Adaptive Learning Rate via Covariance Matrix Based Preconditioning for Deep Neural Networks

Yasutoshi Ida, Yasuhiro Fujiwara and Sotetsu Iwamura
NTT Software Innovation Center, 3-9-11 Midori-cho Musashino-shi, Tokyo, 180-8585, Japan
ida.yasutoshi@lab.ntt.co.jp, fujiwara.yasuhiro@lab.ntt.co.jp, iwamura.sotetsu@lab.ntt.co.jp

Abstract

Adaptive learning rate algorithms such as RMSProp are widely used for training deep neural networks. RMSProp offers efficient training since it uses first order gradients to approximate Hessian-based preconditioning. However, since the first order gradients include noise caused by stochastic optimization, the approximation may be inaccurate. In this paper, we propose a novel adaptive learning rate algorithm called SDProp. Its key idea is effective handling of the noise by preconditioning based on covariance matrix. For various neural networks, our approach is more efficient and effective than RMSProp and its variant.

1 Introduction

Adaptive learning rate algorithms are widely used for the efficient training of deep neural networks. RMSProp [Tieleman and Hinton, 2012] and its follow-on methods [Zeiler, 2012; Kingma and Ba, 2014] are being used in many deep neural networks such as Convolutional Neural Networks (CNNs) [LeCun *et al.*, 1998] since they can be easily implemented with high memory efficiency.

The empirical success of RMSProp could be explained by using Hessian-based preconditioning [Dauphin *et al.*, 2015]. Hessian is the matrix that represents the curvature of the loss function; Hessian-based preconditioning locally changes the curvature of the loss function. When training deep neural networks, pathological curvatures such as saddle points [Dauphin *et al.*, 2014] and cliffs [Pascanu *et al.*, 2013] can slow the progress of first order gradient descent, such as Stochastic Gradient Descent (SGD) [Robbins and Monro, 1951]. Hessian-based preconditioning improves the condition of the curvature, and thus enhances SGD speed. However, SGD with Hessian-based preconditioning incurs high computation cost because it generally computes the inverse matrix of Hessian. Since RMSProp approximates Hessian-based preconditioning by using first order gradients [Dauphin *et al.*, 2015], it achieves efficient training. In addition, RMSProp is easy to implement. Therefore, in terms of practical use, RMSProp and its variants such as AdaDelta [Zeiler, 2012] and Adam [Kingma and Ba, 2014] are still seen as the most powerful approach to training deep neural networks.

However, the first order gradients used in RMSProp include noise caused by stochastic optimization techniques such as mini-batch setting. With batch setting, since the model inputs are fixed in each iteration, only parameter updates change the gradients. On the other hand, with mini-batch setting, since the inputs are not fixed in each iteration, gradients can also be changed by randomly selecting the inputs in each iteration. This change in the mini-batch setting can be seen as noise. Since RMSProp uses the noisy first order gradients to approximate Hessian-based preconditioning, the approximation may be inaccurate. This indicates that the efficiency of RMSProp can be improved by effectively handling the noise in the first order gradients.

This paper proposes a novel adaptive learning rate algorithm called *SDProp*. The key idea is to use covariance matrix based preconditioning instead of Hessian-based preconditioning. The covariance matrix is derived by assuming a distribution for the noise in the observed gradients. Since the distribution effectively captures the noise, SDProp can effectively capture the changes in gradients caused by random input selection in each iteration. Interestingly, our theoretical analysis reveals that SDProp uses the information of directions over past gradients in adapting the learning rate while RMSProp and its variants use the magnitudes of the gradients. In experiments, we compare SDProp with RMSProp. SDProp needs 50 % fewer training iterations than RMSProp to reach the final training loss for CNN in Cifar-10, Cifar-100 and MNIST datasets. In addition, SDProp outperforms Adam, a state-of-the-art algorithm based on RMSProp, in several datasets. Our approach is also more effective than RMSProp for training Recurrent Neural Network (RNN) [Elman, 1990] and very deep fully-connected neural networks.

2 Preliminary

We briefly review the background of this paper. First, we describe SGD, which is a basic algorithm in stochastic optimization such as mini-batch setting. Second, we review RMSProp. Finally, we explain the relationship between Hessian-based preconditioning and RMSProp.

2.1 Stochastic Gradient Descent

Many learning algorithms aim at minimizing loss function $f(\theta)$ with respect to parameter vector, θ[Fujiwara *et al.*, 2016a; 2016b]. SGD is a popular algorithm in the mini-batch

setting. To minimize $f(\theta)$, SGD iteratively updates θ with a mini-batch of samples as follows:

$$\theta_{i,t} = \theta_{i,t-1} - \alpha \nabla f\left(\theta_{i,t-1}; x_{t-1}\right) \quad (1)$$

where α is the learning rate, $\theta_{i,t}$ is the i-th element of the parameter vector at time t, x_{t-1} is the sample or mini-batch at time $t-1$, and $\nabla f\left(\theta_{i,t-1}; x_{t-1}\right)$ is the first order gradient with respect to the i-th parameter given by x_{t-1}. SGD applies Equation (1) to each sample or mini-batch while Gradient Descent (GD) applies Equation (1) to all data in the batch setting. Although $\nabla f\left(\theta_{i,t-1}; x_{t-1}\right)$ includes noise due to the random selection of mini-batch x_{t-1}, SGD uses it in the training phase. Since SGD only uses a part of the data for computing $\nabla f\left(\theta_{i,t-1}; x_{t-1}\right)$, each iteration has reduced computation cost while memory efficiency is high.

2.2 RMSProp

RMSProp is a popular algorithm based on SGD for training neural networks. AdaDelta and Adam are follow-up methods of RMSProp. RMSProp rapidly reduces loss function $f(\theta)$ by adapting the learning rate of SGD. The updating rule of RMSProp is as follows:

$$v_{i,t} = \beta v_{i,t-1} + (1-\beta)\nabla f\left(\theta_{i,t-1}; x_{t-1}\right)^2 \quad (2)$$

$$\theta_{i,t} = \theta_{i,t-1} - \frac{\alpha}{\sqrt{v_{i,t}+\epsilon}} \nabla f\left(\theta_{i,t-1}; x_{t-1}\right) \quad (3)$$

where $v_{i,t}$ is the moving average of uncentered variance over past first order gradients $\nabla f\left(\theta_{i,t-1}; x_{t-1}\right)$, β is the decay rate for computing $v_{i,t}$, and ϵ is the small value for the stable computation. Intuitively, RMSProp divides the learning rate, α, by magnitude $\sqrt{v_{i,t}}$ of the past first order gradients $\nabla f\left(\theta_{i,t-1}; x_{t-1}\right)$. Therefore, if the i-th parameter has large $\nabla f\left(\theta_{i,t-1}; x_{t-1}\right)$ values in terms of the magnitude in the past, RMSProp yields a small learning rate because $\sqrt{v_{i,t}}$ in Equation (3) is large. Empirically, this idea efficiently reduces the loss function for deep neural networks. Follow-up methods such as AdaDelta and Adam are based on this idea. For the convex optimization, regret analysis can be used to explain the efficiency of the methods [Kingma and Ba, 2014]. For non-convex optimization such as deep neural networks, the empirical success of RMSProp could be explained by using Hessian-based preconditioning. We briefly review the relationship between Hessian-based preconditioning and RMSProp by following [Dauphin *et al.*, 2015] in the next section.

2.3 Hessian-based Preconditioning

Some kind of pathological curvature of the loss function slows the progress of SGD [Dauphin *et al.*, 2014]. Therefore, it is important to capture the curvature in order to efficiently train deep neural networks.

Hessian-based preconditioning locally changes the function by using Hessian H, which can capture the curvature of the function. Hessian is the square matrix of the second order gradients of function $f(\theta)$ represented by $H=\nabla^2 f(\theta)$. The condition number of Hessian estimates the extent to which the curvature is pathological. Condition number is defined as $\sigma_{\max}(H)/\sigma_{\min}(H)$ where $\sigma_{\max}(H)$ and $\sigma_{\min}(H)$ are the largest and smallest singular values of H, respectively. The function has less pathological curvature if the condition number has a small value. This is because the function equally curves if it has small condition number. Therefore, we can increase the efficiency of the training by reducing the Hessian condition number [Dauphin *et al.*, 2015].

Hessian-based preconditioning locally transforms an original parameter into another parameter so that the Hessian has small condition number. Preconditioning matrix D gives transformations such as $\hat{\theta}=D^{1/2}\theta$ where $\hat{\theta}$ is the transformed parameter. By using $\hat{\theta}$, function f is transformed into function $\hat{f}$ where $f(\theta)=f(D^{-1/2}\hat{\theta})=\hat{f}(\hat{\theta})$. If $\hat{f}(\hat{\theta})$ has smaller condition number than $f(\theta)$, we can efficiently train a model by applying first order gradient descent to $\hat{\theta}$. The updating rule of $\hat{\theta}$ is $\hat{\theta}_t=\hat{\theta}_{t-1}-\alpha\nabla\hat{f}(\hat{\theta})$. Since $\nabla\hat{f}(\hat{\theta})=D^{-1/2}\nabla f(\theta)$, we have the following form for original parameter θ:

$$\theta_t = \theta_{t-1} - \alpha D^{-1}\nabla f(\theta_{t-1}). \quad (4)$$

If $\hat{H}$ is the Hessian of transformed function $\hat{f}(\hat{\theta})$, $\hat{H}$ is given as $\hat{H} = (D^{-1/2})^{\mathrm{T}} H D^{-1/2}$. When $D^{1/2} = H^{1/2}$, $\hat{H}$ has a smaller condition number because $\hat{H}$ is an identity matrix. In this case, Equation (4) corresponds the Newton method. However, $H^{1/2}$ exists only when H is positive-semidefinite. Since deep neural networks have many saddle points where Hessian can be indefinite [Dauphin *et al.*, 2014], the Newton method is unsuitable for training deep neural networks. On the other hand, the diagonal equilibration matrix of $D=\sqrt{\mathrm{diag}\left(H^2\right)}$ works well even if H is indefinite [Dauphin *et al.*, 2015]. This indicates that GD can efficiently escape from saddle points by preconditioning based on the diagonal equilibration matrix.

In RMSProp, the role of $\sqrt{v_{i,t}}$ in Equation (3) could be explained by using Hessian-based preconditioning [Dauphin *et al.*, 2015]. A comparison of Equation (4) to Equation (3) indicates that $\sqrt{v_{i,t}}$ corresponds to the i-th element of the diagonal preconditioning matrix. In addition, empirical results suggest that $\sqrt{v_{i,t}}$ approximates the i-th element of the diagonal equilibration matrix which can be used to efficiently train deep neural networks [Dauphin *et al.*, 2015]. Thus, RMSProp can be interpreted as Hessian-based preconditioning using an approximated diagonal equilibration matrix in the mini-bath setting. Therefore, since RMSProp is more efficient in escaping from saddle points than SGD, RMSProp and its follow-up methods achieve high efficiency.

3 Proposed Method

We first introduce the novel preconditioning idea. Then, we derive SDProp based on this idea.

3.1 Idea

RMSProp approximates Hessian-based preconditioning by using the first order gradients $\nabla f\left(\theta_{i,t-1}; x_{t-1}\right)$ as described in the preliminary section. However, in stochastic optimization approaches such as mini-batch setting, the first order gradients $\nabla f\left(\theta_{i,t-1}; x_{t-1}\right)$ include noise because input x_{t-1} is randomly selected in each iteration. Since the first order gradients $\nabla f\left(\theta_{i,t-1}; x_{t-1}\right)$ in Equation (2) and the square roots of the uncentered variances $\sqrt{v_{i,t}}$ in Equation (3) contain noise, it is difficult to effectively approximate Hessian-based preconditioning. In order to effectively handle the noise, we

replace Hessian-based preconditioning with covariance matrix based preconditioning.

In covariance matrix based preconditioning, we assume that the first order gradients $\nabla f\left(\theta_{i,t-1}; x_{t-1}\right)$ follow a Gaussian distribution. This is because the field of probabilistic modeling uses Gaussian distributions to model the noise of observations[Sra *et al.*, 2012; Ida *et al.*, 2013; Fukuda *et al.*, 2014; Miyashita *et al.*, 2013]. By following [Sra *et al.*, 2012], we assume the following Gaussian distribution of first order gradient $\hat{g}_t = \nabla f\left(\theta_{t-1}; x_{t-1}\right) \in \mathcal{R}^d$:

$$\hat{g}_t | \bar{g}_t \sim N(\bar{g}_t, C_t) \tag{5}$$

where $\bar{g}_t \in \mathcal{R}^d$ is the true gradient without the noise while $\hat{g}_t \in \mathcal{R}^d$ includes the noise. $N(\bar{g}_t, C_t)$ is a Gaussian distribution with mean $\bar{g}_t$ and covariance matrix C_t; C_t is the covariance matrix of $\hat{g}_t$ whose size is $d \times d$. The diagonal elements in C_t represent the magnitude of oscillation of the first order gradients $\hat{g}_t$ that include the noise. Specifically, let $C_t[i,j]$ be the i-th row and the j-th column element in C_t, $C_t[i,j]$ represents the covariance of the i-th and the j-th first order gradient. Therefore, if the i-th first order gradient strongly correlates with the j-th first order gradient, $C_t[i,j]$ has large absolute value. On the other hand, $C_t[i,i]$ represents the variance of the i-th first order gradient. Therefore, $C_t[i,i]$ has large value if the first order gradient strongly oscillates in the i-th dimension.

Intuitively, large oscillations in i-th dimension incur high variance of updating directions and inefficient progress in plain SGD. However, it is difficult to reduce the oscillation since it can be a result of the noise induced by the mini-batch setting. How can we reduce the oscillation by using C_t ? This is the motivation behind our approach; plain SGD efficiently progresses if we can control the oscillation by utilizing C_t. In this paper, we propose the preconditioning of C_t to control the oscillation. While Hessian-based preconditioning reduces the condition number of Hessian, our preconditioning reduces the condition number of C_t by transforming C_t into an identity matrix. We describe our approach in the next section.

3.2 Covariance Matrix Based Preconditioning

The previous section suggests that large values in the diagonal of C_t prevent the efficient progress of SGD. Therefore, if we could control the values in the diagonal of C_t, we improve the efficiency of SGD. Our covariance matrix based preconditioning transforms C_t into $\rho^2\, \boldsymbol{I}$ where $\boldsymbol{I}$ is an identity matrix whose size is $d \times d$ and ρ is a hyper-parameter that has a positive value. Since the element in the diagonal of C_t represents the variance of first order gradient, we can hold the variance to constant value ρ^2. If the variance is larger than ρ^2, its value is reduced to ρ^2. Therefore, SGD efficiently progresses if we transform C_t into $\rho^2\, \boldsymbol{I}$.

We first describe the approach used to transform C_t into $\boldsymbol{I}$ instead of $\rho^2\, \boldsymbol{I}$. This is because once C_t is transformed into $\boldsymbol{I}$, it is easy to transform $\boldsymbol{I}$ into $\rho^2\, \boldsymbol{I}$ as we describe later. Hessian-based preconditioning transforms first order gradients to yield $\nabla \hat{f}(\hat{\theta}) = D^{-1/2} \nabla f(\theta)$ where D is a preconditioning matrix. The preconditioning matrix of $D^{1/2} = H^{1/2}$ reduces the condition number of Hessian H as described in the preliminary section. Unlike the previous approach, we execute the preconditioning of C_t and so use the transformation $g_p = D^{-1} \hat{g}_t$. In this transformation, g_p is a transformed first order gradient and $\hat{g}_t$ is a first order gradient as defined in Equation (5). Since the transformation is an affine transformation of $\hat{g}_t$ generated from the Gaussian distribution in Equation (5), we have following distribution of g_p:

$$g_p = D^{-1} \hat{g}_t | \bar{g}_t \sim N(D^{-1} \bar{g}_t, D^{-1} C_t (D^{-1})^{\mathrm{T}}). \tag{6}$$

In Equation (6), we use the following major rule to transform Equation (5) into (6): if $X \sim N(m, \Sigma)$ and $Y = AX$, then $Y \sim N(Am, A\Sigma A^T)$; $N(m, \Sigma)$ is a Gaussian distribution that has mean m and covariance matrix Σ, A is a matrix for affine transformation and Y is a transformed variable. By setting $D = C_t^{1/2}$ in Equation (6), we have the following property :

Theorem 1. *If we transform first order gradient $\hat{g}_t$ to yield $g_p = C_t^{-1/2} \hat{g}_t$, we have the following Gaussian distribution:*

$$g_p | \bar{g}_t \sim N\left(C_t^{-\frac{1}{2}} \bar{g}_t,\ \boldsymbol{I}\right) \tag{7}$$

where $\boldsymbol{I}$ is an identity matrix whose size is $d \times d$.

Proof. *By using eigen decomposition, we can represent C_t as $C_t = U \Sigma U^{\mathrm{T}}$ where U is an orthogonal matrix of $d \times d$ and Σ is a diagonal matrix of $(\lambda_1, \lambda_2, ..., \lambda_d)$. Since C_t is assumed to be a positive semi-definite matrix, all eigen values are equal to or higher than 0. Thus, $C_t^{1/2}$ can be computed as $C_t^{1/2} = U \Sigma^{1/2} U^{\mathrm{T}}$. By setting the covariance term of Equation (6) to $D^{-1} = (C_t^{1/2})^{-1} = U \Sigma^{-1/2} U^{\mathrm{T}}$, the Gaussian distribution of g_p is represented as follows:*

$$\begin{aligned} g_p &= C_t^{-1/2} \hat{g}_t | \bar{g}_t \sim N(C_t^{-1/2} \bar{g}_t, C_t^{-1/2} C_t (C_t^{-1/2})^{\mathrm{T}}) \\ &= N(C_t^{-1/2} \bar{g}_t, U \Sigma^{-1/2} U^{\mathrm{T}} U \Sigma U^{\mathrm{T}} (U \Sigma^{-1/2} U^{\mathrm{T}})^{\mathrm{T}}) \\ &= N(C_t^{-1/2} \bar{g}_t,\ \boldsymbol{I}). \end{aligned}$$

In the above formulations, since U is an orthogonal matrix, we use $U U^{\mathrm{T}} = \boldsymbol{I}$ and $(U \Sigma^{-1/2} U^{\mathrm{T}})^{\mathrm{T}} = U \Sigma^{-1/2} U^{\mathrm{T}}$. As a result, we have the distribution of Equation (7). □

The above theorem indicates that the transformation of $g_p = C_t^{-1/2} \hat{g}$ results in the Gaussian distribution of g_p whose covariance matrix is identity matrix $\boldsymbol{I}$. In other words, we can control the covariance matrix to be $\boldsymbol{I}$ by using g_p instead of $\hat{g}$.

Our preconditioning transforms the value of variance for first order gradients into 1 by using g_p. However, g_p may have an extremely large value if the variance is 1. Thus, we introduce hyper-parameter ρ to generalize our preconditioning. Specifically, by using the transformation of $g_p = \rho C_t^{-1/2} \hat{g}$ instead of $g_p = C_t^{-1/2} \hat{g}$, we have the following distribution:

$$\rho C_t^{-\frac{1}{2}} \hat{g}_t | \bar{g}_t \sim N\left(\rho C_t^{-\frac{1}{2}} \bar{g}_t, \rho^2\, \boldsymbol{I}\right). \tag{8}$$

The above equation denotes that ρ controls the value of the covariance matrix while the previous transformation only gives an identity matrix as shown in Equation (7). We show that ρ has the same role as learning rate α when we derive SDProp in the next section.

Since we compute the first order gradients at each time t in SGD, we have to incrementally compute the covariance matrix C_t although Theorem 1 is based on the property that C_t is a positive semi-definite matrix. In order to incrementally compute C_t as a positive semi-definite matrix, we use the online updating rule of [Sra *et al.*, 2012] as follows:

$$C_t = \gamma C_{t-1} + \gamma(1-\gamma)(\hat{g}_t - \mu_{t-1})(\hat{g}_t - \mu_{t-1})^{\mathrm{T}} \quad (9)$$

$$\mu_t = \gamma \mu_{t-1} + (1-\gamma)\hat{g}_t \quad (10)$$

where μ_t is the moving average of $\hat{g}_t$ and γ is the hyper-parameter of the decay rate for the moving average that has $\gamma \in [0, 1)$. C_t and μ_t are initialized as $\mu_1 = \hat{g}_1$ and $C_1 = 0$. The above updating rule gives the following property:

Theorem 2. *If we compute covariance matrix C_t by using Equations (9) and (10), C_t is positive semi-definite.*

Proof. *In order to prove Theorem 2, we first prove that $(\hat{g}_t - \mu_{t-1})(\hat{g}_t - \mu_{t-1})^{\mathrm{T}}$ in Equation (9) is a positive semi-definite matrix. By setting $y = x^{\mathrm{T}}(\hat{g}_t - \mu_{t-1})$, we have:*

$$x^{\mathrm{T}}(\hat{g}_t - \mu_{t-1})(\hat{g}_t - \mu_{t-1})^{\mathrm{T}} x = yy^{\mathrm{T}} \geq 0.$$

By following the definition of positive semi-definite matrixes, if we have matrix A of $d \times d$ such that $x^{\mathrm{T}}Ax \geq 0$ holds for every non-zero column vector x of d real numbers, A is a positive semi-definite matrix. Since the above inequation shows that $x^{\mathrm{T}}(\hat{g}_t - \mu_{t-1})(\hat{g}_t - \mu_{t-1})^{\mathrm{T}} x \geq 0$ holds, it is clear that $(\hat{g}_t - \mu_{t-1})(\hat{g}_t - \mu_{t-1})^{\mathrm{T}}$ is a positive semi-definite matrix even if μ_{t-1} in Equation (10) has any real value.

Then, we prove that C_t in Equation (9) is a positive semi-definite matrix by mathematical induction.

Initial step: If t=1, the initialization yields $C_1 = 0$. Since C_2 is computed as $C_2 = \gamma(1-\gamma)(\hat{g}_2 - \mu_1)(\hat{g}_2 - \mu_1)^{\mathrm{T}}$ by using Equation (9) and (10), C_2 is a positive semi-definite matrix. This is because $(\hat{g}_t - \mu_{t-1})(\hat{g}_t - \mu_{t-1})^{\mathrm{T}}$ is a positive semi-definite matrix as proved above.

Inductive step: We assume that C_{t-1} is a positive semi-definite matrix. Since C_t is computed as $C_t = \gamma C_{t-1} + \gamma(1-\gamma)(\hat{g}_t - \mu_{t-1})(\hat{g}_t - \mu_{t-1})^{\mathrm{T}}$ by using Equations (9) and (10), $x^{\mathrm{T}}C_t x$ is represented as follows:

$$\begin{aligned} x^{\mathrm{T}}C_t x &= x^{\mathrm{T}}(\gamma C_{t-1} + \gamma(1-\gamma)(\hat{g}_t - \mu_{t-1})(\hat{g}_t - \mu_{t-1})^{\mathrm{T}})x \\ &= \gamma x^{\mathrm{T}} C_{t-1} x + \gamma(1-\gamma) x^{\mathrm{T}}(\hat{g}_t - \mu_{t-1})(\hat{g}_t - \mu_{t-1})^{\mathrm{T}} x. \end{aligned}$$

In the above equation, $x^{\mathrm{T}}C_{t-1}x \geq 0$ and $x^{\mathrm{T}}(\hat{g}_t - \mu_{t-1})(\hat{g}_t - \mu_{t-1})^{\mathrm{T}}x \geq 0$ because C_{t-1} and $(\hat{g}_t - \mu_{t-1})(\hat{g}_t - \mu_{t-1})^{\mathrm{T}}$ are positive semi-definite matrices. Therefore, C_t is a positive semi-definite matrix because $x^{\mathrm{T}}C_t x \geq 0$ holds in the above equation. This completes the inductive step. □

Thus, if we compute C_t by using Equations (9) and (10), we can execute the preconditioning specified by Theorem 1.

Note that Hessian-based preconditioning cannot control the oscillation of first order gradients. This is because its transformation results in the distribution of $N(H^{-\frac{1}{2}}\bar{g}_t, H^{-\frac{1}{2}}C_t(H^{-\frac{1}{2}})^{\mathrm{T}})$ where the covariance matrix is uncontrollable. In addition, since Hessian H may not be a positive semi-definite matrix, it is difficult to compute $H^{-1/2}$. Therefore, our covariance matrix based preconditioning inherently differs from Hessian-based preconditioning. Our idea of preconditioning C_t is more suitable than Hessian-based preconditioning in handling the oscillation triggered by the noise of first order gradients.

3.3 Algorithm

Since deep neural networks have a large number of parameters, the idea described in the previous section incurs large memory consumption of $O(d^2)$ where d is the number of parameters. In addition, it costs $O(d^3)$ time to compute $D = C_t^{1/2}$ by using eigenvalue decomposition [Halko *et al.*, 2011]. To avoid these problems, we employ diagonal preconditioning matrix $D = \mathrm{diag}(C_t)^{1/2}$. Since this approach only needs the diagonal terms, the memory and computation costs are $O(d)$. Although this approach ignores the correlation of first order gradients, it is sufficient to control the oscillation in each dimension. This is because the diagonal of C_t represents the variance of the oscillation as described in the previous section. By picking the diagonal of Equation (8), the updating rule is:

$$\theta_t = \theta_{t-1} - \rho \cdot \mathrm{diag}\left(C_t\right)^{-\frac{1}{2}} \nabla f\left(\theta_{i,t-1}; x_{t-1}\right). \quad (11)$$

We rewrite this updating rule (all steps) as follows:

$$\mu_{i,t} = \gamma \mu_{i,t-1} + (1-\gamma)\nabla f\left(\theta_{i,t-1}; x_{t-1}\right) \quad (12)$$

$$c_{i,t}^2 = \gamma c_{i,t-1}^2 + \gamma(1-\gamma)(\nabla f\left(\theta_{i,t-1}; x_{t-1}\right) - \mu_{i,t-1})^2 \quad (13)$$

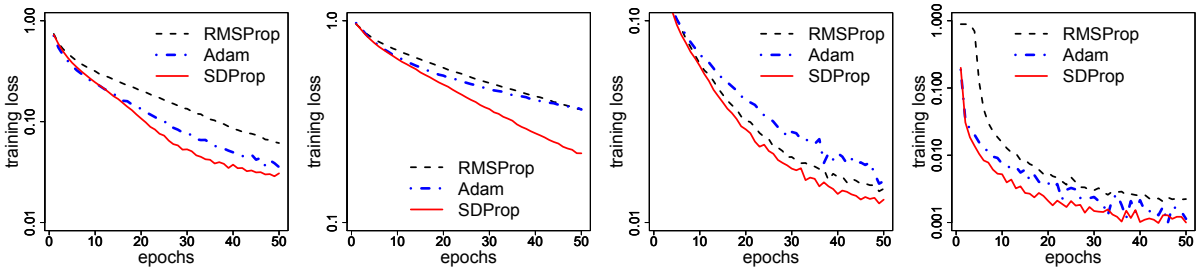
$$\theta_{i,t} = \theta_{i,t-1} - \frac{\rho}{\sqrt{c_{i,t}^2 + \epsilon}} \nabla f(\theta_{i,t-1}; x_{t-1}) \quad (14)$$

where $\mu_{i,t}$ is the moving average of first order gradients for the i-th parameter at time t and γ is the hyper-parameter of the decay rate for the moving average that has $\gamma \in [0, 1)$. $c_{i,t}^2$ is the exponentially moving variance of first order gradients for the i-th parameter at time t. We use γ in Equation (13) as the decay rate of the exponentially moving variance. $\mu_{i,t}$ and $c_{i,t}^2$ are initialized as $\mu_{i,1} = \nabla f\left(\theta_{i,0}; x_0\right)$ and $c_{i,1}^2 = 0$, respectively. For stable computation, ϵ is set at a small positive value. Equation (14) corresponds to Equation (11). We call the algorithm *SDProp* because Equation (14) includes *Standard Deviation* $\sqrt{c_{i,t}^2}$. Although $c_{i,t}$ includes the bias imposed by initialization, we can remove the bias in the same way as [Kingma and Ba, 2014].

Notice that ρ takes the same role as learning rate α in Equation (3) of RMSProp. Therefore, Equation (14) divides the learning rate by the square root of *centered* variance $c_{i,t}^2$ while Equation (3) of RMSProp divides the learning rate by the square root of *uncentered* variance $v_{i,t}^2$. In other words, RMSProp and its follow-up methods such as Adam adapt the learning rate by the magnitude of gradients while we adapt it by the variance of gradients. Although RMSProp and SDProp have similar updating rules, they have totally different goals as described in the previous sections. RMSProp executes Hessian-based preconditioning while SDProp executes covariance matrix based preconditioning.

4 Experiments

We performed experiments to compare SDProp to RMSProp and Adam, a state-of-the-art algorithm based on RMSProp. [Kingma and Ba, 2014] shows that Adam is a more efficient and effective approach than RMSProp or AdaDelta by integrating momentum into RMSProp. First, we show the efficiency and effectiveness of our approach by using CNN.



(a) Cifar-10 (b) Cifar-100 (c) SVHN (d) MNIST

Figure 1: Training losses for CNN. We show the results for (a) Cifar-10, (b) Cifar-100, (c) SVHN and (d) MNIST.

Second, since SDProp effectively handles the oscillation described in the previous section, we evaluate SDProp by using small mini-batches which suffer noise in the first order gradients. Third, we show the efficiency and effectiveness of SDProp for RNN. Fourth, we demonstrate the effectiveness of SDProp for 20 layered fully-connected neural network that is difficult to train due to many sadle points.

4.1 Efficiency and Effectiveness for CNN

We investigate the efficiency and effectiveness of SDProp. We used 4 datasets to assess the classification of images; Cifar-10, Cifar-100 [Krizhevsky and Hinton, 2009], SVHN [Sermanet *et al.*, 2012] and MNIST. The experiments were conducted on a 7-layered CNN with ReLU activation function. The loss function was negative log likelihood. We compared SDProp to RMSProp and Adam. In SDProp, we tried various combinations of hyper-parameters by using $\gamma \in \{0.9, 0.99\}$ and $\rho \in \{0.1, 0.01, 0.001\}$. In RMSProp, we tried combinations of hyper-parameters by using $\beta \in \{0.9, 0.99\}$ and $\alpha \in \{0.1, 0.01, 0.001\}$. As a result, SDProp achieves the lowest loss in the settings of $\gamma = 0.99$, $\rho = 0.001$. RMSProp has the lowest loss when $\beta = 0.99$ and $\alpha = 0.001$. Adam achieves the lowest loss when $\beta_1 = 0.9$, $\beta_1 = 0.999$ and $\alpha = 0.001$. The mini-batch size was 128. The number of epochs was 50. We use the training loss to evaluate the algorithms because they optimize the training criterion.

Figure 1 shows the training losses of each dataset. In Cifar-10, Cifar-100 and SVHN, SDProp yielded lower losses than RMSProp and Adam in early epochs. In MNIST, although the training loss of SDProp and Adam nearly reached 0.0, SDProp reduces the loss faster than Adam. SDProp needs 50 % fewer training iterations than RMSProp to reach its final training loss in Cifar-10, Cifar-100 and MNIST. This suggests that our idea of covariance matrix based preconditioning is more efficient and effective than Hessian-based preconditioning in the mini-batch setting because RMSProp and Adam approximate Hessian-based preconditioning as described in the preliminary section. Since SDProp captures the noise, it effectively reduces the loss even if the gradients are noisy. In the next experiment, we investigate the performance of SDProp in terms of its effectiveness against noise by using noisy first order gradients.

Table 1: Training accuracy percentage for Cifar-10 in CNN for different mini-batch sizes. We tuned the hyper-parameters; the 1st row presents mini-batch size.

	16	*32*	*64*	*128*
RMSProp	81.42	93.10	94.98	95.07
Adam	83.24	93.57	95.48	97.12
SDProp	**90.17**	**94.87**	**96.54**	**97.31**

4.2 Sensitivity of Mini-batch Size

The previous experimental results show that SDProp is more efficient and effective than existing methods because it well handles the noise in our idea and in practice. In other words, SDProp is expected to effectively train the model even if we use small mini-batch sizes that incur noisy first order gradients [Dekel *et al.*, 2012]. Therefore, we investigated the sensitivity of SDProp and existing methods to mini-batch size. While the main purpose of this experiment is to reveal the one performance attribute of SDProp, the result suggests that SDProp can be used on devices with scant memory that must use small mini-batches.

We compared SDProp to RMSProp and Adam using mini-batch sizes of 16, 32, 64 and 128. We used the Cifar-10 dataset for the 10-class image classification task. We used CNN as per the previous section. The hyper-parameters are also the same as the previous section; they are tuned by grid search. The number of epochs was 50.

Table 1 shows the final training accuracies. SDProp outperforms RMSProp and Adam in all mini-batch size values examined. Specifically, although small mini-batch size of 16 incurs very noisy first order gradients, SDProp obviously achieves effective training unlike RMSProp and Adam. In addition, Table 1 shows that the superiority of our approach over RMSProp and Adam increases as mini-batch size falls. For example, if the mini-batch size is 16, our approach has 8.75 percent higher accuracy than RMSProp and 2.24 percent more accurate if the mini-batch size is 128. This indicates that our covariance matrix based preconditioning effectively handles the noise of first order gradients.

4.3 Efficiency and Effectiveness for RNN

We evaluated the efficiency and effectiveness of SDProp for the Recurrent Neural Network (RNN). In this experiment, we predicted the next character by using previous charac-

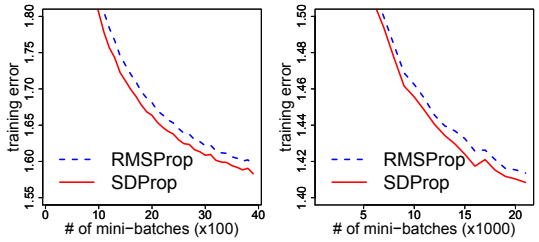


Figure 2: Cross entropies in training RNN for shakespeare dataset (left) and source code of linux kernel (right).

Table 2: Average, Best and Worst training accuracy percentage of 20 layered fully-connected networks.

			Accuracy		
Method	α	β	Ave.	Best	Worst
RMSProp	0.001	0.9	92.86	97.95	84.79
		0.99	98.81	99.11	98.34
	ρ	γ	Ave.	Best	Worst
SDProp	0.001	0.9	93.77	97.9	87.57
		0.99	**99.20**	**99.42**	**99.09**

ters via character-level RNN. We used the subset of shakespeare dataset and the source code of the linux kernel as the dataset [Karpathy *et al.*, 2015]. The size of the internal state was 128. The pre-processing of the dataset followed that of [Karpathy *et al.*, 2015]. The mini-batch size was 128. In SDProp, we tried grid search with $\rho \in \{0.1, 0.01, 0.001\}$ and $\gamma \in \{0.9, 0.99\}$. As a result, SDProp used the settings of $\rho = 0.01$ and $\gamma = 0.99$. In RMSProp, we tried grid search with $\alpha \in \{0.1, 0.01, 0.001\}$ and $\beta \in \{0.9, 0.99\}$. Finally, we used the settings of $\alpha = 0.01$ and $\beta = 0.99$ for RMSProp. The training criterion was cross entropy. We used gradient clipping and learning rate decay. Gradient clipping is a popular approach for scaling down the gradients by manually setting a threshold; it prevents gradients from exploding in RNN training [Pascanu *et al.*, 2013]. We set the threshold to 5.0. We decayed the learning rate α every tenth epoch by the factor of 0.97 for RMSProp following [Karpathy *et al.*, 2015]. In SDProp, ρ was also decayed the same as α of RMSProp.

Figure 2 shows the results of the shakespeare dataset and the source code of the linux kernel. SDProp reduces the training loss faster than RMSProp. Since SDProp effectively handles the noise induced by the mini-batch setting, it can efficiently train models other than CNN, such as RNN.

4.4 20 Layered Fully-connected Neural Network

In this section, we performed experiments to evaluate the effectiveness of SDProp for training deep fully-connected neural networks. [Dauphin *et al.*, 2014] suggests that the number of saddle points exponentially increases with the dimensions of the parameters. Since deep fully-connected networks typically have parameters with higher dimension than other models such as CNN, this optimization problem has many saddle points. This problem is challenging because SGD slowly progresses around saddle points [Dauphin *et al.*, 2014].

We used a very deep fully-connected network with 20 hidden layers, 50 hidden units and ReLU activation functions. We used the MNIST dataset for the 10-class image classification task. This setting is the same as [Neelakantan *et al.*, 2015] used in evaluating the effectiveness of SGD with high dimensional parameters. Note that MNIST is sufficient for our evaluation because, unlike CNN, fully-connected networks do not saturate the accuracy in our experiment. Our purpose is to evaluate the effectiveness under the setting of very high dimensional parameter. Thus, it is sufficient to evaluate effectiveness if the accuracy is not saturated. The training criterion was negative log likelihood. The mini-batch size was 128. We initialized parameters from a Gaussian with mean 0 and standard deviation 0.01 following [Neelakantan *et al.*, 2015]. We compared SDProp to RMSProp. In SDProp, we tried the combinations of hyper-parameters by using $\gamma \in \{0.9, 0.99\}$ and $\rho \in \{0.1, 0.01, 0.001\}$. In RMSProp, we tried the combinations of hyper-parameters by using $\beta \in \{0.9, 0.99\}$ and $\alpha \in \{0.1, 0.01, 0.001\}$. The number of epochs was 50. Although these algorithms are trapped around saddle points, its frequency may depend the initialization of parameter. Therefore, we tried 10 runs for each of the above settings.

Table 2 lists the results for the best setting of α and ρ. It shows averages, best, worst of training accuracies for each setting. The result shows that SDProp achieves higher accuracy than RMSProp for the best setting. In addition, the difference between best and worst accuracy of SDProp is smaller than RMSProp. Since SDProp effectively handles the randomness of noise, it can reduce result uncertainty. The results show that SDProp effectively trains models that have very high dimensional parameters.

5 Conclusion

We proposed SDProp for the effective and efficient training of deep neural networks. Our approach utilizes the idea of using covariance matrix based preconditioning to effectively handle the noise present in the first order gradients. Our experiments showed that, for various datasets and models, SDProp is more efficient and effective than existing methods. In addition, SDProp achieved high accuracy even if the first order gradients were noisy.

References

[Dauphin *et al.*, 2014] Yann Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and Attacking the Saddle Point Problem in High-dimensional Non-convex Optimization. In *NIPS*, pages 2933–2941, 2014.

[Dauphin *et al.*, 2015] Yann Dauphin, Harm de Vries, and Yoshua Bengio. Equilibrated Adaptive Learning Rates for Non-convex Optimization. In *NIPS*, pages 1504–1512, 2015.

[Dekel *et al.*, 2012] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. Optimal Distributed Online Prediction using Mini-batches. *Journal of Machine Learning Research*, 13:165–202, 2012.

[Elman, 1990] Jeffrey Elman. Finding Structure in Time. *Cognitive Science*, 14(2):179–211, 1990.

[Fujiwara *et al.*, 2016a] Yasuhiro Fujiwara, Yasutoshi Ida, Junya Arai, Mai Nishimura, and Sotetsu Iwamura. Fast Algorithm for the Lasso based L1-Graph Construction. *In proceedings of the Very Large Database Endowment(PVLDB)*, 10(3):229–240, 2016.

[Fujiwara *et al.*, 2016b] Yasuhiro Fujiwara, Yasutoshi Ida, Hiroaki Shiokawa, and Sotetsu Iwamura. Fast Lasso Algorithm via Selective Coordinate Descent. In *AAAI*, pages 1561–1567, 2016.

[Fukuda *et al.*, 2014] Yukikatsu Fukuda, Yasutoshi Ida, Takashi Matsumoto, Naohiro Takemura, and Kaoru Sakatani. A Bayesian Algorithm for Anxiety Index Prediction based on Cerebral Blood Oxygenation in the Prefrontal Cortex Measured by Near Infrared Spectroscopy. *IEEE journal of translational engineering in health and medicine*, 2:1–10, 2014.

[Halko *et al.*, 2011] Nathan Halko, Per-Gunnar Martinsson, and Joel Tropp. Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. *SIAM review*, 53(2):217–288, 2011.

[Ida *et al.*, 2013] Yasutoshi Ida, Takuma Nakamura, and Takashi Matsumoto. Domain-dependent/independent Topic Switching Model for Online Reviews with Numerical Ratings. In *CIKM*, pages 229–238, 2013.

[Karpathy *et al.*, 2015] Andrej Karpathy, Justin Johnson, and Fei-Fei Li. Visualizing and Understanding Recurrent Networks. *arXiv preprint arXiv:1506.02078*, 2015.

[Kingma and Ba, 2014] Diederik Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *International Conference in Learning Representations (ICLR)*, 2014.

[Krizhevsky and Hinton, 2009] Alex Krizhevsky and Geoffrey Hinton. Learning Multiple Layers of Features from Tiny Images, 2009.

[LeCun *et al.*, 1998] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[Miyashita *et al.*, 2013] Hiroki Miyashita, Takuma Nakamura, Yasutoshi Ida, Takashi Matsumoto, and Takashi Kaburagi. Nonparametric Bayes-based Heterogeneous Drosophila Melanogaster Gene Regulatory Network Inference: T-process Regression. In *International Conference on Artificial Intelligence and Applications*, pages 51–58, 2013.

[Neelakantan *et al.*, 2015] Arvind Neelakantan, Luke Vilnis, Quoc V Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding Gradient Noise Improves Learning for Very Deep Networks. *arXiv preprint arXiv:1511.06807*, 2015.

[Pascanu *et al.*, 2013] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the Difficulty of Training Recurrent Neural Networks. In *ICML*, pages 1310–1318, 2013.

[Robbins and Monro, 1951] Herbert Robbins and Sutton Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, pages 400–407, 1951.

[Sermanet *et al.*, 2012] Pierre Sermanet, Sandhya Chintala, and Yann LeCun. Convolutional Neural Networks Applied to House Numbers Digit Classification. In *International Conference on Pattern Recognition (ICPR)*, pages 3288–3291, 2012.

[Sra *et al.*, 2012] Suvrit Sra, Sebastian Nowozin, and Stephen Wright. *Optimization for Machine Learning*. Mit Press, 2012.

[Tieleman and Hinton, 2012] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the Gradient by a Running Average of its Recent Magnitude. *COURSERA: Neural Networks for Machine Learning*, 2012.

[Zeiler, 2012] Matthew Zeiler. ADADELTA: An Adaptive Learning Rate Method. *arXiv preprint arXiv:1212.5701*, 2012.