Deep Learning Programming Assignment 2

In this project, you will first build a *convolutional neural network* from scratch and then examine its performance on some standard dataset. In the second part, we will train CNN on open source deep learning libraries.

The goal is:

- Understand CNN's structure and its components.
- Implement convolutional, pooling, Relu and linear layers.
- Implement forward and backward propagation.
- Implement state of the art optimization algorithms on neural nets.
- Learn to model on deep learning libraries including TensorFlow, Pytorch, etc.
- Run code on modern GPU.

Build your own CNN (50 points in total)

We will solve image classification problem on **CIFAR 10** dataset. The CIFAR is a labeled subset of the 80 million tiny images dataset. CIFAR 10 contains 50000 training images in 10 classes.

We provide in the code a simplified deep learning architecture. Although it doesn't offer symbolic programming or auto-differentiation, this is still a good starting point to get familiar with deep learning framework. Based on the code, you will develop your own convolutional neural network.

Your network should have the following structures:

> conv + relu + pooling + linear + relu + linear + softmax

Most of the parameters have been set in the constructor

- `nn` folder contains the source code to run CNN. Follow the instructions in the comment and fill in the place with `#todo` sign.
    1. `cnn.py`: Class definition of CNN, you need to write the `oracle` function to compute loss, score and gradient function. In this part, you should add the details on forward/backward propagation on graph.
    2. `layers.py` : Write `__init__`, `forward` and `backward` function of each layer.
    3. `optimizer.py`: We provide the base class of optimizer from which all the concrete optimizers need to inherit, and a simple SGD. Improve the code and add momentum methods.
- Under the root folder, we provide some basic to read data and test the performance of your code. Write a script to use the gradient checker to make sure that the gradient of each component is computed correctly. **(40 pts)**
- Run your algorithm to fit a subset of training data (100 samples). What is the performance of your algorithms, do they overfit on the small data? (**10 pts**)

Acceleration （15 points in total）

The code you implemented can be very slow on large dataset, now we want to find out how to improve the efficiency of our deep nets. Among the components, linear layers are highly vectorized and allow speed up by Intel architectures, due to the internal C implementation. Convolutional layers and pooling layers applied more sophisticated stencil operators.

Q: Perform profiling on each layer, compare and plot the time cost of each layer, which layer(s) is (are) relatively more expensive and seem(s) to be the bottleneck(s) in your implementation? (**5 pts**)

To accelerate the computation, one way is to implement forward and backward operations in `C/C++.` Rebuilt the code in C++ is time consuming, but luckily, we can put the most intensive computation in `Numba` with the minimum overheads.

`Numba` gives you the power to speed up your applications with high performance functions written directly in Python (their own words :)). Try to rewrite the `Layer.py` using `jit` decorators on your forward and backward functions of those layers that are deemed slow. Read `numba`'s manual carefully to make sure you will squeeze the most from it. The numba subroutine should contain no Python type so that `Numba` will accelerate

Q: Rewrite the least efficient layer in `Numba`. How much time do you expect to save after acceleration? (**10 pts**)

CNN on standard deep learning architecture (35 points in total)

For even larger deep nets and datasets, our code can be inefficient and we need more high performance computing for acceleration. Since most of the computation in deep nets are matrix algebra, we will gain significant acceleration when it is running GPU. To this end, we build deep nets on open source deep learning libraries which support more parallel and distributed computing.

Q: Choose your favorite deep learning library (one of TF, PyTorch, MXnet, etc) and rebuild the same cnn model you wrote, run your code on GPU and how does it compare to the efficiency of your implementation? Can you find some way to improve the generalization performance by changing the network? You are allowed to use at most **THREE** linear or conv layers. (**20 pts**)

Q In addition to the basic SGD, we have learned many other recent algorithms like momentum method, Adagrad and ADAM. Choose at least two algorithms and compare their efficiency. Plot the convergence of loss function and training/validation/testing error. Discuss your observation. (**15 pts**)

Remark on working remote on cluster

As part of this course, we will run the course project on the cluster. The cluster is equipped with GTX-1080Ti and has installed popular packages like TensorFlow, PyTorch and MXNet. The cluster is still in maintenance now, but will be available very soon.

Each one will be assigned a user account that remains active over the whole course. The account will expire at the end of the semester, so be prepared to save your data in time.

All the computing tasks should be running through the scheduling software IBM **LSF**. You are not **ALLOWED** to run the code on the computing node **directly**. For example, you are not allowed to enter `python example.py` for running your project in each node. Instead, you will submit your job by writing a script such that LSF will distribute your job automatically.

We will provide a tutorial and also spread out manuals so that you will learn how to run a job on cluster.

The reason for doing so is that the computing resource in the cluster is extremely limited, we need to make sure that resources are distributed equally. Therefore instructions should be strictly followed.