + corresponding network output) until the intervening item is learned. There were

many different pseudorehearsal strategies tried, and the most effective is sweep pseudorehearsal (see Figure 1) where new random pseudoitems are selected for the buffer for each epoch. The buffer size in the original implementation was 4. See [7] for more specific details about the implementation of pseudorehearsal.

2.1 Biological Inspiration

The biological inspiration behind rehearsal and pseudorehearsal comes from the idea that the brain consolidates the information of the day during sleep [8] [9] [7]. The hypothesis suggested in [8] is that catastrophic forgetting occurred in natural brains, and one of the many roles of sleep is to consolidate (rehearse) memory to mitigate the problem – especially during the Rapid Eye Movement (REM) stage of sleeping. REM sleep is a period of sleep involving large brain activity. When REM sleep is disrupted, memory is impaired [10].

3 Pseudorehearsal in simple MLPs

3.1 Pseudorehearsal in Reinforcement Learning

Pseudorehearsal is has been very influential in reinforcement learning [11] [12] [13] [14]. "Experience Replay" in reinforcement learning [11] [14] is a very similar technique to pseudorehearsal in a supervised learning environment. Rehearsal is useful for reinforcement learning in particular because the network must explore a range of policies from an environment sequentially. Rehearsing past experiences can aid deep Q-learning strategies to develop a better estimate of the quality of each potential action. Other machine learning tasks like image recognition don't necessarily require serial learning, because entire populations are generally available from the inception, or at least new items are learned with the entire dataset of previous items. Because the effectiveness of policies in a Markov Decision Process (MDP) are evaluated and learned in serial, rehearsal can speed up learning in reinforcement learning. For example, pseudorehearsal helps to preserve information from scenarios that occur rarely – e.g. in the difficult task of 2D pole-balancing, initially there are only a few good movements to make, and a multitude of bad ones [13]. Rehearsal helps the network remember the information from the few times the pole did not fall.

An example of a particular network structure used in reinforcement learning is the actor-critic model [15], which is a form of temporal difference learning on a Markov Decision Process. The critic predicts the quality of the current policy made by the actor on the environment, and the actor makes and refines its policies based off a reward factor from the critic. [13] uses an actor-critic reinforcement learning algorithm, with pseudorehearsal applied to the underlying MLPs of the actor and the critic. The task used was the pole-balancing task from [16]. The results showed that pseudorehearsal definitely improved learning performance. The same authors previously tried using pseudorehearsal with function approximation, but the success of pseudorehearsal varied hugely

depending on the parameter values [12]. The three parameters which impacted pseudorehearsal were the learning rate, number of pseudoitems, and the relearning gap (how often new pseudoitems were generated).

3.2 Pseudorehersal in DNNs

A key component in preserving the effectiveness of pseudorehersal for complex tasks, like image recognition, is to pick good pseudoitems. Well-chosen pseudoitems better mimic the original dataset than completely random pseudoitems. An Otago paper published this year, [17], effectively used a Generative Adverserial Network (GAN) [18] attached to a DNN to generate good pseudoitems. The DNNs task was image classification (image *to* label) on a range of different datasets: MNIST (handwritten digits [19]); CIFAR-10 [20]; and SVHN (recognising house numbers) [21].

[17]'s model was made of two components: a DNN to solve the classification task, and a GAN to generate pseudoitems to minimise catastrophic forgetting in the DNN. The DNN used is a varient of "SuperVision" [22], which won ILSVRC2012 [23].

GANs are an unsupervised learning network with two sub-networks – a generative model, which attempts to recreate input data; and a discriminator, which calculates a probability that a sample came from the generative model or the training data. The reward factor from the discriminator is a scalar value representing its surity that the item came from the training set, which means both models can use backprop – the generator's teaching value is the discriminator's probability, and the discriminator's teaching value is given by whether the item was in the training set or from the generator [18].

The effectiveness of [17] is remarkable – with sweep pseudorehearsal, it loses less that 1.5% accuracy in the worst cases for each intervening trial, and sometimes *gaining* accuracy on the base population. This is far better than even the original algorithm (Figure 1) when applied to MLPs, and it shows that using non-random approaches to picking pseudoitems is very rewarding. It could even be more rewarding than refining the rehearsal strategies themselves.

Pseudorehearsal has also been applied to DNNs with complementing Recurrent Neural Networks (RNNs) [24] in a similar way to [17]. Both papers use an existing DNN with an auxiliary network to generate pseudoitems for the DNNs. However, [24] uses a RNN as its auxiliary. The RNN that they use is DRAW [25], an image generator that uses a model of an "encoder", which compresses input images, and a "decoder" that recreates the images. The DNN used is a custom-built classification Convolutional Neural Network (CNN). The base population is numbers 0-4 of MNIST, and the intervening trails are the remaining numbers (5-9). The results of this technique are also impressive, as 95% accuracy is maintained, even after 5 interveining trials.

Having examined literature relating to pseudorehearsal in the original MLP and current deep or convolutional neural networks (CNNs), we now turn to our third goal – an empirical investigation of pseudorehearsal as layers are added to a basic MLP to make it deeper.

3.3 Rehearsal Methods

Before continuing, the various rehearsal techniques used in Section 5 should be defined. Let $f : \mathbb{R}^h \mapsto \mathbb{R}$ be a feed-forward network for a regression task. Let $B = \{b_0, b_1, ..., b_n\}$ be a base population of items (a population trained non-sequentially on f). b_k is a set of input, target vectors for f to learn i.e., $b_k = \{input, target\}$, where $input \in \mathbb{R}^h$ and $target \in \mathbb{R}$. And let $I = \{i_0, i_1, ..., i_k\}$ be a set of intervening items (interventions) to be learned sequentially after f has been trained on the base population, B.

3.3.1 No rehearsal

No rehearsal (catastrophic forgetting) is where B is trained on a network f, and then intervening items (interventions) $i_k \in I$ are trained in sequence on f. The graphed results are the quality of f on B after training I.

3.3.2 Recency rehearsal

Recency rehearsal is where a buffer of the last three learned items is stored, and each intervening item learned is trained in a buffer with the three most recently trained items.

That is, f is first trained on B. Then, for intervention i_0, recency rehearsal trains f on $\{b_{n-2}, b_{n-1}, b_n, i_0\}$ to criteria. For intervention i_1, f is trained on $\{b_{n-1}, b_n, i_0, i_1\}$, and for intervention i_2, f is trained on $\{b_n, i_0, i_1, i_2\}$, and so on. Recency rehearsal requires the storage of the last four items learned, which is not expensive. However, recency rehearsal does not perform very well, as can be seen in Figure 1.

3.3.3 Random rehearsal

Random rehearsal is where a base population is learned, and then all the previously learned items stored. To learn a new, intervening item (intervention), a random subset of the base population is selected from the previously learned items, and the network is trained in a buffer with both the intervening item and the subset.

That is, f is first trained on B. Then, for an intervening item i_x, a new buffer $X = \{b_{i \sim U(0,|B|)}, b_{j \sim U(0,|B|)}, b_{k \sim U(0,|B|)}, i_x\}$ is created, and f is trained on X.

3.3.4 Sweep rehearsal

Sweep rehearsal is similar to random rehearsal, except that the buffer X is regenerated for every epoch. That is, f is trained on B, and then a buffer $X = \{b_{i \sim U(0,|B|)}, b_{j \sim U(0,|B|)}, b_{k \sim U(0,|B|)}, i_x\}$ is generated. For intervention i_k, f is trained on X for *one epoch*, and then a new set $X = \{b_{i \sim U(0,|B|)}, b_{j \sim U(0,|B|)}, b_{k \sim U(0,|B|)}, i_x\}$ is generated, and f is trained on it. Training continues until i_k is trained to criterion.

3.3.5 Random pseudorehearsal

Random pseudorehearsal is where a network is trained on a base population, and then for every intervening item trained, three random input vectors are generated as input to the network. These input vectors are then fed through a network and combined with their respective input vector to form *pseudoitems*, which are learned in a buffer with the intervening item until the intervening item is trained to criterion.

That is, f is first trained on B. Then, for each intervention $i_k \in I$, a buffer $X = \{r_0, r_1, r_2, i_k\}$ is created. $r_k = \{X, f(X)\}$, where $X = \{x_0, x_1, ..., x_h\}$, and $x_u \sim U(0, 1)$. f is trained on X until i_k is trained to criterion.

3.3.6 Sweep pseudorehearsal

Sweep pseudorehearsal is the same as random pseudorehearsal, except that a new X is generated for each epoch of f's training for each $i_k \in I$ (i_k features in each buffer, and the same for sweep rehearsal).

4 Method

The original pseudorehearsal and rehearsal algorithms [7] have been reimplemented in 2 networks – one based off TensorFlow [26], and one based off a ANN library called nimblenet [27]. The results from this implementation are very similar to those of [7], and are shown in Figure 1. The differences that do exist can probably be explained by my looser error criterion (RMS 0.04 as opposed to [7] and [6]'s requirement that each output be within 6% of its target).

Unless otherwise noted, the base populations are pairs random, binary vectors of length 32. The structured datasets are from the UCI Machine Learning repository [28]. Specifically, they are the Boston Housing dataset [29], the Iris dataset [30], the Airfoil Self-Noise dataset [31], and the Wine Quality dataset

'Mean goodness' is an indication of how close the output of a network is to a teaching signal (0 indicating 'chance' performance – the likely performance of a network without training – and 1 indicating the output replicating the teacher). It was introduced by [6]. After the original implementation of the rehearsal strategies was achieved, the metrics were switched to Mean Absolute Error (MAE) for regression tasks and categorical accuracy for classification tasks, as these are very common metrics.

The non-random strategy chosen to generate pseudoinputs for this paper was an autoencoder auxiliary network. This worked by training the initial populations as usual, but also training the autoencoder on the inputs of those populations. When each intervention is trained, the pseudoitems were generated from the output of the autoencoder given random input. These pseudoitems were then used with pseudorehearsal instead of random input straight to the pseudorehearsal buffer. This can be seen in the latter graphs that are have a legend 'with autoencoder.'

5 Results

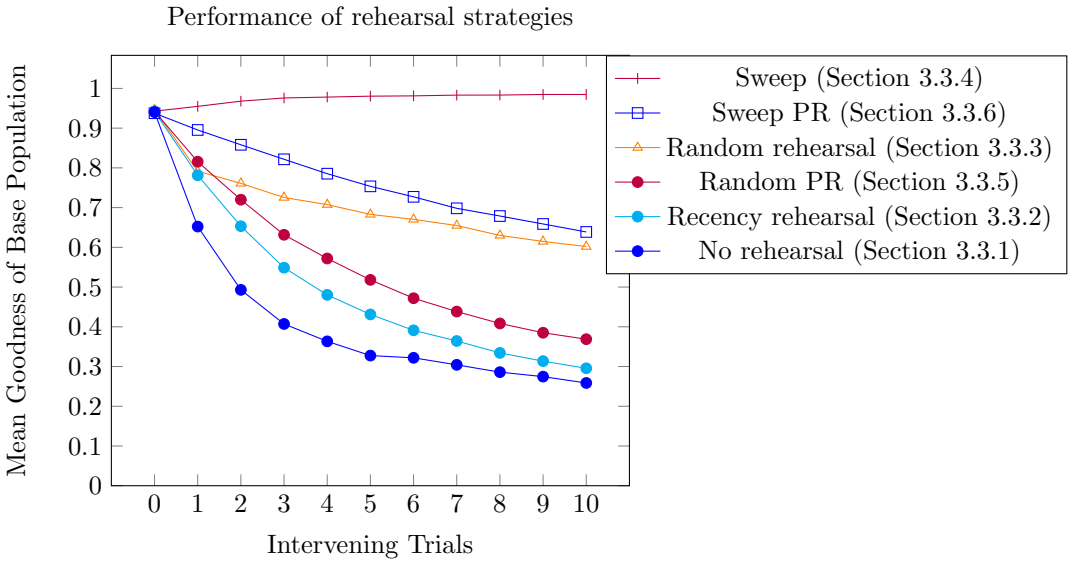


Figure 1: Summary of all rehearsal strategies. Data was averaged over 50 populations with an error criterion (Root Mean Square) of 0.04. The parameters and architecture of the network were modelled as close as possible to [7]h. The network is a 32-16-32 network with random binary inputs (uniform distribution).

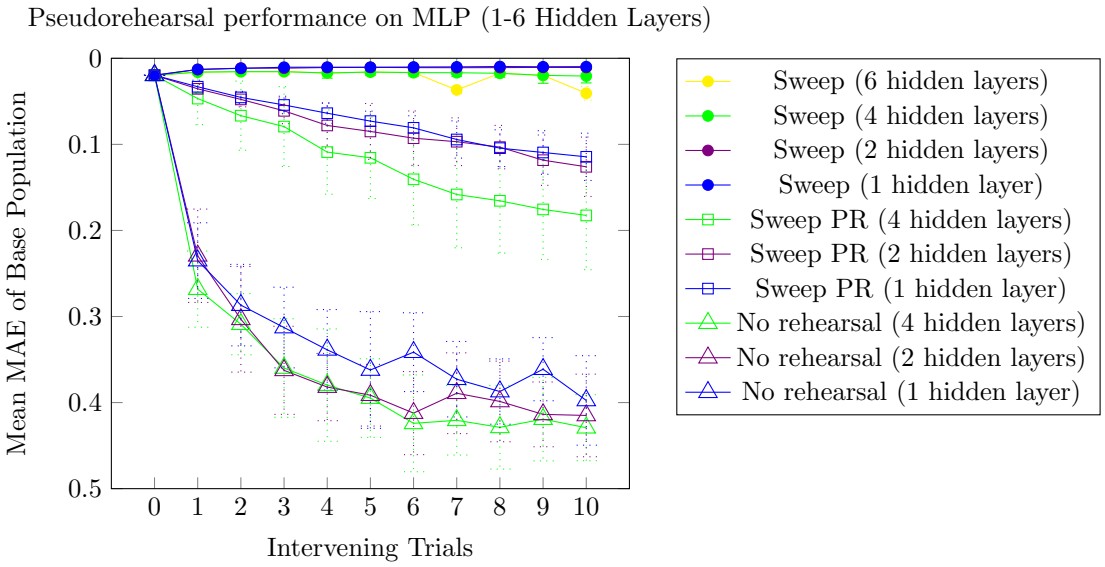


Figure 2: Experiment repeated 20 times. The cost function used is Mean Squared Error (MSE), and the metric function used was the Mean Absolute Error (MAE) (the criteria was ¡ 0.02 MAE). The ReLU layers are the hidden layers. Base population: 5 items, 10 interventions – 32-16-....-16-32 network used. The maximum number of epochs for learning the base population was 200,000 epochs, and the maximum number of epochs to learn each intervention was 70,000.

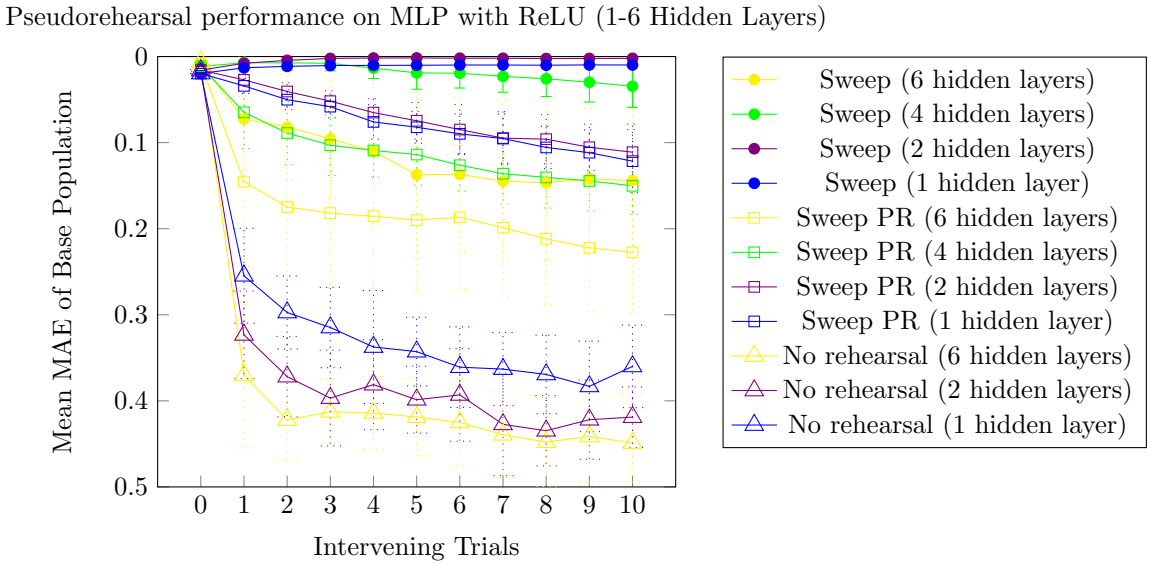


Figure 3: Experiment repeated 20 times. The cost function used is Mean Squared Error (MSE), and the metric function used was the Mean Absolute Error (MAE) (the criteria was ¡ 0.02 MAE). The ReLU layers are the hidden layers. Base population: 5 items, 10 interventions – 32-16-....-16-32 network used. The maximum number of epochs for learning the base population was 200,000 epochs, and the maximum number of epochs to learn each intervention was 70,000.

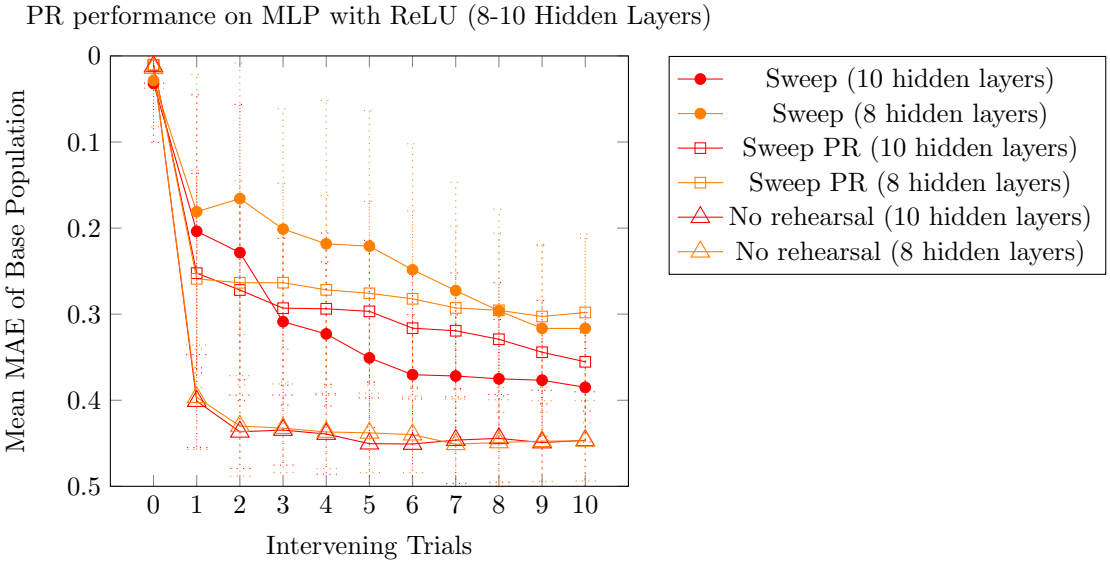


Figure 4: Experiment repeated 20 times. The cost function used is Mean Squared Error (MSE), and the metric function used was the Mean Absolute Error (MAE) (the criteria was ¡ 0.02 MAE). The ReLU layers are the hidden layers. Base population: 5 items, 10 interventions – 32-16-....-16-32 network used. The maximum number of epochs for learning the base population was 200,000 epochs, and the maximum number of epochs to learn each intervention was 70,000.

Impact of depth on PR performance on a simple MLP with Dropout and ReLU

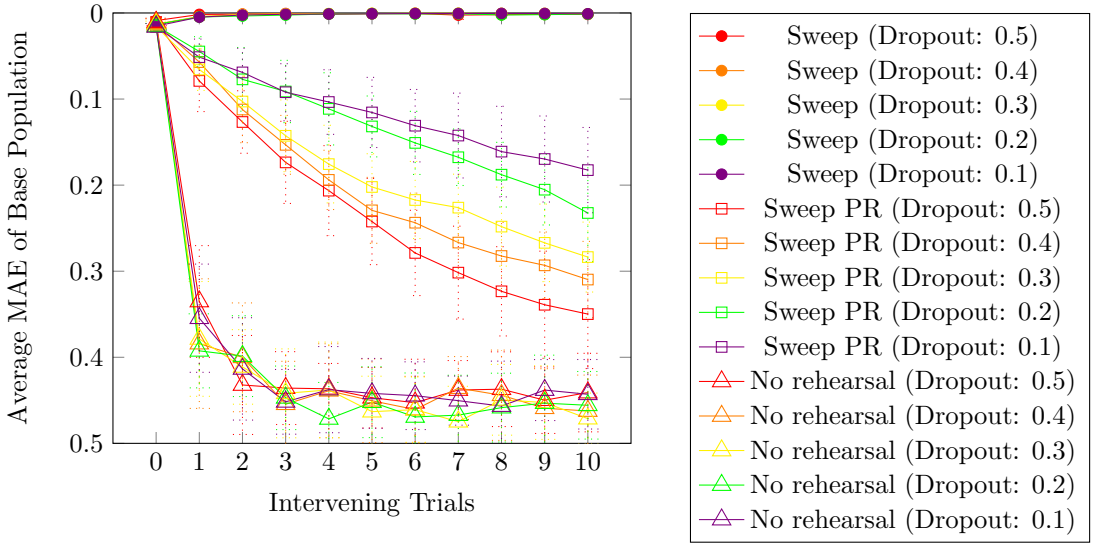


Figure 5: Experiment repeated 20 times. The x in (Droput: x) refers to p, the probability that a unit is used in training. The cost function used is Mean Squared Error (MSE), and the metric function used was MAE (the criteria was 0.02 MAE). The single hidden layer is ReLU. Base population: 5 items, 10 interventions – 32-16-32 network used. The maximum number of epochs for learning the base population was 200,000 epochs, and the maximum number of epochs to learn each intervention was 70,000

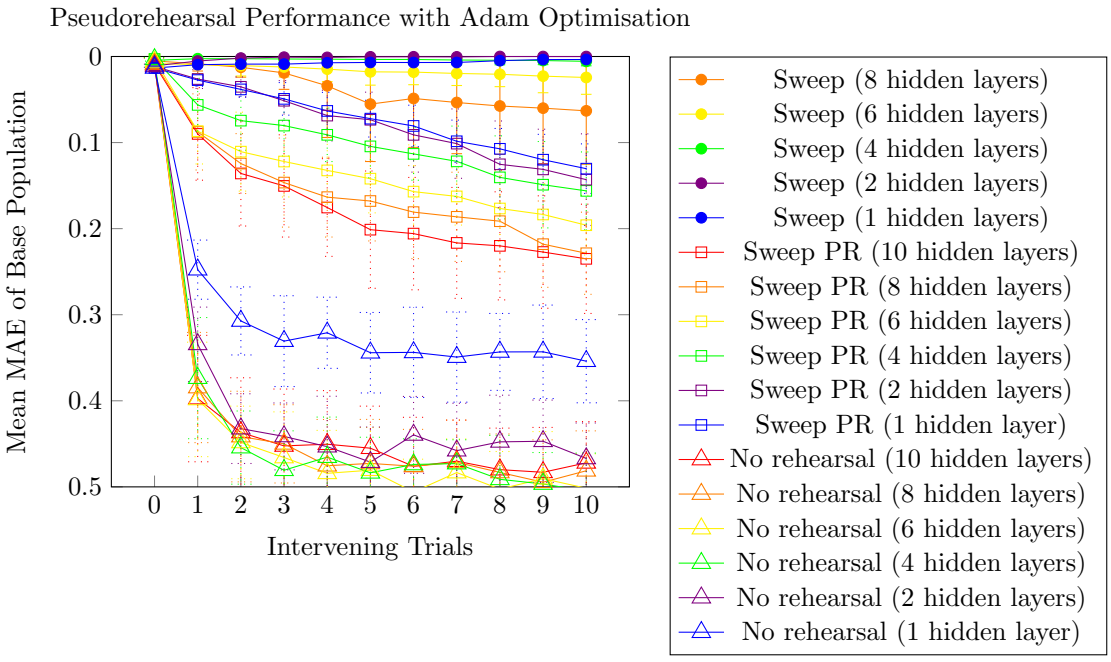


Figure 6: Experiment repeated 20 times. The sweep experiment for 10 hidden layers was accidentally cancelled, but the data was not critical for this graph. The cost function used is Mean Squared Error (MSE), and the metric function used was the Mean Absolute Error (MAE) (the criterion was 0.02 MAE). The ReLU layers are the hidden layers. Base population: 5 items, 10 interventions – 32-16-....-16-32 network used. The maximum number of epochs for learning the base population was 200,000 epochs, and the maximum number of epochs to learn each intervention was 70,000.

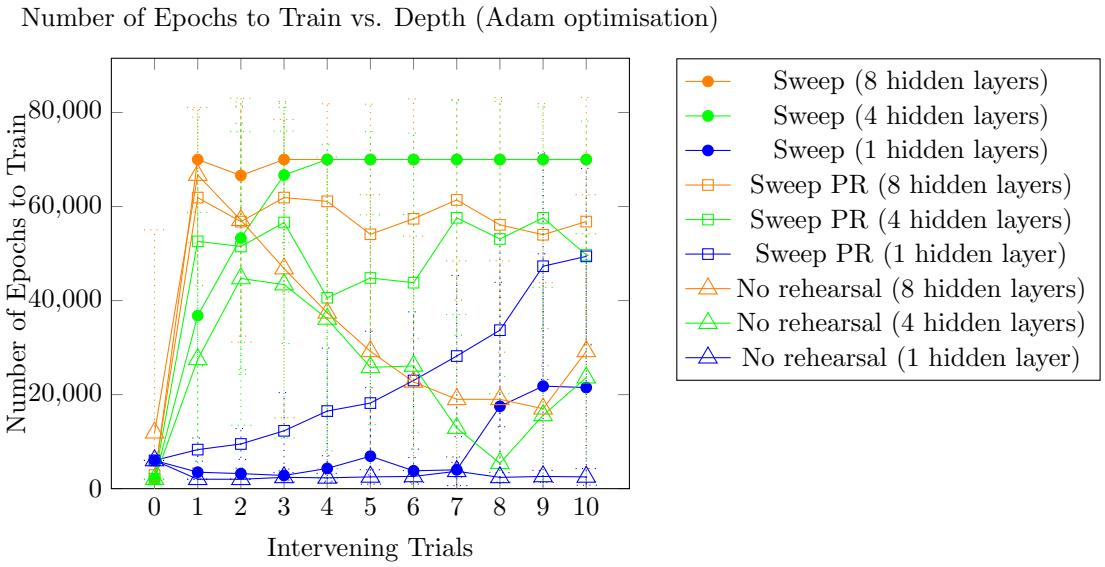


Figure 7: Experiment repeated 20 times. The cost function used is Mean Squared Error (MSE), and the metric function used was the Mean Absolute Error (MAE) (the criterion was 0.02 MAE). The ReLU layers are the hidden layers. Base population: 5 items, 10 interventions – 32-16-....-16-32 network used. The maximum number of epochs for learning the base population was 200,000 epochs, and the maximum number of epochs to learn each intervention was 70,000.

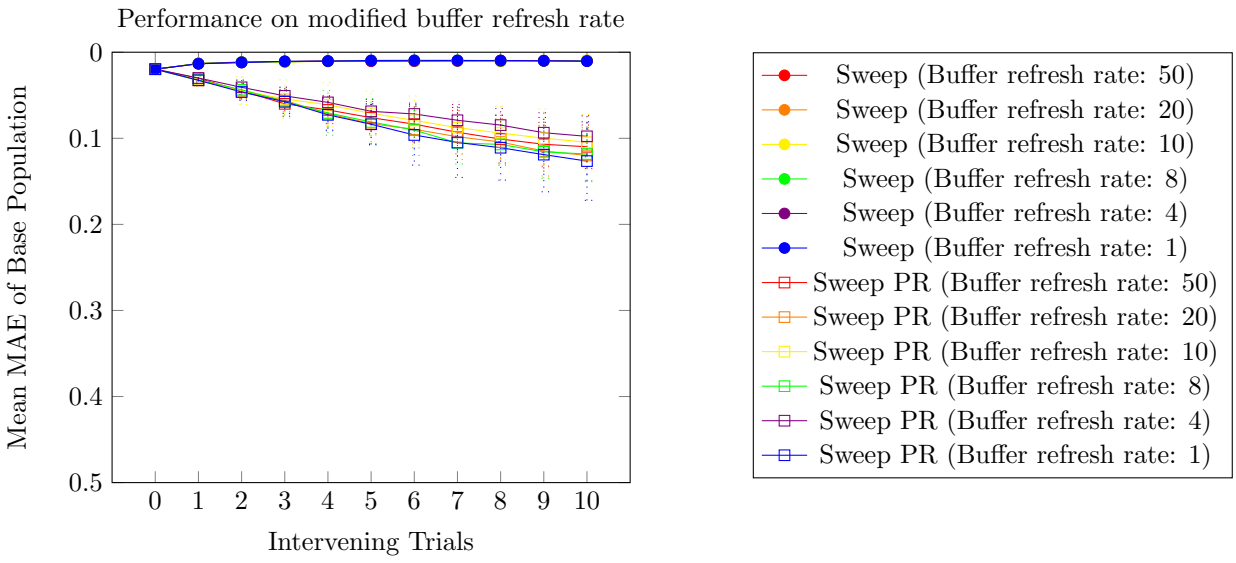


Figure 8: Experiment repeated 20 times. The cost function used is Mean Squared Error (MSE), and the metric function used was the Mean Absolute Error (MAE) (the criteria was ¡ 0.02 MAE). The ReLU layers are the hidden layers. Base population: 5 items, 10 interventions – 32-16-....-16-32 network used. The maximum number of epochs for learning the base population was 200,000 epochs, and the maximum number of epochs to learn each intervention was 70,000.

Epochs taken to learn intervention for modified buffer refresh rate

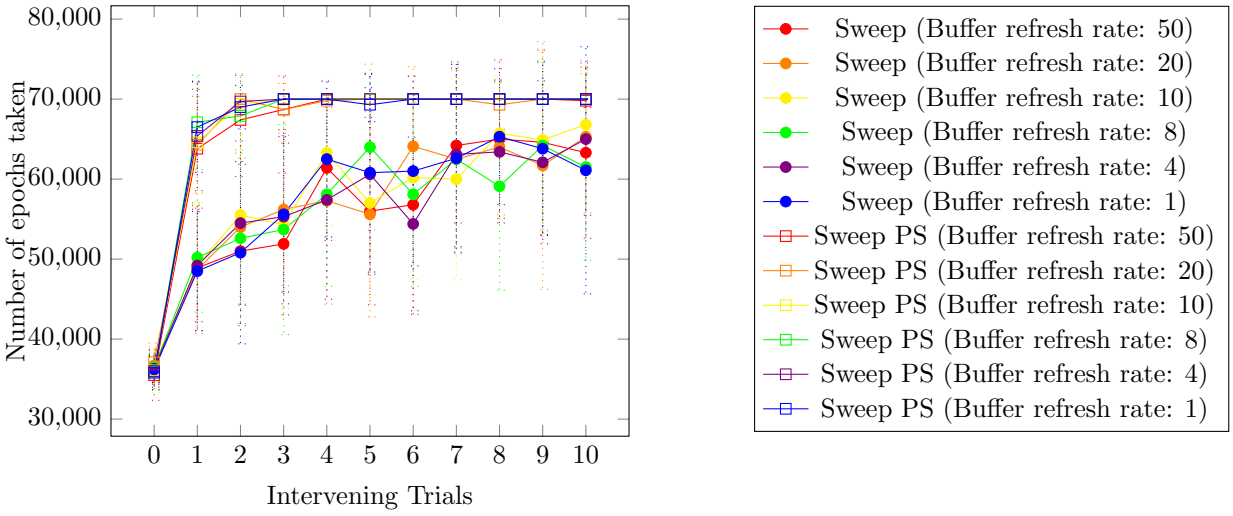


Figure 9: Experiment repeated 20 times. The cost function used is Mean Squared Error (MSE), and the metric function used was the Mean Absolute Error (MAE) (the criteria was ¡ 0.02 MAE). The ReLU layers are the hidden layers. Base population: 5 items, 10 interventions – 32-16-....-16-32 network used. The maximum number of epochs for learning the base population was 200,000 epochs, and the maximum number of epochs to learn each intervention was 70,000.

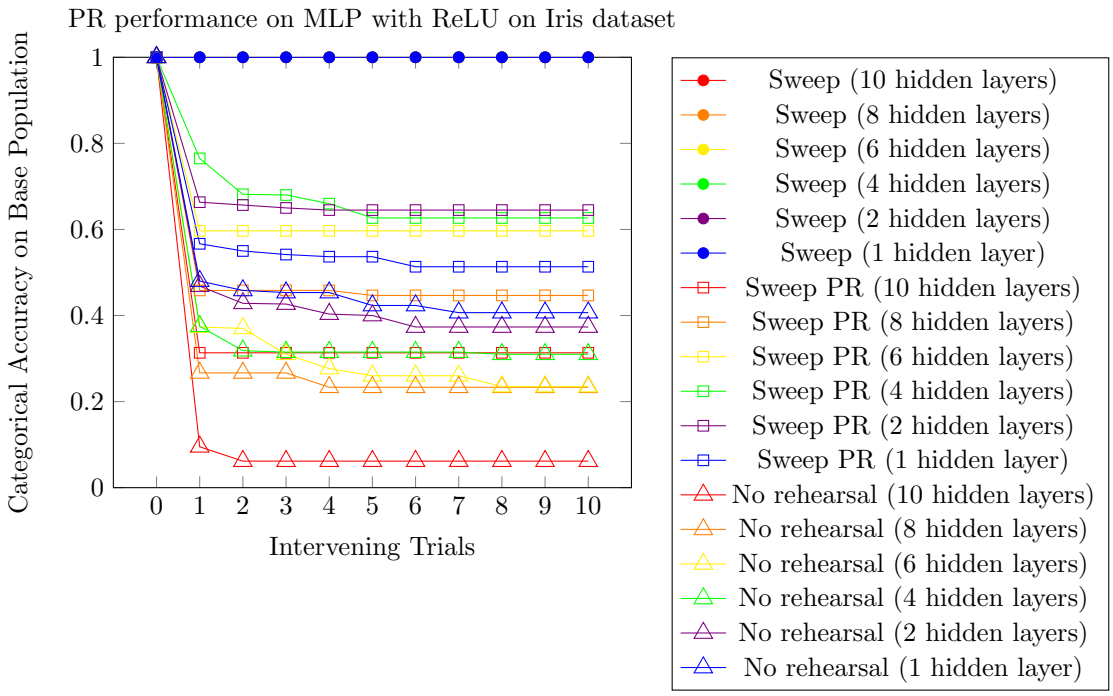


Figure 10: Experiment repeated 30 times. The cost function used is binary cross entropy (from one-hot encoded vectors), and the metric function used was categorical accuracy (the criteria was an accuracy 0.9 (out of 1)). The standard deviation was very high – between 0.2 and 0.4. The hidden layer is relu, but the rest, sigmoid. Base population: 20 items, 10 interventions – 32-16-32 network used. The base population was Iris-setosa and the interventions were Iris-versicolor. The maximum number of epochs for learning the base population was 200,000 epochs, and the maximum number of epochs to learn each intervention was 70,000.

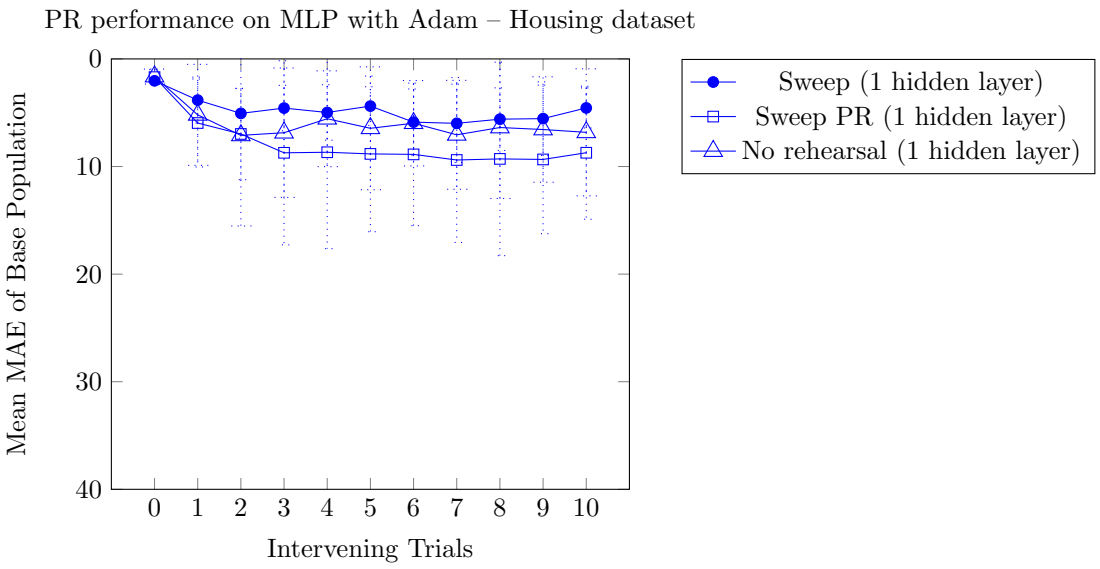


Figure 11: This experiment uses items from the UCI Housing dataset for both the base population and interventions. This caused a lesser drop catastrophic forgetting than expected. See Figure 14. The cost function used is Mean Squared Error (MSE), and the metric function used was the Mean Absolute Error (MAE). The criterion was a MAE of 5. There were no ReLU layers used. Base population: 20 items, 10 interventions – 13-32-1 network used. The maximum number of epochs for learning the base population was 200,000 epochs, and the maximum number of epochs to learn each intervention was 70,000. Experiment repeated 50 times.

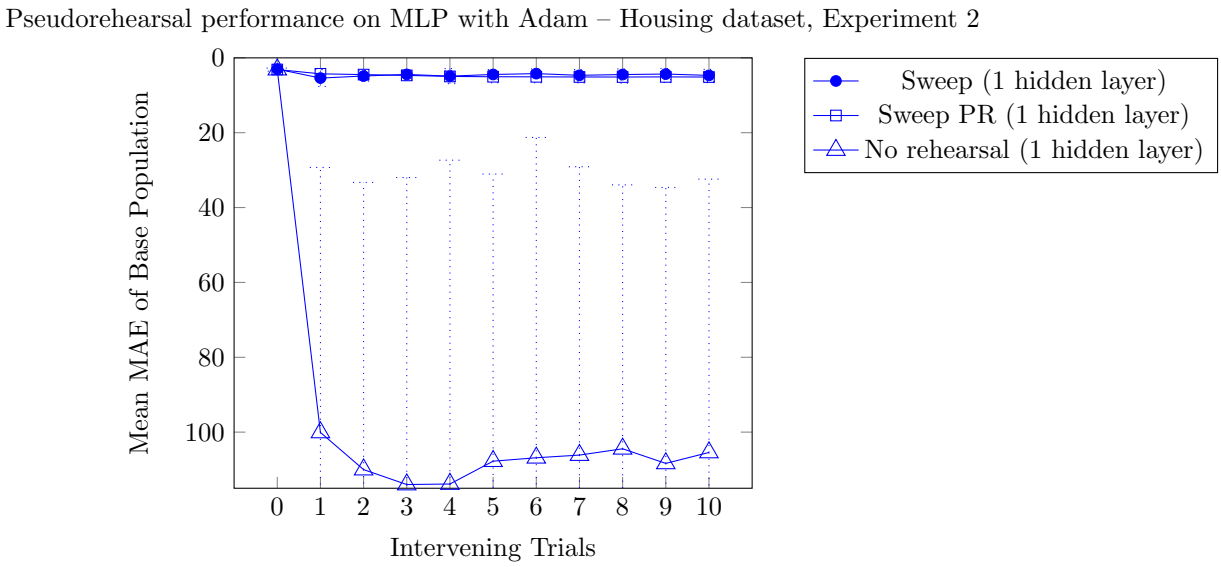


Figure 12: This experiment used a binary random population for the interventions, and a real, structured, non-binary base population. The cost function used is Mean Squared Error (MSE), and the metric function used was the Mean Absolute Error (MAE) (the criteria was 5 MAE). There were no ReLU layers used. Base population: 20 items, 10 interventions – 13-32-1 network used. The maximum number of epochs for learning the base population was 200,000 epochs, and the maximum number of epochs to learn each intervention was 70,000. Experiment repeated 50 times.

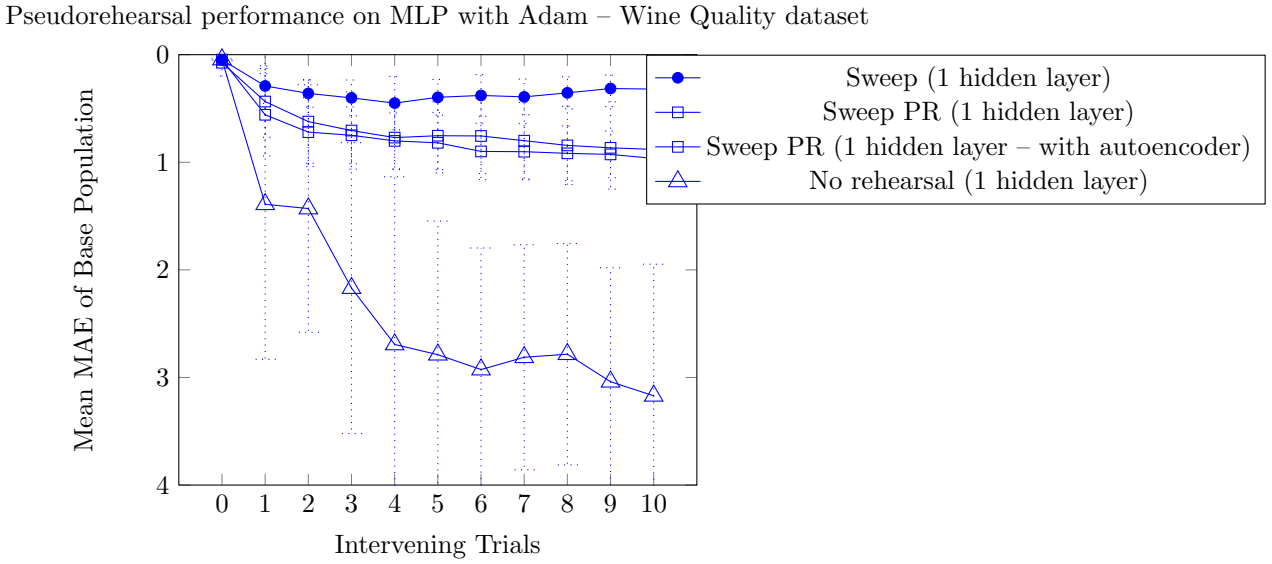


Figure 13: The cost function used is Mean Squared Error (MSE), and the metric function used was the Mean Absolute Error (MAE) (the criteria was 5 MAE). There were no ReLU layers used. Base population: 20 items, 10 interventions – 5-32-1 network used. The maximum number of epochs for learning was 20,000 for everything. Experiment repeated 50 times.

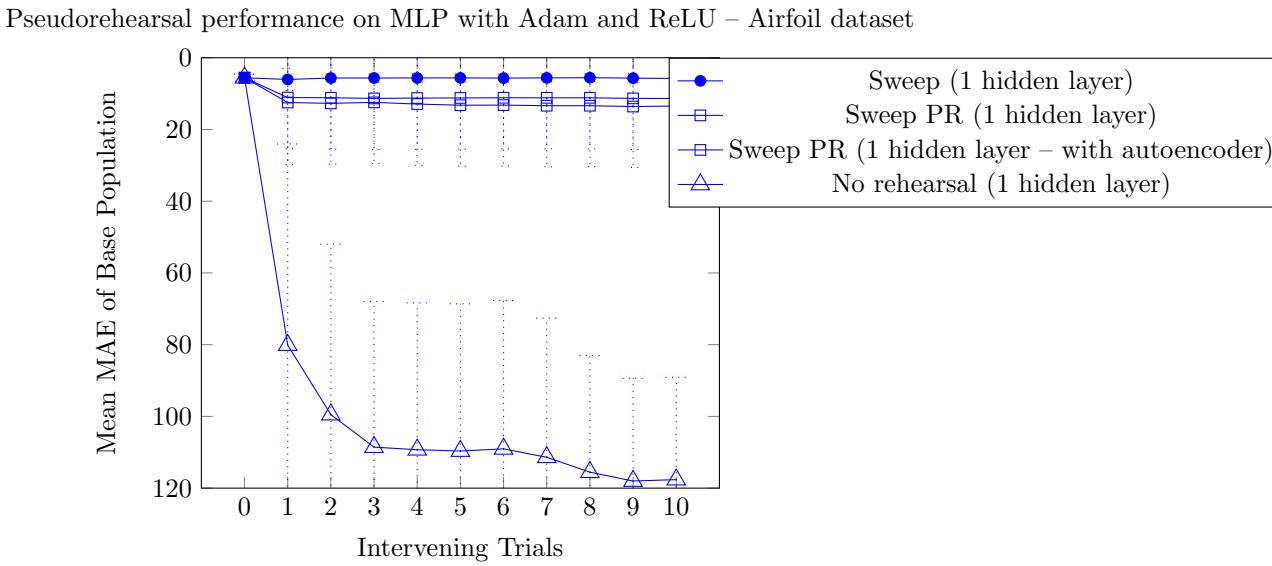


Figure 14: Experiment repeated 50 times. The cost function used is Mean Squared Error (MSE), and the metric function used was the Mean Absolute Error (MAE) (the criteria was 0.05 MAE). There were no ReLU layers used. Base population: 20 items, 10 interventions – 5-32-1 network used. The maximum number of epochs for learning the base population and interventions was 20,000 epochs.

6 Discussion

6.1 Reimplementing Psuedorehearsal

Pseudorehearsal is implemented as per [7] in Figure 1, with goodness given as the metric. In the MLP experiments in Figure 2, we can see what the effect of depth is on pseudorehearsal. The interesting thing about pseudorehearsal and depth is that at some point the rehearsal strategy will crash (so long as there is a limit to the number of epochs allowed to train). On the original implementation that was conducted for this experiment, which allowed a training maximum of 10,000 epochs, pseudorehearsal became catastrophically less effective after about 3-4 hidden layers of depth. Having dropped the base population to 5 items before rehearsal is performed, and upping the maximum number of epochs for the base population, pseudorehearsal begins to degrade at around 6-8 epochs for an MLP and 8-10 epochs with MLP with ReLU; at least for the random vector population trained on in this experiment. However, the population always manages to train for the base population fine, and the training method is the same, except that pseudorehearsal has a buffer. This leads to the conclusion that sweep pseudorehearsal is time-intensive to run on deeper networks.

Figure 1 shows that sweep and sweep pseudorehearsal perform the best – sweep pseudorehearsal even outperforming the other rehearsal strategies even without storage of previous learned items in memory. Therefore, this paper's research focuses mainly on sweep pseudorehearsal.

An interesting note sweep rehearsal is that it not only preserves the goodness of the base population as intervening items are learned, it actually improves the goodness of the base population (Figure 1 and [7]). This is probably because, as the learning buffer is 'refreshed' (refilled) after every epoch with another 3 randomly selected learned items, the exposure of the network to the base population is so great that a natural convergence to a better goodness continues. While sweep rehearsal seems like an ideal algorithm to use, it's not, because all of the previously learned items must be stored while new items are being learned. Pseudorehearsal does not require that any items previously learned be stored.

However, the fact that pseudorehearsal becomes less effective as layers are added doesn't tell us much about its use in modern deeper networks. Adding more densely-connected layers to a neural network exponentially increases the computational cost of training on it. A network of n nodes requires $k*n^2$ vertices for each layer k of size n nodes, each of whose values must be calculated for each prediction and back-propagation training run as a normal MLP. Modern DNNs have overcome this computational nightmare with optimisation techniques, and partially-connected CNN layers linked with pooling layers, for example.

6.2 Investigation 1: Exploring the effect of DNN techniques with pseudorehearsal

Adding extra layers to the network seems to decrease the effectiveness of pseudorehearsal slightly for each layer. This could be because the random pseu-

doinputs do a progressively worse job of preserving the network, or it could be that a network's solution to a task is less generalised as more layers are added. I'm sure that picking non-random pseudoinputs for this task would improve the performance of sweep pseudorehearsal. Results are shown in Figure **??**.

From Figure **??**, it seems that the refresh rate of sweep pseudorehearsal (how long a buffer is trained before being replaced with new pseudoitems) can change with minimal impact on performance. This is interesting, because the only difference between sweep pseudorehearsal and random pseudorehearsal is that in the former, the buffer is refreshed after every single epoch. This is probably inefficient – the buffer refresh rate could be higher with little performance loss on the base population. (N.B., the error criterion for halting learning on each intervening trial is the goodness of the intervening trial, not the goodness of the base population).

There are a lot of differences between the multilayer perceptrons, introduced with the advent of backpropogation in the 1960's/70's, and modern DNNs. Modern DNNs use a series of optimisation techniques to reduce overfitting, conduct feature extraction and speed computation on difficult tasks. Some examples of these optimisation techniques include the ReLU activation function for increasing speed in computation, the usage of 'dropout' (where a random subset of a network's units are excluded from training each time), and 'Adam' optimisation for credit assignment. Pseudorehearsal in its pure form is simply not very effective as network sizes increase [17], and this experiment's purpose is to explore the range between an MLP, where pseudorehearsal is very effective, and a DNN, where better inputs for pseudorehearsal must be generated for it to remain effective.

6.2.1 ReLU

The ReLU (REctified Linear Unit) is a non-linear activation function for neural networks which allows for quicker computation of neural units. They are commonly used in large modern multi-layer networks like CNNs as they increase efficiency. Results are given in Figure 4. With a simple MLP, training time was very slow as new layers were added, even for a the small base population given. ReLU layers greatly sped up training as new layers were added for not much cost on pseudorehearsal, so ReLU activation was used in conjunction with the other optimisation tools like Adam and Dropout for the later experiments. The ReLU activation function is less biologically plausible than the sigmoid activation function [32]. The other form of activation function for a artificial neuron is the hyperbolic tangent activation function, whose range is (1, 1) as opposed to the sigmoid's [0,1]. The activation function for ReLU units is $f(x) = max(0, x)$.

6.2.2 Dropout & ReLU

Dropout is a recent machine learning technique used to avoid overfitting of neural networks [33, 34]. It ensures that individual neurons in a network are not relied upon too heavily by leaving a subset out with some probability p

$\in \mathbb{R}$, $0 \leq p < 1$. p is the probability that an arbitrary neuron n in the network remains.

Results of rehearsal with dropout can be seen in Figure 5.

It seems logical that adding dropout to pseudorehearsal (as well as learning the base population) would increase the effectiveness on pseudorehearsal. However, this is not reflected in the results. Adding dropout doesn't seem to affect either learning with no rehearsal or sweep rehearsal, but there seems to be a slight decline in the effectiveness of pseudorehearsal which comes out of the standard deviation, as dropout's p is increased. This is strange – it could be because of the smaller base populations. Excluding units seems to degrade the knowledge of the base population faster as interventions are changed, possibly because the units that are trained on the interventions are more drastically affected when there are less of them. It could also be that training simply takes longer with dropout – the more training required on the interventions, the bigger the impact of the base population.

6.2.3 Adam

The Adam optimiser is a popular optimisation algorithm for ANNs which improves the performance of ANNs for large datasets using stochastic gradient descent [35]. It is hugely popular and works well across a range of problems, such as in classification and regression tasks. 'Adam' is short for 'adaptive moment estimation.' Adam comes from a family of optimisation techniques, such as RMSProp [36] (Root Mean Square propagation) and Adagrad.

The ReLU and MLP networks started to collapse in training over 4 layers for the ceiling given (70,000 epochs per intervention), but Adam optimisation allows the network to learn over up to 10 layers deep. Also, the fall in effeciveness in pseudorehearsal for Adam is far more modest than for dropout or just normal ReLU, even after 10 layers. This is interesting, and it would be interesting to explore this effect on different types of base populations.

Figure 7 gives a summary of the number of epochs required to train Adam on the base population. It is very sporadic, but for the single hidden layer the time taken to train is quite low, which might account for why it is able to function on higher layers effectively.

6.3 Investigation 2: Alternative Pseudorehearsal/Rehearsal Strategies

6.3.1 Changing buffer refresh rate

An efficient optimisation technique for sweep and pseudosweep rehearsal could be to adjust the amount of epochs that a buffer is trained for before a new buffer is generated. In the implementation in [7], the buffer refresh rate is fixed at 1. This means that the number of times a new buffer has to be generated can be reduced, and so a network can learn quicker – hopefully without losing any quality of learning of the base population. Results of the experimentation

for the buffer size are shown in Figure **??**. Also, the number of epochs required to train is shown in Figure 9.

6.4 Investigation 3: Pseudorehearsal in Classification Tasks

The original paper for pseudorehearsal conducted linear regression on a randomly generated dataset, as has been done here for most of the experiments. Other papers have covered the effectiveness of pseudorehearsal on classification tasks like MNist [17, 24], and it would be good to see some comparison between the random regression task and classification.

6.4.1 Iris

The Iris dataset used was from the UCI database of machine learning tasks, and is a simple classification task. The experiment for pseudorehearsal on the Iris dataset was conducted by choosing a base population of size 20 of 'Iris-Setosa' and having the intervening trials be from the most conflicting task. 'Iris-vericolor.'

Obviously, the input for the pseudoitems should be different as the range of inputs is no longer in the range [0,1]. Each feature has a different range. Therefore, the program searches through the training file before the experiment is conducted and finds the range of each feature (for example, the first feature is the sepal length of the flower, and it ranges from 4.3cm to 7.7cm). This is equivalent to storing the range of each feature – which is cheap to store for an arbitrary number of items to learn. For example, an algorithm could be:

```
generate a small initial range for each feature that is presumed e.g. [0,1]
every time a new item is presented for training:
for each feature in item:
    if the value at that item's feature value lies outside the range stored:
        increase the stored range for that feature to accomodate the new item
```

A value for each feature is picked uniformly from the range to generate a new pseudoitem input (and fed through the network to generate a corresponding output to form the pseudoitem). Generating sensible ranges for pseudoitems by analysing training items' feature ranges is a great optimisation trick which will logically improve pseudorehearsal accuracy by ensuring that the range of inputs are sensible for each feature. For the experiments on random binary input on most of the other items, pseudoitem inputs are just like in [7] – a randomly generated vector $\mathbf{2}^k$ for k inputs.

See Figure 10 for the results

What is interesting from the graph is that while catastrophic forgetting is predictably bad, pseudorehearsal doesn't perform particularly well either. However, the margin of error after each intervention was very high ($\approx$). This could be because the categorical accuracy varied between (1.0 (100% accurate) and 0.0 (0% accuracy)) a lot. The number of epochs that was required to train

this population was very low, as a loss function called 'categorical cross entropy' was used. This loss function, on this small amount of classes and items, very quickly disrupted the network in either a very small way or a very large way based on how the features interleaved for a particular data item.

6.5 Experiments on structured data

The analysis of random vectors through a network is able to generalise an arbitrary machine learning task quite well. However, in order to make generalisations about the effectiveness of pseudorehearsal, and other machine learning techniques like dropout and Adam optimisation, experiments should be conducted on structured data to inspect the behaviour of pseudorehearsal on more realistic tasks. The simple structured classification task that was chose was the Iris dataset (above), and a simple, toy, regression (i.e. 'function fitting' – real value function approximation) task was chosen as well – the Boston Housing dataset from the UCI Machine Learning repository [29]. There were a few changes to the experiments on the dataset compared with the other experiments conducted. A base population of 20 items were learned from the dataset, selected randomly for each experiment. The experiments were run 50 times, as the margin of error was really large when only run 20 times. Psuedorehearsal was conducted in the same way as for iris (i.e. real-value ranges were chosen instead of the classic [0,1,....1,0] binary vectors randomly generated.

Figure 11 shows a graph of an interesting experiment. What happened in this instance is that first, a base population of 20 items was learned from the Boston Housing dataset, and then interventions were chosen from the same dataset. The interventions chosen were chosen from the same pool, so they could technically be the same items, although the chance is quite low, as there were 506 items in the dataset. Another implementation of the same network [37] showed that a MAE of 5 was good for this dataset and network. Catastrophic forgetting was not as severe in this instance, because the intervention items that were trained shared the same features as the base population. After 10 interventions, the MAE was still only 5.

6.6 Investigation 4: Exploring non-random strategies to choose pseudo-inputs

7 Conclusion

The effectiveness of pseudorehearsal as depth is added to a network is very likely diminished in two regards: the first, because of its lesser ability to preserve information as layers are added; and the second, because training pseudorehearsal becomes expensive when layers are added. This result is predictable, as the efficiency of backpropagation learning is also diminished when layers are added, for any population. Pseudorehearsal has been observed to behave differently when different machine learning techniques are applied – for example, dropout

appears to worsen the effect of pseudorehearsal as it is applied more and more heavily, for the random population it was tested on. Conversely, pseudorehearsal works well with adam optimisation, as effectiveness is preserved well, and in fact, pseudorehearsal can be extended to more layers (up to 10 in our experiment) than an MLP can.

In this paper, psuedorehearsal has been applied to a variety of networks using modern optimisation techniques. The impact of naive layer depth on pseudorehearsal has been explored, and the idea of having a variable buffer refresh rate has been introduced. Pseudorehearsal has also been applied to a simple classification task (Iris). Additionally, a review has been conducted of how pseudorehearsal is being used today in its many forms: in reinforcement learning 'experience replay', as well as in image recognition applications.

8 Final notes

The source code for the project is currently on github. The more recent tensorflow implementation of pseudorehearsal can be found at `https://github.com/Robbie-Cook/Deep-Network`. Older versions are at: `https://github.com/Robbie-Cook/tensornetwork`, and the nimblenet implementation is located at `https://github.com/Robbie-Cook/nimblenet`. The Aims and Objectives document is attached.

References

[1] Michael McCloskey and Neal J Cohen. "Catastrophic interference in connectionist networks: The sequential learning problem". In: *Psychology of learning and motivation*. Vol. 24. Elsevier, 1989, pp. 109–165.

[2] Robert M French. "Semi-distributed representations and catastrophic forgetting in connectionist networks". In: *Connection Science* 4.3-4 (1992), pp. 365–377.

[3] Michael E Hasselmo. "Avoiding catastrophic forgetting". In: *Trends in cognitive sciences* 21.6 (2017), pp. 407–408.

[4] James Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the national academy of sciences* (2017), p. 201611835.

[5] Nan-Ying Liang et al. "A fast and accurate online sequential learning algorithm for feedforward networks". In: *IEEE Transactions on neural networks* 17.6 (2006), pp. 1411–1423.

[6] Roger Ratcliff. "Connectionist models of recognition memory: constraints imposed by learning and forgetting functions." In: *Psychological review* 97.2 (1990), p. 285.

[7] Anthony Robins. "Catastrophic forgetting, rehearsal and pseudorehearsal". In: *Connection Science* 7.2 (1995), pp. 123–146.

[8] Anthony Robins. "Consolidation in neural networks and in the sleeping brain". In: *Connection Science* 8.2 (1996), pp. 259–276.

[9] Philippe Peigneux et al. "Sleeping brain, learning brain. The role of sleep for memory systems". In: *Neuroreport* 12.18 (2001), A111–A124.

[10] Carlyle Smith. "REM sleep and learning: some recent findings". In: *The functions of dreaming* (1993), pp. 341–361.

[11] Long-Ji Lin. *Reinforcement learning for robots using neural networks*. Tech. rep. Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

[12] Vladimir Marochko, Leonard Johard, and Manuel Mazzara. "Pseudorehearsal in value function approximation". In: *KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications*. Springer. 2017, pp. 178–189.

[13] Marochko Vladimir, Leonard Johard, and Manuel Mazzara. "Pseudorehearsal in actor-critic agents". In: *arXiv preprint arXiv:1704.04912* (2017).

[14] Zhiyuan Xu et al. "Experience-driven networking: A deep reinforcement learning based approach". In: *arXiv preprint arXiv:1801.05757* (2018).

[15] Vijay R Konda and John N Tsitsiklis. "Actor-critic algorithms". In: *Advances in neural information processing systems*. 2000, pp. 1008–1014.

[16] Richard Stuart Sutton. "Temporal credit assignment in reinforcement learning". In: (1984).

[17] Craig Atkinson et al. "Pseudo-Recursal: Solving the Catastrophic Forgetting Problem in Deep Neural Networks". In: *arXiv preprint arXiv:1802.03875* (2018).

[18] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[19] Corinna Cortes Yann LeCun. *MNIST*. URL: `http://yann.lecun.com/exdb/mnist/`.

[20] Alex Krizhevsky. *CFAR*. URL: `https://www.cs.toronto.edu/~kriz/cifar.html`.

[21] *SVHN*. URL: `http://ufldl.stanford.edu/housenumbers/`.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[23] *ILSVRC2012*. URL: `http://www.image-net.org/challenges/LSVRC/2012/`.

[24] Diego Mellado et al. "Pseudorehearsal Approach for Incremental Learning of Deep Convolutional Neural Networks". In: *Latin American Workshop on Computational Neuroscience*. Springer. 2017, pp. 118–126.

[25] Karol Gregor et al. "Draw: A recurrent neural network for image generation". In: *arXiv preprint arXiv:1502.04623* (2015).

[26] *TensorFlow*. URL: https://www.tensorflow.org/.

[27] Jørgen Grimnes. *Python Neural Network*. URL: http://jorgenkg.github.io/python-neural-network/.

[28] *UCI Machine Learning Repository*. URL: https://archive.ics.uci.edu/ml/datasets.html?format=&task=reg&att=&area=&numAtt=&numIns=&type=&sort=nameUp&view=table.

[29] *Boston Housing Dataset*. URL: https://archive.ics.uci.edu/ml/machine-learning-databases/housing/.

[30] *Iris Data Set*. URL: https://archive.ics.uci.edu/ml/datasets/iris.

[31] *Airfoil Self-Noise Data Set*. URL: https://archive.ics.uci.edu/ml/datasets/airfoil+self-noise.

[32] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, pp. 315–323.

[33] Geoffrey E Hinton et al. "Improving neural networks by preventing co-adaptation of feature detectors". In: *arXiv preprint arXiv:1207.0580* (2012).

[34] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[35] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[36] Tijmen Tieleman and Geoffrey Hinton. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude". In: *COURSERA: Neural networks for machine learning* 4.2 (2012), pp. 26–31.

[37] *Predict house prices: regression*. URL: https://www.tensorflow.org/tutorials/keras/basic_regression.

[38] Anthony Robins. "Catastrophic forgetting in neural networks: the role of rehearsal mechanisms". In: *Artificial Neural Networks and Expert Systems, 1993. Proceedings., First New Zealand International Two-Stream Conference on*. IEEE. 1993, pp. 65–68.