MSBD6000B: Deep Learning

Report for Project

Group 8: Implement of the optimization method for ADMM approach.

|  |  |  |
| --- | --- | --- |
| Student Name | Student ID | Email |
| Tsz Him NG | 20383422 | thngak@connect.ust.hk |
| Kwok Yui LO | 20411920 | kyloam@connect.ust.hk |
| Tsz Kin HO | 20392904 | tkhoac@connect.ust.hk |
| Kin Ho CHAN | 20411918 | khchance@connect.ust.hk |
| Man Leung YEUNG | 00055768 | mlyeungab@connect.ust.hk |

# Background and Purpose

We are working on the Option 1 of the project – Implement of the optimization method for ADMM approach.

There are different methodologies to figure out local maxima in the dataset. However, most of them would not be able to figure out the global maxima easily. Applying the ADMM approach would allow us to find the optimize solution and global maxima easily. Moreover, the loss function does not need to be differentiable. The implementation process is standard with a standard formula, which it is simple to implement.

We have used model with convolutional layers with high dimension, pooling and RELU activation function to compare the accuracy between ADMM and SGD for finding the global maxima.

# Design and Implementation

## Data

We have used the dataset through website. It called CIFAR-10. It contains 10 class. The dimension for dataset is 32X32 and there are 60,000 samples inside the dataset. It is our standard dataset for using to train our model.

## Model

Our model included the convolutional layers with high dimension and we have 3 convolutional layers in the model. All activation function for the convolutional layers are using RELU and run with 250 epochs.

The size of input is 32X32X3 and the first filter size is 5X5X3 and the number of filer is 192. All paddings of the convolution and pooling layer are the same. In the first convolutional layer, there are two sparse 160, 96 filters and the sizes are 1X1. The first max pooling layer is 3X3 with stride 2x2.

The second filter is 5X5x96 and number of filters is 192. In the second convolutional layer, there are two sparse 192,192 filters and they are same as the first one, which is 1X1. The second average pooling size is 3X3 with stride 2x2.

For the third 192 filter size, it is 3X3x192. Both sparse filters 192,10 in the third convolutional layer are 1X1. The size of third average pooling is 8X8 with no padding. At last, the size of output will come out with 10X1.

# Result and Conclusion

When using the model with ADMM approach for training the first 30 epochs with the CFAR-10 dataset, the accuracy is around 30%. In addition, the accuracy increases to around 60% after completed all epochs (i.e. 250 epochs). However, the running speed is slow since the model needs 24 hours to complete all epochs running.

On the other hand, as shown in the **accuracy\_bp.png**, the accuracy is 45% after running SGD (**standard\_bp.py**) for 250 epochs in a model with similar architecture as in ADMM model.

In conclusion, comparing the ADMM and SGD, both of them needs to run with long time to complete all epochs. In addition, SGD needs more epochs than ADMM to reach the same accuracy. However, ADMM depends on tuning sparsity-promoting parameter to achieve high accuracy.

# Reference Training Neural Network Without Gradients: A Scalable ADMM approach

# https://arxiv.org/pdf/1605.02026.pdf

Convex Optimization: What's the advantage of alternating direction method of multipliers (ADMM), and what's the use case for this type of method compared against classic gradient descent or conjugate gradient descent method?

<https://www.quora.com/Convex-Optimization-Whats-the-advantage-of-alternating-direction-method-of-multipliers-ADMM-and-whats-the-use-case-for-this-type-of-method-compared-against-classic-gradient-descent-or-conjugate-gradient-descent-method>

# Appendix ADMM algorithm:

**cifar\_models.py**: A module that constructs the graph of investigated models in tensorflow

**ADMM\_cifar.py**: Python implementation of ADMM-based sparse CNN

**ADMMutils.py**: Utility functions for use with the ADMM algorithm. There are functions to load Cifar10 data and to apply norm-0 sparsity promoting penalty functions

Pre-trained folder: Pre-trained weights in ckpt format (pre-trained folder)

CIFAR-10 will be auto-download in .data/cifar-10-binary/ folder if not existed

**SGD algorithm (as brand mark for reference):**

**read\_Cifar10\_bp.py**: Create hdf5 file to store the train, validation and test data for running CNN in SGD algorithm. (**cifar-10-python.tar.gz** was downloaded and extract to a folder **“\cifar-10-batches-py\**” in the working directory.)

**standard\_bp.py**: Using the data generated from the output of **read\_Cifar10\_bp.py** (inside **“\cifar-10-batches-py\**”) to run CNN in SGD algorithm to produce brand mark result. (create a folder **“\model\**” in the working directory before running **standard\_bp.py**)

**accuracy\_bp.png**: A graph of accuracy against no. of epoch when running CNN in SGD