Continual Learning with Memory Cascades

David Kappel[*] **Franscesco Negri**[*] **and Christian Tetzlaff**
Bernstein Center for Computational Neuroscience
III Physikalisches Institut – Biophysik, Georg-August Universität, 37077 Göttingen, Germany
david.kappel@uni-goettingen.de

Abstract

Continual learning poses an important challenge to machine learning models. Kirkpatrick et al. introduced a model that combats forgetting during continual learning by using a Bayesian prior to transfer knowledge between task switches. This approach showed promising results but the algorithm was given access to the time points when tasks were switched. Using a model of stochastic learning dynamics we show that this model is very closely related to the previously developed cascade model to combat catastrophic forgetting. This general formulation allows us to use the model also for online learning where no knowledge about task switching times is given to the network. Also it allows us to use deeper hierarchies of Bayesian priors. We evaluate this model on the permuted MNIST task. We demonstrate improved task performance during task switching, but find that online learning is still significantly worse when task switching times are unknown to the network.

1 Introduction

The term *Continual Learning* has been attributed to the ability of intelligent systems to keep learning new tasks but at the same time retaining knowledge of previously learned ones, thus being able to maintain a high level of performance on multiple tasks which were learned at different points in time [1]. Forgetting is a naturally occurring phenomenon during continual learning caused by the interference with newly learned tasks. In connectionist approaches, such as artificial neural networks, high performances on the latest task is typically sacrificed to forgetting of previous ones in an abrupt way. This phenomenon is known as *catastrophic forgetting* [2], and it can be tied to the fact that in general, when learning new tasks, machine learning algorithms 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.

Kirkpatrick et al. addressed the continual learning problem by introducing an additional prior distribution for the network weights which takes into account previously learned tasks [3]. Their approach showed promising results, but still requires the intervention of an external observer to let the algorithm know when a new task is being introduced. The *cascade model* has been introduced based on a theoretical argument to optimally combat catastrophic forgetting [4]. This model assumes coupled internal variables in every synapse that evolve on slow time scales and allow the network to robustly memorize and maintain previously learnt information.

Here, we develop a generic theory that allows us to establish a link between the Bayesian model of Kirkpatrick et al. and the cascade model. We show that the latter can be understood as a hierarchical Bayesian model of interacting prior distributions. Our model does not require access to the task switching times as in Kirkpatrick et al. (but they can be included). We apply this model to the permuted MNIST task [5] and evaluate the resulting parameter dynamics compared to a conventional model without parameter priors.

[*]equal contribution

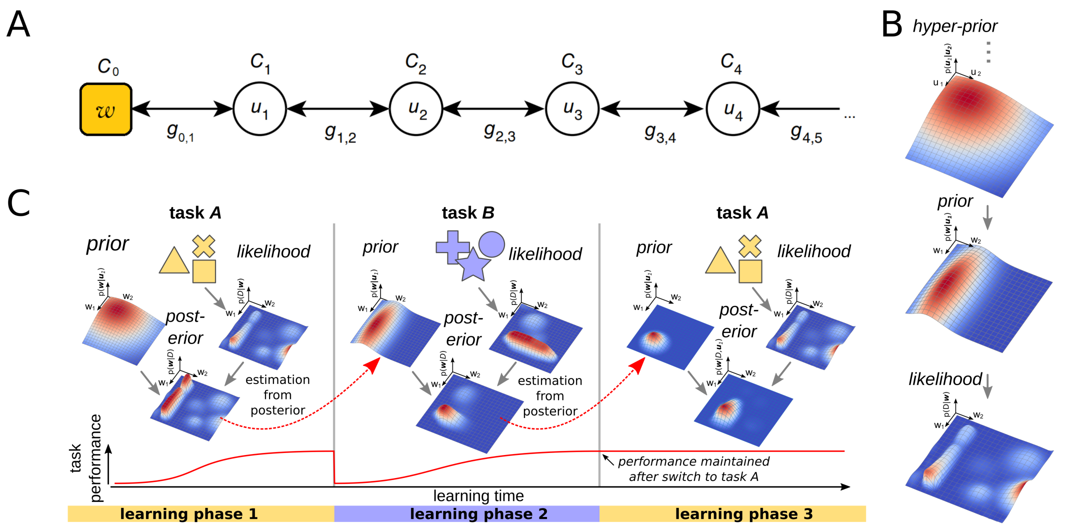

Figure 1: **Illustration of the parameter cascade.** **A:** Each level of the cascade interacts only with its neighbors through the parameter $g_{i,j}$, and mediated by the parameter C_i. The weights of the network w are the outermost level and get actively changed by the learning algorithm. Other level change through interactions with their neighbors (see Eq. (1)). **B:** Illustration of the implicit Bayesian hierarchy that is implied by the model (A). **C:** The prior distribution is used to integrate representations of multiple tasks and carry them throughout task switches (two tasks A and B shown).

1.1 Related work

A review on related work on continual learning and multitask learning can be found in [1] and [6]. Early models to tackle continual learning directly memorize parts of previously learned datasets and showed these to the network periodically to avoid forgetting (e.g. [7]). In contrast, our work follows a regularization approach similar to the one used by Kirkpatrick et al. [3], where a prior to quantify the change in the network from task to task is introduced, to reduce weight update speed for synapses which are most relevant for the performance on the learning problem. A related approach was used by Zenke et al. [5]. Li and Hoiem [8] used a knowledge distillation loss [9] to regularize the task specific parameters from old tasks while at the same time training a new set of parameters for the current task. This approach still requires a persistent memory of each task. Other models have used specialized network architectures [10], in which they freeze weights related to previous tasks; or dynamic architectures [11], by progressively adding sub-networks to be trained on new tasks.

2 Results

We revisit the mathematical model of synaptic memory consolidation developed by Benna and Fusi [12, 2]. This model is inspired by biological mechanisms of synaptic consolidation in mammalian brains, in which the strengthening of synaptic connections is influenced by a cascade of interacting bio-chemical mechanisms. These mechanisms act in a feedback loop and are sequentially linked to each other as shown schematically in Fig. 1. The weight w of a synapse interacts with hidden cascade variables u_k, and the overall system $(w, u_1, ..., u_N)$ constitutes the mechanism of memory consolidation and retention [12]. Linear dynamics describe the interaction between the cascade variables, i.e.

$$C_k \frac{\mathrm{d}u_k}{\mathrm{d}t} = g_{k-1,k}(u_{k-1} - u_k) + g_{k,k+1}(u_{k+1} - u_k), \quad \text{for} \quad 0 < k < N \,, \tag{1}$$

where C_k and $g_{k,j}$ are hyperparameters which determine the speed of the dynamics. In this model, the synaptic weight w is the only one which is changed directly during training, and corresponds to the outermost level of the cascade ($k = 0$). The synaptic weight dynamics are thus given by

$$C_0 \frac{\mathrm{d}w}{\mathrm{d}t} = \frac{\mathrm{d}}{\mathrm{d}w}\mathcal{L}(\mathbf{D}) + g_{0,1}(u_1 - w) \,, \tag{2}$$

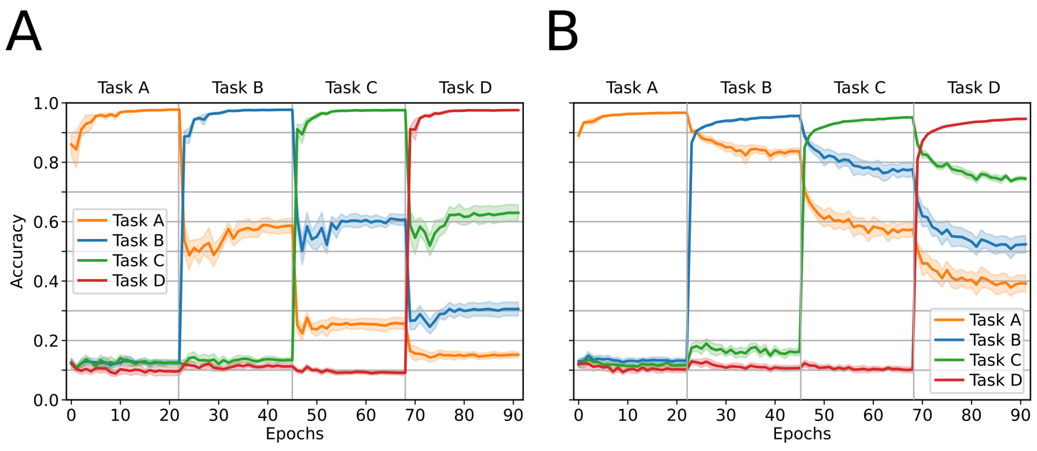

Figure 2: **Accuracy on multitask learning. A:** Accuracy of a network during learning of 4 consecutive tasks (A,B,C,D) without the cascade dynamics. The accuracy on the first task learned (Task A) reaches the level of nearly random guessing by the end of the experiment. **B:** As in (A) but including the parameter cascade. The performance is significantly increased but still declines strongly while learning subsequent tasks.

where C_0 plays here the role of a learning rate. $\mathcal{L}(D)$ is an arbitrary loss function that depends on the training data D.

It is instructive to note, that this model has a close relation to a hierarchical Bayesian model. In Appendix A.1 we show that the dynamics (1) optimize an implicit probabilistic loss function $\mathcal{L}_C(D)$

$$\mathcal{L}_C(D) = \log P(D|w) + \log P(w|u_1) + \log P(u_1|u_2) + \log P(u_2|u_3) + \dots\,, \tag{3}$$

where the likelihood term is defined through the data-dependent loss, i.e. $\log P(D|w) = \mathcal{L}(D)$. The cascade dynamics (1) give rise to a hierarchical Bayesian prior, where every term refines the one from the previous level. This implies that any change gets distributed (within a timescale determined by the hyperparameters C and g) among all levels of the cascade. Thus the induced parameter changes have a lasting effect on all cascade layers, which decays as the system reaches its equilibrium[2]. This allows the system to transfer knowledge if a sequence of different training data $D_A, D_B, D_C, \dots$ is presented to the network. The hierarchy of priors absorbs salient features from all tasks and thus mitigates forgetting and promotes transfer (see Fig. 1C).

Experimental Results

To evaluate the model outlined above we applied it to a deep network to learn the MNIST dataset. The basic architecture can be found at [13]. Each weight in the network was augmented by a parameter cascade that followed the dynamics described in Eq. (1). The loss function $\mathcal{L}(D)$ was given by the mean squared error loss between the true and predicted labels.

To obtain multiple tasks to train our network on, we used the permuted MNIST dataset [5]. Therefore we started from the MNIST dataset and performed random shuffling of the pixels of each image to obtain four new datasets (denoted here by A, B, C, D). The shuffling of the individual pixels was kept fixed within each dataset.

Fig. 2A shows the training result if **no** cascade was used (a conventional deep network). All results show mean and standard deviation over 10 individual trials. The architecture achieved an accuracy of 97.942 ± 0.004 on the unperturbed MNIST task (A). This value will function as ground truth subsequently. After training for 23 epochs we switched the task to B. The network quickly adapts to this task change, but the performance on task A strongly declines. We subsequently performed further tasks switches which leads to further decline of performance on previous tasks. After the fourth task switch, the accuracy on task A drops to about 14% (only slightly above chance level) indicating that the network has almost fully forgotten the first task.

[2]A study of the timescale of this equilibration process was done in [2].

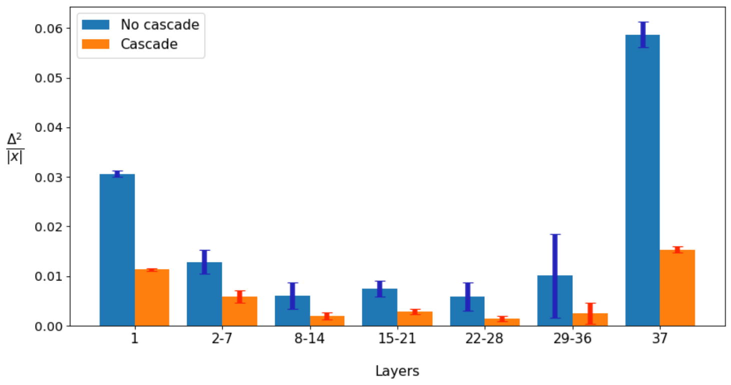

Figure 3: Comparison of the fluctuations in the change in network weights between learning the first task and after training all 4 tasks, evaluated over the same network during two experiments made by turning off and on cascade learning. Each point corresponds to the fluctuations measured in one layer, averaged over 10 simulations. The fluctuations observable in the cascade model are always lower than those of the model which doesn't use the cascades, as shown in figure 10 where we represent the residuals between these two quantities.

Fig. 2B shows our best result on the same experiment when the parameter cascade was active and the algorithm did not have access to the information that the task switch happens. To achieve this result we used an online regulation of the parameters C_0 and $g_{0,1}$ (that did not imply knowledge about task switch times, see Appendix A.2 for details). The performance after task switches was significantly better (task A performance did not drop below 80% after switch to B), but still starkly degraded compared to the baseline performance before task switch.

Next, we evaluated the source of the observed performance increase. To do so, we took a snapshot W_A of the network weights after training on the first task (A) and used it as a reference to compare weights after training subsequent tasks. We first computed the difference distribution $W_i - W_A$, where W_i indicates the network weights after learning task i. Using these quantities we evaluated the mean weight changes per network layer. A comparison of the training result with and without parameter cascade is shown in Fig. 3. Weight changes were consistently lower if the parameter cascade was active. Interestingly parameter fluctuations were lower for intermediate network layers for both methods indicating that hidden representations of the deep network are shared among tasks.

3 Conclusion

We have explored a model for continual learning based on consolidations of synaptic parameters. The model combines previously considered models by Kirkpatrick et al. and the cascade model. We also establish a theoretical correspondence between the two models. As in the original cascade model, we added a cascade of hidden parameters to every network weight. The cascade parameters interact reciprocally and can selectively slow down weight changes. Networks which do not implement these cascades tend to modify their weights much more when learning new tasks, leading to interference between tasks while in our model forgetting happens gradually. Still we found significant performance decrease if the network did not have access to the task switching times. This could be caused by various factors. First, the memory of our network could be a limiting factor to the number of tasks it can learn at the same time, so that obtaining accuracies of 90% and above on all four tasks might be impossible. Additional experiments with increased network sizes are needed to evaluate this possibility. Second, the schedule of the interaction strengths that was used here might not be the optimal one due to the asymptotic behavior which decreases quadratically with the number of epochs (see Appendix A.2). For example, a logarithmic scaling could yield better results. Lastly, another possibility is that the information about task switches is such a strong feature that it is impossible for an algorithm to approach the same performance and learning speed as in Kirkpatrick et al. without this knowledge. Further work is required to evaluate these possible explanations.

References

[1] German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71, 2019.

[2] Marcus K Benna and Stefano Fusi. Computational principles of biological memory. *arXiv preprint arXiv:1507.07580*, 2015.

[3] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, page 201611835, 2017.

[4] Stefano Fusi, Patrick J Drew, and LF Abbott. Cascade models of synaptically stored memories. *Neuron*, 45(4):599–611, 2005.

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

[6] Yu Zhang and Qiang Yang. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2021.

[7] James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3):419, 1995.

[8] Zhizhong Li and Derek Hoiem. Learning without forgetting. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 614–629, Cham, 2016. Springer International Publishing.

[9] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[10] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: An astounding baseline for recognition. In *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 512–519, 2014.

[11] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[12] Marcus K Benna and Stefano Fusi. Computational principles of synaptic memory consolidation. *Nature Neuroscience*, 19(12):1697, 2016.

[13] Jonas Matuzas. https://github.com/matuzas77/mnist-0.17.

[14] David Kappel, Stefan Habenschuss, Robert Legenstein, and Wolfgang Maass. Synaptic sampling: A bayesian approach to neural network plasticity and rewiring. In *Advances in Neural Information Processing Systems*, pages 370–378, 2015.

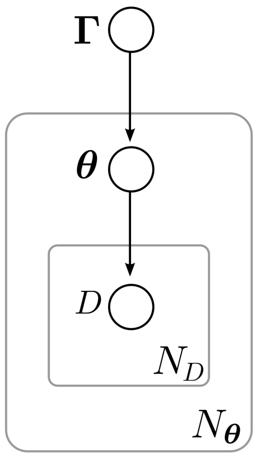

Figure 4: **Bayesian network for the hierarchical model.** Nodes represent random variables, and arrows indicate direct causal effects between these variables. A hierarchy of parameters ($\boldsymbol{\theta}$ and $\mathbf{\Gamma}$) are inferred to explain the data D. N_D and $N_{\boldsymbol{\theta}}$ represent batch sizes (the number of data samples over which the inference is performed).

A Appendix

A.1 Proof of convergence

Here we provide the proof that the cascade dynamics (1) implement a Bayesian learning model that optimizes the probabilistic loss function (5). This proof is a variant of the one provided in [14] and uses stochastic calculus to show convergence of the dynamics to a stationary distribution. The required stochastic term, that is expressed in the proof as an idealized Wiener process, is approximately realized in our experiments through stochastic gradient decent. We first show the general result and then the special case for Gaussian distributions which is identical to the cascade dynamics (1).

Learning in an artificial neural network is commonly described as the task of fitting the network parameters $\boldsymbol{\theta}$ to the data D. The parameter vector $\boldsymbol{\theta}$ is the set of network weights and parameters of all layers of the network. The Bayesian learning theory suggest to view inputs D and also network parameters $\boldsymbol{\theta}$ as random variables (as opposed to fixed values). Learning in this model is done by probabilistic inference over the network parameters. For a given neural network $\mathcal{N}$ that defines a probabilistic model $p_{\mathcal{N}}(D|\boldsymbol{\theta})$ the learning goal is to find the posterior distribution of parameters $p^*(\boldsymbol{\theta} \,|\, D)$ given the data, which by Bayes rule is given by

$$p^*(\boldsymbol{\theta} \,|\, D) \;=\; \frac{1}{\mathcal{Z}} p_{\mathcal{N}}(D|\boldsymbol{\theta})\, p_{\mathcal{S}}(\boldsymbol{\theta}) \;, \tag{4}$$

where $\mathcal{Z}$ is a factor that guarantees normalization.

This Bayesian view on learning induces the need for the prior distribution $p_{\mathcal{S}}(\boldsymbol{\theta})$ that defines a-priori knowledge about the network parameters (e.g. structural rules such as sparseness constraints). The choice for this prior distribution can have a significant impact on the learning speed and robustness. Unfortunately it is in general a hard problem to select good priors and a bad choice can lead to degraded learning performance. Therefore it is commonly assumed that learning the prior distribution is part of the learning problem. To this end, the Bayesian learning scheme is extended by adding additional parameters $\mathbf{\Gamma}$ to the prior distribution, i.e. $p_{\mathcal{S}}(\boldsymbol{\theta} \,|\, \mathbf{\Gamma})$. The model is illustrated in Fig. 4. The learning goal (4) is then replaced by making joint inference about $\boldsymbol{\theta}$ and $\mathbf{\Gamma}$. Inspecting the Bayesian network shown in Fig. 4 we find this to be given by

$$p^*(\boldsymbol{\theta}, \mathbf{\Gamma} \,|\, D) \;=\; \frac{1}{\mathcal{Z}} p_{\mathcal{N}}(D|\boldsymbol{\theta})\, p_{\mathcal{S}}(\boldsymbol{\theta} \,|\, \mathbf{\Gamma})\, p(\mathbf{\Gamma}) \;. \tag{5}$$

Note that the Bayesian learning scheme again induces a prior over the parameters $p(\mathbf{\Gamma})$. This allows us in principle to build up a hierarchy of an arbitrary number of parameters. For the sake of brevity we focus in this proof on the first level of the hierarchical model (Eq. (5)).

Stochastic learning rules for hierarchical Bayesian models

We show here that the learning problem (5) can be solved by stochastic learning rules acting on the synaptic parameters $\boldsymbol{\theta}$ and $\boldsymbol{\Gamma}$. We consider the following Itô stochastic differential equations, operating on sets of synaptic parameters $\boldsymbol{\theta} = \{\theta_i\}$ and hyperparameters $\boldsymbol{\Gamma} = \{\Gamma_j\}$.

$$\begin{aligned} d\theta_i \;&=\; \nu_{\boldsymbol{\theta}} \left(\frac{\partial}{\partial \theta_i} \log p(D|\boldsymbol{\theta}) + T\, \frac{\partial}{\partial \theta_i} \log p(\boldsymbol{\theta}|\boldsymbol{\Gamma}) \right) dt \;+\; \sqrt{2\,\nu_{\boldsymbol{\theta}} T} \cdot d\mathcal{W}_i^t \\ d\Gamma_j \;&=\; \nu_{\boldsymbol{\Gamma}} \left(\frac{\partial}{\partial \Gamma_j} \log p(\boldsymbol{\theta}|\boldsymbol{\Gamma}) + \frac{\partial}{\partial \Gamma_j} \log p(\boldsymbol{\Gamma}) \right) dt \;+\; \sqrt{2\,\nu_{\boldsymbol{\Gamma}}} \cdot d\mathcal{W}_j^t \end{aligned} \tag{6}$$

where $\nu_{\boldsymbol{\theta}}$ and $\nu_{\boldsymbol{\Gamma}}$ act as learning rates and T is a temperature parameter. $\mathcal{W}_{i/j}^t$ denote standard Wiener processes, i.e.

$$\mathcal{W}_{i/j}^t - \mathcal{W}_{i/j}^s \sim \mathcal{N}(0, t-s) \ . \tag{7}$$

Under the assumption that $\boldsymbol{\Gamma}$ evolves slow compared to $\boldsymbol{\theta}$ (see next section) it can be shown that the unique stationary distribution $p^*(\boldsymbol{\theta})$ of the set of SDEs (6) is given by

$$p^*(\boldsymbol{\theta}, \boldsymbol{\Gamma}) = \frac{1}{\mathcal{Z}} p(D|\boldsymbol{\theta})^{\frac{1}{T}}\, p(\boldsymbol{\theta}|\boldsymbol{\Gamma})\, p(\boldsymbol{\Gamma}) \ . \tag{8}$$

where $\mathcal{Z}$ is a normalizer that assures $\iint p^*(\boldsymbol{\theta}, \boldsymbol{\Gamma}) d\boldsymbol{\theta} d\boldsymbol{\Gamma} = 1$. The parameter T acts as a 'temperature' parameter making the probability landscape flat for large values of T (high temperature).

Adiabatic assumption and limits on the learning rates

For the following proof to be valid we require that the SDEs (6) are only weakly coupled. If the dynamics of the prior parameters $\boldsymbol{\Gamma}$ is too fast compared to the dynamics of the parameters $\boldsymbol{\theta}$ priors, the system may start to oscillate. We use here an adiabatic assumption to stabilize the learning and to simplify the analysis of the SDEs. Interestingly this adiabatic assumption is also implicit in the hierarchy of learning rates in [2].

Adiabatic assumptions are well known in physics (e.g. thermodynamics) and states that the dynamics of a process A can be assumed to be independent of another process B, if B is sufficiently slow compared to A. In our model of interacting synaptic parameters, the speed of the parameter dynamics is determined by the learning rate parameters $\nu_{\boldsymbol{\theta}}$ and $\nu_{\boldsymbol{\Gamma}}$ and the temperature T. More precisely, for the example introduced in Fig. 4, we require that $\boldsymbol{\Gamma}$ evolves slow compared to $\boldsymbol{\theta}$, i.e. $\nu_{\boldsymbol{\Gamma}} \ll \nu_{\boldsymbol{\theta}}$. In general, there does not seem to be a theorem that allows to make a strong statement about the precise relationship between $\nu_{\boldsymbol{\Gamma}}$ and $\nu_{\boldsymbol{\theta}}$. The adiabatic assumption is only guaranteed to hold in general for $\nu_{\boldsymbol{\Gamma}} \to 0$.

We derive here a closer estimation, based on ideas from information theory, for the fastest allowed dynamics in our hierarchical Bayesian learning scenario. Analyzing the SDEs (6) we find that the dynamics of $\boldsymbol{\theta}$ is governed by the drift term $A(\boldsymbol{\theta}, \boldsymbol{\Gamma}) = \frac{\partial}{\partial \theta_i} \log p(D|\boldsymbol{\theta}) + T\, \frac{\partial}{\partial \theta_i} \log p(\boldsymbol{\theta}|\boldsymbol{\Gamma})$ and the diffusion $B(\boldsymbol{\theta}, \boldsymbol{\Gamma}) = \sqrt{2\,\nu_{\boldsymbol{\theta}} T} \cdot d\mathcal{W}_i^t$. However, since also $\boldsymbol{\Gamma}$ is subject to Brownian motion, the drift term induces a noise coupling between θ_i and Γ_i. Therefore the total effective noise acting on θ_i is given by $B^{total}(\boldsymbol{\theta}, \boldsymbol{\Gamma}) = \sqrt{2\,\nu_{\boldsymbol{\theta}}\,\nu_{\boldsymbol{\Gamma}}\,T} \cdot d\mathcal{W}_i^t + \sqrt{2\,\nu_{\boldsymbol{\theta}} T} \cdot d\mathcal{W}_i^t$. We assume that the noise coupling vanishes compared to the uncorrelated noise if the effective noise amplitude is suppressed below a threshold ($\ll 1$). This is guaranteed if learning rates are exponentially vanishing over the cascade of priors, which was used in all our experiments (see Appendix A.2), and was also the result established in [2].

Proof of Equation (8)

We prove here that eq. (8) is the unique stationary distribution of the parameter dynamics (6). Under the assumption that $\boldsymbol{\Gamma}$ evolves slowly compared to $\boldsymbol{\theta}$ the Itô SDEs (6) translate into the following

Fokker-Planck equation,

$$\begin{aligned}\frac{d}{dt}p_{\mathrm{FP}}(\boldsymbol{\theta},\boldsymbol{\Gamma},t) = &-\sum_i \frac{\partial}{\partial\theta_i}\left(\left(\nu_{\boldsymbol{\theta}}\frac{\partial}{\partial\theta_i}\log p(D|\boldsymbol{\theta}) + T\,\nu_{\boldsymbol{\theta}}\frac{\partial}{\partial\theta_i}\log p(\boldsymbol{\theta}|\boldsymbol{\Gamma})\right)p_{\mathrm{FP}}(\boldsymbol{\theta},\boldsymbol{\Gamma},t)\right)\\ &-\sum_j \frac{\partial}{\partial\Gamma_j}\left(\left(\nu_{\boldsymbol{\Gamma}}\frac{\partial}{\partial\Gamma_j}\log p(\boldsymbol{\theta}|\boldsymbol{\Gamma}) + \nu_{\boldsymbol{\Gamma}}\frac{\partial}{\partial\Gamma_j}\log p(\boldsymbol{\Gamma})\right)p_{\mathrm{FP}}(\boldsymbol{\theta},\boldsymbol{\Gamma},t)\right)\\ &+\sum_i \frac{\partial^2}{\partial\theta_i^2}\left(T\,\nu_{\boldsymbol{\theta}}\,p_{\mathrm{FP}}(\boldsymbol{\theta},\boldsymbol{\Gamma},t)\right) + \sum_j \frac{\partial^2}{\partial\Gamma_j^2}\left(\nu_{\boldsymbol{\Gamma}}\,p_{\mathrm{FP}}(\boldsymbol{\theta},\boldsymbol{\Gamma},t)\right)\end{aligned}$$

Plugging in the presumed stationary distribution (8) one obtains,

$$\begin{aligned}\frac{d}{dt}p_{\mathrm{FP}}(\boldsymbol{\theta},\boldsymbol{\Gamma},t) = &-\sum_i \frac{\partial}{\partial\theta_i}\left(\left(\nu_{\boldsymbol{\theta}}\frac{\partial}{\partial\theta_i}\log p(D|\boldsymbol{\theta}) + T\,\nu_{\boldsymbol{\theta}}\frac{\partial}{\partial\theta_i}\log p(\boldsymbol{\theta}|\boldsymbol{\Gamma})\right)p^*(\boldsymbol{\theta},\boldsymbol{\Gamma})\right)\\ &-\sum_j \frac{\partial}{\partial\Gamma_j}\left(\left(\nu_{\boldsymbol{\Gamma}}\frac{\partial}{\partial\Gamma_j}\log p(\boldsymbol{\theta}|\boldsymbol{\Gamma}) + \nu_{\boldsymbol{\Gamma}}\frac{\partial}{\partial\Gamma_j}\log p(\boldsymbol{\Gamma})\right)p^*(\boldsymbol{\theta},\boldsymbol{\Gamma})\right)\\ &+\sum_i \frac{\partial^2}{\partial\theta_i^2}\left(T\,\nu_{\boldsymbol{\theta}}\,p^*(\boldsymbol{\theta},\boldsymbol{\Gamma})\right) + \sum_j \frac{\partial^2}{\partial\Gamma_j^2}\left(\nu_{\boldsymbol{\Gamma}}\,p^*(\boldsymbol{\theta},\boldsymbol{\Gamma})\right)\\ = &-\sum_i \frac{\partial}{\partial\theta_i}\left(\left(\nu_{\boldsymbol{\theta}}\frac{\partial}{\partial\theta_i}\log p(D|\boldsymbol{\theta}) + T\,\nu_{\boldsymbol{\theta}}\frac{\partial}{\partial\theta_i}\log p(\boldsymbol{\theta}|\boldsymbol{\Gamma})\right)p^*(\boldsymbol{\theta},\boldsymbol{\Gamma})\right)\\ &-\sum_j \frac{\partial}{\partial\Gamma_j}\left(\left(\nu_{\boldsymbol{\Gamma}}\frac{\partial}{\partial\Gamma_j}\log p(\boldsymbol{\theta}|\boldsymbol{\Gamma}) + \nu_{\boldsymbol{\Gamma}}\frac{\partial}{\partial\Gamma_j}\log p(\boldsymbol{\Gamma})\right)p^*(\boldsymbol{\theta},\boldsymbol{\Gamma})\right)\\ &+\sum_i \frac{\partial}{\partial\theta_i}\left(\nu_{\boldsymbol{\theta}}\left(\frac{\partial}{\partial\theta_i}\log p(D|\boldsymbol{\theta}) + T\frac{\partial}{\partial\theta_i}\log p(\boldsymbol{\theta}|\boldsymbol{\Gamma})\right)p^*(\boldsymbol{\theta},\boldsymbol{\Gamma})\right)\\ &+\sum_j \frac{\partial}{\partial\Gamma_j}\left(\left(\nu_{\boldsymbol{\Gamma}}\frac{\partial}{\partial\Gamma_j}\log p(\boldsymbol{\theta}|\boldsymbol{\Gamma}) + \nu_{\boldsymbol{\Gamma}}\frac{\partial}{\partial\Gamma_j}\log p(\boldsymbol{\Gamma})\right)p^*(\boldsymbol{\theta},\boldsymbol{\Gamma})\right)\\ = &\,0\;.\end{aligned}$$

This proves that (8) is a stationary distribution of the parameter sampling dynamics (6). If the matrix of diffusion coefficients is invertible, and the potential conditions are satisfied, the stationary distribution can be obtained (uniquely) by simple integration. Since the matrix of diffusion coefficients is diagonal in our model (under the adiabatic assumption), the diffusion coeffcient matrix is trivially invertible since all diagonal elements, are positive.

Gaussian special case is equivalent to cascade dynamics

Here we show that the cascade dynamics (1) is identical to the hierarchical Bayesian model from Fig. 1B with univariate Gaussian prior distributions. For the sake of brevity we show this for the fist level of the cascade, and the over levels follow. Eq. (1) for the first level $k = 1$ is given by

$$C_1\frac{\mathrm{d}u_1}{\mathrm{d}t} \;=\; g_{0,1}(w-u_1) + g_{1,2}(u_1-u_2)\;. \tag{9}$$

We show that this expression follows from Eq. 6 if the parameter priors are assumed to be Gaussian, $\log p(\boldsymbol{\theta}|\boldsymbol{\Gamma}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\Gamma},\sigma_{0,1}^2)$ and $\log p(\boldsymbol{\Gamma}) = \mathcal{N}(\boldsymbol{\Gamma}|\boldsymbol{\Gamma}_2,\sigma_{1,2}^2)$, where $\boldsymbol{\Gamma}_2$ are additional hyperparameters and, $\sigma_{0,1}^2$ and $\sigma_{1,2}^2$ are scalar variances. Using this we get

$$\frac{\partial}{\partial u_1}\log p\,(\boldsymbol{\theta}\,|\,\boldsymbol{\Gamma}) \;=\; \frac{1}{\sigma_{0,1}^2}(w-u_1) \tag{10}$$

$$\frac{\partial}{\partial u_1}\log p\,(\boldsymbol{\Gamma}\,|\,\boldsymbol{\Gamma}_2) \;=\; \frac{1}{\sigma_{1,2}^2}(u_1-u_2)\;. \tag{11}$$

Plugging this back into Eq. 6 and defining $g_{0,1} = \frac{1}{\sigma_{0,1}^2}$, $g_{1,2} = \frac{1}{\sigma_{1,2}^2}$ and $C_1 = \nu_{\mathbf{\Gamma}}$ we get

$$d\Gamma_j \;=\; C_1\bigg(g_{0,1}(w - u_1) + g_{1,2}(u_1 - u_2)\bigg)dt \,+\, \sqrt{2\,C_1}\cdot d\mathcal{W}_j^t\,, \tag{12}$$

which is identical to Eq. 9 up to the wiener process which is provided in our implementation through stochastic gradient descent. This results follows for the other levels of the cascade using the same argument as long as the adiabatic assumption is fulfilled. This proves that the cascade model Eq. (1) realizes a hierarchical Bayesian learning algorithm with Gaussian prior distributions.

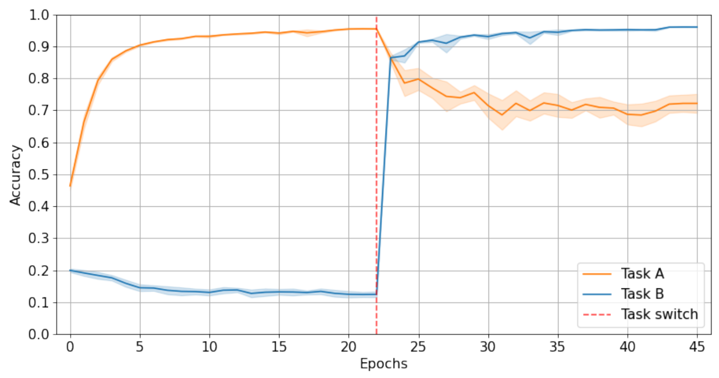

Figure 5: Accuracy of the model on the two tasks during training using cascade parameters for memory retention. The weight c was set to 0.95 and kept constant throughout training. The accuracy on task A plateaus around 0.7 even after training on task B.

A.2 Simulation details

In our simulations we followed the procedure described in [2] and set the values of our parameters C_k and $g_{k,k+1}$ in an exponential manner, with $C_0 > 1$ and $g_{0,1} < 1$, and following the rules $C_k = (C_0)^k$ and $g_{k,k+1} = (g_{1,0})^k$. This implies that the dynamics of the cascade layers slows exponentially with the depth of the cascade which is important for longer memory retention (and is also required for the adiabatic assumption in Appendix A.1). One hyperparameter that can have a great influence on our algorithm is what we call the cascade parameter c, which determines the relative importance of the two loss terms used during training. To illustrate the role of this parameter we consider the loss function, which if a modified version of (2):

$$L = (1-c) \sum_{outputs} (y - y_{pred})^2 + c \left\langle g_{0,1}(u_1 - w)^2 \right\rangle \ , \tag{13}$$

where the symbols $\langle \cdot \rangle$ stand for an average over the weights of the network, and $c \in [0,1]$. The parameter c allows us to decide how much important the prior is given by the cascades, which enforces memory stability, should be with respect to the weight changes that would be dictated by simple maximum likelihood learning. In other words it contains information regarding how stiff the network should be with respect to parameter changes. The weight update described by Eq. (2) can be retrieved by setting $c = 1/2$, which would imply $C_0 = 2$. By setting $c \neq 1/2$ we obtain more control over training and explore further mechanisms of memory retention. These values of c correspond to the cascade model alone with modified coupling parameters $g_{k,j}$ and learning rates C_k. In the following we will illustrate the relative role of the data-dependent loss and the prior through the parameter c.

A.2.1 Preliminary setup

In order to test the functioning of our cascade model, we first experimented with the simpler problem of learning two tasks, and tried out different values for the cascade parameter c, the time scales C and g from equation 1, and the depth of the cascade (the number of cascade layers associated with each weight in the network). As highlighted by Benna and Fusi, the depth of the cascade does not seem to strongly influence the resulting memory, as long as the parameters C_k and $g_{k,j}$ are tuned so that the timescales increase exponentially with the depth of the cascade. During computational simulations we are limited by the range of values allowed by floating point representations, and we found that a good rule of thumb for the exponential increase in timescales is to exploit the full range of values available during computation, so that the memory decay happens on the longest timescale allowed by our machines. In practice, a small value for C_N^{-1} and $g_{N-1,N}^{-1}$, where N is the depth of the cascade, is sufficient to obtain the same level of accuracy, but we found that decreasing it to the limit of floating point precision does not interfere with the results, and theoretically it should yield a longer lasting memory. By small we mean, from our trials, numbers of the order of 10^{-20}, but we did not perform a

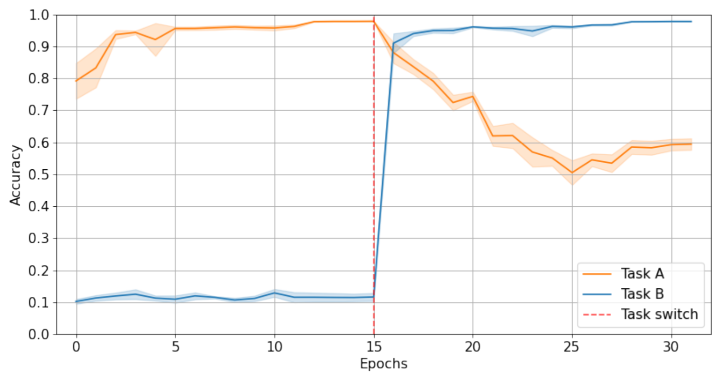

Figure 6: Control experiment used to check the influence of the backpropagation algorithm used during training. In these simulations we used Adam instead of SGD, but kept all other parameters unchanged. The accuracy decreased by more than 10%

precise study on how much we could increase these values without affecting the accuracy. Moreover, the value of c does not seem to influence the accuracy as strongly as we would have initially believed, with networks to which we assigned $c = 0.5$ performing just as well as networks in which we set $c = 0.95$. The averaged results obtained from 10 simulations using 13 cascade layers and $c = 0.95$ are shown in figure 5. We can see, from this graph how the decrease in accuracy after the task switch is no longer as abrupt as it was in the example without the cascade parameters, but the same plateau is reached a few epochs after training on the new task. Overall, the accuracy on task A remained above 70% even after training on task B was completed, with an increase of more than 10% in the accuracy over the model without the cascades.
Additionally, we performed a control experiment to verify if the backpropagation algorithm might have an influence over the results, and we tested the same architecture by using the Adam optimizer. The results are shown in figure 6, where we see how even though the decrease in accuracy is still slower than in the case without any cascade, the performance on task A drops significantly, and reaches the same accuracy as the model without hard-wired memory retention. This is an important result to keep in mind when trying to apply this model to new architectures.

A.2.2 Dynamic control of the cascade parameter

Since the results we obtained did not seem to rely on the value of the parameter c as much as we would have believed, we tried to use a dynamic parameter, and implemented two different strategies.

First of all, we introduced a change with a step function increasing c to a higher value at the time of the task switch (t_{switch}):

$$c(t) = a + b\theta(t - t_{switch}) \ , \tag{14}$$

where $\theta(t)$ is the Heaviside step function. In our experiments, we set $a = 0.5$ and $b = 0.45$ (which respect the condition $c \in [0, 1]$). Here and in the remainder of the discussion t denotes the index of the learning epoch. The results obtained by this method are shown in figure 7: by changing the value of the cascade parameter in this manner we were able to maintain an accuracy on task A of about 90% even after training task B, and without losing on the performance on task B itself. Furthermore, we used this method to test the behavior of the network when the cascade parameter is set to the boundary value $c = 1$ after switching task, as shown in figure 8. As we would have expected, by only keeping the cascade dynamics alive, no learning of task B whatsoever is allowed after the switch. Moreover, even though the cascade layers are directly interacting with the weights of the network, changing their values, we observed no significant drop in accuracy on the first task after effectively turning off the squared differences term in our loss function which was associated with learning. This in a way proves that as we speculated, the dynamics of the decay of the cascades towards equilibrium when they are under no external stimulation happens on timescales which are much longer than those used during training.

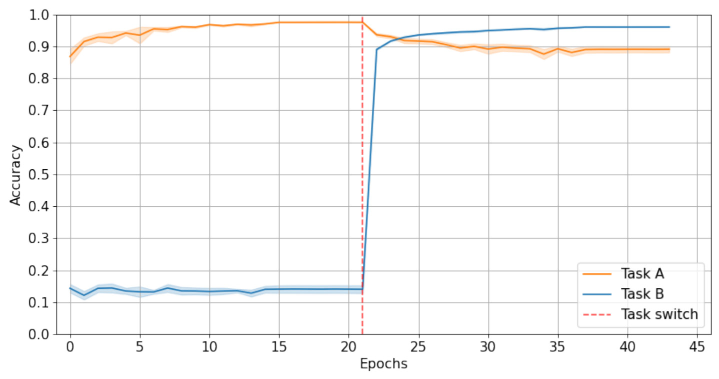

Figure 7: Accuracy on the two tasks measured in simulations in which after switching tasks we changed the weight c from 0.5 to 0.95. This abrupt change serves the purpose of stiffening the network weights to avoid forgetting the preiously learned task A. This method requires the intervention of an external observer with knowledge about the occurrence of the task switch.

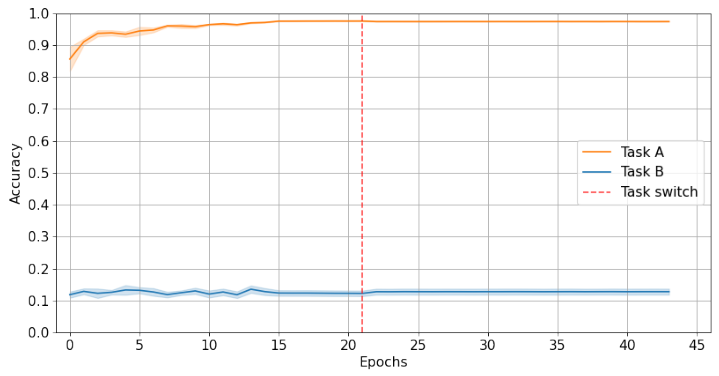

Figure 8: Control experiment in which we set the cascade parameter $c = 1$ after switching tasks, meaning that the only network weight changes allowed to the network are those dictated by the cascade layers' dynamics, and no learning of task B can occur.

A.2.3 Data-independent cascade parameter schedule

The problem with the previous approach to the dynamic behavior of the cascade parameter c is that it requires the presence of an external observer knowing the exact moment at which a task switch happens, t_{switch}, and thus would not work in experimental conditions in which the value of this parameter is not known. What some other experimenters resorted to, to obviate to this kind of problem is to introduce an oracle network which infers when a new task is being given to the network, and performs these parameter changes automatically, so that this process can happen in a completely unsupervised manner[3]. Instead, we tried a different approach which can only work due to the specific model for memory and loss function that we are using: we tried to give the parameter c a continuous functional dependence on the training epoch which should gradually stiffen the network so that its memory gets consolidated independently on which task it is being trained on. For the purposes of this study, we only experimented with the easiest functional relation, a hyperbolic approach to $c = 1$:

$$c(t) = 1 - \frac{1}{1+t} \tag{15}$$

In figure 10 we represent the differences between the two sets of data shown in figure 3, but evaluated at two different points during training: after learning tasks A and B, and after having learnt all 4

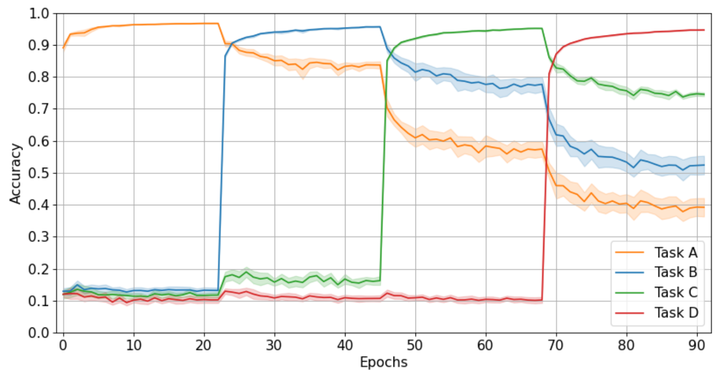

Figure 9: Accuracy during training over 4 consecutive tasks of a network in which the weight parameter is increased as a function of the training epochs elapsed t: $c(t) = 1 - \frac{1}{1+t}$. As we have seen before, keeping a static weight is not optimal, and the resulting 2-task accuracy would be around 70%, while with this method we can achieve an accuracy well above 80%. Furthermore, the accuracy on task A at the end of the experiment is kept to around 40%, significantly higher the values expected for random guessing. The nonetheless considerable decrease in accuracy could be attributed to at least two factors: the limited memory of the architecture under study, which was not optimized to memorize multiple tasks, and the asymptotic decrease in the parameter change, Δc, between epochs, which ideally could be optimized to reach a value closer to 1 given the number of epochs of training in order to exploit the full range of allowed values.

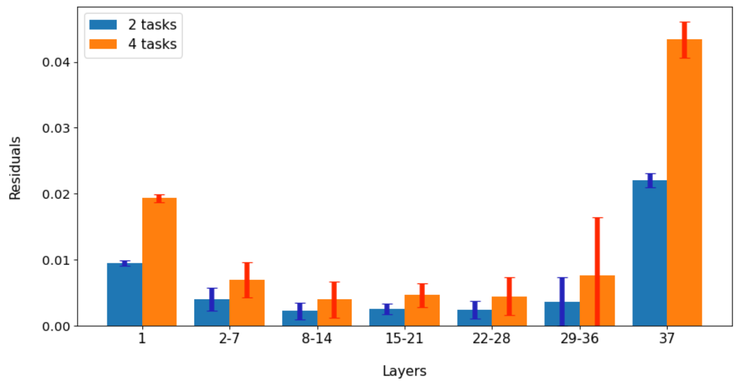

Figure 10: Residuals of the data shown in figure 3, but in which we analyse the change in network weights between the reference weights learned during the first task with those learned after training 2 tasks, and after all 4 tasks. This shows how the more tasks we give the network to learn, the more the model which does not use the cascades diverges from the cascade model, thus resulting in excessive weight changes in the network.

tasks. What this plot tells us is that as we increase the number of tasks that the network is required to remember, the deviation of the models without the cascade from the models with the cascades is increasing, and always positive, meaning that the memory about previously learned tasks decays more and more with respect to our model as we increase the number of tasks, as we would have expected.