NORMALIZED DIRECTION-PRESERVING ADAM

Anonymous authors
Paper under double-blind review

ABSTRACT

Optimization algorithms for training deep models not only affects the convergence rate and stability of the training process, but are also highly related to the generalization performance of the models. While adaptive algorithms, such as Adam and RMSprop, have shown better optimization performance than stochastic gradient descent (SGD) in many scenarios, they often lead to worse generalization performance than SGD, when used for training deep neural networks (DNNs). In this work, we identify two problems of Adam that may degrade the generalization performance. As a solution, we propose the normalized direction-preserving Adam (ND-Adam) algorithm, which combines the best of both worlds, i.e., the good optimization performance of Adam, and the good generalization performance of SGD. In addition, we further improve the generalization performance in classification tasks, by using batch-normalized softmax. This study suggests the need for more precise control over the training process of DNNs.

1 INTRODUCTION

In contrast with the growing complexity of neural network architectures (Szegedy et al., 2015; He et al., 2016; Hu et al., 2017), the training methods remain relatively simple. Most practical optimization methods for deep neural networks (DNNs) are based on the stochastic gradient descent (SGD) algorithm. However, the learning rate of SGD, as a hyperparameter, is often difficult to tune, since the magnitudes of different parameters can vary widely, and adjustment is required throughout the training process.

To tackle this problem, several adaptive variants of SGD have been developed, including Adagrad (Duchi et al., 2011), Adadelta (Zeiler, 2012), RMSprop (Tieleman & Hinton, 2012), Adam (Kingma & Ba, 2014), etc. These algorithms aim to adapt the learning rate to different parameters automatically, by normalizing the global learning rate based on historical statistics of the gradient w.r.t. each parameter. Although these algorithms can usually simplify learning rate settings, and lead to faster convergence, it is observed that their generalization performance tend to be significantly worse than that of SGD in some scenarios (Wilson et al., 2017). This intriguing phenomenon may explain why SGD (possibly with momentum) is still prevalent in training state-of-the-art deep models, especially feedforward DNNs (Szegedy et al., 2015; He et al., 2016; Hu et al., 2017). Furthermore, recent work has shown that DNNs are capable of fitting noise data (Zhang et al., 2017), suggesting that their generalization capabilities are not the mere result of DNNs themselves, but are entwined with optimization (Arpit et al., 2017).

This work aims to fill the gap between SGD and its adaptive variants. To this end, we identify two problems of Adam that may degrade the generalization performance, and show how these problems are (partially) avoided by using SGD with L2 weight decay. The first problem lies in the fact that the directions of Adam parameter updates are different from that of SGD, i.e., Adam does not preserve the directions of gradients as SGD does. This difference has been discussed in rather recent literature (Wilson et al., 2017), where the authors show that adaptive methods can find drastically different solutions than SGD in some cases. Secondly, while the magnitudes of Adam parameter updates are invariant to rescaling of the gradient, the effect of the updates on the *same* overall network function still varies with the magnitudes of parameters. As we show, however, this problem can be partially avoided by using SGD with L2 weight decay, which implicitly normalizes the weight vectors, such that the magnitude of each vector's direction change does not depend on its L2-norm.

Next, we propose the normalized direction-preserving Adam (ND-Adam) algorithm, which preserves the direction of the gradient w.r.t. each weight vector, and incorporates a special form of

weight normalization (Salimans & Kingma, 2016). Compared to SGD and Adam, ND-Adam is more robust to improper initialization, and vanishing or exploding gradients. While retaining the superior optimization performance of Adam, ND-Adam achieves the regularization effect of L2 weight decay in a more consistent and principled manner. By using ND-Adam, we are able to achieve significantly better generalization performance than vanilla Adam, and at the same time, obtain much lower training loss at convergence, compared to SGD with L2 weight decay. By closing the gap between SGD and Adam in terms of generalization ability, we also shed some light on why certain optimization algorithms generalize better than others.

Furthermore, we find that the learning signal backpropagated from the softmax layer varies with the overall magnitude of the logits, without proper control. Based on the observation, we apply batch normalization to the logits with a single tunable scaling factor, which further improves the generalization performance in classification tasks.

In essence, our proposed methods, ND-Adam and batch-normalized softmax, enable more precise control over the directions of parameter updates, the learning rates, and the learning signals.

2 Background and Motivation

2.1 Adaptive Moment Estimation (Adam)

Adaptive moment estimation (Adam) (Kingma & Ba, 2014) is a stochastic optimization method that applies individual adaptive learning rates to different parameters, based on the estimates of the first and second moments of the gradients. Specifically, for n trainable parameters, $\theta \in \mathbb{R}^n$, Adam maintains a running average of the first and second moments of the gradient w.r.t. each parameter as

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\, g_t, \tag{1a}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\, g_t^2. \tag{1b}$$

Here, t denotes the time step, $m_t \in \mathbb{R}^n$ and $v_t \in \mathbb{R}^n$ denote respectively the first and second moments, and $\beta_1 \in \mathbb{R}$ and $\beta_2 \in \mathbb{R}$ are the corresponding decay factors. Kingma & Ba (2014) further notice that, since m_0 and v_0 are initialized to 0's, they are biased towards zero during the initial time steps, especially when the decay factors are large (i.e., close to 1). Thus, for computing the next update, they need to be corrected as

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \hat{v}_t = \frac{v_t}{1 - \beta_2^t}, \tag{2}$$

where β_1^t, β_2^t are the t-th powers of β_1, β_2 respectively. Then, we can update each parameter as

$$\theta_t = \theta_{t-1} - \frac{\alpha_t}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t, \tag{3}$$

where α_t is the global learning rate, and ϵ is a small constant to avoid division by zero. Note the above computations between vectors are element-wise.

A distinguishing merit of Adam is that the magnitudes of parameter updates are invariant to rescaling of the gradient, as shown by the adaptive learning rate term, $\frac{\alpha_t}{\sqrt{\hat{v}_t}+\epsilon}$. However, there are two potential problems when applying Adam to DNNs.

First, in some scenarios, DNNs trained with Adam generalize worse than that trained with stochastic gradient descent (SGD) (Wilson et al., 2017). Zhang et al. (2017) demonstrate that over-parameterized DNNs are capable of memorizing the entire dataset, no matter if it is natural data or meaningless noise data, and thus suggest much of the generalization power of DNNs comes from the training algorithm, e.g., SGD and its variants. It coincides with another recent work (Wilson et al., 2017), which shows that simple SGD often yields better generalization performance than adaptive gradient methods, such as Adam. As pointed out by the latter, the difference in the generalization performance may result from the different directions of updates. Specifically, for each hidden unit, the SGD update of its input weight vector can only lie in the span of all possible input vectors, which, however, is not the case for Adam due to the individually adapted learning rates. We refer to this problem as the *direction missing problem*.

Second, while batch normalization (Ioffe & Szegedy, 2015) can significantly accelerate the convergence of DNNs, the input weights and the scaling factor of each hidden unit can be scaled in infinitely many (but consistent) ways, without changing the function implemented by the hidden unit. Thus, for different magnitudes of an input weight vector, the updates given by Adam can have different effects on the overall network function, which is undesirable. Furthermore, even when batch normalization is not used, a network using linear rectifiers (e.g., ReLU, leaky ReLU) as activation functions, is still subject to ill-conditioning of the parameterization (Glorot et al., 2011), and hence the same problem. We refer to this problem as the *ill-conditioning problem*.

2.2 L2 Weight Decay

L2 weight decay is a regularization technique frequently used with SGD. It often has a significant effect on the generalization performance of DNNs. Despite the simplicity and crucial role of L2 weight decay in the training process, it remains to be explained how it works in DNNs. A common justification for L2 weight decay is that it can be introduced by placing a Gaussian prior upon the weights, when the objective is to find the maximum a posteriori (MAP) weights (Blundell et al., 2015). However, as discussed in Sec. 2.1, the magnitudes of input weight vectors are irrelevant in terms of the overall network function, in some common scenarios, rendering the variance of the Gaussian prior meaningless.

We propose to view L2 weight decay in neural networks as a form of weight normalization, which may better explain its effect on the generalization performance. Consider a neural network trained with the following loss function:

$$\widetilde{L}\left(\theta;\mathcal{D}\right) = L\left(\theta;\mathcal{D}\right) + \frac{\lambda}{2}\sum_{i\in\mathcal{N}}\left\|w_i\right\|_2^2, \tag{4}$$

where $L\left(\theta;\mathcal{D}\right)$ is the original loss function specified by the task, $\mathcal{D}$ is a batch of training data, $\mathcal{N}$ is the set of all hidden units, and w_i denotes the input weights of hidden unit i, which is included in the trainable parameters, θ. For simplicity, we consider SGD updates without momentum. Therefore, the update of w_i at each time step is

$$\Delta w_i = -\alpha\frac{\partial\widetilde{L}}{\partial w_i} = -\alpha\left(\frac{\partial L}{\partial w_i} + \lambda w_i\right), \tag{5}$$

where α is the step size. As we can see from Eq. (5), the gradient magnitude of the L2 penalty is proportional to $\left\|w_i\right\|_2$, thus forms a negative feedback loop that stabilizes $\left\|w_i\right\|_2$ to an equilibrium value. Empirically, we find that $\left\|w_i\right\|_2$ tends to increase or decrease dramatically at the beginning of the training, and then varies mildly within a small range, which indicates $\left\|w_i\right\|_2 \approx \left\|w_i + \Delta w_i\right\|_2$. In practice, we usually have $\left\|\Delta w_i\right\|_2 / \left\|w_i\right\|_2 \ll 1$, thus Δw_i is approximately orthogonal to w_i, i.e. $w_i \cdot \Delta w_i \approx 0$.

Let $l_{\|w_i}$ and $l_{\perp w_i}$ be the vector projection and rejection of $\frac{\partial L}{\partial w_i}$ on w_i, which are defined as

$$l_{\|w_i} = \left(\frac{\partial L}{\partial w_i}\cdot\frac{w_i}{\left\|w_i\right\|_2}\right)\frac{w_i}{\left\|w_i\right\|_2}, l_{\perp w_i} = \frac{\partial L}{\partial w_i} - l_{\|w_i}. \tag{6}$$

From Eq. (5) and (6), it is easy to show

$$\frac{\left\|\Delta w_i\right\|_2}{\left\|w_i\right\|_2} \approx \frac{\left\|l_{\perp w_i}\right\|_2}{\left\|l_{\|w_i}\right\|_2}\alpha\lambda. \tag{7}$$

As discussed in Sec. 2.1, when batch normalization is used, or when linear rectifiers are used as activation functions, the magnitude of $\left\|w_i\right\|_2$ is irrelevant. Thus, it is the direction of w_i that actually makes a difference in the overall network function. If L2 weight decay is not applied, the magnitude of w_i's direction change will decrease as $\left\|w_i\right\|_2$ increases during the training process, which can potentially lead to overfitting (discussed in detail in Sec. 3.2). On the other hand, Eq. (7) shows that L2 weight decay implicitly normalizes the weights, such that the magnitude of w_i's direction change does not depend on $\left\|w_i\right\|_2$, and can be tuned by the product of α and λ. In the following, we refer to $\left\|\Delta w_i\right\|_2 / \left\|w_i\right\|_2$ as the *effective learning rate* of w_i.

While L2 weight decay produces the normalization effect in an implicit and approximate way, we will show that explicitly doing so can result in further improved optimization and generalization performance.

3 NORMALIZED DIRECTION-PRESERVING ADAM

We first present the normalized direction-preserving Adam (ND-Adam) algorithm, which essentially improves the optimization of the input weights of hidden units, while employing the vanilla Adam algorithm to update other parameters. Specifically, we divide the trainable parameters, θ, into two sets, θ^v and θ^s, such that $\theta^v = \{w_i | i \in \mathcal{N}\}$, and $\theta^s = \{\theta \setminus \theta^v\}$. Then we update θ^v and θ^s by different rules, as described by Alg. 1. The learning rates for the two sets of parameters are denoted respectively by α_t^v and α_t^s.

Algorithm 1: Normalized direction-preserving Adam

```
/* Initialization                                                    */
```
$t \leftarrow 0$;
for $i \in \mathcal{N}$ **do**
  $w_{i,0} \leftarrow w_{i,0} / \|w_{i,0}\|_2$;
  $m_0(w_i) \leftarrow 0$;
  $v_0(w_i) \leftarrow 0$;

```
/* Perform T iterations of training                                  */
```
while $t < T$ **do**
  $t \leftarrow t + 1$;
  `/* Update` θ^v `*/`
  for $i \in \mathcal{N}$ **do**
    $\bar{g}_t(w_i) \leftarrow \partial L / \partial w_i$;
    $g_t(w_i) \leftarrow \bar{g}_t(w_i) - (\bar{g}_t(w_i) \cdot w_{i,t-1}) w_{i,t-1}$;
    $m_t(w_i) \leftarrow \beta_1 m_{t-1}(w_i) + (1 - \beta_1) g_t(w_i)$;
    $v_t(w_i) \leftarrow \beta_2 v_{t-1}(w_i) + (1 - \beta_2) \|g_t(w_i)\|_2^2$;
    $\hat{m}_t(w_i) \leftarrow m_t(w_i) / (1 - \beta_1^t)$;
    $\hat{v}_t(w_i) \leftarrow v_t(w_i) / (1 - \beta_2^t)$;
    $\bar{w}_{i,t} \leftarrow w_{i,t-1} - \alpha_t^v \hat{m}_t(w_i) / \left(\sqrt{\hat{v}_t(w_i)} + \epsilon\right)$;
    $w_{i,t} \leftarrow \bar{w}_{i,t} / \|\bar{w}_{i,t}\|_2$;
  `/* Update` θ^s `using Adam */`
  $\theta_t^s \leftarrow \mathrm{AdamUpdate}\left(\theta_{t-1}^s; \alpha_t^s, \beta_1, \beta_2\right)$;
return θ_T;

In Alg. 1, the iteration over $\mathcal{N}$ can be performed in parallel, and thus introduces no extra computational complexity. Compared to Adam, computing $g_t(w_i)$ and $w_{i,t}$ may take slightly more time, which, however, is negligible in practice. On the other hand, to estimate the second order moment of each $w_i \in \mathbb{R}^n$, Adam maintains n scalars, whereas ND-Adam requires only one scalar, $v_t(w_i)$. Thus, ND-Adam has smaller memory overhead than Adam.

In the following, we address the direction missing problem and the ill-conditioning problem discussed in Sec. 2.1, and explain Alg. 1 in detail. We show how the proposed algorithm jointly solves the two problems, as well as its relation to other normalization schemes.

3.1 PRESERVING GRADIENT DIRECTIONS

Assuming the stationarity of a hidden unit's input distribution, the SGD update (possibly with momentum) of the input weight vector is a linear combination of historical gradients, and thus can only lie in the span of the input vectors. As a result, the input weight vector itself will eventually converge to the same subspace.

On the contrary, the Adam algorithm adapts the global learning rate to each scalar parameter independently, such that the gradient of each parameter is normalized by a running average of its magnitudes, which changes the direction of the gradient. To preserve the direction of the gradient w.r.t. each input weight vector, we generalize the learning rate adaptation scheme from scalars to vectors.

Let $g_t(w_i)$, $m_t(w_i)$, $v_t(w_i)$ be the counterparts of g_t, m_t, v_t for vector w_i. Since Eq. (1a) is a linear combination of historical gradients, it can be extended to vectors without any change; or equivalently, we can rewrite it for each vector as

$$m_t(w_i) = \beta_1 m_{t-1}(w_i) + (1-\beta_1) g_t(w_i). \tag{8}$$

We then extend Eq. (1b) as

$$v_t(w_i) = \beta_2 v_{t-1}(w_i) + (1-\beta_2) \|g_t(w_i)\|_2^2, \tag{9}$$

i.e., instead of estimating the average gradient magnitude for each individual parameter, we estimate the average of $\|g_t(w_i)\|_2^2$ for each vector w_i. In addition, we modify Eq. (2) and (3) accordingly as

$$\hat{m}_t(w_i) = \frac{m_t(w_i)}{1-\beta_1^t}, \hat{v}_t(w_i) = \frac{v_t(w_i)}{1-\beta_2^t}, \tag{10}$$

and

$$w_{i,t} = w_{i,t-1} - \frac{\alpha_t^v}{\sqrt{\hat{v}_t(w_i)} + \epsilon} \hat{m}_t(w_i). \tag{11}$$

Here, $\hat{m}_t(w_i)$ is a vector with the same dimension as w_i, whereas $\hat{v}_t(w_i)$ is a scalar. Therefore, when applying Eq. (11), the direction of the update is the negative direction of $\hat{m}_t(w_i)$, and thus is in the span of the historical gradients of w_i.

It is worth noting that only the input to the first layer (i.e., the training data) is stationary throughout training. Thus, for the weights of an upper layer to converge to the span of its input vectors, it is necessary for the lower layers to converge first. Interestingly, this predicted phenomenon may have been observed in practice (Brock et al., 2017).

Despite the empirical success of SGD, a question remains as to why it is desirable to constrain the input weights in the span of the input vectors. A possible explanation is related to the manifold hypothesis, which suggests that real-world data presented in high dimensional spaces (images, audios, text, etc) concentrates on manifolds of much lower dimensionality (Cayton, 2005; Narayanan & Mitter, 2010). In fact, commonly used activation functions, such as (leaky) ReLU, sigmoid, tanh, can only be activated (not saturating or having small gradients) by a portion of the input vectors, in whose span the input weights lie upon convergence. Assuming the local linearity of the manifolds of data or hidden-layer representations, constraining the input weights in the subspace that contains some of the input vectors, encourages the hidden units to form local coordinate systems on the corresponding manifold, which can lead to good representations (Rifai et al., 2011).

3.2 Spherical Weight Optimization

The ill-conditioning problem occurs when the magnitude change of an input weight vector can be compensated by other parameters, such as the scaling factor of batch normalization, or the output weight vector, without affecting the overall network function. Consequently, suppose we have two DNNs that parameterize the same function, but with some of the input weight vectors having different magnitudes, applying the same SGD or Adam update rule will, in general, change the network functions in different ways. Thus, the ill-conditioning problem makes the training process inconsistent and difficult to control.

More importantly, when the weights are not properly regularized (e.g., without using L2 weight decay), the magnitude of w_i's direction change will decrease as $\|w_i\|_2$ increases during the training process. As a result, the effective learning rate for w_i tends to decrease faster than expected, making the network converge to sharp minima (Hoffer et al., 2017). It is well known that sharp minima generalize worse than flat minima (Hochreiter & Schmidhuber, 1997; Keskar et al., 2017).

As shown in Sec. 2.2, L2 weight decay can alleviate the ill-conditioning problem by implicitly and approximately normalizing the weights. However, we still do not have a precise control over $\|\Delta w_i\|_2 / \|w_i\|_2$, since $\|l_{\perp w_i}\|_2 / \left\|l_{\|w_i}\right\|_2$ is unknown and not necessarily stable. Moreover, the approximation fails when $\|w_i\|_2$ is far from the equilibrium due to improper initialization, or drastic changes in the magnitudes of the weight vectors. This problem is also addressed by (Neyshabur et al., 2015), by employing a geometry invariant to rescaling of weights. However, their proposed methods do not preserve the direction of gradient.

To address the ill-conditioning problem in a more principled way, we restrict the L2-norm of each w_i to 1, and only optimize its direction. In other words, instead of optimizing w_i in a n-dimensional space, we optimize w_i on a $(n-1)$-dimensional unit sphere. Specifically, we first obtain the raw gradient w.r.t. w_i, $\bar{g}_t(w_i) = \partial L / \partial w_i$, and project the gradient onto the unit sphere as

$$g_t(w_i) = \bar{g}_t(w_i) - (\bar{g}_t(w_i) \cdot w_{i,t-1}) w_{i,t-1}. \tag{12}$$

Here, $\|w_{i,t-1}\|_2 = 1$. Then we follow Eq. (8)-(10), and replace (11) with

$$\bar{w}_{i,t} = w_{i,t-1} - \frac{\alpha_t^v}{\sqrt{\hat{v}_t(w_i)} + \epsilon} \hat{m}_t(w_i), \tag{13a}$$

and

$$w_{i,t} = \frac{\bar{w}_{i,t}}{\|\bar{w}_{i,t}\|_2}. \tag{13b}$$

In Eq. (12), we keep only the component that is orthogonal to $w_{i,t-1}$. However, $\hat{m}_t(w_i)$ is not necessarily orthogonal as well. In addition, even when $\hat{m}_t(w_i)$ is orthogonal to $w_{i,t-1}$, Eq. (13a) can still increase $\|w_i\|_2$, according to the Pythagorean theorem. Therefore, we explicitly normalize $w_{i,t}$ in Eq. (13b), to ensure $\|w_{i,t}\|_2 = 1$ after each update. Also note that, since $w_{i,t-1}$ is a linear combination of its historical gradients, $g_t(w_i)$ still lies in the span of the historical gradients after the projection in Eq. (12).

As a result, the effective learning rate of a weight vector is

$$\frac{\|\Delta w_{i,t}\|_2}{\|w_{i,t-1}\|_2} \approx \frac{\|\hat{m}_t(w_i)\|_2}{\sqrt{\hat{v}_t(w_i)}} \alpha_t^v, \tag{14}$$

which enables precise control over the learning rate of w_i through a single hyperparameter, α_t^v, rather than two as required by Eq. (7). Note that it is possible to control the effective learning rate more precisely, by normalizing $\hat{m}_t(w_i)$ by $\|\hat{m}_t(w_i)\|_2$, instead of by $\sqrt{\hat{v}_t(w_i)}$. However, by doing so, we lose the information provided by $\|\hat{m}_t(w_i)\|_2$ at different time steps. In addition, since $\hat{m}_t(w_i)$ is less noisy than $g_t(w_i)$, $\|\hat{m}_t(w_i)\|_2 / \sqrt{\hat{v}_t(w_i)}$ becomes small near convergence, which is considered a desirable property of Adam (Kingma & Ba, 2014). Thus, we keep the gradient normalization scheme intact.

Compared to SGD with L2 weight decay, spherical weight optimization explicitly normalizes the weight vectors, such that each update to the weight vectors only changes their directions, and strictly keeps the magnitudes constant. Moreover, the magnitude of each update does not depend on the magnitude of the gradient. Thus, ND-Adam is more robust to improper initialization, and vanishing or exploding gradients. For nonlinear activation functions, such as sigmoid and tanh, an extra scaling factor is needed for each hidden unit to express functions that require unnormalized weight vectors. For instance, given an input vector $x \in \mathbb{R}^n$, and a nonlinearity $\phi(\cdot)$, the activation of hidden unit i is then given by

$$y_i = \phi(\gamma_i w_i \cdot x + b_i), \tag{15}$$

where γ_i is the scaling factor, and b_i is the bias.

3.3 Relation to Weight Normalization and Batch Normalization

A related normalization and reparameterization scheme, weight normalization (Salimans & Kingma, 2016), has been developed as an alternative to batch normalization, aiming to accelerate the convergence of SGD optimization. We note the difference between spherical weight optimization and weight normalization. First, the weight vector of each hidden unit is not directly normalized in weight normalization, i.e, $\|w_i\|_2 \neq 1$ in general. At training time, the activation of hidden unit i is

$$y_i = \phi\left(\frac{\gamma_i}{\|w_i\|_2} w_i \cdot x + b_i\right), \tag{16}$$

which is equivalent to Eq. (15) for the forward pass. For the backward pass, $\|\Delta w_i\|_2 / \|w_i\|_2$ still depends on $\|w_i\|_2$ in weight normalization, hence it does not solve the ill-conditioning problem. At inference time, both of these two schemes can combine w_i and γ_i into a single equivalent weight vector, $w_i' = \gamma_i w_i$, or $w_i' = \frac{\gamma_i}{\|w_i\|_2} w_i$.

While spherical weight optimization naturally encompasses weight normalization, it can further benefit from batch normalization. When combined with batch normalization, Eq. (15) evolves into

$$y_i = \phi\left(\gamma_i \operatorname{BN}\left(w_i \cdot x\right) + b_i\right), \tag{17}$$

where $\operatorname{BN}(\cdot)$ represents the transformation done by batch normalization without scaling and shifting. Here, γ_i serves as the scaling factor for both the normalized weight vector and batch normalization. At training time, the distribution of the input vector, x, changes over time, slowing down the training of the sub-network composed by the upper layers. Salimans & Kingma (2016) observe that, such problem cannot be eliminated by normalizing the weight vectors alone, but can be substantially mitigated by combining weight normalization and mean-only batch normalization.

Additionally, in linear rectifier networks, the scaling factors, γ_i, can be removed (or set to 1), without changing the overall network function. Since $w_i \cdot x$ is standardized by batch normalization, we have

$$\mathbb{E}_x\left[\operatorname{BN}\left(w_i \cdot x\right)^2\right] \approx 1, \tag{18}$$

and hence

$$\operatorname{Var}_x\left[\operatorname{BN}\left(w_i \cdot x\right) + b_i\right] \approx 1. \tag{19}$$

Therefore, y_i's that belong to the same layer, or different dimensions of x that fed to the upper layer, will also have comparable variances, which potentially makes the weight updates of the upper layer more stable. For these reasons, we combine the use of spherical weight optimization and batch normalization, as shown in Eq. (17).

4 Batch-normalized Softmax

For multi-class classification tasks, the softmax function is the *de facto* activation function for the output layer. Despite its simplicity and intuitive probabilistic interpretation, the learning signal it backpropagates may not always be desirable.

When using cross entropy as the surrogate loss with one-hot target vectors, the prediction is considered correct as long as $\arg\max_{c\in\mathcal{C}}(z_c)$ is the target class, where z_c is the logit before the softmax activation, corresponding to category $c \in \mathcal{C}$. Thus, the logits can be positively scaled together without changing the predictions, even though the cross entropy and its derivatives will vary with the scaling factor. Specifically, denoting the scaling factor by η, the gradient w.r.t. each logit is

$$\frac{\partial L}{\partial z_{\hat{c}}} = \eta\left[\frac{\exp\left(\eta z_{\hat{c}}\right)}{\sum_{c\in\mathcal{C}}\exp\left(\eta z_c\right)} - 1\right], \tag{20a}$$

and

$$\frac{\partial L}{\partial z_{\bar{c}}} = \frac{\eta\exp\left(\eta z_{\bar{c}}\right)}{\sum_{c\in\mathcal{C}}\exp\left(\eta z_c\right)}. \tag{20b}$$

where $\hat{c}$ is the target class, and $\bar{c} \in \mathcal{C}\setminus\{\hat{c}\}$.

For Adam and ND-Adam, since the gradient w.r.t. each scalar or vector are normalized, the absolute magnitudes of Eq. (20a) and (20b) are irrelevant. Instead, the relative magnitudes make a difference here. When η is small, we have

$$\lim_{\eta\to 0}\left|\frac{\partial L/\partial z_{\bar{c}}}{\partial L/\partial z_{\hat{c}}}\right| = \frac{1}{|\mathcal{C}|-1}, \tag{21}$$

which indicates that, when the magnitude of the logits is small, softmax encourages the logit of the target class to increase, while equally penalizing that of other classes. On the other end of the spectrum, assuming no two digits are the same, we have

$$\lim_{\eta\to\infty}\left|\frac{\partial L/\partial z_{\bar{c}'}}{\partial L/\partial z_{\hat{c}}}\right| = 1, \lim_{\eta\to\infty}\left|\frac{\partial L/\partial z_{\bar{c}''}}{\partial L/\partial z_{\hat{c}}}\right| = 0, \tag{22}$$

where $\bar{c}' = \arg\max_{c\in\mathcal{C}\setminus\{\hat{c}\}}(z_c)$, and $\bar{c}'' \in \mathcal{C}\setminus\{\hat{c}, \bar{c}'\}$. Eq. (22) indicates that, when the magnitude of the logits is large, softmax penalizes only the largest logit of the non-target classes. The latter case is related to the saturation problem of softmax discussed in Oland et al. (2017). However, they focus on the problem of small absolute gradient magnitude, which does not affect Adam and ND-Adam.

It is worth noting that both of these two cases can happen without the scaling factor. For instance, varying the norm of the weights of the softmax layer is equivalent to varying the value of η, in terms of the relative magnitude of the gradient. In the case of small η, the logits of all non-target classes are penalized equally, regardless of the difference in $\hat{z} - \bar{z}$ for different $\bar{z} \in \mathcal{C} \setminus \{\hat{z}\}$. However, it is more reasonable to penalize more the logits that are closer to $\hat{z}$, which are more likely to cause misclassification. In the case of large η, although the logit that is most likely to cause misclassification is strongly penalized, the logits of other non-target classes are ignored. As a result, the logits of the non-target classes tend to be similar at convergence, ignoring the fact that some classes are closer to each other than the others.

To exploit the prior knowledge that the magnitude of the logits should not be too small or too large, we apply batch normalization to the logits. Nevertheless, instead of setting γ_c's as trainable variables, we consider them as a single hyperparameter, $\gamma_{\mathcal{C}}$, such that $\gamma_c = \gamma_{\mathcal{C}}, \forall c \in \mathcal{C}$. Tuning the value of $\gamma_{\mathcal{C}}$ can lead to a better trade-off between the two cases described by Eq. (21) and (22). We refer to this method as batch-normalized softmax (BN-Softmax).

5 EXPERIMENTS

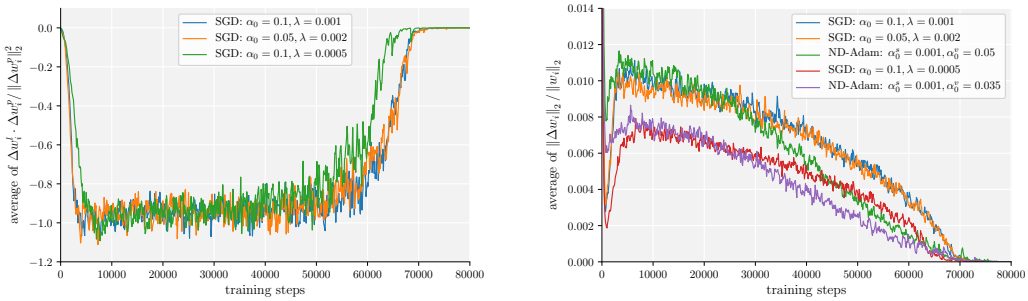
In this section, we provide empirical evidence for the analysis in Sec. 2.2, and evaluate the performance of ND-Adam and BN-Softmax on CIFAR-10 and CIFAR-100.

5.1 THE EFFECT OF L2 WEIGHT DECAY

To empirically examine the effect of L2 weight decay, we train a wide residual network (WRN) (Zagoruyko & Komodakis, 2016) of 22 layers, with a width of 7.5 times that of a vanilla ResNet. Using the notation in Zagoruyko & Komodakis (2016), we refer to this network as WRN-22-7.5. We train the network on the CIFAR-10 dataset (Krizhevsky & Hinton, 2009), with a small modification to the original WRN architecture, and with a different learning rate annealing schedule. Specifically, for simplicity and slightly better performance, we replace the last fully connected layer with a convolutional layer with 10 output feature maps. I.e., we change the layers after the last residual block from `BN-ReLU-GlobalAvgPool-FC-Softmax` to `BN-ReLU-Conv-GlobalAvgPool-Softmax`. In addition, for clearer comparisons, the learning rate is annealed according to a cosine function without restart (Loshchilov & Hutter, 2016; Gastaldi, 2017). We train the model for 80k iterations with a batch size of 128, similar to the settings in Zagoruyko & Komodakis (2016).

As a common practice, we use SGD with a momentum of 0.9, the analysis for which is similar to that in Sec. 2.2. Due to the linearity of derivatives and momentum, Δw_i can be decomposed as $\Delta w_i = \Delta w_i^l + \Delta w_i^p$, where Δw_i^l and Δw_i^p are the components corresponding to the original loss function, $L(\cdot)$, and the L2 penalty term (see Eq. (4)), respectively. Fig. 1a shows the ratio between the scalar projection of Δw_i^l on Δw_i^p and $\|\Delta w_i^p\|_2$, which indicates how the tendency of Δw_i^l to increase $\|w_i\|_2$ is compensated by Δw_i^p. Note that Δw_i^p points to the negative direction of w_i, even when momentum is used, since the direction change of w_i is slow. As shown in Fig. 1a, at the beginning of the training, Δw_i^p dominants and quickly adjusts $\|w_i\|_2$ to its equilibrium value. During the middle stage of the training, the projection of Δw_i^l on Δw_i^p, and Δw_i^p almost cancel each other out. Then, near the end of the training, the gradient of w_i diminishes rapidly to near zero, making Δw_i^p dominant again. Therefore, Eq. (7) holds more accurately during the middle stage of the training.

In Fig. 1b, we show how the value of $\|\Delta w_i\|_2 / \|w_i\|_2$ varies in different hyperparameter settings. By Eq. (7), $\|\Delta w_i\|_2 / \|w_i\|_2$ is expected to remain the same as long as $\alpha\lambda$ stays constant, which is confirmed by the fact that the curve for $\alpha_0 = 0.1, \lambda = 0.001$ overlaps with that for $\alpha_0 = 0.05, \lambda = 0.002$. However, comparing the curve for $\alpha_0 = 0.1, \lambda = 0.001$, with that for $\alpha_0 = 0.1, \lambda = 0.0005$, we can see that the value of $\|\Delta w_i\|_2 / \|w_i\|_2$ does not change proportional to $\alpha\lambda$. On the other hand, by using ND-Adam, we can control the value of $\|\Delta w_i\|_2 / \|w_i\|_2$ more precisely by adjusting the learning rate for weight vectors, α^v. For the same training step, changes in α^v lead to approximately proportional changes in $\|\Delta w_i\|_2 / \|w_i\|_2$, as shown by the two curves corresponding to ND-Adam in Fig. 1b.



(a) The scalar projection of Δw_i^l on Δw_i^p normalized by $\|\Delta w_i^p\|_2$.

(b) The relative magnitude of the weight updates, or the effective learning rate.

Figure 1: An illustration of how L2 weight decay and ND-Adam control the effective learning rate. The results are obtained from the 5th layer of the network, and other layers show similar results.

5.2 Performance Evaluation

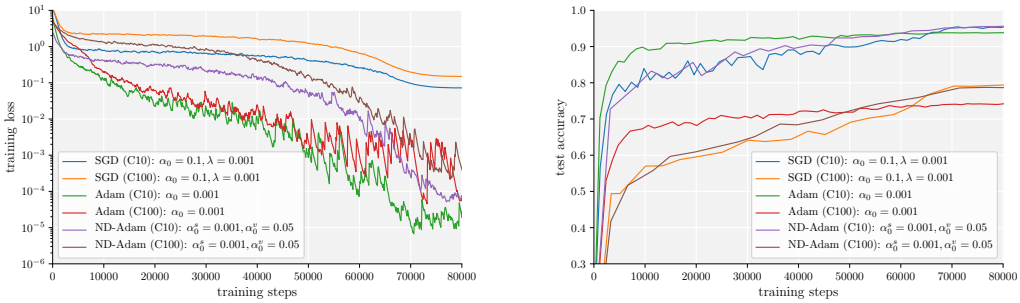
To compare the optimization and generalization performance of SGD, Adam, and ND-Adam, we train the same WRN-22-7.5 network on the CIFAR-10 and CIFAR-100 datasets. For SGD and ND-Adam, we first tune the hyperparameters for SGD ($\alpha_0 = 0.1, \lambda = 0.001$, momentum 0.9), then tune the initial learning rate of ND-Adam for weight vectors to match the effective learning rate to that of SGD ($\alpha_0^v = 0.05$), as shown in Fig. 1b. While L2 weight decay can greatly affect the performance of SGD, it does not noticeably benefit Adam in our experiments. For Adam and ND-Adam, β_1 and β_2 are set to the default values of Adam, i.e., $\beta_1 = 0.9$, $\beta_2 = 0.999$. Although the learning rate of Adam is usually set to a constant value, we observe better performance with the cosine decay scheme. The initial learning rate of Adam (α_0), and that of ND-Adam for scalar parameters (α_0^s) are both tuned to 0.001. We use the same data augmentation scheme as used in Zagoruyko & Komodakis (2016), including horizontal flips and random crops, but no dropout is used.

We first experiment with the use of trainable scaling parameters (γ_i) of batch normalization. As shown in Fig. 2a, ND-Adam converges to training losses comparable to that of Adam, which are much lower than that of SGD. More importantly, as shown in Fig. 2b, the test accuracies of ND-Adam are significantly improved upon vanilla Adam, and matches that of SGD. Note that at the early stage of training, the test accuracy of Adam increases more rapidly than that of ND-Adam and SGD, but remains at a high level afterwards. It is likely that Adam tends to quickly find and get stuck in bad local minima that do not generalize well.

The average results of 3 runs are summarized in the first part of Table 1. Interestingly, compared to SGD, ND-Adam shows slightly better performance on CIFAR-10, but worse performance on CIFAR-100. This inconsistency may be related to the problem of softmax discussed in Sec. 4, that there is a lack of proper control over the magnitude of the logits. But overall, given comparable effective learning rates, ND-Adam and SGD show similar generalization performance. In this sense, the effective learning rate is a more natural learning rate measure than the learning rate hyperparameters.

Next, we repeat the experiments with the use of BN-Softmax. As discussed in Sec. 3.2, γ_i's can be removed from a linear rectifier network, without changing the overall network function. Although this property does not strictly hold for residual networks due to the skip connections, we find that simply removing the scaling factors results in slightly improved generalization performance when using ND-Adam. However, the improvement is not consistent as it degrades performance of SGD. Interestingly, when BN-Softmax is further used, we observe consistent improvement over all three algorithms. Thus, we only report results for this setting.

The scaling factor of the logits, $\gamma_\mathcal{C}$, is set to 2.5 for CIFAR-10, and 1 for CIFAR-100. As shown in Fig. 3a, the training losses of Adam and ND-Adam again are much lower than that of SGD, although they are increased due to the regularization effect of BN-Softmax. As shown in the second part of

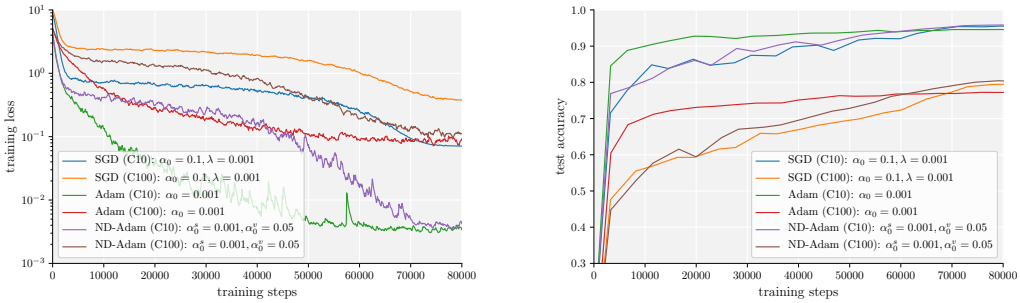


(a) The training losses on CIFAR-10/100.

(b) The test accuracies on CIFAR-10/100.

Figure 2: The training losses and test accuracies of the same network trained with SGD, Adam, and ND-Adam. Batch normalization with scaling factors is used.

Table 1, BN-Softmax significantly improves the performance of Adam and ND-Adam. Moreover, in this setting, we obtain the best generalization performance with ND-Adam, outperforming SGD and Adam on both CIFAR-10 and CIFAR-100. As a side note, the optimal value of $\gamma_{\mathcal{C}}$ tends to remain the same for networks with different widths, but increase for deeper networks. For instance, we observe that $\gamma_{\mathcal{C}} = 3.5$ works better than $\gamma_{\mathcal{C}} = 2.5$ for a WRN-34-7.5 network trained on CIFAR-10.



(a) The training losses on CIFAR-10/100.

(b) The test accuracies on CIFAR-10/100.

Figure 3: The training losses and test accuracies of the same network trained with SGD, Adam, and ND-Adam. Batch normalization *without* scaling factors, and BN-Softmax are used.

Method	CIFAR-10 Error (%)	CIFAR-100 Error (%)
BN w/ scaling factors		
SGD	4.61	20.60
Adam	6.14	25.51
ND-Adam	4.53	21.45
BN w/o scaling factors, BN-Softmax		
SGD	4.49	20.18
Adam	5.43	22.48
ND-Adam	**4.14**	**19.90**

Table 1: Test error rates on CIFAR-10 and CIFAR-100.

6 CONCLUSION

In this paper, we introduced the normalized direction-preserving Adam algorithm, which is a tailored version of Adam for training DNNs. We showed that ND-Adam preserves the direction of gradient for each weight vector, and implements the regularization effect of L2 weight decay in a more consistent and principled way, such that it combines the good optimization performance of Adam, with the good generalization performance of SGD. In addition, we introduced batch-normalized softmax, which regularizes the logits before the softmax activation, in order to provide better learning signals. We showed significantly improved generalization performance by combining ND-Adam and BN-Softmax. From a high-level view, our analysis and empirical results suggest the need for more precise control over the training process of DNNs.

REFERENCES

Devansh Arpit, Stanisław Jastrzębski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. A closer look at memorization in deep networks. *arXiv preprint arXiv:1706.05394*, 2017.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. In *ICML 2015*, 2015.

Andrew Brock, Theodore Lim, JM Ritchie, and Nick Weston. Freezeout: Accelerate training by progressively freezing layers. *arXiv preprint arXiv:1706.04983*, 2017.

Lawrence Cayton. Algorithms for manifold learning. *Univ. of California at San Diego Tech. Rep*, pp. 1–17, 2005.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

Xavier Gastaldi. Shake-shake regularization of 3-branch residual networks. In *ICLR 2017 Workshop*, 2017.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 315–323, 2011.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997.

Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. *arXiv preprint arXiv:1705.08741*, 2017.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pp. 448–456, 2015.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *ICLR 2017*, 2017.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

Ilya Loshchilov and Frank Hutter. Sgdr: stochastic gradient descent with restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Hariharan Narayanan and Sanjoy Mitter. Sample complexity of testing the manifold hypothesis. In *Advances in Neural Information Processing Systems*, pp. 1786–1794, 2010.

Behnam Neyshabur, Ruslan R Salakhutdinov, and Nati Srebro. Path-sgd: Path-normalized optimization in deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 2422–2430, 2015.

Anders Oland, Aayush Bansal, Roger B Dannenberg, and Bhiksha Raj. Be careful what you backpropagate: A case for linear output activations & gradient boosting. *arXiv preprint arXiv:1707.04199*, 2017.

Salah Rifai, Yann N Dauphin, Pascal Vincent, Yoshua Bengio, and Xavier Muller. The manifold tangent classifier. In *Advances in Neural Information Processing Systems*, pp. 2294–2302, 2011.

Tim Salimans and Diederik P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 901–909, 2016.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

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.

Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, 2017.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *ICLR 2017*, 2017.