Analyzing the Effectiveness of K-FAC on Training MLP-Mixers for Image Classification

Salar Hosseini **Jiachen Zhou** **Yuxuan Chen**
University of Toronto

Abstract

Kronecker-Factored Approximate Curvature (K-FAC) [1] is an approximate second-order optimization method for deep learning that speeds up the training process by approximating the Fisher Information Matrix (FIM) which has been shown to achieve state-of-the-art performance on large-scale neural network optimization tasks. K-FAC was originally shown to be effective on logistic autoencoders, and has also been extended to handle convolutional networks [2] and RNNs [3]. For this project, we apply K-FAC to the MLP-Mixer [4], which is a modern architecture that consists of only multi-layer perceptrons, then compare its performance with first order optimizers like SGD with momentum and Adam on CIFAR-10 and CIFAR-100 image classification tasks. We demonstrate that, when no pretraining is involved, K-FAC outperforms SGD and Adam on CIFAR-100 and also when handling large input sizes on CIFAR-10. In addition, we perform learning rate grafting to confirm that the implicit learning rate schedule is the primary factor dictating classification performance. [1]

1 Introduction

Deep networks have achieved state-of-the-art performance in tasks such as image recognition and natural language processing. Stochastic Gradient Descent (SGD) and its variants are widely used as optimizers for neural network training. However, the loss functions of the network parameters are highly non-convex, and the highly imbalanced curvature on the loss surface limits the efficiency of first order optimization methods such as SGD. As a result, it typically requires days of training time to train a top-performing network.

In order to speed up the rate of convergence, second order optimization methods correct the imbalanced curvature of the objective function using second-order information. Traditionally, second-order methods work by inverting a large Hessian matrix (in Newton's method) or the Fisher Information Matrix (FIM) (in natural gradient descent), which does not scale well to deep neural networks with a massive amount of parameters. This has motivated researchers to find invertible approximations of the Gaussian-Newton Hessian or the Fisher Information Matrix which are faster to compute. For instance, Adagrad [5], Adam [6], and RMSProp [7] involve a simple diagonal approximation to the covariance matrix of the gradients. In order to preserve correlations between parameters, TONGA [8] proposed a block-diagonal approximation of the empirical FIM. A similar block-diagonal approach is proposed by [9] that approximates the standard FIM instead of the empirical FIM.

In addition, factored approximations, such as K-FAC [1], approximate various large blocks of the FIM as the Kronecker product of two smaller matrices, which can be estimated and inverted with a reduced computational cost.

Previous works have shown that K-FAC can be applied to fully-connected layers [1], convolutional networks [2], and recurrent neural networks [3]. In this work, we extend K-FAC to the MLP-Mixer [4] and evaluate the approximation against classical gradient descent methods on several image

[1] Code is available at https://github.com/Salarios77/kfac-mlp-mixer

classification tasks. In addition, we (i) further analyze the effectiveness/ineffectiveness of K-FAC on MLP-Mixers by performing grafting [10] (using SGD as the second optimizer) to disentangle the effect of the magnitude and direction imposed by the K-FAC weight updates, and (ii) compare the performance acquired by finetuning a pretrained MLP-Mixer using K-FAC against that of SGD and Adam [6].

2 Related Works

2.1 Approximate second order optimization

Natural Gradient Descent (NGD) is an optimization method proposed by [11] that approximates the curvature of a loss function using the Fisher Information Matrix (FIM). However, due to the massive number of parameters in neural networks, the inverse of the FIM is intractable, which makes NGD impractical for deep learning applications.

Many recent works have proposed various methods to approximate second-order optimization. K-FAC [1] is one of the most efficient natural gradient approximation methods that approximates the FIM such that the inverse is easier to compute. First, K-FAC block-diagonalizes the FIM where each diagonal block corresponds to parameters of each layer of the neural network. Next, K-FAC approximates each block with the Kronecker product of two matrices. Lastly, K-FAC uses the property of the Kronecker product of matrices to compute the inverse of the FIM.

2.2 MLP-Mixer

Convolutional neural networks and attention-based networks such as the vision transformer have become popular for image classification tasks. However, it has been shown in [4] that neither convolutions nor attention are necessary for good performance. [4] proposed an architecture, MLP-Mixer, which only relies on multi-layer perceptrons (MLPs). MLP-Mixer consists of per-patch linear embeddings, mixer layers, and a classifier head. Mixer layers contain a token-mixing MLP for spatial information learning and a channel-mixing MLP for per-location features, each including 2 fully-connected layers and a GELU [12] non-linear activation with standard skip-connections and layer normalization. The model is pretrained using the Adam optimizer [6] and fine-tuned for downstream tasks using SGD with momentum.

2.3 Disentangling step size from direction

One instance of methods which precondition the SGD update by another matrix is the class of adaptive gradient methods. In an attempt to disentangle the effect of adaptive gradient methods on the magnitude and direction of weight updates, Agarwal et. al [10] introduced "learning rate grafting". This method works by querying the direction for the gradient update from one optimizer, and the magnitude from another. Since K-FAC preconditions the SGD update with an approximation of a neural network's Fisher Information Matrix, its effect on the magnitude and direction can also be disentangled to provide further insights on its effectiveness/ineffectiveness when training the MLP-Mixer.

3 Methodology

This section describes our primary contribution: apply K-FAC [1] as the natural gradient approximation method in training MLP-Mixer [4] models on CIFAR [13] image classification tasks.

3.1 MLP-Mixer Network Architecture

For MLP-Mixer to perform classification tasks on image data, it takes as an input a sequence of S non-overlapping patches dividing the input image of size (H, W). Each image patch, of resolution $P \times P$ is projected to a vector of hidden dimension C via a linear layer, resulting in a two-dimensional input table $\mathbf{X} \in \mathbb{R}^{S \times C}$ where sequence length is determined by $S = \frac{H}{P}\frac{W}{P} = \frac{HW}{P^2}$. As shown in Figure 1, MLP-Mixer consists of multiple Mixer layers, each containing two MLP blocks that operates along one dimension of the input table $\mathbf{X}$ respectively. The first one acts on columns of $\mathbf{X}$ to achieve

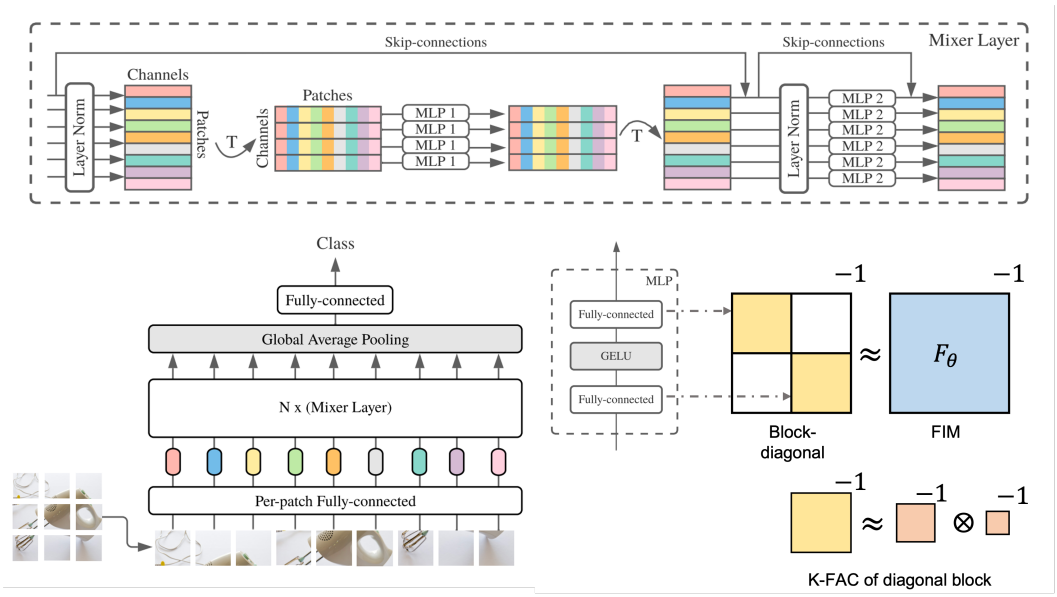

Figure 1: Overall architecture of MLP-Mixer [4] and illustration of K-FAC [1]

token-mixing; the second acts along the rows as a per-location channel-mixing MLP. The use of these two types of layers enables interaction of both input dimensions. Each Mixer layer can be expressed in the following Eqn. 1 and Eqn. 2:

$$\mathbf{U}_{*,i} = \mathbf{X}_{*,i} + \mathbf{W}_2\sigma(\mathbf{W}_1 LN(\mathbf{X})_{*,i}), \text{ for } i = 1 \dots C, \tag{1}$$

$$\mathbf{Y}_{j,*} = \mathbf{U}_{j,*} + \mathbf{W}_4\sigma(\mathbf{W}_3 LN(\mathbf{U})_{j,*}), \text{ for } j = 1 \dots S. \tag{2}$$

where σ is an element-wise GELU [12] non-linearity and the summation represents skip-connections. Apart from the MLP layers, the classification head is a standard global average pooling layer followed by another linear classifier layer.

3.2 K-FAC

Kronecker-Factored Approximate Curvature (K-FAC) [1] is a second-order optimization method which approximates the curvature of loss function by Kronecker factorization that helps reduce the computation complexity of the natural gradient update. Such computational burden comes from inverting the Fisher Information Matrix (FIM) used by Natural Gradient Descent [11]. The FIM of a probabilistic model that outputs a conditional probability of target y given data x and model θ is defined as $F_\theta = \mathbb{E}_{(x,y)}[\nabla \log p(y|x;\theta) \nabla \log p(y|x;\theta)^T]$. In an image classification task, it is common to use the mean value of the negative log-likelihood as the loss function, defined as $L(\theta) = \mathbb{E}_{(x,y)}[-\log p(y|x;\theta)]$. Furthermore, one can show the Hessian of $L(\theta)$ as Eqn. 3:

$$H(L)(\theta) = \nabla^2 \mathbb{E}_{(x,y)}[-\log p(y|x;\theta)] = F_\theta - \mathbb{E}_{(x,y)}[\frac{\nabla^2 p(y|x;\theta)}{p(y|x;\theta)}] \tag{3}$$

Hence, the update rule is $\theta^{(t+1)} = \theta^t - \epsilon F^{-1}_{\theta^{(t)}} \nabla E(\theta^{(t)})$ that requires the inverse of the FIM. Since FIM is a matrix of size $N \times N$ where N is the number of parameters whose order of magnitude is often around millions in a typical deep neural network, the inverse of FIM becomes intractable in NGD [11].

Following [1], we first block-diagonalize the FIM in which parameters of each layer lie along the diagonal blocks as indicated in Figure 1. Next, K-FAC [1] factorizes each block by the Kronecker product of two matrices, using the matrix inversion property of the Kronecker product, $(A \otimes B)^{-1} =$

$A^{-1} \otimes B^{-1}$. Therefore, each diagonal block in the F_θ^{-1} can be expressed as the product of two smaller Kronecker factors, reducing the computation complexity.

In specific, we show derivation on a fully-connected layer since MLP-Mixer [4] only consists of multi-layer perceptrons. The Fisher block in the i-th layer has the expression $F_i = \mathbb{E}[\nabla_i \log p(y|x;\theta) \nabla_i \log p(y|x;\theta)^T]$. During the back propagation, we factorize gradient of the log likelihood as $\nabla_i \log p(y|x;\theta) = g_i \otimes a_{i-1}$, where $a_{i-1} \in \mathbb{R}^{d_{i-1}}$ is the activation output from the previous $(i-1)$ layer, and $g_i = \frac{\partial \log p(y|x;\theta)}{\partial w_i} \in \mathbb{R}^{d_i}$ is the gradient of the output of the i-th layer. Putting together, the Fisher block can be transformed into the following:

$$F_i = \mathbb{E}[\nabla_i \log p(y|x;\theta) \nabla_i \log p(y|x;\theta)^T] = \mathbb{E}[(g_i \otimes a_{i-1})(g_i \otimes a_{i-1})^T] \tag{4}$$

Using the matrix transpose property of the Kronecker product on Eqn. 4, $(A \otimes B)^T = A^T \otimes B^T$, we arrive at:

$$F_i = \mathbb{E}[(g_i \otimes a_{i-1})(g_i^T \otimes a_{i-1}^T)] = \mathbb{E}[g_i g_i^T \otimes a_{i-1} a_{i-1}^T] \tag{5}$$

By applying the K-FAC approximation on Eqn. 5, we can push the expectation inwards and get a Kronecker product of expected values shown as Eqn. 6:

$$F_i \approx \mathbb{E}[g_i g_i^T] \otimes \mathbb{E}[a_{i-1} a_{i-1}^T] = G_i \otimes A_{i-1} \tag{6}$$

where $F_i \in \mathbb{R}^{(d_{i-1}d_i)(d_{i-1}d_i)}$, $G_i \in \mathbb{R}^{d_i d_i}$ and $A_{i-1} \in \mathbb{R}^{d_{i-1}d_{i-1}}$.

4 Experimental Setup

4.1 Datasets

The datasets that we perform training and evaluation on are the CIFAR-10 [13] and CIFAR-100 [13] image classification datasets, both of which contain 32x32 colour images and class labels. The CIFAR-10 dataset has 50000 training images and 10000 testing images, consisting of 10 image classes. The CIFAR-100 dataset has the same number of training and testing images, except with 100 image classes (500 training images and 100 testing images per class). The data augmentations that we perform for both datasets are random horizontal flip (only during training) and normalization with fixed mean and standard deviation. In addition to the base image size of 32x32, we also experiment on 224x224 images as it provides a higher dimensional input space to test optimization on, and since it is the size that the MLP-Mixer [4] was originally trained on. To upsample the images, we follow the procedure used in [4] and first resize the images to 256x256 using bilinear interpolation and then take a 224x224 central crop.

4.2 Model Configuration

We perform training with 5 different specifications of the MLP-Mixer architecture which are presented in Table 1. Following [4], we denote each model by their size "S" for small and "B" for base, followed by the patch size. For 32x32 images we use a patch size of 4, and for 224x224 images we use a patch size of 16. The only architecture which we additionally test the pretrained (on ImageNet[14]) version is B/16, since that is the only one from this list for which [4] has made a pretrained checkpoint publicly available. It is worth mentioning that B/16-PT was pretrained using Adam [4].

Table 1: MLP-Mixer architecture specifications used for training on CIFAR-10 and CIFAR-100.

Model Specification	S/4	B/4	S/16	B/16	B/16-PT
Input image size	32 x 32	32 x 32	224 x 224	224 x 224	224 x 224
Patch resolution	4 x 4	4 x 4	16 x 16	16 x 16	16 x 16
Hidden size C	512	768	512	768	768
Pretraining	N/A	N/A	N/A	N/A	ImageNet[14]

4.3 Training Details

For all experiments, we perform training over 100 epochs with one GPU and a batch size of 64. The optimizers which we compare against K-FAC are SGD with momentum and Adam [6]. For all optimizers, we set momentum to 0.9. For weight decay, we tested values {1e-3, 1e-4} and found 1e-4 to work best for all optimizers. For learning rate, we tested values {1e-2, 1e-3} and found that SGD worked best with a learning rate of 1e-2, while Adam and K-FAC worked best with a learning rate of 1e-3. For K-FAC, we used a damping value of 1e-3. For additional hyperparameters, please refer to the code provided.

5 Experimental Results

In this section we provide the image classification results on CIFAR-10 and CIFAR-100 from training MLP-Mixer models using SGD with momentum, Adam, and K-FAC. In order to evaluate the performance of each optimizer we report the best top-1 accuracy on the test set, and the number of epochs taken to reach a target test accuracy. The target test accuracies are mostly arbitrary, and were roughly chosen based on the average performances of the three optimizers. These values are provided in the captions of the results tables in the following sections. The epochs to target metric is only provided to give a rough sense of how fast each optimizer converged.

5.1 Optimizer Results on CIFAR-10

The results for image classification on CIFAR-10 are presented in Table 2, and the training results are shown in Figure 3 and 2. Upon examining the performances on models with input sizes of 32x32 (S/4 and B/4), it is evident that all three optimizers perform similarly in terms of best test accuracy and number of epochs to reach target test accuracies. This indicates that optimizing the smaller images is equally challenging for all three optimizers. However, the results (without pretraining) on the larger input sizes of 224x224 indicate a greater discrepancy between the performances of the three optimizers. For both S/16 and B/16, K-FAC results in the best performance, followed by SGD and then Adam. This holds true for both test accuracy and convergence speed (epochs to target). We suspect that the improvement of K-FAC here is due to the optimization problem becoming more difficult (i.e. a more imbalanced curvature in the loss landscape) when the small 32x32 image pixels are stretched and it becomes harder to detect the object boundaries.

Lastly, we examine the performance of the three optimizers on finetuning a model with input size 224x224 which is pretrained on ImageNet (B/16-PT). For this setting, we notice that SGD significantly outperforms Adam and K-FAC and reaches the target test accuracy within the very first epoch, while K-FAC takes 40 epochs to reach it, and Adam significantly underperforms and does not reach the target accuracy. This is a surprising result as B/16-PT was originally pretrained using Adam and was expected to perform well when finetuned using the same optimizer. However, in Section 5.2, we show that a consistent result holds for CIFAR-100.

Table 2: **The Top-1 classification accuracy and epochs to target accuracy (both on test set) from training MLP-Mixer models on CIFAR-10** using SGD with momentum, Adam, and K-FAC. The target test accuracies for {S/4, B/4, S/16, B/16, B/16-PT} are {80%, 80%, 70%, 65%, 90%} respectively. 'Pretrained ✓' indicates the model was initialized using a checkpoint pretrained on ImageNet[14]; 'Pretrained ✗' indicates random initialization. "-" indicates that the target test accuracy was not reached.

Model	Input Size	Pretrained	Test Accuracy			Epochs to Target		
			SGD	Adam	K-FAC	SGD	Adam	K-FAC
S/4	32^2	✗	**83.22**	80.88	82.42	38	43	**35**
B/4	32^2	✗	82.73	80.9	**82.81**	**40**	42	**40**
S/16	224^2	✗	74.64	71.99	**80.44**	11	15	**3**
B/16	224^2	✗	72.68	67.09	**79.02**	6	10	**2**
B/16-PT	224^2	✓	**96.73**	72.24	91.00	**0**	-	40

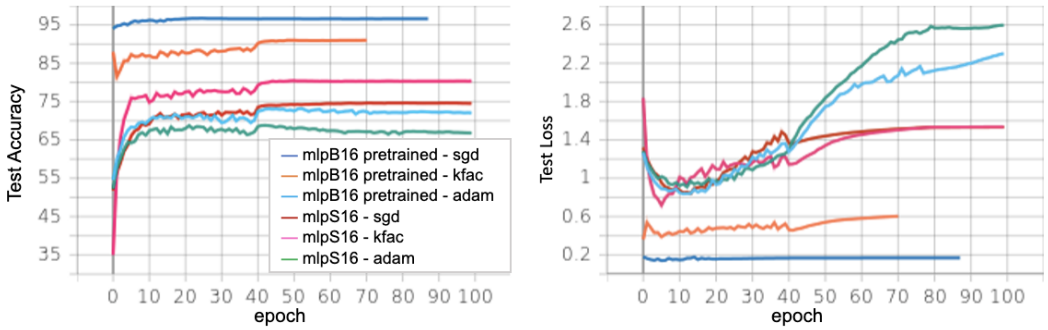

Figure 2: Test results of MLPB16 and MLPS16 on CIFAR-10

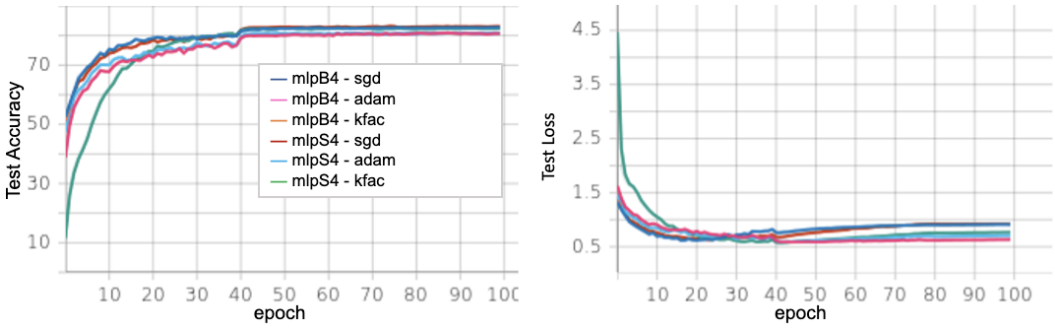

Figure 3: Test results of MLPB4 and MLPS4 on CIFAR-10

5.2 Optimizer Results on CIFAR-100

Here we perform similar experiments to Section 5.1, except on the CIFAR-100 dataset, which is expected to be more difficult to optimize due to its higher number of classes. We only use the "B" (base) models for CIFAR-100, but it would be interesting to experiment on different model sizes in future work. Our results are shown in Table 3 and Figure 4. As shown from the training results on the 32x32 and 224x224 input sizes (without pretraining), K-FAC significantly outperforms SGD and Adam in terms of best test accuracy and epochs to target test accuracy. This is consistent with the CIFAR-10 results on 224x224 images, but now also holds true for 32x32 images (on CIFAR-10 the three optimizers performed equally well on 32x32 images). When examining the finetuning results on the pretrained B/16-PT model, we find that SGD significantly outperforms Adam and K-FAC and reaches the target test accuracy within the first epoch. This is very a similar result to that shown in Table 2 when performing finetuning on CIFAR-10.

Table 3: **The Top-1 classification accuracy and epochs to target accuracy (both on test set) from training MLP-Mixer models on CIFAR-100** using SGD with momentum, Adam, and K-FAC. The target test accuracies for {B/4, B/16, B/16-PT} are {45%, 40%, 70%} respectively. 'Pretrained ✓' indicates the model was initialized using a checkpoint pretrained on ImageNet-1K; 'Pretrained ✗' indicates random initialization. "-" indicates that the target test accuracy was not reached.

Model	Input Size	Pretrained	Test Accuracy			Epochs to Target		
			SGD	Adam	K-FAC	SGD	Adam	K-FAC
B/4	32^2	✗	53.67	48.04	**60.59**	10	11	**3**
B/16	224^2	✗	47.65	42.71	**54.96**	6	7	**3**
B/16-PT	224^2	✓	**83.49**	45.37	70.81	**0**	-	41

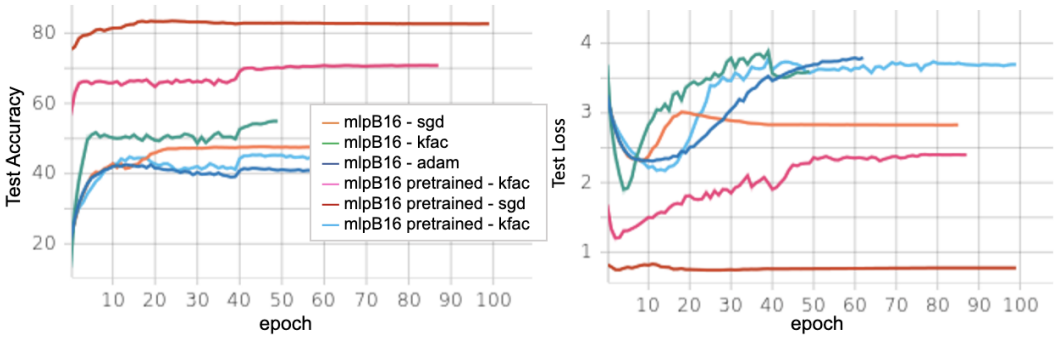

Figure 4: Test results of MLPB16 on CIFAR-100

5.3 Learning Rate Grafting using K-FAC and SGD

In this section we conduct a deeper analysis on the difference between the SGD and K-FAC optimization results obtained on CIFAR-10 (shown in Table 2) by performing learning rate grafting [10]. We only examine the models which resulted in notable differences in performance (i.e. S/16, B/16, and B/16-PT) across the different optimizers, and attempt to understand whether the improvement of the best optimizer came from the magnitude or direction of the gradient steps.

In Table 4, we show the grafting training results for optimizer pairs $(\mathcal{M}, \mathcal{D})$, where optimizer $\mathcal{M}$ determines the step magnitude and optimizer $\mathcal{D}$ determines the step direction. It is also worth noting that no extra hyperparameter tuning was necessary for these experiments and each optimizer was set up with the same settings detailed in Section 4.3. Each row of the tables for each model controls for the implicit step size schedule (magnitude), while each column of the tables isolates preconditioning dynamics (direction). From examining the results for all three models, we can observe that there is very little difference in the performance within each row, while there is a clear difference within each column. This result is consistent with the finding of [10] and shows that the primary factor determining classification performance is the implicit step size schedule. In other words, K-FAC outperforms SGD for S/16 and B/16 mainly due to finding better step sizes, and likewise for SGD on B/16-PT.

Table 4: **Top-1 classification accuracy (on test set) from training MLP-Mixer models on CIFAR-10 using learning rate grafting**, where the magnitude of $\mathcal{M}$'s step is combined with the direction of $\mathcal{D}$'s step. Choices for optimizers $(\mathcal{M}, \mathcal{D})$ are combinations of SGD with momentum and K-FAC. Grafting $(\mathcal{A}, \mathcal{A})$ is equivalent to simply running $\mathcal{A}$.

Model S/16		
$\mathcal{M}\backslash\mathcal{D}$	SGD	K-FAC
SGD	74.64	75.47
K-FAC	80.45	80.44

Model B/16		
$\mathcal{M}\backslash\mathcal{D}$	SGD	K-FAC
SGD	72.68	75.44
K-FAC	78.86	79.02

Model B/16-PT		
$\mathcal{M}\backslash\mathcal{D}$	SGD	K-FAC
SGD	96.73	95.92
K-FAC	90.75	91.00

6 Limitations

One limitation of our method is that we haven't conducted extensive hyper-parameter tuning on all three optimizers. From the results, we can see that the Adam optimizer underperforms by a noticeable margin in all experiments. This could be due to the fact that the hyper-parameters for Adam optimizer are not well-tuned. For future work, we could conduct a more rigorous grid-search to find the best combination of hyper-parameters.

In addition, we could further improve the practicality of K-FAC for use in deep learning by applying it to more modern architectures such as Auto-Encoders [15] and Tansformers [16], then investigate the effectiveness of K-FAC by comparing with other optimizers on those architectures.

7 Conclusions

In this paper, we applied K-FAC to the MLP-Mixer, and evaluated on CIFAR-10 and CIFAR-100. We showed that K-FAC results in the best performance on both datasets in terms of classification accuracy and convergence rate using the B/16 and S/16 MLP-Mixer architectures without pre-training. In addition, we investigated the performance of learning rate grafting on the MLP-Mixer using SGD and K-FAC as the optimizer pair, and confirmed that the implicit step size is the primary factor that dictates the classification performance.

8 Work Division

1. Set up data-loading for the CIFAR-10 and CIFAR-100 datasets [17]. (Sherry)
2. Extend K-FAC to the MLP-Mixer architecture. (Sherry, Salar, Jason)
3. Train the MLP-Mixer on both datasets using K-FAC. (Sherry, Salar, Jason)
4. Compare the performance of K-FAC against SGD with momentum and ADAM. (Sherry, Salar, Jason)
5. Disentangle the effect of K-FAC weight updates by performing grafting using SGD as the second optimizer. (Salar)
6. Evaluate the optimization performance of K-FAC on fine-tuning a MLP-Mixer that was pretrained using a larger dataset. (Sherry, Salar)
7. Write report. (Sherry, Salar, Jason)

References

[1] J. Martens and R. Grosse, "Optimizing neural networks with kronecker-factored approximate curvature," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 2408–2417. [Online]. Available: https://proceedings.mlr.press/v37/martens15.html

[2] J. Ba, R. Grosse, and J. Martens, "Distributed second-order optimization using kronecker-factored approximations," 2016.

[3] J. Martens, J. Ba, and M. Johnson, "Kronecker-factored curvature approximations for recurrent neural networks," in *International Conference on Learning Representations*, 2018.

[4] I. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. P. Steiner, D. Keysers, J. Uszkoreit, M. Lucic, and A. Dosovitskiy, "MLP-mixer: An all-MLP architecture for vision," in *Advances in Neural Information Processing Systems*, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021.

[5] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research*, vol. 12, no. 7, 2011.

[6] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[7] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, 2012.

[8] N. Le Roux, P.-A. Manzagol, and Y. Bengio, "Topmoumoute online natural gradient algorithm," in *NIPS2007*, January 2008.

[9] Y. Ollivier, "Riemannian metrics for neural networks," *CoRR*, vol. abs/1303.0818, 2013. [Online]. Available: http://arxiv.org/abs/1303.0818

[10] N. Agarwal, R. Anil, E. Hazan, T. Koren, and C. Zhang, "Disentangling adaptive gradient methods from learning rates," *arXiv preprint arXiv:2002.11803*, 2020.

[11] S.-I. Amari, "Natural gradient works efficiently in learning," *Neural computation*, vol. 10, no. 2, pp. 251–276, 1998.

[12] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint arXiv:1606.08415*, 2016.

[13] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[14] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[15] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," *CoRR*, vol. abs/2003.05991, 2020. [Online]. Available: https://arxiv.org/abs/2003.05991

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[17] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.