<Training Neural Networks Without Gradients: A Scalable ADMM Approach> Technical Report

Tao Hu
Department of Computer Science
Peking University
No.5 Yiheyuan Road Haidian District, Beijing, P.R.China
taohu@pku.edu.cn

Abstract

This a technical report about "Training Neural Networks Without Gradients: A Scalable ADMM Approach"[3]. **My works mainly about read through the paper; extend the extra knowledge about these paper; think and list all detail algorithm; implement the algorithm with python.** the result is not very satisfactory event though the loss can be convergent. the final code is released at https://github.com/dongzhuoyao/admm_nn.

1 ADMM introduction

the standard problem form of Alternating Direction Method of Multipliers(ADMM) is :

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

firstly, we can write the Augmented Lagrangian as:

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

The detail procedure of ADMM is:

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

generally speaking, for the above sub-problem. we can solve by three ways:(1) linearized ADMM[2]. (2). gradient descent, use the result of gradient descent instead of the optimal of sub-problem[5]. (3). convert it into dual problem which maybe easier to solve

1.1 Scalability

.

The auther realize scalability by a trick called transpose reduction[1]. take sparse least square problem as a example.

$$\text{minimize } \frac{1}{2}||Ax - b||^2$$

Author Info: Taohu ,Peking University

the solution is obvious. $x^* = (A^TA)^{-1}A^Tb$. usually A is skinny matrix that need huge computation. thus distributed computation can be utilized.

$$A^Tb = \sum A_i^T b_i$$
$$A^TA = \sum A_i^T A_i$$

1.2 Problem Simplification

The standard Lagrangian Multipliers form of this problem is:

minimize $l(a_3) + \frac{1}{2}||z_2 - W_1a_1||^2 + \frac{1}{2}||a_2 - \sigma(z_2)||^2 + < \lambda_1, z_2 - W_1a_1 > + < \lambda_2, a_2 - \sigma(z_2) >$
$+\frac{1}{2}||z_3 - W_2a_2||^2 + \frac{1}{2}||a_3 - \sigma(z_3)||^2 + < \lambda_3, z_3 - W_2a_2 > + < \lambda_4, a_3 - \sigma(z_3) >$

However, the number of the constraints of the problem is too large to optimize, the author uses a trick that it just place the constraints into the optimizer as penalty items.

minimize $l(a_3, y) + \frac{1}{2}||z_2 - W_1a_1||^2 + \frac{1}{2}||a_2 - \sigma(z_2)||^2 + \frac{1}{2}||z_3 - W_2a_2||^2 + \frac{1}{2}||a_3 - \sigma(z_3)||^2$

for weight w_l: the problem is convex, and is a least square problem thus exists a closed-form solution. for activation a_l: it can also transfered into a convex least square problem. for inputs z_l: as it involves the activation function which is non-convex, we can not solve it directly. luckily, each dimension of the problem can splitting out and get solved respectively.

non-linear constraints make the problem unstable. so the author add a extra item to make it stable. which can be interpreted as Bregman Iteration[4] or Lagrangian Multipliers.

min $l(a_3) + < \lambda, a_3 > + \frac{1}{2}||z_2 - W_1a_1||^2 + \frac{1}{2}||a_2 - \sigma(z_2)||^2 + \frac{1}{2}|||z_3 - W_2a_2||^2 + \frac{1}{2}||a_3 - \sigma(z_3)||^2$

1.3 interpretation: Bregman Iteration

Firstly, it can be interpretated as Bregman Iteration. The form of Bregmen Splitting is:

$$min_u J(u) + H(u)$$

the procedure is:

Algorithm 1 Bregman Iteration

Inputs:
J(.),H(.)
Initialize:
k=0,$u^0 = 0, p^0 = 0$
while not converge **do**
$u^{k+1} = argmin_u D_J^{p^k}(u, u^k) + H(u)$
$p^{k+1} = p^k - \nabla H(u^{k+1}) \in J(u^{k+1})$
$k = k + 1$
end while

The $D_J^{p^k}$ is called Bregman Distance, and $D_J^{p^k} = J(u) - J(u^k) - < u - u^k, p^k >$. in the circumstance of ADMM. $J(u) = l(a_L, y), H(u) = z_L - w_La_{L-1}$ as we minimize by u, so u-sub-problem can be simplified as $u^{k+1} = argmin_u J(u) - < u, p^k > + H(u)$, which corresponds to

min $l(a_3) + < \lambda, a_3 > + \frac{1}{2}||z_2 - W_1a_1||^2 + \frac{1}{2}||a_2 - \sigma(z_2)||^2 + \frac{1}{2}|||z_3 - W_2a_2||^2 + \frac{1}{2}||a_3 - \sigma(z_3)||^2$

2 Detail Algorithm

As the experiment in the paper using supercomputer which is currently impossible for me. so I generate some small data to test the algorithm. in detail, I choose 2 2-D gaussian distribution data with center (0.2,0.2) and (0.1,0.1), variance 0.01 both. each class have 5000 data points. the detailed data distribution is in Figure1. I wish the algorithm can successfully classify these two classes.

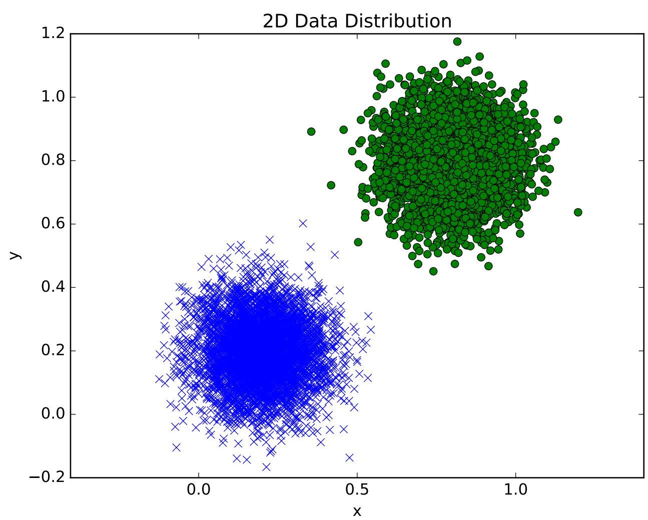

Figure 1: 2D data distribution. This figure is best viewed in colour.

so the feature dimension is 2, I set two hidden layer in the neural network with 10,5 dimensions. as it's a binary classification. the dimension of last layer is 1. specially, no extra layer such as activation layer or sigmoid layer appended to the last layer.

the detailed Algorithm is in Algorithm 2.

Algorithm 2 ADMM_NN

Inputs:

data number:n=10000,
data dimension: m=2,
hidden layer 1 unit number: a=10
hidden layer 2 unit number: b=5
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.

Initialize:

allocate $\{a_l\}_{l=1}^{L}$,$\{z_l\}_{l=1}^{L}$ with i.i.d Gaussian Distribution,and λ

Cache: $a_0^{\dagger}$

Warm Start:

for i=1,..,100 **do**

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

$W_l \leftarrow z_l a_{l-1}^{\dagger}$

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

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

end for

$W_L \leftarrow z_L a_{L-1}^{\dagger}$

$z_L \leftarrow argmin_z l(z,y) + < z, \lambda > + \beta_L ||z - W_L a_{L-1}||^2$

end for

Start ADMM:

while not converge **do**

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

do $W_l \leftarrow z_l a_{l-1}^{\dagger}$

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

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

end for

$W_L \leftarrow z_L a_{L-1}^{\dagger}$

$z_L \leftarrow argmin_z l(z,y) + < z, \lambda > + \beta_L ||z - W_L a_{L-1}||^2$

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

end while

the update of a_l, z_l have closed-form solution. however, for w_l don't have direct solution since the activation function is piecewise function. after classified discussion according to activation. we have the following strategy.

z_L argmin procedure(when l is loss in the paper):

$$z_l = \begin{cases} max(\frac{a_l \gamma_l + W_l a_{l-1} \beta_l}{\gamma_l + \beta_l}, 0) & \text{z} \geq 0 \\ min(W_l a_{l-1}, 0) & \text{z} \leq 0 \end{cases}$$

choose one minimizer z from two choices.

for z_L:

when $y_i = 0$:
$f(z) = \beta z^2 - (2\beta w\_a - \lambda)z + max(z, 0)$
$z^* = max(\frac{2\beta w\_a - \lambda - 1}{2\beta}, 0)$ or
$z^* = min(\frac{2\beta w\_a - \lambda}{2\beta}, 0)$
choose one which make f(z) smaller.

when $y_i = 1$:
$f(z) = \beta z^2 - (2\beta w\_a - \lambda)z + max(1 - z, 0)$
$z^* = max(\frac{2\beta w\_a - \lambda}{2\beta}, 1)$ or
$z^* = min(\frac{2\beta w\_a - \lambda + 1}{2\beta}, 1)$
choose one which make f(z) smaller.

z_L argmin procedure(when l is a standard hinge loss):

when $y_i = -1$: $f(z) = max(1 + z) + \lambda z + \beta(z^2 - 2w\_az)$
$z^* = min(\frac{2\beta w\_a - \lambda}{2\beta}, -1)$ or
$z^* = max(\frac{2\beta w\_a - \lambda - 1}{2\beta}, -1)$ choose one which make f(z) smaller.

when $y_i = 1$:
$f(z) = max(1 - z, 0) + \lambda z + \beta(z^2 - 2w\_az)$
$z^* = min(\frac{2\beta w\_a - \lambda + 1}{2\beta}, 1)$ or
$z^* = max(\frac{2\beta w\_a - a - \lambda}{2\beta}, 1)$
choose one which make f(z) smaller.

3 Conclusion

After implement the algorithm in python, the loss show convergence. however, the prediction accuracy in test data is nearly 50%, which is as normal as "Throw the dice". Everywhere in the algorithm and code have been checked, but no solution is found. even so, I think this is a promising method where I must miss something important.

Acknowledgments

References

[1] Tom Goldstein, Gavin Taylor, Kawika Barabin, and Kent Sayre. Unwrapping admm: Efficient distributed computing via transpose reduction. In *19th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2016.

[2] Yuyuan Ouyang, Yunmei Chen, Guanghui Lan, and Eduardo Pasiliao Jr. An accelerated linearized alternating direction method of multipliers. *SIAM Journal on Imaging Sciences*, 8(1):644–681, 2015.

[3] Gavin Taylor, Ryan Burmeister, Zheng Xu, Bharat Singh, Ankit Patel, and Tom Goldstein. Training neural networks without gradients: A scalable admm approach. In *International Conference on Machine Learning*, 2016.

[4] W Yin, S Osher, D Goldfarb, and J Darbon. Bregman iterative algorithms for l1-minimization with applications to compressed sensing: Siam journal on imaging sciences, 1 (1), 143–168. *CrossRef Google Scholar*, 2008.

[5] Wenliang Zhong and James Kwok. Fast stochastic alternating direction method of multipliers. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 46–54, 2014.