TRAINING DEEP NEURAL NETWORKS VIA OPTIMIZATION OVER GRAPHS

Guoqiang Zhang

Center for Audio, Acoustics and Vibration (CAAV)
School of Electrical and Data Engineering
University of Technology Sydney, Australia
Email: guoqiang.zhang@uts.edu.au

W. Bastiaan Kleijn

School of Engineering and Computer Science
Victoria University of Wellington, New Zealand
Email: bastiaan.kleijn@ecs.vuw.ac.nz

ABSTRACT

In this work, we propose to train a deep neural network by distributed optimization over a graph. Two nonlinear functions are considered: the rectified linear unit (ReLU) and a linear unit with both lower and upper cutoffs (DCutLU). The problem reformulation over a graph is realized by explicitly representing ReLU or DCutLU using a set of slack variables. We then apply the alternating direction method of multipliers (ADMM) to update the weights of the network layer-wise by solving subproblems of the reformulated problem. Empirical results suggest that the ADMM-based method is less sensitive to overfitting than the stochastic gradient descent (SGD) and Adam methods.

Index Terms— Deep learning, DNN, optimization, ADMM

1. INTRODUCTION

In the last decade, research on deep learning has made remarkable progress both in theoretical understanding and in practical applications (see [1] for an overview). Deep learning interprets data at multiple levels of abstraction, realized in a computational model with multiple processing layers. Each layer is composed of a set of simple nonlinear processing units (referred to as *neurons*), which aims to transform the input into progressively more abstract representations [2, 3]. With the composition of multiple processing layers, the model is able to produce data representations that are required by various applications.

In the literature, different types of deep neural networks (DNNs) have been proposed and applied in different applications. For instance, feedforward neural networks have been successfully applied in speech recognition [4, 5]. Convolutional neural networks (CNNs) are popular in computer vision [6, 7]. Recurrent neural networks (RNNs) have proven to be effective for mapping sequential inputs and outputs [8, 9].

The common procedure for training a deep neural network is to iteratively adjust its parameters (referred to as *weights*) such that the network approximates the input-output relations with increasing accuracy, referred to as *supervised learning*.

The traditional supervised learning approach treats a neural network as a large complex model [1] rather than decomposing it as a combination of many small nonlinear models. *Stochastic gradient descent (SGD)* is usually used to back-propagate gradients from the top layer down to the bottom layer on a mini-batch and then adjusts the weights accordingly. In recent years, various advanced methods have been proposed to use the gradient information smartly for either fast convergence or automatic parameter adjustment, such as Adam [10], AdaGrad [11] and RMSprop [12].

Recently, a new supervised learning paradigm has been proposed that decomposes a neural network as a combination of many small nonlinear models. In [13], the authors firstly proposed to decouple the nested structure of DNNs by introducing a set of auxiliary variables and a set of equality constraints. However, computation of the gradient is still required in their work to tackle the nonlinear functions of the neurons. The work of [14] avoids the gradient computation of [13] by using the alternating direction method of multipliers (ADMM) [15]. One Lagrangian multiplier is introduced in [14] to account for an equality constraint at the top layer. The other equality constraints introduced by other layers are only approximated by imposing large quadratic penalty functions. The resulting updating procedure is more close to the Bregman iteration than the standard ADMM.

In this paper, we propose to train a deep neural network by reformulating the problem as an optimization over a factor graph $\mathcal{G} = (\mathcal{V}, \mathcal{C})$ [16, 17]. Every node $r \in \mathcal{V}$ carries a convex function of its node variable while every factor $c \in \mathcal{C}$ carries a nonlinear equality constraint in terms of the node variables connected to the factor. Our graphic formulation is able to handle rectified linear units (ReLUs) (see [18, 19]) and linear units with both upper and lower cutoffs (DCutLUs) at the layer-level. In particular, the ReLUs or DCutLUs are represented in terms of a set of slack variables, which lead to the equality constraints in the factor graph.

We apply ADMM to solve the graph-based problem. Differently from [14], we introduce a set of Lagrangian multipliers, one for each equality constraint extracted from one layer. Experimental results on the MNIST dataset demonstrate that the new training method is less sensitive to overfitting than the SGD and Adam methods. Further, the performance of the new method on the test data is better than the SGD and Adam, which may be due to the flexibility of ADMM.

2. ON TRAINING A DEEP NEURAL NETWORK

Suppose we have a sequence of m training samples, represented by an input matrix $D \in \mathbb{R}^{m \times n_{in}}$ and an output matrix $O \in \mathbb{R}^{m \times n_{out}}$, where the q'th row-vectors of D and O form an input-output pair. Given (D, O), we consider training a deep neural network with the weights $\{(W_i, b_i) | i = 1, \ldots, N\}$ of N layers, where for each i, $W_i \in \mathbb{R}^{n_{i-1} \times n_i}$ is a weight matrix and $b_i \in \mathbb{R}^{1 \times n_i}$ a bias vector. To match the network with the training samples, we let $n_0 = n_{in}$ and $n_N = n_{out}$. The objective is to find the proper weights $\{(W_i, b_i)\}$ so that the network maps the input D to the output O as accurately as possible.

Let us now define the operation of the individual layers. We use V_i to denote the output of layer i, $i \leq N$. We let e be a (column) vector of ones. V_i, $i \leq N-1$, is obtained by performing (element-wise)

nonlinear operation on the matrix product $V_{i-1}W_i + eb_i$, denoted as $V_i = h_i(V_{i-1}W_i + eb_i)$. The popular forms for the nonlinear function h_i are sigmoid, tanh and ReLU [1]. It is found in [19] that ReLU leads to fast convergence using SGD as compared to sigmoid and tanh. We consider ReLU and DCutLU in this paper. Formally, we define h_i in the form of DCutLU as

$$V_i = \min(\max(V_{i-1}W_i + eb_i, l), u) \quad i \leq N-1, \tag{1}$$

where the $\max$ and $\min$ operators are element-wise, and l and u are the lower and upper threshold, respectively. ReLU is a special case of DCutLU by letting $(l, u) = (0, \infty)$.

The procedure of training the above neural network can be formulated as

$$\min_{\{V_i, W_i, b_i\}} \left[f_N(V_N; O) + \sum_{i=1}^{N} g_i(W_i, b_i) \right], \tag{2}$$

where f_N measures the difference between the output V_N and the ground truth O, g_i is a penalty function on (W_i, b_i), and $\{V_i, W_i, b_i\}$ satisfies (1) and

$$V_N = V_{N-1}W_N + eb_N. \tag{3}$$

3. PROBLEM REFORMULATION ONTO A GRAPH

In this section, we reformulate (2)-(3) as an optimization over a factor graph. We first represent the nonlinear function (1) by introducing a set of slack variables. Specifically, (1) can be rewritten as

$$X_i = V_{i-1}W_i + eb_i \tag{4}$$
$$X_i + Y_i = \max(V_{i-1}W_i + eb_i, l) \tag{5}$$
$$X_i + Y_i + Z_i = V_i = \min(X_i + Y_i, u), \tag{6}$$

where for each $i \in \{1, \ldots, N-1\}$, we introduced three slack matrices X_i, Y_i and Z_i to characterize the effect of the upper and lower cutoffs of the function at u and l.

Next, we argue that the $\min$ and $\max$ operators in (5)-(6) can be expressed in terms of constraints on (X_i, Y_i, Z_i). To do so, we introduce two index sets for each layer i:

$$\Omega_i^l = \{(q, j) | x_{i,qj} < l\} \tag{7}$$
$$\Omega_i^u = \{(q, j) | x_{i,qj} > u\}, \tag{8}$$

where $x_{i,qj}$ is the (q, j) element of X_i. At the moment, one can think of Ω_i^l and Ω_i^u as two sets that are preset already, imposing constraints on X_i. We will explain later how to update (Ω_i^l, Ω_i^u) iteratively. Given a set Ω, we let $\mathcal{P}_\Omega(X)$ denote the subset of the elements of X specified by the indices of Ω. The $\max$ operator in (5) can be characterized as

$$\mathcal{P}_{\Omega_i^l}(X_i) < l \tag{9a}$$
$$\mathcal{P}_{\Omega_i^l}(X_i) + \mathcal{P}_{\Omega_i^l}(Y_i) = l \tag{9b}$$
$$\mathcal{P}_{\bar{\Omega}_i^l}(X_i) \geq l \tag{9c}$$
$$\mathcal{P}_{\bar{\Omega}_i^l}(Y_i) = 0, \tag{9d}$$

where $\bar{\Omega}$ denotes the complement of Ω. By inspection of (5) and (6), we conclude that Y_i and Z_i are decoupled given X_i. The $\min$ operator in (6) can thus be characterized as

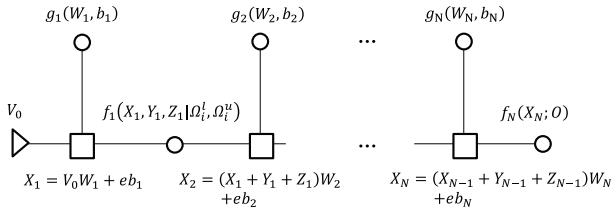

Fig. 1. Problem reformulation over a factor graph $\mathcal{G} = (\mathcal{V}, \mathcal{C})$. $\circ$ is a node in $\mathcal{V}$ and $\square$ is a factor in $\mathcal{C}$. $\triangleright$ represents constant inputs to the graph, where $X_0 = V_0 = D$ is the data input.

$$\mathcal{P}_{\Omega_i^u}(X_i) > u \tag{10a}$$
$$\mathcal{P}_{\Omega_i^u}(X_i) + \mathcal{P}_{\Omega_i^u}(Z_i) = u \tag{10b}$$
$$\mathcal{P}_{\bar{\Omega}_i^u}(X_i) \leq u \tag{10c}$$
$$\mathcal{P}_{\bar{\Omega}_i^u}(Z_i) = 0. \tag{10d}$$

To briefly summarize, we use the constraints (9) and (10) to replace the $\min$ and $\max$ operations in (5)-(6).

Based on the above analysis, the training problem (2)-(3) can be reformulated as

$$\min_{\substack{\{W_i, b_i \\ X_i, Y_i, Z_i\}}} f_N(X_N; O) + \sum_{i=1}^{N} g_i(W_i, b_i) + \sum_{i=1}^{N-1} f_i(X_i, Y_i, Z_i | \Omega_i^l, \Omega_i^u) \tag{11}$$
$$\text{s. t. } X_i = (X_{i-1} + Y_{i-1} + Z_{i-1})W_i + eb_i \quad \forall i = 1, \ldots, N, \tag{12}$$

where $(X_0, Y_0, Z_0) = (V_0, 0, 0)$ and each $f_i(X_i, Y_i, Z_i | \Omega_i^l, \Omega_i^u)$ can be taken as a summation of indicator functions, each defined by one constraint in (9)-(10), given by

$$\begin{aligned} f_i(X_i, Y_i, Z_i | \Omega_i^l, \Omega_i^u) = \Big[& 1_{\mathcal{P}_{\Omega_i^l}(X_i) < l} + 1_{\mathcal{P}_{\Omega_i^l}(Y_i) = 0} + 1_{\mathcal{P}_{\bar{\Omega}_i^l}(X_i) \geq l} \\ & + 1_{\mathcal{P}_{\Omega_i^l}(X_i) + \mathcal{P}_{\Omega_i^l}(Y_i) = l} + 1_{\mathcal{P}_{\Omega_i^u}(X_i) > u} + 1_{\mathcal{P}_{\bar{\Omega}_i^u}(Z_i) = 0} \\ & + 1_{\mathcal{P}_{\bar{\Omega}_i^u}(X_i) \leq u} + 1_{\mathcal{P}_{\Omega_i^u}(X_i) + \mathcal{P}_{\Omega_i^u}(Z_i) = u} \Big], \end{aligned} \tag{13}$$

where the indicator function $1_{(\cdot)}$ equals to 0 when its constraint is satisfied and equals to $+\infty$ otherwise.

Eqn. (11)-(13) define a problem over a factor graph $\mathcal{G} = (\mathcal{V}, \mathcal{C})$ (see [16, 17, 20]), where every node $r \in \mathcal{V}$ carries a (convex) component function of (11) and every factor $c \in \mathcal{C}$ carries an (nonlinear) equality constraint of (12) (see Fig. 1 for demonstration).

Remark 1. *If the ReLU is chosen for layer i of the network, one can simply ignore Z_i and Ω_i^u and let $l = 0$ in (11)-(13).*

4. DISTRIBUTED OPTIMIZATION OVER A GRAPH

We note that (11)-(13) is a nonconvex optimization because of the nonlinear equality constraints (12). We solve (11)-(13) in an iterative fashion using ADMM by solving convex subproblems. It is worth noting that ADMM has already been successfully applied for solving nonnegative matrix factorization (NMF) [21], which is nonconvex.

4.1. Augmented Lagrangian function

To apply ADMM, we introduce a Lagrange multiplier Λ_i for the ith equality constraint in (12). We build an augmented Lagrangian

Table 1. ADMM updating procedure

Initialize: $\{\hat{W}_i, \hat{b}_i, |i = 1, \dots, N\}$
Repeat
  Feed X_0 to the DNN and initialize $\{\hat{X}_i, \hat{Y}_i, \hat{Z}_i, \hat{\Omega}_i^l, \hat{\Omega}_i^u\}$
  Let $\{\hat{\Lambda}_i = 0 | i = 1, \dots, N\}$
  For $i = N, N-1, \dots, 1$ do
    $(\hat{X}_i^{new}, \hat{Y}_i^{new}, \hat{Z}_i^{new})$
    $= \arg\min L_i((X_i, Y_i, Z_i), (\hat{W}_i, \hat{b}_i), \hat{\Lambda}_i | \Omega_i^u, \Omega_i^l)$
    $\hat{\Lambda}_i^{new} = \rho_i(\hat{X}_i^{new} - (\hat{X}_{i-1} + \hat{Y}_{i-1} + \hat{Z}_{i-1})\hat{W}_i - e\hat{b}_i)$
    $(\hat{W}_i^{new}, \hat{b}_i^{new})$
    $= \arg\min L_i((\hat{X}_i^{new}, \hat{Y}_i^{new}, \hat{Z}_i^{new}), (W_i, b_i), \hat{\Lambda}_i^{new} | \Omega_i^u, \Omega_i^l)$
  End for
  $(\hat{W}_i, \hat{b}_i) = (\hat{W}_i^{new}, \hat{b}_i^{new})$ for all i
Until some stopping criterion is met

function as

$$\begin{aligned} &L_{\{\rho_i\}}(\{X_i, Y_i, Z_i, b_i, W_i, \Lambda_i\}, X_N | \{\Omega_i^l, \Omega_i^u\}) \\ &= f_N(X_N; O) + \sum_{i=1}^{N} g_i(W_i, b_i) + \sum_{i=1}^{N-1} f_i(X_i, Y_i, Z_i | \Omega_i^l, \Omega_i^u) \\ &\quad + \sum_{i=1}^{N} p_{i,\rho_i}((X_{i-1}, Y_{i-1}, Z_{i-1}), X_i, (W_i, b_i), \Lambda_i), \end{aligned} \tag{14}$$

where for each $i = 1, \dots, N$, $p_{i,\rho_i}(\cdots)$ is defined as

$$\begin{aligned} p_{i,\rho_i}(\cdots) = \Big[&\frac{\rho_i}{2} \|X_i - (X_{i-1} + Y_{i-1} + Z_{i-1})W_i - eb_i\|^2 \\ &+ \langle \Lambda_i, X_i - (X_{i-1} + Y_{i-1} + Z_{i-1})W_i - eb_i \rangle \Big], \end{aligned} \tag{15}$$

where $\rho_i > 0$, $(X_0, Y_0, Z_0) = (V_0, 0, 0)$, and $\langle \cdot, \cdot \rangle$ denotes dot product. We note that differently from the single learning rate of SGD, each layer i possesses a positive parameter ρ_i, which can be treated as a layer-oriented learning rate.

Our objective now is to reach a saddle point of the Lagrangian function $L_{\{\rho_i\}}$ by minimizing over $\{X_i, Y_i, Z_i, b_i, W_i, \} \cup X_N$ and maximizing over $\{\Lambda_i\}$. A saddle point would satisfy the equality constraints (12).

4.2. Blockwise parameter updating using ADMM

We now consider optimizing the Lagrangian function $L_{\{\rho_i\}}$. We follow a similar updating procedure as the SGD and Adam methods [10]. That is, at each iteration, we initialize all the variables and index sets of $L_{\{\rho_i\}}$ by feeding D to the network through the forward computation. We then update all the variables of $L_{\{\rho_i\}}$ blockwise through backward computation. Differently from SGD which computes gradient directly, the variables of $L_{\{\rho_i\}}$ are updated by solving small-size optimization problems.

Suppose we finished updating variables of layer $i+1$ and would like to update (X_i, Y_i, Z_i), (W_i, b_i) and Λ_i of layer i. We first simplify $L_{\{\rho_i\}}$ by removing irrelevant components,

$$\begin{aligned} &L_i((X_i, Y_i, Z_i), (W_i, b_i), \Lambda_i | \Omega_i^u, \Omega_i^l) \\ &= p_{i+1,\rho_{i+1}}((X_i, Y_i, Z_i), \hat{X}_{i+1}^{new}, (\hat{W}_{i+1}, \hat{b}_{i+1}), \hat{\Lambda}_{i+1}^{new}) \\ &\quad + p_{i,\rho_i}((\hat{X}_{i-1}, \hat{Y}_{i-1}, \hat{Z}_{i-1}), X_i, (W_i, b_i), \Lambda_i) + g_i(W_i, b_i) \\ &\quad + f_i(X_i, Y_i, Z_i | \Omega_i^l, \Omega_i^u), \end{aligned} \tag{16}$$

Table 2. Computing $(\hat{X}_i^{new}, \hat{Y}_i^{new}, \hat{Z}_i^{new})$ for each $i < N$ in Table 1.

Let: $(\hat{X}_i^c, \hat{Y}_i^c, \hat{Z}_i^c) = (\hat{X}_i, \hat{Y}_i, \hat{Z}_i)$ and $(\hat{\Gamma}_i^x, \hat{\Gamma}_i^y, \hat{\Gamma}_i^z) = (0, 0, 0)$
$(\hat{X}_i^{new}, \hat{Y}_i^{new}, \hat{Z}_i^{new})$
$= \arg\min L_{i,\beta_i}((X_i, Y_i, Z_i), (\hat{\Gamma}_i^x, \hat{\Gamma}_i^y, \hat{\Gamma}_i^z), (\hat{X}_i^c, \hat{Y}_i^c, \hat{Z}_i^c))$
$\hat{\Gamma}_i^{x,new} = \beta_i(\hat{X}_i^{new} - \hat{X}_i^c)$
$\hat{\Gamma}_i^{y,new} = \beta_i(\hat{Y}_i^{new} - \hat{Y}_i^c)$
$\hat{\Gamma}_i^{z,new} = \beta_i(\hat{Z}_i^{new} - \hat{Z}_i^c)$
$(\hat{X}_i^{c,new}, \hat{Y}_i^{c,new}, \hat{Z}_i^{c,new}) = \arg\min L_{i,\beta_i}((\hat{X}_i^{new}, \hat{Y}_i^{new}, \hat{Z}_i^{new}), (\hat{\Gamma}_i^{x,new}, \hat{\Gamma}_i^{y,new}, \hat{\Gamma}_i^{z,new}), (X_i^c, Y_i^c, Z_i^c))$
$(\hat{X}_i^{new}, \hat{Y}_i^{new}, \hat{Z}_i^{new}) = (\hat{X}_i^{c,new}, \hat{Y}_i^{c,new}, \hat{Z}_i^{c,new})$

where $\hat{X}_{i+1}^{new}$ and $\hat{\Lambda}_{i+1}^{new}$ are the new estimate obtained from the computation at layer $i+1$. By following the ADMM updating procedure, we first compute $(\hat{X}_i^{new}, \hat{Y}_i^{new}, \hat{Z}_i^{new})$ by optimizing L_i with $(\hat{W}_i, \hat{b}_i)$ and $\hat{\Lambda}_i$ fixed. We then compute $\hat{\Lambda}_i^{new}$ using $\hat{X}_i^{new}$ through dual ascent. Finally, we compute $(\hat{W}_i^{new}, \hat{b}_i^{new})$ by optimizing L_i with $(\hat{X}_i^{new}, \hat{Y}_i^{new}, \hat{Z}_i^{new})$ and $\hat{\Lambda}_i^{new}$ fixed. See Table 1 for the updating procedure.

For the top layer $i = N$, the function L_N takes the form:

$$\begin{aligned} L_N(X_N, (W_N, b_N), \Lambda_N) = &f_N(X_N; O) + g_N(W_N, b_N) \\ &+ p_{N,\rho_N}((\hat{X}_{N-1}, \hat{Y}_{N-1}, \hat{Z}_{N-1}), X_N, (W_N, b_N), \Lambda_N), \end{aligned}$$

where there is no Y_N and Z_N. For this layer, only $\hat{X}_N^{new}$ is computed by optimizing L_N in Table 1.

Remark 2. *In (16),* $(\hat{W}_{i+1}, \hat{b}_{i+1})$ *is used instead of* $(\hat{W}_{i+1}^{new}, \hat{b}_{i+1}^{new})$*, which is found to be much more stable through experiments.*

4.3. Handling of the indicator function

The function $f_i(\cdot)$ in (16) is composed of a set of indicator functions, which makes it difficult to compute $(\hat{X}_i^{new}, \hat{Y}_i^{new}, \hat{Z}_i^{new})$ in Table 1. To facilitate the computation, we introduce the auxiliary variables (X_i^c, Y_i^c, Z_i^c) to replace (X_i, Y_i, Z_i) in $f_i(X_i, Y_i, Z_i | \Omega_i^l, \Omega_i^u)$ with the constraints $X_i^c = X_i$, $Y_i^c = Y_i$ and $Z_i^c = Z_i$. We then apply ADMM again to handle the three equality constraints. To do so, we build a new augmented Lagrangian as

$$\begin{aligned} &L_{i,\beta_i}((X_i, Y_i, Z_i), (\Gamma_i^x, \Gamma_i^y, \Gamma_i^z), (X_i^c, Y_i^c, Z_i^c)) \\ &= p_{i+1,\rho_{i+1}}((X_i, Y_i, Z_i), \hat{X}_{i+1}^{new}, (\hat{W}_{i+1}, \hat{b}_{i+1}), \hat{\Lambda}_{i+1}^{new}) \\ &\quad + p_{i,\rho_i}((\hat{X}_{i-1}, \hat{Y}_{i-1}, \hat{Z}_{i-1}), X_i, (W_i, b_i), \Lambda_i) + g_i(W_i, b_i) \\ &\quad + f_i(X_i^c, Y_i^c, Z_i^c | \Omega_i^l, \Omega_i^u) + \frac{\beta_i}{2} \|X_i - X_i^c\|^2 + \langle \Gamma_i^x, X_i - X_i^c \rangle \\ &\quad + \frac{\beta_i}{2} \|Y_i - Y_i^c\|^2 + \langle \Gamma_i^y, Y_i - Y_i^c \rangle + \frac{\beta_i}{2} \|Z_i - Z_i^c\|^2 + \langle \Gamma_i^z, Z_i - Z_i^c \rangle, \end{aligned} \tag{17}$$

where $\{\Gamma_i^x, \Gamma_i^y, \Gamma_i^z\}$ are the Lagrange multipliers, and $\beta_i > 0$ which has a similar role as ρ_i in $L_{\{\rho_i\}}$. We update the three sets of variables (X_i, Y_i, Z_i), $(\Gamma_i^x, \Gamma_i^y, \Gamma_i^z)$ and (X_i^c, Y_i^c, Z_i^c) one after another (see Table 2). To reduce the computational time, we only update the above variables once instead of multiple iterations.

To briefly summarize, at each iteration, the proposed algorithm performs both forward and backward computations. The forward computation initializes all variables and index sets while the backward computation updates all the variables and the network weights.

The algorithm has a set of learning rates $\{\rho_i\} \cup \{\beta_i\}$, which provides great flexibility to fine-tune the algorithm to have fast convergence (See the first experiment of Section 5 about the parameter setup).

5. EXPERIMENTAL RESULTS

In the simulation, we considered the handwritten-digit recognition problem by using MNIST with the standard division of the training (60000 samples) and test (10000 samples) datasets [22]. In doing so, we built a DNN of three layers ($N = 3$), where the first and second hidden layer consists of 500 and 600 neurons, respectively. The output function was chosen as the summation of the individual cross-entropy functions ([23]). The function $g_i(W_i, b_i)$ was chosen as $\frac{0.1}{2}\|(W_i, b_i)\|^2$. The mini-batch size was set as 3000. The entire training dataset thus consisted of 20 minibatches.

We note that the cross-entropy function makes it difficult to compute $\hat{X}_N^{new}$ analytically in Table 1. When updating the above variable at each iteration, we approximate each cross-entropy term by a quadratic function around the most recent estimate, where the quadratic coefficient is set to 0.05 and the linear coefficient is set to the gradient.

We evaluated the proposed method (referred to *ADMM*) with two proof-of-concept experiments. In the first experiment, we tested ADMM, SGD and Adam [10] using only the ReLUs. In the second experiment, we studied how the learning rates $\{\rho_i\} \cup \{\beta_i\}$ affect the convergence speed of ADMM for both ReLUs and DCutLUs.

5.0.1. *Comparison with the state-of-the-art*

In addition to ADMM, we also evaluated SGD and Adam [10], where Adam represents the state-of-the-art training method. The goal is to study the convergence properties of the proposed algorithm. The learning rate of SGD was chosen as 0.3 (producing stable and fast convergence among $\{0.1, 0.2, 0.3, 0.4\}$). Adam was implemented by following [10] directly. When running SGD and Adam, the gradient of ReLU at zero is set to 0. Finally the learning rates of ADMM were set as $\rho_3 = 0.05$, $\rho_2 = \beta_2 = 0.1$ and $\rho_1 = \beta_1 = 0.2$. The basic principle is to set the learning rates ρ_i and β_i of layer i slightly larger than ρ_{i+1} and β_{i+1} of layer $i+1$. By doing so, the influence of the layers with lower indices on the classification performance is enhanced, making the training procedure more effective.

The experimental results are displayed in Fig. 2 (a). It is seen that the performance gap of ADMM between the test data and training data is relatively stable compared to that of Adam and SGD. Furthermore, ADMM performs better than Adam and SGD on the test data, where the recognition accuracy for the test data at the last iteration is: 98.41(ADMM), 98.23(Adam) and 97.98(SGD). The better performance of ADMM might be due to the introduction of layer-oriented learning rates $\{\rho_i, \beta_i\}$.

The computational time of the three methods was measured on an Apple MacBook Pro and is summarized in Table 3. In general, ADMM is somewhat more expensive than SGD and Adam because it consumes more memory due to the auxiliary variables and involves solving a set of small-size optimization problems per min-batch.

5.0.2. *Effect of different learning rates on convergence speed*

In this experiment, we studied how the learning rates $\{\rho_i, \beta_i\}$ affect the convergence speed of ADMM for both ReLUs and DCutLUs (where $(l, u) = (0, 1)$). To simplifying the evaluation, we let all ρ_i and β_i to be the same per experiment. For each learning rate, we counted the number of iterations over entire training dataset until the average cross-entropy reaches 0.05.

Table 3. Average execution times (per mini-batch) and their standard deviations for the four methods.

	SGD (ReLU)	Adam (ReLU)	ADMM (ReLU)	ADMM (DCutLU)
ave. (second)	0.2257	0.2398	0.9373	1.446
std	0.0444	0.0416	0.0851	0.0949

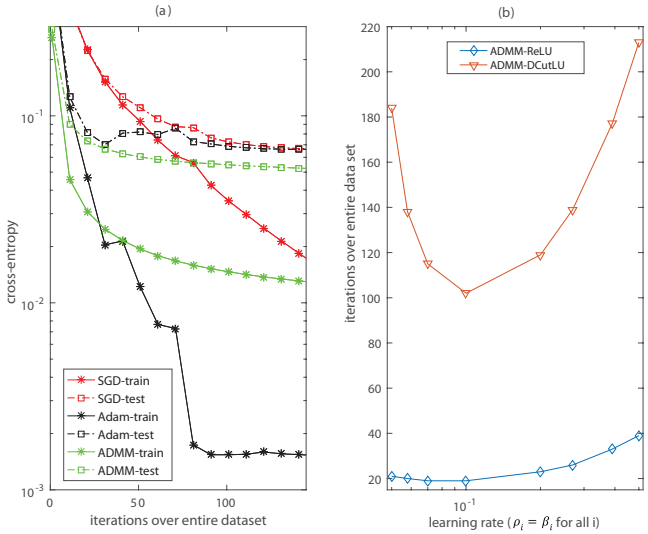

Fig. 2. Performance comparison. Subplot (a) displays the performance of SGD, Adam and ADMM using only ReLUs. Subplot (b) shows the number of iterations over entire training dataset needed to reach a threshold (0.05) of average cross-entropy for each learning rate of ADMM.

The convergence results are displayed in Fig. 2 (b). It is seen that the learning rate indeed affects the convergence speed. There exists an optimal value for the learning rate that leads to the fastest convergence speed for either ReLU or DCutLU operators. Further, it is observed that ReLU needs significantly fewer iterations than DCutLU. Table 3 also shows that the computational time of ReLU is lower than that of DCutLU. This suggests that ReLU is a better choice in practice.

Remark 3. *At the moment, the convergence of the proposed method is only demonstrated by experiments. We leave the theoretical convergence analysis for future investigation.*

6. CONCLUSIONS

We have proposed a new algorithm for training a DNN by performing optimization over a factor graph. The considered nonlinear units are the ReLUs and DCutLUs, which allow for analytic representations by using a set of slack variables. ADMM is then applied to perform distributed optimization over the graphic model. Experimental results indicate that the new algorithm is less sensitive to over-fitting than two references. One future research direction is to adjust the learning rates $\{\rho_i\}$ and $\{\beta_i\}$ of the new algorithm automatically, which likely will lead to reasonable convergence speed for various learning problems.

7. REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *Nature*, vol. 521, pp. 436–444, 2015.

[2] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional neural networks," arXiv preprint arXiv:1311.2901v3, 2013.

[3] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, "Understanding Neural Networks Through Deep Visualization," arXiv preprint arXiv:1506.06579v1, 2015.

[4] A.-R. Mohamed, G. E. Dahl, and G. Hinton, "Acoustic Modeling Using Deep Belief Networks," *IEEE Trans. Audio Speech Lang. Process*, pp. 14–22, 2012.

[5] G.E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-Dependent Pre-Trained Deep Neural Networks for Large Vocabulary Speech Recognition," *IEEE Trans. Audio Speech Lang. Process*, vol. 20, pp. 33–42, 2012.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *NIPS*, 2012.

[7] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deep Face: Closing the Gap to Human-Level Performance in Face Verification," in *Proc. Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1701–1708.

[8] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech Recognition with Deep Recurrent Neural Networks," in *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2013, pp. 6645–6649.

[9] I. Sutskever, Q. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," in *Advances in Neural Information Processing Systems 27*, 2014, pp. 3104–3112.

[10] D. P. Kingma and J. L. Ba, "Adam: A method for Stochastic Optimization," arXiv preprint arXiv:1412.6980v9, 2017.

[11] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

[12] G. Hinton, N. Srivastava and K. Swersky, ," Lecture Notes on Introduction to Neural Networks and Machine Learning, 2014.

[13] M. Carreira-Perpinan and W. Wang, "Distributed Optimization of Deeply Nested Systems," arXiv:1212.5921 [cs.LG], 2012.

[14] G. Taylor, R. Burmeister, Z. Xu, B. Singh, A. Patel, and T. Goldstein, "Training Nueral Networks Without Gradients: A Scalable ADMM Approach," in *Proc. IEEE Int. Conf. Machine Learning*, 2016.

[15] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers," *In Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[16] D. Sontag, A. Globerson, and T. Jaakkola, "Introduction to Dual Decomposition for Inference," in *Optimization for Machine Learning*. 2011, MIT Press.

[17] O. Meshi and A. Globerson, "An Alternating Direction Method for Dual MAP LP Relaxation," in *ECML*, 2011.

[18] X. Glorot, A. Bordes, and Y. Bengio, "Deep Sparse Rectifier Neural Networks," in *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, 2011, pp. 315–323.

[19] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," in *Proceedings of the 27th International Conference on Machine Learning*, 2010.

[20] M. Wainwright and M. Jordan, "Graphical models, exponential families, and variational inference," *Foundations and Trends in Machine Learning*, vol. 1(1-2), pp. 1–305, 2008.

[21] D. Hajinezhad, T.-H. Chang, X. Wang, Q. Shi, and M. Hong, "Nonnegative Matrix Factorization Using ADMM: Algorithm and Convergence Analysis," in *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2016, pp. 4742–4746.

[22] Y. LeCun, C. Cortes, and C. J.C. Burges, "MNIST handwritten digit database," http://yann.lecun.com/exdb/mnist/, 2010.

[23] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, Wiley-Interscience, 2nd edition, 2006.