Active Bias: Training More Accurate Neural Networks by Emphasizing High Variance Samples

Haw-Shiuan Chang, Erik Learned-Miller, Andrew McCallum
University of Massachusetts, Amherst
140 Governors Dr., Amherst, MA 01003
{hschang,elm,mccallum}@cs.umass.edu

Abstract

Self-paced learning and hard example mining re-weight training instances to improve learning accuracy. This paper presents two improved alternatives based on lightweight estimates of sample uncertainty in stochastic gradient descent (SGD): the variance in predicted probability of the correct class across iterations of mini-batch SGD, and the proximity of the correct class probability to the decision threshold. Extensive experimental results on six datasets show that our methods reliably improve accuracy in various network architectures, including additional gains on top of other popular training techniques, such as residual learning, momentum, ADAM, batch normalization, dropout, and distillation.

1 Introduction

Learning easier material before harder material is often beneficial to human learning. Inspired by this observation, curriculum learning [5] has shown that learning from easier instances first can also improve neural network training. When it is not known a priori which samples are easy, examples with lower loss on the current model can be inferred to be easier and can be used in early training. This strategy has been referred to as self-paced learning [25]. By decreasing the weight of difficult examples in the loss function, the model may become more robust to outliers [33], and this method has proven useful in several applications, especially with noisy labels [66].

Nevertheless, selecting easier examples for training often slows down the training process because easier samples usually contribute smaller gradients, and the current model has already learned how to make correct predictions on these samples. On the other hand, and somewhat ironically, the opposite strategy (i.e., sampling harder instances more often) has been shown to accelerate (mini-batch) stochastic gradient descent (SGD) in some cases, where the difficulty of an example can be defined by its loss [18, 29, 44] or be proportional to the magnitude of its gradient [51, 1, 12]. This strategy is sometimes referred to as hard example mining [44].

In the literature, we can see that these two opposing strategies work well in different situations. Preferring easier examples may be effective when either machines or humans try to solve a challenging task containing more label noise or outliers. On the other hand, focusing on harder samples may accelerate and stabilize SGD in cleaner data by minimizing the variance of gradients [1, 12]. However, we often do not know how noisy our training dataset is. Motivated by this practical need, this paper explores new methods of re-weighting training examples that are effective in both scenarios.

Intuitively, if a model has already predicted some examples correctly with high confidence, those samples may be too easy to contain useful information for improving that model further. Similarly, if some examples are always predicted incorrectly over many iterations of training, these examples may just be too difficult/noisy and may degrade the model. This suggests that we should somehow prefer uncertain samples that are predicted incorrectly sometimes during training and correctly at
Figure 1: The proposed methods emphasize uncertain samples based on previous prediction history.

other times, as illustrated in Figure 1. This preference is consistent with common variance reduction strategies in active learning [43].

Previous studies suggest that finding informative unlabeled samples to label is related to selecting already-labeled samples to optimize the model parameters [14]. As reported in the previous studies [42, 6], models can sometimes achieve lower generalization error after being trained with only a subset of actively selected training data. In other words, focusing on informative samples can be beneficial even when all labels are available.

We propose two lightweight methods that actively emphasize uncertain samples to improve mini-batch SGD for classification. One method measures the variance of prediction probabilities, while the other one estimates the closeness between the prediction probabilities and the decision threshold. For logistic regression, both methods can be proven to reduce the uncertainty in the model parameters under reasonable approximations.

We present extensive experiments on CIFAR 10, CIFAR 100, MNIST (image classification), Question Type (sentence classification), CoNLL 2003, and OntoNote 5.0 (Named entity Recognition), as well as on different architectures, including multiple class logistic regression, fully-connected networks, convolutional neural networks (CNNs) [26], and residual networks [16]. The results show that active bias makes neural networks more robust without prior knowledge of noise, and reduces the generalization error by 1% –18% even on training sets having few (if any) annotation errors.

2 Related work

As (deep) neural networks become more widespread, many methods have recently been proposed to improve SGD training. When using (mini-batch) SGD, the randomness of the gradient sometimes slows down the optimization, so one common approach is to use the gradient computed in previous iterations to stabilize the process. Examples include momentum [38], stochastic variance reduced gradient (SVRG) [21], and proximal stochastic variance reduced gradient (Prox-SVRG) [49]. Other work proposes variants of semi-stochastic algorithms to approximate the exact gradient direction and reduce the gradient variance [47, 34]. More recently, supervised optimization methods like learning by learning [3] also show great potential in this problem.

In addition to the high variance of the gradient, another issue with SGD is the difficulty of tuning the learning rate. Like Quasi-Newton methods, several methods adaptively adjust learning rates based on local curvature [2, 40], while ADAGRAD [11] applies different learning rates to different dimensions. ADAM [23] combines several of these techniques and is widely used in practice.

More recently, some studies accelerate SGD by weighting each class differently [13] or weighting each sample differently as we do [18][51][29][12][1][44], and their experiments suggest that the methods are often compatible with other techniques such as Prox-SVRG, ADAGRAD, or ADAM [29][13]. Notice that Gao et al. [12] discuss the idea of selecting uncertain examples for SGD based on active learning, but their proposed methods choose each sample according to the magnitude of its gradient as in ISSGD [1], which actually prefers more difficult examples.

The aforementioned methods focus on accelerating the optimization of a fixed loss function given a fixed model. Many of these methods adopt importance sampling. That is, if the method prefers to
We propose a simple baseline which selects harder examples with higher probability, as done by SGD-Scan [41]. In order to gain more information at \(v\) that SGD Sampled by Easiness learning. Based on the same naming convention, make a model unbiased, importance sampling can be used. That is, we can let \(P\) models usually cannot fit outliers well, so SGD-SD and SGD-WD would not be robust to noise. To SGD Weighted by Difficulty \(v\) of can be set as \(\lambda R\) prediction loss, and \(L = \sum_{i} v_i \cdot \text{loss}_i(W) + \lambda R(W)\), where \(W\) are the parameters in the model, \(\text{loss}_i(W)\) is the prediction loss, and \(\lambda R(W)\) is the regularization term of the model. The weight of the \(i\)th sample \(v_i\) can be set as \(\frac{1}{N_D}(1 - \bar{p}_{H_i}^{-1}(y_i|x_i) + \epsilon_D)\), where \(N_D\) is a normalization constant making the average of \(v_i\) equal to 1. We want to keep the average of the \(v_i\) fixed so that we do not change the global learning rate. We denote this method SGD Weighted by Difficulty (SGD-WD).

Models usually cannot fit outliers well, so SGD-SD and SGD-WD would not be robust to noise. To make a model unbiased, importance sampling can be used. That is, we can let \(P_s(i|H, S_e, D) \propto 1 - \bar{p}_{H_e}^{-1}(y_i|x_i) + \epsilon_D\), and \(v_i \propto N_D(1 - \bar{p}_{H_i}^{-1}(y_i|x_i) + \epsilon_D)^{-1}\), which is similar to an approach used by Hinton [13]. We refer to this as SGD Importance-Sampled by Difficulty (SGD-ISD).

In addition, we propose two simple baselines that emphasize easy examples, as in self-paced learning. Based on the same naming convention, SGD Sampled by Easiness (SGD-SE) denotes that \(P_s(i|H, S_e, D) \propto \bar{p}_{H_i}^{-1}(y_i|x_i) + \epsilon_E\), while SGD Weighted by Easiness (SGD-WE) sets \(v_i = \frac{1}{N_E}(\bar{p}_{H_i}^{-1}(y_i|x_i) + \epsilon_E)\), where \(N_E\) normalizes the \(v_i\)’s to have unit mean.

### 3.2 Prediction Variance

In the active learning setting, the prediction variance can be used to measure the uncertainty of each sample for either a regression or classification problem [41]. In order to gain more information at each SGD iteration, we choose samples with high prediction variances.
Since the prediction variances are estimated on the fly, we would like to balance exploration and exploitation. Adopting the optimism in face of uncertainty heuristics of bandit problems \cite{7}, we draw the next sample based on the estimated prediction variance plus its confidence interval. Specifically, for SGD Sampled by Prediction Variance (SGD-SPV), we let

\[
P_s(iH, S_c, D) \propto \text{std}^\text{conf}_i(H) + \epsilon_V, \text{ where } \text{std}^\text{conf}_i(H) = \sqrt{\text{var}(p_{H_i^{-1}}(y_i|x_i)) + \frac{\text{var}(p_{H_i^{-1}}(y_i|x_i))^2}{|H_i^{-1}| - 1}},
\]

\[
\text{var}(p_{H_i^{-1}}(y_i|x_i)) \text{ is the prediction variance estimated by history } H_i^{-1}, \text{ and } |H_i^{-1}| \text{ is the number of stored prediction probabilities. Assuming } p_{H_i^{-1}}(y_i|x_i) \text{ is normally distributed under the uncertainty of model parameters } w, \text{ the variance of prediction variance estimation can be estimated by } 2 \cdot \text{var}(p_{H_i^{-1}}(y_i|x_i))^2(|H_i^{-1}| - 1)^{-1}. \text{ As we did in the baselines, adding the smoothness constant } \epsilon_V \text{ prevents the low variance instances from never being selected again. Similarly, another variant of the method sets } v_i = \frac{1}{N_v} (\text{std}^\text{conf}_i(H) + \epsilon_V), \text{ where } N_v \text{ normalizes } v_i \text{ like other weighted methods; we call this SGD Weighted by Prediction Variance (SGD-WPV).}
\]

As in SGD-WD, SGD-WE or self-paced learning \cite{4}, we train an unbiased model for several burn-in epochs at the beginning so as to judge the sampling uncertainty reasonably and stably. Other implementation details will be described in the first section of the supplementary material.

Using a low learning rate, model parameters \( w \) would be close to a good local minimum after sufficient burn-in epochs, and thus the posterior distribution of \( w \) can be locally approximated by a Gaussian distribution. Furthermore, the prediction distribution \( p(y_i|x_i, w) \) is often locally smooth with respect to the model parameters \( w \) (i.e., small changes of model parameters only induce small changes in the prediction distribution), so a Gaussian tends to approximate the distribution of \( p_{H_i^{-1}}(y_i|x_i) \) well in practice.

**Example: logistic regression**

Given a Gaussian prior \( Pr(W = w) = \mathcal{N}(w|0, s_0 I) \) on the parameters, consider the probabilistic interpretation of logistic regression:

\[
-\log(Pr(Y, W = w|X)) = -\sum_i \log(p(y_i|x_i, w)) - \frac{c}{s_0} ||w||^2, \quad (2)
\]

where \( p(y_i|x_i, w) = \frac{1}{1 + \exp(-y_i(w\cdot x_i))}, \text{ and } y_i \in \{1, -1\}. \)

Since the posterior distribution of \( W \) is log-concave \cite{39}, we can use \( Pr(W = w|Y, X) \approx \mathcal{N}(w|w_N, S_N) \), where \( w_N \) is maximum a posteriori (MAP) estimation, and

\[
S_N^{-1} = \nabla_w \nabla_w^\top - \log(Pr(Y, W|X)) = \sum_i p(y_i|x_i) (1 - p(y_i|x_i)) x_i x_i^\top + \frac{2c}{s_0} I. \quad (3)
\]

Then, we further approximate \( p(y_i|x_i, W) \) using the first order Taylor expansion \( p(y_i|x_i, W) \approx p(y_i|x_i, w) + g_i(w)(W - w) \), where \( g_i(w) = p(y_i|x_i, w)(1 - p(y_i|x_i, w)) x_i \). We can compute the prediction variance \cite{41} with respect to the uncertainty of \( W \)

\[
\text{Var}(p(y_i|x_i, W)) \approx g_i(w)^\top S_N g_i(w). \quad (4)
\]

These approximations tell us several things. First, \( \text{Var}(p(y_i|x_i, W)) \) is proportional to \( p(y_i|x_i, w)^2(1 - p(y_i|x_i, w))^2 \), so the prediction variance is larger when the sample \( i \) is closer to the boundary. Second, when we have more sample points close to the boundary, the variance of the parameters \( S_N \) is lower. That is, when we emphasize samples with high prediction variances, the uncertainty of parameters tends to be reduced, akin to the variance reduction strategy in active learning \cite{30}. Third, with a Gaussian assumption on the posterior distribution \( Pr(W = w|Y, X) \) and the Taylor expansion, the distribution of \( p(y_i|x_i, W) \) in logistic regression becomes Gaussian, which justifies our previous assumption of \( p_{H_i^{-1}}(y_i|x_i) \) for the confidence estimation of the prediction variance. Notice that there are other methods that can measure the prediction uncertainty, such as the mutual information between labels and parameters \cite{19}, but we found that the prediction variance works better in our experiments.
Figure 2: A toy example which compares different methods in a two-class logistic regression model. To visualize the optimization path for the classifier parameters (the red paths in (c), (e), and (g)) in two dimensions, we fix the weight corresponding to the x-axis to 0.5 and only show the weight for y-axis $\hat{w}[1]$ and bias term $b$. The i-th sample size in (f) and (h) is proportional to $v_i$. The toy example shows that SGD-WPV can train a more accurate model in a noisy dataset.

Figure 2 illustrates a toy example. Given the same learning rate, we can see that the normal SGD in Figure 2c and 2d will have higher uncertainty when there are many outliers, and emphasizing difficult examples in Figure 2e and 2f makes it worse. On the other hand, the samples near the boundaries would have higher prediction variances (i.e., larger circles or crosses in Figure 2h) and thus higher impact on the loss function in SGD-WPV.

After burn-in epochs, $w$ becomes close to a local minimum using SGD. Then, the parameters estimated in each iteration can be viewed, approximately, as samples drawn from the posterior distribution of the parameters $Pr(W = w|X, Y)$ \[31\]. Therefore, after running SGD long enough, $\bar{\text{Var}} \{p(y_i|x_i, W)\}$ can be used to approximate $\text{Var} \{p(y_i|x_i, W)\}$. Notice that if we directly apply bias at the beginning without running burn-in epochs, incorrect examples might be emphasized, which is also known as the local minimum problem in active learning \[14\]. For instance, in Figure 2 if burn-in epochs are not applied and the initial $w$ is a vertical line on the left, the outliers close to the initial boundary would be emphasized, which slows down the convergence speed.

In this simple example, we can also see that the gradient magnitude is proportional to the difficulty because $-\nabla_w \log(p(y_i|x_i, w)) = (1 - p(y_i|x_i, w)) x_i$. This is why we believe the SGD acceleration methods based on gradient magnitude \[12, 13\] can be categorized as variants of preferring difficult examples, and thus more vulnerable to outliers (like the samples on the left or right in Figure 2).

### 3.3 Threshold Closeness

Motivated by the previous analysis, we propose a simpler and more direct approach to select samples whose correct class probability is close to the decision threshold. **SGD Sampled by Threshold Closeness (SGD-STC)** makes $P_\Lambda(i|H, S_c, D) \propto \bar{p}^i_{H_t^{-1}}(y_i|x_i) \left(1 - \bar{p}^i_{H_t^{-1}}(y_i|x_i)\right) + \epsilon_T$, where $\bar{p}^i_{H_t^{-1}}(y_i|x_i)$ is the average probability of classifying sample $i$ into its correct class $y_i$ over all the stored $p(y_i|x_i)$ in $H_t^{-1}$. When there are multiple classes, this measures the closeness of the threshold for distinguishing the correct class out of the union of the rest of the classes (i.e., one-versus-rest). The method is similar to an approximation of the optimal allocation in stratified sampling proposed by Druck and McCallum \[10\].

Similarly, **SGD Weighted by Threshold Closeness (SGD-WTC)** chooses the weight of $i$th sample $v_i = \frac{1}{N_T} \bar{p}^i_{H_t^{-1}}(y_i|x_i) \left(1 - \bar{p}^i_{H_t^{-1}}(y_i|x_i)\right) + \epsilon_T$, where $N_T = \frac{1}{|\mathcal{D}|} \sum_i \bar{p}^i_{H_t^{-1}}(y_i|x_i) \left(1 - \bar{p}^i_{H_t^{-1}}(y_i|x_i)\right) + \epsilon_T$. The weighting can be viewed as combining the SGD-WD and SGD-WE by multiplying their weights.
Table 1: Model architectures. Dropouts and L2 reg (regularization) are only applied to the fully-connected (FC) layer(s).

<table>
<thead>
<tr>
<th>Dataset</th>
<th># Conv. layers</th>
<th>Filter size</th>
<th>Filter number</th>
<th># Pooling layers</th>
<th># BN layers</th>
<th># FC layers</th>
<th>Dropout keep probs</th>
<th>L2 reg</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>2</td>
<td>5x5</td>
<td>32, 64</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0.5, 0.0005</td>
<td>0.01</td>
</tr>
<tr>
<td>CIFAR 10</td>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0.5, 0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>CIFAR 100</td>
<td>26 or 62</td>
<td>3x3</td>
<td>16, 32, 64</td>
<td>0</td>
<td>13 or 31</td>
<td>1</td>
<td>1, 1</td>
<td>0</td>
</tr>
<tr>
<td>Question Type</td>
<td>1</td>
<td>(2, 3, 4)x1</td>
<td>64</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0.5, 0.75</td>
<td>0.001</td>
</tr>
<tr>
<td>CoNLL 2003</td>
<td>3</td>
<td>3x1</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.5, 0.75</td>
<td>0.001</td>
</tr>
<tr>
<td>OntoNote 5.0</td>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1, 1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2: Optimization hyper-parameters and experiment settings

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Optimizer</th>
<th>Batch size</th>
<th>Learning rate</th>
<th>Learning rate decay</th>
<th># Epochs</th>
<th># Burn-in epochs</th>
<th># Trials</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>Momentum</td>
<td>64</td>
<td>0.01</td>
<td>0.95</td>
<td>80</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>CIFAR 10</td>
<td>SGD</td>
<td>100</td>
<td>1e-6</td>
<td>0.5 (per 5 epochs)</td>
<td>30</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>CIFAR 100</td>
<td>Momentum</td>
<td>128</td>
<td>0.1</td>
<td>0.1 (at 80, 100, 120 epochs)</td>
<td>150</td>
<td>90 or 20</td>
<td></td>
</tr>
<tr>
<td>Question Type</td>
<td>ADAM</td>
<td>64</td>
<td>0.001</td>
<td>1</td>
<td>250</td>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td>CoNLL 2003</td>
<td>ADAM</td>
<td>128</td>
<td>0.0005</td>
<td>1</td>
<td>200</td>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>OntoNote 5.0</td>
<td>SGD</td>
<td>128</td>
<td>0.1</td>
<td>1</td>
<td>60</td>
<td>20</td>
<td>10</td>
</tr>
</tbody>
</table>

When using logistic regression, after injecting the bias \(v_i\) into the loss function, approximating the prediction probability based on previous history, removing the regularization and smoothness constant (i.e., \(p(y_i|x_i, w) \approx \bar{p}_{H_{t-1}}(y_i|x_i), 1/s_0 = 0, \) and \(\epsilon_T = 0\)), we can show that

\[
\sum_i \text{Var}(p(y_i|x_i, W)) \approx \sum_i g_i(w)^T S_N g_i(w) \approx N_T \cdot \text{dim}(w),
\]

where \(\text{dim}(w)\) is the dimension of parameters \(w\). This will ensure that the average prediction variance drops linearly as the number of training instance increases. The derivation could be seen in the supplementary materials.

4 Experiments

We test our methods on six different datasets. The results show that the active bias techniques constantly outperform the standard uniform sampling (i.e., SGD-Uni and SGD-Scan) in the deep models as well as the shallow models. For each dataset, we use an existing, publicly available implementation for the problem and emphasize samples using different methods. The architectures and hyper-parameters are summarized in Table 1. All neural networks use softmax and cross-entropy loss at the last layer. The optimization and experiment setups are listed in Table 2. As shown in the second column of the table, SGD in CNNs and residual networks actually refers to momentum or ADAM instead of vanilla SGD. All experiments use mini-batch.

Like most of the widely used neural network training techniques, the proposed techniques are not applicable to every scenario. For all the datasets we tried, we found that the proposed methods are not sensitive to the hyper-parameter setup except when applying a very complicated model to a relatively smaller dataset. If a complicated model achieves 100% training accuracy within a few epochs, the most uncertain examples would often be outliers, biasing the model towards overfitting.

To avoid this scenario, we modify the default hyper-parameters setup in the implementation of the text classifiers in Section 4.3 and Section 4.4 to achieve similar performance using simplified models. For all other models and datasets, we use the default hyper-parameters of the existing implementations, which should favor the SGD-Uni or SGD-Scan methods, since the default hyper-parameters are
We apply our method to a CNN [26] for MNIST. As previously discussed, there are various versions preferring easy or difficult examples. Some (SGD-S*). It might be because we can better estimate the statistics of each sample. The results from SGD-SD and SGD-WD confirm this finding while selecting uncertain examples can give us a similar or larger boost. Furthermore, we test the robustness of our methods by randomly selecting more difficult instances can accelerate learning or improve testing accuracy [18, 29, 13].

4.1 MNIST

We apply our method to a CNN using one of the Tensorflow tutorials. The dataset has high testing accuracy, so most of the examples are too easy for the model after a few epochs. Selecting more difficult instances can accelerate learning or improve testing accuracy [18, 29, 13]. The results from SGD-SD and SGD-WD confirm this finding while selecting uncertain examples can give us a similar or larger boost. Furthermore, we test the robustness of our methods by randomly

---

Table 3: The average of the best testing error rates for different sampling methods and datasets (%).

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Model</th>
<th>SGD-Uni</th>
<th>SGD-SD</th>
<th>SGD-ISD</th>
<th>SGD-SE</th>
<th>SGD-SPV</th>
<th>SGD-STC</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>CNN</td>
<td>0.55±0.01</td>
<td>0.52±0.01</td>
<td>0.57±0.01</td>
<td>0.54±0.01</td>
<td>0.51±0.01</td>
<td>0.51±0.01</td>
</tr>
<tr>
<td>Noisy MNIST</td>
<td>CNN</td>
<td>0.83±0.01</td>
<td>1.00±0.01</td>
<td>0.84±0.01</td>
<td>0.69±0.01</td>
<td>0.64±0.01</td>
<td>0.63±0.01</td>
</tr>
<tr>
<td>CIFAR 10</td>
<td>LR</td>
<td>62.49±0.06</td>
<td>63.14±0.06</td>
<td>62.48±0.07</td>
<td>60.87±0.06</td>
<td>60.66±0.06</td>
<td>61.02±0.06</td>
</tr>
<tr>
<td>QT</td>
<td>CNN</td>
<td>17.70±0.07</td>
<td>17.61±0.07</td>
<td>17.66±0.08</td>
<td>17.92±0.08</td>
<td>17.49±0.08</td>
<td>17.55±0.08</td>
</tr>
</tbody>
</table>

Table 4: The average of the best testing error rates and their standard errors for different weighting methods (%). For CoNLL 2003 and OntoNote 5.0, the values are 1-(F1 score). CNN, LR, RN 27, RN 63 and FC mean convolutional neural network, logistic regression, residual networks with 27 layers, residual network with 63 layers, and fully-connected network, respectively.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Model</th>
<th>SGD-Scan</th>
<th>SGD-WD</th>
<th>SGD-WE</th>
<th>SGD-WPV</th>
<th>SGD-WTC</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>CNN</td>
<td>0.54±0.01</td>
<td>0.48±0.01</td>
<td>0.56±0.01</td>
<td>0.48±0.01</td>
<td>0.48±0.01</td>
</tr>
<tr>
<td>Noisy MNIST</td>
<td>CNN</td>
<td>0.81±0.01</td>
<td>0.72±0.01</td>
<td>0.61±0.02</td>
<td>0.61±0.01</td>
<td>0.61±0.01</td>
</tr>
<tr>
<td>CIFAR 10</td>
<td>LR</td>
<td>62.48±0.06</td>
<td>63.10±0.06</td>
<td>60.88±0.06</td>
<td>60.61±0.06</td>
<td>61.02±0.06</td>
</tr>
<tr>
<td>CIFAR 100</td>
<td>RN 27</td>
<td>34.04±0.06</td>
<td>34.55±0.06</td>
<td>33.65±0.07</td>
<td>33.69±0.07</td>
<td>33.64±0.07</td>
</tr>
<tr>
<td>CIFAR 100</td>
<td>RN 63</td>
<td>30.70±0.06</td>
<td>31.57±0.09</td>
<td>29.92±0.09</td>
<td>30.02±0.08</td>
<td>30.16±0.09</td>
</tr>
<tr>
<td>QT</td>
<td>CNN</td>
<td>17.79±0.08</td>
<td>17.70±0.08</td>
<td>17.87±0.08</td>
<td>17.57±0.07</td>
<td>17.61±0.08</td>
</tr>
</tbody>
</table>

---

reassigning the labels of 10% of the images, and the results indicate that the SGD-WPV improves the performance of SGD-Scan even more while SGD-SD overfits the data seriously.

4.2 CIFAR 10 and CIFAR 100

We test a simple multi-class logistic regression on CIFAR 10 \[24\]. Images are down-sampled significantly to $32 \times 32 \times 3$, so many examples are difficult, even for humans. SGD-SPV and SGD-SE perform significantly better than SGD-Uni here, consistent with the idea that avoiding difficult examples increases robustness to outliers.

For CIFAR 100 \[24\], we demonstrate that the proposed approaches can also work in very deep residual networks \[16\]. To show the method is not sensitive to the network depth and the number of burn-in epochs, we present results from the network with 27 layers and 90 burn-in epochs as well as the network with 63 layers and 50 burn-in epochs. Without changing architectures, emphasizing uncertain or easy examples gains around 0.5% in both settings, which is significant considering the fact that the much deeper network shows only 3% improvement here.

When training a neural network, gradually reducing the learning rate (i.e., the magnitude of gradients) usually improves performance. When difficult examples are sampled less, the magnitude of gradients would be reduced. Thus, some of the improvement of SGD-SPV and SGD-SE might come from using a lower effective learning rate. Nevertheless, since we apply the aggressive learning rate decay in the experiments of CIFAR 10 and CIFAR 100, we know that the improvements from SGD-SPV and SGD-SE cannot be entirely explained by its lower effective learning rate.

4.3 Question Type

To investigate whether our methods are effective for smaller text datasets, we apply them to a sentence classification dataset (i.e. Question Type (QT) \[28\]), which contains 1000 training examples and 500 testing examples. We use the CNN architecture proposed by Kim \[22\]. Like many other NLP tasks, the dataset is relatively small and this CNN classifier does not inject noise to inputs like the implementation of residual networks in CIFAR 100, so this complicated model reaches 100% training accuracy within a few epochs.

To address this, we reduced the model complexity by (i) decreasing the number of filters from 128 to 64, (ii) decreasing convolutional filter widths from 3,4,5 to 2,3,4, (iii) adding L2 regularization with scale 0.01, (iv) performing PCA to reduce the dimension of pre-trained word embedding from 300 to 50 and fixing the word embedding during training. Then, the proposed active bias methods perform better than other baselines in this smaller model.

4.4 Sequence Tagging Tasks

We also test our methods on Named Entity Recognition (NER) in CoNLL 2003 \[46\] and OntoNote 5.0 \[20\] datasets using the CNN from Strubell et al. \[45\]. Similar to Question Type, the model is too complex for our approaches. So we (i) only use 3 layers instead of 4 layers, (ii) reduce the number of filters from 300 to 100, (iii) add 0.001 L2 regularization, (iv) make the 50 dimension word embedding from Collobert et al. \[9\] non-trainable. The micro F1 of this smaller model only drops around 1%-2% from the original big model. Table 4 shows that our methods achieve the lowest error rate (1-F1) in both benchmarks.

4.5 Distillation

Although state-of-the-art neural networks in many applications memorize examples easily \[50\], much simpler models can usually achieve similar performance like those in the previous two experiments. In practice, such models are often preferable due to their low computation and memory requirements.

https://cs231n.github.io/assignments2016/assignment2/
https://www.cs.toronto.edu/~kriz/cifar.html
https://github.com/tensorflow/models/tree/master/resnet
http://cogcomp.org/Data/QA/QC/
https://github.com/dennybritz/cnn-text-classification-tf
https://github.com/iesl/dilated-cnn-ner
We have shown that the proposed method can improve these smaller models as distillation did [17], so it is natural to check whether our methods can work well with distillation. We use an implementation that distills a shallow CNN with 3 convolution layers to a 2 layer fully-connected network in MNIST. The teacher network can achieve 0.8% testing error, and the temperature of softmax is set as 1.

Our approaches and baselines simply apply the sample dependent weights $v_i$ to the final loss function (i.e., cross-entropy of the true labels plus cross-entropy of the prediction probability from the teacher network). In MNIST, SGD-WTC and SGD-WD can achieve similar or better improvements compared with adding distillation into SGD-Scan. Furthermore, the best performance comes from the distillation plus SGD-WTC, which shows that active bias is compatible with distillation in this dataset.

5 Conclusion

Deep learning researchers often gain accuracy by employing training techniques such as momentum, dropout, batch normalization, and distillation. This paper presents a new compatible sibling to these methods, which we recommend for wide use. Our relatively simple and computationally lightweight techniques emphasize the uncertain examples (i.e., SGD-*PV and SGD-*TC).

The experiments confirm that the proper bias can be beneficial to generalization performance. When the task is relatively easy (both training and testing accuracy are high), preferring more difficult examples works well. On the contrary, when the dataset is challenging or noisy (both training and testing accuracy are low), emphasizing easier samples often lead to a better performance. In both cases, the active bias techniques consistently lead to more accurate and robust neural networks as long as the classifier does not memorize all the training samples easily (i.e., training accuracy is high but testing accuracy is low).

Acknowledgements

This material is based on research sponsored by National Science Foundation under Grant No. 1514053 and by DARPA under agreement number FA8750-1 3-2-0020 and HRO011-15-2-0036. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation 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 or the U.S. Government.

References


