Project report IANNwTF

Lane von Bassewitz
1004025, lvonbassewit@uni-osnabrueck.de

Abstract

Artificial neural networks (ANNs) are increasingly integrated into various aspects of our lives, showcasing remarkable performance across numerous domains. However, they exhibit a significant weakness in sequential learning, often leading to catastrophic forgetting. This phenomenon results in a drastic drop in network performance when learning new tasks. To mitigate catastrophic forgetting, several methods have been proposed. Following the approach of Masse et al. (2018), this project implemented three of those methods: Elastic Weight Consolidation, Synaptic Intelligence, and Context-dependent Gating, combined with a feedforward network trained on the permuted MNIST task. All three methods notably enhanced accuracy in sequential task learning. Furthermore, the combination of these methods had an additional positive impact on mean accuracy. However, it was not possible to replicate the results reported from Masse et al. (2018) in the scope of this work.

1 Introduction

Artificial neural networks (ANNs) have been used successfully in a variety of fields, including video games, image processing, as well as science. In some cases, they already have surpassed human performance (Serre, 2019). However, in contrast to humans, ANNs have a huge weakness when it comes to continual learning, which is the ability to learn one task after the other without having a big decrease in performance (Kirkpatrick et al., 2017). For humans, continual learning is extremely important as we have to constantly adapt to a dynamic environment (Wang et al., 2024). When ANNs are confronted with tasks that need sequential learning, they show a phenomenon called catastrophic forgetting. This means that when an ANN is trained sequentially on multiple tasks, its performance on the previous tasks decreases drastically after training on the new task. This phenomenon occurs due to weight updates intended to learn the current task, often made without considering their potential impact on the performance of the previous tasks (Masse et al., 2018). Multiple algorithms and methods have been proposed to solve this problem in different ways (Wang et al., 2024).

Inspired by Masse et al. (2018), this work aims to implement three neuroscience-based methods against catastrophic forgetting. These methods consist of two regularization-based methods, namely Elastic Weight Consolidation (EWC, (Kirkpatrick et al., 2017)) and Synaptic Intelligence (SI, (Zenke et al., 2017)), as well as the Context-dependent Gating method (XdG, (Masse et al., 2018)). All three will be implemented individually; in addition, EWC and SI will each be combined with XdG. To test these methods, a densely connected feed-forward network in conjunction with the permutation MNIST dataset will be used. The achieved results will be analyzed and compared to the findings of Masse et al. (2018).

2 Literature & Related work

The problem of catastrophic forgetting is long known and has been investigated for a long time (McCloskey and Cohen, 1989; French, 1999). When an ANN learns a new task, it undergoes parameter adjustments based on gradients to minimize the associated loss function. However, when the network is subsequently trained on another task, the gradient-based updates for this new task occur without consideration of the loss landscape of the initial task. As a result, this leads to a significant decrease in performance as the network's parameters are being optimized for the new objective. Therefore, the challenge of continuous learning in ANNs is to balance the integration of new tasks and the preservation of knowledge from previous tasks (Hadsell et al., 2020). A critical aspect of continual learning is to understand how the format and identity of inputs and outputs influence task performance. To clarify this concept, various subtypes have been defined (Wang et al., 2024) but, due to the limited scope of this work, further explanation will focus solely on the used problem, the permutated MNIST task. Goodfellow et al. (2015) defined this as an input reformatting problem, where both input and output semantics remain consistent across all tasks, due to simply being different permutations (Masse et al., 2018). For MNIST, the pixel intensities serve as input, and the digit identities as output.

In parallel, there exists a spectrum of methods to increase the ability of ANNs for continual learning, each targeting specific aspects of the learning process. Subgroups can be defined depending on the aspect of the learning process the approach targets. For example, Wang et al. (2024) defines five subcategories:

- **regularization-based approach**: adds a penalty term dependent on the old task
- **replay-based approach**: past data distributions are stored and replayed
- **optimization-based approach**: used specific optimization algorithms
- **representation-based approach**: focuses on learning robust and invariant representations that can be used in multiple tasks
- **architecture-based approach**: adapts the network architecture, for example, dynamically extends or divides the network into submodules

The EWC and SI methods are inspired by the synaptic plasticity observed in the mammalian brain, where intricate mechanisms prevent interference between old and new knowledge (Hadsell et al., 2020), and aim to emulate such mechanisms within ANNs. They achieve this by stabilizing weights crucial for previous tasks by adding penalty terms to the loss function, mitigating changes in these weights. In the best-case, this achieves a good balance in performance trade-offs among the individual tasks. The XdG method, however, is an architecture-based approach and draws inspiration from context signaling in the brain. In this paradigm, context-dependent signals enable the brain to process information task-dependently. Moreover, these signals can selectively excite a non-overlapping set of units, ensuring that changes occur only within a limited set of synapses and do not disrupt the performance of previous tasks. In ANNs, this can be done relatively easily by simply multiplying a specific amount of units per task with zeros and therefore gating them (Masse et al., 2018).

3 Methods & Implementation

3.1 Dataset - permutated MNIST

The MNIST dataset is a commonly used dataset in AI research published by LeCun et al. (2010). The dataset contains greyscale images of handwritten digits from zero to nine (Figure 1). It can be downloaded via the "tf.keras.datasets" module (https://www.tensorflow.org/api_docs/python/tf/keras/datasets/mnist/load_data) and is provided as NumPy arrays with shapes of 28x28 in uint8 type. The dataset is split into train and test sets, where the train set contains 60,000 images and the test set 10,000 images. To generate different tasks, I permutated the original images by randomly rearranging the pixels of the image (Figure 1), therefore every task equals one permutation. This set of tasks is a classification and input reformatting problem.

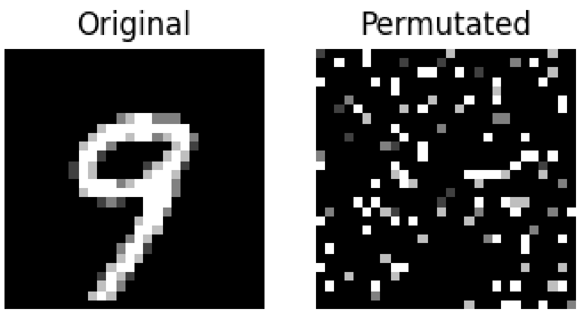

Figure 1: Left: MNIST, right: permutated MNIST

The dataset was downloaded in the form of two tuples, one containing the images and labels for the train set, and the other containing images and labels for the test set. For preprocessing, the images were normalized, casted to a float32 type, and flattened to an array with dimensions 784. Subsequently, the task-specific permutation was made and the dataset was shuffled and batched.

3.2 Network Architecture

Based on Masse et al. (2018), a fully connected feed-forward network with an input layer with 784 units (dimensions of the flattened MNIST image array) was implemented. The network also contains two hidden layers with 2,000 units each, and the Rectifier Linear unit (ReLu) was used as an activation function. Furthermore, there was a

50 % dropout applied to the hidden layers during training. The output layer contained ten units in accordance with the ten different classes.

The TensorFlow sequential application programming interface (API) was used to implement the model. It allows an easy and uncomplicated implementation of standard ANNs using the layer modules also provided by TensorFlow.

3.3 Training

The networks were trained for 20 epochs (batch size of 256) on each task. For weight updates, the Adam optimizer was used, with the parameters being reseted after each task. As this is a classification task, cross-entropy was used as a loss function. After each epoch, validation was performed on the current task and all previously learned tasks.

The basic training loop follows a conventional structure: a nested 'for' loop iterates through each batch for every epoch, with gradients computed using the gradient tape and parameter updates applied via the optimizer. This process tracks metrics such as loss and accuracy. Each method's extensions to this loop will be discussed individually under their respective sections. However, the validation process differs slightly, given multiple validation tasks. These tasks are provided to the function as a list and are subsequently iterated through. Another additional step involves resetting the parameters by reinitializing the optimizer before each new task.

3.4 Synaptic stabilization Methods

Synaptic stabilization methods are part of the regularization-based approach and are built on the concept of synaptic consolidation in the biological brain. The underlying principle is to identify and stabilize the weights crucial for task performance during subsequent training, thereby preventing big changes to these weights during parameter updates. Both methods discussed here (EWC and SI) achieve this by incorporating a quadratic penalty term into the loss function. This penalty term pushes the network to resist changes to important weights, thereby prioritizing the preservation of performance on previous tasks during parameter updates (Hadsell et al., 2020).

The EWC was proposed by Kirkpatrick et al. (2017). A Fisher information matrix is utilized to estimate the importance of weights after training on a task, capturing their sensitivity within a Bayesian framework. By approximating the true posterior distribution over parameters, it propagates information about weight importance from previous tasks. This offline computation, based on a fraction of data samples, allows EWC to effectively balance the trade-off between preserving old knowledge and adapting to new tasks. From this, an extended loss function is composed, existing of the "original" loss function plus a penalty term:

$$L = L_k + \sum_i \frac{\lambda}{2} F_i(\theta_i - \theta_i^{prev})^2 \tag{1}$$

Whereas L_k is the loss for the current task, λ is a weighting term for old vs. new memories, F_i stands for the entire in the fisher matrix and therefore the importance measurement of the parameter θ_i. θ_i^{prev} represents the parameter value at the end of the previous task.

To implement this method, the Fisher matrix had to be computed after finishing training for each task. Therefore, 8192 random samples were picked out of the training set. With the help of the model predictions and the gradient, the Fischer matrix was calculated, saved, and used to determine the penalty loss for the next task. After completing training for a task, the weights of the model are saved to facilitate the calculation of the penalty term, which requires the weights from the end of the previous task. This process ensures that only the weights associated with the most recent task are stored rather than retaining weights from all tasks.

In contrast to EWC, SI estimates the importance of weights online by keeping track of the contributions of each weight during task training. The parameter-specific contributions to the changes in the total loss are summed during training the current task (4) and consequently, changes to parameters with significant impact can be penalized while training the next task (Zenke et al., 2017). These considerations lead to the following loss function:

$$L = L_k + c\sum_i \Omega_i(\theta_i - \theta_i^{prev})^2 \tag{2}$$

With

$$\Omega_i^k = max(0, \sum_{\nu<k} \frac{\omega_i^k}{(\Delta\theta_i)^2 + \xi}) \tag{3}$$

$$\omega_i^k = \sum_t (\theta_i(t) - \theta_i(t-1)) \frac{-\partial L_k(t)}{\partial \theta_i} \tag{4}$$

In this equation, alongside the symbols already utilized in EWC section, Ω denotes the estimated importance of the single parameter. $\Delta\theta_i$ stands for the total change in each parameter. ξ is used as damping parameter to limit the expression in scenarios where $\Delta\theta_i$ goes to zero. The expression $\frac{-\partial L_k(t)}{\partial \theta_i}$ can be interpreted as the per-parameter contribution to the changes in loss.

For the implementation of this method, before the start of training, ω and Ω need to be initialized. Since they represent per-parameter values, they are the same size as the model weights. The training loop has to be extended to keep track of the ω during each task. For this, the running sum of the product of the parameter update and the gradient is tracked per task. In this implementation, the running sum is approximated using the learning rate and gradient. After each task, the ω are used to calculate the Ω after equation XX. Subsequently, for each new task, the value of the ω is reset and the updated Ω is used to calculate the penalty loss. The penalties term calculation requires weights from the end of training of the previous task, which were provided in the same manner as in the EWC implementation.

3.5 Context dependent Gating (XdG)

Inspired by context signals in the biological brain, this method gates for each task a unique set of units in the hidden layer of the network and, through that, forces the network to use only subsets of units for each new task. The number of gated units can be seen as a hyperparameter (Masse et al., 2018).

Accordingly, a custom layer was implemented to apply these masks to the network's hidden layer. When no mask is provided, this layer simply returns the output. However, when a mask is set, the input is multiplied by the mask. Given that the mask contains a specific percentage of zeros, this effectively gates the corresponding percentage of units. These mask layers are directly added after the hidden layers in the network architecture. For each task, masks are generated by creating arrays with 80 % zeros and 20 % ones, matching the dimensions of the hidden layers. These masks are then stored in a list, as they will be required for testing later on. During training, the generated mask specific to each task is utilized. Additionally, the training function receives the list containing masks for all tasks. During testing, the mask corresponding to the respective task is employed.

3.6 Hyperparameter tuning

To identify the most suitable hyperparameters for each method, a grid search with a small hyperparameter space was conducted. The parameter ranges suggested by Masse et al. (2018) were referenced for guidance, but only a subset of these ranges was explored, due to computational constraints. For EWC, the parameter space was defined as c = 0.2, 2, 40, 200, 1000, 5000. It's noteworthy that, contrary to Masse et al. (2018), in this implementation c was multiplied by 0.5. Consequently, a value greater than stated in the paper was employed. As for SI, the search space encompassed the parameters c = 0.01, 0.1, 0.5, 1, 2 and ξ = 0.01.

4 Results

Figure 2 illustrates the mean accuracy achieved by each individual method across 10 tasks. For comparison, a model without a stabilization method (green line) was trained sequentially on 10 MNIST permutation tasks. Throughout these tasks, there was a drastic decline in mean accuracy. The accuracy for a single task was 0.9781, whereas the mean accuracy after ten tasks dropped to 0.5079. Applying the EWC algorithm (orange line) resulted in a significant improvement in mean accuracy, concluding at 0.7842. The model utilizing the SI algorithm (blue line) exhibited a notable improvement, achieving a mean accuracy of 0.8921 after 10 tasks. Lastly, the XdG Method led to a mean task accuracy of 0.8180.

Additionally, similar to the approach by Masse et al. (2018), SI and EWC were combined with the XdG method to assess whether this combination further enhances the mean accuracy. Figure 3 visualizes the results of that. The combination of SI and EWC exhibits superior mean accuracy compared to each method individually. Specifically, when SI is combined with XdG (green line), it achieves the highest performance of 0.9584. Additionally, integrating XdG also enhances the performance of EWC (red), resulting in a mean accuracy of 0.8902.

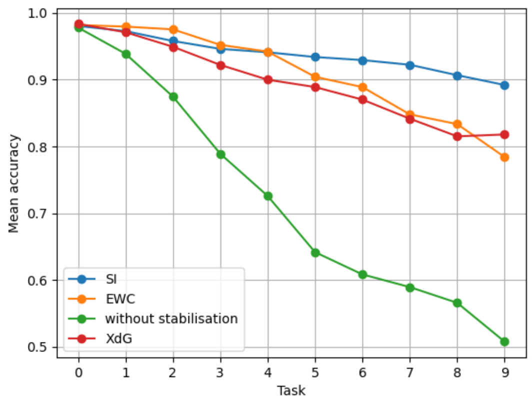

Figure 2: Mean task accuracy for single methods over 10 tasks.

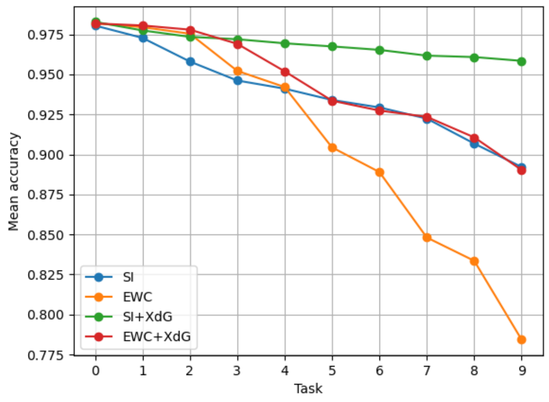

Figure 3: Mean task accuracy over 10 tasks for combined methods.

Unfortunately, no specific hyperparameter settings were provided by Masse et al. (2018). Therefore, a grid search was conducted, but due to time and computational constraints, it was not possible to go through the complete hyperparameter spaces given in Masse et al. (2018). For SI, the parameter c was tested with the following values: 0.01, 0.1, 0.5, 1, and 2. The parameter ξ remained constant and was always set to 0.01. The results are depicted in Figure 4. Based on these results, the optimal hyperparameters identified were c = 2 and ξ = 0.01.

For EWC, a parameter space of c = {0.2, 2, 40, 200, 1000, 5000} was explored. The optimal results were achieved with c = 1000, as illustrated in Figure 5.

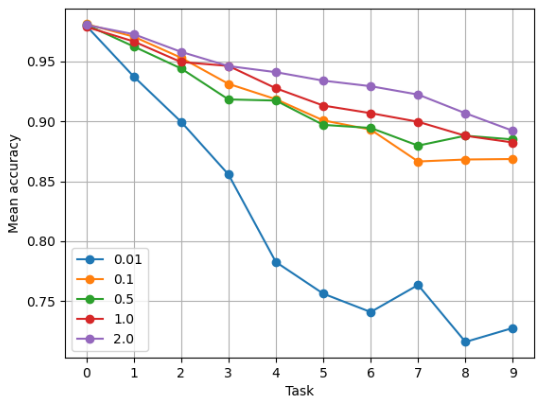

Figure 4: Mean task accuracy over 10 tasks of SI with different values of c and ξ = 0.01.

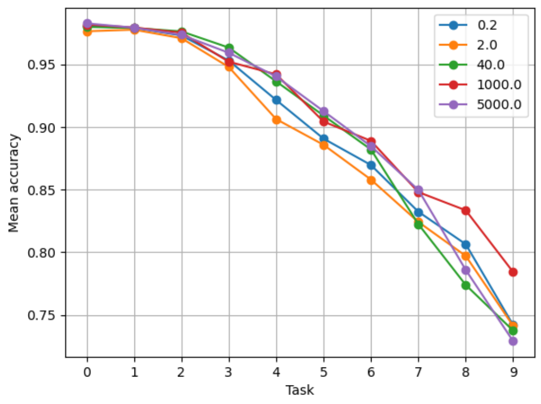

Figure 5: Mean task accuracy over 10 tasks of SI with different values of c.

No hyperparameter search was conducted for the XdG method; instead, the value provided in the Masse et al. (2018) paper, with 80 % of gated units, was adopted. The parameters identified here (SI: c = 2 and ξ = 0.01; EWC: c = 1,000) were also employed to generate the reported results.

5 Discussion

In this project, three distinct methods (SI, EWC, XdG) were implemented to alleviate catastrophic forgetting in ANNs. These methods were evaluated individually (EWC, SI, XdG) and in combination (EWC+XdG, SI+XdG). Each of these three methods by itself significantly improves the network's ability to learn tasks sequentially. However, for all methods, the mean accuracy after 10 tasks was lower than the single-task accuracy. Adding XdG to SI or EWC yielded further improvements in the mean accuracy. Notably, the combination of XdG and SI produced the best result, with accuracy only decreasing by 0.0244. But all results, except for the SI+XdG combination, fell noticeably short of those reported in Masse et al. (2018). In contrast to the reported results for EWC, which reached 0.953 accuracy, the achieved accuracy here was 0.7842. Further, in this project, the SI implementation reached a mean accuracy of 0.8921 for 10 tasks, while (Masse et al., 2018) reported a mean accuracy of 0.97. As for XdG, the reported results indicate an accuracy of 0.971, whereas the achieved accuracy here is 0.8180. Regarding the combined methods, Masse et al. (2018) solely document the mean accuracy for 100 tasks, which, for both methods, stands at 0.954. Therefore, the implementation of the EWC+XdG Methods for less tasks falls notably below that, at 0.8902. As for the combination of SI with XdG, a mean accuracy of 0.9584 can be reported, which is close to the reported accuracy. However, considering that this value is for 100 tasks, it is presumable that the accuracy is still lower than reported in the paper. Unfortunately, aside from the percentage of gated units in XdG, the referenced paper did not provide specific hyperparameter settings but only parameter spaces for each parameter. Since extensive hyperparameter tuning was not conducted in this paper, making a direct comparison is challenging. It remains unclear whether a hyperparameter setting exists that would produce results more similar to those listed in the paper.

Several areas for improvement and further experimentation can be discussed for implementing all three methods. For EWC, it would be interesting to investigate the effects of varying the number of samples used to calculate the Fisher matrix on the results. This could also be seen as a hyperparameter, especially in the context of a trade-off between computation cost and effectiveness. Furthermore, particularly with EWC, a bias towards maintaining high performance for the first task was observed. This suggests that, especially with high values of c, the performance of the first task remained high, even as accuracies for subsequent tasks began to decline. This could potentially be connected to a discussion on the correct formulation of the penalty term for multiple tasks (hus). The implementation presented here includes only one penalty term, with the previous weight defined as the weight at the end of training for the previous task. Further investigation in this area could potentially lead to improvements in implementing this method.

Regarding the SI algorithm, conducting more extensive hyperparameter tuning, particularly exploring different xi values, could result in further improvements. Additionally, during implementation, the parameter updates were approximated only via the learning rate, which is not entirely accurate, as the Adam optimizer considers momentum and other factors. Unfortunately, obtaining the individual parameter updates from the TensorFlow Adam optimizer is not straightforward. Implementing a custom version was not feasible within the given timeframe. Obtaining the correct updates could also potentially lead to improvements.

Due to the omission of my team member, resulting in time constraints, I had to reduce the scope of this project. Initially, the plan was to test the methods on 100 tasks and eventually on the ImageNet dataset, following the approach of Masse et al. (2018). Unfortunately, this goal was not achieved. Nevertheless, all three methods were successfully implemented and showed promising results in alleviating catastrophic forgetting in ANNs.

The code for this report can be found at: `https://github.com/LTvB/IANNwTF23-24/tree/main/FinalProject`

References

Robert M. French. 1999. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135.

Ian J. Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. 2015. An empirical investigation of catastrophic forgetting in gradient-based neural networks.

Raia Hadsell, Dushyant Rao, Andrei A. Rusu, and Razvan Pascanu. 2020. Embracing change: Continual learning in deep neural networks. 24(12):1028–1040. Publisher: Elsevier.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. 2017. Overcoming catastrophic forgetting in neural networks. 114(13):3521–3526.

Yann LeCun, Corinna Cortes, and CJ Burges. 2010. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2.

Nicolas Y. Masse, Gregory D. Grant, and David J. Freedman. 2018. Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization. 115(44).

Michael McCloskey and Neal J. Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. volume 24 of *Psychology of Learning and Motivation*, pages 109–165. Academic Press.

Thomas Serre. 2019. Deep learning: The good, the bad, and the ugly. *Annual Review of Vision Science*, 5(Volume 5, 2019):399–426.

Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. 2024. A comprehensive survey of continual learning: Theory, method and application.

Friedemann Zenke, Ben Poole, and Surya Ganguli. 2017. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3987–3995. PMLR.