This CVPR 2020 paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

Continual Learning with Extended Kronecker-factored Approximate Curvature

Janghyeon Lee[1] Hyeong Gwon Hong[2] Donggyu Joo[1] Junmo Kim[1,2]
[1]School of Electrical Engineering, KAIST
[2]Graduate School of AI, KAIST
{wkdgus9305, honggudrnjs, jdg105, junmo.kim}@kaist.ac.kr

Abstract

We propose a quadratic penalty method for continual learning of neural networks that contain batch normalization (BN) layers. The Hessian of a loss function represents the curvature of the quadratic penalty function, and a Kronecker-factored approximate curvature (K-FAC) is used widely to practically compute the Hessian of a neural network. However, the approximation is not valid if there is dependence between examples, typically caused by BN layers in deep network architectures. We extend the K-FAC method so that the inter-example relations are taken into account and the Hessian of deep neural networks can be properly approximated under practical assumptions. We also propose a method of weight merging and reparameterization to properly handle statistical parameters of BN, which plays a critical role for continual learning with BN, and a method that selects hyperparameters without source task data. Our method shows better performance than baselines in the permuted MNIST task with BN layers and in sequential learning from the ImageNet classification task to fine-grained classification tasks with ResNet-50, without any explicit or implicit use of source task data for hyperparameter selection.

1. Introduction

An artificial neural network catastrophically forgets about what it has learned from previous tasks in a sequential learning scenario as it is optimized to solve its current target problem only [6, 25]. Although many continual learning methods that aim to mitigate catastrophic forgetting have been introduced in recent years, their use in real-world applications remains somewhat limited owing to their unclear hyperparameter settings or their use of out-of-date network architectures. Specifically, many of the current continual learning methods do not explain how to determine hyperparameters, or experiments are usually performed with hyperparameters carefully tuned through a validation over whole tasks including past tasks, which is generally not possible in practice [29]. Further, even though most state-of-the-art deep network architectures [11, 12] cannot be easily trained without batch normalization (BN) layers [14], the effect of BN is usually not considered and out-of-date network architectures which do not require BN are still mostly used to evaluate continual learning methods.

BN is quite tricky to manipulate due to training and evaluation discrepancy, and its statistical parameters, means and variances of activations. Those statistical parameters are naturally model's parameters learned from training data, but they are not free parameters controlled by gradient descent; they are just determined by the distribution of data. Especially, in a sequential learning scenario, they are vulnerable to get beyond control as we are dealing with multiple different tasks whose distributions are not the same in general. Thus, the effect of BN must be discussed and considered carefully.

In this paper, we propose a quadratic penalty method with a Hessian approximation, for continual learning of neural networks that contain BN layers. Before delving into the detailed method, let us briefly introduce and review how Hessian can be used for continual learning. For sequential learning, one of the oracle methods which gives an upper bound performance is the multitask learning method described as the following optimization problem,

$$\min_{\theta} \quad \lambda_s \mathbb{E}_{(x,t)\in\mathcal{D}_s}[\mathcal{L}(x,t;\theta)] + \lambda_t \mathbb{E}_{(x,t)\in\mathcal{D}_t}[\mathcal{L}(x,t;\theta)], \tag{1}$$

where $\mathcal{D}_s$ and $\mathcal{D}_t$ are source and target training dataset, respectively; $\mathcal{L}$ is a loss function; λ_s and λ_t are importance hyperparameters that control how important each task is; and θ is a vectorized model's parameter. As $\mathcal{D}_s$ is not available in a sequential learning scenario, we consider approximating the source task loss function with a function of only θ, excluding data x and label t. In particular, because target task optimization starts with a source task solution θ^* as the initial position, it is natural to take the second-order Taylor series approximation of the source task loss function at $\theta = \theta^*$; one advantage of this is that the first-order term disappears because the gradient at a local minimum is zero.

Thus, if we denote the gradient and Hessian of the source task loss function at $\theta = \theta^*$ averaged over the source dataset as

$$g = \mathbb{E}_{(x,t)\in\mathcal{D}_s}\left[\left.\frac{\partial\mathcal{L}(x,t;\theta)}{\partial\theta}\right|_{\theta=\theta^*}\right], \tag{2}$$

$$H = \mathbb{E}_{(x,t)\in\mathcal{D}_s}\left[\left.\frac{\partial^2\mathcal{L}(x,t;\theta)}{\partial\theta\partial\theta}\right|_{\theta=\theta^*}\right], \tag{3}$$

then

$$\begin{aligned}&\mathbb{E}_{(x,t)\in\mathcal{D}_s}[\mathcal{L}(x,t;\theta)]\\ &\simeq \frac{1}{2}(\theta-\theta^*)^\top H(\theta-\theta^*) + g^\top(\theta-\theta^*) + C\\ &= \frac{1}{2}(\theta-\theta^*)^\top H(\theta-\theta^*) + C\end{aligned} \tag{4}$$

where C is some constant. The gradient g is simply a zero vector because θ^* is a local minimum of the source task loss function. Therefore, combining Equations 1 and 4, we get the following continual learning optimization problem which approximates the multitask learning objective.

$$\min_\theta \quad \lambda_t\mathbb{E}_{(x,t)\in\mathcal{D}_t}[\mathcal{L}(x,t;\theta)] + \frac{1}{2}\lambda_s(\theta-\theta^*)^\top H(\theta-\theta^*) \tag{5}$$

The former term is a usual training loss for the target task and the latter is a quadratic penalty term centered at the initial point θ^* with the curvature of H.

This approximation of the multitask learning objective was used for Hessian pseudopattern backpropagation [7], and it is actually the same as elastic weight consolidation (EWC) [15] and online structured Laplace approximation (OSLA) [31] except that they are grounded in Bayesian approaches. EWC uses diagonal entries of the Hessian only, and later, in OSLA, a Kronecker-factored block diagonal Hessian approximation is adopted to enable a significant improvement in the performance from EWC. However, in EWC and OSLA, the effect of BN was not considered so only shallow networks were used for evaluation, and hyperparameters were found by validations over whole tasks.

We describe our contributions in two parts. In Section 3, for theoretical background, we demonstrate that the current Hessian approximation method is not valid when a network has BN layers as dependence between examples caused by BN is not considered, and we extend the Hessian approximation method by taking into account the inter-example relations, so that the Hessian of such network can be validly approximated. In Section 4, for actual continual learning, we propose the detailed methods including how to apply the proposed quadratic penalty loss with BN layers, how to handle statistical parameters, and how to select hyperparameters without source task data.

2. Related work

Curvature approximation. Various methods for Kronecker factorization of curvature have been studied over the past few years. [24] proposes an efficient method for approximating natural gradient descent in neural networks, Kronecker-factored approximate curvature (K-FAC), which is derived by approximating blocks of the Fisher as being the Kronecker product of two much smaller matrices. Similarly, [4] presents an efficient block diagonal approximation to the Gauss–Newton matrix for second order optimization of neural networks. [2] develops a version of K-FAC that distributes the computation of gradients and additional quantities required by K-FAC across multiple machines. In [10], a tractable approximation to the Fisher matrix for convolutional networks, Kronecker factors for convolution (KFC), is derived based on a structured probabilistic model for the distribution over backpropagated derivatives. For recurrent neural networks, [23] extends the K-FAC method by modelling the statistical structure between the gradient contributions at different time-steps. [8] introduces Eigenvalue-corrected Kronecker factorization, an approximate factorization of the Fisher that is computationally manageable, accurate, and amendable to cheap partial updates.

3. Curvature approximation

3.1. Notation

The l-th learnable linear layer in any feedforward neural network can be described as

$$h_l = W_l\bar{a}_{l-1}, \qquad l = 1, 2, \cdots, L \tag{6}$$

where W_l is a $C_l \times (C_{l-1}+1)$ weight matrix and $\bar{a}_{l-1}$ is a $(C_{l-1}+1)\times N$ activation matrix with a homogeneous dimension appended; in other words, bias terms are absorbed into W_l by appending an all-ones vector to a_{l-1}. The whole network takes $x = a_0$ as its input of size $C_0 \times N$ and produces the corresponding output y of size $C_L \times N$, where N is the size of a mini-batch. We denote the loss of the n-th example in a random mini-batch sample (x,t) as $\mathcal{L}_n(x,t)$, or simply $\mathcal{L}_n$, when it is clear from the context, so that we can distinguish between an example index n and a mini-batch sample (x,t). Thus, the optimization objective can be written as $\mathbb{E}_{(x,t)}[\mathbb{E}_n[\mathcal{L}_n]]$. In this paper, we denote the (i,j)-th entry of a matrix A as $(A)_{i,j}$, and the (i,j)-th block of a block matrix A as $\{A\}_{i,j}$.

3.2. K-FAC

The Hessian of the linear layer described by Equation 6, considering all inter-example relations, is represented by the following theorem.

Theorem 1. ***(The Hessian of a linear layer)***

$$\frac{\partial^2 \mathcal{L}_n}{\partial (W_l)_{a,b} \partial (W_{l'})_{c,d}} = \sum_{m,m'} (\bar{a}_{l-1})_{b,m} (\bar{a}_{l'-1})_{d,m'} \frac{\partial^2 \mathcal{L}_n}{\partial (h_l)_{a,m} \partial (h_{l'})_{c,m'}} \quad (7)$$

Proof. See Appendix A.2. □

If a network does not have any BN layers, then $\mathcal{L}_n$ depends on only the n-th example, and therefore,

$$\mathbb{E}_{(x,t)} \left[\mathbb{E}_n \left[\frac{\partial^2 \mathcal{L}_n}{\partial (W_l)_{a,b} \partial (W_{l'})_{c,d}} \right] \right] = \mathbb{E}_{(x,t)} \left[\mathbb{E}_n \left[(\bar{a}_{l-1})_{b,n} (\bar{a}_{l'-1})_{d,n} \frac{\partial^2 \mathcal{L}_n}{\partial (h_l)_{a,n} \partial (h_{l'})_{c,n}} \right] \right] . \quad (8)$$

As calculating and saving all entries of the Hessian is infeasible in practice owing to its size even for a simple network (*e.g.*, the Hessian of a 1024×1024 fully connected layer takes 4 TB in memory with single-precision floating-point numbers), the K-FAC method factors the Hessian into two relatively small matrices [4, 10, 23, 24] so that Equation 8 is approximated by

$$\mathbb{E}_x \left[\mathbb{E}_n \left[(\bar{a}_{l-1})_{b,n} (\bar{a}_{l'-1})_{d,n} \right] \right] \cdot \mathbb{E}_{(x,t)} \left[\mathbb{E}_n \left[\frac{\partial^2 \mathcal{L}_n}{\partial (h_l)_{a,n} \partial (h_{l'})_{c,n}} \right] \right] . \quad (9)$$

This approximation assumes that there would be a small correlation between the two very different variables, one from activations and the other from gradients, or their joint distribution is well approximated by a multivariate Gaussian so that their higher-order joint cumulants would be small [24]. If we define block matrices H, $\bar{A}$, and $\mathcal{H}$ as

$$(\{H\}_{l,l'})_{i,j} = \mathbb{E}_{(x,t)} \left[\mathbb{E}_n \left[\frac{\partial^2 \mathcal{L}_n}{\partial (\text{vec}(W_l))_i \partial (\text{vec}(W_{l'}))_j} \right] \right] , \quad (10)$$

$$(\{\bar{A}\}_{l,l'})_{b,d} = \mathbb{E}_x \left[\mathbb{E}_n \left[(\bar{a}_{l-1})_{b,n} (\bar{a}_{l'-1})_{d,n} \right] \right] , \quad (11)$$

$$(\{\mathcal{H}\}_{l,l'})_{a,c} = \mathbb{E}_{(x,t)} \left[\mathbb{E}_n \left[\frac{\partial^2 \mathcal{L}_n}{\partial (h_l)_{a,n} \partial (h_{l'})_{c,n}} \right] \right] , \quad (12)$$

then Equation 9, K-FAC, can be written in a simple block matrix form [4, 24] as

$$H \approx \bar{A} * \mathcal{H} \quad (13)$$

where $*$ is the Khatri–Rao product.

3.3. Extended K-FAC

Unfortunately, for a network with BN layers, each example in a mini-batch affects each other, and therefore, $\frac{\partial^2 \mathcal{L}_n}{\partial (h_l)_{a,m} \partial (h_{l'})_{c,m'}}$ is not zero in general, even if $m \neq n$ and $m' \neq n$, and all summands in Equation 7 remain to be computed. If each summand is approximated by K-FAC, it costs the amount of memory and computation about N^2 times more; this could be prohibitive for a large mini-batch size. Hence, we need to take a closer look at and make use of the properties of Equation 7 and BN.

For the n-th example, note that the other examples are statistically indistinguishable if mini-batches are sampled uniformly from the dataset. For example, for any $n' \neq n$, let x' be the mini-batch obtained by interchanging the n-th and n'-th example indices of x. Then, for any function f,

$$\mathbb{E}_x[f((\bar{a}_{l-1})_{b,n})] = \mathbb{E}_{x'}[f((\bar{a}_{l-1})_{b,n'})] \quad (14)$$

$$= \mathbb{E}_x[f((\bar{a}_{l-1})_{b,n'})], \quad (15)$$

where the second equality comes from the fact that $\mathbb{E}_x = \mathbb{E}_{x'}$ when all possible mini-batches of the dataset are sampled equally likely.

Thus, the N^2 summands in Equation 7 can be divided into five statistically distinct groups: (i) $m = m' = n$, (ii) $m = m' \neq n$, (iii) $m = n \neq m'$, (iv) $m \neq n = m'$, and (v) $n \neq m \neq m' \neq n$. If we apply K-FAC in each of the five groups, then we get the following theorem.

Theorem 2. ***(Extended K-FAC)*** *Let π and π' be permutations of $\{1, 2, \cdots, N\}$, and assume that mini-batches are sampled equally likely from all possible mini-batches of the dataset. If*

$$\mathbb{E}_{(x,t)} \left[\mathbb{E}_n \left[(\bar{a}_{l-1})_{b,\pi(n)} (\bar{a}_{l'-1})_{d,\pi'(n)} \cdot \frac{\partial^2 \mathcal{L}_n}{\partial (h_l)_{a,\pi(n)} \partial (h_{l'})_{c,\pi'(n)}} \right] \right] \quad (16)$$

and

$$\mathbb{E}_x \left[\mathbb{E}_n \left[(\bar{a}_{l-1})_{b,\pi(n)} (\bar{a}_{l'-1})_{d,\pi'(n)} \right] \right] \cdot \mathbb{E}_{(x,t)} \left[\mathbb{E}_n \left[\frac{\partial^2 \mathcal{L}_n}{\partial (h_l)_{a,\pi(n)} \partial (h_{l'})_{c,\pi'(n)}} \right] \right] \quad (17)$$

are the same for any l, l′, a, b, c, d, π, and π', then

$$H = \bar{A} * \mathcal{H}' + \frac{1}{\max(N-1, 1)} \left(N\bar{A}' - \bar{A} \right) * (\mathcal{H}'' - \mathcal{H}'), \quad (18)$$

where

$$(\{\bar{A}'\}_{l,l'})_{b,d} = \mathbb{E}_x[\mathbb{E}_n[(\bar{a}_{l-1})_{b,n}]\mathbb{E}_n[(\bar{a}_{l'-1})_{d,n}]], \tag{19}$$

$$(\{\mathcal{H}'\}_{l,l'})_{a,c} = \mathbb{E}_{(x,t)}\left[\mathbb{E}_n\left[\sum_m \frac{\partial^2 \mathcal{L}_n}{\partial(h_l)_{a,m}\partial(h_{l'})_{c,m}}\right]\right], \tag{20}$$

$$(\{\mathcal{H}''\}_{l,l'})_{a,c} = \mathbb{E}_{(x,t)}\left[\mathbb{E}_n\left[\sum_{m,m'} \frac{\partial^2 \mathcal{L}_n}{\partial(h_l)_{a,m}\partial(h_{l'})_{c,m'}}\right]\right]. \tag{21}$$

Proof. See Appendix A.3. □

We will call Equation 18 the extended K-FAC, or simply XK-FAC. Note that $\mathcal{H} = \mathcal{H}' = \mathcal{H}''$ for a BN-free case, and hence, XK-FAC is a good generalization of K-FAC (Equation 13). Further, it can be applied with a large mini-batch size in practice because it requires only twice as much memory and computation as the original K-FAC.

If the Hessian is approximated by the Fisher information matrix [28] to avoid negative eigenvalues in practice, then $\hat{\mathcal{H}}'$ and $\hat{\mathcal{H}}''$ are used instead of $\mathcal{H}'$ and $\mathcal{H}''$, where

$$(\{\hat{\mathcal{H}}'\}_{l,l'})_{a,c} = \mathbb{E}_{(x,y)}\left[\mathbb{E}_n\left[\sum_m \frac{\partial \mathcal{L}_n}{\partial(h_l)_{a,m}}\frac{\partial \mathcal{L}_n}{\partial(h_{l'})_{c,m}}\right]\right], \tag{22}$$

$$(\{\hat{\mathcal{H}}''\}_{l,l'})_{a,c} = \mathbb{E}_{(x,y)}\left[\mathbb{E}_n\left[\left(\sum_m \frac{\partial \mathcal{L}_n}{\partial(h_l)_{a,m}}\right) \cdot \left(\sum_m \frac{\partial \mathcal{L}_n}{\partial(h_{l'})_{c,m}}\right)\right]\right]. \tag{23}$$

They are easily computed from the gradients, where the expectations are taken over the model's distribution. In this case, it is guaranteed that XK-FAC is positive semi-definite by the following theorem.

Theorem 3. ***(Positive semi-definiteness of XK-FAC)***

$$\bar{A}*\hat{\mathcal{H}}' + \frac{1}{\max(N-1,1)}\left(N\bar{A}' - \bar{A}\right)*(\hat{\mathcal{H}}'' - \hat{\mathcal{H}}') \succeq 0 \tag{24}$$

Proof. See Appendix A.4. □

4. Method for continual learning

Though XK-FAC is one of the key elements, XK-FAC alone cannot achieve continual learning with BN. Here we propose additional solutions to using BN properly with continual learning settings and discuss issues with hyperparameter selection.

4.1. Merged weight and batch renormalization

A weight matrix can be expressed by the product of multiple weight matrices in non-unique ways and consecutive weight matrices can be merged into one matrix, so a single network can be parameterized in many different ways. If there are many equivalent parameterization methods, then it would be natural to take the simplest form for efficiency. In particular, it will be computationally efficient and reduce the size of the Hessian if we can treat multiple layers as a single layer.

Let us consider a BN layer with its learnable affine parameters γ and β. First, for any input z, by observing that

$$\frac{z_{jn} - \bar{\mu}_i}{\sqrt{\bar{\sigma}_i^2 + \epsilon}}\gamma_i + \beta_i = \frac{\gamma_i}{\sqrt{\bar{\sigma}_i^2 + \epsilon}} z_{jn} + \left(\beta_i - \frac{\gamma_i \bar{\mu}_i}{\sqrt{\bar{\sigma}_i^2 + \epsilon}}\right), \tag{25}$$

we propose merging the normalization part and the affine transformation part of a BN layer into one equivalent affine transformation layer with its data-dependent learnable parameters $\tilde{\gamma}$ and $\tilde{\beta}$ defined as

$$\tilde{\gamma}_i = \frac{\gamma_i}{\sqrt{\bar{\sigma}_i^2 + \epsilon}}, \qquad \tilde{\beta}_i = \beta_i - \frac{\gamma_i \bar{\mu}_i}{\sqrt{\bar{\sigma}_i^2 + \epsilon}}, \tag{26}$$

where $\bar{\mu}_i$ and $\bar{\sigma}_i^2$ are mini-batch mean and variance, respectively.

If a preceding linear layer exists with its weight w (excluding bias), then this linear layer and the BN layer are merged into one equivalent fully connected layer as well. Thus, we define a merged fully connected layer whose data-dependent merged weight $\tilde{w}$ and merged bias $\tilde{b}$ are

$$\tilde{w}_{ij} = \frac{\gamma_i w_{ij}}{\sqrt{\bar{\sigma}_i^2 + \epsilon}}, \qquad \tilde{b}_i = \beta_i - \frac{\gamma_i \bar{\mu}_i}{\sqrt{\bar{\sigma}_i^2 + \epsilon}}. \tag{27}$$

Then, $\tilde{w}$ and $\tilde{b}$ are concatenated to form a merged weight matrix $\tilde{W}$. By this reparameterization, we can consider the penalty loss $\frac{1}{2}(\tilde{\theta} - \tilde{\theta}^*)^\top H(\tilde{\theta} - \tilde{\theta}^*)$ on the merged parameter $\tilde{\theta} = \text{vec}(\tilde{W})$, where the Hessian H is also taken with respect to $\tilde{\theta}$. As $\tilde{W}$ can represent any real matrix and is a continuous function of w, γ, and β, $\tilde{W}$ can move in any direction in the parameter space by moving w, γ, and β appropriately. Thus, we do not lose any representation power of the original network from the reparameterization.

In addition to efficiency, there are other advantages of reparameterization. If a single weight matrix is factorized to multiple matrices and each factor is penalized by Equation 4, then the factors are penalized to be kept close to their own original values. However, their original values individually are less important because of the non-uniqueness of factorization, and it is enough to keep the value of their multiplication only. Without reparameterization, each factor is unnecessarily over-penalized.

Further, the proposed reparameterization method plays one of the most important roles in continual learning with BN's statistical parameters $\bar{\mu}$ and $\bar{\sigma}$ which are determined by data and preceding weight matrices. If only free parameters are penalized and statistical parameters are not, then the penalty loss is totally not capable of preserving the performance of the original network, even at the global minimum of the penalty loss, due to drifts in statistics. So one may attempt to also directly penalize $\bar{\mu}$ and $\bar{\sigma}$, but this approach will fail because all preceding weights are unnecessarily penalized to keep the original statistics, which extremely conflicts with the penalty loss for the preceding weights. Statistical parameters and weight values cannot simultaneously keep their original values unless the statistics of source and target task data are the same. In contrast, if $\bar{\mu}$ and $\bar{\sigma}$ are merged with free parameters such as w, γ, and β, then free parameters can compensate for changes in $\bar{\mu}$ and $\bar{\sigma}$ caused by domain changes, while keeping the merged parameter similar without affecting preceding weights.

The use of mini-batch statistics causes stochastic fluctuations in $\tilde{W}$ because of the dependence of $\bar{\mu}$ and $\bar{\sigma}$ on mini-batch sampling, which make the penalty loss keep oscillating even at its local minimum. Hence, the penalty loss will converge more stably if such mini-batch dependencies can be eliminated; there is already a good solution for this scenario: batch renormalization (BRN) [13]. BRN uses population statistics instead of mini-batch statistics also in the training mode, while retaining the benefits of the BN by selective gradient propagation. Therefore, by adopting BRN in our framework, the merged parameters become

$$\tilde{w}_{ij} = \frac{\gamma_i w_{ij}}{\sqrt{\bar{\sigma}_i^2 + \epsilon}} r_i, \quad \tilde{b}_i = \beta_i + \gamma_i d_i - \frac{\gamma_i \bar{\mu}_i}{\sqrt{\bar{\sigma}_i^2 + \epsilon}} r_i, \tag{28}$$

where

$$r_i = \sqrt{\frac{\bar{\sigma}_i^2 + \epsilon}{\sigma_i^2 + \epsilon}}, \qquad d_i = \frac{\bar{\mu}_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} \tag{29}$$

are treated as constants so that the gradient is not propagated through r_i and d_i, and where μ_i and σ_i^2 are population statistics.

4.2. Forward/backward passes of the penalty loss

The gradient of the penalty loss can be efficiently calculated by matrix multiplications when the Hessian is approximated by a Kronecker-factored form [23, 31]. If $\mathcal{L}_s = \frac{1}{2}(\tilde{\theta} - \tilde{\theta}^*)^\top H (\tilde{\theta} - \tilde{\theta}^*)$ is the penalty loss, then for XK-FAC,

$$\frac{\partial \mathcal{L}_s}{\partial \tilde{W}_l} = \sum_{l'} \left(\{\mathcal{H}'\}_{l,l'} (\tilde{W}_{l'} - \tilde{W}_{l'}^*) \{\bar{A}\}_{l,l'}^\top + \{\mathcal{H}'' - \mathcal{H}'\}_{l,l'} (\tilde{W}_{l'} - \tilde{W}_{l'}^*) \left\{ \frac{N\bar{A}' - \bar{A}}{\max(N-1, 1)} \right\}_{l,l'}^\top \right), \tag{30}$$

since $(A \otimes B)\mathrm{vec}(C) = \mathrm{vec}(BCA^\top)$. Then, the forward pass can be performed by the following identity.

$$\mathcal{L}_s = \frac{1}{2} \sum_l \mathrm{vec}(\tilde{W}_l - \tilde{W}_l^*)^\top \mathrm{vec}\left(\frac{\partial \mathcal{L}_s}{\partial \tilde{W}_l} \right) \tag{31}$$

Finally, $\frac{\partial \mathcal{L}_s}{\partial \tilde{w}}$ and $\frac{\partial \mathcal{L}_s}{\partial \tilde{b}}$ are obtained by detaching the homogeneous dimension from $\frac{\partial \mathcal{L}_s}{\partial \tilde{W}}$, and they are propagated to $\frac{\partial \mathcal{L}_s}{\partial w}$, $\frac{\partial \mathcal{L}_s}{\partial \gamma}$, $\frac{\partial \mathcal{L}_s}{\partial \beta}$, and $\frac{\partial \mathcal{L}_s}{\partial a}$ by the usual chain rule.

4.3. Damping

The second-order approximation in Equation 4 is trustworthy only in a local neighborhood around the initial point. The quadratic penalty loss can prevent the parameters from moving too far from the initial point when positive semi-definite approximations of Hessian such as the Fisher information matrix [28] or the generalized Gauss–Newton matrix [33] are used; however, the problem can be still ill-posed. That is, if some eigenvalue of a Hessian is very close to zero, the parameter can move along the direction of the corresponding eigenvector with almost no restriction and eventually escape the trust region.

Here, we rethink the weight decay method, which is commonly used to regularize a network. The weight decay loss drags parameters down to the origin so that any single parameter cannot easily dominate the others and hopefully a network generalizes well. However, in our continual learning framework, the weight decay loss interferes with the penalty loss that attempts to maintain parameters near the initial point. Hence, we change the center of the weight decay loss from the origin to the initial point θ^*. This modified weight decay is equivalent to adding a damping matrix λI to H, and it achieves the effect of setting a lower limit λ on eigenvalues.

4.4. Preprocessing and postprocessing

When the first target sample passes through a network to begin continual learning, data-dependent parameters are changed immediately from the source task statistics to target task statistics. If the source and target tasks are not similar, then $\tilde{\theta}$ can be very different from $\tilde{\theta}^*$ at the beginning of training, which leads to a poor Hessian approximation of the source task loss function. To compensate for this drastic change in parameters, we reinitialize γ and β one time at the beginning of the training for each target task using the following preprocessing method. Let μ^* and σ^* be the source task population statistics stored in the source network, and μ and σ be the target task ones, which can be obtained by passing target samples through the network in the evaluation mode. Then, γ and β are reinitialized to

$$\beta_i \leftarrow \beta_i^* + \frac{\gamma_i^*}{\sqrt{\sigma_i^{*2} + \epsilon}} (\mu_i - \mu_i^*), \quad \gamma_i \leftarrow \sqrt{\frac{\sigma_i^2 + \epsilon}{\sigma_i^{*2} + \epsilon}} \gamma_i^*, \tag{32}$$

so that $\tilde{\theta}$ is initialized to $\tilde{\theta}^*$ at the start of a new task, because

$$\frac{z_{jn} - \mu_i^*}{\sqrt{\sigma_i^{*2} + \epsilon}} \gamma_i^* + \beta_i^* = \frac{z_{jn} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} \gamma_i + \beta_i \tag{33}$$

for any z.

After finishing continual learning with the current source and target tasks, the union of those two tasks becomes the new source task for future continual learning. Thus, the source task Hessian H must be updated to contain current target task information by postprocessing. If we let H_t be the Hessian of the current target task loss function, then, from Equation 5, it is obvious that

$$H \leftarrow \lambda_s H + \lambda_t H_t. \tag{34}$$

4.5. Hyperparameter selection

A continual learning method has importance hyperparameters or balancing parameters that control the importance of each task or trade-off between tasks. The choice of such hyperparameters is one of the most important in continual learning because it directly affects the final performance. However, as [29] pointed out, many approaches do not explain why hyperparameters were selected to have such specific values and how to select them in a real-world case, or they presciently use the best hyperparameters selected after all tasks have been processed, which can implicitly violate causality.

In our method, finding importance hyperparameters λ_s and λ_t using the source and target data is the same as optimizing the magnitude of the penalty loss, and thus, they should not be picked by validation over whole tasks. From Equations 1 and 5, it is obvious that λ_s and λ_t directly represent the magnitude of the source and target loss, respectively. Therefore, if each task is equally important, then the importance hyperparameters are set to

$$\lambda_s = \frac{T}{T+1}, \qquad \lambda_t = \frac{1}{T+1} \tag{35}$$

without any validation, where T is the number of tasks the model has learned so far. One can also easily set different importance for each task if needed.

If the best learning rate and damping hyperparameter for each task are found by validation over whole tasks, the source task data is then being used to set a trust region of the source task loss function because a small learning rate implicitly restricts the range that parameters can move and the damping method explicitly restricts it. Thus, for the learning rate and damping hyperparameter λ, we also avoid using the source task data to find them. However, there is not a natural choice for such hyperparameters, unlike the importance hyperparameters.

One advantage of the penalty method is that we can always access an approximation of the source task loss function, $\mathcal{L}_s = \frac{1}{2}(\theta - \theta^*)^\top H(\theta - \theta^*)$, and we propose a simple heuristic approach to make use of it. As we still do not know about source task accuracy, our basic idea is to adaptively scale the source task penalty loss $\lambda_s \mathcal{L}_s$ and target task loss $\lambda_t \mathcal{L}_t$ during training so that they converge to similar values, and to expect that the source and target accuracy drops also would be similar. However, because $\mathcal{L}_s$ does not contain the constant term in Equation 4, a constant C_t has to be subtracted from $\mathcal{L}_t$ for proper comparison, where C_t is a local minimum value for the target loss function that can be obtained from the fine-tuning with the target task.

To keep $\lambda_s \mathcal{L}_s$ and $\lambda_t(\mathcal{L}_t - C_t)$ similar during training, we introduce adaptive scaling hyperparameters α_s and α_t, and minimize $\alpha_s^{-1} \lambda_s \mathcal{L}_s + \alpha_t^{-1} \lambda_t \mathcal{L}_t$ instead of $\lambda_s \mathcal{L}_s + \lambda_t \mathcal{L}_t$. To be brief, α_s and α_t are initialized to 1, and one of them adaptively increases by 1 until $\lambda_s \mathcal{L}_s$ and $\lambda_t(\mathcal{L}_t - C_t)$ have similar values, and they are re-initialized to 1 to repeat the procedure. Specifically, if the average value of $\lambda_s \mathcal{L}_s$ within some interval is greater than that of $\lambda_t(\mathcal{L}_t - C_t)$, then α_t increases by 1 to relatively increase the scale of $\lambda_s \mathcal{L}_s$. α_t continues to increase in this manner at each interval until the average value of $\lambda_s \mathcal{L}_s$ within the interval is eventually smaller than that of $\lambda_t(\mathcal{L}_t - C_t)$, and then, α_t is re-initialized to 1. In the opposite case, α_s increases by 1 instead of α_t, and the procedure is similarly applied in the opposite way.

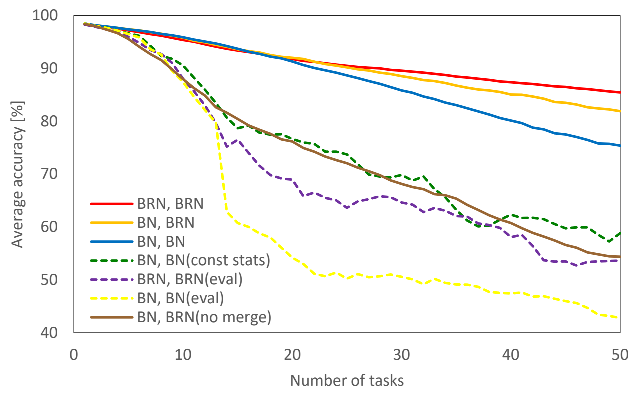
After training with various settings of learning rates and damping parameters, the model with the best target validation accuracy is determined as the final model. As the source loss and target loss are minimized while having similar values during training, we hope that the model with the best target validation accuracy will also perform well with the source task. In this way, we can find hyperparameters by leveraging the penalty loss value, without any access to source data.

5. Experiment

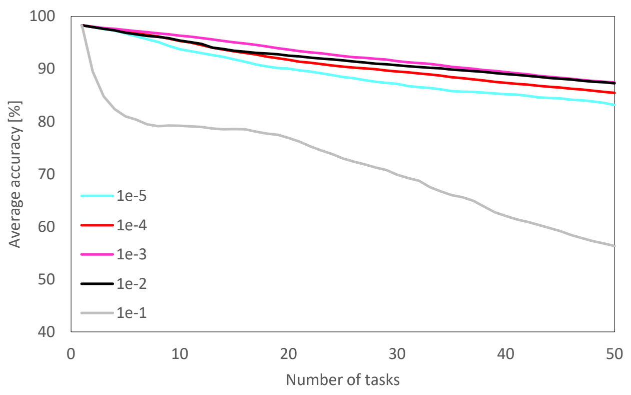
5.1. Permuted MNIST

Task and network architecture. As a first experiment, the permuted MNIST [17] task is performed as in [9, 15, 18, 31]. Each task is generated by a fixed random permutation of the original MNIST images. To investigate the effect of BN, we use a multilayer perceptron (784-128-128-10) with a BN or BRN layer inserted before every ReLU nonlinearity. BN is actually not necessary here, but even for such shallow network there is a significant performance gap between different BN settings in continual learning, and hence we can demonstrate the effectiveness of the proposed method with this simple experiment.

Hyperparameters. We emphasize that any source task data is not used explicitly or implicitly for hyperparameter selection. For all tasks, the same learning rate schedule, optimizer, and damping parameter are used. The learning rate starts at 0.1, and it is divided by 10 for every 5 epochs,



(a)



(b)

Figure 1: Average validation accuracy on permuted MNIST sequential learning. (a) The legend indicates whether BN layers or BRN layers are used for the network architecture and how their weights are interpreted when constructing the merged weights. For instance, for the orange line (BN, BRN), BN layers are used for the network, but their weights are interpreted as those of BRN layers when constructing the merged weights. XK-FAC is used for solid lines, and K-FAC is used for dashed lines. (b) The legend indicates the value of the damping hyperparameter λ. Each experiment was repeated four times.

a SGD optimizer with momentum 0.9 and mini-batch size 128 optimizes networks until 15 epochs, and the damping parameter λ is set to 0.0001, for all tasks. Further, we use early stopping based on the validation set of the current target task only, and not the whole tasks. The importance hyperparameters are set according to Equation 35.

BN and BRN. If the merged weights of the BN or BRN layers are constructed in a standard BN-way (Equation 27) or BRN-way (Equation 28), XK-FAC is used due to the mini-batch dependence of $\bar{\mu}$ and $\bar{\sigma}$. In short sequential learning, up to about 20 tasks, both BN (the blue line in Figure 1a) and BRN (the red line in Figure 1a) networks work similarly. However, as the sequence becomes extremely long, BRN networks prevent more forgetting than BN networks. If a model has learned a large number of tasks, it is much more important to maintain source task accuracy than to solve a target task. Thus, BRN would be preferred for a very long sequence because the penalty loss is more stabilized.

Another interesting finding is that even if BN layers are used in a network architecture, it is better to interpret their weights as those of BRN layers when constructing the merged weights (the orange line in Figure 1a). For mini-batch sizes large enough to assume that population statistics and mini-batch statistics are very similar, a BN and the corresponding BRN layer are almost identical, and thus, the Hessian of BRN layers can still provide a reliable penalty loss to BN layers in a more stabilized way.

Further, we tried other ways of constructing merged weights that circumvent the inter-example relations. First, mini-batch statistics $\bar{\mu}$ and $\bar{\sigma}$ can be treated as if they were constants ('const stats' in Figure 1a). Or, the population statistics μ and σ, which are used in the evaluation mode and considered as constants, can be used to obtain the merged weight instead of the mini-batch statistics ('eval' in Figure 1a). However, the performance drops rapidly if inter-example relations are not considered.

Effect of weight merging. In Section 4.1, we discussed why it is preferable to use merged weights, especially when there exist statistical parameters such as $\bar{\mu}$ and $\bar{\sigma}$ of BN. If we individually penalize each parameter, w, $\bar{\mu}$, $\bar{\sigma}$, γ, and β instead of the merged weight $\tilde{W}$, each penalty loss will conflict with each other to keep statistical parameters, resulting in significant performance degradation (the brown line in Figure 1a).

Damping. The effect of the damping hyperparameter λ is shown in Figure 1b. For moderately small λ, the damping method can slightly improve the performance. If it is too large, the Hessian is severely corrupted and cannot give a good approximation of the source task loss function. We simply set λ to a typical value of 0.0001 in the experiment; however, further investigation is needed for determining the value of λ or better damping methods in future work.

5.2. ImageNet to fine-grained classification tasks

Task and network architecture. For an experiment with deep networks, we perform sequential learning from the ImageNet classification task to multiple fine-grained classification tasks with ResNet-50 [11], as in [21, 22]. Starting with a network pre-trained on ImageNet [32], three fine-grained datasets, CUBS [34], Stanford Cars [16], and Flowers [26], are learned sequentially with 6 different orders. All datasets are preprocessed and augmented in the same way as [22].

Method	active BN	valid curvature	adaptive α_s, α_t	ImageNet	Birds	Cars	Flowers
Fixed BN		✓		29.21	67.05	74.24	74.20
K-FAC	✓			50.37	70.49	**85.98**	88.45
XK-FAC	✓	✓		**61.15**	**78.97**	85.91	**91.84**
Fixed BN		✓	✓	49.57	76.24	84.94	81.30
K-FAC	✓		✓	65.31	80.09	87.35	92.14
XK-FAC	✓	✓	✓	**66.57**	**80.30**	**88.07**	**92.56**

Table 1: Top-1 validation accuracy on ImageNet to fine-grained classification tasks. The results were averaged over 6 possible orderings of the three fine-grained datasets.

Hyperparameters. We also do not use any source task data for hyperparameter selection here. For all experiments, SGD with momentum 0.9 and mini-batch size 32 optimizes networks until 100 epochs, and λ_s and λ_t are set according to Equation 35. The learning rate gradually decreases to 1e-4, and α_s and α_t are updated every 10 iterations. We tried initial learning rates of {1e-1, 5e-2, 2e-2, 1e-2} and damping hyperparameters of {1e-4, 5e-5, 2e-5, 1e-5} for each method, and the final model is determined according to the method described in Section 4.5. For the proposed method, BN layers in a ResNet are treated as BRN layers when constructing the merged weights. As in [13], r_{max} and d_{max} in BRN are initialized to 1 and 0, and they are gradually relaxed to 3 and 5, respectively.

Convolutional network. To apply the proposed method to convolutional networks, we combine our XK-FAC and KFC [10]. As KFC only handles block diagonal parts of the Hessian, the curvature of the penalty loss is approximated by a block diagonal matrix, by considering only the case $l = l'$, as in [31]. Details can be found in Appendix A.5.

Fixed BN baseline. One of the natural baseline methods is a 'fixed BN' method. For a fixed BN method, the parameters of BN, γ, β, μ, and σ, are fixed to the parameters of the ImageNet pre-trained model, and they are not considered as learnable parameters during sequential learning. It is not very restrictive to fix BN parameters in terms of representation power of a network, because w is still a free parameter and a network does not lose any degree of freedom. In this case, K-FAC is a valid curvature approximation since there is no inter-example relations. However, as seen in Table 1, fixed BN baselines do not perform well in both source and target tasks. The curvature of the original source task loss function with BN layers will differ from the Hessian of the corresponding fixed-BN model. On the other hand, for target tasks, the network cannot take advantages of the BN layers, such as the ease of optimization of such a deep network or better generalization. We found that networks diverge for learning rates of 1e-1 or 5e-2 due to the absence of BN layers, so we tried learning rates of {2e-2, 1e-2, 5e-3, 2e-3} only for the fixed BN method.

K-FAC baseline. Another baseline is a 'K-FAC' method, which does not fix BN but uses an invalid curvature approximation, K-FAC, rather than a valid one, XK-FAC. For the K-FAC baselines in Table 1, the experimental settings and the initial network are exactly the same with that of the XK-FAC methods except the Hessian approximation, so all of the performance degradation comes from ignoring inter-example relations when approximating the curvature. It is remarkable that simply using a better Hessian approximation can improve performance. Quantitatively, for the initial network, the mean absolute values of all block diagonal entries in K-FAC and XK-FAC are respectively 5.65e-7 and 4.96e-7, and that of the difference between K-FAC and XK-FAC is 1.19e-7, which is not negligible.

Effect of α_s and α_t. In Section 4.5, we proposed a simple approach to finding hyperparameters by leveraging the penalty loss value without any source data, and the special hyperparameters α_s and α_t were introduced to control the balance between a source task penalty loss and a target task loss. To observe the effect of α_s and α_t, we also experimented with α_s and α_t fixed to 1. As seen in Table 1, even though the proposed method of controlling α_s and α_t is very simple, it can significantly improve the performance of all methods, including the baselines. See Appendix A.6 for more results. It would be worth exploring more sophisticated methods as a future work.

6. Conclusion

In this paper, we extended K-FAC to consider inter-example relations caused by BN layers. Further, detailed methods of applying XK-FAC for continual learning, such as weight merging, use of BRN layers, and hyperparameter selection, were proposed, and we demonstrated that each proposed method works effectively in continual learning.

Acknowledgment

This research was supported by the Engineering Research Center Program through the National Research Foundation of Korea (NRF) funded by the Korean Government MSIT (NRF-2018R1A5A1059921) and was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2019-0-00075, Artificial Intelligence Graduate School Program(KAIST)).

References

[1] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 139–154, 2018.

[2] Jimmy Ba, Roger Grosse, and James Martens. Distributed second-order optimization using Kronecker-factored approximations. In *International Conference on Learning Representations*, 2017.

[3] Chris Bishop. Exact calculation of the Hessian matrix for the multilayer perceptron. *Neural Computation*, 4(4):494–501, 1992.

[4] Aleksandar Botev, Hippolyt Ritter, and David Barber. Practical Gauss–Newton optimisation for deep learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 557–565. JMLR. org, 2017.

[5] Kumar Chellapilla, Sidd Puri, and Patrice Simard. High performance convolutional neural networks for document processing. In *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft, 2006.

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

[7] Robert M French and Nick Chater. Using noise to compute error surfaces in connectionist networks: A novel means of reducing catastrophic forgetting. *Neural computation*, 14(7):1755–1769, 2002.

[8] Thomas George, César Laurent, Xavier Bouthillier, Nicolas Ballas, and Pascal Vincent. Fast approximate natural gradient descent in a Kronecker factored eigenbasis. In *Advances in Neural Information Processing Systems*, pages 9550–9560, 2018.

[9] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. In *International Conference on Learning Representations*, 2014.

[10] Roger Grosse and James Martens. A Kronecker-factored approximate Fisher matrix for convolution layers. In *International Conference on Machine Learning*, pages 573–582, 2016.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[13] Sergey Ioffe. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. In *Advances in Neural Information Processing Systems*, pages 1945–1953, 2017.

[14] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[15] 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*, 114(13):3521–3526, 2017.

[16] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3D object representations for fine-grained categorization. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 554–561, 2013.

[17] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[18] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. In *Advances in neural information processing systems*, pages 4652–4662, 2017.

[19] Shuangzhe Liu. Matrix results on the Khatri–Rao and Tracy–Singh products. *Linear Algebra and its Applications*, 289(1-3):267–277, 1999.

[20] Xialei Liu, Marc Masana, Luis Herranz, Joost Van de Weijer, Antonio M Lopez, and Andrew D Bagdanov. Rotate your networks: Better weight consolidation and less catastrophic forgetting. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 2262–2268. IEEE, 2018.

[21] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 67–82, 2018.

[22] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018.

[23] James Martens, Jimmy Ba, and Matt Johnson. Kronecker-factored curvature approximations for recurrent neural networks. In *International Conference on Learning Representations*, 2018.

[24] James Martens and Roger Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417, 2015.

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

[26] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729. IEEE, 2008.

[27] Dongmin Park, Seokil Hong, Bohyung Han, and Kyoung Mu Lee. Continual learning by asymmetric loss approximation with single-side overestimation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3335–3344, 2019.

[28] Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. In *International Conference on Learning Representations*, 2014.

[29] B. Pfülb and A. Gepperth. A comprehensive, application-oriented study of catastrophic forgetting in DNNs. In *International Conference on Learning Representations*, 2019.

[30] Călin-Adrian Popa. Exact Hessian matrix calculation for complex-valued neural networks. In *International Workshop Soft Computing Applications*, pages 439–455. Springer, 2014.

[31] Hippolyt Ritter, Aleksandar Botev, and David Barber. Online structured Laplace approximations for overcoming catastrophic forgetting. In *Advances in Neural Information Processing Systems*, pages 3742–3752, 2018.

[32] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[33] Nicol N Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural computation*, 14(7):1723–1738, 2002.

[34] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

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

[36] Xian Zhang, ZP Yang, and CG Cao. Inequalities involving Khatri–Rao products of positive semidefinite matrices. *Applied Mathematics E-Notes*, 2:117–124, 2002.