A Theoretical View of Linear Backpropagation and its Convergence

Ziang Li, Yiwen Guo, Haodi Liu, and Changshui Zhang, Fellow, IEEE

Abstract—Backpropagation (BP) is widely used for calculating gradients in deep neural networks (DNNs). Applied often along with stochastic gradient descent (SGD) or its variants, BP is considered as a de-facto choice in a variety of machine learning tasks including DNN training and adversarial attack/defense. Recently, a linear variant of BP named LinBP was introduced for generating more transferable adversarial examples for performing black-box attacks, by (Guo et al. 2020). Although it has been shown empirically effective in black-box attacks, theoretical studies and convergence analyses of such a method is lacking. This paper serves as a complement and somewhat an extension to Guo et al. (2020) paper, by providing theoretical analyses on LinBP in neural-network-involved learning tasks, including adversarial attack and model training. We demonstrate that, somewhat surprisingly, LinBP can lead to faster convergence in these tasks in the same hyper-parameter settings, compared to BP. We confirm our theoretical results with extensive experiments.

Index Terms—Backpropagation, deep neural networks, optimization, convergence, adversarial examples, adversarial training.

I. INTRODUCTION

O

VER the past decade, the surge of research on deep neural networks (DNNs) has been witnessed. Powered with large-scale training, DNN-based models have achieved state-of-art performance in a variety of real-world applications. Tremendous amount of research has been conducted to improve the architecture [2], [3], [4], [5], [6] and the optimization [7], [8], [9], [10], [11] of DNNs.

The optimization of DNNs often involves stochastic gradient descent (SGD) [12] (that minimizes some training loss) and backpropagation (BP) [13] (for computing gradients of the loss function). In the work of Guo et al. published at NeurIPS 2020 [1], a different way of computing “gradients” was proposed. The method, i.e., linear BP (LinBP), keeps the forward pass of BP unchanged while skips the partial derivative of some of the non-linear activations in the backward pass. It was shown empirically that LinBP led to improved results in generating transferable adversarial examples for performing black-box attacks [14], in comparison with using the original BP. This paper was written as a complement and somewhat an extension to Guo et al.’s paper, in a way that we delve deep into the theoretical properties of LinBP. Moreover, we target two more applications of LinBP, i.e., white-box attack and model parameter training, to study the optimization convergence of learning with LinBP.

In this paper, we will provide convergence analyses for LinBP, and we will demonstrate that, in white-box adversarial attack scenarios, using LinBP can produce more deceptive adversarial examples (similar to the results in black-box scenarios as given in [1]), than using the standard BP in the same hyper-parameter settings. Plausible explanations of such fast convergence will be given. Similarly, we will also show theoretically that LinBP helps converge faster in training DNN models, if the same hyper-parameters are used for the standard BP and LinBP. Simulation experiments confirm our theoretical results, and extensive results will verify our findings in more general and practical settings using a variety of widely used DNNs, including VGG-16 [2], ResNet-50 [3], DenseNet-161 [4], MobileNetV2 [15], and WideResNet (WRN) [16].

The rest of this paper is organized as follows. In Section II, we will revisit preliminary knowledge about network training and adversarial examples, to introduce LinBP and discuss its applications. In Section III, we will provide theoretical analyses on the convergence of LinBP in both the white-box attack and model training scenarios. In Section IV, we will conduct extensive simulation and practical experiments to validate our theoretical findings and test LinBP with different DNNs. In Section V, we draw conclusions.

II. BACKGROUND AND PRELIMINARY KNOWLEDGE

A. Model Training

Together with SGD or its variant, BP has long been adopted as a default method for computing gradients in training machine

Authorized licensed use limited to: 162-8828. Downloaded on November 14,2023 at 08:35:22 UTC from IEEE Xplore. Restrictions apply.
learning models. Consider a typical update rule of SGD, we have

\[ w_{t+1} = w_t - \eta \nabla w_t \sum_k L(x^{(k)}, y^{(k)}) \quad (1) \]

where \( \{x^{(k)}, y^{(k)}\} \) contains the \( k \)-th training instance and its label, \( \nabla w \sum_k L(x^{(k)}, y^{(k)}) \) is the partial gradient of the loss function with respect to a learnable weight vector \( w \), and \( \eta \in \mathbb{R}^+ \) represents the learning rate.

**B. White-Box Adversarial Attack**

Another popular optimization task in deep learning is to generate adversarial examples, which is of particular interest in both the machine learning community and the security community. Instead of minimizing the prediction loss as in the objective of model training, this task aims to craft inputs that lead to arbitrary incorrect model predictions [17]. The adversarial examples are expected to be perceptually indistinguishable from the benign ones. Adversarial examples can be untargeted or targeted, and in this paper we focus on the former. In the white-box setting, where it is assumed that the architecture and parameters of the victim model are both known to the adversary, we have the typical learning objectives for generating adversarial examples:

\[ \max_{|r|_p \leq \epsilon} L(x + r, y) \quad (2) \]

and

\[ \min ||r||_p, \text{ s.t., } \arg \max_j \text{prob}_j(x + r) \neq \arg \max_j \text{prob}_j(x) \quad (3) \]

where \( r \) is a perturbation vector whose \( l_p \) norm is constrained (e.g., \( ||r||_p < \epsilon \)) to guarantee that the generated adversarial example \( x + r \) is perceptually similar to the benign example \( x \), function \( \text{prob}(. \text{)} \) provides the prediction probability for the \( j \)-th class. For solving the optimization problems in (2) and (3), a series of methods have been proposed over the last decade. For instance, with \( p = \infty \), i.e., for performing \( l_{\infty} \) attack, Goodfellow et al. [17] proposed the fast gradient sign method (FGSM) which simply calculates the sign of the input gradient, i.e., \( \text{sign}(\nabla_x L(x)) \), and adopted \( r = \epsilon \cdot \text{sign}(\nabla_x L(x)) \) as the perturbation. To enhance the power of the attack, follow-up methods proposed iterative FGSM (I-FGSM) [18] and PGD [19] that took multiple steps of update and computed input gradients using BP in each of the steps. The iterative process of these methods is similar to that of model training, except that the update is performed in the input space instead of the parameter space, and normally some steps are required for attacks. In transfer-based attacks, I-FGSM is widely adopted as a baseline method while it has been reported that PGD is capable of bypassing gradient masking [20]. Other famous attacks include DeepFool [21] and C&W’s attack [22], just to name a few.

**C. Linear Backpropagation**

Inspired by the hypothesis made by Goodfellow et al. [17] that the linear nature of modern DNNs causes the transferability of adversarial samples, LinBP, a method improves the linearity of DNNs, was proposed. For a \( d \)-layer DNN model, the forward pass includes the computation of

\[ f_d(x) = W_d^T \sigma (W_{d-1}^T \cdots W_1^T x) \quad (4) \]

where \( \sigma(\cdot) \) is the activation function and it is often set as the ReLU function, \( W_1, \ldots, W_d \) are learnable weight matrices in the DNN model. In order to generate a white-box adversarial example on the basis of \( x \) with \( f_d(x) \), we shall further compute the prediction loss \( L(x, y) \) for \( x \) and then backpropagate the loss to compute the input gradient \( \nabla_x L(x, y) \), the default BP for computing the input gradient is formulated as

\[ \nabla_x L(x, y) = M_1 \cdots M_{d-1} W_d \frac{\partial L(x, y)}{\partial f_d(x)} \quad (5) \]

where \( M_i = W_i \frac{\partial \sigma(f_i(x))}{\partial f_i(x)} \). LinBP keeps the computation in the forward pass unchanged while removing the influence of the ReLU activation function (i.e., \( \frac{\partial \sigma(f_i(x))}{\partial f_i(x)} \)) in the backward pass. That being said, LinBP computes:

\[ \tilde{\nabla}_x L(x, y) = W_1 \cdots W_d \frac{\partial L(x, y)}{\partial f_d(x)} \quad (6) \]

Since the partial derivative of the activation function has been removed, it is considered linear in the backward pass and thus called linear BP (LinBP). In practice, LinBP may only remove some of the nonlinear derivatives, e.g., only modifies those starting from the \( (m + 1) \)-th layer and uses the following formulation:

\[ \tilde{\nabla}_x L(x, y) = M_1 \cdots M_m W_{m+1} \cdots W_d \frac{\partial L(x, y)}{\partial f_d(x)} \quad (7) \]

Experimental results on CIFAR-10 [23] and ImageNet [24] tested the transferability of generated adversarial examples on different source models and found that LinBP achieved state-of-the-arts [1].

In addition to generating adversarial examples, we can also adopt LinBP to compute the gradient with respect to model weights for training DNNs. Note that the gradient of the loss with respect to \( W_i \) in the standard BP is formulated as

\[ \nabla_{W_i} L(x, y) = \sigma(f_{i-1}(x)) \left( \frac{\partial \sigma(f_i(x))}{\partial f_i(x)} M_{i+1} \cdots M_{d-1} W_d \frac{\partial L(x, y)}{\partial f_d(x)} \right)^T \quad (8) \]

where we abuse the notations and define \( \sigma(f_0(x)) = x \) for simplicity. Just like in (6), LinBP computes a linearized version of the gradient, which is formulated as

\[ \tilde{\nabla}_{W_i} L(x, y) = \sigma(f_{i-1}(x)) \left( W_{i+1} \cdots W_d \frac{\partial L(x, y)}{\partial f_d(x)} \right)^T \quad (9) \]

**III. Theoretical Analyses**

If we consider skipping ReLU in the backward pass as finding an approximation to the gradient of a non-smooth objective function, then LinBP is somewhat related to the straight-through estimation [25]. However, such a possible relation does not provide much insight of the convergence of LinBP, which is the
focus of this paper. In this section, we shall provide convergence analyses for LinBP, and compare it to the standard BP. There have been attempts for studying the training dynamics of neural networks [26], [27], [28], [29]. We follow Tian’s work [28] and consider a teacher-student framework with the ReLU activation function and squared $l_2$ loss, based on no deeper than two-layer networks. We shall start from theoretical analyses for white-box adversarial attacks and then consider model training.

### A. Theoretical Analyses for Adversarial Attack

Compared to the two-layer teacher-student frameworks in Tian’s work [28], we here consider a more general model, which can be formulated as

$$g(W, V, x) = V \sigma(Wx),$$  
(10)

where $x \in \mathbb{R}^{d_1}$ is the input data vector, $W \in \mathbb{R}^{d_2 \times d_1}$ and $V \in \mathbb{R}^{d_3 \times d_2}$ are weight matrices, $\sigma(\cdot)$ is the ReLU function which compares its inputs to 0 in an element-wise manner. In the teacher-student frameworks, we assume the teacher network possesses the optimal adversarial example $x^*$, and the student network aims to learn an adversarial example $x$ from the teacher network, in which the loss function is set as the squared $l_2$ loss between the output of the student and the teacher network, i.e.,

$$\mathcal{L}(x) = \frac{1}{2} \|g(W, V, x) - g(W, V, x^*)\|_2^2. \quad (11)$$

We further define $D(W, x) := \text{diag}(Wx) > 0$. From (10) and (11), we can easily obtain the analytic expression of the gradient with respect to $x$ for the standard BP:

$$\nabla_x \mathcal{L}(x) = W^T D(W, x) V^T (D(W, x) W x - D(W, x^*) W x^*). \quad (12)$$

Compared with (12), the gradient obtained from LinBP removes the derivatives of the ReLU function in the backward pass, i.e.,

$$\nabla_x \mathcal{L}(x) = W^T V^T V (D(W, x) W x - D(W, x^*) W x^*). \quad (13)$$

Inspired by Theorem 1 in Tian’s work [28], we introduce the following lemma to give the analytic expression of the gradients in expectations in adversarial settings.

**Lemma 1:** $G(e, x) := W^T D(W, e) V^T V D(W, x) W x$, where $e \in \mathbb{R}^{d_1}$ is a unit vector, $x \in \mathbb{R}^{d_1}$ is the input data vector, $W \in \mathbb{R}^{d_2 \times d_1}$ and $V \in \mathbb{R}^{d_3 \times d_2}$ are weight matrices. If $W$ and $V$ are independent and both generated from the standard Gaussian distribution, we have

$$\mathbb{E}(G(e, x)) = \frac{d_3 d_2}{2\pi} (\pi - \Theta) x + \|x\| \sin \Theta e,$$

where $\Theta \in [0, \pi]$ is the angle between $e$ and $x$.

The proof of Lemma 1 can be found in the appendices, available online. With Lemma 1, the expectation of (12) and (13) can be formulated as

$$\mathbb{E}[\nabla_x \mathcal{L}(x)] = G(x/\|x\|, x) - G(x/\|x\|, x^*)$$

$$= \frac{d_3 d_2}{2} (x - x^*) + \frac{d_3 d_2}{2\pi} (\Theta x^* - \|x\| \sin \Theta x) \quad (14)$$

and

$$\mathbb{E} [\nabla_x \mathcal{L}(x)] = G(x/\|x\|, x) - G(x^*/\|x^*\|, x^*)$$

$$= \frac{d_3 d_2}{2} (x - x^*), \quad (15)$$

respectively, where $\Theta \in [0, \pi]$ is the angle between $x$ and $x^*$. We here focus on $l_\infty$ attacks. Different iterative gradient-based attack methods may have slightly different update rules. Here, for ease of unified analyses, we simplify their update rules as

$$x^{(t+1)} = \text{Clip} \left( x^{(t)} - \eta \nabla_x \mathcal{L} \left( x^{(t)} \right) \right), \quad (16)$$

where $\text{Clip}(\cdot) = \min(x + \epsilon, \max(x - \epsilon, \cdot))$ performs element-wise input clip to guarantee that the intermediate results always stay in the range fulfilling the constraint of $\|x^{(t+1)} - x\|_\infty \leq \epsilon$. We use the updates of the standard BP and LinBP, i.e., $\nabla_x \mathcal{L}(x)$ and $\nabla_x \mathcal{L}(x)$, to obtain $\{x^{(t)}\}$ and $\{\hat{x}^{(t)}\}$, respectively. Based on all these results above, we can give the following theorem that provides convergence analysis of LinBP.

**Theorem 1:** For the two-layer teacher-student network formulated as in (10), in which the adversarial attack adopts (11) and (16) as the loss function and the update rule, respectively, we assume that $W$ and $V$ are independent and both generated from the standard Gaussian distribution, $x^* \sim N(\mu_1, \sigma_1^2)$, $x^{(0)} \sim N(0, \sigma_2^2)$, and $\eta$ is reasonably small.\(^1\) Let $x^{(t)}$ and $\hat{x}^{(t)}$ be the adversarial examples generated in the $t$-th iteration of attack using BP and LinBP, respectively, then we have

$$\mathbb{E}\|x^* - \hat{x}^{(t)}\|_1 \leq \mathbb{E}\|x^* - x^{(t)}\|_1.$$

The proof of the theorem is deferred to the appendices, available online. Theorem 1 shows that LinBP can produce adversarial examples closer to the optimal adversarial examples $x^*$ (when compared with the standard BP) in the same settings for any finite number of iteration steps, which means that LinBP can craft more powerful and destructive adversarial examples even in the white-box attacks. It is also straightforward to derive from our proof that LinBP produces a more powerful adversarial example, if starting optimization from the benign example with lower prediction loss. The conclusion is somewhat surprising since popular white-box attack methods like FGSM (I-FGSM) and PGD mostly use the gradient obtained by the standard BP, yet we find LinBP may lead to stronger attacks with the same hyper-parameters.

From our proof, it can further be derived that the norm of (15) is larger than the norm of (14), which means LinBP provides larger gradients in expectation. Also, the direction of the update obtained from LinBP is closer to the residual $x - x^*$ in comparison to the standard BP. These may cause LinBP to produce more destructive adversarial examples and more powerful attack in white-box settings. We conducted extensive experiments on

\(^1\)See (22) for more details of the constraint.
deeper networks using common attack methods to verify our theoretical findings. The experimental results will be shown in Section IV-B.

B. Theoretical Analyses for Model Training

For model training, we adopt the same framework as in Section III-A to analyze the performance of LinBP. By contrast, we mainly consider a one-layer teacher-student framework, since the student network may not converge to the teacher network with two-layer and deeper models. We will show in Section IV-B that many of our theoretical results still hold in more complex network architectures.

The one-layer network can be formulated as

$$h(x, w) = \sigma (x^T w),$$

where $x \in \mathbb{R}^d$ is the input vector, $w \in \mathbb{R}^d$ is the weight vector, and $\sigma(\cdot)$ is the ReLU function. Given a set of training samples, we obtain $h(X, w) = \sigma(Xw)$, where $X = [x_1^T; \ldots; x_N^T]$ is the input data matrix, $x_k \in \mathbb{R}^d$ is the $k$-th training instance, for $k = 1, \ldots, N$. We assume the teacher network has the optimal weight, and the loss function for optimizing the one-layer network can be formulated as

$$\mathcal{L}(w) = \frac{1}{2} \|h(X, w) - h(X, w^*)\|^2,$$

where $w$ and $w^*$ are the weight vectors for the student network and the teacher network, respectively. Therefore, we can derive the gradient with respect to $w$ for the standard BP as:

$$\nabla_w \mathcal{L}(w) = X^T D(X, w)(D(X, w)Xw - D(X, w^*)Xw^*).$$

(19)

By contrast, the gradient with respect to $w$ obtained from LinBP is formulated as

$$\nabla_w \mathcal{L}(w) = X^T(D(X, w)Xw - D(X, w^*)Xw^*).$$

(20)

While training the simple one-layer network with SGD, the update rule can be formulated as

$$w(t+1) = w(t) - \eta \nabla_w \mathcal{L}(w(t)),$$

(21)

where we use $\nabla_w \mathcal{L}(w)$ and $\nabla_w \mathcal{L}(w)$ showed in (19) and (20) to obtain $\{w(t)\}$ and $\{w(t)\}$ for the standard BP and LinBP, respectively. The following theorem is given for analyzing the training convergence, when LinBP is used.

**Theorem 2:** For the one-layer teacher-student network formulated as in (17), in which training adopts (18) and (21) as the loss function and the update rule, respectively, we assume that $X$ is generated from the standard Gaussian distribution, $w^* \sim N(\mu_1, \sigma_1^2), w^{(0)} \sim N(0, \sigma_2^2)$, and $\eta$ is reasonably small. $^2$

Let $w(t)$ and $w(t)$ be the weight vectors obtained in the $t$-th iteration of training using standard BP and LinBP, respectively, then we have

$$E\|w^* - w(t)\|_1 \leq E\|w^* - w(t)\|_1.$$

The proof of Theorem 2 can also be found in the appendices, available online. Theorem 2 shows that LinBP may lead the obtained weight vector to get closer toward the optimal weight vector $w^*$ compared to the standard BP with the same learning rate and the same number of training iterations, which means LinBP can also lead to faster convergence in model training.

C. Discussions About $\eta$ and $t$

In Theorems 1 and 2, we assume that the update step size in adversarial attacks and the learning rate in model training are reasonably small. Here we would like to discuss more about these assumptions.

Precisely, we mean that $\eta$ should be small enough to satisfy the following constraints: for Theorem 1, it is required that

$$\sum_{j=0}^{m-1} \eta \frac{d^2}{d\theta^2} \left(1 - \frac{\eta d^2}{2}\right)^{m-1-j} p_{ji} \leq \frac{1 - \frac{\eta d^2}{2}}{m}\left|x_i^* - x_i^{(0)}\right|,$$

(22)

for $m = 1, \ldots, t$, and $i = 1, \ldots, d_1$, where $p_{ij} = \theta_j x^* - \frac{\|x^*\|_1}{\|x^{(0)}\|_1} \sin \theta_j x^{(j)}$ and $j$ indicates the $i$-th entry of a $d_1$-dimensional vector. For Theorem 2, it is required that

$$\sum_{j=0}^{m-1} \eta N \left(1 - \frac{\eta N}{2}\right)^{m-1-j} q_{ji} \leq \frac{1 - \frac{\eta N}{2}}{m}\left\|w_i^* - w_i^{(0)}\right\|,$$

(23)

for $m = 1, \ldots, t$, and $i = 1, \ldots, d$, where $q_{ij} = \theta_j w^* - \frac{\|w^*\|_1}{\|w^{(0)}\|_1} \sin \theta_j w^{(j)}$. It is nontrivial to obtain analytic solutions for (22) and (23). However, we can know that they are both more likely to hold when $t$ is small. Since the maximum number of learning iterations for generating adversarial examples is often set to be small (at most several hundred) in commonly adopted methods like I-FGSM and PGD, the assumption in (22) is easier to be fulfilled than that in (23), i.e., in the setting of model training which can take tens of thousands of iterations, indicating that the superiority of LinBP can be more obvious in performing adversarial attacks.

IV. EXPERIMENTAL RESULTS

In this section, we provide experimental results on synthetic data (see Section IV-A) and real data (see Section IV-B) to confirm our theorem and compare LinBP against BP in more practical settings for white-box adversarial attack/defense and model training, respectively. The practical experiments show that our theoretical results hold in a variety of different model architectures. All the experiments were performed on NVIDIA GeForce RTX 1080/2080 Ti and the code was implemented using PyTorch [30].

---

$^2$See (23) for a precious formulation of the constraint.
We constructed a two-layer neural network and performed adversarial attack following the teacher-student framework described in Section III-A. To be more specific, the victim model is formulated as (10), the loss function is given as (11), and (16) is the update rule to generate adversarial examples. Following the assumption in Theorem 1, the weight matrices $V$ and $W$ were independent and both generated from the standard Gaussian distribution. Similarly, the optimal adversarial example $x^*$ and the initial adversarial example $x^{(0)}$ were obtained via sampling from Gaussian distributions, i.e., $x^* \sim N(\mu_1, \sigma_1^2)$ and $x^{(0)} \sim N(0, \sigma_2^2)$, where $\mu_1, \sigma_1$, and $\sigma_2$ can in fact be arbitrary constants. Here we set $\mu_1 = 1.0, \sigma_1 = 2.0$, and $\sigma_2 = 1.0$. In our experiments, we set $d_1 = 100, d_2 = 20, d_3 = 10, \eta = 0.001, \epsilon = 0.25$. And the maximum number of the iteration step was set to 100. We randomly sampled 10 sets of parameters (including weight matrices, optimal adversarial examples, and initial adversarial examples) for different methods and evaluate the average performance over 10 runs of the experiment. The average $l_1$ distance between the obtained student adversarial examples and the optimal adversarial examples from the teacher network is shown in Fig. 1. It shows that LinBP makes the obtained adversarial examples converge faster to the optimal ones, indicating that LinBP helps craft more powerful adversarial examples in the same hyper-parameter settings, which clearly confirms our Theorem 1.

As we have mentioned in Section III, the norm of the update obtained from LinBP is larger than that obtained from the standard BP, which may cause faster convergence for LinBP. In this experiment, the $l_2$ norm and $l_{\infty}$ norm of the gradient obtained using LinBP are $\sim 1.65 \times 100$ and $\sim 1.60 \times 100$, respectively. For deeper discussions, we have conducted two more experiments. We used $l_2$ norm and $l_{\infty}$ norm to normalize the update obtained from LinBP and standard BP. $\eta$ was set as 0.05 and 0.005, respectively. The results are shown in Fig. 2(a) and (b), where we can find the conclusion still holds with gradient normalization.

2) Model Training: As described in Section III-B, we constructed the one-layer neural network formulated as in (17) and trained it following the student-teacher framework. Equation (18) is the loss function. Similar to the experiments in Section IV-A1, the input data matrix $X$ was generated from the standard Gaussian distribution, the optimal weight vector $w^*$ in the teacher network and the initial weight vector $w^{(0)}$ in the student network were obtained via sampling from Gaussian distributions with $\mu_1 = 1.0, \mu_2 = 0.0, \sigma_1 = 3.0$, and $\sigma_2 = 1.0$. We used SGD for optimization and set $N = 100, d = 10$, and $\eta = 0.001$ in our experiments. The maximal number of optimization iterations was set to 10000 to ensure training convergence. We used the $l_1$ distance between the weight vector in the teacher model and that in the student model to evaluate their difference. Fig. 3 illustrates the average results over 10 runs and compares the two methods. From the experimental results, we can observe that LinBP leads to more accurate approximation (i.e., smaller $l_1$ distance) to the teacher as well as lower training loss in comparison with BP, which clearly confirms our Theorem 2, indicating that LinBP can lead to faster convergence in the same hyper-parameter settings.

We found that the norm of gradient obtained from LinBP is larger than that obtained from the standard BP in the early stage of training, similar to the results in Section IV-A1. To be specific, the $l_2$ norm of the gradient from LinBP is $\sim 2.2 \times 100$ larger than that from BP in the first 100 iterations, yet over all 10000 iterations, it is only $\sim 0.75 \times 100$. We have also conducted the experiment where the gradient is normalized by its $l_2$ norm. $\eta$ was set to 0.0015. The results are shown in Fig. 4. We see that the superiority of LinBP drops a lot (when compared with Fig. 3), which is different from the results obtained in adversarial attacks.
TABLE I

| Method | $K$ | $\epsilon$ | VGG-16 | ResNet-50 | DenseNet-161 | MobileNetV2 | WRN (robust) |
|--------|-----|-------------|--------|-----------|--------------|-------------|--------------|
| BP     | 10  | 8/255      | 72.28% | 66.41%    | 56.89%       | 92.94%      | 33.14%       |
|        | 20  | 8/255      | 87.07% | 80.82%    | 73.33%       | 98.19%      | 38.11%       |
|        | 100 | 8/255      | 96.54% | 92.53%    | 86.46%       | 99.88%      | 38.94%       |
|        | 10  | 16/255     | 77.25% | 71.52%    | 62.79%       | 94.90%      | 33.78%       |
|        | 20  | 16/255     | 94.88% | 91.18%    | 87.38%       | 99.44%      | 44.25%       |
|        | 100 | 16/255     | 99.73% | 99.81%    | 98.33%       | 100.00%     | 59.38%       |
| LinBP  | 10  | 8/255      | 95.52% | 85.83%    | 61.87%       | 99.80%      | 33.48%       |
|        | 20  | 8/255      | 98.50% | 93.11%    | 77.72%       | 99.95%      | 39.55%       |
|        | 100 | 8/255      | 99.25% | 95.82%    | 89.03%       | 99.99%      | 40.44%       |
|        | 10  | 16/255     | 98.78% | 92.68%    | 96.04%       | 99.95%      | 34.21%       |
|        | 20  | 16/255     | 99.99% | 99.51%    | 91.18%       | 100.00%     | 45.22%       |
|        | 100 | 16/255     | 100.00%| 99.97%    | 99.07%       | 100.00%     | 60.78%       |

$K$ is the maximum allowed number of update steps and $\epsilon$ is the perturbation budget. Higher success rate (i.e., lower prediction accuracy on the adversarial examples) indicates more powerful attack. "WRN (robust)" represents a robust WRN-28-10 model [31] available on the RobustBench [32].

B. More Practical Experiments

We also conducted experiments in more practical settings for adversarial attack and model training on MNIST [33] and CIFAR-10 [23], using DNNs with a variety of different architectures, including a simple MLP, LeNet-5 [33], VGG-16 [2], ResNet-50 [3], DenseNet-161 [4], MobileNetV2 [15], and WRN (WRN) [16].

1) Adversarial Attack on DNNs: We performed white-box attacks on CIFAR-10 using different DNNs, including VGG-16, ResNet-50, DenseNet-161, MobileNetV2, and a robust WRN-28-10 [31] available on the RobustBench [32]. CIFAR-10 test images that could be correctly classified by these models were used to generate adversarial examples, and the attack success rate was adopted to evaluate the performance of the attack. We evaluate the results of PGD [19] as it is popular in white-box attacks, and we report the best attack performance over 5 restarts of PGD. Note that, without restart, PGD and I-FGSM [18] achieve similar performance. The gradient steps were normalized by the $l_\infty$ norm in PGD by default. The update step size of the iterative attack $\eta$ was fixed to be 1/255. We tested different settings of the maximum allowed number of update steps $K$ and the perturbation budget $\epsilon$, including $K \in \{10, 20, 100\}$ and $\epsilon \in \{8/255, 16/255\}$. The attack performance using LinBP and BP following all these settings is summarized in Table I.

It is easy to see in Table I that LinBP gains consistently higher attack success rates with PGD, showing that it can help generate more powerful adversarial examples. For instance, with $K = 10$ and $\epsilon = 8/255$, using LinBP improves the success rate of attacking ResNet-50 by 19.42% (85.83% versus 66.41%). With $K = 100$, the success rates of attacking some (non-robust) models approach 100%, and LinBP still helps achieve similar or higher attack performance than using BP. As gradient normalization was by default adopted in PGD, the superiority of LinBP has nothing to do with the norm of gradient, just as in Section IV-A1.

We have also tried using Auto-PGD [34], which was developed as a stronger method for evaluating the adversarial robustness of models. Similar to the experiment with PGD, the best attack performance over 5 restarts are shown (see Table II). It can be observed that LinBP is still more effective than BP, even with Auto-PGD which has adaptive step sizes. To give more discussions on how LinBP enhances the performance in Auto-PGD, we carefully analyzed when the optimizer adjusted its step sizes with LinBP/BP. The results show that LinBP helps Auto-PGD halve the step size faster. More specifically, the average number of iterations before Auto-PGD halves its step size using LinBP is 22.95, 42.18, 58.59, and 72.11, compared with 25.85, 45.11, 61.98 and 77.45 for BP when attacking ResNet-50 with $\epsilon = 8/255$. Since Auto-PGD adjusts its step size when it reaches “convergence” with the current step size, the results show that LinBP is beneficial to convergence even with such a method in practice.

LinBP can be effective with other activation functions. To gain more insight, we further tested with the Swish/SiLU activation function. For simplicity of experiments, we replace the original ReLU activations in ResNet-50 with the Swish/SiLU activations. With $K = 10$ and $\eta = 1/255$, the PGD attack success rate for LinBP are 75.85% and 87.97% when $\epsilon = 8/255$ and $\epsilon = 16/255$, respectively, compared with 74.24% and 82.65% for BP.

2) Training DNNs: For experiments of DNN training, we first trained and evaluated a simple MLP and LeNet-5 on MNIST. The MLP has four parameterized and learnable layers, and the numbers of its hidden layer units are 400, 200, and 100. We used SGD for optimization and the learning rate was 0.001 and 0.005 for the MLP and LeNet-5, respectively. The training batch size was set to 64, and the training process lasted for at most 50 epochs. The training loss and training accuracy are illustrated in Fig. 5, and note that we fix the random seed to eliminate unexpected randomness during training. We can easily observe from the training curves in Fig. 5 that the obtained MLP and LeNet-5 models show lower prediction loss and higher accuracy when incorporating LinBP, especially in the early epochs, which suggests that the incorporation of LinBP can be beneficial to the convergence of SGD. The same observation can be made on the test set of MNIST, and the results are shown in Fig. 6.
**TABLE II**

| Method   | ε         | VGG-16  | ResNet-50 | DenseNet-161 | MobileNetV2 | WRN (robust) |
|----------|-----------|---------|-----------|--------------|-------------|--------------|
| BP       | 8/255     | 98.06%  | 95.77%    | 91.91%       | 99.97%      | 39.46%       |
|          | 16/255    | 99.97%  | 99.98%    | 99.58%       | 100.00%     | 60.94%       |
| LinBP    | 8/255     | 99.65%  | 97.60%    | 93.35%       | 100.00%     | 41.20%       |
|          | 16/255    | 100.00% | 99.99%    | 99.81%       | 100.00%     | 62.99%       |

ε is the perturbation budget. Higher success rate (i.e., lower prediction accuracy on the adversarial examples) indicates a more powerful attack. "WRN(robust)" represents a robust WRN-28-10 model [31] available on the Robustbench [32].

We further report our experimental results on CIFAR-10. ResNet-50 [3], DenseNet-161 [4], and MobileNetV2 [15] were trained and evaluated on the dataset. The architecture of these networks and their detailed settings can be found in their papers. The optimizer was SGD, and the learning rate was 0.002. We set the batch size to 128 and trained for 100 epochs. We evaluated the prediction loss and prediction accuracy of trained models using LinBP and BP for comparison. Similar to the experiment on MNIST, we fixed the random seed. The training results in Fig. 7 and test results in Fig. 8 demonstrate that, equipped with LinBP, the models achieved lower prediction loss and higher prediction accuracy in the same training settings, especially when the training just got started. Nevertheless, as training progressed, the superiority of LinBP became less obvious, and the final performance of LinBP became just comparable to that of the standard BP, which is consistent with our theoretical discussions in Section III-B.

We also noticed that in certain scenarios, optimization using LinBP may fail to converge with very large learning rates. Here we test various learning rate \( \eta \) in training MNIST models to observe the stability of LinBP. Recall that the base learning rates for training the MLP and LeNet-5 are 0.001 and 0.005, respectively, for obtaining our previous results, and we further tried scaling them by \( 10 \times \) and \( 0.1 \times \) to investigate how the training performance was affected. The training curves for all these settings of learning rates are illustrated in Fig. 9. Compared with the results in Fig. 5(b), it can be seen that, when increasing the base learning rate by \( 10 \times \), the training performance of LinBP and that of BP are similar on the MLP, but, on LeNet-5, the training failed to converge with LinBP while it could still converge with BP. Also, the same observation was made when training VGG-16 on CIFAR-10 using LinBP and a (relatively large) base learning rate of 0.005. The results conform that a reasonably small \( \eta \) helps stabilize training using LinBP and guarantees its performance. We also tried using normalized gradients during training and LinBP performed no better than BP, just like the results in Section IV-A2.

We used the cosine annealing learning rate scheduler [9] for obtaining the results in Figs. 7 and 8. One may also be
curious about the training performance with other learning rate scheduler. In this context, we further tested with exponential learning rate scheduler \( 10^{-4} \) with \( \gamma = 0.98 \) and find similar results, which are shown in Fig. 10.

3) Adversarial Training for DNNs: Since the discovery of adversarial examples, the vulnerability of DNNs has been intensively discussed. Tremendous effort has been devoted to improving the robustness of DNNs. Thus far, a variety of methods have been proposed, in which adversarial training has become an indispensable procedure in many application scenarios. In this context, we would like to study how LinBP can be adopted to further enhance the robustness of DNNs. We used the adversarial examples generated by PGD to construct our dataset and we trained our model using simply SGD. There exist multiple strategies of adopting LinBP in adversarial training, i.e., 1) computing gradients for updating model parameters using LinBP as in the model training experiment and 2) generating adversarial examples using LinBP just like in the adversarial attack experiment. We observed that the first strategy is more beneficial than the second strategy when evaluating the robustness of DNNs using BP-generated adversarial examples. Due to overfitting, generating adversarial examples using LinBP during adversarial training leads to obviously larger generalization gap between training and test performance and degraded test robustness against BP-generated adversarial examples.

Though effective in the sense of achieving high robust accuracy after convergence, the first strategy sometimes suffers from unstable performance as the adversarial examples and model parameters are updated asynchronously. Further applying the second strategy in combination with the first strategy stabilize adversarial training and is helpful to achieve reliable performance. See Figs. 11 and 12 for performance under the PGD attack and AutoAttack, respectively. In the figures, we applied the classification accuracy on adversarial examples to evaluate the robustness. The experiment was performed on CIFAR-10 using MobileNetV2 and ResNet-50, where the attack step size was \( 2/255 \), \( \epsilon = 8/255 \), \( K = 5 \), and the training learning rate was 0.01.

The figures demonstrate that, benefit from its fast convergence, LinBP can be used as an alternative to BP when performing adversarial training for achieving robustness (if used appropriately).

V. Conclusion

In this paper, we have studied the convergence of optimization using LinBP, which skips ReLUs during the backward pass, and compared it to optimization using the standard BP. Theoretical analyses have been carefully performed in two popular application scenarios, i.e., white-box attack and model training. In addition to the benefits in black-box attacks which have been demonstrated in [1], we have shown in this paper that LinBP also leads to generating more destructive white-box adversarial examples and obtaining faster model training. Experimental results using synthesized data confirm our theoretical results. Extensive experiments on MNIST and CIFAR-10 further show that the theoretical results hold in more practical settings on a variety of DNN architectures.

REFERENCES

[1] Y. Guo, Q. Li, and H. Chen, “Backpropagating linearly improves transferability of adversarial examples,” in Proc. Int. Conf. Neural Inf. Process. Syst., 2020, Art. no. 8.
[2] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in Proc. Int. Conf. Learn. Representations, 2015.
[3] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 770–778.
[4] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. 2016, pp. 770–778.
[5] A. Vaswani et al., “Attention is all you need,” 2017, arXiv: 1706.03762.
[6] A. Dosovitskiy et al., “An image is worth 16x16 words: Transformers for image recognition at scale,” in Proc. Int. Conf. Learn. Representations, 2021.
[7] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2014, arXiv: 1412.6980.
[8] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” in Proc. Int. Conf. Learn. Representations, 2019.
[9] I. Loshchilov and F. Hutter, “SGDR: Stochastic gradient descent with warm restarts,” in Proc. Int. Conf. Learn. Representations, 2017.
[10] S. J. Reddi, S. Kale, and S. Kumar, “On the convergence of Adam and beyond,” 2019, arXiv: 1904.09237.
I. Sutskever, J. Martens, G. Dahl, and G. Hinton, “On the importance of initialization and momentum in deep learning,” in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. III-1139–III-1147.

L. Bottou, “Large-scale machine learning with stochastic gradient descent,” in *Proc. 19th Int. Conf. Comput. Statist.*, Springer, 2010, pp. 177–186.

Y. LeCun, “A theoretical framework for back-propagation,” in *Proc. Connectionist Models Summer Sch.*, 1988, pp. 21–28.

N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, “Practical black-box attacks against machine learning,” in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2017, pp. 506–519.

M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobieleNetV2: Inverted residuals and linear bottlenecks,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.

S. Zagoruyko and N. Komodakis, “Wide residual networks,” 2016, arXiv:1605.07146.

I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” in *Proc. Int. Conf. Learn. Representations*, 2015.

A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial machine learning at scale,” in *Proc. Int. Conf. Learn. Representations*, 2017.

A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards deep learning models resistant to adversarial attacks,” in *Proc. Int. Conf. Learn. Representations*, 2018.

A. Athalye, N. Carlini, and D. Wagner, “Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples,” in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 274–283.

S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, “DeepFool: A simple and accurate method to fool deep neural networks,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2574–2582.

N. Carlini and D. Wagner, “Towards evaluating the robustness of neural networks,” in *Proc. IEEE Symp. Secur. Privacy*, 2017, pp. 39–57.

A. Krizhevsky et al., “Learning multiple layers of features from tiny images,” Univ. Toronto, 2009.

O. Russakovsky et al., “ImageNet large scale visual recognition challenge,” *Int. J. Comput. Vis.*, vol. 115, pp. 211–232, 2015.

Y. Bengio, N. Léonard, and A. Courville, “Estimating or propagating gradients through stochastic neurons for conditional computation,” 2013, arXiv:1308.3432.

S. Du, J. Lee, H. Li, L. Wang, and X. Zhai, “Gradient descent finds global minima of deep neural networks,” in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 1675–1685.

S. Arora, S. Du, W. Hu, Z. Li, and R. Wang, “Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks,” in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 322–332.

Y. Tian, “An analytical formula of population gradient for two-layered ReLU network and its applications in convergence and critical point analysis,” in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 3404–3413.

S. S. Du, X. Zhai, B. Poczos, and A. Singh, “Gradient descent provably optimizes over-parameterized neural networks,” in *Proc. Int. Conf. Learn. Representations*, 2019.

A. Paszke et al., “PyTorch: An imperative style, high-performance deep learning library,” in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 8024–8035.

J. Zhang, J. Zhu, G. Niu, B. Han, M. Sugiyama, and M. Kankanhalli, “Geometry-aware instance-reweighted adversarial training,” in *Proc. Int. Conf. Learn. Representations*, 2021.

F. Croce et al., “RobustBench: A standardized adversarial robustness benchmark,” 2020, arXiv:2010.09670.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

F. Croce and M. Hein, “Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks,” in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 2206–2216.

Z. Li and S. Arora, “An exponential learning rate schedule for deep learning,” 2019, arXiv:1910.07454.

Ziang Li received the BE degree from Tsinghua University, Beijing, China, in 2019. He is currently working toward the PhD degree with the Department of Automation, Tsinghua University. His current research interests include pattern recognition and machine learning.

Yiwen Guo received the BE degree from Wuhan University, in 2011, and the PhD degree from Tsinghua University, in 2016. He is a research scientist with ByteDance AI Lab, Beijing. Prior to this, he was a staff research scientist with Intel Labs China. His current research interests include computer vision, pattern recognition, and machine learning.

Haodi Liu received the BA degree from Bard College and the BS degree from Columbia University, in 2018 (Dual degree program), and the MS degree from Columbia University, in 2020. Currently, he is currently working toward the PhD degree with the Department of Automation, Tsinghua University. Prior to this, he was an algorithm engineer with AiBee Inc. His current research interests include pattern recognition, machine learning, and computer vision.

Changshui Zhang (Fellow, IEEE) received the BS degree in mathematics from Peking University, Beijing, China, in 1986, and the MS and PhD degrees in control science and engineering from Tsinghua University, Beijing, in 1989 and 1992, respectively. In 1992, he joined the Department of Automation, Tsinghua University, where he is currently a professor. His current research interests include pattern recognition and machine learning. He has authored more than 200 articles.