# CS744 Assignment 2

## Due: Oct 11, 2019

# Overview

In this assignment, you will learn how to write TensorFlow (https://www.tensorflow.org/) applications. You will start with setting up the cluster and running workloads on a single machine. The next task is to modify the workloads so that they can be launched in a distributed way. You will experiment with both Synchronous and Asynchronous SGD, as well as with different APIs like Tensorflow Core and Keras. Finally, you will need to look deep into CPU/MEM/Network usage to have a better understanding of the performance.

# Learning Outcomes

After completing this programming assignment, you should be able to:

- Write simple Tensorflow applications using Tensorflow Core and launch them in the cluster.
- Write Tensorflow applications using Keras API, in both single and distributed modes.
- Have a deeper understanding about Tensorflow's performance.

# Environment Setup

In this assignment, we provide you a CloudLab profile called "shivaram-cs744-fa19-assign1" (profile used for assignment 1) under "UWMadison744-F19" project for you to start your experiment. The profile is a simple 3-node cluster with Ubuntu 16 installed on each machine.

Similar to assignment1, you should firstly make sure that the VMs can ssh each other. To install Tensorflow, you need to run following commands on each VM:

```
sudo apt-get update
sudo apt-get install --assume-yes python3-dev python3-pip
sudo pip3 install tensorflow tensorflow-datasets absl-py
```

# Part 1: Logistic Regression (https://en.wikipedia.org/wiki/Logistic_regression)

In this part, you will need to implement a simple Logistic Regression (https://en.wikipedia.org/wiki/Logistic_regression) application. Building a Tensorflow application mainly consists of two sections: building the computational graph, known as tf.Graph (https://www.tensorflow.org/api_docs/python/tf/Graph) and running it using tf.session (https://www.tensorflow.org/api_docs/python/tf/Session). A graph is a series of Tensorflow operations. The graph is submitted to cluster through session.

Your application should start from a random linear vector `w` as your model. Then, calculate the loss of your model on the dataset using:

$$\mathcal{L}(D_{tr}) = \sum_{y,x \in D_{tr}} -y \log softmax(w^T x)$$

`x` and `y` are training data features and label respectively. Using Tensorflow API, the loss can be computed through:

```
prediction  = tf.nn.softmax(tf.matmul(x, W) + b)
loss = tf.reduce_mean(-tf.reduce_sum(y*tf.log(prediction), reduction_indices=1))
```

Tensorflow provides a set of standard optimization algorithms, which are implemented as sub-classes of tf.train.Optimizer (https://www.tensorflow.org/api_docs/python/tf/train/Optimizer). You can choose one, for example, `tf.train.GradientDescentOptimizer` to minimize the loss.

The dataset for you to train your model is MNIST (https://en.wikipedia.org/wiki/MNIST_database) handwritten digits database. Tensorflow provides convenient API for you to load input data:

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

We provide you a set of scripts and templates that will help you to run your application, including run_code_template.sh (assets/run_code_template.sh), cluster_utils.sh (assets/cluster_utils.sh) and code_template.py (assets/code_template.txt). You need to first modify the cluster specification part of `code_template.py` according to your cluster's setting and then put your code at the bottom of this file. After that, you can execute `run_code_template.sh` to run your application.

## Synchronous and Asynchronous SGD

In distributed mode, dataset is usually spread among the VMs. On each iteration, the gradients are caculated on each worker machine using its shard of data. In synchronous mode, the gradients will be accumulated to update the model and then go to next iteration. However, in asynchronous mode, there is no accumulation process and the worker nodes update the model independently.

After finishing the implementation, you should also monitor the CPU/Memory/Network usage of each VM during training. You can try to use tools like: dstat (http://dag.wiee.rs/home-made/dstat/) or sar (https://linux.die.net/man/1/sar). You are welcome to use any other tool you like to monitor the system.

You can use TensorBoard (https://www.tensorflow.org/guide/summaries_and_tensorboard) to visualize the graphs you created in the LR trainning process you just ran. TensorBoard is a suite of visualization tools that can visualize your graph or plot quantitative metrics to help you understand, debug and optimize TensorFlow programs. See sampleTensorboard.sh (assets/sampleTensorboard.sh) and exampleTensorboard.py (assets/exampleTensorboard.txt) as an example. Screenshot the graphs and include them in your report.

**Task 1.** Implement the LR application and train the model using single node mode. We know that Keras API is very easy to use. However, to help you better understand how things work in Tensorflow, we require you NOT to use it and stick to original API (https://www.tensorflow.org/guide/low_level_intro) for this part of the assignment.

**Task 2.** Implement the LR application in distributed mode using Sync SGD and Async SGD. Plot the performance and test error for both of them and explain any similarity / differences. Monitor the CPU/Memory/Network usage. Show your observations and determine which one is the bottleneck.

**Task 3.** (Optional) Try different batch size and discuss the differences.

# Part 2: LeNet (http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf)

In this part, you will implement LeNet. LeNet was one of the first convolutional neural networks (https://en.wikipedia.org/wiki/Convolutional_neural_network). It has a very simple architecture. It consists of 7 layers, among which there are 3 convolutional layers (C1, C3 and C5), 2 sub-sampling (pooling) layers (S2 and S4), and 1 fully connected layer (F6), that are followed by the output layer. You can find more detailed description of the network in the original paper (http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf).

You will be using same MNIST handwritten digits dataset for training.

**Task 1.** Implement LeNet architecture using Keras API (https://www.tensorflow.org/guide/keras). Run it using one, two and three machines. Monitor the CPU/Memory/Network usage as we go from 1 to 3 machines.

Hints for completing this task:

- You need to use MultiWorkerMirroredStrategy API, which is one of the strategies for distributed training in Tensorflow. Read more about strategies here (https://www.tensorflow.org/guide/distribute_strategy).
- You need to set TF_CONFIG environment variable (https://www.tensorflow.org/guide/distribute_strategy#setting_up_tf_config_environment_variable) on each node in a cluster, that let's the Tensorflow know how many workers are in the cluster and which node refers to which worker (through task index). Example of how TF_CONFIG might look for worker 0 in the cluster:

  ```
  TF_CONFIG='{"cluster": {"worker": ["node0:2223", "node1:2222", "node2:2222"]},
             "task": {"index": 0, "type": "worker"}}'
  ```

  You can specify TF_CONFIG inside your code and pass the task index as an argument to the program.

- Make use of `repeat` when creating the dataset to ensure workers do not run out of examples if we run more epochs.
- Explicitly pass in a number of `steps_per_epoch` to the `fit` function.
- You can write a script which will automate the process of running jobs on each worker by looking at the script given in part 1 called cluster_utils.sh.

**Task 2.** Try different batch size and see the difference.

**Task 3.** (Optional) Run LeNet on one GPU machine in CloudLab and compare that to above experiments. To use GPU-based machines you will need to use an appropriate profile, hardware type in CloudLab and as the GPU nodes are not always available this task is optional.

# Deliverables

You should submit a tar.gz file to Canvas (you will be put into your groups on Canvas so only 1 person should need to submit it), which consists of a brief report (filename: groupx.pdf) and the code of each task. Put the code of each part and each task into separate folders give them meaningful names. Code should be commented well (that will be worth some percentage of your grade for the assignment, grader will be looking at your code). Also put a README file for each task and provide the instructions about how to run your code. Also include a `run.sh` script for each part of the assignment that can re-execute your code on a similar CloudLab cluster.

# Acknowledgements

This assignment uses insights from Professor Aditya Akella's assignment 3 of CS744 Fall 2017 fall and Professor Mosharaf Chowdhury's assignment 2 of ECE598 Fall 2017.