ADMM in PyTorch

Bhushan Sonawane, Mihir Chakradeo, Nishant Borude,
Sri Haindavi
Department of Computer Science
Stony Brook University
Stony Brook, NY 11794
{bsonawane, mchakradeo, nborude, skoppuravuri}
@cs.stonybrook.edu

May 7, 2018

Abstract

The Alternating Direction Method of Multipliers (ADMM) is an optimization algorithm which solves the objective function by breaking it into smaller pieces. This allows us to pass the smaller pieces to separate machines/processors and solve them parallely. We implemented the ADMM serialized and parallel code for Lasso and Ridge objective functions in PyTorch framework.

1 Introduction

Alternating Direction Method of Multipliers [1] is a simple but powerful algorithm (framework) which works really well for distributed convex optimization. The core idea behind it is broadcasting and gathering, where broadcasting means splitting the objective function into smaller parts and giving it to multiple processors/machines for solving, and gathering means gathering the results from these processors/machines and combining them for getting the update rule for the next iteration. The advantage of this algorithm is that there are no constraints on the objective function other than convexity. We do not even need the loss function to be differentiable.

The flow of the report is as follows: We first introduce the existing state of the art implementations for ADMM. Followed by that, we explain the working of ADMM and then move on to our contribution, which is implementation of ADMM for Lasso and Ridge in PyTorch (serial and parallel) and their comparison with sklearn's implementation. Finally, we conclude our report and state our ongoing work and future scope.

2 Previous Work

Previous implementations of ADMM have been task specific. It is difficult to write one generic framework that can deal with the different number of ways in which ADMM can be applied to the problem. There is no ADMM implementation in the well known deep learning frameworks such as PyTorch and Tensorflow. We found the following implementations:

cvxflow

The computer vision group from Stanford has a tensorflow based unofficial implementation of ADMM [2]. This works on specific problems and is cumbersome to setup and use.

ADMM python implementation by Niru Maheswaranathan

His implementation consists of some standalone examples such as Lasso, Woodbury etc in Python using Matlab as backend [3].

3 ADMM Algorithm

The ADMM objective is solving a sum of functions with linear constraints. In general, for parallelizing the minimization process, we divide the sum of the objective function into N parts as follows:

$$\begin{aligned} \min_{x^{(i)},z} \quad & \sum_{i=1}^{N} f_i(x^{(i)}) \\ \text{st} \quad & z = x^{(i)} \end{aligned}$$

The algorithm consists of updating three main steps:

1. **Minimization of x**

$$x_{t+1}^{(i)} = \underset{x}{\text{argmin}} \quad f_i(x) + \nu_t^{(i)}(x^{(i)} - z_t) + \frac{\rho}{2}\|x^{(i)} - z_t\|_2^2$$

2. **Minimization of z**

$$z_{t+1}^{(i)} = \quad \frac{1}{N}\sum_{i=1}^{N}(x_{t+1}^{(i)} + \frac{1}{\rho}\nu_t^{(i)})$$

$$z_{t+1} = \quad \bar{x}_{t+1} + \frac{1}{\rho}\bar{\nu_t}$$

where, $\bar{x}_{t+1} = \frac{1}{N}\sum_{i=1}^{N} x_{t+1}^{(i)}$ and $\bar{\nu}_{t+1} = \frac{1}{N}\sum_{i=1}^{N} \nu_{t+1}^{(i)}$

3. **Update of dual variable ν_t**

$$\begin{aligned}\nu_{t+1}^{(i)} = &\quad \nu_t^{(i)} + \rho(x_{t+1}^{(i)} - z_{t+1}) \\ \bar{\nu}_{t+1} = &\quad \bar{\nu}_t + \rho(\bar{x}_{t+1} - z_{t+1})\end{aligned}$$

Now, the updates for each $x_{t+1}^{(i)}$ and $\nu_{t+1}^{(i)}$ can be done in parallel

4 LASSO Regression

The LASSO objective function is a modified Linear Regression objective function. It involves an L1 regularization term. So the modified objective is:

$$\min_x \frac{1}{2}\|Ax - b\|_2^2 + \lambda\|x\|_1$$

where $x \in \mathbb{R}^d$ is a vector of weights of the model that need to tuned and $A \in \mathbb{R}^{n\text{x}d}$ is a matrix of all the input features and $b \in \mathbb{R}^n$ is the vector of all outputs.

The above function can be written in multiple ways as an ADMM objective function subject to linear constraints. One way to do that is the Serial Vectorized implementation which is as follows:

4.1 Vectorized Serial Implementation

We can split the objective function by renaming one of the two variables and introduce an equality constraint to bring in the ADMM form. The modified LASSO objective is:

$$\begin{aligned}\min_{x,z} \quad & \frac{1}{2}\|Ax - b\|_2^2 + \lambda\|z\|_1 \\ \text{st} \quad & x = z\end{aligned}$$

The update rule for each of the two variables is as follows:

1. **Minimization of x**

$$x_{t+1} = \underset{x}{\text{argmin}} \quad \frac{1}{2}\|Ax - b\|_2^2 + \frac{\rho}{2}\|x - z_t + \frac{1}{\rho}\nu_t\|_2^2$$

which has a closed form solution

$$x_{t+1} = (A^TA + \rho I)^{-1}(A^Tb + \rho z_t - \nu_t)$$

2. **Minimization of z**

$$z_{t+1} = \underset{z}{\text{argmin}} \quad \lambda\|z\|_1 + \frac{\rho}{2}\|x_{t+1} - z + \frac{1}{\rho}\nu_t\|_2^2$$

$$z_{t+1} = x_{t+1} + \frac{\nu_t}{\rho} - \frac{\lambda}{\rho}\text{sign(z)}$$

3. **Update of dual variable** ν_t

$$\nu_{t+1} = \quad \nu_t + \rho(x_{t+1} - z_{t+1})$$

4.2 Parallel Implementation

We can rewrite the objective function of LASSO for running in distributed/parallel fashion by splitting across all the samples:

$$\begin{aligned} \min_{x^{(i)},z} \quad & \frac{1}{2}\sum_{i=1}^{N} \|A^{(i)}x^{(i)} - b^{(i)}\|_2^2 + \lambda\|z\|_1 \\ \text{st} \quad & x^{(i)} - z = 0, \ for\ i = 1, ..., N \end{aligned}$$

The modified update rule for each variable is as follows:

1. **Minimization of** x_i

$$x_{t+1}^{(i)} = \underset{x^{(i)}}{\text{argmin}} \quad \frac{1}{2}\|A^{(i)}x^{(i)} - b^{(i)}\|_2^2 + \frac{\rho}{2}\|x^{(i)} - z_t + \frac{1}{\rho}\nu_t^{(i)}\|_2^2$$

which has a closed form solution

$$x_{t+1}^{(i)} = (A^{(i)T}A^{(i)} + \rho I)^{-1}(A^{(i)T}b^{(i)} + \rho z_t - \nu_t^{(i)}), \ for\ i = 1, ..., N$$

2. **Minimization of** z

$$z_{t+1} = \underset{z}{\text{argmin}} \quad \lambda\|z\|_1 + \frac{\rho}{2}\|\bar{x}_{t+1} - z + \frac{1}{\rho}\bar{\nu}_t\|_2^2$$

$$z_{t+1} = \bar{x}_{t+1} + \frac{\bar{\nu}_t}{\rho} - \frac{\lambda}{\rho}\text{sign(z)}$$

3. **Update of dual variable** ν_t

$$\nu_{t+1}^{(i)} = \quad \nu_t^{(i)} + \rho(x_{t+1}^{(i)} - z_{t+1})$$

5 Ridge Regression

The Ridge Regression objective is similar to the LASSO objective, but in this case we use the L-2 norm squared instead of L-1 norm as regularization term with λ. It is as follows:

$$\min_x \frac{1}{2}\|Ax - b\|_2^2 + \lambda\|x\|_2^2$$

where $x \in \mathbb{R}^d$ is a vector of weights of the model that need to tuned and $A \in \mathbb{R}^{n\text{x}d}$ is a matrix of all the input features and $b \in \mathbb{R}^n$ is the vector of all outputs.

Similar to LASSO we implemented this algorithm in the following two ways:

5.1 Vectorized Serial Implementation

We can split the objective function by renaming one of the two variables and introduce an equality constraint to bring in the ADMM form. The modified Ridge objective is:

$$\begin{aligned} \min_{x,z} \quad & \frac{1}{2}\|Ax-b\|_2^2 + \lambda\|z\|_2^2 \\ \text{st} \quad & x = z \end{aligned}$$

The update rule for each of the two variables is as follows:

1. **Minimization of x**

$$x_{t+1} = \underset{x}{\text{argmin}} \quad \frac{1}{2}\|Ax-b\|_2^2 + \frac{\rho}{2}\|x - z_t + \frac{1}{\rho}\nu_t\|_2^2$$

which has a closed form solution

$$x_{t+1} = (A^TA + \rho I)^{-1}(A^Tb + \rho z_t - \nu_t)$$

2. **Minimization of z**

$$z_{t+1} = \underset{z}{\text{argmin}} \quad \lambda\|z\|_2^2 + \frac{\rho}{2}\|x_{t+1} - z + \frac{1}{\rho}\nu_t\|_2^2$$

$$z_{t+1} = \frac{\rho x_{t+1} + \nu_t}{2\lambda + \rho}$$

3. **Update of dual variable ν_t**

$$\nu_{t+1} = \quad \nu_t + \rho(x_{t+1} - z_{t+1})$$

5.2 Parallel Implementation

We can rewrite the objective function of Ridge for running in distributed/parallel fashion by splitting across all the samples:

$$\begin{aligned} \min_{x^{(i)},z} \quad & \frac{1}{2}\sum_{i=1}^{N}\|A^{(i)}x^{(i)} - b^{(i)}\|_2^2 + \lambda\|z\|_2^2 \\ \text{st} \quad & x^{(i)} - z = 0, \ for\ i = 1, ..., N \end{aligned}$$

The modified update rule for each of the two variables is as follows:

1. **Minimization of x_i**

$$x_{t+1}^{(i)} = \underset{x^{(i)}}{\text{argmin}} \quad \frac{1}{2}\|A^{(i)}x^{(i)} - b^{(i)}\|_2^2 + \frac{\rho}{2}\|x^{(i)} - z_t + \frac{1}{\rho}\nu_t^{(i)}\|_2^2$$

which has a closed form solution

$$x_{t+1}^{(i)} = (A^{(i)T}A^{(i)} + \rho I)^{-1}(A^{(i)T}b^{(i)} + \rho z_t - \nu_t^{(i)}), \ for\ i = 1, ..., N$$

2. **Minimization of z**

$$z_{t+1} = \underset{z}{\text{argmin}} \quad \lambda\|z\|_2^2 + \frac{\rho}{2}\|\bar{x}_{t+1} - z + \frac{1}{\rho}\bar{\nu}_t\|_2^2$$

$$z_{t+1} = \frac{\rho\bar{x}_{t+1} + \bar{\nu}_t}{2\lambda + \rho}$$

3. **Update of dual variable ν_t**

$$\nu_{t+1}^{(i)} = \quad \nu_t^{(i)} + \rho(x_{t+1}^{(i)} - z_{t+1})$$

6 Results

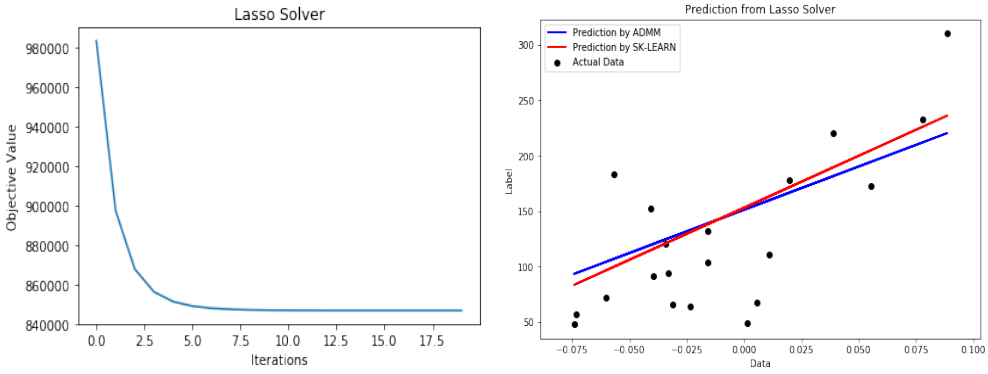
We tested our parallel implementation of ADMM for LASSO and Ridge against the sklearn's implementation on the Diabetes dataset [4].

6.1 Diabetes Dataset

- The diabetes dataset has 10 features:
  age, sex, body mass index, average blood pressure, and six blood serum measurements (S1-S6)
- The output is a quantitative measure of disease progression one year after the measurements of the 10 features
- Number of samples: 442

6.2 LASSO Regression

We ran our ADMM Lasso solver on above Diabetes dataset: Figure 1(a) shows objective values of ADMM Lasso solver.



(a) Lasso Objective Function

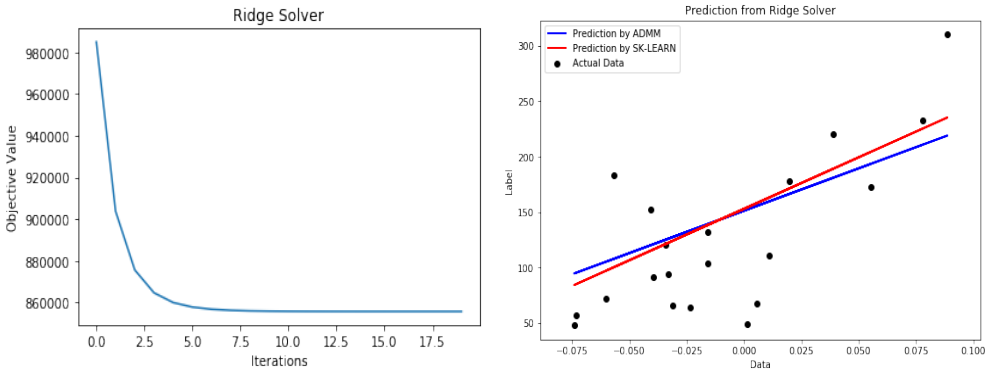
(b) Comparison with Scikit-Learn's Prediction

Figure 1: Lasso Solver

Figure 1(b) shows prediction comparison with Sklearn's lasso solver.

6.3 Ridge Regression

We ran our ADMM Ridge solver on above Diabetes dataset: Figure 2(a) shows objective values of ADMM Ridge solver and figure 2(b) shows prediction comparison with Sklearn's Ridge solver.



(a) Ridge Regression Objective Function (b) Comparison with Scikit-Learn's Prediction

Figure 2: Ridge Solver

The following plot shows a comparison of how our ADMM performs against gradient descent and newton algorithms for a Ridge objective function.

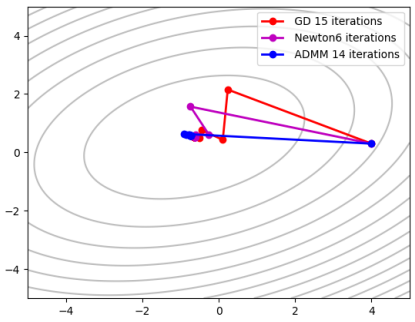


Figure 3: Gradient Descent, Newton and ADMM on Ridge Objective function

6.4 Execution Results

Following is run-time comparison of our vectorized serial implementation with Sklearn's state of the art implementation.

Where, Speed up = Sklearns time / ADMM time

Run	Sklearn's Lasso (ms)	ADMM Lasso (ms)	Lasso Speedup	Sklearn's Ridge (ms)	ADMM Ridge (ms)	Ridge Speedup
1	0.00059104	0.00041389	1.42	0.000633	0.00037718	1.67
2	0.00033402	0.0002811	1.18	0.00053191	0.00030708	1.73
3	0.00030208	0.00026703	1.13	0.0004971	0.00030208	1.64
4	0.00029016	0.00026298	1.10	0.00048494	0.00030184	1.60
5	0.00028491	0.00030899	0.92	0.00048399	0.00031805	1.52

Average speed up for 10000 runs:
1.09X for Lasso regression solver and 1.59X for Ridge regression solver

7 Conclusion

We implemented the above mentioned two algorithms in the PyTorch framework. There are many such algorithms that can be optimized with ADMM. Our implementation gives similar results to the state of the art solvers from scikit learn. As expected, the ADMM implementation performs faster than the scikit learn implementation as seen from the results.

Future Work

We are working on solving the basis pursuit and SVM hinge loss problem. Future work involves making a generic classifier that solves any objective function in the ADMM form.

Source Code

Our Implementation can be found here:
https://github.com/bhushan23/pytorch/blob/master/torch/optim/admm.py
Test files can be found here:
https://github.com/bhushan23/ADMM/blob/master/Diabetes_Test.ipynb

References

[1] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein (2011) Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. *Foundations and Trends in Machine Learning* **3**(1):1–122.
[2] Stanford University Convex Optimization Group, cvxflow,
`https://github.com/cvxgrp/cvxflow/blob/master/cvxflow/solvers/admm.py`, 2017
[3] Niru Maheswaranathan, ADMM,
`https://github.com/nirum/ADMM/blob/master/admm.py`, 2014
[4] Brad Efron, Trevor Hastie. Stanford University.
`https://web.stanford.edu/ hastie/Papers/LARS/diabetes.data`