Online Structured Laplace Approximations For Overcoming Catastrophic Forgetting

Hippolyt Ritter*
University College London

Aleksandar Botev
University College London

David Barber
University College London
& Alan Turing Institute

Abstract

We introduce the Kronecker factored online Laplace approximation for overcoming catastrophic forgetting in neural networks. The method is grounded in a Bayesian online learning framework, where we recursively approximate the posterior after every task with a Gaussian, leading to a quadratic penalty on changes to the weights. The Laplace approximation requires calculating the Hessian around a mode, which is typically intractable for modern architectures. In order to make our method scalable, we leverage recent block-diagonal Kronecker factored approximations to the curvature. Our algorithm achieves over 90% test accuracy across a sequence of 50 instantiations of the permuted MNIST dataset, substantially outperforming related methods for overcoming catastrophic forgetting.

1 Introduction

Creating an agent that performs well across multiple tasks and continuously incorporates new knowledge has been a longstanding goal of research on artificial intelligence. When training on a sequence of tasks, however, the performance of many machine learning algorithms, including neural networks, decreases on older tasks when learning new ones. This phenomenon has been termed 'catastrophic forgetting' [6, 26, 33] and has recently received attention in the context of deep learning [8, 16]. Catastrophic forgetting cannot be overcome by simply initializing the parameters for a new task with optimal ones from the old task and hoping that stochastic gradient descent will stay sufficiently close to the original values to maintain good performance on previous datasets [8].

Bayesian learning provides an elegant solution to this problem. It combines the current data with prior information to find an optimal trade-off in our belief about the parameters. In the sequential setting, such information is readily available: the posterior over the parameters given all previous datasets. It follows from Bayes' rule that we can use the posterior over the parameters after training on one task as our prior for the next one. As the posterior over the weights of a neural network is typically intractable, we need to approximate it. This type of Bayesian online learning has been studied extensively in the literature [31, 7, 13].

In this work, we combine Bayesian online learning [31] with the Kronecker factored Laplace approximation [34] to update a quadratic penalty for every new task. The block-diagonal Kronecker factored approximation of the Hessian [23, 2] allows for an expressive scalable posterior that takes interactions between weights within the same layer into account. In our experiments we show that this principled approximation of the posterior leads to substantial gains in performance over simpler diagonal methods, in particular for long sequences of tasks.

*Corresponding author: j.ritter@cs.ucl.ac.uk

Preprint. Work in progress.

2 Bayesian online learning for neural networks

We are interested in optimizing the parameters θ of a single neural network to perform well across multiple tasks $\mathcal{D}_1, \ldots, \mathcal{D}_T$, specifically finding a MAP estimate $\theta^* = \arg\max_\theta p(\theta|\mathcal{D}_1, \ldots, \mathcal{D}_T)$. However, the datasets arrive sequentially and we can only train on one of them at a time.

In the following, we first discuss how Bayesian online learning solves this problem and introduce an approximate procedure for neural networks. We then review recent Kronecker factored approximations to the curvature of neural networks and how to use them to obtain a better fit to the posterior. Finally, we introduce a hyperparameter that acts as a regularizer on the approximation to the posterior.

2.1 Bayesian online learning

Bayesian online learning [31], or Assumed Density Filtering [25], is a framework for updating an approximate posterior when data arrive sequentially. Using Bayes' rule we would like to simply incorporate the most recent dataset $\mathcal{D}_{t+1}$ into the posterior as:

$$p(\theta|\mathcal{D}_{1:t+1}) = \frac{p(\mathcal{D}_{t+1}|\theta)p(\theta|\mathcal{D}_{1:t})}{\int d\theta' p(\mathcal{D}_{t+1}|\theta')p(\theta'|\mathcal{D}_{1:t})} \tag{1}$$

where we use the posterior $p(\theta|\mathcal{D}_{1:t})$ from the previously observed tasks as the prior over the parameters for the most recent task. As the posterior given the previous datasets is typically intractable, Bayesian online learning formulates a parametric approximate posterior q with parameters ϕ_t, which it iteratively updates in two steps:

Update step In the update step, the approximate posterior q with parameters ϕ_t from the previous task is used as a prior to find the new posterior given the most recent data:

$$p(\theta|\mathcal{D}_{t+1}, \phi_t) = \frac{p(\mathcal{D}_{t+1}|\theta)q(\theta|\phi_t)}{\int d\theta' p(\mathcal{D}_{t+1}|\theta')q(\theta'|\phi_t)} \tag{2}$$

Projection step The projection step finds the distribution within the parametric family of the approximation that most closely resembles this posterior, i.e. sets ϕ_{t+1} such that:

$$q(\theta|\phi_{t+1}) \approx p(\theta|\mathcal{D}_{t+1}, \phi_t) \tag{3}$$

Opper and Winther [31] suggest minimizing the KL-divergence between the approximate and the true posterior, however this is mostly appropriate for models where the update-step posterior and a solution to the KL-divergence are available in closed form. In the following, we therefore propose using a Laplace approximation to make Bayesian online learning tractable for neural networks.

2.2 The online Laplace approximation

Neural networks have found wide-spread success and adoption by performing simple MAP inference, i.e. finding a mode of the posterior:

$$\theta^* = \arg\max_\theta \ \log p(\theta|\mathcal{D}) = \arg\max_\theta \ \log p(\mathcal{D}|\theta) + \log p(\theta) \tag{4}$$

where $p(\mathcal{D}|\theta)$ is the likelihood of the data and $p(\theta)$ the prior. Most commonly used loss functions and regularizers fit into this framework, e.g. using a categorical cross-entropy with L_2-regularization corresponds to modeling the data with a categorical distribution and placing a zero-mean Gaussian prior on the network parameters. A local mode of this objective function can easily be found using standard gradient-based optimizers.

Around a mode, the posterior can be locally approximated using a second-order Taylor expansion, resulting in a Normal distribution with the MAP parameters as the mean and the Hessian of the negative log posterior around them as the precision. Using a Laplace approximation for neural networks was pioneered by MacKay [22].

We therefore proceed in two iterative steps similar to Bayesian online learning, using a Gaussian approximate posterior for q, such that $\phi_t = \{\mu_t, \Lambda_t\}$ consists of a mean μ and a precision matrix Λ:

Update step As the posterior of a neural network is intractable for all but the simplest architectures, we will work with the unnormalized posterior. The normalization constant is not needed for finding a mode or calculating the Hessian. The Gaussian approximate posterior results in a quadratic penalty encouraging the parameters to stay close to the mean of the previous approximate posterior:

$$\begin{aligned} \log p(\theta|\mathcal{D}_{t+1}, \phi_t) &\propto \log p(\mathcal{D}_{t+1}|\theta) + \log q(\theta|\phi_t) \\ &\propto \log p(\mathcal{D}_{t+1}|\theta) - \frac{1}{2}(\theta - \mu_t)^\top \Lambda_t (\theta - \mu_t) \end{aligned} \tag{5}$$

Projection step In the projection step we approximate the posterior with a Gaussian. We first update the mean of the approximation to a mode of the new posterior:

$$\mu_{t+1} = \arg\max_\theta \ \log p(\mathcal{D}_{t+1}|\theta) + \log q(\theta|\phi_t) \tag{6}$$

and then perform a quadratic approximation around it, which requires calculating the Hessian of the negative objective. This leads to a recursive update to the precision with the Hessian of the most recent log likelihood, as the Hessian of the negative log approximate posterior is its precision:

$$\Lambda_{t+1} = H_{t+1}(\mu_{t+1}) + \Lambda_t \tag{7}$$

where $H_{t+1}(\mu_{t+1}) = -\left.\frac{\partial^2 \log p(\mathcal{D}_{t+1}|\theta)}{\partial\theta\partial\theta}\right|_{\theta=\mu_{t+1}}$ is the Hessian of the newest negative log likelihood around the mode. The precision of a Gaussian is required to be positive semi-definite, which is the case for the Hessian at a mode. In order to numerically guarantee this in practice, we use the Fisher Information as an approximation [24] that is positive semi-definite by construction.

The recursion is initialized with the Hessian of the log prior, which is typically constant. For a zero-mean isotropic Gaussian prior, corresponding to an L_2-regularizer, it is simply the identity matrix times the prior precision.[2]

A desirable property of the Laplace approximation is that the approximate posterior becomes peaked around its current mode as we observe more data. This becomes particularly clear if we think of the precision matrix as the product of the number of data points and the average precision. By becoming increasingly peaked, the approximate posterior will naturally allow the parameters to change less for later tasks. At the same time, even though the Laplace method is a local approximation, we would expect it to leave sufficient flexibility for the parameters to adapt to new tasks, as the Hessian of neural networks has been observed to be flat in most directions [36].

We will also compare to fitting the true posterior with a new Gaussian at every task for which we compute the Hessian of all tasks around the most recent MAP estimate:

$$\Lambda_{t+1} = H_{prior} + \sum_{i=1}^{t+1} H_i(\mu_{t+1}) \tag{8}$$

This procedure differs from the online Laplace approximation only in evaluating all Hessians at the most recent MAP parameters instead of the respective task's ones. Technically, this is not a valid Laplace approximation, as we only optimize an approximation to the posterior. Hence the optimal parameters for the approximate objective will not exactly correspond to a mode of the exact posterior. However, as we will use a positive semi-definite approximation to the Hessian, this will only introduce a small additional approximation error.

Calculating the Hessian across all datasets requires relaxing the sequential learning setting to allowing access to previous data 'offline', i.e. between tasks. We use this baseline to check if there is any loss of information in using estimates of the curvature at previous parameter values.

[2] Huszár [14] recently discussed a similar recursive Laplace approximation for online learning, however with limited experimental results and in the context of using a diagonal approximation to the Hessian.

2.3 Kronecker factored approximation of the Hessian

Modern networks typically have millions of parameters, so the size of the Hessian is several terabytes. An approximation that is simple to implement with automatic differentiation frameworks is the diagonal of the Fisher matrix, i.e. the expected square of the gradients, where the expectation is over the datapoints and the conditional distribution defined by the model. While this approximation has been used successfully [16], it ignores interactions between the parameters.

Recent works on second-order optimization [23, 2] have developed block-diagonal approximations to the Hessian. They exploit that, for a single data point, the diagonal blocks of the Hessian of a feedforward network — corresponding to the weights of a single layer — are Kronecker factored, i.e. a product of two relatively small matrices.

We denote a neural network as taking an input $a_0{=}x$ and producing an output h_L. The input is passed through layers $1, \ldots, L$ as the linear pre-activations $h_l{=}W_l a_{l-1}$ and the activations $a_l{=}f_l(h_l)$, where f_l is a non-linear elementwise function. The outputs then parameterize the log likelihood of the data, and, using the chain rule, we can write the Hessian w.r.t. the weights of a single layer as:

$$H_l = \frac{\partial^2 \log p(\mathcal{D}|h_L)}{\partial \operatorname{vec}(W_l) \partial \operatorname{vec}(W_l)} = \mathcal{Q}_l \otimes \mathcal{H}_l \tag{9}$$

where $\operatorname{vec}(W_l)$ is the weight matrix of layer l stacked into a vector and we define $\mathcal{Q}_l = a_{l-1} a_{l-1}^\top$ as the covariance of the inputs to the layer. $\mathcal{H}_l = \frac{\partial^2 \log p(\mathcal{D}|\theta)}{\partial h_l \partial h_l}$ is the pre-activation Hessian, i.e. the second derivative w.r.t. the pre-activations h_l of the layer. We provide the basic derivation of Eq. (9) and the recursive formula for calculating $\mathcal{H}_l$ in Appendix A. To maintain the Kronecker factorization in expectation, i.e. for an entire dataset, [23] and [2] assume the two factors to be independent and approximate the expected Kronecker product by the Kronecker product of the expected factors.

The block-diagonal approximation splits the Hessian-vector product in the quadratic penalty across the layers. Due to the Kronecker factored approximation, it can be calculated efficiently for each layer using the following well-known identity:

$$(\mathcal{Q}_l \otimes \mathcal{H}_l) \operatorname{vec}(W_l - W_l^*) = \operatorname{vec}(\mathcal{H}_l \left(W_l - W_l^*\right)^\top \mathcal{Q}_l) \tag{10}$$

where vec stacks the columns of a matrix into a vector and we use that $\mathcal{H}$ is symmetric.

The block-diagonal Kronecker factored approximation corresponds to assuming independence between the layers and factorizing the covariance between the weights of a layer into the covariance of the columns and rows, resulting in a matrix normal distribution [11]. The same approximation has been used recently to sample from the predictive posterior [34, 9]. While it still makes some independence assumptions about the weights, the most important interactions — the ones within the same layer — are accounted for. In order to guarantee for the curvature being positive semi-definite, we approximate the Hessian with the Fisher Information as in [23] throughout our experiments.

2.4 Regularizing the approximate posterior

Kirkpatrick *et al.* [16], who develop a similar method inspired by the Laplace approximation, suggest using a multiplier λ on the quadratic penalty in Eq. (5). This hyperparameter provides a way of trading off retaining performance on previous tasks against having sufficient flexibility for learning a new one. As modifying the objective would propagate into the recursion for the precision matrix, we instead place the multiplier on the Hessian of each log likelihood and update the precision as:

$$\Lambda_{t+1} = \lambda H_{t+1}(\mu_{t+1}) + \Lambda_t \tag{11}$$

The multiplier affects the width of the approximate posterior and thus the location of the next MAP estimate. As it acts directly on the parameter of a probability distribution, its optimal value can inform us about the quality of our approximation: if it strongly deviates from its natural value of 1, our approximation is a poor one and over- or underestimates the uncertainty about the parameters. We visualize the effect of λ in Fig. 5 in Appendix B.

3 Related work

Our method is closely related to Bayesian online learning [31] and to Laplace propagation [4]. In contrast to Bayesian online learning, as we cannot update the posterior over the weights in closed form, we use gradient-based methods to find a mode and perform a quadratic approximation around it, resulting in a Gaussian approximation. Laplace propagation, similar to expectation propagation [27], maintains a factor for every task, but approximates each of them with a Gaussian. It performs multiple updates, whereas we use each dataset only once to update the approximation to the posterior.

The most similar method to ours for overcoming catastrophic forgetting is Elastic Weight Consolidation (EWC) [16]. EWC approximates the posterior after the first task with a Gaussian. However, it continues to add a penalty for every new task [17]. This is more closely related to Laplace propagation, but may be overcounting early tasks [14] and does not approximate the posterior. Furthermore, EWC uses a simple diagonal approximation to the Hessian. Lee *et al.* [20] approximate the posterior around the mode for each dataset with a diagonal Gaussian in addition to a similar approximation of the overall posterior. They update this approximation to the posterior as the Gaussian that minimizes the KL divergence with the individual posterior approximations. Nguyen *et al.* [30] implement online variational learning [7, 13], which fits an approximation to the posterior through the variational lower bound and then uses this approximation as the prior on the next task. Their Gaussian is fully factorized, hence they do not take weight interactions into account either.

[34] and [9] have independently proposed the use of block-diagonal Kronecker factored curvature approximations [23, 2] to sample from an approximate Gaussian posterior over the weights of a neural network. They find that this requires adding a multiple of the identity to their curvature factors as an ad-hoc regularizer, which is not necessary for our method. In our work, we use an approximate posterior with the same Kronecker factored covariance structure as a prior for subsequent tasks. We iteratively update this approximation for every new dataset. The curvature factors that we accumulate throughout training could be used on top of our method to approximate the predictive posterior similar to [34, 9]. However, both the curvature factors and the mode that our method finds will be different to performing a Laplace approximation in batch mode. Our work links the Kronecker factored Laplace approximation [34] to Bayesian online learning [31] similar to how Variational Continual Learning [30] connects Online Variational Learning [7, 13] to Bayes-by-Backprop [1].

We discuss additional related methods without a Bayesian motivation in Appendix C.

4 Experiments

In our experiments we compare our online Laplace approximation to the approximate Laplace approximation of Eq. (8) as well as EWC [16] and Synaptic Intelligence (SI) [41], both of which also add quadratic regularizers to the objective. Further, we investigate the effect of using a block-diagonal Kronecker factored approximation to the curvature over a diagonal one. We also run EWC with a Kronecker factored approximation, even though the original method is based on a diagonal one. We implement our experiments using Theano [39] and Lasagne [3] software libraries.

4.1 Permuted MNIST

As a first experiment, we test on a sequence of permutations of the MNIST dataset [19]. Each instantiation consists of the 28×28 grey-scale images and labels from the original dataset with a fixed random permutation of the pixels. This makes the individual data distributions mostly independent of each other, testing the ability of each method to fully utilize the model's capacity.

We train a feed-forward network with two hidden layers of 100 units and ReLU nonlinearities on a sequence of 50 versions of permuted MNIST. Every one of these datasets is equally difficult for a fully connected network due to its permutation invariance to the input. We stress that our network is smaller than in previous works as the limited capacity of the network makes the task more challenging. Further, we train on a longer sequence of datasets. Optimization details are in Appendix D.

Fig. 1 shows the mean test accuracy as new datasets are observed for the optimal hyperparameters of each method. We refer to the online Laplace approximation as 'Online Laplace', to the Laplace approximation around an approximate mode as 'Approximate Laplace' and to adding a quadratic

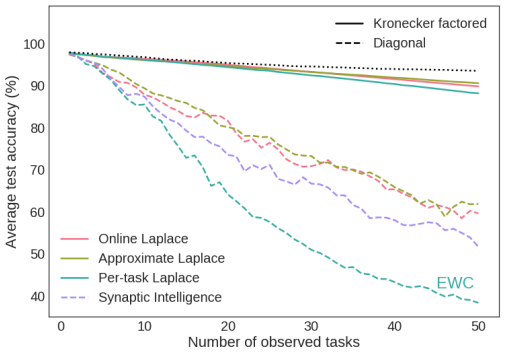
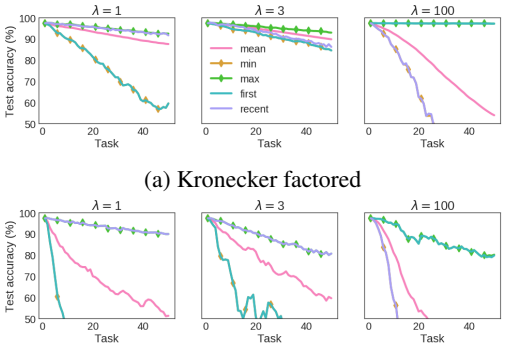


Figure 1: Mean test accuracy on a sequence of permuted MNIST datasets. We categorize SI as a diagonal method, as it does not account for parameter interactions. The dotted black line shows the performance of a single network trained on all observed data at each task.



(a) Kronecker factored

(b) Diagonal

Figure 2: Effect of λ for different curvature approximations for permuted MNIST. Each plot shows the mean, minimum and maximum across the tasks observed so far, as well as the accuracy on the first and most recent task.

penalty for every set of MAP parameters as in [16] as 'Per-task Laplace'. The per-task Laplace method with a diagonal approximation to the Hessian corresponds to EWC.

We find our online Laplace approximation to maintain higher test accuracy throughout training than placing a quadratic penalty around the MAP parameters of every task, in particular when using a simple diagonal approximation to the Hessian. However, the main difference between the methods lies in using a Kronecker factored approximation of the curvature over a diagonal one. Using this approximation, we achieve over 90% average test accuracy across 50 tasks, almost matching the performance of a network trained jointly on all observed data. Recalculating the curvature for each task instead of retaining previous estimates does not significantly affect performance.

Beyond simple average performance, we investigate different values of the hyperparameter λ on the permuted MNIST sequence of datasets for our online Laplace approximation. The goal is to visualize how it affects the trade-off between remembering previous tasks and being able to learn new ones for the two approximations of the curvature that we consider. Fig. 2 shows various statistics of the accuracy on the test set for the smallest and largest value of the hyperparameter on the quadratic penalty that we tested, as well as the one that optimizes the validation error.

We are particularly interested in the performance on the first dataset and the most recent one, as a measure for memory and flexibility respectively. For all displayed values of the hyperparameter, the Kronecker factored approximation (Fig. 2a) has higher test accuracy than the diagonal approximation (Fig. 2b) on both the most recent and the first task, as well as on average. For the natural choice of $\lambda = 1$ (leftmost subfigure respectively), the network's performance decays for the first task for both curvature approximations, yet it is able to learn the most recent task well. The performance on the first task decays more slowly, however, for the more expressive Kronecker factored approximation of the curvature. Increasing the hyperparameter, corresponding to making the prior more narrow as discussed in Section 2.4, leads to the network remembering the first task much better at the cost of not being able to achieve optimal performance on the most recently added task. Using $\lambda = 3$ (central subfigure), the value that achieves optimal validation error in our experiments, the Kronecker factored approximation leads to the network performing similarly on the most recent and first tasks. This coincides with optimal average test accuracy. We are not able to find such an ideal trade-off for the diagonal Hessian approximation, resulting in worse average performance and suggesting that the posterior cannot be matched well without accounting for interactions between the weights. Using a large value of $\lambda = 100$ (rightmost subfigure) reverts the order of performance between the most recent and the first task for both approximations: while for small λ the first task is 'forgotten', the network's performance now stays at a high level — for the Kronecker factored approximation it remembers it perfectly — which comes at the cost of being unable to learn new tasks well.

We conclude from our results that the online Laplace approximation overestimates the uncertainty in the approximate posterior about the parameters for the permuted MNIST task, in particular with a

diagonal approximation to the Hessian. Overestimating the uncertainty leads to a need for regularization in the form of reducing the width of the approximate posterior, as the value that optimizes the validation error is $\lambda = 3$. Only when regularizing too strongly the approximate posterior underestimates the uncertainty about the weights, leading to reduced performance on new tasks for large values of λ. Using a better approximation to the posterior leads to a drastic increase in performance and a reduced need for regularization in the subsequent experiments. We note that some regularization is still necessary, suggesting that even the Kronecker factored approximation overestimates the variance in the posterior, and a better approximation could lead to further improvements. However, it is also possible that the Laplace approximation as such requires a large amount of data to estimate the interaction between the parameters sufficiently well; hence it might be best suited for settings where plenty of data are available.

4.2 Disjoint MNIST

We further experiment with the disjoint MNIST task, which splits the MNIST dataset into one part containing the digits '0' to '4', and a second part containing '5' to '9' and training a ten-way classifier on each set separately. Previous work [20] has found this problem to be challenging for EWC, as during the first half of training the network is encouraged to set the bias terms for the second set of labels to highly negative values. This setup makes it difficult to balance out the biases for the two sets of classes after the first task without overcorrecting and setting the biases for the first set of classes to highly negative values. Lee *et al.* [20] report just over 50% test accuracy for EWC, which corresponds to either completely forgetting the first task or being unable to learn the second one, as each task individually can be solved with around 99% accuracy.

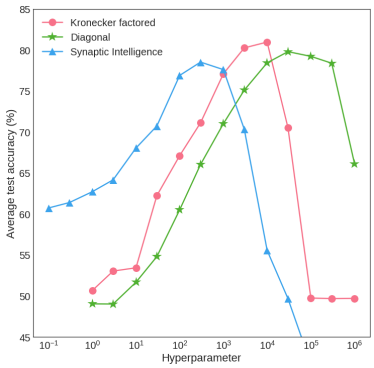


Figure 3: Disjoint MNIST test accuracy for the Laplace approximation (hyperparameter: λ) and SI (hyperparameter: c). 'Kronecker factored' and 'Diagonal' refer to the respective curvature approximation for the Laplace method.

We use an identical network architecture to the previous section and found stronger regularization of the approximate posterior to be necessary. For the Laplace methods, we tested values of $\lambda \in \{1, 3, 10, \ldots, 3{\times}10^5, 10^6\}$, and $c \in \{0.1, 0.3, 1, \ldots, 3{\times}10^4, 10^5\}$ for SI. We train using Nesterov momentum with a learning rate of 0.1 and momentum of 0.9 and decay the learning rate by a factor of 10 every 1000 parameter updates using a batch size of 250. We decay the initial learning rate for the second task depending on the hyperparameter to prevent the objective from diverging. We test various decay factors for each hyperparameter, but as a rule of thumb found $\frac{\lambda}{10}$ to perform well for the Kronecker factored, and $\frac{\lambda}{1000}$ for the diagonal approximation. The results are averaged across ten independent runs.

Fig. 3 shows the test accuracy for various hyperparameter values for a Kronecker factored and a diagonal approximation of the curvature as well as SI. As there are only two datasets, the three Laplace-based methods are identical, therefore we focus on the impact of the curvature approximation. Approximating the Hessian with a diagonal corresponds to EWC. While we do not match the performance of the method developed in [20], we find the Laplace approximation to work significantly better than reported by the authors. The Kronecker factored approximation gives a small improvement over the diagonal one and requires weaker regularization, which further suggests that it better fits the true posterior. It also outperforms SI.

4.3 Vision datasets

As a final experiment, we test our method on a suite of related vision datasets. Specifically, we train and test on MNIST [19], notMNIST[3], Fashion MNIST [40], SVHN [29] and CIFAR10 [18] in this order. All five datasets contain around $50,000$ training images from 10 different classes. MNIST contains hand-written digits from '0' to '9', notMNIST the letters 'A' to 'J' in different computer

[3]Originally published at `http://yaroslavvb.blogspot.co.uk/2011/09/notmnist-dataset.html` and downloaded from `https://github.com/davidflanagan/notMNIST-to-MNIST`

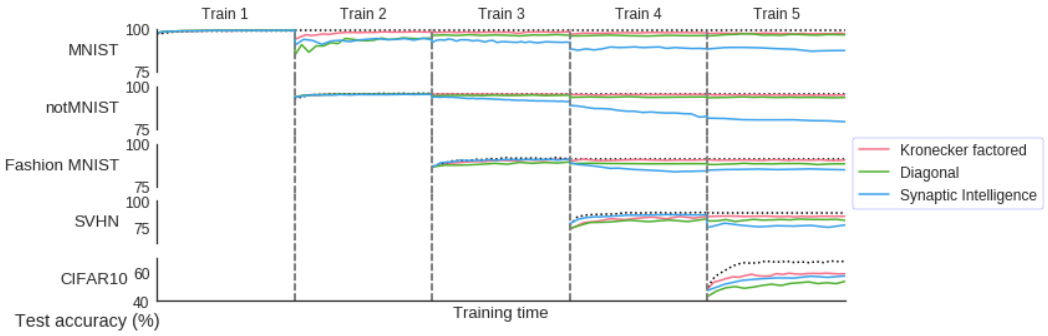


Figure 4: Test accuracy of a convolutional network on a sequence of vision datasets. We train on the datasets separately in the order displayed from top to bottom and show the network's accuracy on each dataset once training on it has started. The dotted black line indicates the performance of a network with the same architecture trained separately on the task. The diagonal and Kronecker factored approximation to the Hessian both use our online Laplace method to prevent forgetting.

fonts, Fashion MNIST different categories of clothing, SVHN the digits '0' to '9' on street signs and CIFAR10 ten different categories of natural images. We zero-pad the images of the MNIST-like datasets to be of size 32×32 and replicate their intensity values over three channels, such that all images have the same format.

We train a LeNet-like architecture [19] with two convolutional layers with 5×5 convolutions with 20 and 50 channels respectively and a fully connected hidden layer with 500 units. We use ReLU nonlinearities and perform a 2×2 max-pooling operation after each convolutional layer with stride 2. An extension of the Kronecker factored curvature approximations to convolutional neural networks is presented in [10]. As the meaning of the classes in each dataset is different, we keep the weights of the final layer separate for each task. We optimize the networks as in the permuted MNIST experiment and compare to five baseline networks with the same architecture trained on each task separately.

Overall, the online Laplace approximation in conjunction with a Kronecker factored approximation of the curvature achieves the highest test accuracy across all five tasks (see Appendix E for the numerical results). However, the difference between the three Laplace-based methods is small in comparison to the improvement stemming from the better approximation to the Hessian. We therefore plot the test accuracy curves through training only for the online Laplace approximation in the main text in Fig. 4 to show the difference to SI and between the two curvature approximations. The corresponding figures for having a separate quadratic penalty for each task and the approximate Laplace approximation are in Appendix F.

Using a diagonal Hessian approximation for the Laplace approximation, the network mostly remembers the first three tasks, but has difficulties learning the fifth one. SI, in contrast, shows decaying performance on the initial tasks, but learns the fifth task almost as well as our method with a Kronecker factored approximation of the Hessian. However, using the Kronecker factored approximation, the network achieves good performance relative to the individual networks across all five tasks. In particular, it remembers the easier early tasks almost perfectly while being sufficiently flexible to learn the more difficult later tasks better than the diagonal methods, which suffer from forgetting.

5 Conclusion

We proposed the online Laplace approximation, a Bayesian online learning method for overcoming catastrophic forgetting in neural networks. By formulating a principled approximation to the posterior, we were able to substantially improve over EWC [16] and SI [41], two recent methods that also add a quadratic regularizer to the objective for new tasks. By further taking interactions between the parameters into account, we achieved considerable increases in test accuracy on the problems that we investigated, in particular for long sequences of datasets. Our results demonstrate the importance of going beyond diagonal approximation methods which only measure the sensitivity of individual parameters. Dealing with the complex interaction and correlation between parameters is necessary in moving towards a more complete response to the challenge of continual learning.

References

[1] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight Uncertainty in Neural Networks. In *International Conference on Machine Learning*, pages 1613–1622, 2015.

[2] A. Botev, H. Ritter, and D. Barber. Practical Gauss-Newton Optimisation for Deep Learning. In *International Conference on Machine Learning*, pages 557 – 565, 2017.

[3] S. Dieleman, J. Schlüter, C. Raffel, E. Olson, S. K. Sønderby, D. Nouri, et al. Lasagne: First release., August 2015.

[4] E. Eskin, A. J. Smola, and S. Vishwanathan. Laplace Propagation. In *Advances in Neural Information Processing Systems*, pages 441–448, 2004.

[5] 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.

[6] R. M. French. Catastrophic Forgetting in Connectionist Networks. *Trends in Cognitive Sciences*, 3:128–135, 1999.

[7] Z. Ghahramani. Online Variational Bayesian Learning. 2000. Slides from talk presented at NIPS 2000 workshop on Online Learning.

[8] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio. An Empirical Investigation of Catastrophic Forgetting in Gradient-based Neural Networks. *arXiv preprint arXiv:1312.6211*, 2013.

[9] E. Grant, C. Finn, S. Levine, T. Darrell, and T. Griffiths. Recasting Gradient-Based Meta-Learning as Hierarchical Bayes. In *International Conference on Learning Representations*, 2018.

[10] R. Grosse and J. Martens. A Kronecker-factored Approximate Fisher Matrix for Convolution Layers. In *International Conference on Machine Learning*, pages 573–582, 2016.

[11] A. K. Gupta and D. K. Nagar. *Matrix Variate Distributions*, volume 104. CRC Press, 1999.

[12] X. He and H. Jaeger. Overcoming Catastrophic Interference using Conceptor-Aided Backpropagation. In *International Conference on Learning Representations*, 2018.

[13] A. Honkela and H. Valpola. On-line Variational Bayesian Learning. In *4th International Symposium on Independent Component Analysis and Blind Signal Separation*, pages 803–808, 2003.

[14] F. Huszár. Note on the Quadratic Penalties in Elastic Weight Consolidation. *Proceedings of the National Academy of Sciences*, 2018.

[15] D. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[16] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell. Overcoming Catastrophic Forgetting in Neural Networks. *Proceedings of the National Academy of Sciences*, pages 3521–3526, 2017.

[17] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell. Reply to Huszár: The Elastic Weight Consolidation Penalty is Empirically Valid. *Proceedings of the National Academy of Sciences*, 2018.

[18] A. Krizhevsky and G. Hinton. Learning Multiple Layers of Features from Tiny Images. 2009.

[19] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based Learning Applied to Document Recognition. In *Proceedings of the IEEE*, pages 2278 – 2324, 1998.

[20] S.-W. Lee, J.-H. Kim, J. Jun, J.-W. Ha, and B.-T. Zhang. Overcoming Catastrophic Forgetting by Incremental Moment Matching. In *Advances in Neural Information Processing Systems*, pages 4655–4665, 2017.

[21] D. Lopez-Paz and M. Ranzato. Gradient Episodic Memory for Continual Learning. In *Advances in Neural Information Processing Systems*, pages 6470–6479, 2017.

[22] D. J. C. MacKay. A Practical Bayesian Framework for Backpropagation Networks. *Neural Computation*, 4:448–472, 1992.

[23] J. Martens and R. Grosse. Optimizing Neural Networks with Kronecker-factored Approximate Curvature. In *International Conference on Machine Learning*, pages 2408–2417, 2015.

[24] J. Martens. New Insights and Perspectives on the Natural Gradient Method. *arXiv preprint arXiv:1412.1193*, 2014.

[25] P. Maybeck. *Stochastic Models, Estimation and Control*, chapter 12.7. Academic Press, 1982.

[26] M. McCloskey and N. J. Cohen. Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. *Psychology of Learning and Motivation - Advances in Research and Theory*, 24:109–165, 1989.

[27] T. P. Minka. Expectation Propagation for Approximate Bayesian Inference. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, pages 362–369, 2001.

[28] Y. Nesterov. A Method of Solving a Convex Programming Problem with Convergence Rate O (1/k2). *Soviet Mathematics Doklady*, 27:372–376, 1983.

[29] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading Digits in Natural Images with Unsupervised Feature Learning. In *NIPS workshop on deep learning and unsupervised feature learning*, page 5, 2011.

[30] C. V. Nguyen, Y. Li, T. D. Bui, and R. E. Turner. Variational Continual Learning. *International Conference on Learning Representations*, 2018.

[31] M. Opper and O. Winther. A Bayesian Approach to On-line Learning. *On-line Learning in Neural Networks, ed. D. Saad*, pages 363–378, 1998.

[32] B. T. Polyak. Some Methods of Speeding up the Convergence of Iteration Methods. *USSR Computational Mathematics and Mathematical Physics*, 4:1–17, 1964.

[33] R. Ratcliff. Connectionist Models of Recognition Memory: Constraints Imposed by Learning and Forgetting Functions. *Psychological Review*, 97:285–308, 1990.

[34] H. Ritter, A. Botev, and D. Barber. A Scalable Laplace Approximation for Neural Networks. *International Conference on Learning Representations*, 2018.

[35] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive Neural Networks. *arXiv preprint arXiv:1606.04671*, 2016.

[36] L. Sagun, U. Evci, V. U. Guney, Y. Dauphin, and L. Bottou. Empirical Analysis of the Hessian of Over-Parametrized Neural Networks. *arXiv preprint arXiv:1706.04454*, 2017.

[37] J. Serrà, D. Surís, M. Miron, and A. Karatzoglou. Overcoming Catastrophic Forgetting with Hard Attention to the Task. *arXiv preprint arXiv:1801.01423*, 2018.

[38] H. Shin, J. K. Lee, J. Kim, and J. Kim. Continual Learning with Deep Generative Replay. *arXiv preprint arXiv:1705.08690*, 2017.

[39] Theano Development Team. Theano: A Python Framework for Fast Computation of Mathematical Expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[40] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[41] F. Zenke, B. Poole, and S. Ganguli. Continual Learning through Synaptic Intelligence. In *International Conference on Machine Learning*, pages 3987–3995, 2017.

A Derivation of the Kronecker factorization of the diagonal blocks of the Hessian

Martens and Grosse [23] and Botev *et al.* [2] both develop block-diagonal Kronecker factored approximations to the Fisher and Gauss-Newton matrix of fully connected neural networks respectively, which in turn both are positive semi-definite approximations of the Hessian. Both use their approximations for optimization, hence the positive semi-definiteness is crucial in order to prevent parameter updates that increase the loss. We require this property as well, as we perform a Laplace approximation and the Normal distribution requires its covariance to be positive semi-definite.

In the following, we provide the basic derivation for the diagonal blocks of the Hessian being Kronecker factored as developed in [2] and state the recursion for calculating the pre-activation Hessian.

We denote a neural network as taking an input $a_0 = x$ and producing an output h_L. The input is passed through layers $l = 1, \dots, L$ as linear pre-activations $h_l = W_l a_{l-1}$ and non-linear activations $a_l = f_l(h_l)$, where W_l denotes the weight matrix and f_l the elementwise activation function. Bias terms can be absorbed into W_l by appending a 1 to every a_l. The weights are optimized w.r.t. an error function $E(y, h_L)$, which can usually be expressed as a negative log likelihood.

Using the chain rule, the gradient of the error function w.r.t. an individual weight can be calculated as:

$$\frac{\partial E}{\partial W^l_{a,b}} = \sum_i \frac{\partial h^l_i}{\partial W^l_{a,b}} \frac{\partial E}{\partial h^l_i} = a^{l-1}_b \frac{\partial E}{\partial h^l_a} \tag{12}$$

Differentiating again w.r.t. another weight within the same layer gives:

$$[H_l]_{(a,b),(c,d)} \equiv \frac{\partial^2 E}{\partial W_{a,b} \partial W_{c,d}} = a^{l-1}_b a^{l-1}_d \, [\mathcal{H}_l]_{(a,c)} \tag{13}$$

where

$$[\mathcal{H}_l]_{a,b} \equiv \frac{\partial^2 E}{\partial h^l_a \partial h^l_b} \tag{14}$$

is defined to be the pre-activation Hessian.

This can also be expressed in matrix notation as a Kronecker product:

$$H_l = \frac{\partial^2 E}{\partial \operatorname{vec}(W^l) \partial \operatorname{vec}(W^l)} = \left(a_{l-1} a_{l-1}^\top\right) \otimes \mathcal{H}_l \tag{15}$$

Similar to backpropagation, the pre-activation Hessian can be calculated as:

$$\mathcal{H}_l = B_l W_{l+1}^\top \mathcal{H}_{l+1} W_{l+1} B_l + D_l \tag{16}$$

where the diagonal matrices B_l and D_l are defined as

$$B_l = \operatorname{diag}(f_l'(h_l)) \tag{17}$$

$$D_l = \operatorname{diag}(f_l''(h_l) \frac{\partial E}{\partial a_l}) \tag{18}$$

f' and f'' denote the first and second derivative of f. The recursion for $\mathcal{H}$ is initialized with the Hessian of the error w.r.t. the network outputs, i.e. $\mathcal{H}_L \equiv \frac{\partial^2 E}{\partial h_L \partial h_L}$. For the derivation of the recursion and how to calculate the diagonal blocks of the Gauss-Newton matrix, we refer the reader to [2], and to [23] for the Fisher matrix.

B Visualization of the effect of λ for a Gaussian prior and posterior

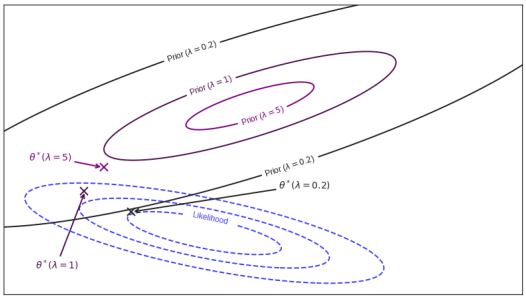


Figure 5: Contours of a Gaussian likelihood (dashed blue) and prior (shades of purple) for different values of λ. Values smaller than 1 shift the joint maximum θ^*, marked by a '$\times$',towards that of the likelihood, i.e. the new task, for values greater than 1 it moves towards the prior, i.e. previous tasks.

A small λ resulting in high uncertainty shifts the mode towards that of the likelihood, i.e. enables the network to learn the new task well even if our posterior approximation underestimates the uncertainty. Vice versa, increasing λ moves the joint mode towards the prior mode, improving how well the previous parameters are remembered. The optimal choice depends on the true posterior and how closely it is approximated.

In principle, it would be possible to use a different value λ_t for every dataset. In our experiments, we keep the value of λ the same across all tasks as the family of posterior approximation is the same throughout training. Furthermore, using a separate hyperparameter for each task would let the number of hyperparameters grow linearly in the number of tasks, which would make tuning them costly.

C Additional related work

Various methods for overcoming catastrophic forgetting without a Bayesian motivation have also been proposed over the past year. Zenke *et al.* [41] develop 'Synaptic Intelligence' (SI), another quadratic penalty on deviations from previous parameter values where the importance of each weight is heuristically measured as the path length of the updates on the previous task. Lopez-Paz and Ranzato [21] formulate a quadratic program to project the gradients such that the gradients on previous tasks do not point in a direction that decreases performance; however, this requires keeping some previous data in memory. Shin *et al.* [38] suggest a dual architecture including a generative model that acts as a memory for data observed in previous tasks. Other approaches that tackle the problem at the level of the model architecture include [35], which augments the model for every new task, and [5], which trains randomly selected paths through a network. Serrà *et al.* [37] propose sharing a set of weights and modifying them in a learnable manner for each task. He and Jaeger [12] introduce conceptor-aided backpropagation to shield gradients against reducing performance on previous tasks.

D Optimization details

For the permuted MNIST experiment, we found the performance of the methods that we compared to mildly depend on the choice of optimizer. Therefore, we optimize all techniques with Adam [15] for 20 epochs per dataset and a learning rate of 10^{-3} as in [41], SGD with momentum [32] with an initial learning rate of 10^{-2} and 0.95 momentum, and Nesterov momentum [28] with an initial learning rate of 0.1, which we divide by 10 every 5 epochs, and 0.9 momentum. For the momentum based methods, we train for at least 10 epochs and early-stop once the validation error does not improve for 5 epochs. Furthermore, we decay the initial learning rate with a factor of $\frac{1}{1+kt}$ for the momentum-based optimizers, where t is the index of the task and k a decay constant. We set k using a coarse grid search for each value of the hyperparameter λ in order to prevent the objective from diverging towards the end of training, in particular with the Kronecker factored curvature approximation. For the Laplace approximation based methods, we consider $\lambda \in \{1, 3, 10, 30, 100\}$; for SI we try $c \in \{0.01, 0.03, 0.1, 0.3, 1\}$. We ultimately pick the combination of optimizer, hyperparameter and

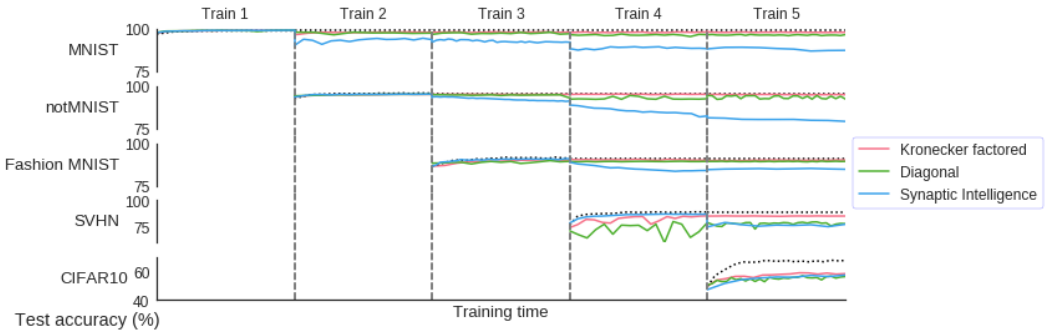
decay rate that gives the best validation error across all tasks at the end of training. For the Laplace-based methods, we found momentum based optimizers to lead to better performance, whereas Adam gave better results for SI.

E Numerical results of the vision experiment

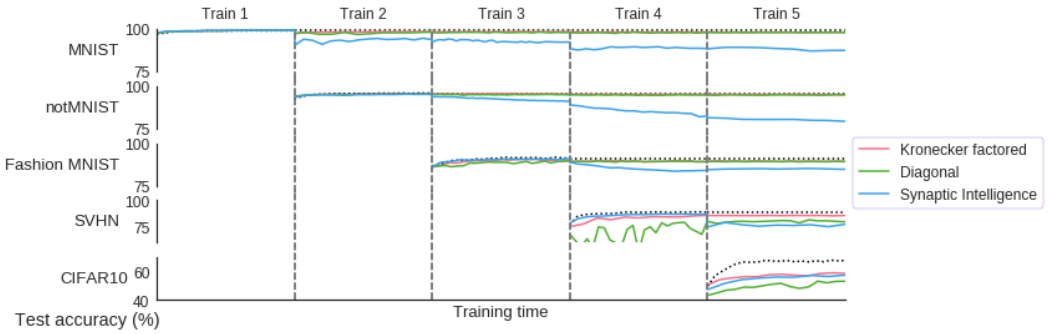
Table 1: Per dataset test accuracy at the end of training on the suite of vision datasets. SI is Synaptic Intelligence [41] and EWC Elastic Weight Consolidation [16]. We abbreviate Per-Task Laplace (one penalty per task) as PTL, Approximate Laplace (Laplace approximation of the full posterior at the mode of the approximate objective) and our Online Laplace approximation as OL. nMNIST refers to notMNIST, fMNIST to FashionMNIST and C10 to CIFAR10.

		Test Error (%)					
Method	Approximation	MNIST	nMNIST	fMNIST	SVHN	C10	Avg.
SI	n/a	87.27	79.12	84.61	77.44	57.61	77.21
PTL	Diagonal (EWC)	97.83	94.73	89.13	79.80	53.29	82.96
	Kronecker factored	97.85	94.92	89.31	85.75	58.78	85.32
AL	Diagonal	96.56	92.33	89.27	78.00	56.57	82.55
	Kronecker factored	97.90	94.88	90.08	85.24	58.63	85.35
OL	Diagonal	96.48	93.41	88.09	81.79	53.80	82.71
	Kronecker factored	97.17	94.78	90.36	85.59	59.11	85.40

F Additional figures for the vision experiment



(a) Approximate Laplace



(b) Per-task Laplace

Figure 6: Test accuracy of a convolutional network on a sequence of vision datasets for different methods for preventing catastrophic forgetting. We train on the datasets separately in the order displayed from top to bottom and show the network's accuracy on each dataset once training on it has started. The dotted black line indicates the performance of a network with the same architecture trained separately on the task.