Annealed Gradient Descent for Deep Learning

Hengyue Pan Hui Jiang
Department of Electrical Engineering and Computer Science
York University, 4700 Keele Street, Toronto, Ontario, Canada
Emails: panhy@cse.yorku.ca hj@cse.yorku.ca

Abstract

Stochastic gradient descent (SGD) has been regarded as a successful optimization algorithm in machine learning. In this paper, we propose a novel annealed gradient descent (AGD) method for non-convex optimization in deep learning. AGD optimizes a sequence of gradually improved smoother mosaic functions that approximate the original non-convex objective function according to an annealing schedule during the optimization process. We present a theoretical analysis on its convergence properties and learning speed. The proposed AGD algorithm is applied to learning deep neural networks (DNNs) for image recognition on MNIST and speech recognition on Switchboard. Experimental results have shown that AGD can yield comparable performance as SGD but it can significantly expedite training of DNNs in big data sets (by about 40% faster).

1 INTRODUCTION

The past few decades have witnessed the success of gradient descent algorithms in machine learning. By only calculating the local gradient information of the loss function, gradient descent (GD) algorithms may provide reasonably good optimization results for different types of problems. Among many, *stochastic gradient descent (SGD)* is a very popular method for modern learning systems, which only use one or a few randomly-selected training samples to update the model parameters in each iteration (Bottou, 1998). In comparison to the batch GD algorithms, SGD requires far less computing resources especially when it deals with a huge task involving big data. In (LeCun and Bottou, 2004), it has been proved that SGD can process asymptotically more training samples than batch GD algorithms given the same amount of computing resources. (Roux et al., 2012) proposed a new SGD framework that can achieve a linear convergence rate for strongly-convex optimization problems. In (Shamir and Zhang, 2013), the performance of SGD was analyzed on a number of non-smooth convex objective functions and a bound on the expected optimization errors was provided. On the other hand, SGD also has its own drawbacks. The random noise introduced by data sampling leads to noisy gradient estimates, which may slow down the convergence speed and degrage the performance (Murata and Amari, 1999). Moreover, because of its sequential nature, SGD is hard to parallelize. More recently, several different methods have been proposed to parallelize SGD to accelerate its training speed for big-data applications. Initially, some researchers have proposed to implement the SGD method across multiple computing nodes that are synchronized for updating model parameters. Unfortunately, it has been found that the delay by the required server synchronization is always much longer than the time needed to calculate the gradient. Therefore, several other methods have been proposed to parallelize SGD without frequent synchronization among computing nodes. For example, Zinkevich et al. (2009) has presented a parallelized SGD algorithm, which dispatches all training samples into several computing nodes to update the parameters independently, and the final models will be combined by averaging all separate models at the end of each training epoch. Moreover, Bockermann and Lee (2012) have proved that the performance of this parallelized SGD algorithm depends on the number of SGD runs, and they have successfully used it to train large-scale support vector machines. However, the convergence of this simple parallelized SGD method requires that the learning process is convex. Moreover, Agarwal and Duchi (2012) have shown that the delays introduced by asynchronous model updates can be asymptotically neglected when we optimize a smooth and convex problem. Similarly, Feng et al. (2011) has proposed a parallelized SGD framework called *HOGWILD!*, which has mostly removed the memory locking and synchronization. However, it has found that this method works well only for sparse models. In summary, these parallelized SGD methods heavily rely on the assumptions that the learning problems are convex and/or sparse. These methods may suf-

fer the performance degradation when dealing with more general non-convex optimization problems such as those in deep learning.

Recently, deep learning (Bengio, 2009) has achieved huge successes in many real-world applications, such as speech recognition and computer vision. It becomes a very interesting problem to learn large-scale deeply-structured neural networks, such as *deep neural networks (DNNs)*, from big data sets. We know that the training of DNNs is highly non-convex. Moreover, it is relatively expensive to compute the gradients of the objective function for DNNs since it needs to run the time-consuming back-propagation algorithm. To accelerate the large scale DNN training for big data, it has proposed a weight sharing method in (LeCun et al., 1989), which reduces the number of free parameters in the neural network and thus speeds up the training procedure. Even though today's development of computing hardware makes it possible to train large DNNs directly, it is still very slow to train state-of-the-art DNNs for many real-world applications since the major training of DNNs still depends on the mini-batch SGD algorithm. Therefore, it is much needed in deep learning to develop new optimization methods that are faster to solve large-scale training problems in deep learning. One idea is 'starting small' (Elman, 1993), in which the network training will begin from simple training data with small working memory, and gradually increase the data complexity and network memory. This process simulates the learning procedure of human beings, especially in some complex domains like language. Krueger et al. have implemented the so-called 'shaping' learning in the neural network training (Krueger and Dayan, 2009). During the training, the task is split into several sub-components and a suitable training sequence is used to boost the training speed. In (Bengio et al., 2009), Bengio et al. have proposed a training strategy for deep learning called curriculum learning. The basic idea is to start the learning process from small tasks that are easy to solve, and gradually increase the complexity of the tasks in the later learning stage. Experimental results imply that when using a suitable curriculum, this training strategy may provide a similar performance as unsupervised pre-training and it helps the algorithm to find a better local minimum. The curriculum learning method can serve as an important basis for the work in this paper.

In this paper, we propose a new algorithm called *annealed gradient descent (AGD)*. Instead of directly optimizing the original non-convex objective function, the basic idea of AGD is to optimize a low resolution approximation function that may be smoother and easier to optimize. Furthermore, the approximation resolution is gradually improved according to an annealing schedule over the optimization course. In this work, we have proposed to approximate a non-convex objective function based on some pre-trained codebooks, where the approximation precision can be easily controlled by choosing different number of codewords in the codebook. In comparison with (Bengio et al., 2009), the main contribution of this paper is that AGD provides a suitable way for approximation (through pre-trained codebooks), and more importantly, we show a bound for the difference between the parameters derived by AGD and the regular GD algorithms. This new method has several advantages: Firstly, the low resolution approximation by codebooks lead to a much smoother risk function, which may result in finding a good local minimum more easily. Secondly, because the size of each codebook is much smaller than that of the training set, we can use a fast batch algorithm to learn the model at the beginning and this part can be easily parallelized. In this work, we have applied AGD to training DNNs for various tasks to verify its efficiency and effectiveness. Experiments have shown that the AGD algorithm yields about 40% speed-up in total training time of DNNs, and also leads to similar recognition performance as the regular mini-batch SGD.

The remainder of this paper is organized as follows. In section 2, we provide some background information about empirical risk function and two kinds of gradient descent algorithm. Section 3 shows the mosaic risk function and some related theoretical analysis. In section 4, we present the proposed AGD algorithm. Section 5 reports experiments on different tasks, and we conclude this paper in section 6.

2 PRELIMINARIES

In this section, we first review some preliminary definitions in machine learning, which serve as important notation bases for this work.

2.1 EMPIRICAL RISK FUNCTION

In machine learning, we normally use a *loss function*, $Q(x, y, \theta)$, to measure the 'cost' of a given event x (y is the corresponding label of x) and the underlying model parameters are denoted as θ, and the expected value of the loss function is the so-called *expected risk function*, $R(\theta)$:

$$R(\theta) = \mathbf{E}[Q(x, y, \theta)] \triangleq \int Q(x, y, \theta) dP(x, y) \qquad (1)$$

where $P(x, y)$ denotes the ground truth distribution over all possible events. The fundamental goal of many machine learning problems is to minimize the above expected risk function. In practice, however, it is extremely hard to do so because $P(x, y)$ is always unknown. Therefore, In practice, we normally use a finite training set that includes N independent pairs of sample $O_N = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$, which are presumably randomly sampled from the above unknown distribution. Based on the training set, we may derive the so-called *empirical risk function*, $R_N(\theta)$, to approximate the expected

risk function in eq.(1):

$$R(\theta) \approx R_N(\theta) = \frac{1}{N}\sum_{n=1}^{N} Q(x_n, y_n, \theta) \tag{2}$$

If the training set is sufficiently large, under some minor conditions, minimizing the empirical risk function in eq.(2) may also minimize the expected risk function in eq.(1) (Vapnik, 1998). For notational clarity, without confusion, we drop label y_n from the loss function for the rest of this paper.

2.2 GRADIENT DESCENT ALGORITHM

To minimize the empirical risk function, we can use gradient descent algorithms, which update θ along the direction of the negative gradient based on a pre-defined learning rate λ. Generally speaking, there are two different types of gradient descent algorithms: batch gradient descent (batch GD) and stochastic gradient descent (SGD).

In each iteration, the batch GD considers all of the training samples to calculate the average gradient and then update the parameters accordingly:

$$\begin{aligned} \hat{\theta}_{t+1} &= \hat{\theta}_t - \lambda_t \cdot \nabla_\theta R_N(\hat{\theta}_t) \\ &= \hat{\theta}_t - \lambda_t \cdot \frac{1}{N}\sum_{n=1}^{N} \frac{\partial Q(x_n, \hat{\theta}_t)}{\partial \theta} \end{aligned} \tag{3}$$

where λ_t is the learning rate at iteration t. In contrast, SGD only takes one training sample (which is randomly sampled from the training set) into account in each iteration:

$$\bar{\theta}_{t+1} = \bar{\theta}_t - \lambda_t \cdot \frac{\partial Q(x_n, \bar{\theta}_t)}{\partial \theta}. \tag{4}$$

If we set a suitable learning rate, under some conditions, both batch GD and SGD can finally converge to a local minimum θ^* of the empirical risk function (Bottou, 2004). In practice, to reduce variance of the estimated gradients in SGD, a variant SGD, called mini-batch SGD, is normally used, where a small set (called mini-batch) of randomly selected data samples are used to estimate the gradient for each model update, as opposed to only one sample in SGD.

As we know, the batch GD works well for convex optimization while SGD may be used to solve non-convex optimization problems due to the random noises in its gradient estimation. Meanwhile, SGD requires far less computing resources in comparison to batch algorithms, but on the downside, its convergence speed is very slow due to sampling noise, and it is very hard to parallelize SGD. Therefore, when dealing with some large scale tasks, SGD may run very slowly. In this paper, we propose a new optimization method to solve some large-scale non-convex optimization problems in deep learning. The new method will be compared with SGD in terms of convergence speed and learning performance.

3 THE MOSAIC RISK FUNCTION

Some previous work has considered the problem of critical points (including local optima and saddle points) in non-convex optimizations. According to (Choromanska et al., 2014), some poor local minima may hinder the optimization process especially in small-scale neural networks. (Dauphin et al., 2014) argued that for the practical high dimensional problems, the saddle points may become the most difficult problem to deal with, rather than local optima. In practice, any local search algorithms may be easily trapped into a nearby shallow local optimum point or saddle point, which makes it hard for optimization to proceed further. SGD relies on the sampling noise to alleviate this problem. Another way to tackle this problem is to optimize a smoother approximation of the original rugged non-convex function. In this work, we propose to approximate the original objective function based on a relatively small codebook, which is generated by clustering the whole training set. In this way, we may provide a low resolution approximation of the objective function, which is much smoother and easier to optimize with simple local search algorithms.

Assume we use a discrete codebook, denoted as $C = \{c_1, c_2, ..., c_M\}$, where $M \ll N$, to approximate the original training set. For a training sample x_n, we select its nearest codeword in C as its approximation:

$$c^{(n)} = \arg\min_{c_m \in C} \parallel x_n - c_m \parallel \tag{5}$$

and the quantization error ϵ_n is $\parallel x_n - c^{(n)} \parallel$.

Next, we may derive a low resolution risk function $\tilde{R}_\epsilon$, called *mosaic risk function*, to approximate the empirical risk function $R_N(\theta)$ (where $\epsilon \equiv \max_n \epsilon_n$):

$$\tilde{R}_\epsilon(\theta) = \frac{1}{N}\sum_{n=1}^{N} Q(c^{(n)}, \theta) = \sum_{m=1}^{M} \frac{\omega_m}{N} \cdot Q(c_m, \theta) \tag{6}$$

where ω_m is the number of training samples in the whole training set that are approximated by the codeword c_m, i.e., $\omega_m = \sum_{n=1}^{N} \delta(c^{(n)} - c_m)$, where $\delta()$ denotes the Kronecker delta function.

Assume that the loss function $Q(x, \theta)$ is twice Lipschitz-continuous with respect to the input sample x and the model parameter θ, that is,

$$\parallel Q(x_i, \theta) - Q(x_j, \theta) \parallel < L_0 \parallel x_i - x_j \parallel \tag{7}$$

$$\parallel Q'(x_i, \theta) - Q'(x_j, \theta) \parallel < L_1 \parallel x_i - x_j \parallel \tag{8}$$

$$\parallel Q(x, \theta_i) - Q(x, \theta_j) \parallel < L_0 \parallel \theta_i - \theta_j \parallel \tag{9}$$

$$\parallel Q'(x, \theta_i) - Q'(x, \theta_j) \parallel < L_1 \parallel \theta_i - \theta_j \parallel \tag{10}$$

In this case, it is easy to show that for any θ, the mosaic risk function can provide a bounded approximation for the

empirical risk function:

$$\| R_N(\theta) - \tilde{R}_\epsilon(\theta) \| < \epsilon \cdot L_0 \ \ (\forall \theta). \tag{11}$$

When we deal with a non-convex loss function, the mosaic risk function will give a very important benefit due to its low resolution: because we use a smaller codebook to approximate the training set, and one codeword may represent a large number of different training samples, the mosaic risk function normally corresponds to a smoother curve that may get rid of a lot of critical points comparing with the original empirical risk function. (Bengio et al. (2009) may support this argument.) Therefore, if we use the gradient descent method to optimize the mosaic risk function, named as *mosaic gradient descent (MGD)*, we can find its local minimum much easier and much faster, and this local minimum on mosaic risk function is a good initialization for further learning. If we can use a batch algorithm to optimize the mosaic risk function, it may significantly speed up the training phase due to a smaller number of codewords.

When we use MGD to minimize the mosaic risk function, we can get the following parameter update sequence:

$$\tilde{\theta}_{t+1} = \tilde{\theta}_t - \lambda_t \cdot \nabla_\theta \tilde{R}_\epsilon = \tilde{\theta}_t - \lambda_t \sum_{m=1}^{M} \frac{\omega_m}{N} \cdot \frac{\partial Q(c_m, \tilde{\theta}_t)}{\partial \theta} \tag{12}$$

Obviously, MGD generates a different sequence of the model parameters $\tilde{\theta}_t$.

Moreover, we may extend the above MGD to a stochastic version using only a random mini-batch of data for each model update in eq.(12) rather than the whole training set. All data in the selected mini-batch are approximated by codewords as in eq.(5). This is called mini-batch MGD. Of course, it may be better to use a much larger batch size in mini-batch MGD than that of mini-batch SGD to explore the overall structure of the mosaic function.

In the following, we will show that under some minor conditions, minimization of the mosaic risk function leads to convergence into a bounded neighborhood of a local optimum of the empirical risk function. Moreover, we also show that MGD may provide faster a convergence rate than GD and SGD under certain conditions.

3.1 CONVERGENCE ANALYSIS

As we know, if the learning rates satisfy some minor conditions, the batch GD algorithm in eq. (3) is guaranteed to converge to a critical point of the empirical risk function. In the following, let's first compare the MGD update sequence in eq.(12) with the GD update in eq.(3). Obviously, we have the following lemma:

Lemma 1 (MGD vs. GD) *Assume that the two update sequences in eq.* (3) *and eq.* (12) *start from the same initial parameters θ_0, and use the same sequence of learning rates λ_t, then we have:*

$$\| \tilde{\theta}_t - \hat{\theta}_t \| < \epsilon \cdot L_1 \cdot \sum_{\tau=1}^{t-1} \lambda_\tau \tag{13}$$

Proof: *(1) At $t = 1$, assume that GD and MGD start from the same initialization θ_0 and share the same sequence of learning rate. Based on the Lipschitz-continuous condition in eq.* (8)*, it is easy to show:*

$$\begin{aligned}
&\| \tilde{\theta}_1 - \hat{\theta}_1 \| = \frac{\lambda_0}{N} \| \sum_{n=1}^{N} \frac{\partial Q(x_n, \theta_0)}{\partial \theta_0} - \sum_{n=1}^{N} \frac{\partial Q(c^{(n)}, \theta_0)}{\partial \theta_0} \| \\
&\leq \frac{\lambda_0}{N} \cdot \sum_{n=1}^{N} \| \frac{\partial Q(x_n, \theta_0)}{\partial \theta_0} - \frac{\partial Q(c^{(n)}, \theta_0)}{\partial \theta_0} \| \\
&\leq \frac{\lambda_0}{N} \cdot L_1 \cdot \sum_{n=1}^{N} \| x_n - c^{(n)} \| \\
&\leq \epsilon \cdot L_1 \cdot \lambda_0
\end{aligned} \tag{14}$$

(2) Assume that the Lemma 1 holds for t, i.e.,

$$\| \tilde{\theta}_t - \hat{\theta}_t \| < \epsilon \cdot L_1 \cdot \sum_{\tau=1}^{t-1} \lambda_\tau. \tag{15}$$

For $t + 1$, considering the condition in eq. (10)*, we have:*

$$\begin{aligned}
&\| \tilde{\theta}_{t+1} - \hat{\theta}_{t+1} \| \\
&= \| (\tilde{\theta}_t - \hat{\theta}_t) + \frac{\lambda_t}{N} \sum_{n=1}^{N} (\frac{\partial Q(x_n, \hat{\theta}_t)}{\partial \theta} - \frac{\partial Q(c^{(n)}, \tilde{\theta}_t)}{\partial \theta}) \| \\
&\leq \| \tilde{\theta}_t - \hat{\theta}_t \| + \frac{\lambda_t}{N} \sum_{n=1}^{N} \| \frac{\partial Q(x_n, \hat{\theta}_t)}{\partial \theta} - \frac{\partial Q(c^{(n)}, \tilde{\theta}_t)}{\partial \theta} \| \\
&\leq \epsilon \cdot L_1 \cdot \sum_{\tau=1}^{t-1} \lambda_\tau + \lambda_t \cdot L_1 \cdot \| x_n - c^{(n)} \| \\
&= \epsilon \cdot L_1 \cdot \sum_{\tau=1}^{t} \lambda_\tau
\end{aligned} \tag{16}$$

Therefore, Lemma 1 also holds for $t + 1$. ■

Lemma 1 means that if we run both MGD and the batch GD algorithm for t iterations, the difference between two resultant model parameters is bounded and it is proportional to the maximum quantization error, ϵ, in the mosaic function.

Based on Lemma 1, we have the following theorem:

Theorem 2 (MGD vs. GD) *When we use the empirical risk function eq.* (2) *to measure the two parameters $\tilde{\theta}_t$ in eq.* (12) *and $\hat{\theta}_t$ in eq.*(3)*, the difference is also bounded as:*

$$\| R_N(\tilde{\theta}_t) - R_N(\hat{\theta}_t) \| \leq \epsilon \cdot L_0 \cdot L_1 \cdot \sum_{\tau=1}^{t} \lambda_\tau \tag{17}$$

Proof: *Based on the condition in eq.*(7) *we have:*

$$\begin{aligned}
&\| R_N(\tilde{\theta}_t) - R_N(\hat{\theta}_t) \| = \\
&\frac{1}{N} \cdot \| \sum_{n=1}^{N} (Q(x_n, \tilde{\theta}_t) - Q(x_n, \hat{\theta}_t)) \| \\
&\leq \frac{1}{N} \cdot \sum_{n=1}^{N} \| Q(x_n, \tilde{\theta}_t) - Q(x_n, \hat{\theta}_t) \| \qquad (18) \\
&\leq L_0 \cdot \| \tilde{\theta}_t - \hat{\theta}_t \| \\
&\leq L_0 \cdot \epsilon \cdot L_1 \cdot \sum_{\tau=1}^{t} \lambda_\tau. \quad \blacksquare
\end{aligned}$$

In Lemma 1 and Theorem 2, the bounds are proportional to the summation of all used learning rates. However, in many deep learning practices such as DNN/CNN training, we need to use a sequence of quickly-decayed learning rates to guarantee the convergence. In these situations, the summation of all learning rates is clearly bounded. Therefore, Theorem 2 shows that model parameters $\tilde{\theta}_t$ derived by MGD provide a good estimation of $\hat{\theta}_t$ learned by the regular batch GD algorithm when they are measured with the empirical risk function. The difference is bounded by a quantity proportional to the quantization error in the mosaic function. As a result, if the quantization error is sufficiently small, MGD converges into a bounded neighborhood of a critical point of the original empirical risk function.

3.2 FASTER LEARNING

Here we study the learning speed of MGD. If we want to optimize the empirical risk function $R_N(\theta)$ up to a given precision ρ, i.e., $\| \theta - \theta^* \| < \rho$, by using batch gradient descent in eq. (3), it will take $O(\log \frac{1}{\rho})$ iterations, and the complexity of each iteration is $O(N)$. Thus the overall complexity of the batch algorithm is $O(N \log(\frac{1}{\rho}))$ (Bottou, 2012).

Alternatively, we can run the MGD algorithm on eq. (12) for t iterations, and based on Lemma 1 we have:

Lemma 3 *If we run the MGD algorithm in eq.* (12) *for* t *iterations, the model parameters can reach the precision as:*

$$\| \tilde{\theta}_t - \theta^* \| < \rho + \epsilon \cdot L_1 \cdot \sum_{\tau=1}^{t} \lambda_\tau \qquad (19)$$

and the overall computational complexity of MGD is $O(M \cdot t)$.

Based on Lemma 3, we can have Theorem 4 as below:

Theorem 4 (MGD vs. GD) *Assume there exists a codebook* C *containing* M *codewords, which can approximate the whole training set well enough, and* M *is sufficiently small, i.e.*

$$\epsilon \ll \frac{\rho}{L_1 \cdot \sum_{\tau=1}^{t} \lambda_\tau} \quad \textit{and} \quad M < \frac{N \cdot O(\log(\frac{1}{\rho}))}{t} \qquad (20)$$

then to reach the same optimization precision, optimizing the mosaic risk function using MGD requires less computing resources and yields faster convergence speed than the batch GD in eq. (3)*.*

Similar to Theorem 4, we also have Theorem 5 that compares the resource requirement between MGD and SGD:

Theorem 5 (MGD vs. SGD) *In SGD, we need to run* $O(\frac{1}{\rho})$ *iterations to achieve the optimization precision* ρ *(Bottou, 1998). Similar to Theorem 4, if we find a codebook which satisfies the quantization error requirement and remains sufficiently small as follows:*

$$\epsilon \ll \frac{\rho}{L_1 \cdot \sum_{\tau=1}^{t} \lambda_\tau} \quad \textit{and} \quad M < \frac{1}{t} \cdot O(\frac{1}{\rho}) \qquad (21)$$

then MGD will require less computation resource than SGD to achieve the optimization precision ρ*.*

In the case of big data, i.e., N is extremely large ($N \gg M$), or in an early stage of optimization, when we only require a rough optimization precision, i.e., ρ is allowed to be relatively large, such codebook may exist. In these cases, it may be beneficial to run MGD instead of GD or SGD since MGD has faster convergence speed than batch GD and SGD. Moreover, as opposed to pure serial computation in SGD, the gradient computation in each MGD iteration can be easily parallelized. Therefore, MGD may provide an even faster training speed if multiple computing units are available.

4 ANNEALED GRADIENT DESCENT

Theorems 4 and 5 imply that MGD may possibly converge faster than either GD or SGD but it remains unclear how to find a codebook that simultaneously satisfy both conditions in these theorems. Moreover, we may require different levels of optimization precision in various stages of a training process. For example, at the beginning, when all model parameters stay far away from any local optimal point, we may not need to calculate a very accurate gradient, i.e., ρ is allowed to be relatively large at this time. On the other hand, as the parameters move towards a close neighborhood of an optimal point, we may require a very small ρ to perform an accurate local search to converge more effectively. As suggested by eq.(20), the required quantization error, ϵ, is proportionally related to ρ. For a fixed set of training data, the quantization error, ϵ, in turn depends on the size of the codebook, M. This suggests we use an annealing schedule of $\{\epsilon_1, \epsilon_2, \cdots\}$ (with $\epsilon_{i+1} < \epsilon_i$) for

the whole training process, where ϵ gradually decreases as training continues. At the beginning, we can use a small low resolution codebook (relatively big ϵ) to run MGD to learn model parameters. As training proceeds, we gradually reduce ϵ by using increasingly larger codebooks. At the final stage, we may even use all original training samples to fine-tune the model parameters.

Therefore, the basic idea of annealed gradient descent (AGD) is to construct deeply-structured hierarchical codebooks, in which quantization error ϵ slowly decreases from the top layer down to the bottom layer, and the last layer is finally connected to the original training set. During the training procedure, we first start from the top to use each layer of codebooks to do MGD updates in eq.(12) and gradually move down the hierarchy based on a pre-specified annealing schedule until we finally use the training samples to fine-tune the model parameters. If a proper annealing schedule is used, this annealed learning process may accelerate the training speed as implied by Theorems 4 and 5. More importantly, it may help to converge to a better local optimum at the end because AGD optimizes much smoother mosaic objective functions from the early stage of training.

In this section, we first briefly discuss the hierarchical codebooks, and then present the AGD training algorithm.

4.1 HIERARCHICAL CODEBOOKS

In this work, we use a regular K-means based top-down hierarchical clustering algorithm (Zhou et al., 2015) to construct the required hierarchical codebooks, where the centroid of each cluster is used as a codeword for each layer. The structure of the hierarchical codebooks is shown in Figure 1.

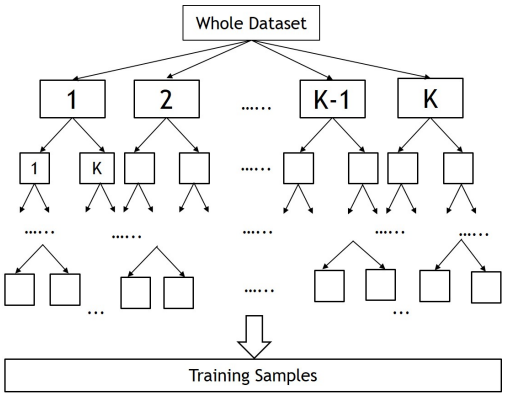

Figure 1: Illustration of a hierarchical codebook for AGD

When building the codebooks, we first divide the training set into several subsets based on the class labels of data samples (each subset only contains all training samples from one class label), then conduct the hierarchical top-down K-means clustering on each subset. Note that the clustering process is very easy to parallelize because all subsets are independent with each other and we can run hierarchical K-means on them separately. In the K-means clustering, we first define a suitable K, then use K-means to split each subset into K clusters. The centroids of all clusters are used to build the first layer of codebooks. Next, we continuously apply the K-means clustering on all clusters to further divide them into K sub-clusters to derive the codebooks in the next layer. We repeat this procedure several times until the quantization error in the last layer becomes small enough. Finally, we connect the original training samples at the bottom as the leaf nodes to obtain a hierarchical codebook as shown in Figure 1, in which the sizes of the codebooks are gradually increased from the top layer to the bottom layer. In this way, the K-means codewords in the various layers of the hierarchy may be used to approximate each training sample in the leaf node up to different precision levels as required in the following AGD algorithm.

4.2 ANNEALED GRADIENT DESCENT ALGORITHM

In AGD, we first specify an annealing schedule, i.e., $\{\epsilon_1, \epsilon_2, \cdots\}$ ($\epsilon_{i+1} < \epsilon_i$). In each epoch of AGD, for each training sample in the selected data mini-batch, we select a codeword from the uppermost layer of the hierarchical codebooks that barely satisfies the required quantization error ϵ_i, to construct the mosaic function for MGD at this stage. Since ϵ_i gradually decreases in the annealing schedule, we slowly move to use more and more precise codewords (eventually the original data samples) in AGD. In AGD, a hierarchical search list is constructed for each layer in C based on the average quantization errors in K-means. As a result, in each AGD step, the corresponding codewords can be found very efficiently based on this search list. The annealed gradient descent (AGD) algorithm is shown as in Algorithm 1. During the AGD training, we may even use varying batch sizes. For example, we can start from a very large batch size (even the whole training set) at the beginning and slowly decrease the batch size from epoch to epoch. In general, we normally use much larger batch sizes than those used in the regular mini-batch SGD. If a suitable annealing schedule is specified, many initial AGD epochs may be designated to run faster MGD with smaller codebooks, yielding faster training speed than the mini-batch SGD or GD in overall.

5 EXPERIMENTS

In this section, we apply the proposed AGD algorithm to learning sigmoid fully connected deep neural networks (DNNs) for image recognition in the MNIST database and

Algorithm 1 Annealed Gradient Descent(AGD)

Input: training set O, hierarchical codebook C, annealing schedule $\{\epsilon_1, \epsilon_2, ..., \epsilon_T \mid \epsilon_{i+1} < \epsilon_i\}$
for each epoch $i = 1$ **to** T **do**
  for each batch X **do**
    For each sample in X, select a codeword $c^{(n)}$ at the uppermost layer of the C satisfying ϵ_i;
    Use MGD to optimize the mosaic risk function;
  end for
end for

Table 1: The total training time of K-means clustering.

Database	Number of CPUs	Training Time
MNIST	5	4.3 *(hr)*
Switchboard	8	9.2 *(hr)*

speech recognition in the Switchboard database. AGD and the regular mini-batch SGD are both used to train DNNs based on the minimum cross-entropy error criterion. AGD is compared with mini-batch SGD in terms of the total training time and the final recognition performance.

For each data set, we first use a standard K-means algorithm to build a deeply-structured hierarchical codebook. We use $K = 5$ in MNIST and $K = 4$ in Switchboard, which can result in a hierarchical codebook with sufficient depth. In our experiments, the MNIST database contains 10 classes and Switchboard contains 8991 classes, thus the hierarchical K-means has a good potential for parallel training. Here, the K-means clustering procedure is parallelized among multiple CPUs to speed it up as much as possible. In our experiments, the total running time of the K-means clustering is about 4.3 hours in MNIST (using 5 CPUs) and about 9.2 hours in Switchboard (using 8 CPUs), as shown in Table 1. If we use more CPUs, the K-means running time can be further reduced. As shown later, the K-means training time is not significant when comparing with the necessary DNN training times. And if we use more CPUs to do K-means, the clustering time can be further reduced (approximately less than 3 hours for MNIST and less then 5 hours for Switchboard). Moreover, for each database we only need to run K-means once and after that we can use the same codebook to train DNNs based on different annealing schedules. Therefore, we do not take into account the running time of the K-means clustering in the following comparisons. Note that no pre-training is used for DNNs in our experiments.

5.1 MNIST: IMAGE RECOGNITION

The MNIST database (LeCun et al., 1998) contains 60,000 training images and 10,000 test images, and the images are 28-by-28 in size. Here, we use data augmentation through image deformation at the beginning of each epoch to enlarge the training set as in (Ciresan et al., 2010). We use the configuration of 3-hidden-layer DNNs (1500, 1000, 500 nodes in each hidden layer) in (Ciresan et al., 2010) as our network structures and use SGD and AGD to do network training. Following (Ciresan et al., 2010), we fine-tune all hyper-parameters towards the best possible performance. In SGD, we use a mini-batch of 10 samples and an initial learning rate of 0.4. Notice that in MNIST experiments we do not use momentum during the training phase. In MGD, we use a larger mini-batch size of 4500 and an initial learning rate of 0.8. The training process runs for 550 epochs to guarantee the convergence of the augmented MNIST database.

As we know, we should shrink the learning rates during the training process for better convergence. Specifically, when the training mean square error (MSE) becomes smaller than a pre-defined threshold r, we use the formula

$$\lambda_{t+1} = \lambda_t \cdot \frac{p}{p+t}$$

to gradually decrease the learning rate, where p is a pre-defined constant to control the decreasing speed of the learning rates. In our experiments, we set $r = 0.17$ and $p = 10000$. As for the AGD annealing schedule in MNIST, we start from $\epsilon_1 = 7.5$ (this value is based on the average quantization error in the first layer of the codebook) and $\epsilon_{i+1} = 0.999 \cdot \epsilon_i$.

During the annealing phase we use MGD to train the network while in the regular phase we use SGD with the same configurations as the baseline. Note that we only do image deformation during the regular phase. In Figure 2, we have shown the learning curves of both SGD and AGD in terms of cross-entropy and classification error rate on the MNIST training set. Since each MGD epoch runs much faster than a SGD epoch, all learning curves are plotted as a function of total training time instead of epochs.

From the two pictures in Figure 2 we can see that AGD (in blue) finishes the same number of epochs much earlier than SGD (in red). In addition, in Table 2, we also give the total training time and the best classification error on the test set for both AGD and SGD. From results in Figure 2 and Table 2, we can see that the proposed AGD training algorithm yields slightly better classification performance in the test set, and more importantly reduces the total DNN training time by about 40%.

5.2 SWITCHBOARD: SPEECH RECOGNITION

Switchboard is a 320-hour English transcription task, which contains 332,576 utterances in its training set (amounting to about 126 millions of training samples in total). We select the standard NIST Hub5e2000 as the test

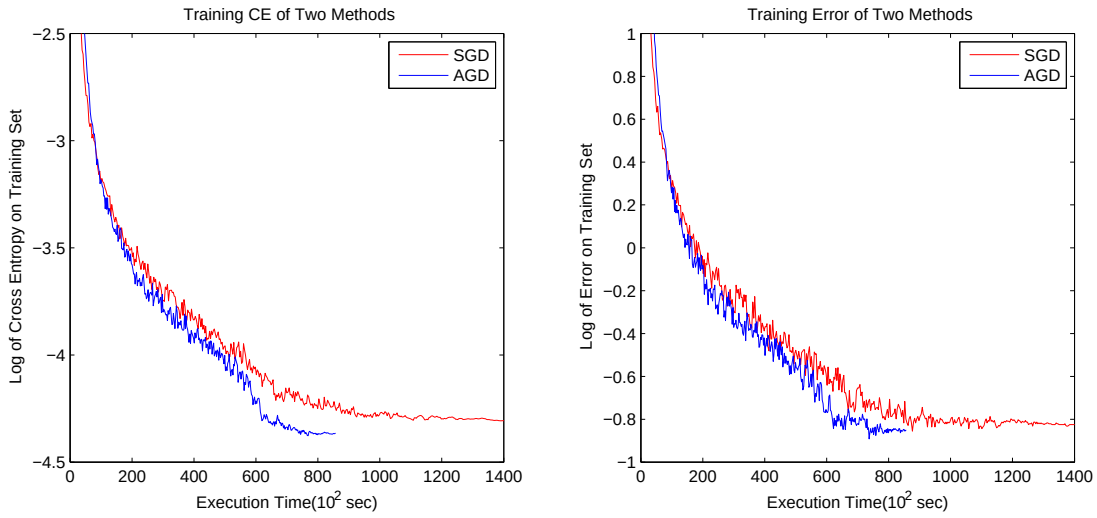

Figure 2: Learning curves on the MNIST training set (Left: cross entropy error; Right: classification error) of SGD and AGD as a function of elapsed training time.

Table 2: Comparison between SGD and AGD in terms of total training time (using one GPU) and the best classification error rate on MNIST.

Method	**Training Time**	**Test Error**
SGD	38.8 *(hr)*	0.47%
AGD	23.6 *(hr)*	0.46%

set in this work, which has 1831 utterances.[1] Following (Seide et al., 2011; Bao et al., 2013; Pan et al., 2012), we train a 6-hidden-layer DNN with 2048 nodes per layer based on the minimum cross-entropy criterion. We compare the cross entropy and frame classification errors on the training and test sets to evaluate the performance of SGD and AGD, meanwhile we also evaluate word error rates in speech recognition for the test set.

Here we use similar hyper-parameters as in (Pan et al., 2012; Xue et al., 2014). For example, we use a mini-batch of 1024 samples and an initial learning rate of 0.2 in SGD, and a mini-batch of 6144 and an initial learning rate of 1.0 in MGD. We use 0.9 as the momentum in both SGD and MGD. We run 10 epochs in SGD and 17 epochs in AGD. During the training process, we need to shrink the learning rates slowly. Specifically, we multiply the learning rate by 0.8 every epoch after the 5-th epoch in SGD and the 12-th epoch in AGD. Note that our SGD baseline is solid and comparable with the best results reported on this task (Seide et al., 2011; Hinton et al., 2012; Xue et al., 2014) in terms of both training speed and recognition performance.

As for the AGD annealing schedule in Switchboard, unlike MNIST, it starts from an initial value ϵ_1 (17.5 in this case), and use the formula

$$\epsilon_{i+1} = \epsilon_i - \Delta_\epsilon$$

to reduce ϵ_i by subtracting a constant value every epoch until it reaches the pre-defined value (8.5 in this case). The reason for this formula is that we run much less epochs here. In Switchboard, we have evaluated three different annealing schedules to show how they affect the training speed and the final recognition performance as shown in Table 3. In these three schedules, we use different values for Δ_ϵ, e.g. 1.0, 0.9 and 0.8 respectively. We can see that the annealing schedule 1 ($\Delta_\epsilon = 1.0$) decreases fastest and it provides the best performance but relatively slower training speed. In contrast, schedule 3 ($\Delta_\epsilon = 0.8$) gives the fastest training speed but slightly worse performance because more epochs will be dispatched to run MGD.

Figure 3 shows the learning curves of both SGD and AGD (using the annealing schedule with $\Delta_\epsilon = 1$) in terms of cross-entropy and frame errors on the Switchboard training set as a function of elapsed training time. Clearly, AGD runs much faster than SGD on Switchboard as well. Meanwhile, AGD can also achieve a slightly better local minimum than SGD as measured in both figures.

In Table 3, we give the total training times and the word error rates in speech recognition. We report the experimental results for all 3 different annealing schedules. The results have shown that the proposed AGD method can yield similar recognition performance with much faster training

[1]Due to the copyright issue, all Switchboard experiments were conducted at NELSLIP, University of Science and Technology of China.

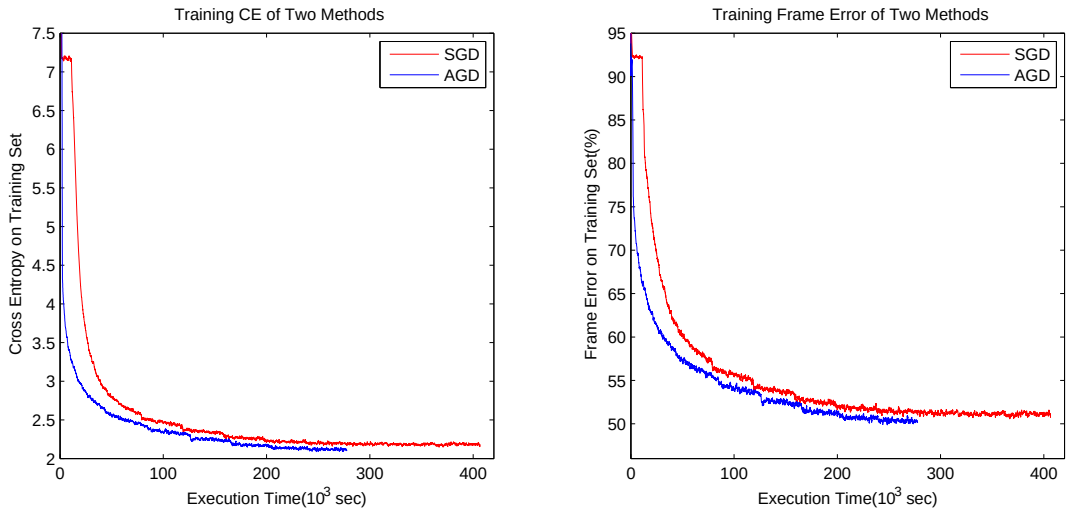

Figure 3: Learning curves on the Switchboard training set (Left: cross entropy error; Right: frame classification error) of SGD and AGD as a function of elapsed training time.

Table 3: Comparison between SGD and AGD in terms of total training time (using one GPU) and word error rate in speech recognition on Switchboard.

Method	**Training Time**	**Word Error**
SGD	114.05 *(hr)*	16.4%
AGD ($\Delta_\epsilon = 1.0$)	78.79 *(hr)*	16.3%
AGD ($\Delta_\epsilon = 0.9$)	66.63 *(hr)*	16.7%
AGD ($\Delta_\epsilon = 0.8$)	55.35 *(hr)*	17.5%

speed than SGD (about a 30% to 40% reduction in total training time) when a suitable annealing schedule is used. Results in Table 3 also imply how quantization errors in the codebooks may affect the final classification performance: a slower schedule means more epochs will be dispatched to run MGD, which uses each layer of the codebooks to train the DNNs. In this case, the quantization error of the codebook may bring about some negative influence on the performance but provide faster learning speed. In practice, when we are sensitive to the total training time of DNNs, we may use slower decreasing schedules to accelerate the DNN training dramatically.

6 CONCLUSIONS

In this paper, we have proposed a new annealed gradient descent (AGD) algorithm for non-convex optimization in deep learning, which can converge to a better local minimum with faster speed when compared with the regular mini-batch SGD algorithm. In this work, AGD has been applied to training large scale DNNs for image classification and speech recognition tasks. Experimental results have shown that AGD significantly outperforms SGD in terms of the convergence speed. Therefore, the AGD algorithm is especially suitable for the large scale non-convex optimization problems in deep learning. In the future, we may apply AGD to training convolutional neural networks (CNNs) for other more challenging image recognition tasks, where we may build the hierarchical codebooks by clustering image patches instead of the whole input images. Moreover, AGD may be applied to the recently proposed unsupervised learning algorithm in (Zhang and Jiang, 2015; Zhang et al., 2015) as well.

Acknowledgments

This work was partially supported by an NSERC discovery grant from the Canadian Federal Government. The first author is supported by a scholarship from China Scholarship Council (CSC). We appreciate Dr. Pan Zhou from NELSLIP, University of Science and Technology of China for his help in conducting the Switchboard experiments in this paper.

References

A. Agarwal and J. C. Duchi. Distributed delayed stochastic optimization. In *IEEE Annual Conference on Decision and Control (CDC)*, pages 5451–5452, 2012.

Y. Bao, H. Jiang, L. Dai, and C. Liu. Incoherent training of deep neural networks to de-correlate bottleneck features for speech recognition. In *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 6980–6984, 2013.

Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.

C. Bockermann and S. Lee. Scalable stochastic gradient descent with improved confidence. In *NIPS Workshop on Big Learning–Algorithms, Systems, and Tools for Learning at Scale*, 2012.

L. Bottou. Online learning and stochastic approximations. *On-line learning in neural networks*, 17:9, 1998.

L. Bottou. Stochastic learning. In *Advanced lectures on machine learning*, pages 146–168. Springer, 2004.

L. Bottou. Stochastic gradient tricks. *Neural networks: tricks of the trade. Springer, Berlin*, pages 430–445, 2012.

A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun. The loss surface of multilayer networks. *arXiv preprint arXiv:1412.0233*, 2014.

D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber. Deep big simple neural nets excel on handwritten digit recognition. *Neural Computation*, 22 (12):3207–3220, 2010.

Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems (NIPS'14)*, pages 2933–2941, 2014.

J. L. Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99, 1993.

N. Feng, R. Benjamin, R. Christopher, and J. W. Stephen. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing System 24 (NIPS'11)*, 2011.

G. Hinton, L. Deng, D. Yu, G. Dahl, A.Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modelling in speech recognition. *IEEE Signal Processing Magazine*, 29, 2012.

K. A. Krueger and P. Dayan. Flexible shaping: How learning in small steps helps. *Cognition*, 110(3):380–394, 2009.

Y. LeCun and L Bottou. Large scale online learning. In *Advances in neural information processing systems 17 (NIPS'04)*, page 217, 2004.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

N. Murata and S. Amari. Statistical analysis of learning dynamics. *Signal Processing*, 74(1):3–28, 1999.

J. Pan, C. Liu, Z. Wang, Y. Hu, and H. Jiang. Investigations of deep neural networks for large vocabulary continuous speech recognition: Why DNN surpasses GMMs in acoustic modelling. In *Proc. of International Symposium on Chinese Spoken Language Processing*, 2012.

N. L. Roux, M. Schmidt, and F. R. Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In *Advances in Neural Information Processing Systems 25 (NIPS'12)*, pages 2672–2680, 2012.

F. Seide, G. Li, and D. Yu. Conversational speech transcription using context-dependent deep neural networks. In *Proc. of Interspeech*, pages 437–440, 2011.

O. Shamir and T. Zhang. Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 71–79, 2013.

V. N. Vapnik. *Statistical Learning Theory*. John Wiley and Sons, 1st edition, 1998.

S. Xue, O. Abdel-Hamid, H. Jiang, L. Dai, and Q. Liu. Fast adaptation of deep neural network based on discriminant codes for speech recognition. *IEEE/ACM Trans. on Audio, Speech and Language Processing*, 22(12):1713–1725, 2014.

S. Zhang and H. Jiang. Hybrid orthogonal projection and estimation (hope): A new framework to probe and learn neural networks. In *arXiv:1502.00702*, 2015.

S. Zhang, L. Dai, and H. Jiang. The new hope way to learn neural networks. In *Proc. of Deep Learning Workshop at ICML 2015*, 2015.

P. Zhou, H. Jiang, L. Dai, Y. Hu, and Q. Liu. State-clustering based multiple deep neural networks modeling approach for speech recognition,. *IEEE/ACM Trans. on Audio, Speech and Language Processing*, 23(4):631–642, 2015.

M. Zinkevich, J. Langford, and E. J. Smola. Slow learners are fast. In *Advances in Neural Information Processing Systems 22 (NIPS'09)*, pages 2331–2339, 2009.