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

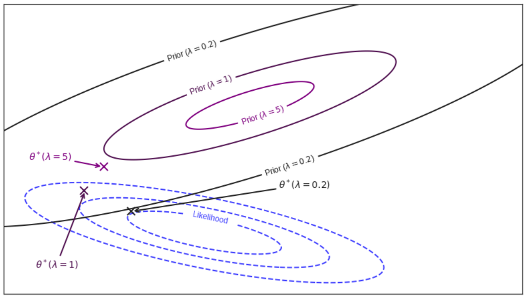


Figure 5: Contours of a Gaussian likelihood (dashed blue) and prior (shades of purple) for different values of λ. Values smaller than 1 shift the joint maximum θ^*, marked by a '$\times$',towards that of the likelihood, i.e. the new task, for values greater than 1 it moves towards the prior, i.e. previous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

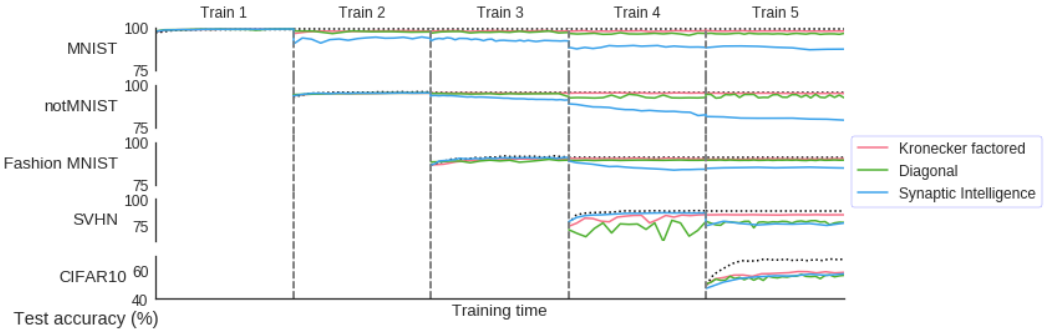
For the permuted MNIST experiment, we found the performance of the methods that we compared to mildly depend on the choice of optimizer. Therefore, we optimize all techniques with Adam [15] for 20 epochs per dataset and a learning rate of 10^{-3} as in [41], SGD with momentum [32] with an initial learning 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

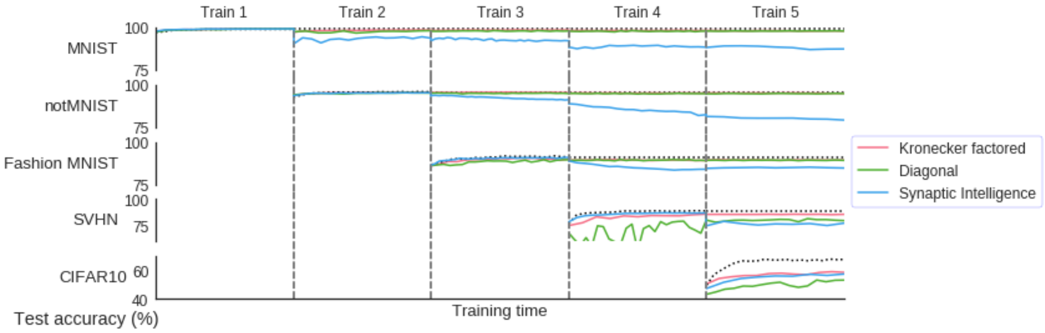
Table 1: Per dataset test accuracy at the end of training on the suite of vision datasets. SI is Synaptic Intelligence [41] and EWC Elastic Weight Consolidation [16]. We abbreviate Per-Task Laplace (one penalty per task) as PTL, Approximate Laplace (Laplace approximation of the full posterior at the 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.