GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Fakultät für Physik
Friedrich-Hund-Platz 1
37077 Göttingen

Applying the cascade model for memory consolidation to Deep Networks for Continual Learning

Francesco Negri
Tutor: David Kappel
Professor: Christian Tetzlaff
SoSe 2022

Abstract

Continual Learning has been an important limitation to machine learning models since the very beginning of their development. In latest years some new promising approaches in deep learning have been proposed that could solve this problem in a seemingly reliable manner by adding a prior distribution to the weights of the network. In our study we try to use a different approach based on a computational model for synaptic consolidation with memory cascades which could serve as a valid and more general alternative to these solutions. This model involves the inclusion of additional parameters connected to the weights of the network slowing down their dynamics in order to retain memory about previous events but without preventing the network to store new information. However, we show that this method does not work well when applied onto Deep Learning architectures when applied to the split MNIST and shuffled MNIST datasets, and we hypothesize that this is due to two main factors, the strict linearity of the equations involved in the memory cascades, and the fact that the assumption of uncorrelated memories is not applicable in the consistent training procedures of this field.

ERRATUM

In reviewing the previous iteration of our work on the cascade model[9], we encountered a problem in our implementation using the TensorFlow software package[11] in which the the automatically computed gradients of the loss function given in equation 4 were not calculated correctly, thus invalidating our previous results. In order to avoid this problem, we manually implemented the gradients and verified with a toy model that they corresponded to the expected values before running any simulation. With this premise, this report aims to correct these mistakes and make a more rigorous analysis of the cascade model applied to continual learning.

1 Introduction - Continual Learning

The term *Continual Learning* has been attributed to the ability of intelligent systems to keep learning new tasks but at the same time retaining their knowledge regarding previously learned ones, thus being able to maintain a high level of performance on multiple tasks which were learned at different points in time. This ability is of utmost importance in living beings as it allows them to adapt to their environment while keeping memory about past experiences which might prove useful for future events. In the field of machine learning, though, continual learning has been a challenge which is still far from being overcome. Forgetting is a naturally occurring phenomenon caused by the interference with newly learned tasks, but in biological organisms it generally happens gradually. Instead, in connectionist approaches such as artificial neural networks, due to the intrinsic stochasticity of the learning algorithms what commonly occurs when trying to learn multiple tasks sequentially is that even though we can achieve high performances on the latest task, we generally forget most of what we had learned before almost immediately, thus reducing the performance on all previous tasks in an abrupt way. This phenomenon is known as *catastrophic forgetting*[1], and it can be tied to the fact that generally, when learning new tasks, machine learning algorithms do not contain any hard-wired mechanism intended to retain memory about past events, but instead try to maximize performance regardless of how much the parameters of our model need to be changed - which can be translated into the amount of information about the past that gets erased. Until a few years ago, the most common models used for the purpose of continually learning machines have used methods that try to get around the problem, without really solving it: memorizing parts of previously learned datasets and showing these to the network periodically to avoid forgetting[3]. This method is obviously not optimal, as both the size of the memories stored and the length of training can increase fast as the number of tasks increases. Instead, more recently, some new approaches seem to have solved, up to a certain point, this problem by the introduction of an additional prior distribution for the weights which takes into account previously learned tasks[6]. Their approach shows much promise, but still requires the intervention of an external observer to let the machine know when a new task is being introduced. In our model we follow a very similar method but avoiding influence from external factors and no knowledge about the occurrence of a task switch.

2 Methods

2.A The cascade model for memory consolidation

The solution we propose is based on a mathematical model of memory consolidation developed by Benna and Fusi [2],[5]. This model is inspired by biological mechanisms of synaptic consolidation in mammalian brains, in which the strengthening of synaptic connections is influenced by cascade reactions which act in a feedback loop and are sequentially linked to each other as shown schematically in figure 2. From figure 2 we can also extrapolate the overall behavior of the cascade model: the weight of the synapse w interacts with hidden cascade weights u_i, and the system $(w, u_1, ..., u_N)$ is what constitutes the mechanism of memory consolidation and retention. These interactions are described by the equations that determine their dynamics, which are shared among all the different levels in the cascade, shown in equation 1:

$$C_k \frac{\mathrm{d}u_k}{\mathrm{d}t} = g_{k-1,k}(u_{k-1} - u_k) + g_{k,k+1}(u_{k+1} - u_k), \tag{1}$$

where C_k and $g_{k,j}$ are hyperparameters which determine the timescale of the dynamics, while the parameters u_k are the quantities describing each cascade level k. The synaptic weight corresponds to the zero-th cascade level u_0. Following the usual notation convention for Deep Networks, we will refer to the network weights corresponding to u_0 simply as w.

In this model, the synaptic weight w is the only parameter which is changed directly during training, and corresponds to the outermost level ($k = 0$). On the other hand, the dynamics of the inner levels is exclusively determined by equation 1. The innermost level, in the iteration of the model developed by Benna and Fusi is a leaking variable, meaning that the long term attractor for this variable is zero, due to the fact that, given N cascade levels, then $u_{N+1} = 0$. In general, in biological systems, biochemical interactions do not simply stack in a line as shown here, but the cascade model does not require the reactions to take a specific configuration, and the links in these chains of interactions can be reorganized in more complex shapes (with a consequent adaptation of the equations to accommodate additional interactions), forming more realistic representations of biochemical cascades observed in nature. Lastly, in their paper, Benna and Fusi discretized the values that their parameters can adopt, which in their view is justified by the fact that it would be unrealistic to expect these variables to vary continuously, with infinite precision.

The conceptual idea of this model, applied to machine learning, can be summarized as follows: whenever a parameter change happens during training, it will be mediated by its interactions with the parameters in the cascade levels, which will generally tend to drive the whole system to the equilibrium state -in which all the cascade weights share the same value, and memory of previous events is lost. This implies that any change gets distributed (within a timescale determined by the hyperparameters C_k and g_k) among all levels, and thus the induced parameter change has a lasting effect on all cascade levels, which decays as the system equilibrates. A study of the timescale of this equilibration process has already been made in the article by Benna and Fusi[5]. Following their evaluation, this timescale should be much longer than any timescale relevant for training in order to effectively retain past memories. The idea behind this is that the weights associated with synapses which are important for the task will be kept at a relatively constant value which minimizes the loss. This in turn propagates into the inner levels of the cascade, driving them towards equilibrium, and imposing a specific configuration to the cascade weights. This configuration is dynamic, but on shorter timescales, small perturbations in the outermost level (like the ones expected during training of an unrelated task) will be countered by the cascade weights. This counteracting force will be stronger for synapses which are important for the performance on a given task, as the cascade parameters associated to these synapses will have been driven more strongly out of equilibrium during training than those of irrelevant synapses, which in general will have experienced more random stimulations. The resulting effect is a force opposing changes in important synapses, implying a longer retention of the encoded memories, but still allowing "unused" synapses to undergo plastic changes and encode new memories until the whole capacity of the network is filled. Following this discussion it seems important to highlight two relevant facts:

- Due to the interactions with the cascade, which act in an opposing way to parameter changes, learning will be inevitably slowed down. The magnitude of this effect is shown in figure 1, where we show how the cascade parameters influence the dynamics of weight changes.
- Following our reasoning, memory retention will be dependent on training time T, and even if the resulting accuracy would not increase by prolonged training, the state of the cascade could benefit from longer

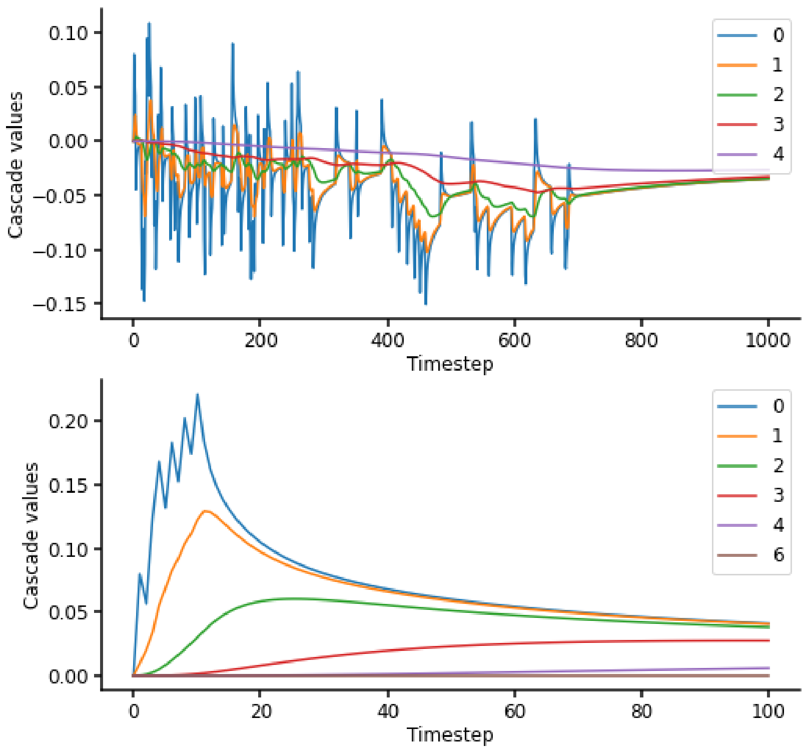

Figure 1: Examples of the dynamics of a system implementing the cascade of interactions described by equation 1. On the top: A stimulation procedure in which positive and negative stimuli are happening with the same frequency. Here we can observe a sort of oscillatory behavior in the system induced by the stimuli. We can notice how as we go further in the cascade levels, these fluctuations are smoothed out and the dynamics is significantly slowed down. On the bottom: A stimulation consisting of only positive stimuli (this case is particularly relevant in the case of Deep Learning, as discussed in the following sections). Here we can see how overall the behavior of the system is to induce a slow decay almost all the way back to the original configuration of the cascade parameters.

periods of stimulation in order to settle into a parameter space amenable for the current task. This implies that different training procedures could lead to very different results, as the state of the cascade can greatly influence task performance. In our simulations, we tried various approaches in order to optimize this effect, see section 3.

2.B Datasets

During our simulations, we tested two different datasets, split MNIST and shuffled MNIST. Split MNIST consists in splitting the MNIST categories in 5 different tasks containing 2 categories each. In our case the categories were [0,1],[2,3],[4,5],[6,7] and [8,9]. For the shuffled MNIST dataset, we performed a random shuffling with fixed indexing to the pixels of each image in order to obtain new datasets[7]. This way it is possible to obtain a virtually unlimited number of different datasets. In our study we used 5 of these shuffled datasets, in order to obtain 10 separate tasks. Some examples are shown in figure 3.

2.C Architecture and Learning algorithm

Following the approach taken in [4] and [6], we applied our model to networks with the same architecture adopted by them, so that it would be possible to compare our results to theirs. For the split MNIST dataset,

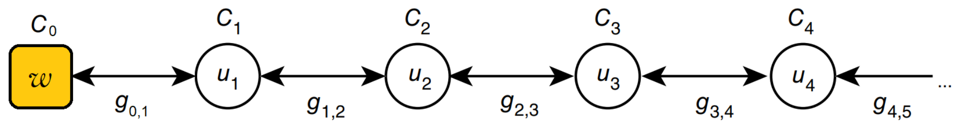

Figure 2: Graphical representation of the cascade levels in the cascade model. Each level interacts only with its two neighbors through the parameter $g_{k,j}$, and its value is mediated by the parameter C_k. A change in the first level will be transmitted to the inner levels only through these neighbor-neighbor interactions. The weights of the network w are the outermost level, here represented in yellow, and these are the only parameters which are actively changed by the learning algorithm, while every other level is only affected indirectly through the interactions described in equation 1

we used a two hidden layers MLP with 256 units per layer using the ReLU activation function. As described by [4], one problem with using multiple tasks in a sequential way is that most of the output neurons would be constantly set to zero, unless they are involved in the current task, and this would lead to the network learning to turn these outputs off and it would thus prevent it from learning more than one task at a time. In order to avoid this complication we used a multi-head approach in which the loss at the output layer was computed only for the digits present in the current task. For the shuffled MNIST, due to the increased complexity of the task (each task contains 10 categories), we increased the number of units per layer to 2000. Additionally, in order to implement the cascade model in our networks, we connected each weight of the network to a set of coupled parameters which will be updated after each weight modification according to the dynamics described in equation 1.

The inclusion of these parameters can also be described in a Bayesian inference framework as a hierarchical bayesian model of interacting prior distributions. The proof was previously shown in [9]. As a consequence, the loss function for the learning algorithm can be modified to include this regularizing prior, which is reflected in the expression for the posterior distribution shown in equation 2.

$$\log P(w|D) = \log P(D|w) + \log P(w) \tag{2}$$

The objective then becomes to maximize the log posterior distribution $\log P(w|D)$ of the weights of the network w given the data D by maximizing the sum of the likelihood and the prior, or equivalently minimizing the losses associated with these two terms. In general, the prior term to be Gaussian distributed corresponds to minimizing a squared differences loss, which, when differentiated with respect to the parameters of the network, would lead to the term shown in equation 3:

$$\Delta w_{prior} = g_{0,1}(u_1 - w) \quad . \tag{3}$$

where w is the weight of a generic synapse and u_1 would represent the mean of the Gaussian distribution. Interestingly, this expression coincides with the equation for the first level of the cascade in equation 1, so that following this Bayesian approach with a generic Gaussian prior we are changing the weights according to the cascade dynamics as we required. The resulting loss function is shown in equation 4, where the symbols $\langle\cdot\rangle$ stand for an average over the weights of the network, and $c \in [0, 1]$. As is often done with composite loss functions, we also introduced a hyperparameter c which allows us to modulate the importance of the cascade-related loss to the purely data-driven loss during our simulations. This gives us an additional way to control the efficacy of our model, while maintaining it compatible with the pure Bayesian model discussed above, since it can be viewed as a normalization constant.

$$L = (1-c) \sum_{k \in \text{output}} t_k \log(y_k) + c \left\langle g_{0,1}(u_1 - w)^2 \right\rangle \ , \tag{4}$$

3 Results

3.A Preliminary Note - Adam vs SGD

One initial note on some of the results shown by previous studies on catastrophic forgetting on the two datasets used in our study regards the choice of the optimizer used during training. A generally common choice for the

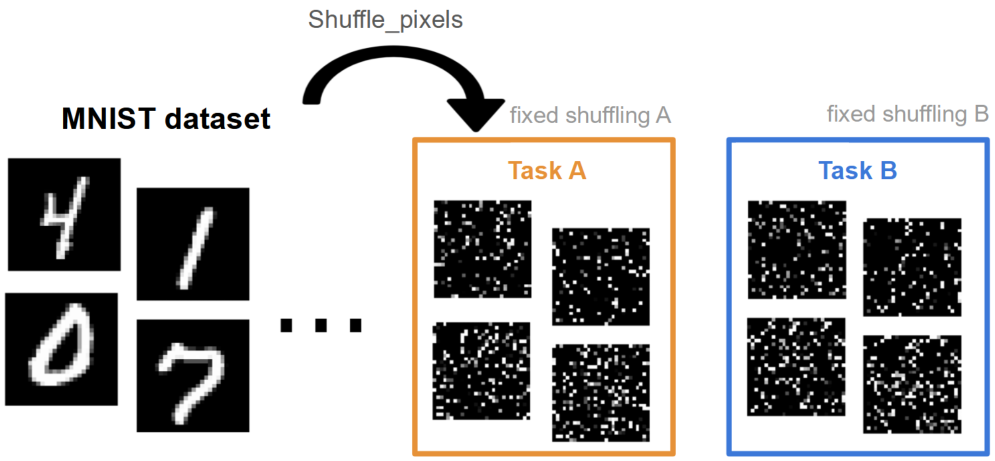

Figure 3: Representation of the operations involved in the creation of the tasks for the shuffled MNIST dataset. Each task is obtained by performed a permutation of the pixel by using a fixed indexing in order to keep the permutation consistent across the whole dataset. Different permutations will thus give rise to different tasks.

optimizer in Deep Learning is the Adam (Adaptive Moment Estimation) optimizer, due to its high efficacy in finding a good landscape in parameter space and significantly decrease the time needed for training[8]. The basic mechanism implemented by Adam is that of making the path towards the minimum of the loss smoother by renormalizing the learning rate by the variance of the gradient, and instead of always strictly moving in the direction of the gradient, to average its direction by computing an exponential moving average. These characteristics are particularly amenable for speeding up the learning process, but in the case of Continual Learning, this seems to worsen the effects of catastrophic forgetting. In order to show this effect, in figure 4 we compare the testing accuracy obtained on the split MNIST dataset while training with Adam or with simple gradient descent. Due to this, in our experiments we only used the SGD optimizer, as in both the naive bayesian model and the cascade model this resulted in less catastrophic forgetting.

3.B Base implementation

In our first attempts, we simply implemented the memory cascades as described by Benna and Fusi[5], using the same parameter values, and determining the optimal number of cascade levels by using the method described in their paper. We then proceeded to train the network, and validating it on the test set. Regardless of the parameters chosen for our simulations, we found it impossible to reach the baseline accuracies obtained with a simple MLP without regularization. The results compared with simple SGD for the split MNIST dataset are shown in figure 6. In order to understand why the cascade seemed to promote catastrophic forgetting instead of preventing it, we looked at the evolution of a subset of the weights of the network during training. In figure 5 we show one example from each layer in the network. From this figure it becomes apparent that the effect of the cascade is not exactly what we expected: instead of adapting to the new, specific weight configuration which allows the network to perform well on a new task, it generally pulls the weights back to their original configuration, thus rendering training ineffective. This seems to be mostly caused by the fact that the stimuli which modify the weights are not entirely uncorrelated and distributed around zero, but are instead highly biased towards one specific direction (either potentiation or depression). By observing these plots we can also see one possible reason for this problematic behavior: during training, weights are not being constantly stimulated, but instead are inactive during most of the procedure, and except rare cases, they receive only very sparsified stimulations. These two considerations, from the perspective of the cascade model are not auspicable, because due to its intrinsically slower dynamics, the cascade of parameters associated with the weights, will not be able to reach a configuration in which the parameters will be stable against perturbations as we hoped, but will instead reach an intermediate value which is not optimal for performance on the previously learned tasks. This is clearly shown in the accuracy plot, figure 6, where if we look at the accuracy on the first task, it seems to be constantly degrading whereas in the case of SGD it stayed close to its maximum value for the whole simulation. Another unrelated but relevant general trend which can be observed from figure 7 is that the

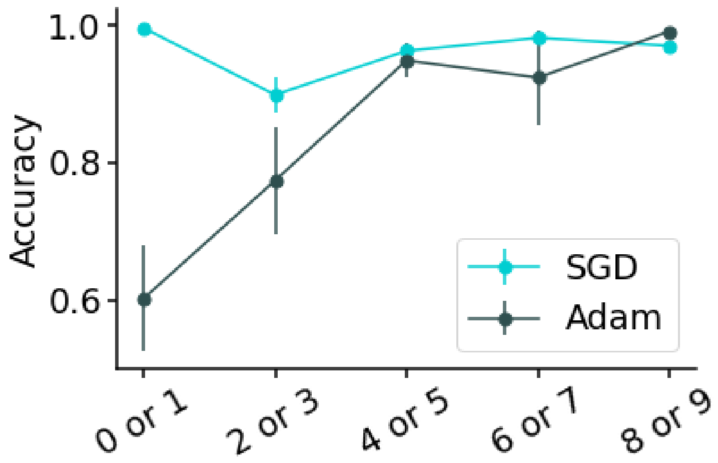

Figure 4: Testing accuracy on each of the tasks for the split MNIST datasetd after completing the training procedure. The Adam optimizer, used with the same configuration from Zenke et al[4], shows much higher signs of forgetting than simple SGD.

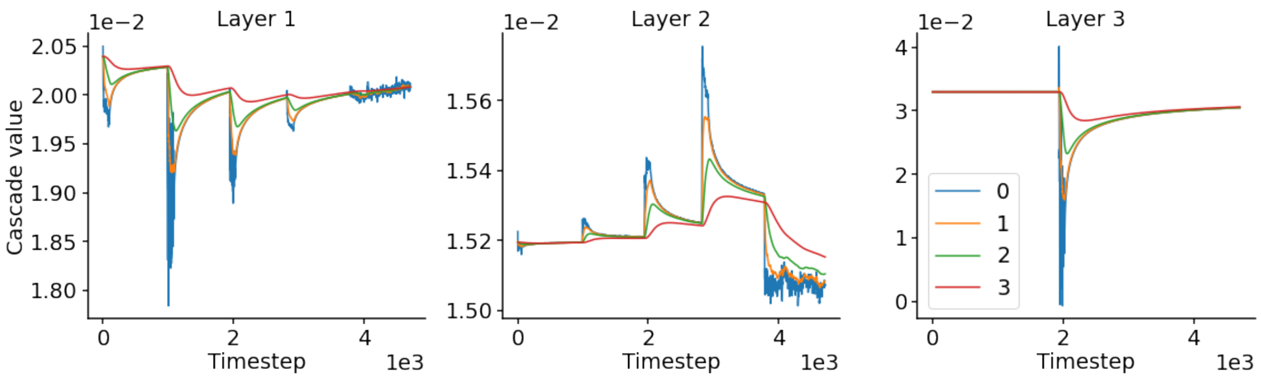

Figure 5: Time evolution of one parameter per each layer of our network when implementing the base mechanism of the cascade model. We can see how the cascade parameters slowly react to the stimulations induced by training and adapt to the new state of the weight, while at the same time pulling the weight itself back towards its initial state. Here we only show 3 levels of the cascade in order to avoid cluttering the plot, but the additional parameters will be simply undergoing slower dynamics as shown in figure 1

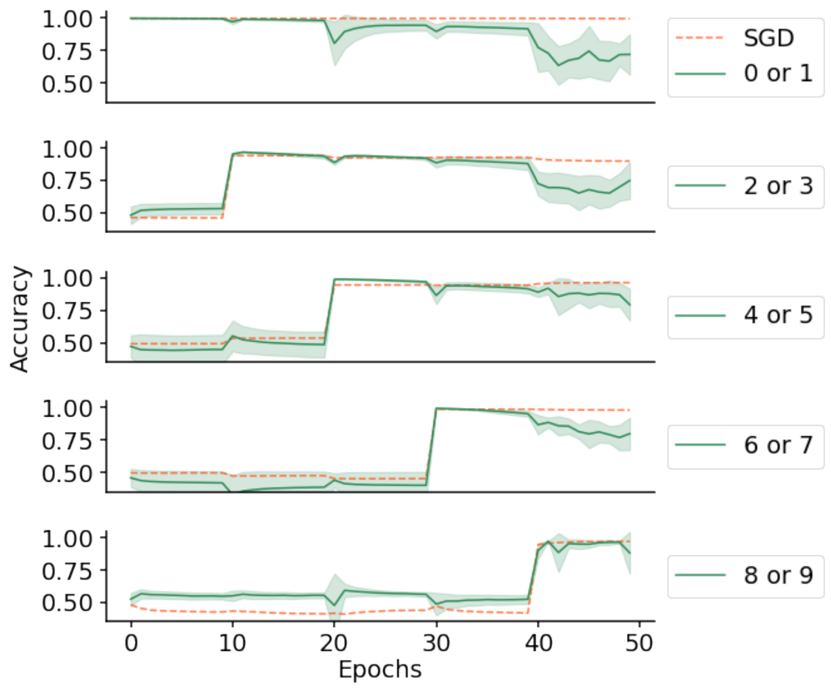

Figure 6: Accuracy on each of the tasks from the split MNIST dataset measured during training for the base implementation of the cascade. The dashed line is the mean accuracy obtained with simple SGD for comparison. We can clearly see that whereas the accuracy in the SGD network remains mostly stable after learning a task, the cascade model seems to induce faster forgetting and there is a consistent decay in performance as training continues.

amount of stimulation received by a weight decreases as a function of the depth, especially in the last layer of the network. This is compatible with the idea that the deeper layers of a network are generally used to learn more abstract representations, which in theory could remain relevant even across different tasks[10]. Specifically, if we look at a generic weight during training from any of the iterations of our model, shown in figure 9, we can see how the weights to the output layer seem to be significantly modified only in one or two separate instances during training.

From these considerations, we proceeded to slightly modify the training procedure in order to try to counteract these phenomena and hopefully achieve better performance.

3.C Consolidation I - Per Epoch

The first approach we followed was the idea that the cascade parameters should be allowed some time to consolidate the changes imposed onto them, so that they could stabilize around an optimal value. With this in mind, we stopped training at the end of each epoch and allowed the parameters of our network to evolve only due to the dynamics introduced by the cascade parameters. This way, the perturbations introduced during each epoch can be transferred into the deeper levels of the cascade and thus ideally set a memory state which is more amenable for our experiments.

This idea, as shown in figure 10 did not prove useful, and instead not only worsened the performance of our model and the effects of catastrophic forgetting, but seemed to prevent the network from even learning a single task efficiently. The problem with this approach lies within two fundamental misconceptions in our reasoning:

- We thought that if a parameter is particularly relevant to the performance of a task, the training procedure would tendentially set this parameter to a specific value. This would mean that if the cascade needs more time to settle into a state in which this parameter is stable against perturbation, then the training procedure would keep systematically pushing its value towards that particular state. The problem with this idea is that due to the extremely high dimensionality of our system, and due to the fact that the cascade itself tends to effectively move our network through this parameter space, then the state at which

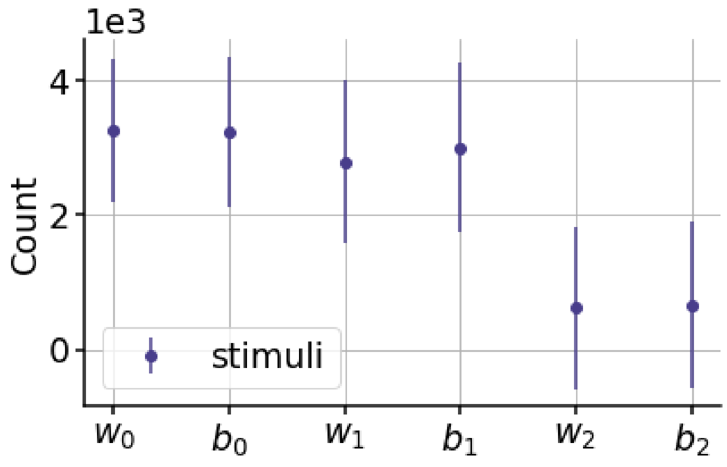

Figure 7: Average number of stimuli received by the parameters of the network during training as a function of the depth in the network. w stands for a weight and b for its bias, and the indices indicate the layer they belong to. We can see that the amount of stimulation each parameter is subject to decreases with the depth it belongs to, and this effect is especially significant in the last layer of the network.

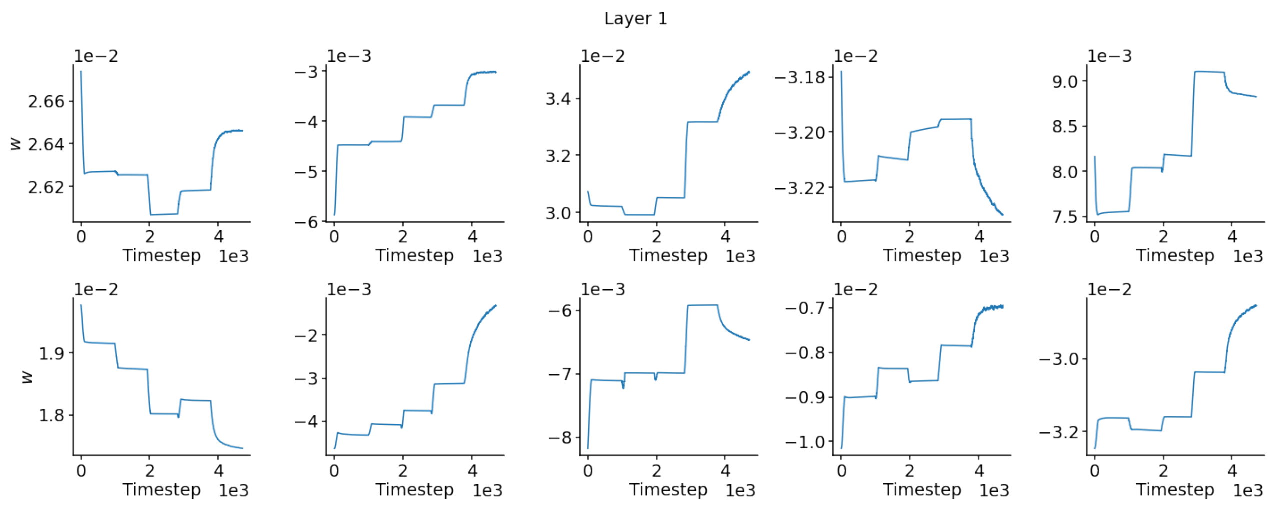

Figure 8: Time evolution of a randomly sampled set of 10 parameters selected from the parameters of our network during training. It appears that the parameters only receive short bursts of plasticity stimuli instead of consistent stimulations. Furthermore, we can see that this activity bursts seem to happen mostly once per each task (in our case there are 5 in total). This could imply that the rest of the time spent learning that task might be influencing other parameters.

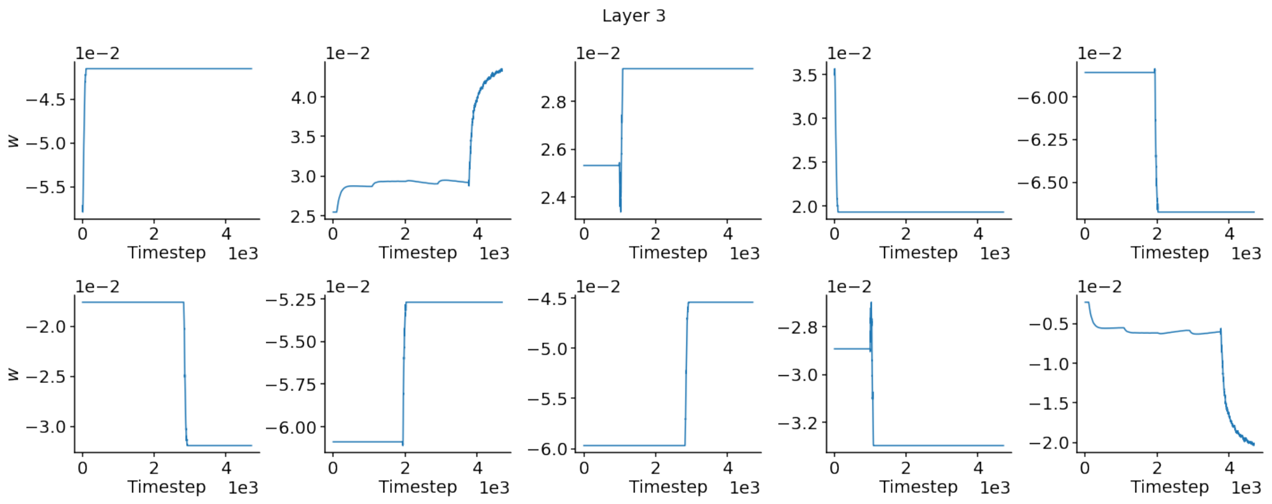

Figure 9: Time evolution of a random set of parameters selected from the last layer of our network during training. We can see that in this case most of the weights experience intense stimulation only once or sometimes twice over the entire training procedure.

the network will be after evolving the cascade might require, in order to minimize the loss, *other* parameter changes which are entirely different than before. For this reason, inducing consolidation at the end of each epoch is detrimental to the performance of the network, since we are altering its entire information content and destroying previously learned features.

- Compatible with the first point, a direct observation which we can make from looking at the dynamics of the parameters of the network during training is that most of the time these parameters mostly receive one single burst of stimulation for each task (see figure 8), and then remain almost unchanged, at least until the next task is introduced. So, introducing consolidation *after* this burst has already happened will not be useful, as the effect of training on this particular parameter for this particular task will most likely be over already, and consolidation will only bring this parameter away from its optimized state.

3.D Consolidation II - Per Batch

Following the second point from the previous section, we tried to elaborate on the idea of introducing consolidation during these bursts of activity that happen during training, so we introduced the concept of consolidation of the cascade after each batch is shown to the network during training. The idea is very similar to the one introduced in the previous experiment, but this time we consider the effect that each step of the training procedure has on each of the parameters, so that effectively we are slowing down the pace of learning in order to allow the cascade to have more time to process information and pass it along to the slower components.

The results from this approach (shown in figure 11) seem to make up for the pitfalls of the previous two methods, and the training procedure seems to be more similar to that obtained during simple SGD. Nonetheless, if we look at the final accuracy of the network on each of the tasks from figure 12, we can see that this method, while slightly better than the simple implementation of stochastic gradient descent with no regularization on the second task, is not significantly better than it, meaning that the introduction of the cascade regularization does not seem to mark a significant improvement on performance as we expected. Nonetheless, we still proceeded to run one last simulation with an additional intuition about the training procedure's effect on the network parameters.

3.E Remove cascade mechanism in first layer

The intuition which came from our previous observations, and especially from figure 7 is that the parameters in the first layers of our network seem to be the most dynamic ones during training, reasonably so due to their direct interaction with the different outputs presented from each task. For this reason, we tried to exclude the

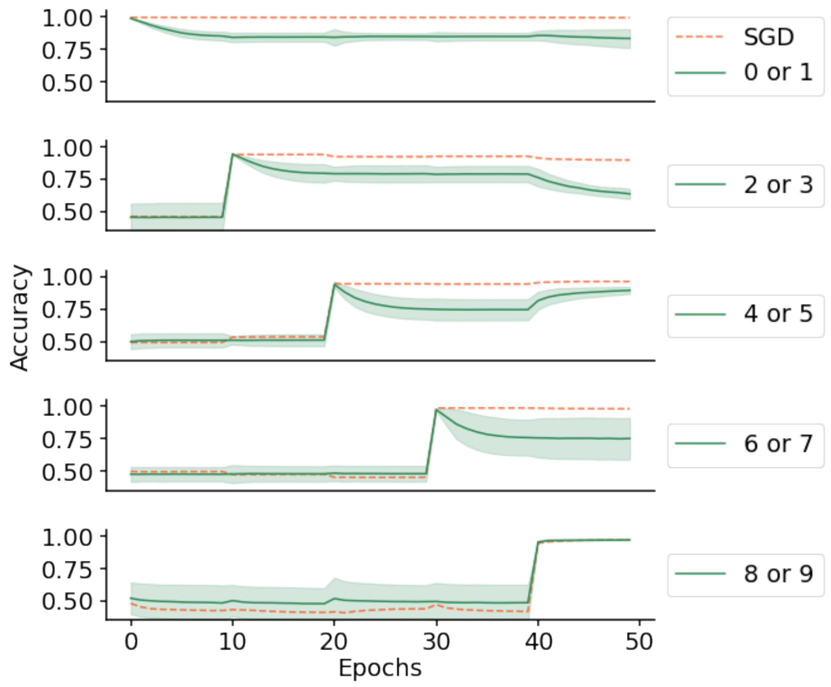

Figure 10: Time evolution of the accuracy of our network during training when using the consolidation technique (only evolving the cascade parameters without any input shown to the network) at the end of each epoch. We can see that this method seems to actually significantly worsen the performance on each of the tasks, as the accuracy decays to a lower value in all cases. The dotted line represents the comparison with the network trained using only SGD.

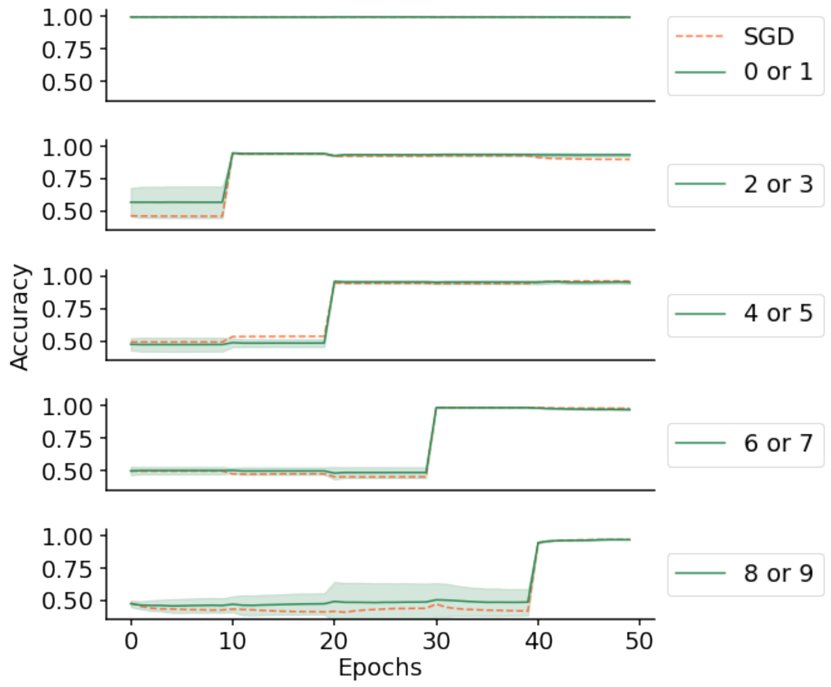

Figure 11: Time evolution of the accuracy of the network on each of the tasks in split MNIST during training for the training procedure using the batch consolidation method (BC). We can see that in this case the network seems to follow more closely the behavior observed with the standard SGD optimization.

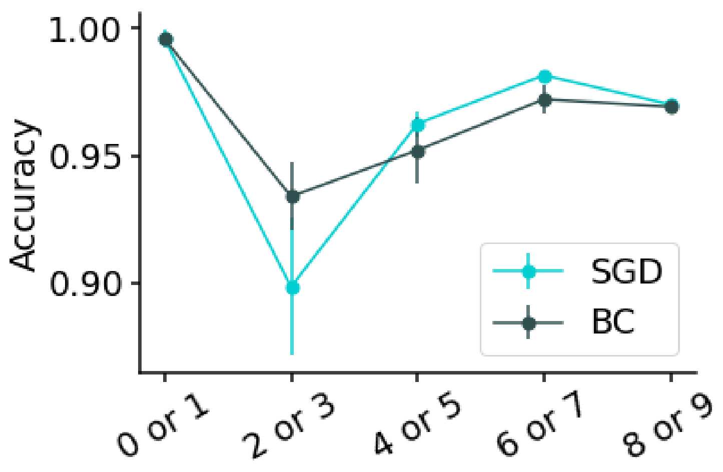

Figure 12: Accuracy on the split MNIST tasks for the training procedure in which we allow consolidation of the cascade after each batch is presented to the network (referred to as batch consolidation, or BC). Even though performance is significantly improved with respect to the previous iterations, the simple SGD model is performing at least on par with our model on all except one of the tasks.

first layer of our network from the cascade dynamics, while keeping the cascade active in the second and third layers. For the cascade, we kept in place the mechanism of batch consolidation used in the previous section, as it proved to be the better consolidation method for our purposes.

This approach was the first one to prove significantly better than the simple implementation of SGD, as shown in figure 13, where we show the final accuracies on each task.

Using this positive result, we moved on to compare the performance of this approach on the shuffled MNIST dataset, which we did not test on the previous iterations of the model since it is reasonable to expect them to perform worse due to their poor performance on the simpler split MNIST. We thus trained two networks on the shuffled MNIST dataset, one using simple stochastic gradient descent, and the other using the cascade regularization with batch consolidation applied to the last two layers.

Unfortunately, as shown in figure 14 even though the results were comparable with each other, our model still underperformed compared to the simpler and faster unregularized network.

4 Conclusions

In our experiments, we tested the performance of the cascade model presented by Benna and Fusi in [5] on Continual Learning problems in the framework of Deep Learning. From our experiments, it appears that the cascade model in itself cannot outperform the simpler maximum-likelihood approach of stochastic gradient descent. The reason for this seems to lie in the linearity of the chain of parameter interactions which characterize the cascade model and the fact that in Deep Learning, two of the fundamental assumptions of the original cascade model are not satisfied: the memories stored in the system are not random and uncorrelated, and the degree of potentiation and depression is strongly unbalanced towards one or the other for each parameter (in particular for this effect, see figure 15. These two factors in the original paper implied that the value of a synaptic weight would be oscillating around a mean value (usually set to zero) due to these random and balanced external stimulations, and these oscillations would be broadcasted to the inner levels of the cascade, which would in turn slow down their dynamics and reduce the effect of each single stimulation by smoothing their effect this can be somewhat observed in figure 1, where we see that the inner levels, which effectively constitute an attractor for the outer parameter, are much less affected by the effects of this noise-like perturbations. Since the assumption of balanced potentiation or depression is not satisfied in the Deep Learning paradigm, this fundamental aspect of the model is not displayed by the models in our experiments, and instead a detrimental effect of the cascade is introduced, where the cascade effectively works as a strong l_2 regularization which forces the network parameters back to their original initialization state. Even by initializing the network with the cascade parameters set to

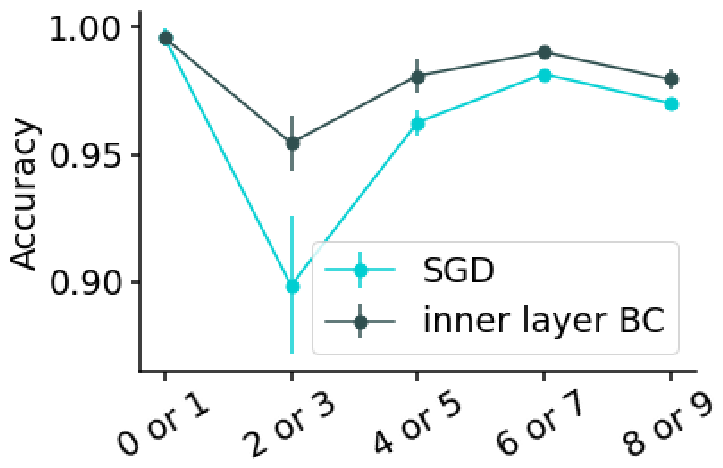

Figure 13: Accuracy on the split MNIST tasks obtained by the last iteration of our model, in which we only applied the cascade to the second and third (last) layers of our network. Furthermore we used the batch consolidation (BC) technique in order to improve performance. This time, our model seems to noticeably outperform the simple maximum likelihood optimization with SGD.

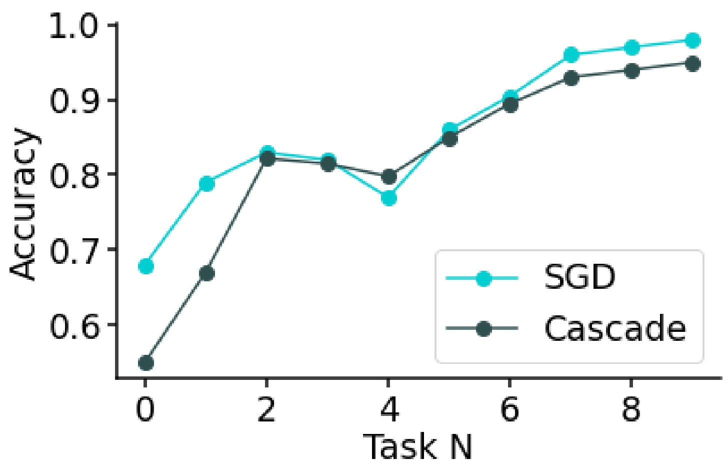

Figure 14: Final accuracy of the network on the shuffled MNIST tasks after completing the training procedure. Even though the performance on most of the tasks is comparable, the cascade model network seems to have undergone significantly more forgetting for the first two tasks. This simulation was performed with our best performing iteration of the cascade model, and marked the final proof that this model might not be applicable to continual learning without substantial modifications.

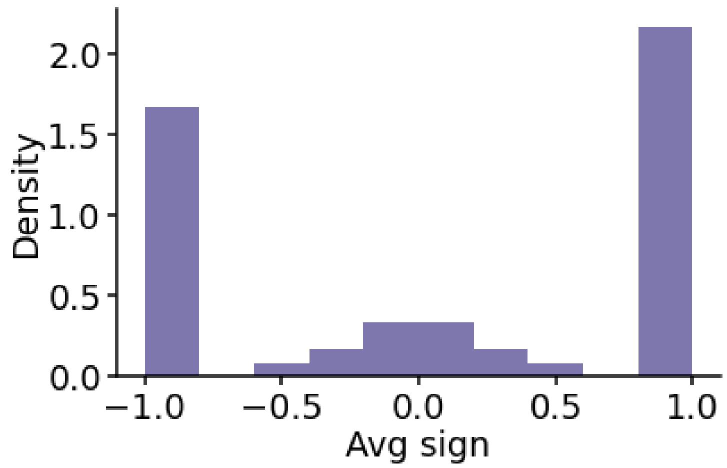

Figure 15: Distribution of the average sign of weight changes induced in the parameters of our network during training. We can clearly see that even though part of the distribution seems to be compatible with a gaussian centered around zero (which would imply a balance between depression and potentiation), we have two peaks at $+1$ and -1, which imply that a large portion of the parameters of our network receive almost only one type of stimulation, either depression or potentiation.

zero (instead of copying the initialization values of the parameters of the network), the effect of the cascade is detrimental to learning. In order to obtain a useful regularization we would need to slow down the dynamics of the cascade to a level in which we would essentially be implementing simple l_2 regularization and the inner levels of the cascade would barely have any effect. Introducing a consolidation effect on the cascade seemed to have a promising effect on performance, at least on the split MNIST dataset, where we could obtain a better performance than with the simple SGD optimization, but this did not translate well into the more complex tasks from the shuffled MNIST dataset.

Based on our observations on the distribution of the weight modifications during learning, we propose that one possible approach which might mitigate the detrimental effect of the cascade model to learning by restoring the validity of the assumptions made by Benna and Fusi would be to introduce background noise into the network, which would introduce consistent and controllable stimulations. Under these conditions the burst-like stimuli induced by the stochastich gradient descent optimization would represent only slight deviations from the balanced condition introduced by the background noise, and the system would hopefully follow the behavior observed in the original cascade model paper. The implementation of this alternative solution is out of the scope of this report, and will thus be left to future investigations.

References

[1] Michael McCloskey, Neal J. Cohen, Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem, Editor(s): Gordon H. Bower, Psychology of Learning and Motivation, Academic Press, Volume 24, 1989, Pages 109-165, ISSN 0079-7421, ISBN 9780125433242,

[2] Stefano Fusi, Patrick J. Drew, L.F. Abbott, Cascade Models of Synaptically Stored Memories, Neuron, Volume 45, Issue 4, 2005:599-611, ISSN 0896-6273.

[3] McClelland JL, McNaughton BL, O'Reilly RC (1995) Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory. Psychol Rev 102(3): 419–457

[4] Friedemann Zenke, Ben Poole, and Surya Ganguli. 2017. Continual learning through synaptic intelligence. Proceedings of the 34th International Conference on Machine Learning - Volume 70 (ICML'17). JMLR.org, 3987–3995.

[5] Benna MK, Fusi S. Computational principles of synaptic memory consolidation. Nat Neurosci. 2016 Dec;19(12):1697-1706. doi: 10.1038/nn.4401. Epub 2016 Oct 3. PMID: 27694992.

[6] J. Kirkpatrick, R. Pascanu, et al, Overcoming catastrophic forgetting in neural networks, Vol 114(13), 3521–3526, 2017, DOI 10.1073/pnas.1611835114, National Academy of Sciences, ISSN 0027-8424, Proceedings of the National Academy of Sciences

[7] https://deepai.org/dataset/mnist

[8] Kingma2015AdamAM,Adam: A Method for Stochastic Optimization,Diederik P. Kingma and Jimmy Ba,CoRR, 2015

[9] Continual Learning with Memory Cascades, David Kappel and Francesco Negri and Christian Tetzlaff,I (Still) Can't Believe It's Not Better! NeurIPS 2021 Workshop, 2021, https://openreview.net/forum?id=E1xIZf0E7qr

[10] Feature Visualization by Optimization, Olah et al.,2017, 10.23915/distill.00007

[11] TensorFlow software, https://www.tensorflow.org/