Training Neural Networks Without Gradients: A Scalable ADMM Approach

HU TAO,

PEKING UNIVERSITY.

2017.6.19

Acknowledgements:Some of the elements are copied from the video:http://techtalks.tv/talks/training-neural-networks-without-gradients-a-scalable-admm-approach/62581/

Author Info

Gavin Taylor	TAYLOR@USNA.EDU
Ryan Burmeister	
Zheng Xu	XUZH@CS.UMD.EDU
Bharat Singh	BHARAT@CS.UMD.EDU
Ankit Patel	ABP4@RICE.EDU
Tom Goldstein	TOMG@CS.UMD.EDU

Overview

ADMM Introduction

ADMM in full-connected network

Experiment

Conventional Method

SGD,Momentum,AdaGrad,SAG,SVRG,SAGA

$$x^{k+1} = x^k - \alpha_k f'_{i_k}(x^k)$$

Drawbacks:

Node Satuation

Gradient decay

Very susceptible to condition number

ADMM Introduction

minimize H(u) + G(v)
subject to Au + Bv = b

Use Augmented Lagrange Multipliers

$$max_{\lambda} min_{u,v} H(u) + G(v) + < \lambda, b - Au - Bv > + \frac{\tau}{2} ||b - Au - Bv||^2$$

Alternating Direction Method of Multipliers

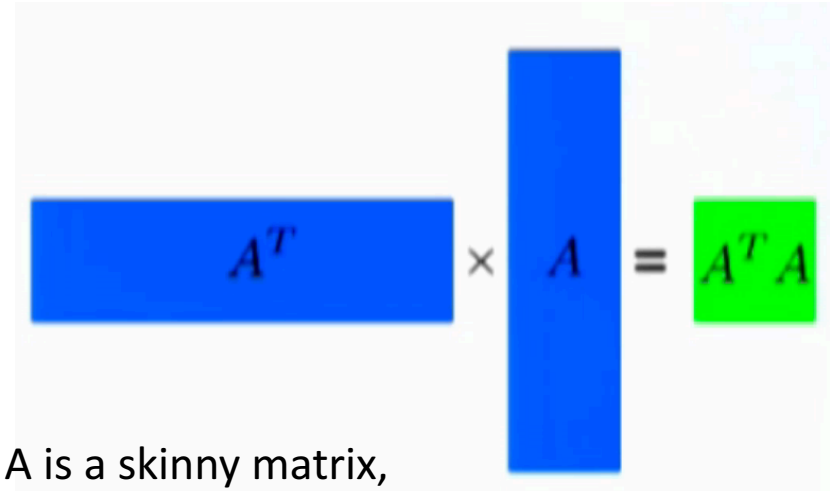
$$u_{k+1} = argmin_u H(u) + < \lambda_k, -Au > + \frac{\tau}{2} ||b - Au - Bv_k||^2$$
$$v_{k+1} = argmin_v G(v) + < \lambda_k, -Bv > + \frac{\tau}{2} ||b - Au_{k+1} - Bv||^2$$
$$\lambda_{k+1} = \lambda_k + \tau(b - Au_{k+1} - Bv_{k+1})$$

Transpose Reduction

$$\text{minimize } \frac{1}{2}||Ax - b||^2$$
$$x^* = (A^T A)^{-1} A^T b$$

$$A^T b = \sum A_i^T b_i$$
$$A^T A = \sum A_i^T A_i$$

$$A = \begin{pmatrix} A_1 \\ A_2 \\ \dots \\ A_N \end{pmatrix}$$



A is a skinny matrix, and height is too large ,need distributed

Overview

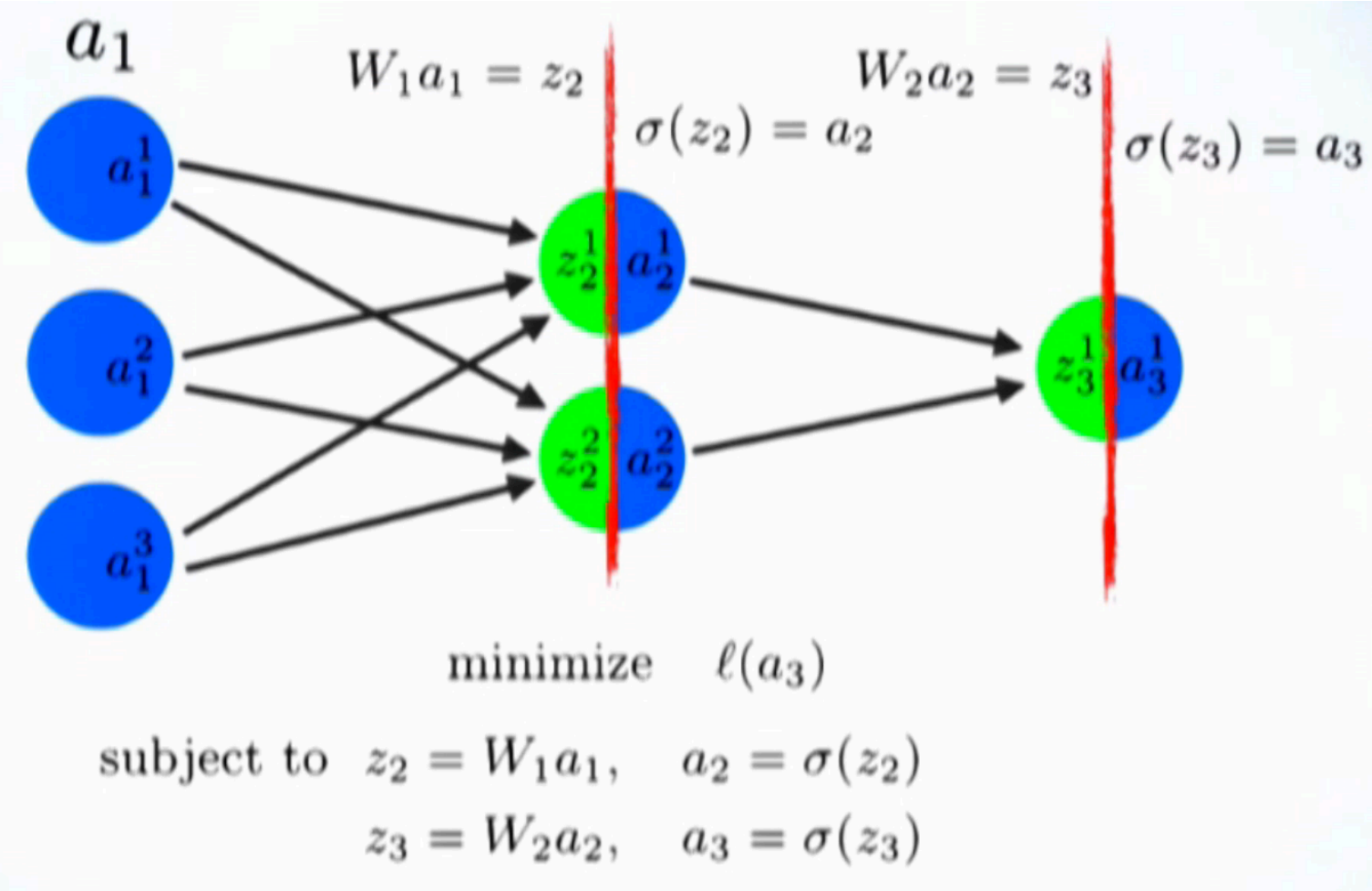
ADMM Introduction

ADMM in full-connected network

Experiment

Notations

data number:n=10000,
data dimension: m=648,
hidden layer 1 unit number: a=100
hidden layer 2 unit number: b=50
output layer unit number: 1
a_0 m-n dimension,
W_1 : a-m dimension
z_1 : a-n dimension
a_1 : a-n dimension
W_2 : b-a dimension
z_2 : b-n dimension
a_2 : b-n dimension
W_3 : 1-b dimension
z_3 : 1-n dimension
labels:y 1-n dimension
λ : 1-n dimension
activation function h is ReLu.



Initial optimization formulation

Minimization Steps

minimize $l(a_3, y) + \frac{1}{2}||z_2 - W_1 a_1||^2 + \frac{1}{2}||a_2 - \sigma(z_2)||^2 + \frac{1}{2}||z_3 - W_2 a_2||^2 + \frac{1}{2}||a_3 - \sigma(z_3)||^2$

Solve for weight W: least squares(convex)

Solve for activations a: least squares + ridge penalty(convex)

Solve for inputs z: coordinate-minimization (non-convex but global)

data number:n=10000,
data dimension: m=648,
hidden layer 1 unit number: a=100
hidden layer 2 unit number: b=50
output layer unit number: 1
a_0 m-n dimension,
W_1 : a-m dimension
z_1 : a-n dimension
a_1 : a-n dimension
W_2 : b-a dimension
z_2 : b-n dimension
a_2 : b-n dimension
W_3 : 1-b dimension
z_3 : 1-n dimension
labels:y 1-n dimension
λ : 1-n dimension
activation function h is ReLu.

Lagrange Multipliers

Classic ADMM

$$\text{minimize } l(a_3) + \tfrac{1}{2}||z_2 - W_1 a_1||^2 + \tfrac{1}{2}||a_2 - \sigma(z_2)||^2 + < \lambda_1, z_2 - W_1 a_1 > + < \lambda_2, a_2 - \sigma(z_2) > + \tfrac{1}{2}|||z_3 - W_2 a_2||^2 + \tfrac{1}{2}||a_3 - \sigma(z_3)||^2 + < \lambda_3, z_3 - W_2 a_2 > + < \lambda_4, a_3 - \sigma(z_3) >$$

…unstable because of non-linear constraints

Bregman Iteration

$$\text{minimize } l(a_3) + < \lambda, a_3 > + \tfrac{1}{2}||z_2 - W_1 a_1||^2 + \tfrac{1}{2}||a_2 - \sigma(z_2)||^2 + \tfrac{1}{2}|||z_3 - W_2 a_2||^2 + \tfrac{1}{2}||a_3 - \sigma(z_3)||^2$$

Interpretation (1)-Bregman Splitting

Bregman iteration (also known as the method of multipliers) is a general framework for solving constrained optimization problems. Methods of this type have been used extensively in the sparse optimization literature (Yin et al., 2008). Consider the general problem of minimizing

$$\min_u J(u) \quad \text{subject to} \quad Au = b \tag{9}$$

for some convex function J and linear operator A. Bregman iteration repeatedly solves

$$u^{k+1} \leftarrow \min D_J(u, u^k) + \frac{1}{2}\|Au - b\|^2 \tag{10}$$

where $p \in \partial J(u^k)$ is a (sub-)gradient of J at u^k, and $D_J(u, u^k) = J(u) - J(u^k) - \langle u - u^k, p\rangle$ is the so-called Bregman distance. The iterative process (10) can be viewed as minimizing the objective J subject to an inexact penalty that approximately obtains $Ax \approx b$, and then adding a linear term to the objective to weaken it so that the quadratic penalty becomes more influential on the next iteration.

Interpretation (2)-Lagrangian Multipliers

In addition to the Bregman interpretation, the proposed method can also be viewed as an approximation to the method of multipliers, which solves constrained problems of the form

$$\min_u J(u) \quad \text{subject to} \quad Au = b \tag{11}$$

for some convex function J and (possibly non-linear) operator A. In its most general form (which does not assume linear constraints) the method proceeds using the iterative updates

$$\begin{cases} u^{k+1} & \leftarrow \min J(u) + \langle \lambda^k, A(u) - b \rangle + \frac{\beta}{2} \|A(u) - b\|^2 \\ \lambda^{k+1} & \leftarrow \lambda^k + \partial_u \{ \frac{\beta}{2} \|A(u) - b\|^2 \} \end{cases}$$

where λ^k is a vector of Lagrange multipliers that is generally initialized to zero, and $\frac{\beta}{2} \|A(u) - b\|^2$ is a quadratic penalty term. After each minimization sub-problem, the gradient of the penalty term is added to the Lagrange multipliers. When the operator A is linear, this update takes the form $\lambda^{k+1} \leftarrow \lambda^k + \beta A^T (Au - b)$, which is the most common form of the method of multipliers.

Final Step

$$\begin{aligned} \underset{\{W_l\},\{a_l\},\{z_l\}}{\text{minimize}} \quad & \ell(z_L, y) \\ \text{subject to} \quad & z_l = W_l a_{l-1}, \text{ for } l = 1, 2, \cdots L \\ & a_l = h_l(z_l), \text{ for } l = 1, 2, \cdots L-1. \end{aligned} \tag{3}$$

$$\begin{aligned} \underset{\{W_l\},\{a_l\},\{z_l\}}{\text{minimize}} \quad & \ell(z_L, y) + \beta_L \|z_L - W_L a_{L-1}\|^2 \\ & + \sum_{l=1}^{L-1} \left[\gamma_l \|a_l - h_l(z_l)\|^2 + \beta_l \|z_l - W_l a_{l-1}\|^2\right] \end{aligned} \tag{4}$$

$$\begin{aligned} \underset{\{W_l\},\{a_l\},\{z_l\}}{\text{minimize}} \quad & \ell(z_L, y) \\ & + \langle z_L, \lambda \rangle + \beta_L \|z_L - W_L a_{L-1}\|^2 \\ & + \sum_{l=1}^{L-1} \left[\gamma_l \|a_l - h_l(z_l)\|^2 + \beta_l \|z_l - W_l a_{l-1}\|^2\right]. \end{aligned} \tag{5}$$

Final ADMM for NN

Algorithm 1 ADMM for Neural Nets

Input: training features $\{a_0\}$, and labels $\{y\}$,
Initialize: allocate $\{a_l\}_{l=1}^{L=1}$, $\{z_l\}_{l=1}^{L}$, and λ
repeat
for $l = 1, 2, \cdots, L-1$ **do**
$W_l \leftarrow z_l a_{l-1}^{\dagger}$
$a_l \leftarrow (\beta_{l+1} W_{l+1}^T W_{l+1} + \gamma_l I)^{-1} (\beta_{l+1} W_{l+1}^T z_{l+1} + \gamma_l h_l(z_l))$
$z_l \leftarrow \arg\min_z \gamma_l \|a_l - h_l(z)\|^2 + \beta_l \|z_l - W_l a_{l-1}\|^2$
end for
$W_L \leftarrow z_L a_{L-1}^{\dagger}$
$z_L \leftarrow \arg\min_z \ell(z, y) + \langle z_L, \lambda \rangle + \beta_L \|z - W_L a_{l-1}\|^2$
$\lambda \leftarrow \lambda + \beta_L (z_L - W_L a_{L-1})$
until converged

Overview

ADMM Introduction

ADMM in full-connected network

Experiment

Experiment

The ADMM approach was implemented in Python on a Cray XC30 **supercomputer** with Ivy Bridge processors, and communication between cores performed via MPI.

SGD, conjugate gradients, and L-BFGS are run as imple- mented in the Torch optim package on NVIDIA Tesla K40 GPUs.

SVHN

120,290 train data with 648d HOG Feature

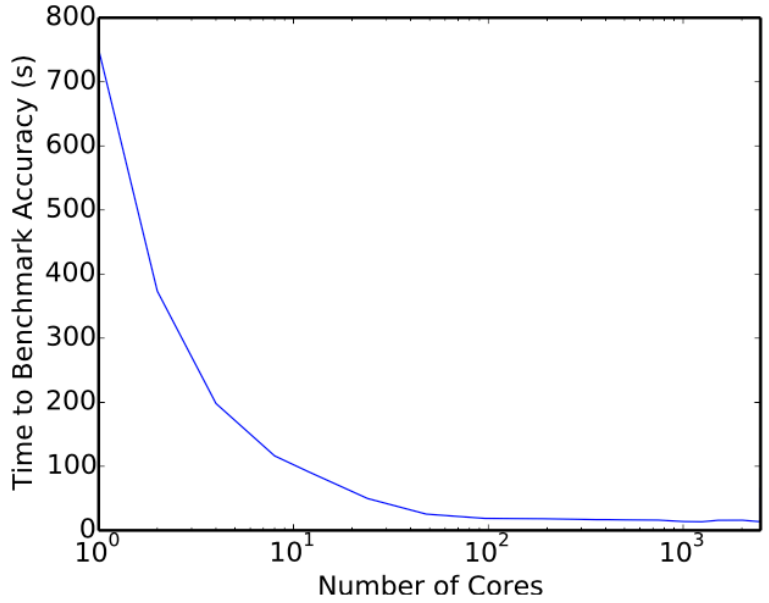
5,893 test data

Higgs

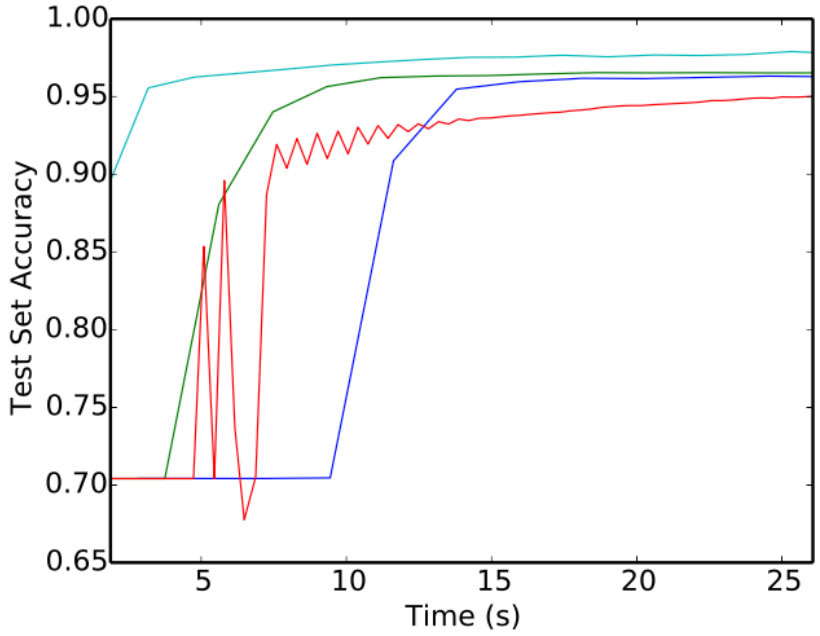
10,500,000 Train data with 28d feature

500,000 test data

Experiment-SVHN

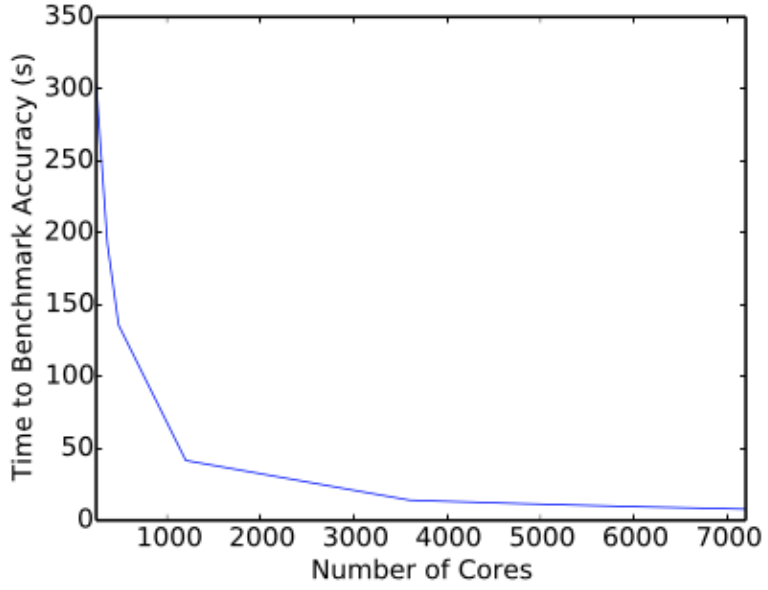


(a) **Time required for ADMM to reach 95% test accuracy vs number of cores**. This problem was not large enough to support parallelization over many cores, yet the advantages of scaling are still apparent (note the x-axis has log scale). In comparison, on the GPU, L-BFGS reached this threshold in 3.2 seconds, CG in 9.3 seconds, and SGD in 8.2 seconds.

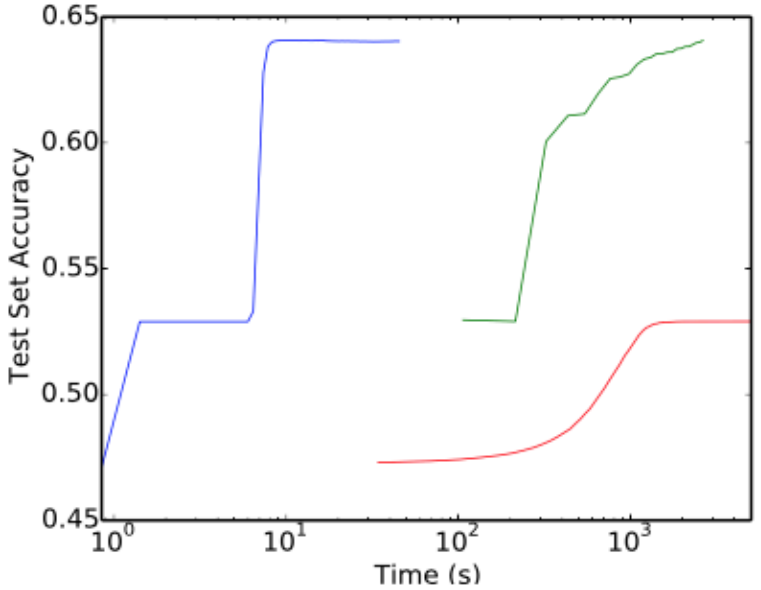


(b) **Test set predictive accuracy as a function of time in seconds** for ADMM on 2,496 cores (blue), in addition to GPU implementations of conjugate gradients (green), SGD (red), and L-BFGS (cyan).

Experiment-Higgs



(a) **Time required for ADMM to reach 64% test accuracy when parallelized over varying levels of cores**. L-BFGS on a GPU required 181 seconds, and conjugate gradients required 44 minutes. SGD never reached 64% accuracy.



(b) **Test set predictive accuracy as a function of time** for ADMM on 7200 cores (blue), conjugate gradients (green), and SGD (red). Note the x-axis is scaled logarithmically.

Many Typos

Algorithm 1:

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

$$z_L \leftarrow \arg\min_z \ell(z, y) + \langle z_L, \lambda \rangle + \beta_L \|z - W_L a_{L-1}\|^2$$

Loss function:

$$\ell(z, a) = \begin{cases} \max\{1 - z, 0\}, \text{ when } a = 1, \\ \max\{z, 0\}, \text{ when } a = 0. \end{cases}$$

Reference

[1].Bregman Iterative Algorithms for 1-Minimization with Applications to Compressed Sensing

[2]. http://icml.cc/2016/reviews/1215.txt

[3].Unwrapping ADMM: Efficient Distributed Computing via Transpose Reduction

Any Ideas?

Apply the algorithm to Convolutio, Reinforcement Learning.