Selective Forgetting to Save Capacity in Neural Networks for Continual Learning

Prithvijit Chakrabarty
University of Massachusetts
Amherst
pchakrabarty@umass.edu

Abstract

Catastrophic interference is a well known problem with neural networks. Training on a task B after task A overwrites the weights and degrades performance on A. This report investigates an interesting and closely related feature: selective forgetting.

Preventing catastrophic interference will allow a model to learn multiple tasks sequentially. Selective forgetting allows the user to then select a learned task to be forgotten. This can be used to save model capacity in a continual learning setting. Given a stream of incoming tasks, a continual learning agent may need to learn many tasks, each of which appears for a short period of time. Instead of using large network to learn and remember all tasks, a smaller model may be employed. This uses a policy to identify unimportant tasks and forgets them periodically to free up capacity. A simple modification to the Elastic Weight Consolidation algorithm enables this feature. The report investigates its properties with experiments on permuted MNIST tasks and demonstrates its benefits by applying it to model continual face recognition.

1. Introduction

Catastrophic interference is a well known limitation of neural networks. Once trained to perform task A, training the same network on task B may diminish performance on A. Initially identified in [12], numerous attempts have been made over the years to solve this problem ([6], [17], [2], [15]).

Continual learning requires an agent to learn tasks from a stream of incoming data. The data may be change over time, requiring the agent to be adaptive and learn multiple tasks. At various points in time, the agent may have to learn several tasks together, or sequentially. Clearly, if the data stream presents task A and then task B, failing to prevent catastrophic forgetting will leave the agent performing poorly on A throughout the rest of the stream. Thus, studying this problem is vital for developing autonomous or lifelong learning systems.

1.1. Selective Forgetting

Prior work largely focuses on preventing catastrophic forgetting. While this is necessary, it is only one of the elements required to perform well while learning continually. We suggest that to work on a real world data stream with a large number of tasks, the ability to forget unimportant tasks can be significantly helpful too.

Any model with a finite, fixed size must compromise its effective capacity in order to remember old tasks. If we have a large number of tasks over a long period of time, remembering every task will eventually lead the model to saturation.

A naive solution would be to increase the model capacity as new tasks are introduced (adding new hidden units). This is cumbersome, requiring direct changes to the architecture every time it saturates, which is a slow procedure. Besides, we may be able to identify tasks which will not appear after a certain time. The weights learned for these tasks will not contribute at all to the future performance and preserving them is a waste of memory. This is unlikely to be a scalable solution.

Instead of steadily growing the model, a mechanism to selectively forget tasks would work faster and also save memory. The model can be trained till saturation, following which we may specify a "forget policy" in order to the select which tasks to overwrite as new ones are learned. It is worth noting that schemes which use dynamic architectures (such as [11]) may be indispensable. Indeed, if the number of tasks to be handled simultaneously increases dramatically, we cannot use a small network to learn them all, and will be forced to add more capacity. However, the ability to selectively forget tasks gives us the option to intelligently choose between adding more capacity or simply forgetting a useless task. This will be particularly useful in scenarios where the model must learn a large number of simple tasks, most of which will are useful only for a short period of time.

Besides saving capacity, selective forgetting may have other uses which are useful in practice. For example, consider a scenario in which we have trained a model on multiple tasks, using some scheme to prevent catastrophic forgetting. We then identify a task which was trained with incorrect labels or training data. With forgetting, we can simply forget this task and train with the correct data on the same head.

1.2. Related Work

A variety of approaches have been proposed to prevent forgetting in neural networks. However, these may be grouped into a small number of broad categories. Section 2 in [20] describes one way to categorize them. The surveys [13] and [8] describe further details of these categories and also other ways of classifying them. For selective forgetting, it will be useful to look at 2 of these broad types of approaches, as the problem is significantly different for these categories:

1. *Regularization methods*: These approaches ensure that the optimization for the new task does not affect the previously seen tasks by penalizing a change made to parameters useful for previous tasks. [1] uses a genetic algorithm to identify the routes in the network which are active for a given task and block them while learning the new task. [11] uses the response of the previously learned model on the new task as a regularization factor and adds another head for the new task.

While these methods explicitly block selected weight updates or augment a part of the architecture, a more convenient is a "soft" weighting which indicates the importance of connections. This is done in [9] and [20]. Both these works have a connection to "Optimal Brain Damage" [10], where unimportant weights in a trained model were discovered to reduce the number of parameters. [9] proposes Elastic Weight Consolidation, which identifies important weights in between training 2 tasks using the Fisher information. On the other hand, Synaptic Intelligence [20] runs in an online manner, maintaining a dynamic scale which asses the importance of a weight depending on its activity. In practice, both methods show similar performance ([20] compared them on the standard benchmarks of Permuted MNIST [4], split MNIST and Split CIFAR 10/100 to demonstrate this).

2. *Rehearsal methods*: These methods save data from previous tasks and use it as replay memory to prevent forgetting. [3] introduced GeppNet+STM, which incrementally learned MNIST digit classes. This stores the entire training set from previous tasks. [16] demonstrated that, we can achieve good performance by retaining only a small set of exemplar samples from previous classes.

More recently, the development of generative models has led to increased attention on pseudo-rehearsal methods. Here, instead of directly saving training samples, a generator provides samples of old classes when a new class is added. [19], [7] and [18] are variants of this idea.

In practice, regularization methods work well for multi-task problems where the data of different tasks are independent. Rehearsal based methods seem to be effective for incremental class learning. This is mentioned in [13] and experimentally demonstrated in [8].

The scheme for selective forgetting will be different for these two approaches. Forgetting in the multi-task setting requires forgetting an entire task, whereas the same in incremental learning involves specifically reducing the performance on *a single class* (in order to replace it with another class). Solving both versions will have significant practical utility.

This project focuses on selective forgetting with regularization based methods. Performing the same in the incremental learning setting is left for future work.

2. Selective Forgetting with EWC

The algorithm proposed in [9] is Elastic Weight Consolidation or EWC. Consider a model M dependent on a parameter vector θ. At the current timestep, M has learned N tasks $(T_1, ..., T_N)$ sequentially, i.e., the current parameter vector is θ^* with which M performs well on all tasks seen so far. Let the data corresponding to these tasks be $\mathcal{D}_{T_1}, \mathcal{D}_{T_2}, ..., \mathcal{D}_{T_N}$.

Learning a new task T_{N+1} involves finding a new θ for which the loss on all tasks including the new one is low. With EWC, this is achieved by minimizing the following loss function:

$$\mathcal{L}(\theta) = \mathcal{L}_{T_{N+1}}(\theta) + \sum_i \frac{\lambda}{2}(F_{D_1,...,D_N})_i(\theta_i - \theta_i^*)^2$$

where $(F_{D_1,...,D_N})_i$ is the entry on the diagonal of the Fisher information for the i^{th} parameter of M, computed using the data for all tasks 1 to N. λ is a scaling factor which is chosen as a hyperparameter in practice (see section 2.2).

2.1. Saturation Behavior

The regularization term in EWC penalizes changes made to weights learned for the previous tasks. If the tasks are independent, the network must allocate new weights for each task. As we learn new tasks, more weights are reserved, which reduces the effective capacity of the network.

To study this, we perform a simple empirical capacity measuring experiment. The experiment in [9] is replicated using networks of varying sizes. This requires learning 10 tasks from the permuted MNIST benchmark [4]. A large network (2 layers with $H = 1000$ hidden units each) learns all 10 tasks without a drastic drop in performance (it achieves an accuracy over 0.96 on all tasks). Using $H = 100$ hidden units per layer drives the model to saturation quickly, with degrading performance on new tasks (the final validation accuracy is about 0.95 for the first task and 0.62 for the last).

To check that new weights are being allocated for new task, we measure the correlation between the task-specific Fisher information matrices (Figure1). Cell (i, j) in these matrices is $< \tilde{F}_i, \tilde{F}_j >$, where $\tilde{F}_i$ and $\tilde{F}_j$ are the normalized Fisher information matrices for tasks i and j. If different weights are given importance in these tasks, the matrices will be orthogonal, and the cell (i, j) will be close to 0.

Figure1(a), shows the result on using the large network. There are enough weights to allocate for the new tasks, which means the Fisher information matrices can be orthogonal. Using the smaller network forces some parameters to be (Figure1(b)) to be used for multiple tasks. As we are using EWC with a high λ, we prefer remembering tasks, which compromises performance. Also note that the more tasks we add, the worse the correlation becomes (the bright red cells in Figure1(b) at the bottom right end). This indicates that the model has used almost all its capacity, and any weight we try to allocate will have been already used for a previously seen task.

The effect of our selective forgetting algorithm (section 2.3) is shown in Figure1(c). We observed that due to the limited capacity, we may not be able to eliminate correlation between weights for different tasks. The forgetting procedure allows us to control it and direct it onto selected tasks identified as unimportant, freeing space for those which need to be retained.

2.2. The Scaling Parameter

The scaling factor λ plays an important role in EWC. Setting this parameter is challenging in practice. Its value directly influences the loss and performance varies widely depending on the task and model architecture. To find the optimal value, we must treat this as a hyperparameter. Indeed, this is done in [9], where the validation data from previous tasks are used to select the value of λ which works best. However, this would be not allowed in a strict continual learning setting where we lose access to all data for previously seen tasks.

In our experiments, we first choose a high value for λ which retains learned weights for the given model and task. This is kept constant as new tasks are introduced.

It also must be noted that using a constant λ does not reduce the algorithm to blindly allocating new weights for every new task. Appendix 4.3 in [9] empirically demonstrated that optimizing the EWC loss leads the model to share weights among tasks when possible. To confirm that our scheme to choose λ does not affect this property, we trained the small network ($H = 100$) on controlled variations of the permuted MNIST tasks. Instead of applying a fixed permutation on the entire image, we shuffle only a fraction p of the pixels. Thus, for p close 0, the tasks will be similar, while $p \approx 1$ will lead to a permuting all the pixels in the image. Figure2 shows the results. A low value of p causes a lower drop in performance, suggesting that the model saves capacity by re-using weights for similar tasks. As p is grows to 1, the tasks become increasingly dissimilar, and more distinct hidden units are required to learn them all. As we are using a small model, retaining the previous tasks reduces the capacity and compromises performance on the latter tasks. As control, we also train the model without EWC using $p = 0.2$ and confirm that the model retains the tasks only when using the EWC loss with the fixed scaling factor.

Adjusting λ gives us a limited control over forgetting previous tasks. However, it controls forgetting *all* previously learned tasks. Setting λ too low will lead to forgetting all tasks, while a high value will lead to remembering them all. Instead of this, we would like to be able to specify which tasks in the history were important and remember those while forgetting others to free capacity.

2.3. Weighted FIM Reconstruction

To control performance on selected tasks we decompose the regularization term in the EWC loss into task-specific parts.

EWC assumes that the data for the tasks are independent:

$$\mathbb{P}[\mathcal{D}_j | \mathcal{D}_i] = \mathbb{P}[\mathcal{D}_j] \qquad \forall i, j \in [1, N], j > i$$

Using this and the chain rule for Fisher information, we may decompose F into task dependent components as:

$$F_{\mathcal{D}_1, \ldots, \mathcal{D}_{T_N}} = \sum_{i=1}^{N} F_{D_i | D_1, \ldots, D_{i-1}} = \sum_{i=1}^{N} F_{D_i}$$

Thus, the final Fisher information matrix (FIM) is the sum of Fisher matrices per task.

We store the task-specific Fisher matrices. To selectively forget a task, we use a 'remember vector' $\tau \in [0, 1]^N$. $\tau_i = 1$ indicates we want to remember the i^{th} task, while $\tau_i = 0$ may lead to forgetting the corresponding task (i.e., training on T_{N+1} may overwrite the weights used for T_i)

With these, we construct a new weighting matrix $\hat{F}_i = \sum_{i=1}^{N} \tau_i F_{D_i}$ and use this in the EWC loss function:

$$\mathcal{L}_T(\theta) = \mathcal{L}_{T_{N+1}}(\theta) + \frac{\lambda}{2} \sum_i \hat{F}_i (\theta_i - \theta_i^*)^2$$

Figure2 shows the results of using this algorithm. As before we, train 10 tasks from the permuted MNIST dataset on a small network ($H = 100$). Fig. 2(a) shows what happens when we try to remember all the tasks. Initially the model is plastic and uses as many hidden units as required to attain the high validation accuracy. As we retain more tasks,

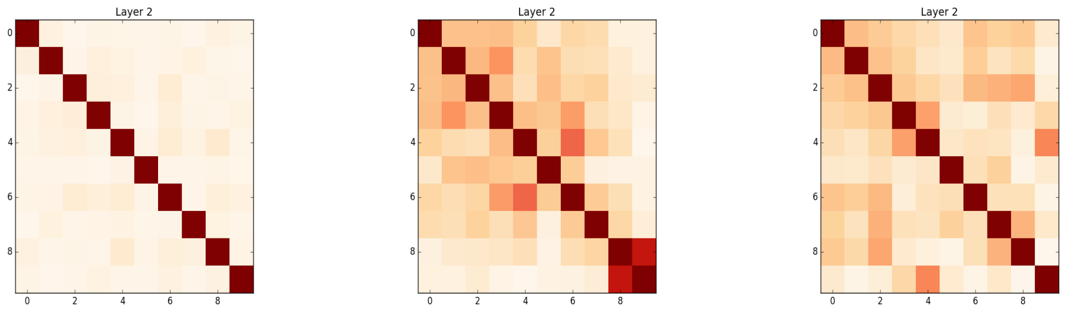

Figure 1. Orthogonality of the task dependent FIMs with (a) $H = 1000$ (b) $H = 1000$, remembering all tasks and (c) $H = 100$ with selective forgetting

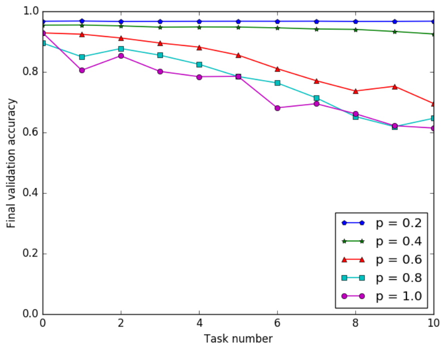

Figure 2. Saturating a small network with tasks involving different degrees of permutation

the effective capacity reduces and the model converges to a lower validation accuracy.

The effect of not using EWC is shown in Figure2(b). Gradient descent with dropout causes every new task to interfere with previous ones (learned weights are overwritten). Once a task is learned, the validation accuracy on that drops as soon as a new task is introduced.

Selective forgetting is shown in Fig. 2(c). The forget policy in this experiment was: while learning tasks after task 5, forget tasks 0, 1, 3 and 5. Clearly, the maximum validation the slowly drops till task 5. At this point, we forget using this policy and free capacity. For the next task, the model quickly converges to a higher validation accuracy. Also, the procedure forgets only the selected tasks. The performance on tasks 2 and 4 are unchanged.

The final validation accuracies for the different settings are shown in fig. 3. Remembering all tasks lead to a steady drop, while direct gradient descent with dropout makes the model work well only on the most recent task. With our procedure, we can control which tasks should be forgotten. Only those suffer the drop in accuracy, making space for the important tasks.

3. Modeling Continual Face Recognition

In this section, we demonstrate a practical use of our method on the problem of continual face recognition. This also demonstrates a use of the algorithm for tasks more complex than permuted MNIST digits. Consider the problem of developing an agent A which must recognize faces over a long period of time. The agent lives in a world with K environments. We model it as a stochastic process, randomly moving across these environments. Each environment also has a set of members in it (a *distinct* set of N people per environment). For example, these environments may correspond to home, school, gym, etc., and the model must move from one to the other, recognizing people it it.
At every timestep, A must do one of 2 things:
1. Recognize people: This corresponds to A entering an environment and ”meeting” a person there. We measure the ability of the agent to do this by directly computing the accuracy of the face recognition model on the validation sets of people in the environment.
2. Learn a new set of people in the environment: The members in an environment may change membership (in the example, this may correspond to the agent changing school, moving homes, etc.). If this occurs, the agent may forget the identities of the previous set of people in that environment (we assume they will not return again).

In this setting, the agent must learn to handle at least K tasks simultaneously. At no point in its lifetime, will it be required to perform well on more than K tasks. Also, selective forgetting is vital in this setting, i.e., we must be able to specify which task we want the agent to forget, depending on which environment sees a change in members.

One approach to solving this problem would be using a large model with many classes recognizing all possible

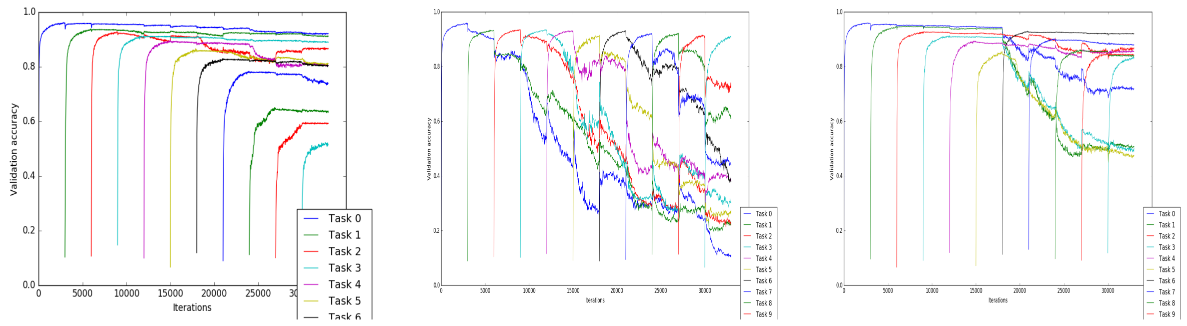

Figure 3. Validation curves on permuted MNIST tasks (a) remembering all tasks (b) SGD+dropout (c) Selective forgetting

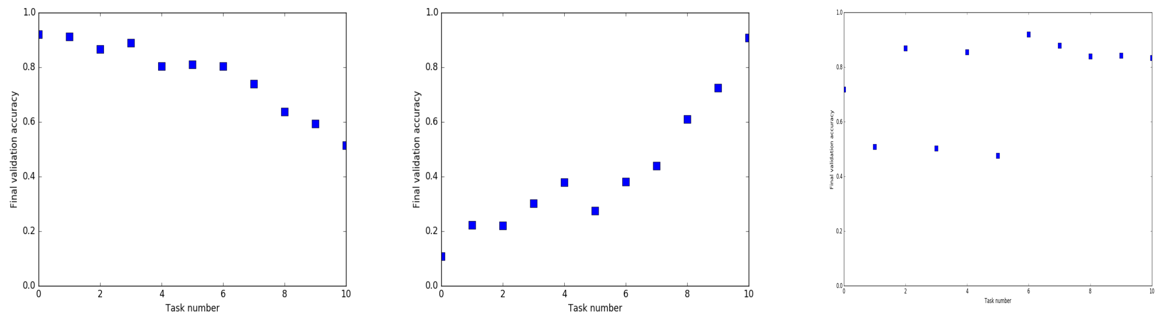

Figure 4. Final validation accuracy (a) remember all tasks (b) SGD+dropout (c) selective forgetting

faces that might be encountered. Another approach would be to use the embeddings and store identities in an auxiliary data structure. Here, we propose using selective forgetting as an interesting alternative, where a model performs well by capitalizing on the fact that certain tasks will not be seen again.

To implement this, we used the embeddings from a standard face recognition model [14]. These are embeddings in $\mathbb{R}^{4096}$ obtained as the output of the $fc7$ layer in the model. We use K classification heads (one corresponding to each environment) which map the embeddings to a one-hot person identity. 500 random identities from the MSCeleb dataset [5] were chosen such that each person had at least 100 image samples. We used 40 samples for training and 60 for validation. These people were divided into 50 non-overlapping groups, each corresponding to a different task (10 people per task). Thus, for each task, we had a balanced training set with 400 samples and a validation set with 600 samples. This may be a small number of samples on absolute terms, but is sufficient, as we are using a pre-trained feature extractor and each individual task is simple. Our agent is modeled in a world with $K = 4$ environments. At each timestep, we randomly move to one of these environments. When a change in membership is encountered, we simply forget the task learned at that environment and replace it with the new task. Thus, at every timestep, the model is well-trained on 4 *active tasks*.

To measure if the model performs well, we track performance on the active tasks across time. Figure5(a) shows the results. This shows the minimum validation accuracy on the active tasks across time as new tasks are learned. As selective forgetting focuses using the capacity on these active tasks, performance on them remains relatively high. The lower performance on the latter tasks while trying to remember everything pulls down performance on active tasks.

3.1. Fluctuations in Validation Curves

Figure5 shows the validation accuracy curves with and without selective forgetting. For clarity, we show the accuracy for 4 out of the 30 tasks. As we might expect, in Figure5(b), the performance on the tasks rise on training, is held high while it remains an active task on an environment, and drops when forgotten. There are also spikes in the curve. These correspond to timesteps when new tasks

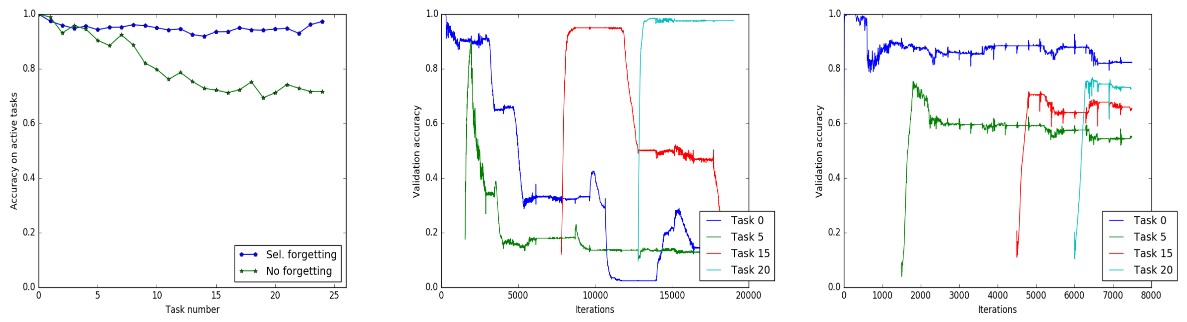

Figure 5. Performance on the face recognition tasks. Validation accuracy on (a) active tasks (b) with forgetting (c) without forgetting

were learned. As we train on a new task, the performance on the old one initially decreases. After the training, the EWC is eventually minimized, which restores performance on the old task. This effect can also be seen (a limited extent) in Figure3(a). As we are log the accuracy once every 5 rounds, these variations appear as short spikes.

An intriguing effect is the rise in accuracy for a new task. In Figure5(c), task 15 and 20 performed better than task 5. This is unlike the behavior in the MNIST experiments, where the model was steadily saturated with decreasing accuracies on new tasks. In fact, there were 9 other instances when the newly introduced task performed better than some previous tasks. We attribute this to the fact that tasks in this setting are not truly independent. Being able to classify 10 identities . With the permuted MNIST, we had much greater control of the tasks, and could make them different from each other. This would force the model to use new weights for different tasks. However, if a model can classify faces from one set of 10 identities, there is a significant chance that it performs well on another set as well. This is particularly true here as we are using a fixed feature extractor and the model is only finding a separating hyperplane. This plane may work well for multiple sets of people. This also supports the hypothesis that the scheme to set λ does not cause blindly allocating new weights for every task (NOTE: An experiment to confirm this would involve measuring the task similarity and correlating this with the rise in the validation curves. This is not reported as the implementation is currently incomplete).

4. Experiment Details

Experiments were run using small models, with up to 2 hidden layers of varying sizes with ReLU units. All training was done with the Adam optimizer in TensorFlow.

We used a random hyperparameter search scheme. This was used to find the values for the batch size, number of epochs and learning rate. To reduce the size of the search space, we used standard values for hyperparameters which are known to work well: we use a dropout of 0.5, and fixed values $\epsilon = 1e-8$ $\beta_1 = 0.9$ and $\beta_2 = 0.999$ for the Adam optimizer (default in TensorFlow). The search scheme involved trying 60 random combinations of values and selecting the one which minimized the sum of the task loss and EWC loss.

Hyperparameter	Value
Learning rate	$1e-3$
Batch size	100
Dropout	0.5
Epochs	200
EWC scaling (λ)	400

Table 1. Hyperparameters for permuted MNIST experiments

4.1. MNIST Experiments

The model used for permuted MNIST experiments had 2 layers, 100 units each. We first used the hyperparameter search every time the model is trained on a new task. On doing this, we observed that the search returned similar hyperparameter values, across tasks. Hence, we use an approximation of these values. This yielded similar performance to using the search for every new task. These values are shown in table 1. The images were normalized to the range $[0, 1]$ and flattened to $\mathbb{R}^{784}$ for training.

4.2. Face Recognition Experiment

The model in the face recognition experiment used 2 layers with 400 units each. The images we used were the cropped and aligned faces from the MSCeleb dataset. Before training, we subtract the channel-wise mean to normalize the images. A fixed EWC scaling of $4e6$ was used for all tasks. Unlike the MNIST experiment, we were unable to find fixed hyperparameter values which worked across all tasks. We suspect this is due to the varying complexity of the face classification task. Certain sets of faces may be easier to classify, and the model converges quickly for those tasks. It was also noted the convergence time (number of epochs from the hyperparameter search) increased as more tasks were learned up till the first K tasks (all of which must be remembered). After this, the convergence time varied, depending on the simplicity of the new task or its similarity with previous tasks (this is just the variation in learning the new task from the current parameter setting).

5. Future Work

The problems described here show relatively simple forget policies. The MNIST experiments essentially used an ad-hoc policy to demonstrate forgetting, while the face recognition model made many simplifying assumptions to identify exactly which tasks can be forgotten. This is not realistic. We would like the model to identify which tasks are important on its own. One possible way to do this would be to use another model which scans the history and outputs an attention vector indicating useful tasks.

6. Conclusion

In this project, we introduced the idea selective forgetting. We showed a method to implement it with a simple modification to a known algorithm (EWC). Its properties were analyzed with experiments on permuted MNIST tasks. We also showed its application to save capacity in a more complex problem: continual face recognition. This used a model to continually classify faces from the MSCeleb dataset. From the results and benefits observed, selective forgetting seems to be an interesting feature and a useful component in an efficient continual learning agent.

References

[1] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

[2] R. M. French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[3] A. Gepperth and C. Karaoguz. A bio-inspired incremental learning architecture for applied perceptual problems. *Cognitive Computation*, 8(5):924–934, 2016.

[4] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

[5] Y. Guo, L. Zhang, Y. Hu, X. He, and J. Gao. MS-Celeb-1M: A dataset and benchmark for large scale face recognition. In *European Conference on Computer Vision*. Springer, 2016.

[6] G. E. Hinton and D. C. Plaut. Using fast weights to deblur old memories. In *Proceedings of the ninth annual conference of the Cognitive Science Society*, pages 177–186, 1987.

[7] N. Kamra, U. Gupta, and Y. Liu. Deep generative dual memory network for continual learning. *arXiv preprint arXiv:1710.10368*, 2017.

[8] R. Kemker, A. Abitino, M. McClure, and C. Kanan. Measuring catastrophic forgetting in neural networks. *arXiv preprint arXiv:1708.02072*, 2017.

[9] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

[10] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[11] Z. Li and D. Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[12] M. McCloskey and N. J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[13] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter. Continual lifelong learning with neural networks: A review. *arXiv preprint arXiv:1802.07569*, 2018.

[14] O. M. Parkhi, A. Vedaldi, A. Zisserman, et al. Deep face recognition. In *BMVC*, volume 1, page 6, 2015.

[15] D. C. Plaut. Relearning after damage in connectionist networks: Toward a theory of rehabilitation. *Brain and language*, 52(1):25–82, 1996.

[16] S.-A. Rebuffi, A. Kolesnikov, and C. H. Lampert. icarl: Incremental classifier and representation learning. In *Proc. CVPR*, 2017.

[17] A. Robins. Consolidation in neural networks and in the sleeping brain. *Connection Science*, 8(2):259–276, 1996.

[18] A. Seff, A. Beatson, D. Suo, and H. Liu. Continual learning in generative adversarial nets. *arXiv preprint arXiv:1705.08395*, 2017.

[19] H. Shin, J. K. Lee, J. Kim, and J. Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, pages 2994–3003, 2017.

[20] F. Zenke, B. Poole, and S. Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pages 3987–3995, 2017.