FACTOR NORMALIZATION FOR DEEP NEURAL NETWORK MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep neural network (DNN) models often involve features of ultrahigh dimensions. In most cases, the ultrahigh dimensional features can be decomposed into two parts. The first part is a low-dimensional factor model. The second part is the residual feature, with much-reduced variability and interfeature correlation. This leads to a number of interesting theoretical findings for deep neural network training. Accordingly, we are inspired to develop a new *factor normalization* method for fast deep neural network training. The proposed method contains three interesting steps. First, it decomposes the ultrahigh dimensional features into two parts. One is due to factors and the other one is due to residuals. Second, it modifies a given DNN model slightly so that the effect of the latent factor and residual feature can be processed separately. Last, to train a modified DNN model, a new SGD algorithm is developed that allows adaptive learning rates for the factor part and the residual part. A number of empirical experiments are presented to demonstrate its superior performance. The code is available at `https://github.com/HazardNeo4869/FactorNormalization`

1 INTRODUCTION

In recent decades, the progress of deep learning, together with advances in GPU devices, has led to growing popularity of deep neural network (DNN) models in both academia and industry. DNN models have been widely used in various fields, such as image classification (Simonyan & Zisserman, 2014; He et al., 2016a), speech recognition (Hinton et al., 2012; Maas et al., 2017), and machine translation (Wu et al., 2016; Vaswani et al., 2017). However, training deep neural networks is a process that highly depends on empirical experience and is extremely time consuming. Therefore, it is necessary to develop a novel methodology for fast DNN training.

In this regard, the stochastic gradient descent (SGD) algorithm has been widely used (Robbins & Monro, 1951). A standard SGD algorithm has two important features. First, it randomly divides the whole dataset into a number of nonoverlapping batches. Second, SGD only computes the first-order derivatives (i.e., the gradient) on batches. The parameters are then updated accordingly. By doing so, the SGD algorithm can handle large-sized datasets with limited computational resources. This makes the SGD algorithm the most popular optimization method for DNN training. Unfortunately, the practical feasibility of SGD comes at the cost of slow convergence (Nocedal, 2016). This may partially due to the fact that SGD only computes the gradient directions, while the information contained in the Hessian matrix is ignored.

For better convergence speed, various modified SGD algorithms have been developed. They can be roughly divided into two classes. The first class of methods attempts to use different update directions or adaptive learning rates. This class includes various momentum methods, such as the popularly used momentum method (Qian, 1999) and the Nesterov accelerated gradient (NAG) (Nesterov, 1983; Bengio et al., 2013). It also includes various adaptive gradient methods, such as AdaGrad (Duchi et al., 2011), AdaDelta (Zeiler, 2012), RMSprop (Tieleman & Hinton, 2012), and Adam (Kingma & Ba, 2014). The second class of methods try to determine the learning rates in a smart way. For example, Loshchilov & Hutter (2016)proposed a warmup strategy, which suggests a very low learning rate for the early training stage and gradually increases it to a

"regular" learning rate. (Vaswani et al., 2017; Devlin et al., 2018; Gotmare et al., 2019). Another popularly used strategy is SGD with cyclical learning rates (Smith, 2017). It seems that none of these optimization methods has considered the covariance structure of ultrahigh dimensional input features. The feature covariance is a piece of information closely related to the Hessian matrix of the loss function. Thus, it should be extremely useful for optimization.

In fact, ample amounts of empirical experience suggest that most ultrahigh dimensional features are of a strong factor type covariance structure. In other words, a significant amount of the feature variability can be explained by a latent factor with very low dimensions. As a consequence, we can decompose the original feature into two parts. The first is the factor part, which accounts for a large proportion of volatility but with a very low dimension. The second is the residual part, which is of the same dimension as the original feature but has the factor effects removed. Consequently, it has a smaller variability contribution. Moreover, the interfeature correlation is also reduced substantially. Accordingly, the original learning problem with respect to the ultrahigh dimensional features can be decomposed into two sublearning problems. The first problem is a learning problem with respect to the latent factor. This is relatively simple because the dimension of the factor is very low. The second problem is related to the residual feature. Unfortunately, this is still a challenging problem with ultrahigh dimensions. However, compared with the original one, it is much easier because the interfeature dependence has been substantially reduced.

The above interesting discussion suggests that for ultrahigh dimensional features with a strong factor structure, new models and optimization methods should be developed for better DNN training. To this end, we propose a novel factor normalization (FN) method. The new method is designed to enhance the performance of a given DNN model. The proposed method contains three interesting steps. First, it decomposes an ultrahigh dimensional feature into two parts. One is due to factor, and the other is due to residual. Second, it modifies a given DNN model slightly so that the effect of the latent factor and residual feature can be processed separately. Last, to train a modified DNN model, a new SGD algorithm is developed, which allows different learning rates for the factor part and the residual part. The rest of this article is organized as follows. Section 2 develops our theoretical motivation with statistical insights. Section 3 provides the details of the proposed method. Section 4 demonstrates the empirical experiments. Section 5 concludes the article with a brief discussion for future research.

2 Theoretical Motivation

To motivate our new method, we propose a number of interesting theoretical motivations from different perspectives. Since an SGD algorithm is a stochastic version of a GD algorithm, we thus focus on a standard GD algorithm in this section for simplicity.

2.1 The GD Algorithm

Let (X_i, Y_i) be the observation collected from the i-th instance with $1 \leq i \leq N$, where Y_i is often the class label and $X_i = (X_{i1}, ..., X_{ip})^\top \in \mathbb{R}^p$ is the associated p-dimensional feature. The loss function evaluated at i is defined as $\ell(Y_i, X_i^\top \theta)$, where $\theta \in \mathbb{R}^p$ is the unknown parameter. Then, the global loss is given by $\mathcal{L}_N(\theta) = N^{-1} \sum_{i=1}^N \ell(Y_i, X_i^\top \theta)$. The global gradient is given by $\dot{\mathcal{L}}_N(\theta) = N^{-1} \sum_{i=1}^N \dot{\ell}(Y_i, X_i^\top \theta) X_i$ and $\dot{\ell}(Y_i, z) = \partial \ell(Y_i, z)/\partial z$. Let $\hat{\theta}^{(t)}$ be the estimator obtained in the t-th iteration. Then, the GD algorithm updates the parameter as $\hat{\theta}^{(t+1)} = \hat{\theta}^{(t)} - \alpha \dot{\mathcal{L}}_N(\hat{\theta}^{(t)})$. Here, α is a scalar and is referred to as the learning rate (Robbins & Monro, 1951). Assume that $\mathcal{L}_N(\hat{\theta}^{(t)})$ reaches its minimum at $\hat{\theta}$ such that $\dot{\mathcal{L}}_N(\hat{\theta}) = 0$. We then apply Taylor expansion for $\dot{\mathcal{L}}_N(\theta)$ at $\hat{\theta}$. This leads to

$$\begin{aligned} \hat{\theta}^{(t+1)} - \hat{\theta} &= \left\{ I_p - \alpha \ddot{\mathcal{L}}_N(\hat{\theta}) \right\} (\hat{\theta}^{(t)} - \hat{\theta}) + o(\|\hat{\theta}^{(t)} - \hat{\theta}\|^2) \\ &= \kappa(\hat{\theta}^{(t)} - \hat{\theta}) + o(\|\hat{\theta}^{(t)} - \hat{\theta}\|^2), \end{aligned}$$

where $\kappa = I_p - \alpha \ddot{\mathcal{L}}_N(\hat{\theta})$. We refer to κ as a *contraction operator*, and it plays a very important role in optimization. Intuitively, all the eigenvalues of κ should lie in $(-1, 1)$. Otherwise, the algorithm

might not converge numerically. Furthermore, let $\|A\|$ denote the maximum absolute eigenvalue of an arbitrary symmetric matrix A. Then, the size of $\|\kappa\|$ controls the convergence speed. Obviously, the smaller $\|\kappa\|$ is, the faster convergence is.

Proposition 1 *Assume $\ddot{\mathcal{L}}_N(\hat{\theta})$ to be a positive definite matrix. Let $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_p > 0$ be the eigenvalues of $\ddot{\mathcal{L}}_N(\hat{\theta})$. To have the GD algorithm converge, we should have $\alpha < 1/\lambda_1$.*

2.2 Slow Convergence under factor structure

As we mentioned previously, most ultrahigh dimensional features are of a strong factor model structure. Consequently, we should have only a few very large top eigenvalues and most other eigenvalues should be very small. To illustrate the idea, assume X_i has a one-dimensional factor structure as $X_{ij} = Z_i + e_{ij}$, with Z_is and e_{ij}s independently generated from a standard normal distribution. Consider a linear regression model as $Y_i = X_i^\top \theta + \varepsilon_i$ for some $\theta \in \mathbb{R}^p$ with true value θ_0. Consider a standard l_2 loss function $\mathcal{L}_N(\theta) = \|Y - X\theta\|^2/N$, where $Y = (Y_1..., Y_N)^\top \in \mathbb{R}^N$, $X = (X_1..., X_N)^\top \in \mathbb{R}^{N\times p}$. The corresponding Hessian matrix is given by $\ddot{\mathcal{L}}_N(\theta) = 2\hat{\Sigma} = 2X^\top X/N$,where $\hat{\Sigma}$ is an estimator for $\Sigma = \mathrm{E}(X_i X_i^\top)$. Then, we have the following proposition.

Proposition 2 *Assume $X_{ij} = Z_i + e_{ij}$ with Z_is and ε_{ij}s are independently generated from a standard normal distribution. Then, the eigenvalues of Σ are given by $\lambda_1 = p + 1$, $\lambda_2 = \cdots = \lambda_p = 1$.*

Proposition 2 studies a highly simplified factor model structure for illustration purposes. However, it seems to be sufficiently illuminating. In this case, we have the largest eigenvalue of Σ is given by $\lambda_1 = p + 1$. Then, by Proposition 1, we know that $\alpha < 1/2\lambda_1 = 1/2(p + 1)$; otherwise, the GD algorithm might not converge. Accordingly, we have $||\kappa|| = (1 - 2\alpha\lambda_p) > 1 - 1/(p + 1) \approx 1$. This explains why SGD converges so slowly. To further illustrate the problem, define $\hat{\theta}^{(0)} = (0, 0, \cdots, 0)^\top \in \mathbb{R}^p$. Given $\hat{\theta}^{(t)}$, by a standard GD algorithm, one can verify that $\hat{\theta}^{(t+1)} = \hat{\theta}^{(t)} - 2\alpha N^{-1} X^\top (Y - X\hat{\theta}^{(t)})$. Then, we have the following proposition.

Proposition 3 *Let $\hat{\theta}^{(t)}$ be the GD estimator obtained in the t-th step with $\hat{\theta}^{(0)} = (0, 0, \cdots, 0)^\top \in \mathbb{R}^p$. Assuming $p < N$, then $\hat{\theta}^{(t)}$ has the explicit solution form as $\hat{\theta}^{(t)} = \{I - (I - 2\alpha\hat{\Sigma})^t\}\hat{\theta}_{OLS}$, where $\hat{\theta}_{OLS} = \left(X^\top X\right)^{-1} X^\top Y$ and $\hat{\Sigma} = X^\top X/N$.*

By Proposition 3, we know that $||\hat{\theta}^{(t)} - \hat{\theta}_{OLS}|| \leq ||\hat{\theta}_{OLS}||^2 \|(I - 2\alpha\hat{\Sigma})^t\| \leq ||\hat{\theta}_{OLS}||^2 (1 - 2\alpha\lambda_p)^t$. Recall that $\alpha < 1/2(p + 1)$ and $\lambda_p = 1$ under the above factor model structure. We thus have $||\hat{\theta}^{(t)} - \hat{\theta}_{OLS}|| \leq ||\hat{\theta}_{OLS}||^2 \{1 - 1/(p + 1)\}^t$. This indicates a very slow convergence when p is large. It seems to suggest that, for a model with ultrahigh dimensional features, a standard GD (or SGD) with a universal learning rate can never converge quickly.

2.3 The Factor Linear Subspace

To further motivate the proposed method, we provide some theoretical justification from different perspectives. Assume a slightly more complicated factor model as $X_i = BZ_i + \mathcal{E}_i$, where $Z_i \in \mathbb{R}^d$ is a vector with low dimensionality (i.e., $d \ll p$) and $B \in \mathbb{R}^{p\times d}$ is the corresponding factor loading matrix with $d \ll p$. Consider a general loss function $\mathcal{L}(\theta)$. Recall that the global gradient is $\dot{\mathcal{L}}_N(\theta) = N^{-1}\sum_{i=1}^N \dot{\ell}(Y_i, X_i^\top\theta)X_i$. Then, we have $\dot{\mathcal{L}}_N(\theta) = Q_1 + Q_2$, where $Q_1 = BN^{-1}\sum_{i=1}^N \dot{\ell}(Y_i, X_i^\top\theta)Z_i \in \mathcal{S}(B)$, $Q_2 = N^{-1}\sum_{i=1}^N \dot{\ell}(Y_i, X_i^\top\theta)\varepsilon_i$, and $\mathcal{S}(B)$ represents the linear subspace spanned by the column vectors of B. Then, the covariance structure of the global gradient can be written as $\mathrm{cov}\{\dot{\mathcal{L}}_N(\theta)\} = \mathrm{cov}(Q_1) + \mathrm{cov}(Q_2)$, where $\mathrm{cov}(Q_1) = B\Sigma_z B^\top/N$, $\Sigma_z = \mathrm{cov}\{\dot{\mathcal{L}}(Y_i, X_i^\top\theta)Z_i\} \in \mathbb{R}^{d\times d}$, $\mathrm{cov}(Q_2) = \Sigma_\varepsilon/N$, and $\Sigma_\varepsilon = \mathrm{cov}\{\dot{\mathcal{L}}(Y_i, X_i^\top\theta)\varepsilon_i\} \in \mathbb{R}^{d\times d}$. It is then of interest to study the relative sizes of $\mathrm{cov}(Q_1)$ and $\mathrm{cov}(Q_2)$ under appropriate metrics.

Proposition 4 *Assume there exists constant $0 < \tau_{min} < \tau_{max} < \infty$, such that $\tau_{min} < \lambda_{min}(\Sigma_z) \leq \lambda_{max}(\Sigma_z) < \tau_{max}$ and $\tau_{min} < \lambda_{min}(\Sigma_\varepsilon) \leq \lambda_{max}(\Sigma_\varepsilon) < \tau_{max}$. Furthermore, assume that $\lambda_{min}(B^\top B/p) > \tau_{min}$. We then have $\mathrm{tr}\{\mathrm{cov}(Q_1)\}/\mathrm{tr}\{\mathrm{cov}(Q_2)\} \geq \tau_{min}^2/\tau_{max}$.*

For most DNN models, the parameter dimension p is usually very large. This makes the model structure sufficiently flexible. Accordingly, one might expect the associated GD (or SGD) algorithm should search the entire ultrahigh dimensional parameter space (e.g., $\mathbb{R}^p$) in a very flexible way. However, the above proposition indicates that the estimated gradient direction is not as flexible as we might expect. In contrast, it always has a significant portion (i.e., Q_1) trapped in a very low-dimensional linear subspace $\mathcal{S}(B)$. This brings a positive effect. Since the linear subspace $\mathcal{S}(B)$ is of a very low dimension (i.e., d), the overfitting effect due to Q_1 becomes negligible. However, this positive effect also comes at a significant cost. By trapping a significant portion of the gradient direction in $\mathcal{S}(B)$, a GD algorithm's capability to explore directions other than $\mathcal{S}(B)$ is considerably sacrificed. This problem can be solved if the directions due to the factor (i.e., Q_1) and directions due to the residual features (i.e., Q_2) can be treated separately. This is another important theoretical insight that drives us to develop our new method.

3 THE PROPOSED NEW METHOD

Inspired by the theoretical findings discussed in the previous subsection, we develop below a novel methodology for fast DNN model training under a factor model structure. For convenience, we refer to the new method as a factor normalization (FN) method. The proposed method contains the following three key components. They are factor decomposition, model reconstruction, and adaptive learning. The details are discussed in the following subsections.

3.1 FACTOR DECOMPOSITION

In this step, we conduct a standard principal component analysis for the ultrahigh dimensional features. By doing so, a low-dimensional factor can be estimated. Accordingly, we can decompose the original features into two parts. The first part is due to the factor and is referred to as the factor part. The second part is the original features but has factor effects removed. This part is referred to as the residual part (or residual feature). Specifically, the input feature matrix is written as $X = (X_1..., X_N)^\top \in \mathbb{R}^{N\times p}$. We should centralize X by column so that its column mean is 0. Recall that the covariance matrix can be estimated by $\hat{\Sigma} = X^\top X/N$. Next, we perform eigenvalue decomposition on $\hat{\Sigma}$ as $\hat{\Sigma} = U\Lambda U^\top = \sum_{k=1}^{p} \lambda_i u_i u_i^\top$, where $U = (u_1..., u_p)$ is an orthogonal matrix and $\Lambda = \text{diag}(\lambda_1..., \lambda_p)$ is a diagonal matrix with $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_p \geq 0$. Define $U_d = (u_1..., u_d)^\top$. We then estimate the latent factor by $\hat{Z} = XU_d^\top$, where $\hat{Z} \in \mathbb{R}^{N\times d}$ represents the estimated factor part. Last, we use a linear regression to calculate the residual feature matrix as follows $\hat{\mathcal{E}} = X - \hat{Z}\hat{B}$: where $\hat{B} = (\hat{Z}^\top \hat{Z})^{-1}\hat{Z}^\top X \in \mathbb{R}^{d\times p}$ is the estimated factor loading matrix. By doing so, the original feature matrix X is decomposed into two parts: the factor part $\hat{Z}$ and the residual part $\hat{\mathcal{E}}$.

3.2 MODEL RECONSTRUCTION

In this step, we modify a given neural network model (e.g., a ResNet) into a new one for substantially improved performance. More specifically, consider a standard DNN model with (1) an input layer with original feature X; (2) an output layer, which is often of a dense type structure, and (3) sophisticated latent layers between the input and output layers. We then replace the original feature of the input layer (i.e., X) with the residual feature (i.e., $\hat{\mathcal{E}}$). Next, let the residual feature be fully processed by the sophisticated latent layers until the output layer. Before the DNN model constructs the last dense layer for output, we reinsert the estimated latent factor (i.e., $\hat{Z}$) back to the output layer to compensate for the information loss due to factor decomposition. Then, a dense type neural network can be constructed for the final output. This leads to a slightly modified new network structure. By doing so, the factor part and the residual part can be treated separately without any information loss. We thus refer to this step as a model reconstruction; see Figure 1 for a more intuitive explanation.

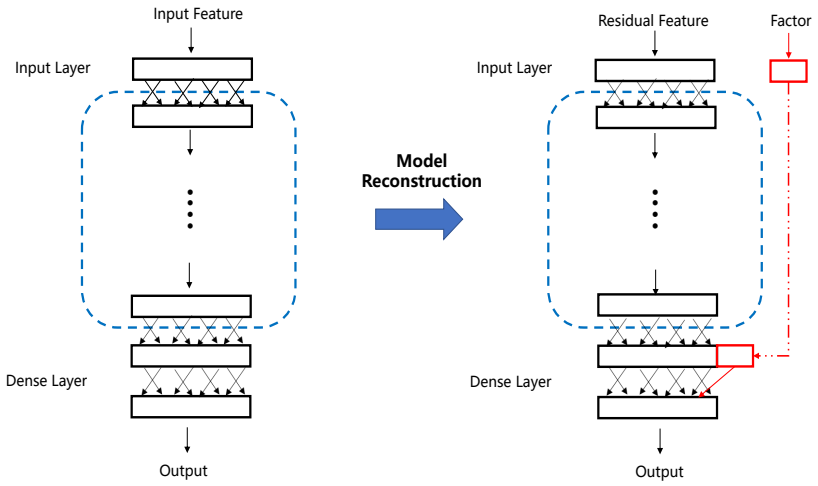

Figure 1: A graphic illustration of MODEL RECONSTRUCTION for factor normalization. The rectangles represent the feature maps in a DNN. The blue dashed rectangle represents the latent layers between the input and the last dense layer. The left panel shows an original DNN structure with the original input feature. The right panel shows a modified DNN structure with factor and residual features. As one can see, the factor and residual feature are treated separately. The latent factor is reinserted back into the model before the last dense layer for output.

3.3 ADAPTIVE LEARNING

As we mentioned before, a standard SGD algorithm cannot be used to train a DNN model. Otherwise, the convergence speed can still be very slow. For a fast convergence speed, a small learning rate should be used for the parameters associated with the factor because factor variability is large. In contrast, much larger learning rates can be used for the parameters associated with the residual features since the convergence condition for the residual part is much improved. This is because the factor part is removed and uncorrelated with the residual part. Since the number of parameters associated with residual features is very large, it is practically infeasible to assign different learning rates for them. Accordingly, the learning rates used for those parameters are set to be the same. More specifically, assume the learning rate used in standard SGD is α. We then set the learning rate for the factor -associated parameter to be $\alpha_j = \alpha\lambda_1/\lambda_j$ for $j = 1..., d$. Recall that λ_j is the jth largest eigenvalue of $\hat{\Sigma}$. In contrast, the learning rates used for the residual associated parameters are set to be the same, which is $\alpha_\varepsilon = \alpha\lambda_1/\lambda_{d+1}$. The consequence is that the learning rates used for the latent factors become much smaller than those of the residual features

4 EXPERIMENTS

To empirically demonstrate the proposed FN method, we conducted various experiments on different models, including logistic regression, multilayer fully connected neural networks and deep convolutional neural networks.

4.1 LOGISTIC REGRESSION

We start with a simple logistic regression model and evaluate the proposed method on the MNIST dataset. This is a classification problem with 10 classes, 60,000 instances for training and 10,000 instances for validation. The input feature is a 28×28 pixel matrix. To implement the logistic regression, we reshape the input feature into 784 dimension vectors. Subsequently, the FN methods

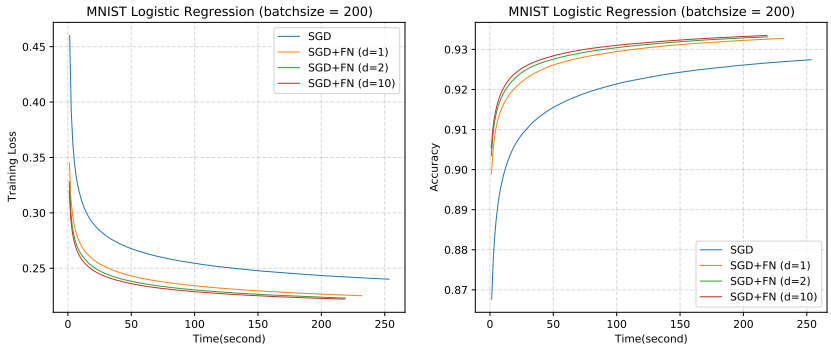

Figure 2: The left panel shows the training loss on the MNIST dataset for the logistic regression model for the algorithms with and without the FN method. The right panel shows the prediction accuracy on the validation dataset. Different numbers of factors are tested.

with various factor dimensions ($d = 1, 2, 10$) are used to reconstruct the model. Adaptive learning rates as suggested in section 3.3 are used for fast training. The resulting empirical performance is compared with the standard SGD method with a learning rate of $\alpha = 0.01$. The detailed results are given in Figure 2. For comparison purposes, all the methods are trained for 200 epochs with the batch size fixed to 200. The left panel of Figure 2 reports the training loss for various methods. We find that the FN methods with various factor dimensions consistently outperform a standard SGD method. Additionally, little difference is observed between different FN methods. Similar observations are also obtained for the prediction accuracy on the validation dataset. See the right panel of Figure 2 for details. We are then inspired to fix $d = 1$ for the subsequent experiments.

4.2 Multilayer Neural Networks

We next consider a more complicated multilayer neural network model. The proposed model has two fully connected hidden layers with 1,000 units. The ReLU transformation is used for activation (Kingma & Ba, 2014). The experiment is also conducted on MNIST with batch sizes of 200 and 200 training epochs. For simplicity, the FN method is conducted with factor dimension $d = 1$. The learning rates are set to be 0.05, 0.01 and 0.005. The detailed results are given in Figure 3. The top panel of Figure 3 reports the training loss in log scale for various methods with different learning rates. We find that the FN methods always achieve smaller training loss than the standard SGD method. The bottom panel of Figure 3 reports the prediction accuracy on validation datasets. The findings are quantitatively similar to those for training loss.

4.3 Convolutional Neural Networks

Convolutional neural networks (CNNs) are the most popular models used in image classification and object detection tasks. The main difference between CNNs and multilayer neural networks is that CNNs contain many convolutional and pooling layers, which provide efficient feature extraction and lead to excellent prediction performance. However, CNNs usually have complicated model structures with a large number of parameters. Here, we consider two classical CNN models. They are AlexNet (Krizhevsky et al., 2012) and ResNet50 (He et al., 2016b). The dataset used here is CIFAR10 (Krizhevsky & Hinton, 2009). The results of AlexNet are summarized in Figure 4, while those of ResNet50 are given in Figure 5. For comparison purposes, a total of four baseline methods are also evaluated. They are SGD (Robbins & Monro, 1951), NAG (Bengio et al., 2013), AdaGrad (Duchi et al., 2011) and Adam (Kingma & Ba, 2014).

The left panel of Figure 4 reports the time consumption required by different methods (i.e., FN, SGD, NAG, AdaGrad and Adam) to reach the 80% prediction accuracy in the validation dataset. We find that for all four baseline algorithms (i.e., SGD, NAG, AdaGrad and Adam), the time is substantially reduced if the FN technique is used. The reduction is significant in the case of SGD and NAG. The middle panel of Figure 4 compares training loss in log scale. For a fair comparison,

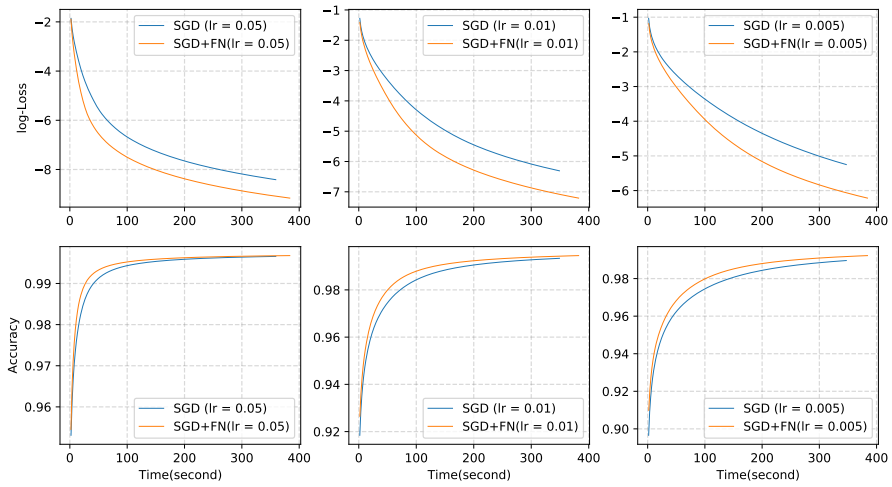

Figure 3: The top panel shows the log training loss of SGD (blue line) versus SGD+FN (orange line) under different learning rates (i.e., $\alpha = 0.05, 0.01, 0.005$). The bottom panel shows the validation accuracy of SGD (blue line) versus SGD+FN (orange line) under the same learning rates.

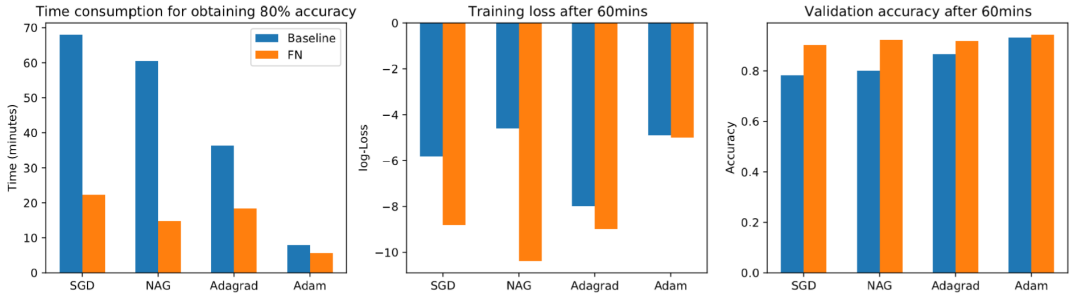

Figure 4: Results on AlexNet. The left panel shows the time cost for each algorithm to obtain an 80% out-of-sample accuracy on the CIFAR10 dataset with and without the FN method; The middle panel shows the log training loss after 1 hour for the four algorithms with and without FN. The right panel shows the out-of-sample accuracy after 1 hour for the four algorithms with and without FN.

60 minutes are given for each method. We find that the training loss can be significantly reduced if the FN method is adopted. The right panel of Figure 4 summarizes the prediction accuracy on the validation dataset after 60 minutes of training. We find that for all the FN-based algorithms, the validation accuracy is higher than their standard counterparts. Overall, we find that the FN technique can significantly improve the performance of a baseline method in terms of time cost, training loss, and prediction accuracy. The results of ResNet50 given in Figure 5 are qualitatively similar. However, it is remarkable that in the middle panel of Figure 5, the training loss for NAG with FN is not as good as the baseline NAG algorithm. Nevertheless, we find that its validation accuracy is still better than NAG without FN. This can be seen from the right panel of Figure 5.

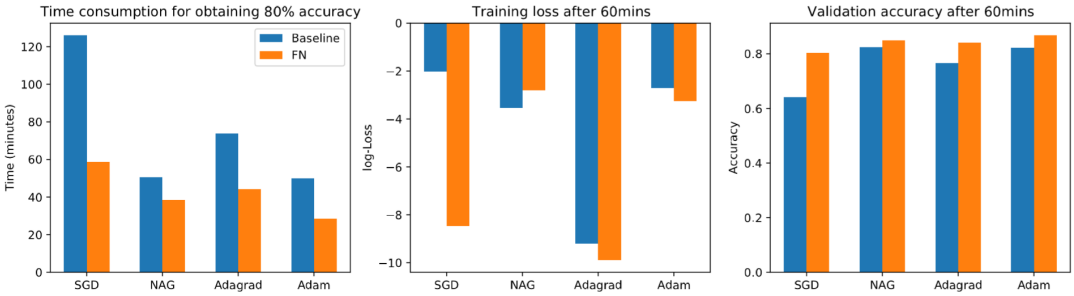

Figure 5: Results on ResNet50. The left panel shows the time cost for each algorithm to obtain an 80% out-of-sample accuracy on the CIFAR10 dataset with and without the FN method. The middle panel shows the log training loss after 1 hour for the four algorithms with and without FN. The right panel shows the out-of-sample accuracy after 1 hour for the four algorithms with and without FN.

5 Conclusion

In this paper, we propose a novel *factor normalization* method for fast deep neural network training. The idea was inspired by the fact that many DNN models involve ultrahigh dimensional features and these features are often of a strong factor structure. The proposed method has three key components. First, it decomposes an ultrahigh dimensional input feature into two parts. One is the factor part with low dimensionality, and the other is the residual part. Second, it modifies a given DNN model slightly so that the effect of the latent factor and residual feature can be processed separately. Last, to train a modified DNN model, a new SGD algorithm is developed that allows adaptive learning rates for the factor part and the residual part.

To conclude, this article, we present here a number of interesting topics for future study. First, our current FN method conducts factor decomposition on the input feature. In fact, even if different components of the input feature are completely independent of each other, the resulting feature map after a number of convolutional layers might still have a strong factor structure. In this case, whether factor decomposition should be conducted for the feature maps is a problem worth studying. Second, the current FN method does not insert the estimated factors back into the DNN models until the last layer. What would happen if the estimated factors were also inserted into earlier layers? This is another problem of great interest.

Acknowledgments

This research is supported by the National Natural Science Foundation of China (No.71702185, No.71873137, No.11971504, No.71532001, No.11525101, No.71332006), Beijing Municipal Social Science Foundation (No.19GLC052), National Statistical Science Research Project (No.2020LZ38), Ministry of Education Focus on Humanities and Social Science Research Base (Major Research Plan 17JJD910001) and China's National Key Research Special Program (No. 2016YFC0207704).

References

Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in optimizing recurrent networks. In *IEEE International Conference on Acoustics*, 2013.

Jacob Devlin, Ming Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 2018.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(7):257–269, 2011.

Akhilesh Gotmare, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation. In *International Confer-*

ence on Learning Representations, 2019. URL https://openreview.net/forum?id=r14EOsCqKX.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Sun Jian. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. 2016b.

G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, and T. N. Sainath. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *Computer ence*, 2014.

A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Computer Science Department, University of Toronto, Tech. Rep*, 1, 01 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. pp. 1097–1105, 2012.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. 2016.

Andrew L. Maas, Peng Qi, Ziang Xie, Awni Y. Hannun, Christopher T. Lengerich, Daniel Jurafsky, and Andrew Y. Ng. Building dnn acoustic models for large vocabulary speech recognition. *Computer Speech and Language*, 41:195–213, 2017.

Y. E. Nesterov. A method for solving the convex programming problem with convergence rate $o(1/k^2)$. *Dokl.akad.nauk Sssr*, 269, 1983.

Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2), 2016.

Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Netw*, 12(1): 145–151, 1999.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.

Leslie Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2017.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6e rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 2012.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Matthew D. Zeiler. Adadelta: An adaptive learning rate method. *Computer ence*, 2012.