SELF-STABILIZED DEEP NEURAL NETWORK

*Pegah Ghahremani**

Johns Hopkins University, Baltimore
pghahre1@jhu.edu

Jasha Droppo

Microsoft Research, Redmond
jdroppo@microsoft.com

ABSTRACT

Deep neural network models have been successfully applied to many tasks such as image labeling and speech recognition. Mini-batch stochastic gradient descent is the most prevalent method for training these models. A critical part of successfully applying this method is choosing appropriate initial values, as well as local and global learning rate scheduling algorithms. In this paper, we present a method which is less sensitive to choice of initial values, works better than popular learning rate adjustment algorithms, and speeds convergence on model parameters. We show that using the Self-stabilized DNN method, we no longer require initial learning rate tuning and training converges quickly with a fixed global learning rate. The proposed method provides promising results over conventional DNN structure with better convergence rate.

***Index Terms*—** self-stabilizer, stochastic gradient descent, learning rate, scaling, deep neural network

1. INTRODUCTION

Deep neural networks provide huge improvement relative to state-of-the-art Gaussian Mixture model(GMM) systems in speech recognition tasks [1]. The networks are typically trained using minibatch stochastic gradient descent (SGD).

Getting good performance with SGD requires tuning the initial and final learning rates and designing an annealing learning rate schedule. Many researchers have proposed different global or local per-parameter learning adjustment techniques [2], or try to involve second order information into SGD algorithm, but the majority still use the fast, simple SGD.

Setting initial learning rate is a challenging problem, where low values can result in slow learning and a larger learning rate can cause instability and divergence in training. It is not clear in very high dimensional parameter space, whether it is the best to have a single global learning rate, that can be estimated robustly, or a set of local per-dimension learning rates, whose estimation will be less robust.

In this work, we introduce a novel DNN structure that results in faster convergence and makes the model insensitive to initial choice of learning rate. This model can be combined with other learning rate annealing methods such as AdaGrad [3] or Natural Gradients [4]. In the new model structure, a single parameter is added to each layer as a trainable self stabilizer. This stabilizer scales parameters in each layer with respect to the error back-propagated in this layer during the training. These extra parameters are jointly trained with whole network.

A related approach used in [5] is usefull in reducing activation saturation by shrinking the parameters using some shrinkage factor. In this approach, the optimal shrinkage factors for updatable transformations are computed using BFGS algorithm on validation sets at end of each epochs. This method needs an extra computation step and it doesn't back-propagate the error through these shrinkage parameters during training and they also need learning rate scheduling. In another approach, batch normalization makes normalization a part of the model architecture to reduce internal covariant shift [6], making the overall model less sensitive to parameter initialization.

This paper is organized as follows. Section 2 gives an overview of different learning rate scheduling techniques used for DNN training. Section 3 describes the new method in detail, including some theoretical analysis. Section 4 presents and analyzes the results. Section 4.2 investigates the relation of this method to activation in each layer during training time. Section 4.3 investigates effectiveness of new method and its combination with other learning rate scheduling techniques on AMI [7] and section 4.4 presents the results on the Switchboard [8] task. Section 5 presents the conclusions.

2. DNN TRAINING USING LEARNING RATE SCHEDULING TECHNIQUES

2.1. Global learning rate scheduling methods

The goal of training a DNN is to update the set of parameters $\mathbf{W}$ in order to optimize objective function $\mathcal{L}$. The gradient descent algorithms attempt to optimize the objective function by following the negative of gradient direction $\frac{\partial \mathcal{L}}{\partial w_t}$ defined as g_t. It is mathematically proved that SGD convergence strongly depends on step size in convex problems [9]. A good learning rate schedule can result in faster convergence to a better local optima.

There are several methods for learning rate scheduling. Exponential or power scheduling are the most widely used techniques, where the learning rate decays by some decreasing functions such as $\eta_t = \eta_0(1+\alpha t)^{-\beta}$ ([10], [11]) or $\eta_t = \eta_0 \times \exp^{-\beta t}$, where β_t is the learning rate at iteration t.

Some approaches monitor the gradient's behavior and automatically set the learning rate [12]. Others, such as the learning rate auto-adjustment method [13], set the learning rate based on objective function performance, where learning rate is reducing by some rate if objective function degrades on training or cross-validation set.

2.2. Local per-dimension learning rate scheduling methods

Researchers have proposed techniques that develop per-parameter learning rates. Most of these methods try to approximate second order information using 1st oder information such as gradients. Approaches based on the natural gradient [4] take the Riemannian metric of parameter space into account to compute gradients [14]. They use empirical Fisher information matrix, which is estimated using

*The work of Pegah Ghahremani was performed during an internship at Microsoft Research.

the gradient covariance matrix or its diagonal approximation and apply it as a precondition to the updates.

The Natural Newton method algorithm [15] involves uncertainty between the true and empirical gradient of the objective function. It results in a direction similar to the natural gradient direction with a slight difference. They showed that if the number of data-points goes to infinity, the effect of gradient covariance matrix in computing optimum direction vanishes, which is the result of converging empirical and true loss and reducing the uncertainty in the gradient estimation. Another approach is Adagrad [3], where the learning rate for parameter i is approximated as $\eta_i(t) = \frac{\eta_0}{\sqrt{\sum_{n=0}^{t}(\nabla_{\theta_i}^{n})^2}}$.

As shown all these methods involve some hyper-parameters, which must be tuned to give the best performance. The main contribution of our method is that it can combine with these methods in order to decrease the sensitivity to hyper-parameters and reduce the burden of hyper parameter tuning. By combining our method, there is no need of learning rate annealing schedule, particularly in large scale problems. In most of learning rate scheduling techniques, the learning rate is monotonically decaying, but in this method, the introduced scaling parameters as learning rate candidates can increase as well as decrease, which is helpful for non-stationary problems, as navigating the properties of optimization landscape changing continuously [16].

3. SELF-STABILIZED DNN TRAINING

In this paper, we introduce an extra scalar parameter to each layer of the DNN model. It is designed to stabilize the stochastic gradient descent training, and is jointly trained with the original network parameters. We intrepret these new parameters as a per-layer stabilizer, where the data determines when to decrease or increase their value to progress on minimizing the objective function $\mathcal{L}$. In its simplest form, we augment the parameters of a DNN layer with a scalar β:

$$\mathbf{y} = \phi(\beta \times \mathbf{W}\mathbf{x} + \mathbf{b}), \tag{1}$$

where $\mathbf{x}$ is the input vector, $\mathbf{W}$ and $\mathbf{b}$ define an affine transformation, and $\mathbf{y}$ is the output vector. We initialize $\beta = 1$, so the initial model is identical with or without the new parameter.

During training, the stabilizer parameters β are simply another parameter to learn with stochastic gradient descent. The update rule for β is simple to derive. During the backward pass, the gradient with respect to input vector x in layer i is computed as

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \beta \mathbf{W}^T \frac{\partial \mathcal{L}}{\partial \mathbf{y}} \tag{2}$$

and the gradient with respect to parameter β is computed as

$$\frac{\partial \mathcal{L}}{\partial \beta} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}} \times \frac{\partial \mathbf{y}}{\partial \beta} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}}^T \mathbf{W}\mathbf{x} \tag{3}$$

So the gradient for parameter β can be written as

$$\frac{\partial \mathcal{L}}{\partial \beta} = \frac{1}{\beta}(\frac{\partial \mathcal{L}}{\partial \mathbf{x}})^T \mathbf{x} = \frac{\mathbf{1}}{\beta} < \frac{\partial \mathcal{L}}{\partial \mathbf{x}}, \mathbf{x} > \tag{4}$$

so β is

$$\beta_{t+1} = \beta_t - \frac{\eta}{\beta} < \frac{\partial \mathcal{L}}{\partial \mathbf{x}}, \mathbf{x} > \tag{5}$$

The change in β is directly related to how the layer input $\mathbf{x}$ relates to the gradient of the objective function with respect to that input. If the objective function would be improved by scaling $\mathbf{x}$ up, β will increase. If the objective function would be improved with a smaller $\mathbf{x}$, then β will decrease. If the relative direction of the input and its gradient is random, then β will stabilize. We expect this to happen near convergence.

The step size for parameter w_{ij} is directly controlled by the current value of β.

$$g_{ij} = \frac{\partial \mathcal{L}}{\partial w_{ij}} = \beta \frac{\partial \mathcal{L}}{\partial y_i} x_j \tag{6}$$

In practice, we use $\exp(\beta)$ instead of β in our experiments, and we initialize $\beta = 0$. This constrains the effective stabilizer to be positive, and to decay more slowly as it approaches zero.

$$\mathbf{y} = \phi(\exp(\beta) \times \mathbf{W}\mathbf{x} + \mathbf{b}), \tag{7}$$

Our experiments show that this substitution produces similar models without the danger of the stabilizer parameter changing sign, which can be disruptive to the DNN training.

4. EXPERIMENTAL RESULTS

4.1. Evolution of self-stabilizer parameter during training

Figure 1 shows stabilizer changes per layer during training for self-stabilized parameters with same DNN architecture using different learning rate scheduling techniques. Layer i corresponds to parameter in i^{th} hidden layer, and the model consists of six layers. These graphs were generated using our AMI setup, which is described in Section 4.3.

The figure demonstrates the interaction between the learning rate, the self stabilizer values, and the auto adjustment algorithm. In the figure, learning rates are specified with two numbers. The value 0.1:0.6 indicates the trainer applied a learning rate of 0.1 per mini-batch with a mini-batch size of 256 for the first epoch, and then switched to a learning rate of 0.6 per mini-batch with a mini-batch size of 1024. The auto-adjustment algorithm we used multiplied the learning rate by 0.618 at the end of every epoch that didn't improve the model's performance on our development set.

As can be seen in Figure 1a and 1b, the self-stabilizer parameters are larger when the initial learning rate is smaller. This inverse scaling with learning rate is an indication that the self-stabilizer parameters are compensating for any sub-optimal choice of learning rate.

Figures 1a and 1c illustrate the same system with and without the auto-adjustment algorithm active. When the auto-adjustment algorithm is active, the global learning rate decreases over epoch, and the self-stabilizer parameters approach constant values. Without the auto-adjustment algorithm, the self-stabilizer parameters clearly decay over epochs. The new parameters adjust the effective global learning rate down as the model converges, compensating for the non-ideal fixed learning rate schedule.

4.2. Effects of self-stabilizer on Activation

We found that self-stabilized DNN models converge much faster than our baseline DNNs, and the effect is more pronounced when using the Sigmoid nonlinearity. Suspecting that the technique allows all layers to train at the same rate, we did some experiments on the LVCSR Switchboard task to investigate evolution of activations during training. The Switchboard setup is described in more detail in Section 4.4.

The theoretical reason for slow convergence when using a sigmoid nonlinearity is its non-zero mean, which results in important singular value in the Hessian [16].

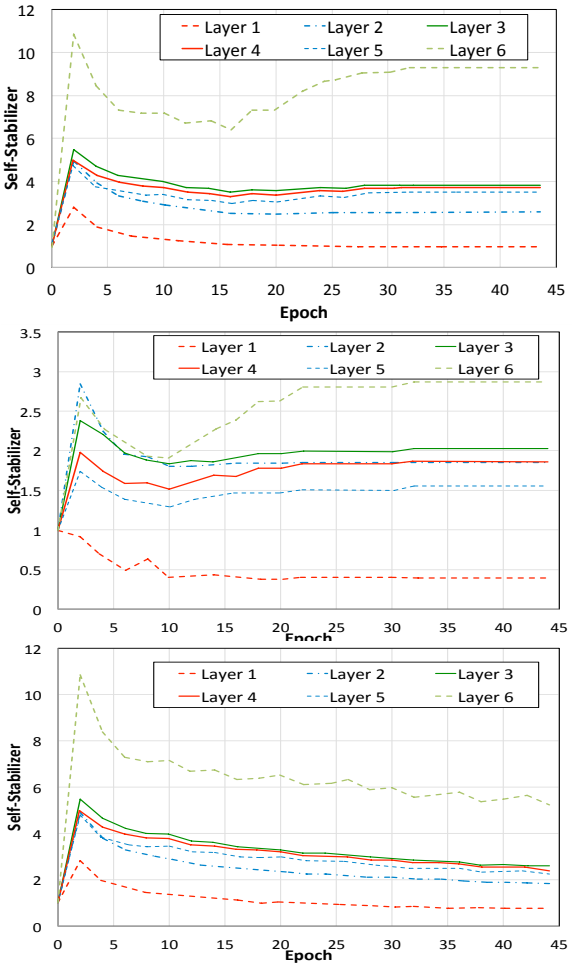

Fig. 1: (a) Initial learning rate 0.1:0.6 with auto-adjustment (b) Initial learning rate 0.8:3.2 with auto-adjustment (c) Initial learning rate 0.1:0.6 with No learning rate adjustment.

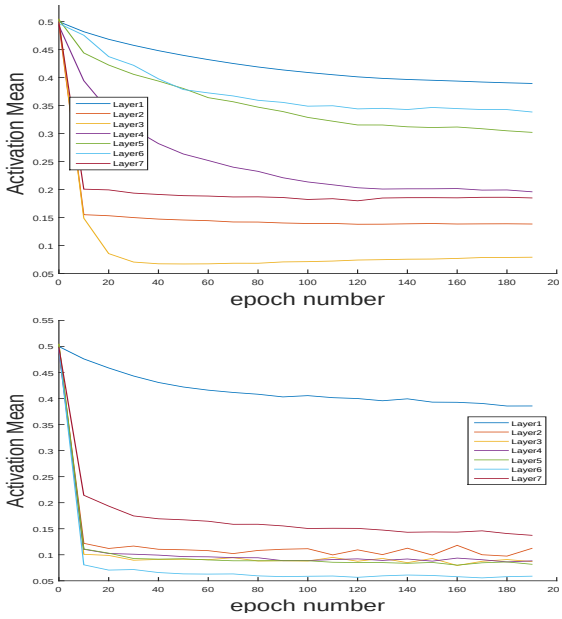

Fig. 2: standard deviation of activation values with sigmoid activation during training for different hidden layers with standard DNN (top) vs. Self-stabilized DNN.

Figure 2 shows the mean of the activation values after nonlinearity in each layer with and without self-stabilizer during training. Layer 1 refers to the output of first hidden layer, and the graph shows the mean of activations averaged averaged over 44000 fixed examples and these values are computed at different time during training.

Figure 2a shows the evolution of activation mean for different layers in conventional DNN. As shown, the activation values of some layers pushed to average values near 0.5 and this type of saturation lasts very long during training. The same behavior has been shown in [17] for sigmoid nonlinearity. Figure 2b shows different behavior of activations in in self-stabilized network and the activations at different layers have closer mean and similar distribution in the new structure that results in better flow of back-propagated gradients through network and faster training. The reason for this behavior in the new structure is that the self-stabilizers scale down the parameters in each layer that can be useful to overcome activation saturation and improves the flow of gradient through all layers. The self-stabilizer helps the network to scale the parameters in the beginning of training to be in a region of parameter space that is closer to the local optima of cost function.

If the activation variations on a layer is going to be small, we can substitute x_i by $E[x]$ in equation 5, so the update for parameter β is $E[x] \sum \frac{\partial \mathcal{L}}{\partial \mathbf{x}}$. The second term in the update approximates the sum of back-propagated error over activations. If the sum of gradients or activation mean at each layer are getting larger, β-update is going to be large and β changes to scale up or down the parameters and when these values are small, the β is going to be almost constant. Small gradient-sum over activations in each layers shows that the parameters in this layer are close to the local optima and they no longer need to be scaled. Also if the mean of activation are above 0.5 in sigmoid, this means the activations are saturated and β-update is going to be large and the parameter β changes to scale down the parameter to overcome saturation.

4.3. Results on AMI

In Table 1, we report the results on AMI-IHM database using different optimization techniques. The DNN configuration in all experiments contain 1080 dim input layer, with 15 spliced frames (7 frames on each side of current frames) of 72 dimensional features. These features contain 24 dim MFCC features and their velocity and acceleration. The network uses sigmoind nonlinearity in its 6 hidden layers, each of which is 1024 neurons wide. The network parameters are initialized randomly and each epoch contains about 24 out of 84 hours of training data. We trained each system for 50 epochs.

4.4. Results on Switchboard

The CNTK auto-adjust learning rate algorithm "adjust after epoch" is used in some of experiments. It is a global learning rate scheduling technique, where the global learning rate is reduced by some factor, if the cross entropy degrades on a validation set. Momentum is also another technique that can improve training by considering the curvature information specially in later steps [18]. In all experiments without AdaGrad in Table 1, the momentum with factor 0.9 is applied to gradients during training.

Optimization Method	Auto-Adjust	Initial Learning Rate	Training CE	Training % Frame Err	Validation CE	Validation % Frame Err	%WER
SGD	✓	**0.1:0.6**	2.3	53.1	2.5	57.3	**40.2**
SGD	✓	**0.8:3.2**	1.7	43.5	2.2	51.0	**32.4**
SGD+Self-Stabilizer	✓	0.1:0.6	1.5	38.7	2.13	49.8	32.5
SGD+Self-Stabilizer	✓	0.8:3.2	1.6	39.8	2.1	49.7	32.0
SGD+Adagrad	x	0.8:3.2	1.8	44.6	2.3	52.0	33.5
SGD+AdaGrad+Self-Stabilizer	x	0.8:3.2	1.49	39.2	2.2	51.3	32.2
SGD+ Self-Stabilizer	x	0.1:0.6	1.3	36.3	2.3	51.0	32.1
SGD+ Self-Stabilizer+l2reg	x	**0.1:0.6**	1.4	37.3	2.2	50.7	**31.8**

Table 1: Results on AMI using different optimization methods.

The first 2 experiments in Table 1 show that SGD method is sensitive to initial learning rate and the cross entropy degrades with small learning rate. It may need to more training time to compensate small learning rate, or it could be stuck in a local minimum.

In the next 2 experiments, the self-stabilizer parameters are added to conventional DNN model and the result shows that self-stabilized DNN is less sensitive to the learning rate initialization and final cross entropy is very similar using different initial learning rate.

In the experiments 5 and 6, the learning rate is fixed during training and the effect of combining self-stabilizer with AdaGrad is investigated. AdaGrad [3] uses $1st$ order information to estimate some properties of second order methods, where the gradient per dimension in each step is normalized with l_2 norm of all previous gradients. These experiments used a modified version of AdaGrad [13]. This implementation has two modifications with respect to the canonical version of the algorithm. First, the squared gradient is not accumulated over all time. A leaky integrator is used to turn the accumulation into a running average with an exponential weighting function. This eliminates the tendency for AdaGrad to prematurely limit the learning rate. Second, the AdaGrad contribution is scaled so that the average AdaGrad factor over all parameters is 1.0. This allows for a more direct comparison of learning rates over experiments with and without AdaGrad. As shown, the self-stabilizer helps to improve the results using AdaGrad with fixed learning rate during training.

The last two experiments use self-stabilized DNN model with no use of global or local learning rate scheduling techniques and the initial learning rate is small remains fixed during training. The self-stabilized network gives good improvement in terms of training cross entropy and some degradation can be seen on validation set compared to $3rd$ experiments. The reason could be related to overtraining. In the last experiment, l_2 regularization technique is used during training to overcome overtraining. This regularization is applied to all parameters, including the self-stabilizer parameters. As can be seen, this model with fixed learning rate and no learning rate scheduling gives the best WER results.

Figure 3 shows results from training on 2000 hrs of Switchboard and Fisher conversational telephone speech. The DNN configuration contains six 2048 neuron wide hidden layers with sigmoid nonlinearity. As with the AMI experiments, simple SGD is used in training and the parameters are randomly initialized. The input layer contains 920 units, which is $+/-11$ frames of 40 dimensional MFCCs spliced together. The output layer size is 9000. The learning rate per mini-batch for stabilized DNN is 0.1 for the 1^{st} epoch and 1 for next epochs. The baseline DNN system uses learning rate 0.1 for 1^{st} epoch and 1 for next 100 epochs and then uses an auto-adjustment algorithm for the next 100 epochs in second half of training. The auto-adjustment method does not need development data, but instead chooses the learning rate based on the projected cross entropy over the next 500 mini batches [13].

Results comparing baseline and self-stabilized DNN are presented in Figure 3. During training, the self-stabilized DNN converged more quickly to a better optimum than the Baseline DNN, which used the auto-adjustment technique. The figure shows the evolution of word error rate during training. The self-stabilized DNN is has a consistent 1.5% absolute WER improvement over the baseline.

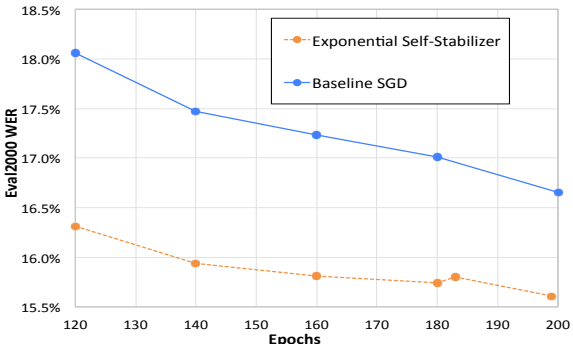

Fig. 3: (a) Training Cross Entropy (b) WER results during training on Switchboard.

5. CONCLUSION

We proposed a new type of network architecture to speed up DNN training. This network is less sensitive to parameter initialization and choice of learning rates. The effectiveness of this method comes from training a self-stabilizer parameter for each layer during training, which can scale parameters in each layer w.r.t the gradient back-propagated to this layer to stabilize activation distribution throughout training. Results using Self-stablized DNN method shows less sensitivity to global learning rate and faster convergence to better local optimum. The future work is to add a diagonal self-stablizer per layer, which enables the model to scale each dimension of activations separately.

6. REFERENCES

[1] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman 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," *Signal Processing Magazine, IEEE*, vol. 29, no. 6, pp. 82–97, 2012.

[2] Abraham P George and Warren B Powell, "Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming," *Machine learning*, vol. 65, no. 1, pp. 167–198, 2006.

[3] John Duchi, Elad Hazan, and Yoram Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *The Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

[4] Shun-Ichi Amari, Hyeyoung Park, and Kenji Fukumizu, "Adaptive method of realizing natural gradient learning for multilayer perceptrons," *Neural Computation*, vol. 12, no. 6, pp. 1399–1409, 2000.

[5] Xiaohui Zhang, Jan Trmal, Daniel Povey, and Sanjeev Khudanpur, "Improving deep neural network acoustic models using generalized maxout networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 215–219.

[6] Sergey Ioffe and Christian Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[7] Thomas Hain, Luká Burget, John Dines, Philip N Garner, František Grézl, Asmaa El Hannani, Marijn Huijbregts, Martin Karafiat, Mike Lincoln, and Vincent Wan, "Transcribing meetings with the amida systems," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 20, no. 2, pp. 486–498, 2012.

[8] John J Godfrey, Edward C Holliman, and Jane McDaniel, "Switchboard: Telephone speech corpus for research and development," in *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on*. IEEE, 1992, vol. 1, pp. 517–520.

[9] Léon Bottou, "Online learning and stochastic approximations," *On-line learning in neural networks*, vol. 17, no. 9, pp. 25, 1998.

[10] Léon Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, pp. 177–186. Springer, 2010.

[11] Herbert Robbins and Sutton Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[12] Tom Schaul, Sixin Zhang, and Yann LeCun, "No more pesky learning rates," *arXiv preprint arXiv:1206.1106*, 2012.

[13] Dong Yu, Adam Eversole, Mike Seltzer, Kaisheng Yao, Zhiheng Huang, Brian Guenter, Oleksii Kuchaiev, Yu Zhang, Frank Seide, Huaming Wang, et al., "An introduction to computational networks and the computational network toolkit," Tech. Rep., Tech. Rep. MSR, Microsoft Research, 2014, http://codebox/cntk, 2014.

[14] Amari Shun-ichi, *Differential-geometrical methods in statistics*, vol. 28, Springer Science & Business Media, 2012.

[15] Nicolas L Roux and Andrew W Fitzgibbon, "A fast natural newton method," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, pp. 623–630.

[16] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*, pp. 9–48. Springer, 2012.

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

[18] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th international conference on machine learning (ICML-13)*, 2013, pp. 1139–1147.