INTERSPEECH 2017
August 20–24, 2017, Stockholm, Sweden

Backstitch: Counteracting Finite-sample Bias via Negative Steps

Yiming Wang[1], *Vijayaditya Peddinti*[1,2], *Hainan Xu*[1],
Xiaohui Zhang[1], *Daniel Povey*[1,2], *Sanjeev Khudanpur*[1,2]

[1]Center for Language and Speech Processing
[2]Human Language Technology Center of Excellence
The Johns Hopkins University, Baltimore, MD 21218, USA
{yiming.wang,vijay.p,hxu31,xiaohui,dpovey1,khudanpur}@jhu.edu

Abstract

In this paper we describe a modification to Stochastic Gradient Descent (SGD) that improves generalization to unseen data. It consists of doing two steps for each minibatch: a backward step with a small negative learning rate, followed by a forward step with a larger learning rate. The idea was initially inspired by ideas from adversarial training, but we show that it can be viewed as a crude way of canceling out certain systematic biases that come from training on finite data sets. The method gives $\sim 10\%$ relative improvement over our best acoustic models based on lattice-free MMI, across multiple datasets with 100-300 hours of data.

Index Terms: deep learning, stochastic gradient descent, speech recognition.

1. Introduction

Recently, the concept of training on adversarial examples has been proposed [1, 2, 3, 4]. We had previously attempted to use adversarial training for speech-recognition tasks, but failed to obtain any improvement. It occurred to us that a "model-space" version of adversarial training might work better, and this became the *backstitch* method, which is as follows. When processing a minibatch, instead of taking a single SGD step, we first take a step with $-\alpha$ times the current learning rate, for a small α (e.g. 0.3), and then a step with $1 + \alpha$ times the learning rate, with the same minibatch (and a recomputed gradient). So we are taking a small negative step, then a larger positive step. This resulted in unexpectedly large improvements – around 10% relative improvements for our best speech recognition models based on lattice-free MMI (LF-MMI) [5], and the improvement was consistent across datasets.

In this paper we will develop the outlines of a theory why backstitch training might be working, based on the notion that it counteracts finite-sample bias. We will show results on a number of LVCSR tasks and find consistent improvements compared with conventional SGD.

In Section 2 we will discuss finite-sample bias and how it affects optimization tasks where we are training on samples from an underlying distribution. Section 3 analyzes backstitch training from this perspective and discusses the conditions under which it can be considered to be counteracting finite-sample bias. Section 4 discusses the interaction of backstitch training with various other aspects of our training framework. Section 5 shows our experimental results on multiple datasets, and we conclude in Section 6.

This work was partially supported by NSF CRI Award no. 1513128, DARPA LORELEI Contract No HR0011-15-2-0024, and IARPA BABEL Contract No 2012-12050800010. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, IARPA, DoD/ARL or the U.S. Government.

2. Finite-sample bias

By finite-sample bias what we specifically mean is that any time we train using gradients from samples that we have seen before, for most model types this will cause the gradient estimate to be systematically biased. We first illustrate this via an example and will then show what we mean more generally. Our treatment of finite-sample bias in this section is quite general and abstract; the reason we include this material is that we later use it to motivate the backstitch algorithm.

2.1. Example

Probably the simplest example of finite-sample bias is the case where we are estimating the mean and variance of a Gaussian distribution to maximize the likelihood of some data. Maximizing the likelihood of the data from finite samples leads to variances which are systematically smaller than the real variance. This could be formulated as maximizing a function

$$f(x;\boldsymbol{\theta}) = \log N(x;\mu,\sigma^2) \tag{1}$$

where $\boldsymbol{\theta} = (\mu, \sigma^2)$. Although there are easier ways to estimate a mean and variance, it's quite possible to do so via SGD; and using a likelihood-based objective function, this will converge to the biased maximum likelihood estimate.

2.2. Finite-sample bias (more general case)

Suppose we are minimizing $E_{x\sim P}[f(x;\boldsymbol{\theta})]$, where P is the underlying distribution of the data x, and that there is a global optimum parameter $\boldsymbol{\theta}^*$ (optimum as far as the underlying distribution of x is concerned). In a machine-learning context, each x would typically represent a (feature, label) pair.

We assume that the objective function is approximated well enough by a quadratic function of $\boldsymbol{\theta}$ that given a sampled set of training-data, we can get close to the data-specific optimum by a single iteration of Newton's method starting from $\boldsymbol{\theta}^*$. In real life, if we had access to $\boldsymbol{\theta}^*$ we would not bother training further, but this is for the sake of argument.

We also assume that the expected value (over the data distribution) of the Hessian w.r.t $\boldsymbol{\theta}$, evaluated at $\boldsymbol{\theta}^*$, equals the identity matrix $\mathbf{I}$; this will simplify some of the following expressions. This can be achieved via a change of variables, as long as the original Hessian is full rank.

We are training on samples x_i for $i = 1, \ldots, I$. To establish some compact notation for the gradients and Hessians for the training samples x_i, let:

$$\mathbf{g}(x) \triangleq \left.\frac{\partial f(x;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*} \quad (2)$$

$$\mathbf{H}(x) \triangleq \left.\frac{\partial^2 f(x;\boldsymbol{\theta})}{\partial^2 \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*} \quad (3)$$

and as a shorthand:

$$\mathbf{g}_i \triangleq \mathbf{g}(x_i) \quad (4)$$

$$\mathbf{H}_i \triangleq \mathbf{H}(x_i) \quad (5)$$

and let the averages of these be:

$$\hat{\mathbf{g}} \triangleq \frac{1}{I}\sum_i \mathbf{g}_i \quad (6)$$

$$\hat{\mathbf{H}} \triangleq \frac{1}{I}\sum_i \mathbf{H}_i \quad (7)$$

The estimated parameter vector given a single iteration of Newton's method starting from $\boldsymbol{\theta}^*$ is:

$$\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}^* - \hat{\mathbf{H}}^{-1}\hat{\mathbf{g}} \quad (8)$$

Since we previously specified that the "true" expected Hessian $\mathbf{H}$ is identity, we can write

$$\hat{\mathbf{H}}^{-1} \simeq 2\mathbf{I} - \hat{\mathbf{H}} \quad (9)$$

which comes from retaining the first-order term in the series $(\mathbf{I}+\mathbf{D})^{-1} = \mathbf{I} - \mathbf{D} + \mathbf{D}^2 \ldots$ (valid for $\mathbf{D}$ with singular values less than one), where $\mathbf{D} \triangleq \hat{\mathbf{H}} - \mathbf{I}$. This gives us:

$$\begin{aligned}\hat{\boldsymbol{\theta}} &\simeq \boldsymbol{\theta}^* - 2\hat{\mathbf{g}} + \hat{\mathbf{H}}\hat{\mathbf{g}} \\ &= \boldsymbol{\theta}^* - 2\hat{\mathbf{g}} + \frac{1}{I^2}\sum_{i,j}\mathbf{H}_i\mathbf{g}_j \quad (10)\end{aligned}$$

We are interested in the expected bias in the estimate of $\boldsymbol{\theta}$, i.e. in $E[\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^*]$, where the expectations are taken over the set of I training samples generated from their underlying distribution. In (10), the term $-2\hat{\mathbf{g}}$ does not lead to any expected bias, because the expected value of $\mathbf{g}$ is zero. Nor do the terms in summation on the right for $i \neq j$, because (due to independence) the expectation can be decomposed into the product of two terms, one of which (involving $\mathbf{g}$) is zero. The only potential bias comes from the "self-terms" (for $i = j$), i.e. from the quantity $\frac{1}{I^2}\sum_i \mathbf{H}_i\mathbf{g}_i$, so we can write:

$$E[\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^*] = \frac{1}{I}E_x[\mathbf{H}(x)\mathbf{g}(x)] \quad (11)$$

Since we previously ensured that the expected Hessian is identity, if we add an extra term $\mathbf{c}$ to the derivative of $f(x;\boldsymbol{\theta})$ during training it would approximately become an offset $-\mathbf{c}$ in the estimated parameter, so in order to cancel out the bias of Equation (11) it is reasonable to add a correction term:

$$\mathbf{c} = \frac{1}{I}E_x[\mathbf{H}(x)\mathbf{g}(x)] \quad (12)$$

to the gradients we use to train the network. The quantities $\mathbf{H}(x)$ and $\mathbf{g}(x)$ are of course defined w.r.t. the optimal parameter $\boldsymbol{\theta}^*$ which is unknown, and in practice any expressions will use the current $\boldsymbol{\theta}$.

More generally, if the expected Hessian at $\boldsymbol{\theta}^*$ is not identity but is some positive definite matrix $\overline{\mathbf{H}}$, then the expression above becomes:

$$\mathbf{c} = \frac{1}{I}E_x[\mathbf{H}(x)\overline{\mathbf{H}}^{-1}\mathbf{g}(x)] \quad (13)$$

See [6] for the derivation.

Of all the assumptions and approximations we made here, the assumption that the objective function is well approximated by a quadratic seems to be the most problematic in practice. In toy experiments with quadratic objective functions, our expression for the bias seems to be consistent with experimental results; but, when we applied this analysis to the bias when estimating the variance of a Gaussian with unknown mean, we got a result that was wrong by a constant factor.

3. Backstitch training and relation to finite-sample bias

We will now introduce some notation for backstitch training and show how, under certain conditions, it acts to counter the finite-sample bias of Equation (11).

Suppose a single iteration of conventional SGD is written

$$\boldsymbol{\theta}_{n+1} \leftarrow \boldsymbol{\theta}_n - \nu\mathbf{g}(x_n, \boldsymbol{\theta}_n) \quad (14)$$

where x_n is whichever sample we choose for the n'th SGD iteration and $\mathbf{g}(x, \boldsymbol{\theta}_n)$ is the derivative of $f(x,\boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$, evaluated at $\boldsymbol{\theta} = \boldsymbol{\theta}_n$. Although in practice we use minibatches, we will not develop special notation for that, since it does not affect the math— in principle we could let the samples x be whole minibatches. In backstitch training, we do two steps:

$$\boldsymbol{\theta}'_{n+1} \leftarrow \boldsymbol{\theta}_n + \alpha\nu\,\mathbf{g}(x_n, \boldsymbol{\theta}_n) \quad (15)$$

$$\boldsymbol{\theta}_{n+1} \leftarrow \boldsymbol{\theta}'_{n+1} - (1+\alpha)\nu\,\mathbf{g}(x_n, \boldsymbol{\theta}'_{n+1}) \quad (16)$$

where the constant $\alpha > 0$ determines how strongly we are applying the backstitch training. View this as a small backwards step followed by a larger forwards step. For purposes of analysis we can telescope these two iterations into one by making a quadratic approximation around $\boldsymbol{\theta} = \boldsymbol{\theta}_n$:

$$\boldsymbol{\theta}_{n+1} \leftarrow \boldsymbol{\theta}_n - \nu\mathbf{g}(x_n, \boldsymbol{\theta}_n) - \nu^2(\alpha + \alpha^2)\mathbf{H}(x_n, \boldsymbol{\theta}_n)\mathbf{g}(x_n, \boldsymbol{\theta}_n) \quad (17)$$

If we were to view this is a correction term to $\mathbf{g}$, it would look like:

$$\boldsymbol{\theta}_{n+1} \leftarrow \boldsymbol{\theta}_n - \nu(\mathbf{g}(x_n, \boldsymbol{\theta}_n) + \mathbf{c}_n) \quad (18)$$

where

$$\mathbf{c}_n = \nu(\alpha + \alpha^2)\mathbf{H}(x_n, \boldsymbol{\theta}_n)\mathbf{g}(x_n, \boldsymbol{\theta}_n). \quad (19)$$

Comparing this with Equation (12), this would make sense as an exact bias-correction method if $\nu(\alpha + \alpha^2) = 1/I$, where I is the number of samples, and if we were in a space were the expected Hessian were identity. Of course these are quite strong assumptions, and we have not developed a theory of how this method is expected to perform when the assumptions are not quite met.

Our theory predicts that backstitch should counteract the training bias in exactly the same way when the forward and backward steps are reversed. Experiments (not shown here) show that in practice the improvement from backstitch is reduced when the order of the two steps is reversed. This shows that the theory presented here is at best only a partial explanation of why it works.

4. Other aspects of backstitch training

4.1. Efficiency and backstitch interval

Naively implemented, backstitch training would take twice as long per epoch as regular training because we need to train twice on each minibatch. This is inconvenient. To speed it up, we have experimented with versions where we do the backstitch training every m minibatches, and do normal SGD updates in between. We call this m value the "backstitch interval"; $m = 1$ means doing it every update, $m = 4$ means doing it every 4th update. We have found that the performance improvement we obtained with $\alpha = 0.3$ and $m = 1$ could be more efficiently obtained with $\alpha = 1.0$ and $m = 4$.

4.2. Interaction with natural gradient

The assumption that the expected Hessian is identity is actually not too unreasonable because we are using a Natural Gradient (NG) method for training [7, 8, 9], as described in [10]. Our method uses a similar factorization to K-FAC [9]. Natural Gradient can be equivalently viewed as a change of variables into a space where a certain factored estimate of the Fisher matrix is identity; and if the objective function can be interpreted as a log probability or likelihood of some kind, under certain regularity conditions the Fisher should be the negative of the Hessian [11]. So from a theoretical perspective there is reason to expect that backstitch training should work particularly well with NG. Our preliminary experiments show backstitch training is also effective without NG, but its improvement is less than with NG, as expected.

We should mention regarding the interaction with NG more generally: we have used terminology such as "a parameter space where the Hessian is identity", as if we were actually performing a change of variables, and we have mentioned how NG takes us closer to such a situation. NG training can indeed be *formulated* as a change of variables, but actually we implement it as a modification to the derivatives that is more like a matrix multiplication (by the inverse of a factored Fisher matrix). This equivalence with a change of variables holds for backstitch training too, so the interaction is straightforward; we just mention this, as it could be a source of confusion. Also, like all practical implementations of NG, our NG method makes various approximations and factorizations, so the argument above about the Hessian being identity is not exact— at best, the Hessian would be *closer* to the identity.

4.3. Interaction with natural gradient updates

We are using the "online NG" method described in [10]. This involves updating an estimate of the Fisher matrix on each minibatch. It is important for the convergence of the overall SGD, that the estimate of the Fisher matrix should be obtained from *previous* minibatches. Otherwise it could cause a bias due to effects similar to the finite-sample bias we are discussing in this paper. For backstitch training we modified our NG implementation to ensure that the Fisher-matrix estimates used in the NG code are not contaminated with the current minibatch. With reference to the two-step training procedure of Equations (15) and (16), we freeze the Fisher-matrix estimates during the first step and allow them to be updated only during the second. We experimented with doing it the "wrong way" (updating the Fisher matrix on the first step, allowing the second step to use contaminated Fisher-matrix estimates), and as expected this degraded the results.

4.4. Backstitch training and parameter maximum changes

One aspect of our training procedure is that to prevent instability, we enforce a maximum parameter-change. This is done at two levels: per component (which very roughly means: for each layer of the neural network), and globally. There is no maximum change per individual parameter. Our normal defaults are a max-change of 0.75 (in Euclidean distance in parameter space) per component per minibatch, and a max-change of 2.0 globally per minibatch, enforced first per component and then globally.

When implementing backstitch training we tried to ensure that the same "effective" learning-rate matrix (after imposing the max-change) was used in both the first and second steps. The way we did this was by scaling the max-change constraints before applying them, by α in the first pass and $1 + \alpha$ in the second pass.

4.5. Backstitch training and momentum

The interaction of backstitch training with momentum is non-trivial and we have not implemented the combination. Most of our systems do not use momentum anyway, as NG is effective in preventing instability and we usually find that momentum hurts performance. None of our LF-MMI systems [5] use momentum; the only systems where we use momentum as a matter of course are our cross-entropy (CE) trained LSTM-based systems where we use the moderate momentum value of 0.5. We are currently working on a momentum-friendly version, so that it can be used in toolkits that rely on momentum for good performance.

4.6. Slow start backstitch training

We noticed that when backstitch is introduced partway through training, there is a sharp degradation in both training-set and validation-set objective function, which is then quite rapidly reversed. If we start training with backstitch enabled, this degradation is visible at the start of training. (These objective function values are measured separately from the normal training process, and are independent from the expected objective-function degradation when we process each minibatch the second time). We do not understand why this happens, but in order to prevent any possible bad effects, we always linearly increase α from zero to the specified value over the first 15 iterations [1].

5. Experiments

We conducted experiments with the Kaldi speech recognition toolkit [12], with techniques including speed perturbation [13], i-vector adaptation [14] and pronunciation and silence probability modeling [15]. We did extensive experiments on three different LVCSR tasks using backstitch SGD training with different setups, including different criteria (CE and LF-MMI [5]), different network architectures (Time-Delay Neural Network [16, 17] (TDNN) with ReLU nonliearities [18, 19], Bidirectional LSTM [20, 21] (BLSTM) consisting of stacked LSTMP layers [22], and a mixture of TDNN and unidirectional LSTM [23] (TDNN-LSTM) which we recently found not only outperforms BLSTM, but is also computationally more efficient). The backstitch scale α, backstitch interval n (see Sec. 4.1) and number of epochs were tuned for each individual backstitch training experiment, therefore these hyper-parameters vary across different setups. The baseline systems' number of epochs had been tuned previously. A common configuration is, $\alpha = 1.0, m = 4$, meaning we use $\alpha = 1.0$ once every 4 minibatches. Some older experiments, before we introduced the faster version, use $\alpha = 0.3, m = 1$. The num-

[1]An iteration is how long it takes for each job to process a fixed amount of data in our framework, tuned to take about 2 to 5 minutes' worth of GPU time. For a typical acoustic model this is $\sim$ 1600 parameter updates.

ber of epochs with backstitch training is 1.5 $\sim$ 2 times the number of epochs with normal training, because we found that WER continues to improve for more epochs with this method. All the other hyper-parameters are kept unchanged from those in the normal training experiment[2]. The results of LF-MMI and CE systems are shown in Table 1 and Table 2 respectively ($\alpha = 0.0$ means the baseline SGD training). While improvements are consistently observed across all the setups, they are most prominent with LF-MMI+TDNN-LSTM (which happen to be our best systems) at around 10% relative WER improvement.

Table 1: *Comparison of Backstitch SGD training with normal SGD training in LF-MMI systems.*

LF-MMI					
Dataset	$\alpha[/m]$	WER (%)			
		TDNN-LSTM		BLSTM	
		dev	eval	dev	eval
AMI-SDM	0.0	37.9	41.1	39.7	42.9
	1.0/4	34.1	37.6[3]	36.7	40.4
(use IHM alignment)	Rel. Gain (%)	**10.0**	**8.5**	**7.6**	**5.8**
		fsh_fg	tg	fsh_fg	tg
Switchboard	0.0	14.1	15.5	14.3	15.8
	0.3/1	12.7	14.0	13.4	14.7
	Rel. Gain (%)	**10.0**	**9.7**	**6.3**	**5.8**
		dev	test	dev	test
TED-LIUM	0.0	9.4	8.8	9.4	9.0
	1.0/4	8.3	7.8	8.7	8.1
[24]	Rel. Gain (%)	**11.7**	**11.4**	**7.4**	**10.0**

Table 2: *Comparison of Backstitch SGD training with normal SGD training in CE systems.*

CE					
Dataset	$\alpha[/m]$	WER (%)			
		TDNN-LSTM		BLSTM	
		dev	eval	dev	eval
AMI-SDM	0.0	36.8	40.7	37.5	41.3
	0.3/1 1.0/4	35.1	39.1	35.7	39.2
(use IHM alignment)	Rel. Gain (%)	**4.6**	**3.9**	**4.8**	**5.1**
		fsh_fg	tg	fsh_fg	tg
Switchboard	0.0	15.3	16.4	14.6	15.6
	1.0/4	14.9	16.1	14.1	15.2
	Rel. Gain (%)	**2.6**	**1.8**	**3.4**	**2.6**
		dev	test	dev	test
TED-LIUM	0.0	10.9	10.0	10.3	9.4
	1.0/4	10.8	9.7	9.8	9.0
[24]	Rel. Gain (%)	**0.9**	**3.0**	**4.9**	**4.3**

Figure 1 compares the training log-probabilities between regular SGD and the backstitch method, on our LF-MMI+TDNN-LSTM system on the AMI-SDM dataset. The main observation, which we consistently see, is that the difference between train and validation objective functions is smaller when using backstitch.

We also tested the backstitch method in language modeling tasks with training recurrent neural network language models [25] (RNNLMs) on the AMI-SDM text data. We used nnet3 implementation of neural networks of Kaldi to train RNNLMs

[2]Except that momentum is disabled when doing backstitch training in CE systems. See Section 4.5 for more explanations.

[3]To the best of our knowledge, this is the best number ever reported on AMI-SDM.

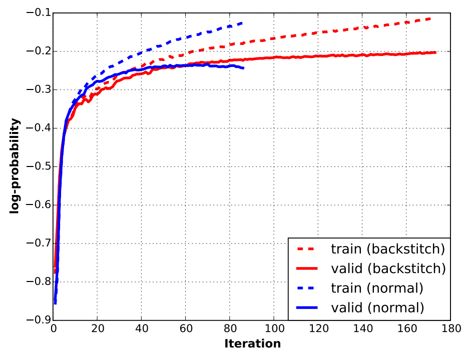

Figure 1: *Plot of log-probability vs iterations on AMI-SDM using LF-MMI+TDNN-LSTM (We continued backstitch training for twice the epochs of normal training).*

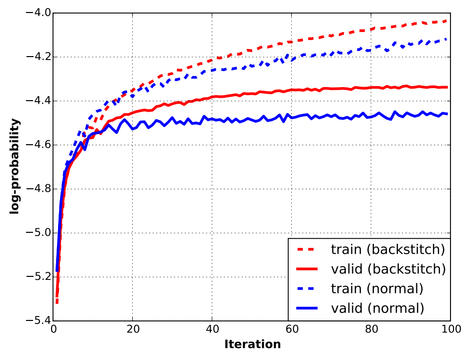

Figure 2: *Plot of log-probability vs iterations on AMI-SDM RNNLMs.*

with CE objective functions and the backstitch scale is tuned to be 0.8. Figure 2 shows the comparisons between regular SGD and the backstitch method, where we see similar trends. We are encouraged that the backstitch method's usefulness might not be limited to acoustic modeling tasks. However, we have less confidence in our RNNLM results than with our ASR results, since the setup is much newer and may not be as well tuned.

6. Conclusion and Future work

In this paper we proposed a modified Stochastic Gradient Descent method consisting of two steps of update for each minibatch. We showed that the proposed method can be considered as a way to approximately eliminate the systemic bias that comes from training on finite data. We observed around 10% relative improvement in WER on multiple LVCSR datasets, versus best systems. The improvement on language modeling also suggests its potential effectiveness in other tasks.

As future work, we would like to evaluate the proposed method in other tasks such as object recognition/classification and machine translation, etc. We are also working on the combination of backstitch with momentum.

7. References

[1] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” *ICLR*, 2015.

[2] R. Huang, B. Xu, D. Schuurmans, and C. Szepesvári, “Learning with a strong adversary,” *ICLR*, 2016.

[3] A. Nokland, “Improving back-propagation by adding an adversarial gradient,” *arXiv preprint arXiv:1510.04189*, 2015.

[4] P. Tabacof and E. Valle, “Exploring the space of adversarial images,” in *Neural Networks (IJCNN), 2016 International Joint Conference on*. IEEE, 2016, pp. 426–433.

[5] D. Povey, V. Peddinti, D. Galvez, P. Ghahrmani, V. Manohar, X. Na, Y. Wang, and S. Khudanpur, “Purely sequence-trained neural networks for ASR based on lattice-free MMI,” *Proc. Interspeech*, 2016.

[6] Y. Wang, H. Hadian, S. Ding, K. Li, H. Xu, X. Zhang, D. Povey, and S. Khudanpur, “Backstitch: Counteracting finite-sample bias via negative steps,” in *NIPS (submitted)*, 2017.

[7] H. H. Yang and S.-i. Amari, “Complexity issues in natural gradient descent method for training multilayer perceptrons,” *Neural Computation*, vol. 10, no. 8, pp. 2137–2157, 1998.

[8] N. L. Roux, P.-A. Manzagol, and Y. Bengio, “Topmoumoute online natural gradient algorithm,” in *Advances in neural information processing systems*, 2008, pp. 849–856.

[9] J. Martens and R. B. Grosse, “Optimizing neural networks with kronecker-factored approximate curvature.” in *ICML*, 2015, pp. 2408–2417.

[10] D. Povey, X. Zhang, and S. Khudanpur, “Parallel training of deep neural networks with natural gradient and parameter averaging,” *ICLR: Workshop track*, 2015.

[11] Y. Pawitan, *In all likelihood: statistical modelling and inference using likelihood*. Oxford University Press, 2001.

[12] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlíček, Y. Qian, P. Schwarz *et al.*, “The kaldi speech recognition toolkit,” *Proc. ASRU*, 2011.

[13] T. Ko, V. Peddinti, D. Povey, and S. Khudanpur, “Audio augmentation for speech recognition.” *Proc. Interspeech*, pp. 3586–3589, 2015.

[14] G. Saon, H. Soltau, D. Nahamoo, and M. Picheny, “Speaker adaptation of neural network acoustic models using i-vectors.” in *ASRU*, 2013, pp. 55–59.

[15] G. Chen, H. Xu, M. Wu, D. Povey, and S. Khudanpur, “Pronunciation and silence probability modeling for asr.” *Proc. Interspeech*, pp. 533–537, 2015.

[16] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, “Phoneme recognition using time-delay neural networks,” *IEEE transactions on acoustics, speech, and signal processing*, vol. 37, no. 3, pp. 328–339, 1989.

[17] V. Peddinti, D. Povey, and S. Khudanpur, “A time delay neural network architecture for efficient modeling of long temporal contexts,” *Proc. Interspeech*, 2015.

[18] M. D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean *et al.*, “On rectified linear units for speech processing,” in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 3517–3521.

[19] A. L. Maas, A. Y. Hannun, and A. Y. Ng, “Rectifier nonlinearities improve neural network acoustic models,” in *Proc. ICML*, vol. 30, no. 1, 2013.

[20] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[21] A. Graves and J. Schmidhuber, “Framewise phoneme classification with bidirectional LSTM and other neural network architectures,” *Neural Networks*, vol. 18, no. 5, pp. 602–610, 2005.

[22] H. Sak, A. W. Senior, and F. Beaufays, “Long short-term memory recurrent neural network architectures for large scale acoustic modeling.” *Proc. Interspeech*, pp. 338–342, 2014.

[23] V. Peddinti, Y. Wang, D. Povey, and S. Khudanpur, “Low latency modeling of temporal contexts.” *IEEE Signal Processing Letters (submitted)*, 2017.

[24] A. Rousseau, P. Deléglise, and Y. Estève, “TED-LIUM: an automatic speech recognition dedicated corpus.” in *LREC*, 2012, pp. 125–129.

[25] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, “Recurrent neural network based language model.” in *Interspeech*, vol. 2, 2010, p. 3.