Training Deep Neural Networks via Optimization Over Graphs

Guoqiang Zhang and W. Bastiaan Kleijn

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 layerwise by solving subproblems of the reformulated problem. Empirical results suggest that by proper parameter selection, the ADMM-based method converges considerably faster than gradient descent method.

Index Terms—Deep learning, DNN, optimization, ADMM.

I. INTRODUCTION

In the last decade, research on deep learning has made remarkable progress both in theoretical understanding and in practical successful applications (see [1] for an overview). In general, deep learning intends to learn representations of data with multiple levels of abstraction through a computational model with multiple processing layers. Each processing layer is composed of a set of simple but 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 computational model is able to produce data representations that are required by various applications.

A major advantage of deep learning is that it can process raw data (e.g., images or audio signals) directly and still produce competitive or better results than traditional machine learning methods. Traditional methods often require careful preprocessing of data such as manual design and the selection of representations (known as features) of data by experts. Therefore, deep learning is commonly recognised as a breakthrough towards real artificial intelligence (AI).

Roughly speaking, deep learning includes three different types of computational models: deep neural networks, convolutional neural networks (CNNs) and recurrent neural networks (RNNs). Deep neural networks have been successfully applied in speech recognition [4], [5] while CNNs are popular in computer vision [6], [7]. RNNs have proven to be effective for mapping sequential inputs and outputs [8], [9].

The most common procedure for training a neural network is based on optimizing an objective function that measures the difference or similarity between outputs of the network and their desired values. For instance, the desired values for a classification problem consists of a vector of scores, one for each category. By optimizing the objective function, the network gradually adjusts its parameters (also referred to as weights) to approach the desired values. This procedure is often referred to as supervised learning.

At the moment, stochastic gradient descent (SGD) is probably the most popular method for adjusting the weights of a network under supervised learning. The basic idea of the method is to randomly select and feed a few training samples to the network each time. It then computes an averaged gradient vector of the weights w.r.t. the training samples. Finally, the method adjusts the weights by a small amount in the direction that optimizes the objective function [10]. The amount of adjustment is explicitly controlled by a parameter called learning-rate. In general, the learning-rate has to be chosen carefully to ensure convergence with reasonable speed, which is often problem dependent.

The fact that the single learning rate determines the parameter-updating for all the layers may limit performance. A natural question is how to design a training method that allows each individual layer to learn most effectively in the learning process, leading to fast convergence.

In this paper, we propose to train a deep neural network by reformulating the problem as an optimization over a factor graph \( G = (V, C) \) [11], [12]. Every node \( r \in V \) carries a function of its node variable while every factor \( c \in C \) carries a nonlinear equality constraint in terms of the node variables connected to the factor. Our graphical formulation is able to handle rectified linear units (ReLUs) (see [13], [14]) and linear units with both upper and lower cutoffs (DCutLUs). 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.

To solve the graph-based problem, we build an augmented Lagrangian function, where each Lagrangian multiplier is introduced for an equality constraint of a factor \( c \in C \), which corresponds to one layer in the network. We then update the weights of the network layerwise by applying the alternating direction method of multipliers (ADMM) (see [15] for an overview) to optimize subfunctions of the overall augmented Lagrangian function. ADMM has already demonstrated faster convergence than other methods for solving nonnegative matrix factorization (NMF) [16], which is nonconvex. Experimental results in this paper suggest that with proper parameter selection, ADMM also converges considerably faster than the gradient descent (GD) for training a deep neural network.

We note that the work of [17] also proposes to decouple
the nested structure of DNNs by introducing a set of auxiliary variables and a set of equality constraints. However, there are three major differences between [17] and our work. Firstly, in [17], the nonlinear units (e.g., ReLUs) remain in the reformulated problem, which therefore must rely on a traditional method such as gradient descent to handle the nonlinearity. Secondly, [17] proposes to perform a two-step optimization: optimizing over all the weights by fixing all the auxiliary variables and vice versa. On the other hand, our method computes the weights layerwise where the weights of each layer are updated with the help of optimizing a subset of slack variables. Thirdly, the method of [17] needs to adjust a parameter gradually towards infinity to reach optimality.

II. ON TRAINING A DEEP NEURAL NETWORK

Suppose we have a sequence of \( m \) training samples, represented by an input matrix form \( D \in \mathbb{R}^{m \times n_{in}} \) and an output matrix form \( O \in \mathbb{R}^{m \times n_{out}} \), where the \( q \) th row-vectors of \( D \) and \( O \) forms an input-output pair. Given \((D,M)\), 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 weighting 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 research objective is to find the proper weights \( \{W_i,b_i\} \) so that the corresponding network maps the input \( D \) to the output \( O \) as accurate as possible.

In general, the selection of the nonlinear function \( h_i \) for neurons in hidden layer \( i \), \( i = 1, \ldots, N-1 \), is flexible. In recent work, ReLU is preferred over sigmoid and tanh as it leads to fast convergence using SGD [14]. We consider ReLU and DCutLU in this paper. Formally, we define DCutLU to be

\[
y = h(x) = \min(\max(x,l),u) \quad l < u,
\]

where \( l \) and \( u \) are the lower and upper threshold, respectively. ReLU is a special case of DCutLU by letting \( l = 0 \) and \( u = \infty \). Given the input matrix \( D \), we use \( V_i \) to denote the output of layer \( i \), \( i \leq N \), which is obtained by \( V_i = h_i(V_{i-1}W_i + b_i) \), where \( h_i(\cdot) \) operates on the matrix elements and \( e \) is a column vector of ones. When \( i = 1 \), \( V_0 = D \).

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

\[
\begin{align*}
\min_{\{V_i,W_i,b_i\}} & \left[ f_N(V_N;O) + \sum_{i=1}^{N} g_i(W_i,b_i) \right] \\
\text{s. t.} & \quad V_i = h_i(V_{i-1}W_i + b_i) \quad \forall i = 1, \ldots, N-1, \quad (3) \\
& \quad V_N = V_{N-1}W_N + eb_N, \quad (4)
\end{align*}
\]

where \( f_N \) measures the difference between the output \( V_N \) and the ground truth \( O \), and \( g_i \) is a penalty function on \( W_i \) and \( b_i \), which could be, for example, the \( l_2 \) or \( l_1 \) norm of \( (W_i,b_i) \).

**Remark 1.** DCutLU can be used to approximate sigmoid or tanh functions by choosing \((l, u)\) properly. One example is the work in [18] which considers using a sigmoid to approximate DCutLU for speech enhancement.

III. PROBLEM REFORMULATION ONTO A GRAPH

In this section, we reformulate \((2)-(4)\) as an optimization over a factor graph. The key step is to represent the nonlinear functions \( \{h_i\} \) explicitly by introducing a set of slack variables. To simplify the analysis, we assume that every \( h_i \) is a DCutLU, characterised by the two thresholds \( l_i \) and \( u_i \). The usage of ReLU is easily addressed in the framework.

From the above analysis, Equ. (3) can be rewritten as

\[
\begin{align*}
X_i &= V_{i-1}W_i + eb_i \\
X_i + Y_i &= \max(V_{i-1}W_i + eb_i, l_i) \quad (5) \\
X_i + Y_i + Z_i &= V_i = \min(\max(V_{i-1}W_i + eb_i, l_i), u_i) \quad (6)
\end{align*}
\]

where the max and min operators are element-wise, and for each \( i \in \{1, \ldots, N-1\} \) we introduce three slack matrices \( X_i, Y_i, Z_i \) to explicitly characterize the effect of the upper and lower cutoffs of \( h_i \) at \( u_i \) and \( l_i \).

Next we show that \((6)-(7)\) can be alternatively expressed in terms of a few constraints on \((X_i,Y_i,Z_i)\). To do so, we introduce two index sets for each layer \( i \):

\[
\begin{align*}
\Omega^f_i &= \{(i,j,q)|x_{i,j,q} < l_i\} \quad (8) \\
\Omega^u_i &= \{(i,j,q)|x_{i,j,q} > u_i\}, \quad (9)
\end{align*}
\]

where \( x_{i,j,q} \) is the \((q, j)\) element of \( X_i \). At the moment, one can think of \( \Omega^f_i \) and \( \Omega^u_i \) as two sets that are preset already, imposing a group of constraints on \( X_i \). We will explain later how to update \((\Omega^f_i, \Omega^u_i)\) iteratively. Given \( \Omega^f_i, (10) \) can be rewritten as

\[
\begin{align*}
\mathcal{P}_{\Omega^f_i}(X_i) &> l_i \quad (10a) \\
\mathcal{P}_{\Omega^f_i}(X_i) + \mathcal{P}_{\Omega^u_i}(Y_i) &> l_i \quad (10b) \\
\mathcal{P}_{\Omega^f_i}(X_i) &> l_i \quad (10c) \\
\mathcal{P}_{\Omega^u_i}(Y_i) &> 0 \quad (10d)
\end{align*}
\]

where \( \bar{\Omega} \) denotes the complement of \( \Omega \) and \( \mathcal{P}_\Omega \) denotes the projection on each and every element of \( \Omega \). By inspection of \((6)\) and \((7)\), we conclude that \( Y_i \) and \( Z_i \) are decoupled given \( X_i \). As a result, \((7)\) can be rewritten as constraints on \((X_i, Z_i)\)

\[
\begin{align*}
\mathcal{P}_{\Omega^f_i}(X_i) &> u_i \quad (11a) \\
\mathcal{P}_{\Omega^f_i}(X_i) + \mathcal{P}_{\Omega^u_i}(Z_i) &> u_i \quad (11b) \\
\mathcal{P}_{\Omega^f_i}(X_i) &> u_i \quad (11c) \\
\mathcal{P}_{\Omega^u_i}(Z_i) &> 0 \quad (11d)
\end{align*}
\]

Based on the above analysis, the training problem \((2)-(4)\) can be reformulated as

\[
\begin{align*}
\min_{\{W_i,b_i\},X_i,Y_i,Z_i} & \left[ f_N(V_N;O) + \sum_{i=1}^{N} g_i(W_i,b_i) + \sum_{i=1}^{N-1} f_i(X_i,Y_i,Z_i|\Omega^f_i,\Omega^u_i) \right] \\
\text{s. t.} & \quad X_i = (X_{i-1} + Y_{i-1} + Z_{i-1})W_i + eb_i \quad \forall i = 1, \ldots, N \quad (12)
\end{align*}
\]
where \((X_0, Y_0, Z_0) = (V_0, 0, 0)\) and each \(f_i(X_i, Y_i, Z_i; \Omega_i^t, \Omega_i^u)\) can be taken as a summation of indicator functions, each defined by one constraint in (10)-(11), given by

\[
f_i(X_i, Y_i, Z_i; \Omega_i^t, \Omega_i^u) = \left[ 1_{\Omega_i^t}(X_i) = l_i + 1_{\Omega_i^u}(Y_i) = l_i + 1_{\Omega_i^u}(Z_i) = 0 + 1_{\Omega_i^u}(X_i) \leq u_i + 1_{\Omega_i^u}(Y_i) + 1_{\Omega_i^u}(Z_i) = u_i \right], \tag{14}
\]

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

Eqns. (12)-(14) define a problem over a factor graph \(G = (\mathcal{V}, \mathcal{C})\) (see [12], [11], [19]), where every node \(r \in \mathcal{V}\) carries a component function in (12) and every factor \(c \in \mathcal{C}\) carries one equality constraint of (13) (see Fig. 1).

**Remark 2.** If the ReLU is chosen for a layer \(i\) of the network, one can simply ignore \(Z_i\) and \(\Omega_i^u\) and let \(l_i = 0\) in (12)-(14).

### IV. Optimization Over a Graph

We note that (12)-(14) is a nonconvex optimization because of the nonlinear constraints (13). To tackle (12)-(14), each time we solve a (convex) subproblem of it over a few variables by fixing the remaining variables by using ADMM. ADMM is well known to converge with reasonable speed under mild conditions and has found many successful applications ([15]).

In order to apply ADMM, we introduce a Lagrangian multiplier \(\Lambda_i\) for the \(i\)th equality constraint in (13). We then build an augmented Lagrangian function

\[
L_{\{\rho_i\}|s \geq i)}(X_i, Y_i, Z_i, b_i, W_i, \Lambda_i) = f_i(X_i; N; O) + \sum_{i=1}^{N-1} g_i(W_i, b_i) + \sum_{i=1}^{N} f_i(X_i, Y_i, Z_i; \Omega_i^t, \Omega_i^u) + \sum_{i=1}^{N} \rho_i (X_i-\lambda_i)+Y_i+Z_i-\lambda_i)W_i-e\cdot b_i)^2
\]

where for each \(i = 1, \ldots, N\), \(\rho_i(\cdots)\) is defined as

\[
\rho_i(\cdots) = \left[ \frac{\rho_i}{2} \|X_i-\lambda_iW_i-e\cdot b_i\|^2 + \langle \Lambda_i, X_i-\lambda_iW_i-e\cdot b_i \rangle \right], \tag{16}
\]

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 \(\rho_i\).

Next we optimize the Lagrangian function \(L_{\{\rho_i\}}\). We note that each function \(\rho_i(\cdots)\) involves variable-multiplications (e.g., \(X_{i-1}W_i\)) within the \(l_2\) norm, which makes it difficult to update all the variables simultaneously. To proceed with the optimization, we perform blockwise computation of the variables. At each iteration, we update \((W_i, b_i, \Lambda_i)\) sequentially from \(i = N \) (i.e., top layer) until \(i = 1\) (i.e., bottom layer) (see Table I). Each estimate \((W_i, b_i, \Lambda_i)\) is computed by following the updating procedure of ADMM. When all the weights and bias vectors are computed sequentially from the top to bottom layers, the index sets \(\{\Omega_i^t, \Omega_i^u\}\) are then updated by feeding \(X_0\) (or \(D\)) to the neural network.

Suppose \((W_i, b_i, \Lambda_i)\) is to be updated. We fix all the variables with indices less than \(i\) by replacing them with their recent estimates. Also we fix the variables \((W_s, b_s, \Lambda_s)\) for all \(s > i\) (See Fig. 2). The Lagrangian function \(L_{\{\rho_i\}}\) in this situation can be simplified as

\[
L_{\{\rho_i\}|s \geq i)}(W_i, b_i, \Lambda_i) = g_i(W_i, b_i) + \rho_i(\hat{X}_{i-1}, \hat{Y}_{i-1}, \hat{Z}_{i-1}, X_i, W_i, b_i, \Lambda_i) + \sum_{s=1}^{N-1} f_s(X_s, Y_s, Z_s; \Omega_s^t, \Omega_s^u) + f_N(X_N; O) + \sum_{s=i}^{N-1} \rho_s (X_{s-1}+W_{s-1}+Z_{s-1}+X_s, \hat{W}_s, \hat{b}_s, \hat{\Lambda}_s)\|s\|, \tag{17}
\]

which corresponds to a convex subproblem of (12)-(14).

The next step is to apply ADMM to the function \(L_{\{\rho_i\}|s \geq i)}\). To briefly summarize, we update the three sets of variables \((X_s, Y_s, Z_s; |s \geq i\}), (W_i, b_i)\) and \(\Lambda_i\) sequentially and iteratively until the method converges (see the most inner
iteration of Table II. The computation of \((\hat{W}_i, \hat{b}_i)\) is relatively simple if \(g_i(W_i, b_i)\) takes a simple functional-form (e.g., a quadratic function). The computation of \(\{X_s, \hat{Y}_s, Z_s|s \geq i\}, X_N\) is difficult because of the indicator functions in (14) or equivalently the constraints (10)-(11).

We now elaborate on the computation of \(\{\hat{X}_s, \hat{Y}_s, Z_s|s \geq i\}, X_N\) in Table II. To be able to tackle the indicator functions in (14), we introduce the auxiliary variables \((X'_s, Y'_s, Z'_s)\) to replace \((X_s, Y_s, Z_s)\) in \(f_s(X_s, Y_s, Z_s|\Omega'_s, \Omega''_s)\) with the constraints \(X'_s = X_s, Y'_s = Y_s, Z'_s = Z_s\). We then apply ADMM again to perform the computation. To handle the constraints, we build the augmented Lagrangian as

\[
L_\beta(X, Y, Z) = \sum_{s=1}^N q_s (X_s - \hat{X}_s)^2 + \frac{\beta_s}{2} \|Y_s - \hat{Y}_s\|^2 + \frac{\beta_s}{2} \|Z_s - \hat{Z}_s\|^2 + \frac{\beta_s}{2} \|X_s - X'_s\|^2 + \frac{\beta_s}{2} \|Y'_s - Y'_s\|^2 + \frac{\beta_s}{2} \|Z'_s - Z'_s\|^2 + \sum_{s=1}^N f_s(X'_s, Y'_s, Z'_s|\Omega'_s, \Omega''_s),
\]

where \(\{\beta_s\}\) are the Lagrangian multipliers, and \(\beta_s > 0\) for any \(s \geq i\) which has a similar role as \(\rho_s\) in \(L_\rho(X, \hat{Y}, \hat{Z})\). We update the three sets of variables \((X_s, Y_s, Z_s|s \geq i\), \(X_N\), \(\{X'_s, Y'_s, Z'_s|s \geq i\}\) and \(\{\beta_s\}\) sequentially and iteratively until the method converges (see Table II).

V. EXPERIMENTAL RESULTS

In the simulation, we considered the handwritten-digits recognition problem by using MNIST with the standard division of the training and test datasets. In doing so, we built a deep neural network of three layers \(N = 3\). Each of the two hidden layers consists of 800 neurons \((n_1 = n_2 = 800)\). The output function was chosen as the summation of the individual cross-entropy functions (20), one per training sample. The function \(g_i(W_i, b_i)\) was chosen as \(\frac{1}{2}\|\langle W_i, b_i\rangle\|^2\) to avoid computing the inversion of singular matrices. The performance of the algorithms (implemented by Matlab) was evaluated by measuring the execution time on an Apple macbook pro.

We note that the cross-entropy function makes it difficult to compute \((\hat{X}_s, \hat{Y}_s), Z_s|s \geq i\), \(X_N=3)\) analytically in Table II. When updating the above variables 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 5 and the linear coefficient as the gradient.

In designing the stopping criterion for the double-loop ADMM in Table I and II we measure the \(l_\infty\) norm of the difference between \(X'_i\) and its last estimate for the inner-loop ADMM and the \(l_\infty\) norm of the difference between \(\hat{X}_i\) and its last estimate for the outer-loop ADMM. The thresholds for the inner and outer loops were set as \(10^{-3}\) and \(10^{-2}\), respectively.

We tested the proposed algorithm for both the ReLUs and DCutLUs (where \(\{l_i = 0\}\) and \(\{u_i = 1\}\) in the network, referred to as ADMM-ReLU and ADMM-DCutLU, respectively. To make a comparison, we also tested the gradient descent method for the network with ReLUs, referred to as GD-ReLU.

In the simulation, all the training samples \((n=60,000)\) were processed simultaneously instead of processing mini-batches.

We evaluated the methods by conducting two experiments. In the first experiment, we tested the convergence properties of the three methods. The learning rate of GD-ReLU was chosen as 0.4 (producing the fastest convergence among \(\{0.1, 0.2, 0.3, 0.4, 0.5\}\) divided by \(m = 60,000\) in order to normalize the objective function and the gradient) while all the other parameters \(\{\rho\}\) and \(\{\beta\}\) of ADMM-ReLU and ADMM-DCutLU were set to 1 for simplicity. The experimental results are displayed in Fig. 3(a) and (b). It is seen that ADMM-ReLU converges fast while GD-ReLU converges fast in the beginning and gets slower later on. Furthermore, ADMM-ReLU exhibits overfitting, which reduces performance on the test data after 20 minutes, while the other two methods tend to have increasing recognition accuracy over a relatively long time.

In the second experiment, we studied how the parameters \(\{\rho\}\) and \(\{\beta\}\) affect the convergence speed of ADMM-ReLU. Three different configurations of the parameters were tested, where the results are displayed in Fig. 3 (c) and (d). It is observed that large values of \(\{\rho\}\) slow down the convergence speed while the recognition accuracy increases compared to the case of small \(\{\rho\}\). Further, the parameters \(\{\beta\}\) have little effect on the recognition accuracy for fixed \(\{\rho\}\).

**Remark 3.** Additional experiments were conducted showing that the weights of all the layers can also be updated simultaneously using ADMM. The results will be reported in a longer version of the paper.

VI. CONCLUSIONS

We have proposed a new algorithm for training a DNN by performing optimization over a factor graph. The key step is to explicitly represent the ReLUs or DCutLUs by a set of slack variables, which enables a problem reformulation over a factor graph. The weights of the networks are updated layerwise by applying ADMM to solve different (convex) subproblems of the overall problem on the graph. One future research direction is to adjust the parameters \(\{\rho\}\) and \(\{\beta\}\) automatically and properly, which can lead to reasonable convergence rates for various learning problems.
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:1311.2001v3, 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. 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] L. Bottou and O. Bousquet, “The Tradeoffs of Large Scale Learning,” in Proc. Advances in Neural Information Processing Systems 20, 2007, pp. 161–168.
[11] D. Sontag, A. Globerson, and T. Jaakkola, “Introduction to Dual Decomposition for Inference,” in Optimization for Machine Learning, MIT Press, 2011.
[12] O. Meshi and A. Globerson, “An Alternating Direction Method for Dual MAP LP Relaxation,” in ECML, 2011.
[13] X. Glorot, A. Bordes, and Y. Bengio, “Deep Sparse Rectifier Neural Networks,” in Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS), 2011, pp. 315–323.
[14] V. Nair and G. E. Hinton, “Rectified Linear Units Improve Restricted Boltzmann Machines,” in Proceedings of the 27th International Conference on Machine Learning, 2010.
[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. 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 (ICASSP), 2016, pp. 4742–4746.
[17] M. Carreira-Perpinan and W. Wang, “Distributed Optimization of Deeply Nested Systems,” arXiv:1212.5921 [cs.LG], 2012.
[18] C. Taal, J. Jensen, and A. Leijon, “On Optimal Linear Filtering of Speech for Near-End Listening Enhancement,” IEEE Signal Processing Letters, vol. 20, pp. 225–228, May 2013.
[19] 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.
[20] T. M. Cover and J. A. Thomas, Elements of Information Theory, 2nd ed. Wiley-Interscience, 2006.