Training Neural Networks Without Gradients: A Scalable ADMM Approach

ICML2016

Gavin Taylor, Ryan Burmeister et al.

Reporter: Minjie Cheng

June 7, 2023

Overview

- Problem Formulation-Typical neural networks
- The limitations of SGD and backprop
- ADMM-based method without gradients
- Experiments
- Advantages and Limitations

Problem Formulation

- Consider a typical neural network consists of L layers:

$$f(\boldsymbol{a}_0; \boldsymbol{W}) = \boldsymbol{W}_l(h_{(l-1)}(\ldots \boldsymbol{W}_3(h_2(\boldsymbol{W}_2(h_1(\boldsymbol{W}_1 \boldsymbol{a}_0)))))) \tag{1}$$

  - $\boldsymbol{W}_l$ is a linear operator.
  - h_l is a non-linear neural activation function
- We define the loss function $\mathscr{L}$

$$\min_{\boldsymbol{W}} \mathscr{L}(f(\boldsymbol{a}_0; \boldsymbol{W}), \boldsymbol{y}) \tag{2}$$

- Most network are trained by SGD with backpropagation.

The limitations of SGD and backpropagation

- Gradient Vanishing and Exploding
- Sensitivity to Hyperparameters and Initialization
- Convergence to Local Optima
- Lack of Parallelization in CPU
  - Data Dependency: In SGD, each parameter update relies on the gradient computed in the previous step.
  - Thread Scheduling Overhead: Parallel computing on multi-core CPUs involves overhead related to thread scheduling and management.
  - For example, for several experiments reported in (Dean et al., 2012), the distributed SGD method runs **slower** with 1500 cores than with 500 cores.

ADMM (Alternating Direction Method of Multipliers)

- Definition:

$$\min_{x,y} f(x) + g(y) \quad s.t. Ax + By = C \tag{3}$$

- Augmented Lagrangian Form:$L(x, y, \lambda)$

$$\min_{x,y,\lambda} f(x) + g(y) + < \lambda, Ax + By - C > + \rho B_\phi(Ax - C, -By) \tag{4}$$

- The iterative optimization process of ADMM is as follows:

for $i = 1...T$:

$$x^{t+1} = \arg\min_x L(x, y^{(t)}, \lambda^{(t)}) \tag{5}$$

$$y^{t+1} = \arg\min_x L(x^{x+1}, y, \lambda^{(t)}) \tag{6}$$

$$\lambda^{t+1} = \lambda^(t) + (Ax^{(t+1)} + By^{(t+1))} \tag{7}$$

Proposed Method

- Typical neural network and loss function

$$f(\boldsymbol{a}_0; \boldsymbol{W}) = \boldsymbol{W}_l(h_{(l-1)}(...\boldsymbol{W}_3(h_2(\boldsymbol{W}_2(h_1(\boldsymbol{W}_1\boldsymbol{a}_0)))))) \tag{8}$$

$$\min_{\boldsymbol{W}} \mathscr{L}(f(\boldsymbol{a}_0; \boldsymbol{W}), \boldsymbol{y}) \tag{9}$$

- The view of optimization:

$$\begin{aligned} &\min_{\{\boldsymbol{W}_l\},\{\boldsymbol{a}_l\},\{\boldsymbol{z}_l\}} \mathscr{L}(\boldsymbol{z}_L; \boldsymbol{y}) \\ s.t. \quad & \boldsymbol{z}_l = \boldsymbol{W}_l\boldsymbol{a}_{l-1}, \text{ for } l = 1, 2, ...L \\ & \boldsymbol{a}_l = \boldsymbol{h}_l\boldsymbol{z}_l, \text{ for } l = 1, 2, ...L-1 \end{aligned} \tag{10}$$

Proposed Method

- Relax the constraints by adding an l_2 penalty function to the objective funciton.

$$\min_{\{\boldsymbol{W}_l\},\{\boldsymbol{a}_l\},\{\boldsymbol{z}_l\}} \mathscr{L}(\boldsymbol{z}_L;\boldsymbol{y}) + \beta_L\|\boldsymbol{Z}_L - \boldsymbol{W}_L\boldsymbol{a}_{L-1}\|^2 + \sum_{l=1}^{L-1}[\gamma_l\|\boldsymbol{a}_l - \boldsymbol{h}_l(\boldsymbol{z}_l)\|^2 + \beta_l\|\boldsymbol{z}_l - \boldsymbol{W}_l(\boldsymbol{a}_{l-1})\|^2], \tag{11}$$

- Add Lagrange multiplier term based on Bregman iteration.

$$\min_{\{\boldsymbol{W}_l\},\{\boldsymbol{a}_l\},\{\boldsymbol{z}_l\}} \mathscr{L}(\boldsymbol{z}_L;\boldsymbol{y}) + <\boldsymbol{Z}_L,\boldsymbol{\lambda}> + \beta_L\|\boldsymbol{Z}_L - \boldsymbol{W}_L\boldsymbol{a}_{L-1}\|^2 + \sum_{l=1}^{L-1}[\gamma_l\|\boldsymbol{a}_l - \boldsymbol{h}_l(\boldsymbol{z}_l)\|^2 + \beta_l\|\boldsymbol{z}_l - \boldsymbol{W}_l(\boldsymbol{a}_{l-1})\|^2], \tag{12}$$

- Now update $\{\boldsymbol{W}_l\}$, $\{\boldsymbol{a}_l\}$, $\{\boldsymbol{z}_l\}$ and λ

1-Weight update

- Update $\{\boldsymbol{W}_l\}$, holding all other variables fixed.

$$F(\boldsymbol{W}_l) = \min_{\{\boldsymbol{W}_l\},\{\boldsymbol{a}_l\},\{\boldsymbol{z}_l\}} \mathscr{L}(\boldsymbol{z}_L;\boldsymbol{y}) + \beta_L\|\boldsymbol{Z}_L - \boldsymbol{W}_L\boldsymbol{a}_{L-1}\|^2 + \sum_{l=1}^{L-1}[\gamma_l\|\boldsymbol{a}_l - \boldsymbol{h}_l(\boldsymbol{z}_l)\|^2 + \beta_l\|\boldsymbol{z}_l - \boldsymbol{W}_l(\boldsymbol{a}_{l-1})\|^2], \tag{11}$$

- This is simply a least squares problem, $\frac{\partial}{\partial \boldsymbol{W}_l}F(\boldsymbol{W}_l) = 0$, the solution is $\boldsymbol{W}_l = \boldsymbol{z}_l\boldsymbol{a}_{l-1}^{\dagger}$

2-Activations update

- Update $\{\boldsymbol{a}_l\}$

$$F(\boldsymbol{W}_l) = \min_{\{\boldsymbol{W}_l\},\{\boldsymbol{a}_l\},\{\boldsymbol{z}_l\}} \mathscr{L}(\boldsymbol{z}_L;\boldsymbol{y}) + \beta_L\|\boldsymbol{Z}_L - \boldsymbol{W}_L\boldsymbol{a}_{L-1}\|^2 + \sum_{l=1}^{L-1}[\gamma_l\|\boldsymbol{a}_l - \boldsymbol{h}_l(\boldsymbol{z}_l)\|^2 + \beta_l\|\boldsymbol{z}_l - \boldsymbol{W}_l(\boldsymbol{a}_{l-1})\|^2], \tag{11}$$

- Minimize equation 13 for each layer

$$\beta_l\|\boldsymbol{Z}_{l+1} - \boldsymbol{W}_{l+1}\boldsymbol{a}_l\|^2 + \gamma_l\|\boldsymbol{a}_l - \boldsymbol{h}_l(\boldsymbol{z}_l)\|^2 \tag{13}$$

- The solutin is

$$(\beta_{l+1}\boldsymbol{W}_{l+1}^T\boldsymbol{W}_{l+1} + \gamma_l\boldsymbol{I})^{-1}(\beta_{t+1}\boldsymbol{W}_{t+1}^T\boldsymbol{z}_{l+1} + \gamma_l h_l(\boldsymbol{z}_l)) \tag{14}$$

3-Outputs update

- Update $\{\boldsymbol{z}_l\}$

$$F(\boldsymbol{W}_l) = \min_{\{\boldsymbol{W}_l\},\{\boldsymbol{a}_l\},\{\boldsymbol{z}_l\}} \mathscr{L}(\boldsymbol{z}_L; \boldsymbol{y}) + \beta_L \|\boldsymbol{Z}_L - \boldsymbol{W}_L \boldsymbol{a}_{L-1}\|^2 + \sum_{l=1}^{L-1} [\gamma_l \|\boldsymbol{a}_l - \boldsymbol{h}_l(\boldsymbol{z}_l)\|^2 + \beta_l \|\boldsymbol{z}_l - \boldsymbol{W}_l(\boldsymbol{a}_{l-1})\|^2], \tag{11}$$

- Minimize equation 15 for each layer

$$\min_{\boldsymbol{z}} \gamma_l \|\boldsymbol{a}_l - \boldsymbol{h}_l(\boldsymbol{z})\|^2 + \beta_l \|\boldsymbol{z} - \boldsymbol{W}_l(\boldsymbol{a}_{l-1})\|^2 \tag{15}$$

- Suppose $h(z)$ is Relu, because the entries in $\boldsymbol{z}_l$ are de-coupled, it can be solved in closed form(id-else logic).

4-Lagrange multiplier update

- After minimizing for $\{\boldsymbol{W}_l\}$,$\{\boldsymbol{a}_l\}$,$\{\boldsymbol{z}_l\}$, the Lagrange multiplier update is give simply by:

$$\boldsymbol{\lambda} = \boldsymbol{\lambda} + \beta_L(\boldsymbol{z}_L - \boldsymbol{W}_L(\boldsymbol{a}_{L-1})) \tag{16}$$

Algorithm-ADMM for neural nets

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

Distributed implementation using data parallelism

- The main advantage of the proposed method is its high degree of scalability.
- We update $\boldsymbol{W}_l$ by $\boldsymbol{W}_l = \boldsymbol{z}_l \boldsymbol{a}_{l-1}^{\dagger}$
- Consider distributing the algorithm across N worker nodes.
- The ADMM method is scaled using a data parallelization strategy, in which different nodes store activations and outputs corresponding to different subsets of the training data.
- For each layer, the $\boldsymbol{z}_l$, $\boldsymbol{a}_l$ matrix is broken into **columns** subsets.

Distributed implementation using data parallelism

- Parallel Weight update

$$\begin{aligned} \boldsymbol{W}_l = \boldsymbol{z}_l \boldsymbol{a}_{l-1}^{\dagger} &= \boldsymbol{z}_l (\boldsymbol{a}_l^T (\boldsymbol{a}_l \boldsymbol{a}_l^T)^{-1}) \\ &= (\sum_{n=1}^{N} \boldsymbol{z}_l^n (\boldsymbol{a}_l^n)^T)(\sum_{n=1}^{N} \boldsymbol{a}_l^n (\boldsymbol{a}_l^n)^T)^{-1} \end{aligned} \tag{17}$$

- Parallel Activations update

$$\boldsymbol{a}_l^n = (\beta_{l+1} \boldsymbol{W}_{l+1}^T \boldsymbol{W}_{l+1} + \gamma_l \boldsymbol{I})^{-1} (\beta_{t+1} \boldsymbol{W}_{t+1}^T \boldsymbol{z}_{l+1}^n + \gamma_l h_l(\boldsymbol{z}_l^n)) \tag{18}$$

Parallelism-Example

$$\min_{\{w_l\},\{a_l\},\{z_l\}} L(z_L, y)$$

$$\text{s.t.}\quad z_l = W_l a_{l-1} \quad \text{for } l = 1, 2 \dots L$$

$$a_l = h_l(z_l) \quad \text{for} \dots l = 1, 2, L-1$$

parallelism

$z_l \in R^{n\times d}$

$a_l \in R^{n\times d}$

$W^l \in R^{n\times n}$

columns

$$a_0 \in R^{5\times 4}\quad \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \\ a_{40} & a_{41} & a_{42} & a_{43} \end{bmatrix} \qquad W_1 : R^{5\times 5}\quad \begin{bmatrix} W_{00} & W_{01} & W_{02} & W_{03} & W_{04} \\ W_{10} & W_{11} & W_{12} & W_{13} & W_{14} \\ W_{20} & W_{21} & W_{22} & W_{23} & W_{24} \\ W_{30} & W_{31} & W_{32} & W_{33} & W_{34} \\ W_{40} & W_{41} & W_{42} & W_{43} & W_{44} \end{bmatrix} \longrightarrow z_1 \begin{bmatrix} z_{00} & z_{01} & z_{02} & z_{03} \\ z_{10} & z_{11} & z_{12} & z_{13} \\ z_{20} & z_{21} & z_{22} & z_{23} \\ z_{30} & z_{31} & z_{32} & z_{33} \\ z_{40} & z_{41} & z_{42} & z_{43} \end{bmatrix}$$

$$(a_1, z_1) \longrightarrow a_1 \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \\ a_{40} & a_{41} & a_{42} & a_{43} \end{bmatrix}$$

$$W \leftarrow z_L a_{L-1}^{+} = (z_L a_l^T)(a_l a_l^T)^{-1} = (z_1 a_0^T)(a_0 a_0^T)^{-1}$$

$$= \left(\sum_{l=1}^{n} z_l^n (a_l^n)^T\right)\left(\sum_{n=1}^{N} a_l^n (a_l^n)^T\right)^{-1}$$

$z_L \in R^{n\times d}$, $a_{L-1} \in R^{n\times d}$, $W \in R^{n\times n}$; $n\times 1 \times 1 \times n$

$$z a_0^T = \begin{bmatrix} z_{00} & z_{01} & z_{02} & z_{03} \\ z_{10} & z_{11} & z_{12} & z_{13} \\ z_{20} & z_{21} & z_{22} & z_{23} \\ z_{30} & z_{31} & z_{32} & z_{33} \\ z_{40} & z_{41} & z_{42} & z_{43} \end{bmatrix} \begin{bmatrix} a_{00} & a_{10} & a_{20} & a_{30} & a_{40} \\ a_{01} & a_{11} & a_{21} & a_{31} & a_{41} \\ a_{02} & a_{12} & a_{22} & a_{32} & a_{42} \\ a_{03} & a_{13} & a_{23} & a_{33} & a_{43} \end{bmatrix}$$

distributed implementation using data parallelism.

$$= \begin{bmatrix} z_{00} a_{00} + z_{10} a_{10} + z_{20} a_{20} + z_{30} a_{30} + z_{40} a_{40} \\ \\ \end{bmatrix}$$

Distributed implementation using data parallelism

- Parallel Outputs update

$$\min_{\boldsymbol{z}_l^n} \gamma_l \|\boldsymbol{a}_l^n - \boldsymbol{h}_l(\boldsymbol{z}_l^n)\|^2 + \beta_l \|\boldsymbol{z}_l^n - \boldsymbol{W}_l(\boldsymbol{a}_{l-1}^n)\|^2 \tag{19}$$

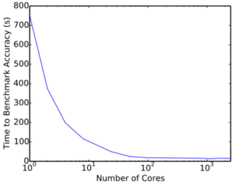
- Parallel Lagrange multiplier update

$$\boldsymbol{\lambda}^n = \boldsymbol{\lambda}^n + \beta_L(\boldsymbol{z}_L^n - \boldsymbol{W}_L(\boldsymbol{a}_{L-1}^n)) \tag{20}$$

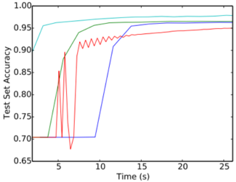
Experiments

- Binary classification task.
- Loss function: $f(z,y) = \begin{cases} \max\{1-z, 0\}, & y=1 \\ \max\{1+z, 0\}, & y=-1 \end{cases}$
- Two datasets
  - SVHN: train:120290 datapoints, test: 5893
  - Higgs: train:10500000 datapoints, test:500000
- Warm start without Lagrange multiplier updatas
- Initalize $\{\boldsymbol{a}_l\}$, $\{\boldsymbol{z}_l\}$ with Gaussian random variables.
- Baselines: SGD, conjugate gradients, L-BFGS.

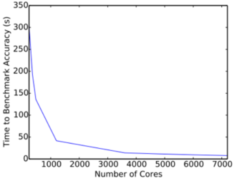
Experiments



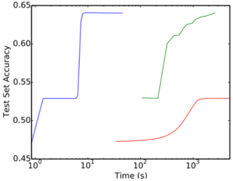
(a) **Time required for ADMM to reach 95% test accuracy vs number of cores**. This problem was not large enough to support parallelization over many cores, yet the advantages of scaling are still apparent (note the x-axis has log scale). In comparison, on the GPU, L-BFGS reached this threshold in 3.2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



(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.

Conclusion

- A new paradigm for updating neural networks without gradients.
- Emphasizing CPU parallelism may not guarantee performance.