Backstitch: Counteracting Finite-sample Bias via Negative Steps

Yiming Wang[†] **Hossein Hadian**[†] **Shuoyang Ding**[†] **Ke Li**[†]

Hainan Xu[†] **Xiaohui Zhang**[†] **Daniel Povey**[†,‡] **Sanjeev Khudanpur**[†,‡]

[†]Center for Language and Speech Processing
[‡]Human Language Technology Center of Excellence
The Johns Hopkins University, Baltimore, MD 21218, USA
{yiming.wang,hhadian,dings,kli26,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, and then a forward step with a larger learning rate. This was initially inspired by ideas from adversarial training, but we show that it can be viewed as a crude way of canceling out the bias that arises when we train on a finite data set. Experiments with deep neural networks on a range of machine learning tasks show consistent improvements using our proposed method.

1 Introduction

Recently, the concept of training on adversarial examples has been proposed [6, 11, 20, 34]. 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 $\alpha > 0$ (e.g. $\alpha = 0.3$), 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, and then a larger positive step. This resulted in quite large improvements – around 10% relative improvement [37] – for our best speech recognition DNNs trained with lattice-free MMI (LF-MMI) [25].

In this paper we try this method on a number of applications of DNNs, and present a theory that suggests why it might work.

In Section 2 we will discuss finite-sample bias and how it affects learning 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 cancels finite-sample bias. Section 4 discusses the interaction of backstitch training with various other aspects of the training algorithms. Section 5 shows our experimental results on multiple datasets, and we conclude in Section 6.

2 Finite-sample bias

When we estimate parameters from a finite amount of data, the estimates may be biased. A classic example is estimating a mean and variance from data using Maximum Likelihood: the expected value of the estimated variance is less than the true variance. In this section we derive a quite general expression for the bias that arises when we estimate parameters from a finite amount of data. Later we will show that, under appropriate conditions, the backstitch algorithm can counteract this bias.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

Suppose we are minimizing a loss function $L_{\boldsymbol{\theta}}(X)$, where $\boldsymbol{\theta}$ is a parameter vector (e.g. of a neural network) and the random variable X is a training example; in a classification example, a sample x might consist of $\langle$feature vector, class label$\rangle$ pairs.

Let us suppose that $\boldsymbol{\theta}^*$ minimizes the expected loss $E\left[L_{\boldsymbol{\theta}}(X)\right]$ w.r.t the underlying distribution of X. Let $\hat{\boldsymbol{\theta}}$ be an estimate of $\boldsymbol{\theta}$ obtained by minimizing the empirical loss $\frac{1}{n}\sum_{i=1}^{n} L_{\boldsymbol{\theta}}(x_i)$ over n i.i.d. samples, $x_1, \ldots x_n$ of X. We will assume that $L_{\boldsymbol{\theta}}(\cdot)$ is twice differentiable at $\boldsymbol{\theta}^*$ and has a full-rank Hessian, and that the order of expectation (integration) and differentiation may be interchanged.

We further suppose that the objective function is well enough approximated by a quadratic function, that we can get a good estimate $\hat{\boldsymbol{\theta}}$ by doing a single iteration of Newton's method starting from the true global optimum $\boldsymbol{\theta}^*$. (Obviously in real life we would never have access to $\boldsymbol{\theta}^*$.) This gives us:

$$\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}^* - \widehat{\mathbf{H}}^{-1}\hat{\mathbf{g}} \tag{1}$$

where

$$\hat{\mathbf{g}} \triangleq \frac{1}{n}\sum_{i=1}^{n} \mathbf{g}_i \qquad\qquad \widehat{\mathbf{H}} \triangleq \frac{1}{n}\sum_{i=1} \mathbf{H}_i \tag{2}$$

with

$$\mathbf{g}_i \triangleq \mathbf{g}_{\boldsymbol{\theta}}(x_i) = \left.\frac{\partial L_{\boldsymbol{\theta}}(x_i)}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*} \qquad\qquad \mathbf{H}_i \triangleq \mathbf{H}_{\boldsymbol{\theta}}(x_i) = \left.\frac{\partial^2 L_{\boldsymbol{\theta}}(x_i)}{\partial^2 \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*} \tag{3}$$

In other words, $\hat{\mathbf{g}}$ is the parameter gradient averaged over n training samples and $\widehat{\mathbf{H}}$ is Hessian averaged in the same way; everything in (1) except $\boldsymbol{\theta}^*$ is a random variable because of the dependency on the randomly drawn samples $x_1 \ldots x_n$.

We would like to evaluate the bias $E[\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^*]$, but this is nontrivial because of the nonlinear nature of matrix inverse in $\widehat{\mathbf{H}}^{-1}$. Let us write the expected Hessian as $\overline{\mathbf{H}}$, and define

$$\mathbf{D} \triangleq \widehat{\mathbf{H}} - \overline{\mathbf{H}} \tag{4}$$

thus we have $\widehat{\mathbf{H}} = \overline{\mathbf{H}} + \mathbf{D}$; we expect $\mathbf{D}$ to be small if n is large. Then, $\widehat{\mathbf{H}}^{-1} = (\overline{\mathbf{H}} + \mathbf{D})^{-1} = \overline{\mathbf{H}}^{-0.5}(\mathbf{I} + \overline{\mathbf{H}}^{-0.5}\mathbf{D}\overline{\mathbf{H}}^{-0.5})^{-1}\overline{\mathbf{H}}^{-0.5}$, and retaining up to the first order term[1] of the series $(\mathbf{I} + \mathbf{X})^{-1} = \mathbf{I} - \mathbf{X} + \mathbf{X}^2 - \mathbf{X}^3 \ldots$, we obtain:

$$\widehat{\mathbf{H}}^{-1} \simeq 2\overline{\mathbf{H}}^{-1} - \overline{\mathbf{H}}^{-1}\widehat{\mathbf{H}}\overline{\mathbf{H}}^{-1} \tag{5}$$

Substituting this approximation into (1), taking expectations and rearranging to get the bias, we have:

$$E[\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^*] \simeq E\left[-\left(2\overline{\mathbf{H}}^{-1} - \overline{\mathbf{H}}^{-1}\widehat{\mathbf{H}}\overline{\mathbf{H}}^{-1}\right)\hat{\mathbf{g}}\right] \tag{6}$$

Since $E[\hat{\mathbf{g}}] = 0$ and $\overline{\mathbf{H}}$ is not a random variable, we can drop the first term, leaving:

$$E[\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^*] \simeq E[\overline{\mathbf{H}}^{-1}\widehat{\mathbf{H}}\overline{\mathbf{H}}^{-1}\hat{\mathbf{g}}] \tag{7}$$

We can compute the expectation on the r.h.s. in (7) by expanding the summations in (2) and noticing that the "cross-terms" (between differently numbered samples) are zero, giving:

$$E[\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^*] \simeq \frac{1}{n}E[\overline{\mathbf{H}}^{-1}\mathbf{H}\overline{\mathbf{H}}^{-1}\mathbf{g}] \tag{8}$$

If the gradient and Hessian are independent, the bias is zero.

But if they're not, what would it take to cancel out this bias? Suppose we add a non-random (fixed) correction term $\mathbf{c}$ to the gradients $\mathbf{g}$ used in (1), it is clear that it would change the estimated value $\hat{\boldsymbol{\theta}}$ by $-\widehat{\mathbf{H}}^{-1}\mathbf{c}$; and in expectation this would be the same as $-\overline{\mathbf{H}}^{-1}\mathbf{c}$ (ignoring $O(1/n^2)$ terms). So to cancel out the bias of (8), we would want to train with a modified gradient

$$\mathbf{g}' = \mathbf{g} + \mathbf{c} \tag{9}$$

where

$$\mathbf{c} = \frac{1}{n}E[\mathbf{H}\overline{\mathbf{H}}^{-1}\mathbf{g}] \tag{10}$$

We will show in the next section that backstitch training leads to an extra term in the gradient whose expected value looks a lot like the expression for $\mathbf{c}$ in (10).

[1] If we were being more careful we would show that the discarded terms are $o(n^2)$ in a suitable sense, but this gets complicated fast.

3 Backstitch training

In this section we write down the algorithm used in backstitch training and show how it may help to counteract the estimation bias that we discussed above. We write as if we were training on individual training samples, not minibatches. The use of minibatches does not fundamentally change the math because we can treat the minibatch itself as the random variable X.

We first write down an iteration of regular SGD with learning rate ν:

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \nu \mathbf{g}_t \tag{11}$$

where $\mathbf{g}_t$ is the gradient w.r.t. the sample x_{i_t} chosen on the t'th iteration:

$$\mathbf{g}_t \triangleq \frac{\partial}{\partial \boldsymbol{\theta}} L_{\boldsymbol{\theta}}(x_{i_t})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t} \tag{12}$$

In backstitch, we first take a step with the opposite sign, and then a larger step with normal sign. Let $\alpha \geq 0$ be the backstitch scale. Taking slight liberties with notation, we split the iteration into two halves:

$$\boldsymbol{\theta}^{t+\frac{1}{2}} = \boldsymbol{\theta}^t + \alpha\nu \mathbf{g}_t \tag{13}$$

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^{t+\frac{1}{2}} - (1+\alpha)\nu \mathbf{g}_{t+\frac{1}{2}} \tag{14}$$

where $\mathbf{g}_{t+\frac{1}{2}}$ is the gradient computed with the updated parameters but with the same data, i.e.:

$$\mathbf{g}_{t+\frac{1}{2}} \triangleq \frac{\partial}{\partial \boldsymbol{\theta}} L_{\boldsymbol{\theta}}(x_{i_t})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_{t+\frac{1}{2}}} \tag{15}$$

We can telescope the two updates into one by making a quadratic approximation to $L_{\boldsymbol{\theta}}(x_{i_t})$, giving:

$$\boldsymbol{\theta}^{t+1} \simeq \boldsymbol{\theta}^t - \nu \mathbf{g}_t - \alpha(1+\alpha)\nu^2 \mathbf{H}_t \mathbf{g}_t \tag{16}$$

If we view this as a correction term $\mathbf{c}$ added to the gradient as in (9), then we could write it as:

$$\mathbf{c}_t = \alpha(1+\alpha)\nu \mathbf{H}_t \mathbf{g}_t \tag{17}$$

If the problem is "perfectly conditioned" in the sense that the expected Hessian $\overline{\mathbf{H}}$ is identity, and we choose α such that $\alpha(1+\alpha)\nu = 1/n$, where n is the number of training samples (or the number of minibatches, if we are using minibatches), then the expected value of $\mathbf{c}_t$ would be equal to the "ideal" $\mathbf{c}$ in (10). This suggests why backstitch might be useful[2].

3.1 Discussion of the theory

We have shown— not very rigorously— that under some rather strong conditions, our backstitch training algorithm could cancel out the principal $O(1/n)$ terms in the bias in the parameter estimate, giving us a less-biased way of estimating the parameters. We will leave it to other authors to perform a more rigorous analysis, as our main interest in the method is a practical one. We feel that at least we have shown that backstitch is not a *completely* crazy thing to do. We also hope that the ideas presented may inspire others to come up with more precise methods to reduce parameter-estimation bias.

An alternative and perhaps more intuitive way of looking at backstitch is to say that in the first step, we are subtracting the effect of the current minibatch from the model, so that in the second step the data acts more like freshly sampled data, and less like "tainted" data that we have already trained on.

The analysis above inspired us to try replacing (14) with a version that used a more general combination of the two computed gradients $\mathbf{g}_t$ and $\mathbf{g}_{t+\frac{1}{2}}$, leading to a method that has two tunable parameters instead of the single value α. However we did not find any experimental evidence that the more general version was better than the simple version, so we will not say anything more about it.

We believe that of the assumptions we made in the math, the most problematic one is the assumption the loss function is well approximated by a quadratic. Numerical experiments bear out our conclusions as long as our loss functions are quadratic; but when we applied our analysis to the classic example of the variance estimate being biased when estimating the mean and variance of a distribution, we got an expression for the bias that was wrong by a constant factor.

[2]The reader might notice that we treated $\mathbf{c}$ as a fixed vector in the analysis in the previous section, but the $\mathbf{c}_t$ here is random. The analysis is a little more complicated with random $\mathbf{c}$, but we believe the basic conclusion is the same.

4 Other aspects of backstitch training

In this section we discuss some implementation issues in backstitch training and how it interacts with various other methods.

4.1 Efficiency and backstitch interval

Naïvely implemented, backstitch training would take twice as long per epoch as regular training because we need to train twice on each minibatch. 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 $m = 1$ can be more efficiently obtained with $\alpha = 1.0$ and $m = 4$, at least for speech recognition tasks (we have not done such extensive experiments on other applications).

4.2 Interaction with natural gradient

Many of our experimental results are in a Kaldi[24]-based setup where we use Natural Gradient (NG)[39, 28, 18] for training, as described in [26]. Natural Gradient can be viewed as a change of variables into a space where 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 and Hessian should have the same value (up to the negative sign [21]. So from a theoretical perspective there is a reason to expect that backstitch training should work particularly well with NG. We have experimented with backstitch with NG disabled (details not in this paper), and while the improvement from backstitch is somewhat less when NG is not used, it does still help. Also, some of our experiments reported here are with setups where NG is not supported, and we still see improvements.

4.3 Interaction with natural gradient updates

We are using the "online NG" method described in [26]. This involves updating an estimate of the Fisher matrix every few minibatches. It is necessary for the SGD to converge to an optimum, that the estimate of the Fisher matrix should be obtained from *previous* minibatches; otherwise a bias arises. 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 (13) and (14), 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 this degraded the results.

4.4 Backstitch training and parameter maximum changes

In our Kaldi-based neural net training tools, to prevent instability we enforce a maximum parameter-change (in Euclidian distance in parameter space). This is done at two levels: per component (roughly: per layer), 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 setups do not use momentum, as NG is effective in preventing instability and we usually find that momentum hurts performance. The only experiments here that used momentum are the baseline (non-backstitch) versions of the cross-entropy trained TDNN-LSTM systems, with momentum at 0.5.

4.6 Backstitch training and dropout

For experiments with dropout, we ensure that the dropout masks used in the first and second passes for a particular minibatch are the same. We found that if we do not ensure this, backstitch does not improve performance.

4.7 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, without applying backstitch. We do not understand why this happens, but in order to prevent any possible bad effects, in the Kaldi-based setups we always linearly increase α from zero to the specified α value over the first 15 iterations[3].

5 Experiments

We did experiments on four applications: speech recognition, image classification, RNNLM based language modeling, and neural machine translation.

5.1 Speech Recognition

We conducted experiments with the Kaldi speech recognition toolkit [24], with techniques including speed perturbation [14], i-vector adaptation [31] and pronunciation and silence probability modeling [4]. We did extensive experiments on three different LVCSR tasks using backstitch SGD training with different setups, including different criteria (CE and LF-MMI [25]) and different network architectures– Time-Delay Neural Network [36, 22] (TDNN) with ReLU nonliearities [42, 17], Bidirectional LSTM [10, 7] (BLSTM) consisting of stacked LSTMP layers [29], and a mixture of TDNN and unidirectional LSTM [23] (TDNN-LSTM) which was found not only outperforms BLSTM, but is also computationally more efficient. The backstitch scale α, backstitch interval m (see Sec. 4.1) and number of epochs were tuned for each individual backstitch training experiment, therefore these hyper-parameters vary across different setups[4]. 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 number of epochs with backstitch training is $1.5 \sim 2$ times the number of epochs with normal training because we found that word error rate (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[5]. The results of LF-MMI and CE systems are shown in Table 1 ($\alpha = 0.0$ 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: Backstitch versus regular SGD in LF-MMI (left) and CE (right) systems.

	LF-MMI					CE				
Dataset	$\alpha[/m]$	WER (%)				$\alpha[/m]$	WER (%)			
		TDNN-LSTM		BLSTM			TDNN-LSTM		BLSTM	
		dev	eval	dev	eval		dev	eval	dev	eval
AMI-SDM	0.0	37.9	41.1	39.7	42.9	0.0	36.8	40.7	37.5	41.3
	1.0/4	34.1	37.6[6]	36.7	40.4	0.3/1 1.0/4	35.1	39.1	35.7	39.2
(use IHM alignment)	Rel. Gain (%)	**10.0**	**8.5**	**7.6**	**5.8**		**4.6**	**3.9**	**4.8**	**5.1**
		eval2000[7]	rt03	eval2000	rt03		eval2000	rt03	eval2000	rt03
Switchboard	0.0	14.1	16.7	14.3	16.6	0.0	15.3	17.7	14.6	16.8
	0.3/1	12.7	14.9	13.4	15.7	1.0/4	14.9	16.9	14.1	16.4
	Rel. Gain (%)	**10.0**	**10.8**	**6.3**	**5.4**		**2.6**	**4.5**	**3.4**	**2.4**
		dev	test	dev	test		dev	test	dev	test
TED-LIUM	0.0	9.4	8.8	9.4	9.0	0.0	10.9	10.0	10.3	9.4
	1.0/4	8.3	7.8	8.7	8.1	1.0/4	10.8	9.7	9.8	9.0
[27]	Rel. Gain (%)	**11.7**	**11.4**	**7.4**	**10.0**		**0.9**	**3.0**	**4.9**	**4.3**

[3] 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 a few hundred minibatches.

[4] However we never used Switchboard rt03 for tuning; if the reader is concerned that this might bias the results, please focus on the rt03 numbers in Table 1.

[5] Except that momentum is disabled when doing backstitch training in CE systems. See Section 4.5.

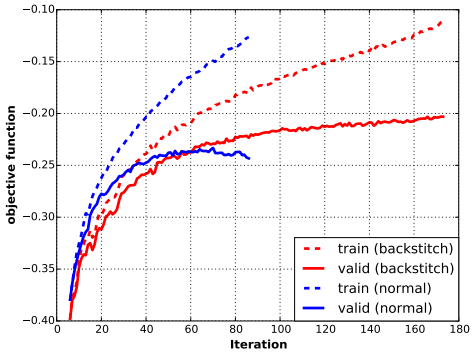


Figure 1: Plot of objective function vs iterations on AMI-SDM using LF-MMI+TDNN-LSTM (we continued backstitch training for twice the epochs of normal training).

Figure 1 compares the training objective functions 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.

5.2 Image Classification

We also used Kaldi's "nnet3" neural network toolkit for image classification experiments with CIFAR [16]. The architecture is a form of Wide Residual Network [40].

Our residual blocks [8] have pre-activation nonlinearities, but with ReLU first, so a non-bottleneck res-block would be (ReLU, batch-norm, conv, ReLU, batch-norm, conv); we did this because our batch-norm [12] implementation does not include the trainable scale and shift, so to retain modeling power it needs to come after the ReLU. The skip-layer ("residual" connection) passes through the first ReLU and batch-norm. The ResNet configuration is shallow, with two res-blocks operating on a 32x32 image, then two res-blocks on a 16x16 image, then three bottleneck res-blocks [9] on an 8x8 image, then spatial averaging over the 8x8 image and a projection to the the output soft-max. Between the three groups of res-blocks we downsample with a single convolutional layer with 3x3 filters (without a nonlinearity), and there is also a convoulutional layer at the beginning of the network (again, without a nonlinearity), so the total number of convolutions is 20. The number of filters in the 3 groups of res-blocks was (48, 128 and 384) respectively, and the bottleneck dimension in the 3rd group was 192. The model has 2.6 million parameters.

The minibatch size is 256. We do not use momentum, and the learning rate schedule decreases exponentially from 0.003 to 0.0003. However, we add gradients, rather than average, over samples, so the conventionally described learning rate varies from about 0.76 to 0.076. Instability is controlled by maximum parameter-changes (Sec. 4.4) and NG [26][8]. After the specified number of epochs we use a training-data subset to optimize a linear combination of the models from the last few checkpoints/iterations of training (this actually helps more for image recognition than it does for speech tasks). Our toolkit does not support L2 regularization because the interaction with NG (at least, for non-convolutional layers) is non-trivial, so we approximate it by periodically scaling down the parameters; this turned out to be essential for image classification.

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

[7]These results are with the entirety of eval2000, not the easier Switchboard-only subset which has become popular lately. Our best WER, 12.7%, is 8.4% on the subset. Numbers close to 6% have been published on this subset [38, 30] but these are not comparable because we trained only on Switchboard and used a conventional ARPA language model.

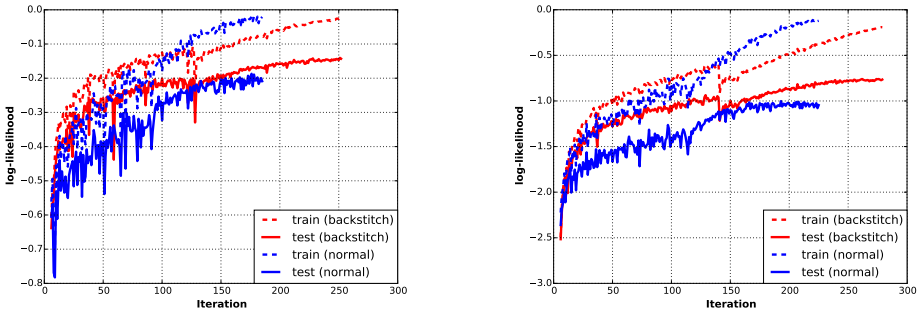
[8]For convolutional layers we apply it slightly differently than described in the paper: we first compute the gradient over the whole minibatch, then when estimating the column-space Fisher matrix we treat the different rows of the parameter matrix as the observations, and vice versa for the row-space Fisher matrix— instead of treating different training samples as the observations. The parameter matrix for convolutional layers has num-rows equal to the number of output filters, and num-cols equal to the number of input filters times the number of pixels in the patch.

For augmentation we use horizontal flipping and up to 4 pixels of random horizontal and vertical shift (padding with nearest pixel); we do not use any image preprocessing or normalization such as ZCA or mean/std normalization.

Our baseline experiments were run for 140 epochs on CIFAR-10 and 170 epochs on CIFAR-100, while backstitch training was run for 190 epochs on CIFAR-10 and 210 epochs on CIFAR-100 with $\alpha = 0.5, m = 1$ (using more epochs did not help the baseline). Table 2 shows the results (we only ran this once). The test error rate improves by 0.34 and 1.97 percentage points respectively. Figure 2 gives the log-likelihood comparison between backstitch training and normal training, which shows a similar trend to those in our speech experiments.

Table 2: Backstitch versus normal SGD on CIFAR-10 and CIFAR-100.

Dataset	$\alpha[/m]$	Error Rate (%)	
		train	test
CIFAR-10	0.0	0.00	4.35
	0.5/1	0.32	4.01
	Abs. Δ	0.32	**-0.34**
		train	test
CIFAR-100	0.0	0.46	22.15
	0.5/1	3.58	20.18
	Abs. Δ	3.02	**-1.97**



(a) (b) .

Figure 2: Log-likelihood vs iterations on CIFAR-10 (a) and CIFAR-100 (b).

5.3 Language Modeling

For the RNNLM [19] based language modeling task, we trained a two-layer LSTM language model with CE objective functions on English Penn Treebank dataset [35], using Keras with the TensorFlow backend. The code is from [5], which is a Keras reimplementation of [13]. The vocabulary size is $10,000$, the hidden dimension is 896, and the total number of parameters is 30 million. SGD is used as the baseline with minibatch size 32 and the learning rate being halved if the perplexity was not reduced significantly on the validation set after one epoch of training. The backstitch scale is tuned to be 0.2. We stopped the training when the loss on the validation set started to increase, and evaluated the perplexity on the test set. We see similar trends from their objective plots (not shown here) as before. The final perplexity on the test set is reduced from 118.5 to 109.8 using backstitch training.

5.4 Neural Machine Translation

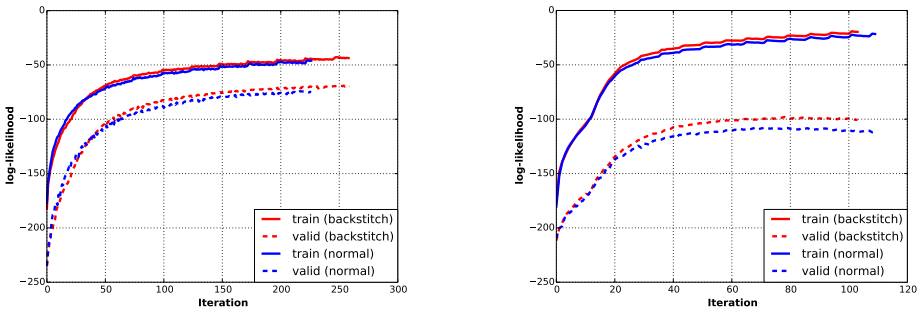
We carried out neural machine translation experiments with backstitch under two different language pairs and data settings. For the Chinese-English (`zh-en`) experiment, we used a collection of LDC parallel data amount to 2.04 million sentences on each side and 42.7 million tokens on the English side. We used `eval05` and `eval08` dataset from NIST OpenMT Evaluation for validation and testing. For the Romanian-English (`ro-en`) experiment, we used the official evaluation data from WMT16 news translation task [2]. The training set of the evaluation data is consisted of 0.608 million sentences on each side and 15.7 million tokens on the English side. All the data except Chinese were tokenized and truecased with the scripts in Moses [15], while the Chinese data was only tokenized with the Stanford Chinese Segmenter [3]. The BPE encoding [33] was then applied to the training set of both language pairs to reduce vocabulary size. We chose the number of new symbols created to be 39.5k.

All the experiments were conducted with the standard attention-based seq2seq model [1] implementation in the Nematus toolkit [32]. For the seq2seq model, we set the vocabulary size as 40k and used hidden dimension size 1024. We performed dropout with dropout rate 0.2 for the input bi-directional encoding and the hidden layer, and 0.1 for the source and target word embedding. To avoid gradient explosion, gradient clipping constant 1.0 was used. We chose AdaDelta [41] as the optimization algorithm for training and implemented backstitch on top of it. We used decay rate $\rho = 0.95$, $\varepsilon = 10^{-6}$ for all the reported results. For `zh-en` experiments, backstitch scale $\alpha = 0.5$ was used and we trained the system until the log probability on validation set stopped increasing for 10k updates, while for `ro-en` we used $\alpha = 0.3$, and we stopped if we do not observed an increase for 25k updates.

The results are in Table 3. We observed improvements in BLEU score for all the experiments with backstitch training. On the Romanian-English task, where there is less parallel data and the model is more prone to overfitting, we see more improvement. Log-likelihood plots are in Figure 3.

Table 3: Backstitch versus normal AdaDelta on `zh-en` and `ro-en` neural machine translation.

Language Pairs	α [/m]	BLEU	
		dev	test
ZH-EN	0.0	18.67	29.20
	0.5	19.14	29.85
	Abs. Gain	**0.47**	**0.65**
		dev	test
RO-EN	0.0	18.96	18.23
	0.3	21.33	20.13
	Abs. Gain	**2.37**	**1.90**



(a) (b) .

Figure 3: Log-likelihood vs iterations on `zh-en` (a) and `ro-en` (b) neural machine translation.

6 Conclusion

In this paper we proposed a modified Stochastic Gradient Descent method consisting of a negative and a positive step for each minibatch. We showed that the proposed method can be considered as a way to approximately eliminate the parameter estimation bias that comes from training on finite data. We showed that it reduces the difference between training and validation objective functions, and improves results on a variety of applications of neural networks.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] Ondrej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, et al. Findings of the 2016 conference on machine translation (wmt16). In *Proceedings of the First Conference on Machine Translation (WMT)*, volume 2, pages 131–198, 2016.

[3] Pi-Chuan Chang, Michel Galley, and Christopher D Manning. Optimizing chinese word segmentation for machine translation performance. In *Proceedings of the third workshop on statistical machine translation*, pages 224–232. Association for Computational Linguistics, 2008.

[4] Guoguo Chen, Hainan Xu, Minhua Wu, Daniel Povey, and Sanjeev Khudanpur. Pronunciation and silence probability modeling for asr. *Proc. Interspeech*, pages 533–537, 2015.

[5] José A. R. Fonollosa. Character-aware neural language models. a keras-based implementation. `https://github.com/jarfo/kchar`.

[6] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *ICLR*, 2015.

[7] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610, 2005.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016.

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[11] Ruitong Huang, Bing Xu, Dale Schuurmans, and Csaba Szepesvári. Learning with a strong adversary. *ICLR*, 2016.

[12] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 448–456, 2015.

[13] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[14] Tom Ko, Vijayaditya Peddinti, Daniel Povey, and Sanjeev Khudanpur. Audio augmentation for speech recognition. *Proc. Interspeech*, pages 3586–3589, 2015.

[15] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions*, pages 177–180. Association for Computational Linguistics, 2007.

[16] Alex Krizhevsky. Learning multiple layers of features from tiny images. *Tech Report*, 2009.

[17] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. *Proc. ICML*, 30(1), 2013.

[18] James Martens and Roger B Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *ICML*, pages 2408–2417, 2015.

[19] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

[20] Arild Nokland. Improving back-propagation by adding an adversarial gradient. *arXiv preprint arXiv:1510.04189*, 2015.

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

[22] Vijayaditya Peddinti, Daniel Povey, and Sanjeev Khudanpur. A time delay neural network architecture for efficient modeling of long temporal contexts. *Proc. Interspeech*, 2015.

[23] Vijayaditya Peddinti, Yiming Wang, Daniel Povey, and Sanjeev Khudanpur. Low latency modeling of temporal contexts. *IEEE Signal Processing Letters (submitted)*, 2017.

[24] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukáš Burget, Ondřej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlíček, Yanmin Qian, Petr Schwarz, et al. The kaldi speech recognition toolkit. *Proc. ASRU*, 2011.

[25] Daniel Povey, Vijayaditya Peddinti, Daniel Galvez, Pegah Ghahrmani, Vimal Manohar, Xingyu Na, Yiming Wang, and Sanjeev Khudanpur. Purely sequence-trained neural networks for ASR based on lattice-free MMI. *Proc. Interspeech*, 2016.

[26] Daniel Povey, Xiaohui Zhang, and Sanjeev Khudanpur. Parallel training of deep neural networks with natural gradient and parameter averaging. *ICLR: Workshop track*, 2015.

[27] Anthony Rousseau, Paul Deléglise, and Yannick Estève. TED-LIUM: an automatic speech recognition dedicated corpus. In *LREC*, pages 125–129, 2012.

[28] Nicolas L Roux, Pierre-Antoine Manzagol, and Yoshua Bengio. Topmoumoute online natural gradient algorithm. In *Advances in neural information processing systems*, pages 849–856, 2008.

[29] Hasim Sak, Andrew W Senior, and Francoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. *Proc. Interspeech*, pages 338–342, 2014.

[30] George Saon, Gakuto Kurata, Tom Sercu, Kartik Audhkhasi, Samuel Thomas, Dimitrios Dimitriadis, Xiaodong Cui, Bhuvana Ramabhadran, Michael Picheny, Lynn-Li Lim, et al. English conversational telephone speech recognition by humans and machines. *arXiv preprint arXiv:1703.02136*, 2017.

[31] George Saon, Hagen Soltau, David Nahamoo, and Michael Picheny. Speaker adaptation of neural network acoustic models using i-vectors. In *ASRU*, pages 55–59, 2013.

[32] Rico Sennrich, Orhan Firat, Kyunghyun Cho, Alexandra Birch, Barry Haddow, Julian Hitschler, Marcin Junczys-Dowmunt, Samuel Läubli, Antonio Valerio Miceli Barone, Jozef Mokry, et al. Nematus: a toolkit for neural machine translation. *arXiv preprint arXiv:1703.04357*, 2017.

[33] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016.

[34] Pedro Tabacof and Eduardo Valle. Exploring the space of adversarial images. In *Neural Networks (IJCNN), 2016 International Joint Conference on*, pages 426–433. IEEE, 2016.

[35] Ann Taylor, Mitchell Marcus, and Beatrice Santorini. The penn treebank: an overview. In *Treebanks*, pages 5–22. Springer, 2003.

[36] Alex Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin J Lang. Phoneme recognition using time-delay neural networks. *IEEE transactions on acoustics, speech, and signal processing*, 37(3):328–339, 1989.

[37] Yiming Wang, Vijayaditya Peddinti, Hainan Xu, Xiaohui Zhang, Daniel Povey, and Sanjeev Khudanpur. Backstitch: Counteracting finite-sample bias via negative steps. *Interspeech (submitted)*, 2017.

[38] Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Mike Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. Achieving human parity in conversational speech recognition. *arXiv preprint arXiv:1610.05256*, 2016.

[39] Howard Hua Yang and Shun-ichi Amari. Complexity issues in natural gradient descent method for training multilayer perceptrons. *Neural Computation*, 10(8):2137–2157, 1998.

[40] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016.

[41] Matthew D Zeiler. ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[42] Matthew D Zeiler, M Ranzato, Rajat Monga, Min Mao, Kun Yang, Quoc Viet Le, Patrick Nguyen, Alan Senior, Vincent Vanhoucke, Jeffrey Dean, et al. On rectified linear units for speech processing. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 3517–3521. IEEE, 2013.