Weight Features for Predicting Future Model Performance of Deep Neural Networks

Yasunori Yamada, Tetsuro Morimura
IBM Research - Tokyo
19-21 Nihonbashi, Hakozaki-cho, Chuo-ku, Tokyo, Japan
{ysnr,tetsuro}@jp.ibm.com

Abstract

Deep neural networks frequently require the careful tuning of model hyperparameters. Recent research has shown that automated early termination of underperformance runs can speed up hyperparameter searches. However, these studies have used only learning curve for predicting the eventual model performance. In this study, we propose using weight features extracted from network weights at an early stage of the learning process as explanation variables for predicting the eventual model performance. We conduct experiments on hyperparameter searches with various types of convolutional neural network architecture on three image datasets and apply the random forest method for predicting the eventual model performance. The results show that use of the weight features improves the predictive performance compared with use of the learning curve. In all three datasets, the most important feature for the prediction was related to weight changes in the last convolutional layers. Our findings demonstrate that using weight features can help construct prediction models with a smaller number of training samples and terminate underperformance runs at an earlier stage of the learning process of DNNs than the conventional use of learning curve, thus facilitating the speed-up of hyperparameter searches.

1 Introduction

Deep neural networks (DNNs) continue to deliver state-of-the-art performances on a variety of machine learning tasks such as image processing [Krizhevsky *et al.*, 2012] and speech recognition [Hinton *et al.*, 2012a]. The parameters of the DNN, called network weights, are trained using back-propagation, unsupervised learning, or other discriminative algorithms [Schmidhuber, 2015].

While DNNs have enjoyed many successes, it is well known that they are difficult to optimize, especially for non-experts. The difficulty mostly comes from the fact that the performance of the DNN heavily depends on the setting of model hyperparameters [Snoek *et al.*, 2012] that include the learning rate, the number of layers, the dropout parameters [Hinton *et al.*, 2012b], and the selection of activation functions [Nair and Hinton, 2010]. One needs to carefully tune the hyperparameters by either manual control or an automated method. There are already several approaches to automated hyperparameter optimization, including a simple grid search, the gradient search [Bengio, 2000], random search with a low effective dimensionality [Bergstra and Bengio, 2012], and Bayesian optimization [Snoek *et al.*, 2012; Bergstra *et al.*, 2013]. While these automated methods can outperform human experts in terms of optimization of the model performance, they require many long training runs until they are convergent and usually come at a high computational cost.

In contrast, human experts can make the optimization more efficient by terminating a simulation run as soon as possible if the run seems to do poorly. In recent years, an automated early termination approach has been proposed in which the eventual model performance at the end of the learning process is predicted [Hara *et al.*, 2014; Swersky *et al.*, 2014; Domhan *et al.*, 2015]. This previous study reported that early termination combined with hyperparameter optimization methods can speed up hyperparameter searches almost two-fold [Domhan *et al.*, 2015]. However, previous research on this topic has been limited to use of the model performance history, which is called the learning curve, and has not examined any other features related to neural networks such as network weights or their changes.

In this study, we propose the use of features extracted from network weights for predicting the eventual model performance of DNNs and compare the prediction performance with the use of learning curve. We conducted hyperparameter searches of DNNs with various types of hyperparameter on three popular datasets for image recognition: the MNIST [LeCun *et al.*, 1998], CIFAR-10 [Krizhevsky, 2009], and ImageNet 2012 datasets [Russakovsky *et al.*, 2015]. Through the experiments, we demonstrate how the use of weight features improves the prediction performance and what kinds of weight feature mainly contribute to this improvement.

2 Prediction of future model performance

We predict the eventual model performance of DNNs using features that can be calculated from the DNNs at an early stage of the learning process. We train n DNNs with different hyperparameters until epoch T, where T is the last epoch

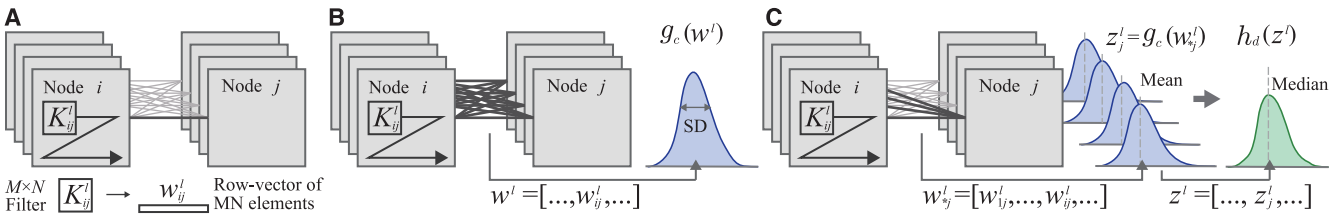

Figure 1: Proposed network weight features in the case of convolutional layer. (A) Definition of $\boldsymbol{w}_{ij}^l$ using $\boldsymbol{K}_{ij}^l$. (B)(C) Examples of weight features calculated by Eqs. (1) and (3), respectively.

of the learning iterations for updating weight parameters. We then obtain n pairs of the eventual model performances at epoch T and features calculated from the DNNs until epoch t ($t \ll T$). Here, n is the training sample number for a prediction model and t is the pre-run epoch for DNNs. We next construct a model that predicts the eventual model performance using the features as explanation variables. In this study, we propose novel features for the explanation variables of the prediction model for the eventual model performance, which are extracted from network weights. We finally evaluate the predictive performance by comparing this model with the conventional approach in which the prediction model uses learning curves as the explanation variables. Better features for the prediction model make it possible to save the number of training samples n and accomplish target predictive accuracies at an earlier stage of the learning process t of DNNs, which will enable more efficient search of the hyperparameters of DNNs.

Here, we describe how to extract the proposed weight features from the DNSs. The weight features were calculated in each layer of DNNs. We first define weight vector $\boldsymbol{w}_{ij}^l$ in layer l that connects between node i and node j (Figure 1A). In the case of convolutional layers, $\boldsymbol{w}_{ij}^l$ is a weight row-vector of MN elements, which consists of weight parameter $M \times N$ filter $\boldsymbol{K}_{ij}^l$ and $\boldsymbol{w}_{ij,(a-1)N+b}^l = \boldsymbol{K}_{ij,ab}^l$. Using $\boldsymbol{K}_{ij}^l$, convolutional layer output $y_{j,ab}^l$ in the (a,b) component of node j is calculated as

$$y_{j,ab}^l = f\left(\sum_i \sum_{c=0}^{M-1} \sum_{d=0}^{N-1} K_{ij,cd}^l x_{i,(a+c)(b+d)}^l + b_j^l\right),$$

where f is activation function, $x_{i,ab}^l$ is input to convolutional layer l in (a,b) component of node i, and b_j^l is bias input. In the case of fully connected layers, $\boldsymbol{w}_{ij}^l$ is a scalar value and weight between neuron i and neuron j. Let $\boldsymbol{w}^l$, $\boldsymbol{w}_{i*}^l$, and $\boldsymbol{w}_{*j}^l$ be defined as $\boldsymbol{w}^l = [..., \boldsymbol{w}_{ij}^l, ...]$, $\boldsymbol{w}_{i*}^l = [\boldsymbol{w}_{i1}^l, ..., \boldsymbol{w}_{ij}^l, ...]$, $\boldsymbol{w}_{*j}^l = [\boldsymbol{w}_{1j}^l, ..., \boldsymbol{w}_{ij}^l, ...]$. The weight features $\boldsymbol{F}(t)$ are calculated using the $\boldsymbol{w}^l(t)$, $\boldsymbol{w}_{i*}^l(t)$, $\boldsymbol{w}_{*j}^l(t)$, and $\boldsymbol{w}_{ij}^l(t)$ of network weights at epoch t, as (e.g., Figures 1B and 1C)

$$g_c\left(\boldsymbol{w}^l(t)\right), \tag{1}$$

$$h_d\left(\boldsymbol{v}^l\right), \boldsymbol{v}^l = [..., v_i^l, ...], v_i^l = g_c\left(\boldsymbol{w}_{i*}^l(t)\right), \tag{2}$$

$$h_d\left(\boldsymbol{z}^l\right), \boldsymbol{z}^l = [..., z_j^l, ...], z_j^l = g_c\left(\boldsymbol{w}_{*j}^l(t)\right), \tag{3}$$

$$h_d\left(\boldsymbol{r}^l\right), \boldsymbol{r}^l = [r_{11}^l, ..., r_{ij}^l, ...], r_{ij}^l = g_c\left(\boldsymbol{w}_{ij}^l(t)\right), \tag{4}$$

where g_c for $c \in \{1, ..., 13\}$ represents functions for calculating mean, quantiles (0.25, 0.5, 0.75), standard deviation, skewness, kurtosis, p-th central moment (p=1,2,3,4,5), and entropy estimated using histograms with k bins of the same size ($k = 32$). The function h_d for $d \in \{1, ..., 5\}$ calculates mean, median, standard deviation, maximum value, and minimum value. In the case of fully connected layers, we did not calculate weight features using $\boldsymbol{w}_{ij}^l(t)$ in the manner of Eq. (4) because $\boldsymbol{w}_{ij}^l(t)$ is a scalar value. In addition, for the $\boldsymbol{w}_{i*}^l(t)$ and $\boldsymbol{w}_{*j}^l(t)$, we calculated vectors $\boldsymbol{m}^l$ and $\boldsymbol{n}^l$ consisting of pair-wise Kolmogorov-Smirnov distances between all possible pairs in $\boldsymbol{w}_{i*}^l(t)$ and $\boldsymbol{w}_{*j}^l(t)$, respectively, and obtained weight features as $h_d\left(\boldsymbol{m}^l\right)$ and $h_d\left(\boldsymbol{n}^l\right)$. In total, we extracted 218 and 153 weight features from one convolutional layer and one fully connected layer, respectively. We also calculated weight features using the weight changes between two epochs $\Delta\boldsymbol{w}_\tau^l(t) = \boldsymbol{w}^l(t) - \boldsymbol{w}^l(t-\tau)$ in the same manner.

Regarding learning curves, we used two standard measures that have been used for evaluating the model performance during the learning process: training errors and validation score [Swersky *et al.*, 2014; Domhan *et al.*, 2015]. Training errors are computed using an output layer with softmax activation followed by cross-entropy on training data, where the neural networks learn to minimize the training errors [Bottou, 2012; Schmidhuber, 2015]. Validation score is defined as the accuracy on test data. In this study, we utilized datasets for image recognition and then used, as the validation score, the top-1 classification accuracy that is the proportion of correctly classified images. The eventual model performance was also judged using this top-1 classification accuracy after all learning processes had been completed. For predicting the eventual model performance, we used the history of both training errors and validation scores until epoch t as the explanation variables.

For the prediction model, we used the random forest algorithm [Breiman, 2001], which is implemented in MATLAB (MathWorks Inc., Natick, MA). To evaluate prediction performance, we trained the prediction model using n training samples and calculated root mean square errors (RMSEs) for the other samples to evaluate prediction performance.

3 Experiments

3.1 Experimental setup

To investigate the relationship between eventual model performance and weight features extracted in partially trained

Table 1: All hyperparameters of DNNs on MNIST and CIFAR-10 datasets. Init.: initialization.

Hyperparameter	Value
Initial learning rate (lr)	$0.05, 0.01, 0.005, ..., 5 \times 10^{-6}$
lr schedule (choice)	{fixed, exp. decay}
γ (exp. decay)	$0.001, 0.0005, 0.0001, ..., 5 \times 10^{-5}$
p (exp. decay)	1.0, 0.75, 0.5
Momentum	0.9, 0.8, 0.7,..., 0.1, 0
Weight decay	$0.05, 0.01, 0.005, ..., 1 \times 10^{-6}$
Batch size	300, 200, 100, 50, 20, 10
No. of conv. layers	6, 5, 4, 3
No. of filters	128, 64, 32, 16
Filter size	7, 5, 3
Act. func. (choice)	{ReLU, Sig, TanH, Abs}
Dropout ratio	0.75, 0.7, 0.65, ..., 0.05, 0
Pooling func. (choice)	{Max, Average}
Initial bias value	$0.001, 0.0001, ..., 1 \times 10^{-7}$
Weight init. (choice)	{Constant, Gaussian}
Gaussian init. σ	$0.1, 0.01, ..., 1 \times 10^{-5}$

DNNs, we performed hyperparameter searches on the typical image recognition benchmarks MNIST [LeCun *et al.*, 1998], CIFAR-10 [Krizhevsky, 2009], and ImageNet 2012 datasets [Russakovsky *et al.*, 2015]. The MNIST dataset consists of 28 $\times$ 28 pixel grayscale images of handwritten digits from 0 to 9. The dataset has 60,000 training and 10,000 validation examples. The CIFAR-10 dataset consists of 32 $\times$ 32 color images in 10 classes such as airplane and bird. The dataset is divided into 50,000 training images and 10,000 validation images. The ImageNet dataset consists of 227 $\times$ 227 color images in 1,000 classes and contains about 1.3 million training images and 50,000 validation images. We preprocessed the images by subtracting the mean values of each pixel of the training images. The number of training iterations for DNNs, T, was set to 2,000, 40,000, and 250,000 for the MNIST, CIFAR-10, and ImageNet datasets, respectively. All experiments in this study were performed using the software package *Caffe* [Jia *et al.*, 2014].

We did hyperparameter searches using a convolutional type of DNN [Deng and Yu, 2014]. To explore a broad range of network types, we targeted the following 17 hyperparameters. The hyperparameters related to network weight updates are learning rate and its schedule, momentum, weight decay, and batch size. The learning rate was either fixed or decaying exponentially at a rate defined as $\alpha_t = \alpha_0(1 + \gamma\ t)^{-p}$, where α_t is learning rate at t and γ and p are hyperparameters. We applied stochastic gradient descent [Bottou, 2012] with momentum in all simulations at the end of each batch. We also examined three hyperparameters related to network architectures: the number of convolutional layers, the number of filters, and the size of filters. Activation functions for convolutional layers and fully connected layers were also considered hyperparameters and we chose within four functions: rectified linear (ReLU) [Nair and Hinton, 2010], Sigmoid (Sig), hyperbolic tangent (TanH), and absolute value (Abs). Dropout [Hinton *et al.*, 2012b] was optionally used on the convolutional layers including the input of the network and its ratio was also one of the hyperparameters examined in this

Table 2: All hyperparameters of DNNs on ImageNet dataset.

Hyperparameter	Value
Initial learning rate	$0.05, 0.02, 0.01, 0.005, 0.002$
Weight decay	$0.005, 0.0025, 0.0005, 0.00025, 0.00005$
No. of conv. layers	6, 5, 4, 3
Dropout ratio	0.75, 0.5, 0.25
Act. func. (choice)	{ReLU, Sig, TanH}

study. In the pooling layers, max or average type of pooling was selected by one hyperparameter. Weights for convolutional and fully connected layers were initialized with either constant values determined by the method [Glorot and Bengio, 2010] or Gaussian distribution, whose standard deviation was also one of the searched hyperparameters. Bias inputs for each layer were initialized to a constant value.

For the MNIST and CIFAR-10 datasets, we changed all of the above 17 hyperparameters (Table 1). The neural networks contain from three to six convolutional layers and one fully connected layer. After each convolutional layer, we set layers for normalizing over local input regions within channels for suppressing extremely large or small outputs. We generated neural networks by changing one hyperparameter within its range while all others were given randomly selected fixed values. We repeated these procedures 20 and 10 times and trained a total of 2,640 and 1,320 DNNs for the MNIST and CIFAR-10 datasets, respectively.

For ImageNet datasets, we used the AlexNet convolutional neural network architecture [Krizhevsky *et al.*, 2012] and changed five hyperparameters: learning rate, weight decay, number of convolutional layers, dropout ratio, and activation functions (Table 2). The DNNs had from three to six convolutional layers and three fully connected layers. We generated a total of 120 DNNs with different hyperparameters.

Because the number of convolutional layers was different between the DNNs, we extracted weight features from network weights of the first, second, and last convolutional layers, which can be defined in all trained DNNs in this study. In addition, we extracted all fully connected layers. We extracted weight features from these layers using both raw weight values and weight changes. For calculating weight changes, we set the parameter τ as 10, 100, and 100 for the MNIST, CIFAR-10, and ImageNet datasets, respectively.

The results of training DNNs showed a broad range of eventual model performance up to 0.996, 0.804, and 0.566 for the MNIST, CIFAR-10, and ImageNet datasets, respectively (Figure 2A). As shown in Figure 2A, there were many DNNs with low scores. Specifically, the runs that showed lower scores less than 0.3 were 40.3%, 49.5%, and 65.8% of all runs for the MNIST, CIFAR-10, and ImageNet datasets, respectively. Figure 2B shows time series of the learning curve (validation scores and training errors) at the early learning processes in which we investigated the prediction performance.

3.2 Prediction performance at each epoch

We constructed prediction models using the random forest algorithm and then evaluated prediction performance by cross-

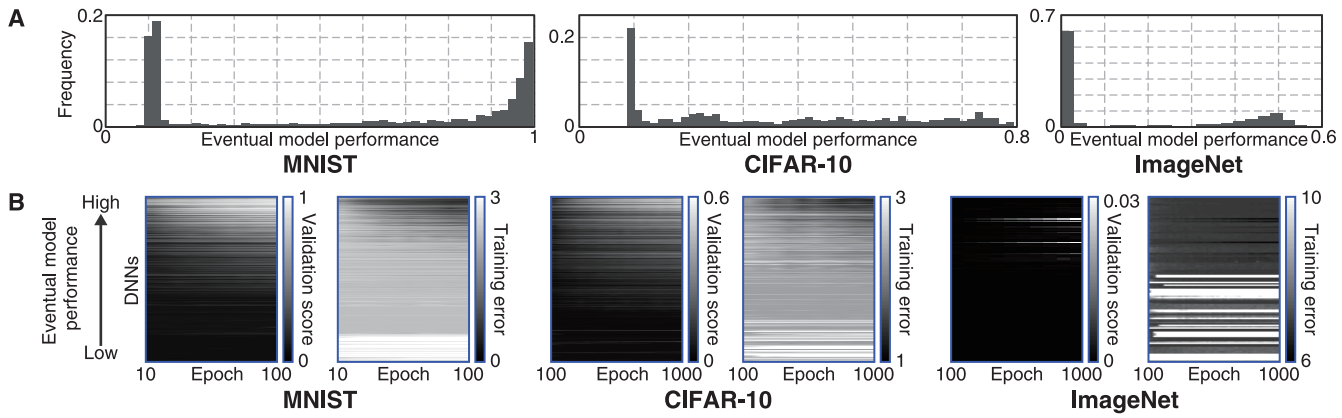

Figure 2: Training results of DNNs. (A) Histograms of eventual model performance on test data. (B) Change in validation scores and training errors at early stage of learning processes in all DNNs. Row represents each DNN and the order is sorted in accordance with eventual model performance.

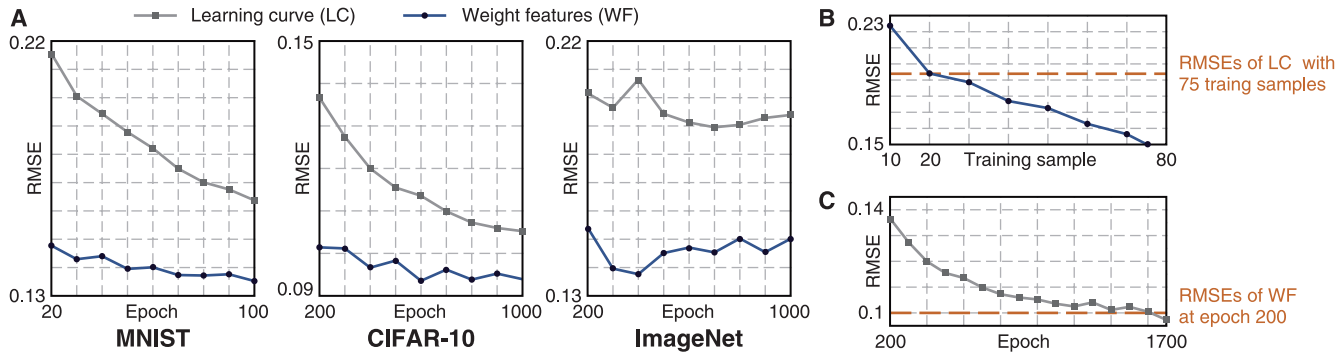

Figure 3: Prediction performance for eventual model performance of DNNs using learning curve or proposed weight features as the explanation variables. (A) RMSEs over the training epoch of DNNs. (B) RMSEs on ImageNet dataset when the number of training samples varies. The training epoch of DNNs was fixed at 1,000. (C) Epoch needed to achieve RMSEs of weight features at epoch 200 for the use of learning curve on the CIFAR-10 dataset.

validation. Regarding the parameter for the random forest algorithm, we set the number of decision trees to 400 through the experiments. We set the number of training samples n for the prediction model to 1,200, 400, and 75 for the MNIST, CIFAR-10, and ImageNet datasets, respectively.

We evaluated the prediction performance over the various pre-run epochs, at which the features were calculated from the DNNs. The results in Figure 3A show that the use of the weight features always provides better prediction performance. Compared with the use of the learning curve, the RMSEs in the case of the weight features at the same epoch decreased by 17-31% over epoch 100 for the MNIST, 10-26% over epoch 1,000 for the CIFAR-10, and 21-33% over epoch 1,000 for the ImageNet datasets.

3.3 Training sample numbers and pre-run epochs

We showed that using the weight features improves prediction performance compared with using the learning curve. Next, we investigated to what extent the use of the weight features can decrease the number of samples n and pre-run epochs t needed for accomplishing the same prediction performance with the learning curve.

First, by decreasing the number of samples for training the prediction model of the weight features, we investigated how many samples are saved for achieving the same prediction performance using the learning curve. Figure 3B shows the relationship between the number of training samples and RMSEs of the prediction model using the weight features at epoch 1,000 for the ImageNet dataset. In this example, the number of required samples declined from 75 for the learning curve to 20 for the weight features. Similarly, the number of samples decreased from 1,200 to 500 at epoch 100 for the MNIST dataset and from 400 to 230 at epoch 1,000 for the CIFAR-10 dataset. On average, relative to the learning curves, the use of the weight features decreased the required sample size by 73% over epoch 100 for the MNIST, 58% over epoch 1,000 for the CIFAR-10, and 76% over epoch 1,000 for the ImageNet dataset.

We also investigated how much earlier the use of the weight features accomplished the same prediction performance of the learning curve by increasing the number of epochs used for the prediction model of the learning curve.

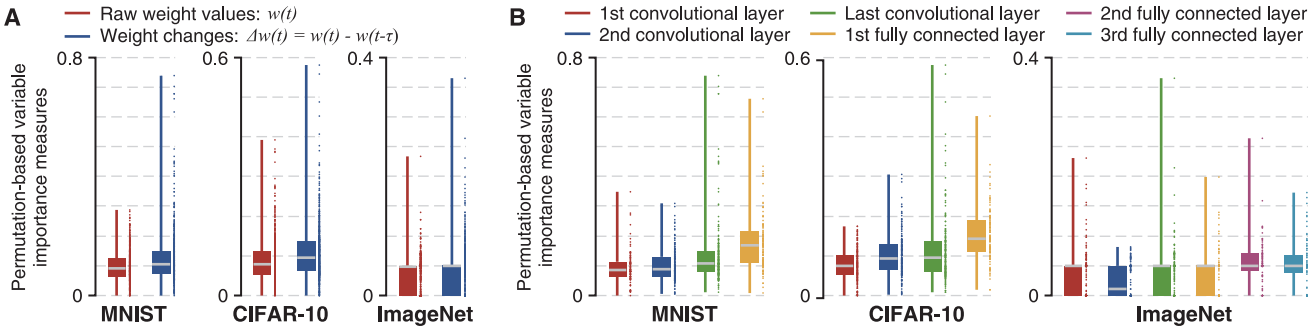

Figure 4: Permutation-based variable importance measures of the weight features for the random forest algorithm. (A) Comparison between those of weight features extracted from raw weight values and weight changes. (B) Comparison among those of features extracted from the weight changes in each layer of DNNs.

For the reference prediction performance, we utilized the RMSEs of 0.15 and 0.1 for the MNIST dataset and CIFAR-10 datasets, respectively. Figure 3C shows the changes in RMSEs in accordance with the increase of pre-run epochs for DNNs on the CIFAR-10 dataset. In this example, the epoch needed for achieving the target prediction performance (RMSE $= 0.1$) increased from epoch 200 for the weight features to epoch 1,700 for the learning curve. For the MNIST dataset, the use of the learning curve also increased the required pre-run epoch from 20 to 180.

3.4 Important variables in weight features

To gain insight into what kinds of weight feature contribute to improving the prediction performance, we analyzed variable importance for each explanation variable using the permutation accuracy importance measure for the random forest as a means of variable selection [Breiman, 2001; Strobl *et al.*, 2007].

First, we compared the importance measures between weight features extracted from raw weight values and weight changes. The result shows that features extracted from weight changes had stronger contributions than those of raw weight values (Figure 4A). Next, we compared them among layers and found that weight features extracted from the last convolutional layers showed the highest value compared with those of the other convolutional layers and fully connected layers (Figure 4B). This result was common to all three datasets investigated in this study.

3.5 Weight history and time difference

To clarify the relationship between the prediction performance and the weight features, we conducted additional investigations. First, we compared the prediction performance between the above weight features and their history until epoch t. The weigh features $\boldsymbol{F}(t)$ were calculated from the two items of weight data at epochs t and $t-\tau$. Here, we investigated the prediction performance using the history of $\boldsymbol{F}(t)$ at certain intervals in the same way as the learning curve. Figure 5A shows RMSEs of the prediction models using the weight history at intervals of epoch 100 for the CIFAR-10 dataset. For example, the weight history at epoch 400 consists of $\boldsymbol{F}(400)$, $\boldsymbol{F}(300)$, and $\boldsymbol{F}(200)$. The result showed that

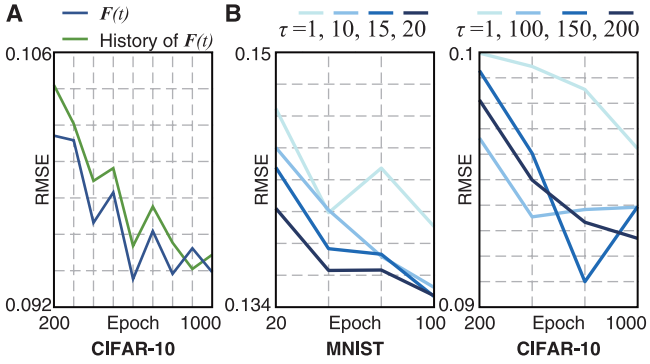

Figure 5: RMSEs of the prediction model associated with change in the weight features. (A) Comparison of weight features $\boldsymbol{F}(t)$ extracted from weights at two epochs and their history. (B) Changes in RMSEs in accordance with increase in time difference τ for calculating weight changes.

there was little improvement of the prediction performance using the weight history, suggesting that when the weight features are extracted in the way proposed in this study, just two items of weight data might be sufficient to construct the prediction model.

Through the investigation on the variable importance, we found that weight changes contribute to the better prediction performance. We investigated whether and how prediction performance changes in accordance with increasing τ. Figure 5B shows the relationship between the RMSEs of the prediction models and τ for the MNIST and the CIFAR-10 datasets. We observed the tendency that increasing τ improves the prediction performance to a certain degree of τ.

3.6 Correlation with eventual model performance

We found that the proposed weight features provided better prediction performance and that the most important features were related to weight changes in last convolutional layers. To investigate whether these results depend on the random forest algorithm, we additionally investigated the correlation of each variable with the eventual model performance. For the correlation measure, we utilized Spearman's rank correlation coefficient, which is more suitable than the Pearson lin-

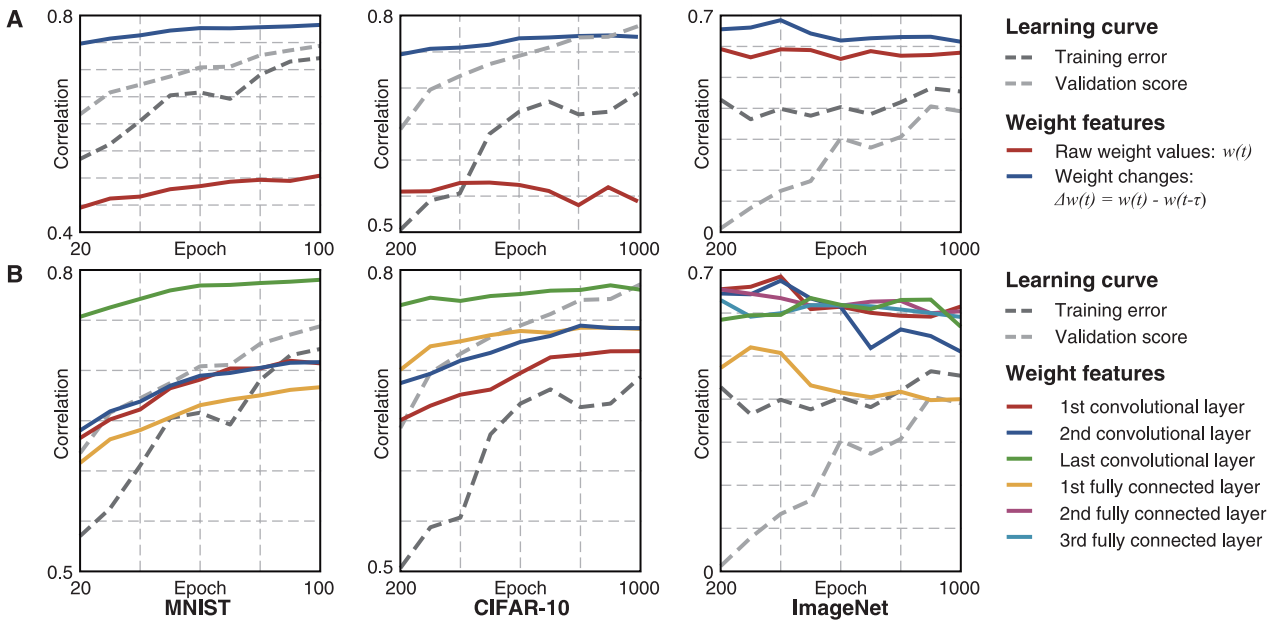

Figure 6: Correlations of each weight feature group with eventual model performance compared with those of learning curve. (A) Comparison between features of raw weight values and weight changes. (B) Comparison among weight features of each layer of DNNs. Each plot for weight features shows time series of correlation of one feature that exhibits maximal value among feature groups as shown by legends.

ear correlation coefficient to investigate variable relation that is not necessarily linear. We used the maximal correlation coefficient among each feature group and compared them.

In all three datasets, the weight features showed higher correlation coefficients than learning curve from the early learning process of DNNs (Figure 6). Comparing them between features of raw weight values and weight changes, we found that the features extracted from weight changes are more highly correlated with the eventual model performance of DNNs in all tested epochs (Figure 6A). In addition, we compared correlation coefficients between the features of each layer (Figure 6B). For the MNIST and CIFAR-10 datasets, the features extracted from last convolutional layers always showed the highest values in all tested epochs. For the ImageNet dataset, features in the last convolutional layers were also the greatest numbers of the highest correlation values over the epoch (4 out of 9), although the difference of correlation values with the other layers was comparatively small.

4 Conclusion

Deep neural networks currently deliver a strong performance in many areas such as image recognition and natural language processing. However, they require careful tuning of hyperparameters, which can dramatically affect the model performance. To speed up the hyperparameter searches by predicting the eventual performance at an early stage of the learning process and terminating underperformance runs, we proposed the use of network weight features for the prediction. We changed various types of hyperparameter in DNNs and gathered sets of the eventual model performance and features calculated from the network weights at early stages of the learning processes. Using the weight features as explanation variables, we constructed the prediction model using the random forest algorithm for predicting the eventual model performance of DNNs.

We first showed that the use of the proposed weight features provided better prediction accuracy compared with the use of the learning curve in all three tested datasets. Furthermore, we demonstrated that the use of the weight features decreased the training samples for the prediction model to, at a maximum, one-fourth the number required for achieving the same predictive performance using the learning curve. Regarding the number of pre-running epochs of DNNs, the weight features also accomplished the specific prediction performance at approximately 8 to 9 times earlier training epoch of DNNs.

We also investigated important variables within the weight features using the permutation accuracy importance measure for the random forest algorithm and correlation of each variable with the eventual model performance. The results showed the stronger contribution of weight changes than raw weight values in all three tested datasets. We also found that among the weight features, the weight changes in the last convolutional layers seem to be most important for prediction.

Taken together, our results demonstrate that the use of weight features can help construct prediction models with a smaller number of samples and terminate underperformance runs at an earlier stage of the training process of DNNs than the conventional use of learning curve, thus facilitating the speed-up of hyperparameter searches.

Acknowledgments

This research was partially supported by CREST, JST.

References

[Bengio, 2000] Yoshua Bengio. Gradient-based optimization of hyperparameters. *Neural computation*, 12(8):1889–1900, 2000.

[Bergstra and Bengio, 2012] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13, 2012.

[Bergstra *et al.*, 2013] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of The 30th International Conference on Machine Learning*, pages 115–123, 2013.

[Bottou, 2012] Léon Bottou. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade*, pages 421–436. Springer, 2012.

[Breiman, 2001] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[Deng and Yu, 2014] L. Deng and D. Yu. Deep learning: Methods and applications. *Foundations and Trends in Signal Processing*, 7, 2014.

[Domhan *et al.*, 2015] T. Domhan, J. T. Springenberg, and F. Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, 2015.

[Glorot and Bengio, 2010] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010.

[Hara *et al.*, 2014] S. Hara, R. Raymond, T. Morimura, and H. Muta. Predicting halfway through simulation: Early scenario evaluation using intermediate features of agent-based simulations. In *Proceedings of the Winter Simulation Conference*, pages 334–343, 2014.

[Hinton *et al.*, 2012a] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdelrahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. 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.

[Hinton *et al.*, 2012b] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[Jia *et al.*, 2014] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678, 2014.

[Krizhevsky *et al.*, 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[Krizhevsky, 2009] Alex Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, 2009.

[LeCun *et al.*, 1998] 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.

[Nair and Hinton, 2010] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning*, pages 807–814, 2010.

[Russakovsky *et al.*, 2015] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, et al. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[Schmidhuber, 2015] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[Snoek *et al.*, 2012] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[Strobl *et al.*, 2007] Carolin Strobl, Anne-Laure Boulesteix, Achim Zeileis, and Torsten Hothorn. Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC Bioinformatics*, 8(1):1–21, 2007.

[Swersky *et al.*, 2014] Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Freeze-thaw bayesian optimization. *arXiv preprint arXiv:1406.3896*, 2014.