Report of Applying ADMM to Neural Network

Qunjie Zhou and Zhenzhang Ye

I. INTRODUCTION

A. Artificial Neural Networks

Artificial neural networks(ANN) are computational models, inspired by natural neural networks to learn useful representation from data. They have been widely applied to tackle real-world problems and brought breakthrough to fields such as image recognition and natural language processing. Before a neural network can actually perform pattern recognition tasks, it requires numerous learning on a specific dataset, which refers to the training process.

In ANN, a neuron as the core composing unit can be mathematically represented as a function $g : a \to z$

$$g(a,W) = h(\sum_k a_k w_k) = h(Wa) \tag{1}$$

where a is the input data, W is the weight matrix (Here we consider bias is integrated into W) and h is an activation function[1]. A layer is composed by a set of artificial neurons with identical activation function. A feed-forward neural network where the output of neurons of previous layer is feed forward to the neurons of current layer, can be formed by consecutively connecting multiple layers.

To be specific, a 3-layer feed-forward neural network can be mathematical defined as:

$$f(a_0,W) = W_3(h_2(W_2(h_1(W_1 a_0)))) \tag{2}$$

where $W = \{W_1, W_2, W_3\}$ denotes the ensemble of weight matrices, and each column of input a_0 is a training sample.

B. Training a neural network with SGD

Such a neural network defined in (2) is learned by varying weight matrix W so that the output activation a_L match to the target label y. Using a loss function l as the evaluation criteria, the training task can be proposed as an optimization problem:

$$\min_W l(f(a_0,W),y) \tag{3}$$

As the activation function h introduces non-linearity into the network model, problem (3) results in a highly non-convex optimisation problem. The most extensively applied approach for training a neural network is stochastic gradient descent (SGD), using backpropagation to calculate the gradients. Although SGD performs well in serial setting where modern GPUs are necessary for efficient training, it fails to maintain strong scaling when parallelizing over CPUs machines [2]. In addition, the class of gradient descent based approaches also has the severe problem of being trapped in poor local optima [3] or a saddle point [4], especially for deeper architecture.

C. Alternating Direction Method of Multipliers

In the context of optimisation, except for the gradient-based algorithms, there are still more sophisticated optimisation methods. The alternating direction method with multipliers (ADMM) has been one of most powerful and successful methods for solving constrained convex problems. Recent studies (e.g. [5], [6], [7], [8]) on ADMM for non-convex optimization problem, show that ADMM also performs well on a broad class of solve non-convex problems. Compared with SGD methods, in ADMM the original complex problem is decomposed into several simpler subproblems that are often solvable in closed form. Furthermore, the decomposition leads to possibility of large-scale distributed computing environments.

In this project, we are interested applying the specific optimisation method ADMM to the neural network training task. We follow the approach proposed in work [9] and investigate whether ADMM can get rid of the troubles of SGD mentioned before. In section II, we state the concrete problem with mathematic notations. The Goldstein ADMM method proposed by work [9] is fully studied in section III, with practical experiments and theoretical analysis. In the following section IV, an improved version of this method is proposed and demonstrated. Then, the section V, the improved version is evaluated on different datasets. Finally, section VI comes to a conclusion based on previous results.

II. PRELIMINARIES

To verify the feasibility of training ANN with ADMM, we select the simplest feed-forward neural network as our training model. To define the problem, let's assume the network consists of L layers. Given an input a_{l-1} and its label y, the goal of training is to find the best weight W which minimizes a loss function $\mathscr{L}$. The concrete mathematical model is defined as:

$$\begin{aligned} \underset{\{W_l\},\{a_l\},\{z_l\}}{\text{minimize}} \quad & \mathscr{L}(z_L,y) \\ \text{s.t.} \quad & z_l = W_l a_{l-1}, l = 1,2,...,L \\ & a_l = h_l(z_l), l = 1,2,...L-1 \end{aligned} \tag{4}$$

Following the idea of [9], the original unconstrained problem (3) is equivalent to the proposed constrained problem (4), where separating the objective function at each layer of a neural network into two terms: one term measuring the relation between the weights and the input activations, and the other term containing the nonlinear activation function.

III. Study on Goldstein ADMM

A. The Proposed Method

In work [9], they proposed a method to solve the constrained problem (4) by combining Bregman iterative update with an alternating minimization strategy. They first relax the constraints by adding l_2 penalty functions to the objective function, which is actually a quadratic penalty formulation. Then Lagrange multipliers are applied to output term z_L to exactly enforce equality constraints, which yields the following unconstrained problem:

$$\begin{aligned}\underset{\{W_l\},\{a_l\},\{z_l\}}{\text{minimize}}\ & \mathscr{L}(z_L,y) \\ & +\langle z_L,\lambda\rangle+\beta_L||z_L-W_La_{L-1}||^2 \\ & +\sum_{l=1}^{L-1}[\gamma_l||a_l-h_l(z_l)||^2+\beta_l||z_l-W_la_{l_1}||^2]\end{aligned} \tag{5}$$

where $\{\gamma_l\}$ and $\{\beta_l\}$ are constants that control the weight of each constraint; λ is a vector of Lagrange multipliers with the same dimensions as z_L.

Noting (5) is not a classical ADMM formulation, because the Lagrange multiplier is added only to the objective term z_L. A true ADMM formulation should be:

$$\begin{aligned}\underset{\{W_l\},\{a_l\},\{z_l\}}{\text{minimize}}\ & \mathscr{L}(z_L,y) \\ & +\sum_{l=1}^{L}[\langle\lambda_l,z_l-W_la_{l-1}\rangle+\beta_l||z_l-W_la_{l-1}||^2] \\ & +\sum_{l=1}^{L-1}[\langle\eta_l,a_l-h_l(z_l)\rangle+\gamma_l||a_l-h_l(z_l)||^2]\end{aligned} \tag{6}$$

where $\{\lambda_l\}$, $\{\eta_l\}$ are sets of Lagrange multipliers corresponding to each constraint. They also mentioned the intuition behind (5) is to apply Bregmen Iteration [10] for the problem (4).

B. Bregman Iteration Method

Here we will shortly explain the idea of Bregman Iteration Method and derive (5) in detail. For a given problem:

$$min \quad J(u) \quad s.t. \quad Au=b \tag{7}$$

where $J(u)$ is convex and A is a linear operator. The Bregman method iteratively solve:

$$u^{k+1}\leftarrow\underset{u}{\operatorname{argmin}}\,D_J^{p^k}(u,u^k)+\frac{1}{2}||Au-b||^2 \tag{8}$$

where $p^k\in\partial J(u^k)$ is the subgradient of J at u^k and $D_J^{p^k}(u,u^k)=J(u)-J(u^k)-\langle p^k,u-u^k\rangle$ is called Bregman Distance. Therefore, by definition of $D_J^{p^k}(u,u^k)$, the multiplier is added only to the objective variable u, leaving the constraints as l_2 penalty terms.

In our problem (4), the objective variable is the output layer z_L. Thus $J(u)=\mathscr{L}(z_L,y)$ and constraint $Au=b$ corresponds to only $z_L=W_La_{L-1}$. Therefore, applying (8) to (4) gives the update of z_L^{k+1}:

$$\begin{aligned}z_L^{k+1} &= \underset{z_L}{\operatorname{argmin}}\,D_l^{p^k}(z_L,z_L^k)+\beta_L||z_L-W_La_L-1||^2 \\ &= \underset{z_L}{\operatorname{argmin}}\,\mathscr{L}(z_L)-\mathscr{L}(z_l^k)-\langle\partial z_L^k,z_L-z_L^k\rangle+\beta_L||z_L-W_La_L-1||^2 \\ &= \underset{z_L}{\operatorname{argmin}}\,\mathscr{L}(z_L)-\langle\partial(z_L^k),z_L\rangle+\beta_L||z_L-W_La_L-1||^2\end{aligned}$$

The first-order optimality condition of z_L gives immediately λ update:

$$\begin{aligned}&0\in\partial(z_L^{k+1})-\partial(z_L^k)+2*\beta_L(z_L^{k+1}-W_La_{L-1}) \\ &-\partial(z_L^{k+1})\in-\partial(z_L^k)+2*\beta_L(z_L^{k+1}-W_La_{L-1}) \\ &Setting\ \lambda\in-\frac{1}{2}\partial(z_L) \\ &\lambda_{k+1}=\lambda^k+\beta_L(z_L^{k+1}-W_La_{L-1})\end{aligned}$$

According to the paper, they somehow combine the Bregmen Iteration with the l_2 penalty of the left constraints i.e. constraints except for z_L, which leads to exactly (5). Then by applying an alternating direction method to formulation (5) and addressing each term separately, we get the following algorithm:

Algorithm 1: ADMM for Neural Networks

Input: training features $\{a_0\}$, and labels $\{y\}$
initialize $\{a_l\}_{l=1}^L$, $\{z_l\}_{l=1}^L$, λ and weights
repeat
  for *l = 1, 2, ..., L-1* **do**
    $W_l\leftarrow z_la_{l-1}^\dagger$
    $a_l\leftarrow(\beta_{l+1}W_{l+1}^TW_{l+1}+\gamma_lI)^{-1}$
      $\cdot(\beta_{l+1}W_{l+1}^Tz_{l+1}+\gamma_lh_l(z_l))$
    $z_l\leftarrow argmin_z\gamma_l||a_l-h_l(z)||^2+\beta_l||z-W_la_{l-1}||^2$
  $W_L\leftarrow z_La_{L-1}^\dagger$
  $z_L\leftarrow argmin_z\mathscr{L}(z,y)+<z,\lambda>+\beta_L||z-W_La_{L-1}||^2$
  $\lambda\leftarrow\lambda+\beta_L(z_L-W_La_{L-1})$
until *converged*;

C. Implementation and Evaluation

We implemented the updates of Algorithm 1 using Python. In this section, we will explain the details on how we implement these sub-steps. First, we choose the most widely adopted rectified linear units (ReLUs) to be the activation function h, which is defined as:

$$h(x)=\begin{cases}x, & \text{if } x>0\\ 0, & \text{otherwise}\end{cases}$$

In the following, we discuss only the updates of z_l and z_L, which is not as straightforward as the others.

Update z_l: Since this sub problem is separable, we can compute z_l element-wisely. The original problem can be written as:

$$z_l(i,j)\leftarrow argmin_z\gamma_l(a_l(i,j)-h_l(z))^2+\beta_l(z-W_la_{l-1}(i,j))^2 \tag{9}$$

where $a_l(i,j)$ represents the element in i-th row and j-th column of a_l, and analogously for $W_l a_{l-1}(i,j)$.

Even though the element-wise problem is nonconvex, it can be solved to global optimality. There are two cases for possible $z_l(i,j)$:

$$
\begin{aligned}
&1.\text{assume } z_l(i,j) \geq 0 \\
&\quad z_l(i,,j) \leftarrow argmin_z \ \gamma_l(a_l(i,j)-z)^2+\beta_l(z-W_l a_{l-1}(i,j))^2 \\
&\quad \Rightarrow \begin{cases} \widehat{z}=\frac{\gamma_l a_l(i,j)+\beta_l W_l a_{l-1}(i,j)}{\beta_l+\gamma_l}, & \gamma_l a_l(i,j) \geq -\beta_l W_l a_{l-1}(i,j) \\ 0, & otherwise \end{cases} \\
&2.\text{assume } z_l(i,j) \leq 0 \\
&\quad z_l(i,,j) \leftarrow argmin_z \ \beta_l(z-W_l a_{l-1}(i,j))^2 \\
&\quad \Rightarrow \begin{cases} \widehat{z}=W_l a_{l-1}(i,j), & \beta_l W_l a_{l-1}(i,j) \leq 0 \\ 0, & otherwise \end{cases}
\end{aligned}
\tag{10}
$$

Therefore, for a given W_l, a_l and a_{l-1}, there are three possible choices of z_l for each element. We can substitute all of possible z_l into equation (9). The feasible and optimal z_l is chosen for each element to get the optimal solution.

Update z_L: We have to first decide the choice of loss function l. We tried to use hinge loss, mean square and softmax cross entropy, it turned out softmax has the best numerical result. However, using softmax as the loss function makes the z_L update unsolvable in a closed form. Thus, we tried to use both gradient descent and proximal gradient to solve the minimization subproblem i.e. the update of z_L. Both of them worked out and achieved reasonable results.

To evaluate the method, we tested it on Mnist dataset with different network architectures i.e. various number of hidden layers and its dimension. The weight matrix W is initialized with random samples from Gaussian distribution with zero mean and standard deviation ε^2.

According to the methods published in "The MNIST Dataset" [1], ANN can achieve at least 95% accuracy. Also, we implement the ANN according to the tutorial implementation which uses normal SGD (batch size = 100, iteration = 2000, no momentum and no regularization) and results an accuracy around 92%. However, after we tuned a list of hyper-parameters (e.g. ε, β_l, γ_l, etc.), the best accuracy our network can achieve is around 87%, which is far away from the result stated on the Mnist website.

D. Analysis

In this section, we want to investigate the potential reasons of the low performance of our model. By looking at the energy, we found that the energy does not converge smoothly to the optimal. As illustrated by the Fig. 1, the energy stops changing around 2.0 and it is supposed to decay to 0.

Recall the difference between the Goldstein ADMM and the classical ADMM, the Lagrange multiplier and the Bregman iteration might contribute to the low performance. Therefore, we remove the Lagrange item and only include the quadratic penalties into objective function.

[1] http://yann.lecun.com/exdb/mnist/

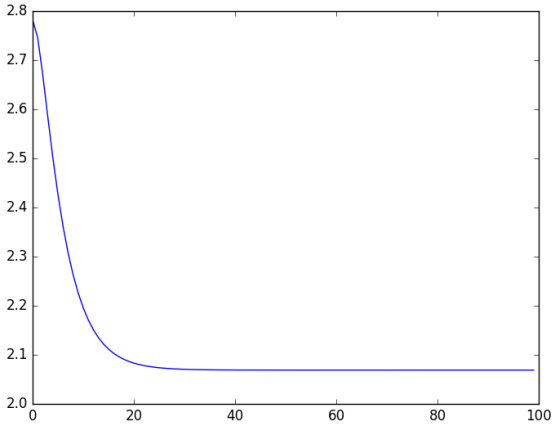


Fig. 1: The energy converges to 2.08 with Goldstein ADMM

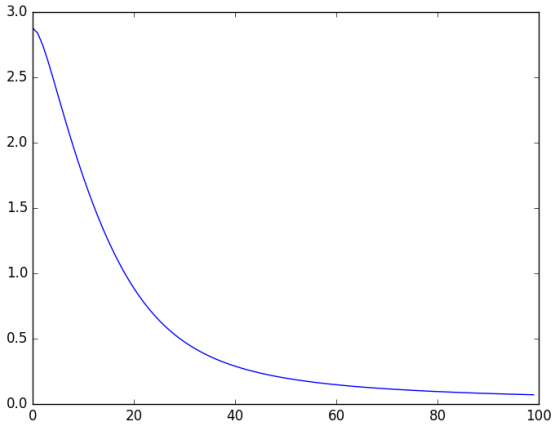


Fig. 2: The energy converges to 0 without λ

We apply Algorithm 1 with $\lambda = 0$ and deleting line 10 to solve equation (16). A result with lower energy and higher accuracy is achieved, which is illustrated in Fig.2. Based on this fact, we guess the Bregman iteration and the Lagrange multiplier are the main reason for the low performance.

To analyze the reason mathematically, we introduce a simple problem:

$$
\begin{aligned}
\underset{w,a}{\text{minimize}} \quad & (a-1)^2 \\
\text{s.t.} \quad & w+a=2
\end{aligned}
\tag{11}
$$

which can be viewed as an ANN with 1 layers and the dimension of a is $\mathbb{R}$. Since the Bregman iteration is applied for linear problems, we simplify $w \cdot a$ with $w+a$. Applying the idea of equation (5):

$$
\underset{w,a}{\text{minimize}} \quad (a-1)^2+\lambda \cdot a+\beta(2-w-a)^2 \tag{12}
$$

Above problem can be solve by using Algorithm 1, which

giving us:

$$\begin{aligned} w^{k+1} &= 2 - a^k \\ a^{k+1} &= \frac{\lambda^k + 2 + 4\beta - 2\beta w^{k+1}}{2+2\beta} \\ \lambda^{k+1} &= \lambda^{k+1} + 2\beta(2 - w^{k+1} - a^{k+1}) \end{aligned} \tag{13}$$

This test example cannot return the result $w = 1, a = 1$ unless one of them is initialized as 1. However, the result can always satisfy the constraint. This is because of the property of Bregman iteration: once a feasible result is achieved, the iteration stops and it is the optimal one. It is obvious that when w^k and a^k satisfy the constraints, equation (18) will stop updating. To show the rest of this property, recall that Bregman iteration wants to minimize the Bregman distance $D_J^p(u,v) = J(u) - J(v) - < p, u - v >, p \in \partial J(v)$ which is nonnegative. Therefore, substitute our problem into the nonnegative inequality, for any a, we can get

$$\begin{aligned} J(a^k) &\leq J(a) - (a - a^k) \cdot \lambda^k \\ &= J(a) - (a - (2 - w^k)) \cdot \lambda^k \end{aligned} \tag{14}$$

which means for any a that satisfies $a + w^k = 2$, we can get the optimal $J(a)$. However, we cannot ensure that this w^k is the optimal value, i.e. if the final w^k is not optimal, the corresponding a is not optimal.

In fact, we can construct a new value $u = [w, a]^T$ and rewrite the problem in equation (11)

$$\begin{aligned} \underset{u}{\text{minimize}} \quad & ([0,1] \cdot u - 1)^2 \\ \text{s.t.} \quad & [1,1] \cdot u = 2 \end{aligned} \tag{15}$$

then if we apply Bregman iteration to above problem, which is classical Bregman iteration, the result is correct. In this case, the Lagrange multiplier λ is a 2x1 matrix, which means updating w should consider λ.

All above are discussed in linear case, we also tested the same idea in nonlinear case, this Bregman iteration cannot work neither. Therefore, we conclude that the reason why Goldstein ADMM cannot work is that the Bregman iteration and Lagrange multiplier should not only be considered for z_L only.

IV. Improvement based on Goldstein ADMM

In this section, we propose our improvement based on Goldstein ADMM. We first try to solve equation (4) using classical ADMM. It turns out that classical ADMM cannot perform stably. Inspired by the quadratic penalty method, we include only l_2 norm penalty in to objective function. This method can return a sequence with decaying energy to 0. Nonetheless, the constraints in equation (4) are satisfied slowly and even not satisfied if the parameter γ, β are not chose correctly. Additionally, we notice that if β in Algorithm 1 is sufficient small, it is almost same as the quadratic penalty method. Therefore, we figure out one algorithm which can converge to optimal solution to a certain degree and the constraints are satisfied.

A. Quadratic penalty method

According to previous analysis, the Lagrange multiplier and Bregman iteration are the causes why Algorithm 1 fails. Thus, we manage to solve the equation (4) by using quadratic penalty method, which leads to the equation:

$$\begin{aligned} \underset{\{W_l\},\{a_l\},\{z_l\}}{\text{minimize}} \; & l(z_L, y) + \beta_L ||z_L - W_L a_{L-1}||^2 \\ & + \sum_{l=1}^{L-1} [\gamma_l ||a_l - h_l(z_l)||^2 + \beta_l ||z_l - W_l a_{l_1}||^2] \end{aligned} \tag{16}$$

This optimization problem can be solved by Algorithm 1 with a little modification mentioned before.

It is obvious that once the optimal value is got, the corresponding $\{W_l\}$, $\{a_l\}$,$\{z_l\}$ are feasible for equation (4). However, the experiments show that the norms of constraints decays slowly, as illustrated in Fig.3. This is caused by the property of ADMM that original complex problem is decomposed into several simpler subproblems. Updating $\{a_l\}$ and $\{z_l\}$ become a trade-off between satisfying the constraints and minimizing loss function. If increasing the weight of each constraint, constraints are satisfied faster with higher loss energy. Additionally, increasing the iteration number can satisfy the constraints and lower the loss energy at the same time, which accounts for more computational expensive especially for large data set.

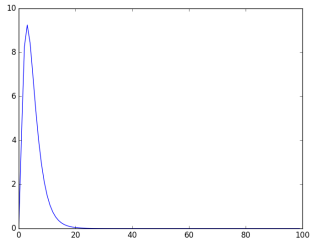
B. Bregman iteration with increasing penalty weights

To overcome the disadvantage of quadratic penalty method, a time-saving algorithm, which can satisfy constraints and minimize loss function at the same time, is proposed. Inspired by above discussion about the weight of each constraints, in this algorithm, the penalty weights are increasing at each iteration. To improve the efficiency, we also implement mini-batch for this algorithm. Besides, the regularization item is involved in the object function to prevent over-fitting and numerical errors.

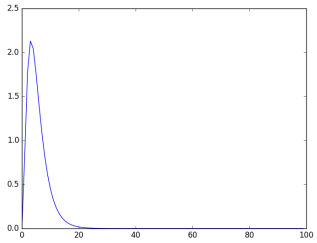
$$\begin{aligned} \underset{\{W_l\},\{a_l\},\{z_l\}}{\text{minimize}} \quad & l(z_L, y) + \sum_{l=1}^{L} \alpha ||W_l||^2 \\ \text{s.t.} \quad & z_l = W_l a_{l-1}, l = 1, 2, ..., L \\ & a_l = h_l(z_l), l = 1, 2, ...L-1 \end{aligned} \tag{17}$$

where α is the weight of regularization. The new algorithm to solve this optimization problem is described in Algorithm 2.

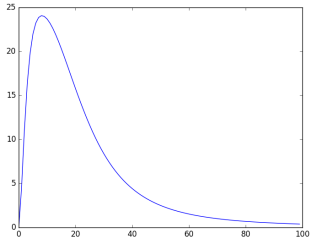
Increasing penalty weights: according to the discussion in quadratic penalty method, the weights of penalty play an important role in satisfying constraints and minimizing loss function. Therefore, the main idea is that the initial value of the weights are sufficient small and at each iteration, the weights are multiplied by a constant $\rho > 1$. We also involve the Lagrange multiplier λ and apply the same Bregman iteration as Goldstein ADMM to force the constraints to be satisfied faster. The main reason is that with small $\{beta_L\}$, the λ has ignorable effect on updating z_L since λ is initialized with 0. Thus, at the beginning iterations, a z_L which can minimizing the loss function sufficiently, is achieved. When



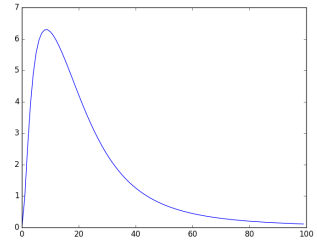
(a) $||a_l - h_l(z_l)||^2$ with Lagrange multiplier λ



(b) $||z_l - W_l a_{l-1}||^2$ with Lagrange multiplier λ



(c) $||a_l - h_l(z_l)||^2$ without Lagrange multiplier λ



(d) $||z_l - W_l a_{l-1}||^2$ without Lagrange multiplier λ

Fig. 3: The norm of constraints, $l = 1$

TABLE I: Parameters and Results on MNIST

Hidden Layers	$\{\beta_l\}$	$\{\gamma_l\}$	ρ	inIteration	outIteration	time	final energy	train accuracy	test accuracy
(300)	1e-6	1e-6	1.05	4	400	220s	0.15378	95.4%	94.8%
(500,300)	1e-8	1e-8	1.05	4	400	504s	0.15871	95.2 %	95.1%

TABLE II: Parameters and Results on CrescentMoon

Hidden Layers	$\{\beta_l\}$	$\{\gamma_l\}$	ρ	iters	time	final energy	train acc	test acc
(50,50)	1e-2	1e-2	1.01	450	71s	0.1026	95.40%	94.74%
(10,10,10,10,10)	1e-2	1e-2	1.02	450	18s	0.1063	95.46 %	94.80%

the $\{beta_L\}$ becomes large enough, z_L has to satisfy the constraints as well. To validate our idea, we still use equation (11). β is initialized with 0.0001 and *rho* is 1.01. For each iteration k, we get

$$\begin{aligned} w^{k+1} &= 2 - a^k \\ a^{k+1} &= \frac{\lambda^k + 2 + 4\beta - 2\beta w^{k+1}}{2 + 2\beta} \\ \lambda^{k+1} &= \lambda^{k+1} + 2\beta(2 - w^{k+1} - a^{k+1}) \\ \beta &= \beta \cdot \rho \end{aligned} \tag{18}$$

The test result is $x = 1.0001, w = 0.9999$. Even if the iteration number is increased, it cannot converge to the optimal result, which means this algorithm can only converges to the neighborhood of the optimal result. The radius of the neighborhood depends on the initial value of β. If β is initialized with smaller positive value, a final result which is closer to optimal one is achieved, which coincides with above analysis. Another parameter ρ is determined based on the initial value of β. Small ρ leads to a slow speed of satisfying the constraints. Too large ρ results in a high loss energy. Besides, this algorithm is also tested on nonlinear case and can work as well. Therefore, even with the disadvantage that it can only reach the neighborhood, we decide to apply this algorithm to solve equation (4).

Other improvement: We implement the mini-batch to our algorithm. The basic idea is that a batch is randomly chose from the data set. Then the neural network is trained by this batch. After several iterations, a new batch is chose and train the neural network based on previous results. Besides, we also include the regularization item for $\{W_l\}$ to solve the numerical errors and over-fitting. The new problem then becomes:

V. Experiments

In this section, we show the numeric results of Alg 2 on three different datasets: Mnist and CresentMoon. The following experiments were run on CPU (MacBook Air Early 2014 with 1,7 GHz Intel Core i7). For every case, we pick weight decay as 1e-3.

A. Mnist

The Mnist dataset has 60,000 training images and 10000 tesing images. We used all 60,000 examples for training and didn't have any validation examples. To validate the algorithm, we trained the network with two structure and the corresponding result is shown in table I. In both case, we initialize the all weight matrices w_l as zero mean Gaussian

Algorithm 2: ADMM for Neural Networks

Input: training features $\{a_0\}$, and labels $\{y\}$

initialize $\{a_l\}_{l=1}^{L}$, $\{z_l\}_{l=1}^{L}$, λ and weights

for *i = 1, ..., outIteration* **do**

  $a_0 \leftarrow$ random batch

  for *j = 1, ..., inIteration* **do**

    for *l = 1, 2, ..., L-1* **do**

      $W_l \leftarrow (\alpha+I)^{-1} z_l a_{l-1}^T (I + a_{l-1} a_{l-1}^T)^{-1}$

      $a_l \leftarrow (\beta_{l+1} W_{l+1}^T W_{l+1} + \gamma_l I)^{-1}$

        $\cdot (\beta_{l+1} W_{l+1}^T z_{l+1} + \gamma_l h_l(z_l))$

      $z_l \leftarrow argmin_z ||a_l - h_l(z)||^2 + \beta_l ||z - W_l a_{l-1}||^2$

      $\beta_l \leftarrow \beta_l \cdot \rho$

    $W_L \leftarrow (\alpha+I)^{-1} z_L a_{L-1}^T (I + a_{L-1} a_{L-1}^T)^{-1}$

    $z_L \leftarrow argmin_z \mathscr{L}(z, y)$

      $+ < z, \lambda > + \beta_L ||z - W_L a_{L-1}||^2$

    $\lambda \leftarrow \lambda + \beta_L (z_L - W_L a_{L-1})$

    $\beta_L \leftarrow \beta_L \cdot \rho$

random weights with standard deviation ε = 1e-1.We applied mini-batch for training both of them for time efficiency. Each batch has 400 images, and 4 ADMM steps are applied per batch. We found that both networks are better trained by starting with very small penalty parameters $\{\beta_l\}$, $\{\gamma_l\}$ and then increasing them with small growing step ρ. Additionally, the same networks are trained without mini batch. The accuracy around 97% and energy lower than 0.1 can be achieved, while the computational time becomes extremely long.

B. CrescentMoon

This dataset is much smaller than Mnist, with 6867 training images, no validation images and 3133 testing image. This time we didn't use mini-batch during training. Two different network structures are trained and the corresponding result is shown in table II. In both case, we initialize the all weight matrices w_l as zero mean Gaussian random weights with standard deviation ε = 1e-2. For the second case with more hidden layers i.e. (10,10,10,10,10), we notice much more unstable performance of the Alg 2, compared with the first case. In table II, the first configuration we can always get accuracy around 95.4%. However, with the second configuration, the algorithm sometimes give testing accuracy only around 80%, even though one can get accuracy around 95.4% after several tryings. In addition, we found for case two several settings can reach close testing accuracy, but also has similar unstable behavior. The reason of such an instability is not clear to us yet.

VI. Conclusion

We implemented the ADMM from Goldstein on MNIST data set. However, this ADMM fails to achieve an expected result, which is caused by the Bregman iteration according to our analysis on a simple optimizing problem. To improve the Goldstein ADMM, we propose the ADMM with increasing weight parameters for constraints and the Bregman iteration is still applied to our ADMM. The experiement results show that this ADMM can achieve a higher accuracy and lower energy. There are still some open questions about this ADMM: i) the algorithm can only converge to the neighborhood of optimal results; ii) the instability for different data set; iii) the converge analysis of this ADMM.

References

[1] "The Machine Learning Dictionary" , http://www.cse.unsw.edu.au/~billw/mldict.html

[2] Dean, Jeffrey, et al. "Large scale distributed deep networks." Advances in neural information processing systems. 2012.

[3] Choromanska, Anna, et al. "The Loss Surfaces of Multilayer Networks." AISTATS. 2015.

[4] Dauphin, Yann N., et al. "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization." Advances in neural information processing systems. 2014.

[5] Xu, Zheng, et al. "An empirical study of admm for nonconvex problems." arXiv preprint arXiv:1612.03349 (2016).

[6] Hong, Mingyi, Zhi-Quan Luo, and Meisam Razaviyayn. "Convergence analysis of alternating direction method of multipliers for a family of nonconvex problems." SIAM Journal on Optimization 26.1 (2016): 337-364.

[7] Wang, Yu, Wotao Yin, and Jinshan Zeng. "Global convergence of ADMM in nonconvex nonsmooth optimization." arXiv preprint arXiv:1511.06324 (2015).

[8] Wang, Fenghui, Zongben Xu, and Hong-Kun Xu. "Convergence of Bregman alternating direction method with multipliers for nonconvex composite problems." arXiv preprint arXiv:1410.8625 (2014).

[9] Taylor, Gavin, et al. "Training neural networks without gradients: A scalable admm approach." International Conference on Machine Learning. 2016.

[10] Goldstein, Tom, and Stanley Osher. "The split Bregman method for L1-regularized problems." SIAM journal on imaging sciences 2.2 (2009): 323-343.