Analyzing Plasticity Through Utility Scores

Comparing Continual Learning Algorithms via Utility Score Distributions

Aldas Lenkšas[1]

Supervisors: Wendelin Böhmer[1]**, Laurens Engwegen**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 21, 2025

Name of the student: Aldas Lenkšas
Final project course: CSE3000 Research Project
Thesis committee: Wendelin Böhmer, Laurens Engwegen, Megha Khosla

Abstract

One of the central problems in continual learning is the loss of plasticity, which is the model's inability to learn new tasks. Several approaches have been previously proposed, such as Continual Backpropagation (CBP). This algorithm uses utility scores, which represent how useful the individual neurons are for computing the answer. We have analysed such utility score distributions for different algorithms: backpropagation, L2 regularization, Shrink and Perturb, CBP, and its variants with L2 regularization and Shrink and Perturb. Our results reveal that well-performing algorithms maintain better-balanced utility score distributions and fewer neurons with scores near zero, indicating higher plasticity. In particular, CBP and its variants achieve better accuracy by actively redistributing utility and reinitializing underused neurons. These findings suggest that utility scores are a valuable analysis tool for understanding and improving continual learning systems.

The code and experiments are available in the GitHub repository[1].

1 Introduction

A common approach to deep learning (and machine learning in general) in research and industry is dividing the process into two steps. Firstly, the model is trained using the training data. Afterwards, the model is applied to provide predictions in a product or research. The values of the learnable parameters, such as the weights of the deep neural network, are only changed during the first of these steps, whereas in the second step, they are kept constant. However, in practice, there are scenarios where new data keeps arriving and the model must be updated. The example is dealing with financial data or any other situation where machine learning is employed in adaptive environments [1]. The field of research that studies this is called *continual learning*.

One of the key problems addressed in continual learning is *loss of plasticity* [2]. Plasticity is the model's ability to learn new tasks with a level of accuracy comparable to that achieved on previously learned tasks. It is common for a model to lose performance when learning new tasks, indicating loss of plasticity. Ash and Adams [1] have shown that a model trained on a previous task has a worse generalization performance on a new task, compared to the model without previous training. A similar result was presented by Dohare et al. [2], where it was illustrated that in the Slowly-Changing Regression problem (a non-stationary task) for a model which uses backpropagation with Stochastic Gradient Descent (SGD) the online training error increases, regardless what activation function is used.

There are some proposed hypotheses for the loss of plasticity. A paper by Lyle et al. [3] has shown that plasticity is connected to saturated units and the curvature of the loss landscape. Additionally, networks with a low percentage of dead neurons (neurons whose output is always zero) and low network average weight magnitude were demonstrated to perform better in continual learning settings [2]. Nevertheless, even with these hypotheses, the precise causes of loss of plasticity remain unclear.

To tackle loss of plasticity, some of the well-known methods were tested and several new approaches have been proposed previously. The simplest one is to use *warm-starting*, in which the model is further trained on new data (similar to using the pre-trained model for a new task). However, it was shown that this does not perform well due to the signs of the loss of plasticity, such as decreasing accuracy, increasing network's weight magnitude, and the ratio of dead neurons [1, 2]. There are state-of-the-art methods which were created to solve other machine learning problems: L2 regularization [4] , Dropout [5], Adam [6], Online Normalization [7]. These approaches can be used to mitigate loss of plasticity, presumably due to the fact that these methods potentially reduce the number of dead neurons or the weight magnitude. More recently, an algorithm called *Shrink and Perturb* was introduced, in which the learnable parameters are periodically made smaller and then random parameter noise from the normal distribution is added. This results in a more plastic network likely due to the constant shrinking of the weights [1] and perturbing the units to keep them away from the extreme values [3].

The *Continual Backpropagation* algorithm was also proposed as a solution to the loss of plasticity problem [2]. Its novelty is keeping track of the *utility scores*, which are meant to give a score on how much each unit in the neural network contributes to the calculation. Such scores show how useful the neurons are in the computation of the output. After applying backpropagation, the algorithm reinitializes a small fraction of neurons that have low utility scores, which makes them useful in the training again and reduces the number of dead neurons.

The previous research has not examined the utility scores, such as the ones from the Continual Backpropagation, and their connection to the effectiveness of the algorithms or as an explanation for the loss of plasticity. This research aims to fill this gap by answering the following question: *What are the effective utility score distributions for continual learning?* We hypothesize that the well-performing algorithm should have little of the neurons whose utility score is (close to) zero since such neurons are not contributing to the calculation. Furthermore, the utility scores should avoid growing large, as that might mean that the weights (which are included in the expression for the utility score) are increasing, indicating the potential loss of plasticity. Finally, in an efficient algorithm, all neurons should exhibit similar utility scores, implying that each neuron contributes equally to the computation of the output.

Our main contribution is the comparison of several algorithms using different metrics with an emphasis on utility scores. The algorithms that we have compared are: standard backpropagation (BP), L2 regularization (L2), Shrink and Perturb (SnP), Continual Backpropagation (CBP), as well

[1] https://github.com/Aldas25/loss-of-plasticity

as the variations of Continual Backpropagation with L2 regularization (CBP+L2) and Shrink and Perturb (CBP+SnP), all of which used SGD during training. We have shown that the utility score distribution, and its change, are important metrics for the algorithm performance because the algorithms that perform well have a more even distribution compared to the worse-performing ones.

The paper is structured in the following way. Section 2 presents background and related work for our research. The methodology of our experiments is presented in Section 3. Section 4 contains the details of the experimental setup and the results obtained. The reflection on the responsible research is given in Section 5. The discussion of the results can be found in Section 6, and the conclusions are presented in Section 7.

2 Background and Related Work

In this section, we present the background and previous work related to our research. Sections 2.1-2.4 explain the various techniques to mitigate loss of plasticity, whereas Section 2.5 discusses its different reasons analyzed in the past research.

2.1 Application of the Standard Techniques to the Continual Learning Setting

Several techniques that are widely applied in various deep learning settings were previously evaluated for plasticity. The simplest method one could do is to straightly apply the backpropagation algorithm (also referred to as *warm-starting* [1]). This works similarly to pre-training: the model is trained on the first task, and then such a trained model is later used to train on the subsequent task. Nonetheless, there are other algorithms as well. The first of them is *L2 regularization* (sometimes also referred to as ℓ_2 *regularization*, or *weight decay*) which prevents the learnable parameters obtaining large magnitude by penalizing them with the additional term of $\lambda\|w\|_2^2$ to the loss function, where λ is the penalty constant and w are the weights (or learnable parameters) [4]. Next, *Dropout* was introduced to address the overfitting by dropping some units during training and thus creating a thinned net [5]. The third one is *Adam*. It is an optimizer that computes adaptive learning rates based on the estimates of the first and second moments of the gradients [6]. Lastly, there is *Online Normalization* [7]. It gets inspiration from *Batch Normalization* which normalizes the inputs of each layer [8]. Differently from Batch Normalization, Online Normalization does not need to rely on large mini-batch sizes and works even when training is done without batching since the algorithm calculates exponential moving mean and variance instead. Even if these algorithms were not introduced for continual learning, they have the potential to remedy the problem because of the induced side effects, such as low weight magnitudes (for example, in the case of L2 regularization).

These five approaches (backpropagation, L2 regularization, Dropout, Adam, Online Normalization) were tested in the paper by Dohare et al. [2]. Using the Slowly-Changing Regression problem (see Section 4.1), they have shown that backpropagation suffers from loss of plasticity, and different learning rates or commonly used activation functions do not remedy this problem. In addition, along with some other novel approaches, the five mentioned algorithms were tested in the same paper using the Online Permuted MNIST problem (see Section 4.2) [2]. All of their online classification accuracy was decreasing, and the percentage of dead units was increasing. Overall, the results in the paper show that these five standard approaches perform poorly in the continual learning setting, as compared to Continual Backpropagation (or even Shrink and Perturb).

2.2 Shrink and Perturb

Shrink and Perturb was introduced as a way to close the generalization gap that occurs during warm-starting. The paper by Ash and Adams [1] discusses that even if warm-starting is similar to the idea of pre-training, it "damages generalization to warm-start in data-rich situations". The idea of Shrink and Perturb is very simple: periodically shrink the learnable parameters by a constant and add a small amount of parameter noise from the normal distribution. The paper does not discuss in great detail the impact of the frequency by which the SnP is applied, but it mentions that SnP is very similar to using noisy L2 regularization (see Appendix A). Despite this similarity, it performs better than L2 regularization in the Online Permuted MNIST problem. What is more, in the same problem setting it achieves better results than the algorithms discussed in Section 2.1, but is still outperformed by Continual Backpropagation [2].

2.3 Continual Backpropagation

Continual Backpropagation was introduced by Dohare et al. [2] as an improvement to a backpropagation algorithm to enhance the neural network's plasticity. The algorithm keeps track of the utility scores (the formula given in Section 3.1), and the maturity age of the neurons which indicates how many iterations have passed since the last re-initialization of the unit. The algorithm occasionally (based on the value of the replacement rate) reinitializes the unit with the lowest utility score among the neurons whose maturity is above the threshold. When doing this, the incoming weights are reinitialized from the original distribution, whereas the outgoing weights are reinitialized to zero. This way, the neuron can start learning again, and it may become more useful for the computation. In the paper, it was shown that this algorithm outperforms the backpropagation, Shrink and Perturb, and other algorithms for the Online Permuted MNIST problem, and works well for different problems in deep learning and reinforcement learning.

2.4 Other Approaches to Remedy Loss of Plasticity

There are a lot more algorithms proposed that tackle the loss of plasticity problem which were not analyzed in this paper. One of them is *ReDo*, or *Recycling Dormant Neurons* [9]. The paper in which it was introduced addresses *the dormant neuron phenomenon* in reinforcement learning settings, which are related to continual learning due to their non-stationarity. It was observed that once a neuron becomes

dormant (showing little activity), it remains dormant for the rest of training and that such neurons hinder the ability to learn new tasks. The proposed algorithm deals with this by reinitializing them. It works similarly to Continual Backpropagation: *ReDo* periodically reinitializes neurons whose dormancy scores are below the given threshold. When combined with algorithms, such as DQN [10] and DrQ(ε) [11], the algorithm makes the network keep a low percentage of the dormant neurons during the training, and noticeably increase the performance of the algorithm.

Another novel approach is *Hare and Tortoise Networks* [12]. This algorithm is inspired by the hypothesis of how the hippocampus and the neocortex, two complementary learning systems, work in the human brain. Hare network adjusts its learnable parameters using a gradient-based optimization algorithm, which lets the network adapt to a new task quickly. The Tortoise network keeps the exponential moving average of the Hare network's parameters. The Hare network then periodically resets its values by copying them from the Tortoise network. Lee et al. [12] have shown that their method maintains both the network's plasticity and ability to generalize in different problem settings.

There are even more approaches discussed in the past literature [13], such as UPGD (applying less gradient descent change to more useful neurons, but using an alternative form of the utility score) [14].

2.5 Possible Explanations for the Loss of Plasticity

Various hypotheses for the reasons for loss of plasticity have been proposed before in the paper by Lyle et al. [3]. One of them deals with the saturated units. If the neuron activation function is stuck at the extreme value, it is mostly unresponsive to new inputs due to vanishing gradients. Saturated units were demonstrated to lead the model to failure to learn and, thus, plasticity loss. However, such saturated units are not enough to explain the reasons for plasticity loss, since it appears even in the absence of them. Loss of plasticity was observed to be deeply connected to the geometry of the loss landscape (which is quantified using the Hessian matrix). Lyle et al. [3] highlighted that models trained on reinforcement learning tasks were more tedious to train on new subsequent tasks, indicating a loss of plasticity associated with changes in the loss landscape's curvature.

Some of the other explanations have been discussed in the paper by Dohare et al. [2]. Similarly to the saturated units, a crucial insight is that the network's ability to learn new tasks is connected to the ratio of dead neurons (which also agrees with the *dormant neuron phenomemon* [9]). A drop of the effective rank and the increasing average weight magnitudes were phenomena occurring to the networks that have shown loss of plasticity. Ultimately, there might be a parallel with the lottery ticket hypothesis [2, 15]. The hypothesis states that randomly initialised feed-forward networks contain subnetworks (that are called *winning tickets*) which can reach a performance similar to the one of the original network. Once the network is trained, the randomness is decreased, leading to a fewer number of possible *winning tickets*. This hinders the ability to learn a new task.

3 Methodology

To gain insights about the utility score distributions, we have evaluated algorithms in non-stationary, continual learning problems also used in the work by Dohare et al. [2], but slightly changed due to computing resources constraints (explicit description of modifications is stated in Section 4). The algorithms chosen to be evaluated were standard backpropagation, L2 regularization, Shrink and Perturb (implemented as L2 with perturbation step, see Appendix A), Continual Backpropagation, Continual Backpropagation combined with L2 regularization, and Continual Backpropagation combined with Shrink and Perturb. All algorithms used SGD for training, and had ReLU as the activation functions, as this was the case in the work by Dohare et al. [2].

To understand how well a model can train in the continual learning setting, the online training accuracy (or, depending on the problem, the online training error) was taken as the main metric. The average weight magnitude and the ratio of dead neurons were taken into consideration as well, as they are deeply connected to the plasticity and were crucial for the analysis of the utility scores. The ratio of dead neurons is calculated at the end of each task by rerunning the first 2 000 inputs and calculating which activations of the neurons produce output equal to zero. If for all given inputs the output was zero, the neuron is considered a dead neuron.

When running the experiments, the hyperparameter search was performed as a first step in order to find the best hyperparameters for the modified problem settings and examine if the results from the previous papers could still be reproduced. In this first experiment iteration, the experiments were repeated a smaller number of times (with different random seeds). With the best-performing hyperparameters, the experiment was rerun with more random seeds to obtain more stable and accurate results.

The experiments were carried out using the computational resources of the DelftBlue supercomputer, provided by the Delft High Performance Computing Centre [16], and the Delft AI Cluster (DAIC) at TU Delft [17].

3.1 Utility Scores

The neuron utility is calculated the same way as in the paper by Dohare et al. [2]. The formula used is as follows: "the contribution utility, $\mathbf{u}_l[i]$, of the ith hidden unit in layer l at time t is updated as

$$\mathbf{u}_l[i] = \eta \times \mathbf{u}_l[i] + (1-\eta) \times |\mathbf{h}_{l,i,t}| \times \sum_{k=1}^{n_{l+1}} |\mathbf{w}_{l,i,k,t}|,$$

in which $\mathbf{h}_{l,i,t}$ is the output of the ith hidden unit in layer l at time t, $\mathbf{w}_{l,i,k,t}$ is the weight connecting the ith unit in layer l to the kth unit in layer $l+1$ at time t and n_{l+1} is the number of units in layer $l+1$", and η is the decay rate [2]. By multiplying the magnitudes of the neuron output and the sum of the outgoing weights, this formula gives a reliable approximation of the neuron's contribution to the computation of the output. In addition, a moving average of the score is maintained, allowing reinitialization to consider trends over time instead of relying solely on the most recent value.

Analyzing utility scores can be a good measure to see how well the algorithm is performing. Thus, we have calculated and analyzed distributions of such scores at different training iterations. The distribution is the binned histogram with the utility score values along x-axis, and their frequencies along y-axis. Such a plot represents the distributions of the utility scores of the neurons in all the hidden layers of the network at a given iteration and is averaged over multiple runs. For the networks with multiple layers, the scores have been normalized layer-wise (per layer of the network) to the range $[0; 1]$ (by dividing by the maximum value) before computing the histograms, in order to remove the influence of different magnitudes of scores between different layers while preserving the notion of zero value (which corresponds to a useless, possibly dead, neuron, and is important to the analysis).

Two artificial examples of such scores are present in Figure 1. The left one can be interpreted as a distribution of poorly performing algorithms since all units have very low utility, with most units having it close to zero. On the contrary, the distribution on the right can be interpreted as a distribution of a well-performing algorithm due to a more even distribution of utility: a lot of units share the utility while few of them have close to zero utility and only a few have higher utility than the rest. Such a histogram would indicate that the network shares its computation among the units, and could be considered as an ideal histogram.

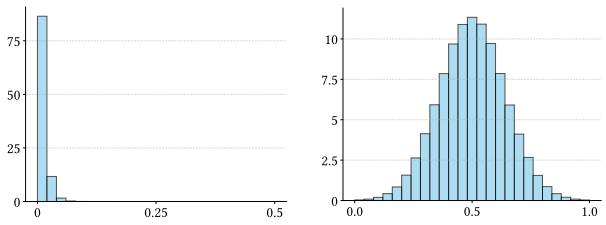

Figure 1: Artificial examples of possible utility score distributions. x-axis denotes the utility scores, and y-axis denotes the binned frequency of the utility scores. Left: distribution of non-normalized utility scores of a one-layer network that supposedly performs poorly. Right: distribution of normalized utility scores of a multi-layer network that likely performs well.

For the further analysis of utility scores, the utility score average, and its change during training, was also considered as it allowed us to easily compare the scores of different algorithms. Moreover, we have analyzed the individual neurons' utility score progression. For the networks with a small number of neurons, this meant that we plotted the maximum utility score, along with the second maximum, and so on. For the bigger networks, the neurons were grouped, and the plot had the utility scores of the 50 biggest utility scores, then the next 50, and so on. Such plots are useful to analyze how the utility scores change during the training.

4 Experimental Setup and Results

We have investigated the utility scores for two problems: Slowly-Changing Regression problem (Section 4.1), and Online Permuted MNIST problem (Section 4.2).

4.1 Slowly-Changing Regression

Slowly-Changing Regression problem (or, the *Bit-Flipping problem* [18]) is a computationally-inexpensive problem used in the work by Dohare et al. [2], which makes a good playground to analyse plasticity. It is a regression problem, using a squared loss, where the learner network receives binary inputs consisting of $m + 1$ bits. The first f bits are the slowly-changing part, then $m - f$ bits that are randomly generated and the last bit is a bias term with a constant one. The first f bits change only every T sample. The original problem only flips one bit, whereas in our modification we randomly reinitialize the first f bits. This choice was made to force the network switch to a completely new task instead of having a somewhat continuous change of non-stationarity. Such a decision led us to a better reproduction of the paper results and made the analysis more trustworthy. For more analysis refer to Appendix B.

The outputs were generated using the target network. This network represents a more complex target function since it has 100 hidden units, whereas the learner network has only 5 such neurons (both networks have one hidden layer). The weights of the target network were randomly chosen to be between -1 and +1. The activation was Linear Threshold Unit with the threshold $\theta_i = (m+1)\times\beta - S_i$, where S_i is the number of input weights with negative value [2, 19].

The chosen values correspond to the experimental setup by Dohare et al. [2]: $m = 20$, $f = 15$, $\beta = 0.7$, $T = 10\,000$. The experiment was run with 3 million data points. For the hyperparameter search, for each algorithm and each choice of hyperparameters, we performed 5 independent runs. Afterwards, for the full analysis with chosen hyperparameters, we performed 100 independent runs for each algorithm.

The weights in the learner network were initialized using the Kaiming distribution [2, 20]. Hyperparameters were mostly chosen the same as in the paper by Dohare et al. [2]. However, since the experimental setup is slightly different, for each algorithm we have experimented with multiple values, which can be seen in the Appendix C.

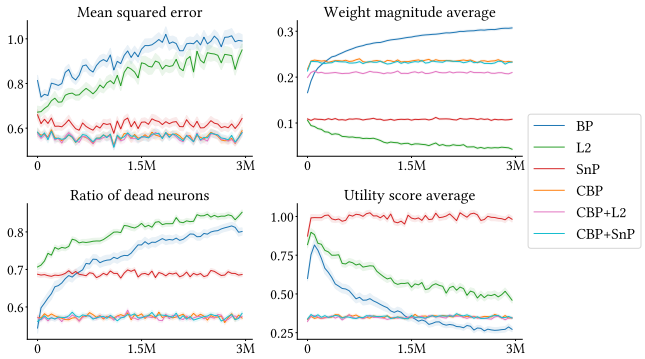

Figure 2: Results of the Slowly-Changing Regression problem for backpropagation (BP), L2 regularization (L2), Shrink and Perturb (SnP), Continual Backpropagation (CBP), CBP variation with L2 (CBP+L2), and CBP variation with SnP (CBP+SnP). The results are averaged over 100 runs. The solid lines represent the mean and the shaded regions correspond to ±1 standard error.

Figure 2 shows the four metrics during the training for all six algorithms. Backpropagation has a visible loss of plasticity due to its increasing error metric, as well as bad performance with weight magnitude and the ratio of dead neurons. L2 performs slightly better, and has the lowest, and constantly decreasing, weight magnitude. SnP shows good plasticity since it is able to keep stable all three metrics. However, CBP outperforms SnP since it has a lower error and the ratio of dead neurons, even though SnP has a smaller weight magnitude. The variants of CBP (CBP+L2 and CBP+SnP) perform very close to CBP without demonstrating any significant differences. From the utility score averages, it can be seen that SnP and CBP with their variants can contain similar utility scores during training, whereas BP and L2 show a significant drop, with BP obtaining even lower scores than CBP at the end of training.

Figure 3 illustrates the utility score histograms of the last iteration of training for all six algorithms. Here it is evident that all algorithms suffer from units with very low utility scores, where it is most visible for BP, L2, and relatively less for SnP. CBP and its variants' histograms are very similar, and they show that CBP can have more neurons used in the computation, and those neurons have utility scores mostly between 0 and 1, whereas, for example, SnP has neurons that have utility scores close to 3. This suggests that the share of the computation for the units in CBP is more even than in BP, L2 or SnP.

Figure 4 supports this claim by presenting the utility score evolution of neurons. BP has a higher maximum, but its neurons lose utility over time, with most of it concentrated at only one unit at the end. Contrastingly, CBP can keep the scores similar. While it still has one unit with substantially higher utility, and one unit with a score close to zero, the algorithm can keep the utility distributed among more units than BP. Similar results are presented in Appendix D.

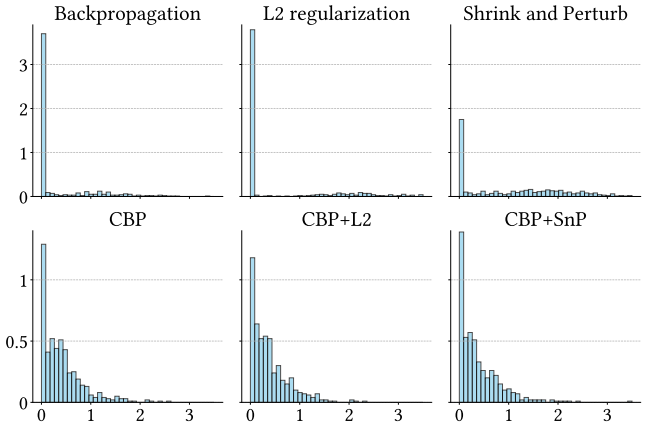

Figure 3: Binned utility score histograms for different algorithms in the last iteration of the training in the Slowly-Changing Regression problem. x-axis denotes the utility scores, and y-axis denotes the binned frequency of the utility scores. The histograms are averaged over 100 runs. *CBP* stands for Continual Backpropagation, *CBP+L2* is CBP variation with L2 regularization, and *CBP+SnP* is CBP variation with Shrink and Perturb.

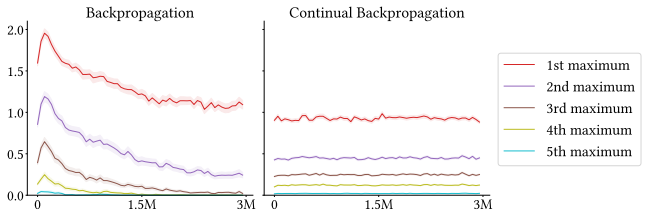

Figure 4: Utility scores of the individual units in the Slowly-Changing Regression problem for backpropagation and Continual Backpropagation. *1st maximum* shows the progression of the maximum utility score, *2nd maximum* shows one of the 2nd maximum, and so on. The results are averaged over 100 runs. The solid lines represent the mean and the shaded regions correspond to ±1 standard error.

4.2 Online Permuted MNIST

Online Permuted MNIST is a modification introduced by Dohare et al. [2] to the classical MNIST problem [21]. We have used the same experimental setting but made some constants smaller to reduce the computational complexity. The dataset used consists of 10 000 grayscale (pixel values normalized to the range $[0; 1]$) images of size 28×28 of handwritten digits. These images are randomly selected samples from the original dataset consisting of 60 000 images. To make inputs for the problem, 800 tasks have been created, each with a random permutation of pixels applied to all of the 10 000 images. The feed-forward neural network consisting of three hidden layers is then trained with these 800 tasks without the indication when the task switches. In the work by Dohare et al. [2], they used 2 000 units per layer, but we have decreased it to 100 neurons due to computational constraints and to make the network complexity shrink similarly as we have shrunk the complexity of the input data.

As in the Slowly-Changing Regression problem, the weights for the network were initialized using the Kaiming distribution, and the hyperparameters were mostly chosen

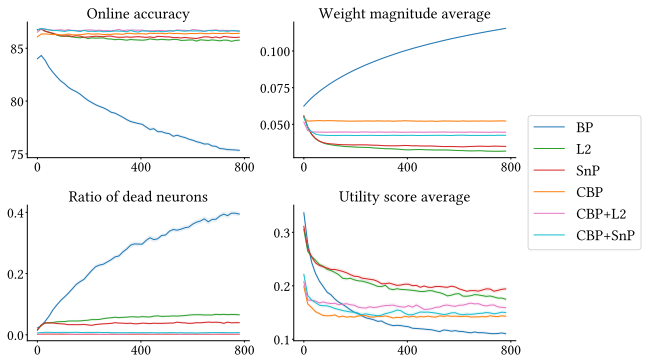

Figure 5: Results of Online Permuted MNIST problem for the backpropagation (BP), L2 regularization (L2), Shrink and Perturb (SnP), Continual Backpropagation (CBP), CBP variation with L2 (CBP+L2), and CBP variation with SnP (CBP+SnP). Utility scores are normalized layer-wise to the range $[0; 1]$. The results are averaged over 30 runs. The solid lines represent the mean and the shaded regions correspond to ±1 standard error.

the same as in the original paper, but with a hyperparameter search to account for the slightly changed experimental setup. For this, the experiment was run 5 times (with different random seeds), and later with the best found values it was repeated with 30 runs. The hyperparameter values can be found in the Appendix C.

Figure 5 presents the results for the Online Permuted MNIST problem. Differently from the Slowly-Changing Regression problem, L2 performs relatively well: it maintains an accuracy of almost 86%, whereas the accuracy of CBP variants results in only a little bit above 86%. Moreover, in this problem set, CBP variants show a better performance than CBP, both in accuracy metric and in lower weight magnitude, whereas the ratio of dead neurons is close to zero for both CBP and its variants. On the contrary, the utility score averages are still similar for CBP and its variants. In addition, BP experiences a significant drop in this metric, whereas L2 and SnP have a slow decline during training, with L2 having a slightly lower utility score average.

The histograms of the normalized utility scores for the last iteration of training are presented in Figure 6. Most units in BP have utility close to zero, while L2 and SnP have less of such units and slightly more units of higher utility. CBP shows even better distribution, and its variants, in this problem setting, can flatten it out even more: CBP variants have a lower number of units with zero utility score and more units with higher utility.

Similarly to Figure 4, the utility score maximums from Figure 7 establish that CBP can distribute the utility among units better than BP where a large number of units experience a significant drop of utility with only the top 50 (out of 300) units having high utility. These claims also hold for other algorithms that outperform BP, as shown in Appendix D.

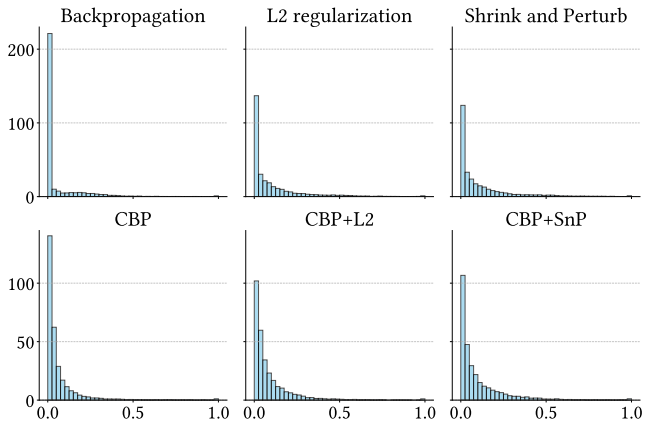

Figure 6: Binned utility score histograms for different algorithms in the last iteration of the training in the Online Permuted MNIST problem. x-axis denotes the utility scores, and y-axis denotes the binned frequency of the utility scores. The utility scores are normalized layer-wise to the range $[0; 1]$. The histograms are averaged over 30 runs. *CBP* stands for Continual Backpropagation, *CBP+L2* is CBP variation with L2 regularization, and *CBP+SnP* is CBP variation with Shrink and Perturb.

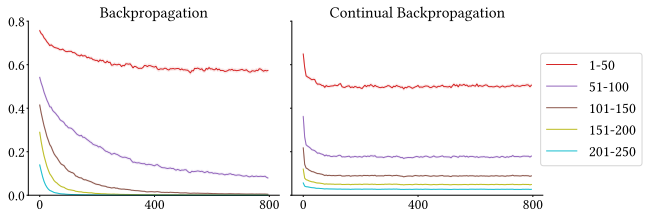

Figure 7: Utility scores of the groups of units in the Online Permuted MNIST problem for backpropagation and Continual Backpropagation. Utility scores are normalized layer-wise to the range $[0; 1]$. The line *1-50* shows the progression of the average of the highest 50 utility scores, *51-100* shows one of the next 50 highest scores, and so on. The results are averaged over 30 runs. The solid lines represent the mean and the shaded regions correspond to ±1 standard error.

5 Responsible Research

In accordance with the TU Delft Code of Conduct [22], we carefully considered responsible research aspects, such as reproducibility, transparency and ethical integrity. All of the code files are saved, along with the development process visible in the commit history, in the GitHub repository[2]. We have maintained the code to be minimal and well-documented, leaving most of the original authors' settings the same, while all deviations have been reported. The hyperparameter values that we have experimented with have been clearly explained in the paper. The MNIST dataset [21] is publicly available, and all the data we have used can be generated the same way using the scripts provided in the codebase.

The algorithms discussed in this research paper are important to various research and industry scenarios, such as reinforcement learning and robotics, some of which have broad implications and can be directly or indirectly dangerous to humans, animals, or the environment. However, this paper's research on its own does not have any direct impact, and no critical ethical issues are present, especially because no personal data was used during the process. We presented all results, including those with weaker significance (moved to the appendices), to avoid selective reporting or any associated bias. All deviations from original experimental setups are clearly stated and justified in the main text, as well as indicated in the code.

To ensure result robustness, the experiments were repeated a significant number of times (100 runs for the Slowly-Changing Regression problem, and 30 runs for the Online Permuted MNIST problem). Random seeds (for the used libraries: *numpy*, *torch*, and *torch.cuda*) were preselected and stored in configuration files, enabling consistent and repeatable results.

5.1 Statement on the Use of Generative AI and Large Language Models

Generative AI tools and Large Language Models were occasionally used for grammar and syntax checking (Grammarly), single-line code completion during debugging

[2]https://github.com/Aldas25/loss-of-plasticity

(Github Copilot), and clarification of programming errors, syntax or example library usage (ChatGPT, Claude). The prompts that were asked to chatbots are presented in Appendix E. All outputs from such tools were critically evaluated and double-checked to avoid factual errors or hallucinations.

6 Discussion

Our experiments reveal patterns that not only replicate but also extend previous findings from Ash and Adams [1], Dohare et al. [2], and Lyle et al. [3]. The results confirm the previous research which has stated that backpropagation consistently shows the poorest performance across both tasks, and the reason for this is likely the increase of the ratio of dead neurons and weight magnitudes. The high concentration of utility in a few neurons (Figures 3 and 6) additionally suggests that there is an inefficient allocation of computation with many neurons becoming useless over time. Similarly to what was found in the past, L2 regularization, although successful in constraining weight magnitudes, failed to mitigate the accumulation of dead neurons and demonstrated lower plasticity than Shrink and Perturb or Continual Backpropagation. This suggests that simply keeping weight magnitudes small is insufficient to preserve the model's adaptability in non-stationary environments.

In contrast, both Shrink and Perturb and Continual Backpropagation performed relatively well. SnP's perturbation step and CBP's reinitialization appear to be critical for maintaining the plasticity of the network. This can be a result of reinitializing saturated units [3] which are unresponsive to new inputs, as both SnP and CBP possibly help the neurons move away from the extreme values they are stuck at. Moreover, our results show that CBP's neuron reinitialization lets the network redistribute the computation more effectively. This is clear in both the utility score histograms (Figures 3 and 6) and the stable level of the individual neurons and neuron groups utilities (Figure 4 and 7).

CBP variants (with either L2 regularization or Shrink and Perturb) do not offer significant improvements over CBP alone. This can mean that the utility-based reinitialization is already highly effective, and additional modifications do not substantially improve the performance in the experimental settings we have tried.

Most importantly, these findings highlight that the current hypotheses, such as dead neurons or weight magnitude growth, do not fully explain the phenomenon of loss of plasticity. More comprehensive understanding requires also analysing how the calculation of the output is shared across the neural network. Because of that, utility scores and their distributions provide a new angle for the explanation of the performance and are important measures for the algorithms that deal with continual learning problems.

7 Conclusions and Future Work

This research has dealt with the utility score distributions in two different problems (Slowly-Changing Regression problem and Online Permuted MNIST), using different algorithms: backpropagation, L2 regularization, Shrink and Perturb, Continual Backpropagation, and the combinations of Continual Backpropagations with L2 regularization and Shrink and Perturb. The differences in the utility scores and their progression were investigated, and the possible reasons were discussed, while also linking them to the algorithm effectiveness, which was measured using the error, or accuracy, metrics, as well as the ratio of dead neurons and weight magnitudes.

We have hypothesized that ideally, the utility score distribution should have a low number of neurons whose score is close to zero, as they are almost useless in the computation, and that the neurons should all have similar scores, which would mean that they all are equally important to the calculations.

The results have shown that the algorithms that perform well contain a smaller number of neurons that have very small scores, and their utility scores are better distributed. That means that even if most likely the neurons are not equally important to the computation, they still have meaningful importance to the calculations. On the contrary, the computation of the other algorithms, such as backpropagation, have their calculations concentrated only in a small number of units, which helps to explain the network's loss of plasticity. These results also indicate that the utility score distributions are a valuable tool for the analysis of the algorithm plasticity.

Future work. We understand that these two problem settings that we have used are not enough to fully explain the loss of plasticity and the link between the utility scores and performance. The future work should try more problems (for example, class-incremental CIFAR-100 or ant-locomotion problem [2]), and experiment with more algorithms (such as *ReDo* [9], *Hare and Tortoise Networks* [12], and *UPGD* [14]), as well as using different activation functions or different optimizers for training (our research has only used SGD, and has not dealt with modern techniques, such as buffers or Adam). Moreover, we understand that some analysis has been restrained by the choice of the utility score formula (presented in Section 3.1), and the normalization that we had to make before the analysis. Further work could deal with different utility definitions. One possibility is the expression used in UPGD algorithm [14] that would reduce the need for normalization before the analysis. As we are still lacking in understanding how to induce certain utility score distributions, one way would be to try combining existing methods with the utility scores and analyze how that changes the utilities. As an example, one might investigate how the dropout layer that is based on the utility scores would affect the utility score distribution. Hopefully, future work will assist in an even better understanding of the role of neuron utility scores and their relation to plasticity.

References

[1] Jordan T. Ash and Ryan P. Adams. "On Warm-Starting Neural Network Training". In: *arXiv preprint arXiv:1910.08475* (2020). arXiv: 1910.08475 [cs.LG].

[2] Shibhansh Dohare et al. "Loss of plasticity in deep continual learning". In: *Nature* 632 (2024). Published online: 21 August 2024, pp. 768–774. DOI: 10.1038/s41586-024-07711-7.

[3] Clare Lyle et al. "Understanding Plasticity in Neural Networks". In: *Proceedings of the 40th International Conference on Machine Learning*. Ed. by Andreas Krause et al. Vol. 202. Proceedings of Machine Learning Research. PMLR, 23–29 Jul 2023, pp. 23190–23211.

[4] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022.

[5] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958.

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

[7] Vitaliy Chiley et al. "Online Normalization for Training Neural Networks". In: *CoRR* abs/1905.05894 (2019). arXiv: 1905.05894.

[8] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. DOI: 10.48550/arXiv.1502.03167. arXiv: 1502.03167 [cs.LG].

[9] Ghada Sokar et al. "The Dormant Neuron Phenomenon in Deep Reinforcement Learning". In: *Proceedings of the 40th International Conference on Machine Learning*. Ed. by Andreas Krause et al. Vol. 202. Proceedings of Machine Learning Research. PMLR, 23–29 Jul 2023, pp. 32145–32168.

[10] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. DOI: 10.1038/nature14236.

[11] Denis Yarats, Ilya Kostrikov, and Rob Fergus. "Image Augmentation Is All You Need: Regularizing Deep Reinforcement Learning from Pixels". In: *International Conference on Learning Representations*. 2021.

[12] Hojoon Lee et al. "Slow and Steady Wins the Race: Maintaining Plasticity with Hare and Tortoise Networks". In: *arXiv preprint arXiv:2406.02596* (2024). Accepted at ICML 2024. DOI: 10.48550/arXiv.2406.02596.

[13] Timo Klein et al. *Plasticity Loss in Deep Reinforcement Learning: A Survey*. WorkingPaper. arXiv, 2024. DOI: 10.48550/arXiv.2411.04832.

[14] Mohamed Elsayed and A. Rupam Mahmood. "Addressing Loss of Plasticity and Catastrophic Forgetting in Continual Learning". In: *Proceedings of the 12th International Conference on Learning Representations (ICLR)*. 2024. DOI: 10.48550/arXiv.2404.00781. arXiv: 2404.00781 [cs.LG].

[15] Jonathan Frankle and Michael Carbin. "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks". In: *International Conference on Learning Representations*. 2019.

[16] Delft High Performance Computing Centre (DHPC). *DelftBlue Supercomputer (Phase 2)*. https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2. 2024.

[17] Delft AI Cluster (DAIC). *The Delft AI Cluster (DAIC), RRID:SCR$_0$25091*. 2024. DOI: 10.4233/rrid:scr_025091. URL: https://doc.daic.tudelft.nl/.

[18] Shibhansh Dohare, A. Rupam Mahmood, and Richard S. Sutton. "Continual Backprop: Stochastic Gradient Descent with Persistent Randomness". In: *CoRR* abs/2108.06325 (2021). arXiv: 2108.06325.

[19] Richard S. Sutton and Steven D. Whitehead. "Online Learning with Random Representations". In: *Proceedings of the Tenth International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann, 1993, pp. 314–321.

[20] Kaiming He et al. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. IEEE Computer Society, Dec. 2015, pp. 1026–1034. DOI: 10.1109/ICCV.2015.123.

[21] Yann LeCun et al. "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[22] Sabine Roeser and Samantha M. Copeland. *TU Delft Code of Conduct: Why What Who How*. English. Report. Accessed 20 June 2025. Delft, Netherlands: Delft University of Technology, 2020, p. 12. DOI: 10.4233/uuid:704e72b8-6b14-4cf1-a931-9c0f93c50152. URL: https://doi.org/10.4233/uuid:704e72b8-6b14-4cf1-a931-9c0f93c50152.

A Equivalence of the L2 and Shrink and Perturb

As discussed by Ash and Adams [1], Shrink and Perturb is very similar to noisy L2 regularization [4]. Indeed, if the optimizer is SGD, and Shrink and Perturb is performed at every iteration, it is equivalent to L2 regularization with an extra perturbation (parameter noise) step. Here we present the formal proof. In our research, we have used this result in the implementation.

Theorem A.1. *Suppose Stochastic Gradient Descent is used. Let's denote the new value that the learnable parameter θ_i obtains after applying the gradient descent step when using Shrink and Perturb as $\theta_{i,snp}$, and when using L2 regularization with a perturb step, as $\theta_{i,l2}$. Then, for each hyperparameter choice for L2, there exists a hyperparameter choice for Shrink and Perturb, such that for all learnable parameters θ_i it holds that $\theta_{i,l2} = \theta_{i,snp}$.*

Proof. Let's denote the learning rate when using L2 regularization as η_{l2}, the parameter penalization value as λ_{l2}, and the additional random perturb value as p_{l2}. Then, the loss function can be written as

$$L(\theta)_{l2} = l(\theta) + \lambda_{l2}\|\theta\|_2^2,$$

where $l(\theta)$ is the loss function without L2 regularization. Let's take arbitrary values for η_{l2}, λ_{l2}, and p_{l2}.

The number of learnable parameters is denoted as n. Let's choose the hyperparameters for Shrink and Perturb as follows: learning rate as $\eta_{snp} = \frac{\eta_{l2}}{1-\eta_{l2}\frac{2\lambda_{l2}}{n}}$, shrink value as $\lambda_{snp} = 1 - \eta_{l2}\frac{2\lambda_{l2}}{n}$, and random perturb value as $p_{snp} = p_{l2}$.

Let's take an arbitrary learnable parameter θ_i. When using Shrink and Perturb, once the gradient descent is applied, the parameter will obtain the value

$$\theta_{i,snp} = \lambda_{snp}\left(\theta_i - \eta_{snp}\frac{\partial l(\theta)}{\partial \theta_i}\right) + p_{snp}.$$

When using L2 regularization (without perturb), the value is updated as

$$\theta_i := \theta_i - \eta_{l2}\frac{\partial L(\theta)}{\partial \theta_i} = \theta_i - \eta_{l2}\left(\frac{\partial l(\theta)}{\partial \theta_i} + \frac{2\lambda_{l2}}{n}\theta_i\right).$$

If the perturb is also used with L2, then the new value obtained is

$$\theta_{i,l2} = \theta_i - \eta_{l2}\left(\frac{\partial l(\theta)}{\partial \theta_i} + \frac{2\lambda_{l2}}{n}\theta_i\right) + p_{l2}.$$

After rewriting, we get that

$$\begin{aligned}\theta_{i,l2} &= \left(1 - \eta_{l2}\frac{2\lambda_{l2}}{n}\right)\left(\theta_i - \frac{\eta_{l2}}{1-\eta_{l2}\frac{2\lambda_{l2}}{n}}\frac{\partial l(\theta)}{\partial \theta_i}\right) + p_{l2} \\ &= \lambda_{snp}\left(\theta_i - \eta_{snp}\frac{\partial l(\theta)}{\partial \theta_i}\right) + p_{snp} \\ &= \theta_{i,snp}.\end{aligned}$$

Since θ_i and L2 hyperparameter values were chosen arbitrarily, then for all learnable parameters θ_i and for all L2 hyperparameter values it holds that there is a hyperparameter choice for Shrink and Perturb that $\theta_{i,l2} = \theta_{i,snp}$. □

B Comparison of Different Bit-Flipping Strategies in Slowly-Changing Regression

Slowly-Changing Regression problem, as explained by Dohare et al. [2], flips only one bit of the bit-flipping part every T iteration, which is meant to correspond to a somewhat continuous input distribution change. However, we argue that this is only an arbitrary choice, and flipping all bits from the bit-flipping input part is equally reasonable, as it corresponds to a task change (rather than an input distribution change).

Figure 8 shows our attempt to reproduce the results from the paper by Dohare et al. [2]. Using the same bit-flipping strategy has failed to achieve similar results, whereas employing a strategy of flipping all bits when the task changes resulted in a way which corresponds to the presented results in the paper by Dohare et al. [2]. To be more precise, the different learning rates show clear trends and are easier to analyze and compare. Because of that, this bit-flipping strategy was chosen for our experimental setup.

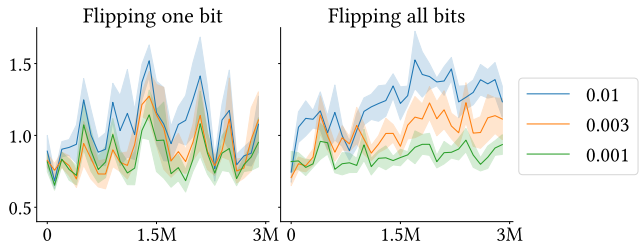

Figure 8: Comparison of different experimental setups for Slowly-Changing Regression. The lines indicate the mean squared error of backpropagation with different learning rates. The results are averaged over 5 runs. The solid lines represent the mean and the shaded regions correspond to ±1 standard error.

C Hyperparameters

Because the experimental setup was changed from the one used by Dohare et al. [2], we have performed a hyperparameter search. The values are presented in Table 1, in which the best performing (based on error or accuracy) values are bolded.

D More Analysis of the Utility Scores

Figure 9 presents how the utility scores progress during the training for all six algorithms. BP experiences a significant drop in the utility of most of the neurons in both of the problems. Only the small part keeps high utility scores, indicating that most of the computation is concentrated in only a few units, while most of the other ones become useless. L2 shows a similar trend, but the utilities decrease slower. SnP's utilities have different trends in different problems. In Slowly-Changing Regression, the algorithm can maintain the same level of utility for the neurons throughout the training duration. However, in Online Permuted MNIST, the utilities experience a slight decrease.

Algorithm	Hyperparameter	Slowly-Changing Regression	Online Permuted MNIST
Backpropagation	Learning rate	$\{\mathbf{0.001}, 0.003, 0.01\}$	$\{\mathbf{0.001}, 0.003, 0.01\}$
L2 regularization	Learning rate	0.01	0.003
	Penalty term λ	$\{10^{-5}, 10^{-4}, 10^{-3}, \mathbf{10^{-2}}, 10^{-1}\}$	$\{10^{-5}, 10^{-4}, \mathbf{10^{-3}}\}$
Shrink and Perturb	Learning rate	0.01	0.003
	Penalty term λ	$\{10^{-5}, 10^{-4}, 10^{-3}, \mathbf{10^{-2}}, 10^{-1}\}$	$\{10^{-5}, 10^{-4}, \mathbf{10^{-3}}\}$
	Perturb noise	$\{10^{-6}, 10^{-5}, \mathbf{10^{-4}}\}$	$\{10^{-6}, \mathbf{10^{-5}}, 10^{-4}\}$
CBP	Learning rate	0.01	0.003
	Maturity threshold	100	100
	Decay rate	0.99	0.99
	Replacement rate	$\{10^{-6}, 10^{-5}, \mathbf{10^{-4}}, 10^{-3}, 10^{-2}\}$	$\{10^{-6}, 10^{-5}, \mathbf{10^{-4}}\}$
CBP + L2	Learning rate	0.01	0.003
	Maturity threshold	100	100
	Decay rate	0.99	0.99
	Replacement rate	$\{10^{-6}, 10^{-5}, \mathbf{10^{-4}}, 10^{-3}, 10^{-2}\}$	$\{10^{-6}, \mathbf{10^{-5}}, 10^{-4}\}$
	Penalty term λ	$\{10^{-5}, 10^{-4}, \mathbf{10^{-3}}, 10^{-2}, 10^{-1}\}$	$\{10^{-5}, 10^{-4}, \mathbf{10^{-3}}\}$
CBP + SnP	Learning rate	0.01	0.003
	Maturity threshold	100	100
	Decay rate	0.99	0.99
	Replacement rate	$\{10^{-6}, 10^{-5}, \mathbf{10^{-4}}, 10^{-3}, 10^{-2}\}$	$\{\mathbf{10^{-6}}, 10^{-5}, 10^{-4}\}$
	Penalty term λ	$\{10^{-5}, \mathbf{10^{-4}}, 10^{-3}, 10^{-2}, 10^{-1}\}$	$\{10^{-5}, 10^{-4}, \mathbf{10^{-3}}\}$
	Perturb noise	$\{10^{-6}, \mathbf{10^{-5}}, 10^{-4}\}$	$\{\mathbf{10^{-6}}, 10^{-5}, 10^{-4}\}$

Table 1: Summary of the hyperparameters used for each algorithm and each problem. For the values that were tried in the hyperparameter search, all of them are presented in the set notation, with the best performing one bolded. Note that if in the same problem using the same algorithm, several hyperparameters are in the set notation, it means that during the hyperparameter search each combination was tried, resulting in a Cartesian product of the given sets. *CBP* stands for Continual Backpropagation, *CBP+L2* is CBP variation with L2 regularization, and *CBP+SnP* is CBP variantion with Shrink and Perturb.

Contrary to the other algorithms, CBP maintains the stable level of utilities in both of the problems. One could also note that for Online Permuted MNIST a major part of the neurons have significantly lower utilities than the 50 highest ones. Finally, from the plots it is evident that in both of the problems adding L2 or SnP to the CBP does not affect the utility scores in a significant way.

To analyse how the task change affects utility scores, Figure 10 displays utility scores in different iterations of the last task for both problems. In Slowly-Changing Regression, BP and L2 do not experience any strong reaction to a task change, while the other algorithms (SnP, CBP and its variants) have an increase in all of the utility scores (except the closest to zero) in the 100th, as compared to the 10 000th (last) iteration. Moreover, the most noticable reaction to the task change is the increase of the neurons that have utility scores close to zero during the first 1 000 iterations. This number decreases for the 3 000th iterations and later even more for the 10 000th one. In Online Permuted MNIST, BP does not show any noticable changes when task switches, but the other algorithms experience a slight increase of the number of neurons with utility close to zero during first 1 000 iterations, which later decreases. In general, the reaction to the change of task is mostly evident in Slowly-Changing Regression, and in both of the problem it appears mostly as an increase of the neurons with utility close to zero.

E Prompts for Chatbots

Chatbots (ChatGPT[3] and Claude[4]) were utilized for the clarification of programming errors, syntax and example library usage. Here are the example prompts that we have asked:

- (ChatGPT) "How to save array of Torch tensors to a file, and later read from it? Possible to use the Pickle library.";
- (ChatGPT) "How to do the modulus operation in shell script, is it: index % num_runs?";
- (Claude) "How to do English quotation marks in LaTeX?"
- (Claude) "I am getting error: Traceback (most recent call last): File …".

The other used prompts are very similar to the shown ones.

[3]https://chatgpt.com/

[4]https://claude.ai/

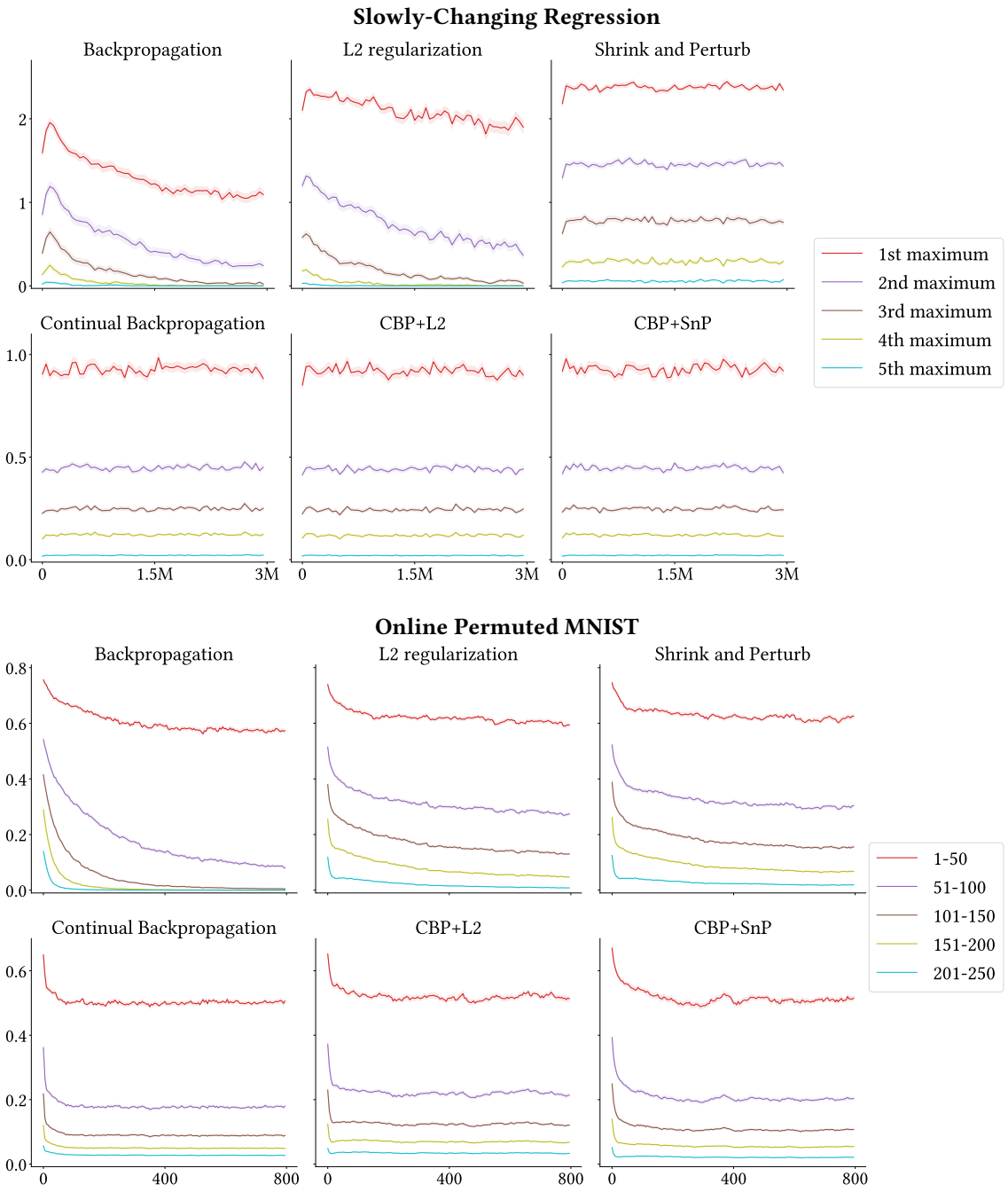

Figure 9: Utility scores of the individual units or unit groups for all algorithms. *CBP+L2* and *CBP+SnP* stand for the Continual Backpropagation variantions with L2 regularization and Shrink and Perturb correspondingly. In the Slowly-Changing Regression, *1st maximum* shows the progression of the maximum utility score, *2nd maximum* shows one of the 2nd maximum, and so on. Similarly, for Online Permuted MNIST, the line *1-50* shows the progression of the average of the highest 50 utility scores, *51-100* shows one of the next 50 highest scores, and so on. The results are averaged over 100 runs for Slowly-Changing Regression, and over 30 runs for Online Permuted MNIST. The solid lines represent the mean and the shaded regions correspond to ±1 standard error. For Online Permuted MNIST, utility scores are normalized layer-wise to the range $[0; 1]$.

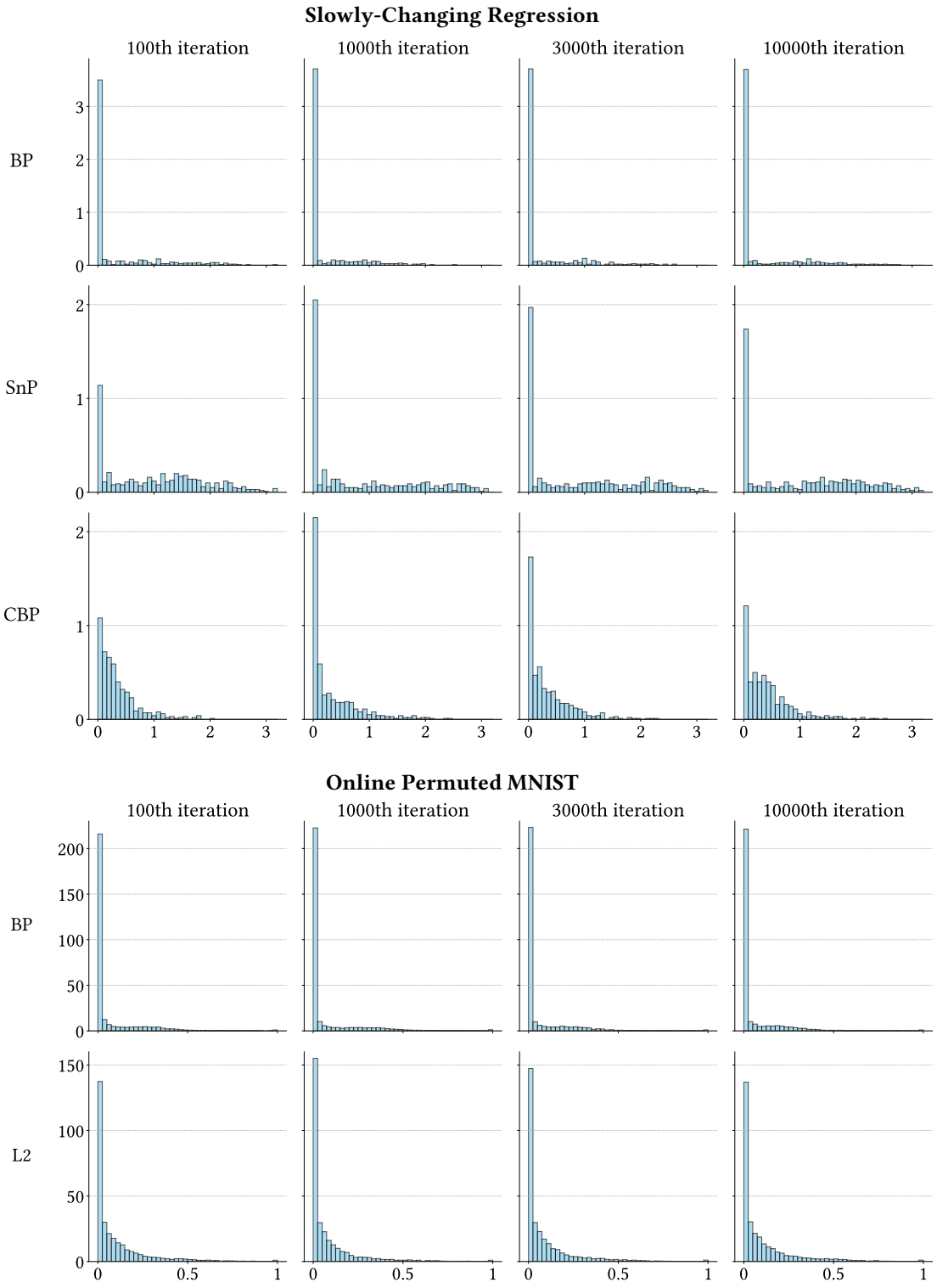

Figure 10: Change of utility score distributions for the last task in Slowly-Changing Regression (300th task) and Online Permuted MNIST (800th task). In both problems, one task spans 10^4 iterations, and here the histograms of the 100th, 1 000th, 3 000th, and 10 000th (last one) iterations are presented. x-axis denotes the utility scores, and y-axis denotes the binned frequency of the utility scores. Histograms are averaged over 100 runs for Slowly-Changing Regression, and over 30 runs for Online Permuted MNIST. For Slowly-Changing Regression, L2 regularization (L2) is omitted as it behaves the same way as Backpropagation (BP); Continual Backpropagation variations with L2 (CBP+L2) and Shrink and Perturb (CBP+SnP) are omitted as they behave the same way as Continual Backpropagation (CBP). For Online Permuted MNIST, Shrink and Perturb (SnP), CBP and its variations are omitted as they behave the same way as L2.