Except where otherwise indicated, this report is my own original work.

Chao Liu
12 June 2020
Acknowledgments

I would like to thank my supervisor Assoc Professor Peter Strazdins for providing me this excellent research opportunity. His patient guidance throughout the project helped me develop my research skills, which I believe is very valuable for my future career. This project cannot be completed without the instructions and assistance from Assoc Professor Peter Strazdins.

I also would like to thank the course convener Prof Weifa Liang for arranging the meeting and tutorials. He provided many suggestions for delivering presentations and writing the report.
Deep learning is a very popular machine learning method currently, and it can be used to solve various tasks. There are now many open source deep learning tools that can build various deep learning network models, such as convolutional neural network (CNN) and recurrent neural network (RNN). However, deep learning workloads are becoming increasingly more compute-intensive, so training deep learning networks is usually a very time-consuming process. Almost all deep learning frameworks support the use of GPU in order to speed up the calculation of deep learning models. This project will involve benchmark to analyze the performance of CPU and GPU in training deep learning workloads. ParaDnn is a micro-benchmark for deep learning, which can compare the running time of the workloads between various devices. In addition, a simple deep learning model is also built to compare the running time between one GPU and multiple GPUs. The results of this report demonstrate that running workloads on GPU is faster than CPU, and multiple GPUs are faster than one GPU.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgments</td>
<td>iii</td>
</tr>
<tr>
<td>Abstract</td>
<td>v</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivations</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Objective</td>
<td>1</td>
</tr>
<tr>
<td>1.3 Project Scope</td>
<td>1</td>
</tr>
<tr>
<td>1.4 Contribution</td>
<td>2</td>
</tr>
<tr>
<td>1.5 Report Outline</td>
<td>2</td>
</tr>
<tr>
<td>2 Background and Related Work</td>
<td>3</td>
</tr>
<tr>
<td>2.1 Architecture of GPU</td>
<td>3</td>
</tr>
<tr>
<td>2.2 GPU Computing Capability</td>
<td>3</td>
</tr>
<tr>
<td>2.3 Deep Learning Framework</td>
<td>4</td>
</tr>
<tr>
<td>2.3.1 TensorFlow</td>
<td>4</td>
</tr>
<tr>
<td>2.3.2 Keras</td>
<td>7</td>
</tr>
<tr>
<td>2.4 Benchmark</td>
<td>7</td>
</tr>
<tr>
<td>3 Methodology</td>
<td>9</td>
</tr>
<tr>
<td>3.1 ParaDnn</td>
<td>9</td>
</tr>
<tr>
<td>3.1.1 Construction of fully connected models</td>
<td>9</td>
</tr>
<tr>
<td>3.1.2 Construction of Recurrent Neural Network</td>
<td>10</td>
</tr>
<tr>
<td>3.1.3 Run on CPU</td>
<td>11</td>
</tr>
<tr>
<td>3.1.4 Run on GPU</td>
<td>12</td>
</tr>
<tr>
<td>3.2 CNN and Keras</td>
<td>12</td>
</tr>
<tr>
<td>3.2.1 Construction of CNN Models</td>
<td>12</td>
</tr>
<tr>
<td>3.2.2 Run on CPU and GPU</td>
<td>13</td>
</tr>
<tr>
<td>4 Results and Evaluation</td>
<td>15</td>
</tr>
<tr>
<td>4.1 Results of ParaDnn</td>
<td>15</td>
</tr>
<tr>
<td>4.2 Results of Keras</td>
<td>18</td>
</tr>
<tr>
<td>5 Conclusion and Future Work</td>
<td>23</td>
</tr>
<tr>
<td>Bibliography</td>
<td>25</td>
</tr>
</tbody>
</table>
List of Figures

2.1 Block diagram of a GPU card ........................................... 4
2.2 Inside an SM .................................................................. 5
2.3 The performance of GPUs has increased dramatically compared to CPUs 6
2.4 The performance of GPUs has increased dramatically compared to CPUs 6

3.1 The structure of fully connected neural network ......................... 10
3.2 The structure of recurrent neural network ................................. 11
3.3 A CNN for character recognition. Each rectangle represents a feature map, and the last two layers are fully connected layers. ............... 12

4.1 FC models with large nodes are better suited for GPU than CPU .... 16
4.2 Running on GPU is faster than CPU for RNN ......................... 17
4.3 Running time of each epoch for CNN on whale server ................. 18
4.4 The average running time .................................................. 19
4.5 Running time of each epoch for CNN on the laptop ................... 20
Introduction  

This chapter includes the motivations, objective, project scope, contribution and outline for this report.

1.1 Motivations

In the past decade, deep learning has been successfully applied in various application fields such as computer vision, image classification, speech recognition and natural language processing, etc [1]. However, as the amount of data increases, a large amount of calculations required for training a deep learning model, which often consumes many days or even months. Therefore, many methods are applied to optimize their computing performance.

With the rapid development of GPU from a configurable graphics processor to a programmable parallel processor, programs are increasingly using the massive parallel computing capabilities of GPUs to achieve superior performance and efficiency [2]. Today, GPU computing makes it possible for applications that we previously thought were impossible to achieve due to the long execution time [2]. We hope to build deep learning models and run them on the CPU and GPU to obtain running time, and then analyze the performance of the deep learning workloads between various devices by using a benchmark for deep learning.

1.2 Objective

The objective of this project is to choose an appropriate benchmark to analyze the performance between various devices in training deep learning workloads. The running time of deep learning models in different devices can be used to compare the computing performance of CPU and GPU.

1.3 Project Scope

In this report, we focus on analyzing the running time performance of FC, RNN and CNN models. This project could be divided into two tasks:
1. Construction of ParaDnn benchmark – ParaDnn is a tool that can generate parameterized end-to-end models to run on target platforms [3]. Fully connected networks (FC) and recurrent neural network were generated by ParaDnn [3]. TensorFlow framework was used in this task to build these deep learning models.

2. Construction of convolutional neural network based on Keras – A convolutional neural network was built in this task based on Keras, and the dataset for training the model is MNIST (Mixed National Institute of Standards and Technology database), which is comprised of a training set of 60,000 images and a test set of 10,000 images.

1.4 Contribution

The main contribution of this project is the study of analyzing performance of CPU and GPU in training deep learning models. In this report, various deep learning models was developed, and they were ran on CPU and GPU respectively. This project could provide a simple model of deep learning to compare the computing performance of CPU and GPU.

1.5 Report Outline

Chapter 1 provides the introduction of this project. Chapter 2 introduces some backgrounds that are related to our work. In Chapter 3, we describe the architecture of deep learning models and explain basic processes for constructing these models. Chapter 4 analyzes the results and discusses reasons that related to the results. Chapter 5 gives conclusion and what will we do in future works.
Chapter 2

Background and Related Work

This chapter introduces the background of the project. Section 2.1 and Section 2.2 give background of GPU. Section 2.3 describes deep learning frameworks that we used in this project. Section 2.4 introduces some common benchmarks for deep learning.

2.1 Architecture of GPU

Nvidia is a well-known company that can design and manufacture GPUs. In 2007, Nvidia released a programming methodology, which consists of a programming model named Compute Unified Device Architecture (CUDA) and a compiler that supports the C language with GPU specific extensions for local, shared and global memory, texture memory, and multithreaded programming [4].

As shown in Figure 2.1 [5] and Figure 2.2 [5], a GPU is an array of streaming multiprocessors (SMs), and each has a number of streaming processors (SPs). One GPU device contains multiple SMs. GPU can handle more tasks at the same time if there are more SMs in the GPU. These SMs are first connected to a shared memory (L1 cache), and then they are connected to an L2 cache. Each SM needs to access a register file, which is a set of storage units that can work at the same speed as SP, so accessing this set of storage units requires almost no waiting time [5].

In addition, there is a shared memory that is only accessible internally by each SM, which can be used as a cache for program management [5]. For texture memory, constant memory and global memory, each SM has a bus that can access them independently [5]. Texture memory is a special view of global memory that is useful for data that has interpolation, such as using 2D or 3D lookup tables [5]. It has the ability to interpolate based on hardware. Constant memory is used to store read-only data, which is cached by all GPU cards [5]. Like texture memory, constant memory is also a view of global memory [5].

2.2 GPU Computing Capability

As shown in Figure 2.3 [6], GPUs have higher computational power than CPUs, and the divergence of CPU and GPU computational power is becoming larger and
Background and Related Work

Figure 2.1: Block diagram of a GPU card

larger. The main reason of the divergence between CPU and GPU is fundamental architectural differences [6]. Data parallel in graphics computing allows GPUs to use additional transistors for calculations more directly, achieving higher arithmetic intensity with the same number of transistor count [6]. Thus, GPUs are widely used in accelerating computing for complicated projects. There are many deep learning frameworks that can run on both CPU and GPU, such as TensorFlow, PyTorch and Keras. In this project, GPU was used to accelerate deep learning workloads based on TensorFlow and Keras.

2.3 Deep Learning Framework

2.3.1 TensorFlow

TensorFlow is a popular machine learning system that is widely used for building various neural network models and deep learning models. TensorFlow integrates the most common units in deep learning frameworks and supports many latest networks with different settings, such as CNN and RNN [1]. TensorFlow supports both large-scale training and inference and it can effectively use many powerful servers such as GPU for rapid training [7]. Figure 2.4 shows the structure of TensorFlow. TensorFlow source code contains a file named Stream Executor [8], which can call CUDA platform to use GPU.
Figure 2.2: Inside an SM
Background and Related Work

**Figure 2.3:** The performance of GPUs has increased dramatically compared to CPUs

**Figure 2.4:** The performance of GPUs has increased dramatically compared to CPUs
2.3.2 Keras

Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow [9]. It was developed with a focus on enabling fast experimentation [9].

2.4 Benchmark

In order to evaluate the acceleration effect of GPU on deep learning projects, the benchmark is required. There are many deep learning benchmarks, such as DAWN-Bench and MLPerf. DAWN-Bench is a benchmark that measures end-to-end training time to achieve a state-of-the-art accuracy level, as well as inference time with that accuracy [10]. MLPerf is a benchmark contains sub-items in different fields, including image classification, object recognition, translation, recommendation, speech recognition, sentiment analysis, and reinforcement learning [11].
Methodology

This chapter describes the main method of the project and can be divided into two sections. In the first section, fully connected and recurrent neural networks models are constructed by ParaDnn. We run these two models on CPU and GPU of whale server respectively. The second section describes the construction of the convolutional neural network models based on Keras. We run this model on CPU, single GPU and two GPUs of whale server respectively, and also run the model on the laptop.

3.1 ParaDnn

ParaDnn can create fully connected and recurrent neural network models. These models are parameterizable, so ParaDnn models are equal to or greater in size compared to today’s real-world models [3].

3.1.1 Construction of fully connected models

FC models are comprised of multiple fully-connected layers. The architecture of FC is

\[
\text{Input} \rightarrow \text{[Layer[Node]]} \rightarrow \text{Output},
\]

where [Layer] means the number of layers is variable. We can change the number of layers, the number of nodes per layer, and the numbers input and output units of the datasets.

The neural network can be regarded as a black box that can fit any function. As long as the training data is sufficient, given a specific x, we can get the desired y. Fully connected means that each neuron in the \(Nth\) layer is connected to each neuron in the \(N-1th\) layer, and each connection has a weight. As shown in the figure 3.1, there are 2 nodes in the input layer numbered 1 and 2. The hidden layer also has two nodes numbered 3 and 4. The output layer has two nodes numbered 5 and 6, \(b_1\) and \(b_2\) are bias nodes. \(w_{ji}\) represents the weight between the \(j-th\) node (unbiased node located at the \(Nth\) layer) and the \(i-th\) node (unbiased node located at the \(N-1th\) layer), where \(j\) is the target node and \(i\) is the source node. \(w_{jb}\) represents the weight...
between the $j$-th node (the unbiased node at the $N$th layer) and the biased node at the upper layer. $a_j$ represents the output value of the $j$-th node.

Taking node 3 as an example, the input value of node 3 is

$$w_{31}x_1 + w_{32}x_2 + w_{3b},$$

and the output value of node 3 is

$$a_3 = \sigma(w_{31}x_1 + w_{32}x_2 + w_{3b}),$$

where $\sigma$ is the activation function.

### 3.1.2 Construction of Recurrent Neural Network

The fully connected neural network can only take one input after another, and the previous input is completely unrelated to the next input. However, some tasks need to be able to better process the sequence information, that is, the previous input and the subsequent input are related, such as continuous speech and continuous hand written text. RNN is good at this kind of problem.

Figure 3.2 shows the structure of RNN. On the left is the RNN model that is not expanded by time series. Here, $x^{(t)}$ represents the input of the training sample at the sequence index number $t$. $h^{(t)}$ represents the hidden state of the model at the sequence index number $t$, which is determined jointly by $x^{(t)}$ and $h^{(t-1)}$. $o^{(t)}$ represents the output of the model at the sequence index number $t$ and is only determined by the current hidden state $h^{(t)}$ of the model. $L^{(t)}$ represents the loss function of the model at the sequence index number $t$. The three matrices $U$, $V$ and $W$ are the linear relationship parameters of the model, and they are shared throughout the RNN.

For any sequence index number $t$, the hidden state $h^{(t)}$ is obtained from $x^{(t)}$ and
Figure 3.2: The structure of recurrent neural network

$h^{(t-1)}$, that is:

$$h^{(t)} = \sigma(z^{(t)}) = \sigma(Ux^{(t)} + Wh^{(t-1)} + b),$$

where $\sigma$ is the activation function of the RNN, and $b$ is the bias. When the serial index number is $t$, the expression of the model $o^{(t)}$ is:

$$o^{(t)} = Vh^{(t)} + c$$

The prediction output for the final sequence index number $t$ is:

$$\hat{y}^{(t)} = \sigma(o^{(t)})$$

The loss function $L^{(t)}$ such as the log-likelihood loss function can quantify the loss of the model at the current position, that is, the difference between $\hat{y}^{(t)}$ and $y^{(t)}$.

In this report, each token of the input sequence is embedded within a fixed length vector, and the length of the vector is the embedding size. We can change the number of layers and the embedding size. We can also change the batch size, which is the number of samples selected in one training. The variables in the dataset include the maximum length per input sequence and vocabulary size.

### 3.1.3 Run on CPU

The CPU is Intel(R) Xeon(R) Gold 6134 from the ANU whale server, which has 8 cores and 16 threads. It operates at 3.2 GHz with a TDP of 130 W and a turbo boost
3.1.4 Run on GPU

The GPU is an NVIDIA Tesla P100-PCIE GPU platform that contains 4 Nvidia NVLink ports of 40 GB per second. One node has 16 GB of memory and 732 GB/s memory bandwidth [13]. The parameters of models running on the GPU are same as running on the CPU.

3.2 CNN and Keras

Convolutional neural network is a hierarchical model whose input is raw data, such as images. The convolutional neural network extracts high-level semantic information from the original data through a series of operations such as convolution operation, pooling operation and nonlinear activation function mapping operation. Different types of operations are generally called layers in convolutional neural networks: convolution operations correspond to convolutional layers, and pooling operations correspond to pooling layers. Finally, the last layer of the convolutional neural network formalizes its target tasks (classification or regression) into an objective function. Figure 3.3 shows the structure of the convolutional neural network.

3.2.1 Construction of CNN Models

This example is a convolutional neural network built by Keras. There are many data in Keras that can be used to train CNN. Here we use mnist function, which is a database containing 60,000 handwritten digit pictures that is commonly used for training various image processing systems. We built the following network architecture in our project:

$$[\text{Convolution}]^2-\text{[Maxpooling]}-\text{[Fully Connected]}^2$$
We want to train CNN to implement handwritten digit recognition and classification. First, build two convolutional layers using Conv2D function, which can extract features from 28 * 28 pixels pictures. Next, build a pooling layer by MaxPooling2D to compress the input features, which can simplify network computing complexity. Then, build two fully connected layers using Dense function, which can connect all the features and send the output value to the classifier.

When a complete data set passes through the neural network once and returns once, this process is called an epoch. It is not enough to transfer the complete data set once in the neural network, and we need to pass the complete data set multiple times in the same neural network. In this project, the epochs of the CNN model are 12.

3.2.2 Run on CPU and GPU

First, run the model on the CPU of whale server. Then, run the model on single GPU of whale server. Next, modify parameters of the model to run the model on two GPUs on whale server. Finally, run the model on the laptop, which has an Intel i7-6700HQ CPU and a Nvidia GTX950M GPU.
Methodology
Results and Evaluation

This chapter includes the evaluation methods and evaluation results of FC models, RNN models and CNN models. The evaluation method is to compare the running time of the same model on both CPU and GPU.

4.1 Results of ParaDnn

Figure 4.1 is a scatterplot, which shows the speedups of GPU over CPU. The x-axis is the number of model parameters and y-axis is speedups parameters, where

\[
\text{Params} = (\text{layer} + 1) \times (\text{node} \times \text{node} + \text{node}) \\
\text{Speedups} = \frac{\text{running time on CPU}}{\text{running time on GPU}}
\]

Data points with different numbers of nodes are represented in different colors. The speedups of FC models have large ranges, from 1 to 10. The running time of GPU is much less than the running time of CPU. As the complexity of the model increases, the effect of GPU acceleration becomes more significant.

For FC models, GPU is a better platform as its architecture can perform large-scale parallelism calculations. In addition, FC models rarely reuse weights and large models have more parameters, so they put a lot of pressure on the storage system [3]. The memory bandwidth of the GPU is higher, so running FC models on GPU is faster than CPU.

Figure 4.2 shows the result of RNN models running on CPU and GPU of the whale server. The x-axis is the number of model parameters and y-axis is speedups parameters, where

\[
\text{Params} = \text{vocabsize} \times (2 \times \text{embedding} + 1) + 601 \times \text{embedding} \times \text{layer} \\
\text{Speedups} = \frac{\text{running time on CPU}}{\text{running time on GPU}}
\]

We use different colors to represent data points with different batch sizes. It is obvious that the running time of GPU is much less than the running time of CPU. Batch size is the number of samples selected in one training. The batch size has a great influence on the speed of models. As the batch sizes of the model increases, the effect of GPU acceleration becomes more significant. This is mainly because increasing the batch size can improve memory utilization through parallelization.
**Figure 4.1:** FC models with large nodes are better suited for GPU than CPU.
Figure 4.2: Running on GPU is faster than CPU for RNN
4.2 Results of Keras

Figure 4.3 shows the each epoch running time on whale server. Figure 4.4 shows the average running time. It takes 19 seconds to run an epoch on the CPU, 7 seconds to run an epoch on a GPU, and less than 5 seconds to train with two GPUs. This shows that the GPU has a significant acceleration effect on computing, and the more GPUs used, the shorter the running time.

Figure 4.5 shows the result of running the CNN model on the laptop with an Intel i7-6700HQ CPU and a Nvidia GTX950M GPU. The running time of an epoch of the model on the CPU is about 70 seconds, and the running time on the GPU is about 20 seconds. It can be seen that the divergence between the GPU and CPU running time on the laptop is still very large. However, it can be seen that the running time on the GPU of the laptop is a little longer than the running time of the CPU on the whale server. This shows that not all GPUs are faster than CPUs because it depends on the computing capability of the hardware. Generally speaking, the computing capability of the GPU on a computer is faster than the computing capability of the CPU on this computer. The hardware equipment of different platforms is different. For example, the hardware performance of the server is much stronger than that of the laptop. Thus, we can use GPUs that have higher computing capabilities to accelerate computation and reduce running time.
Figure 4.4: The average running time
Figure 4.5: Running time of each epoch for CNN on the laptop
The GPU is composed of thousands of streaming processors, and single instruction multi-thread (SIMT) mode is used for execution. There are three dimensions in CUDA programming: grids, blocks and threads. A large number of threads form a block, and a large number of blocks form a grid. All threads in each block perform the same operation. If two 1000-element matrices are added together, 1000 threads will be started on the GPU, and one instruction is executed on 1000 threads at a time. Compared with the single-core CPU pipelined serial operation, simultaneous calculation through a large number of threads will obtain a considerable acceleration effect when the amount of data is very large, although the computing power of a single core of the GPU is weak. For CNN, the use of GPU acceleration is mainly in the convolution process, and the convolution process can be parallelized by CUDA like the vector addition. NVIDIA provides the cuDNN library. The deep learning framework implements the core operations of CNN and RNN models by passing tensors and calling the cuDNN library.
Conclusion and Future Work

This report provides a benchmark analysis comparing the performance of GPU and CPU on deep learning workloads. We use ParaDnn and keras to construct commonly used deep learning models, and we compare the performance of CPU, single GPU and two GPUs. Generally, running workloads on GPU is faster than CPU and use two GPUs can also reduce the running time compared to only one GPU. Not all GPUs are faster than CPUs, it depends on the device computing capabilities.

However, this project still has some limitations. The models we used are too simple, and the datasets are not complicated enough. The number of hardware platforms is not enough. In future, we need to find more benchmarks with more datasets, and compare the running time on more hardware platforms.
Conclusion and Future Work
Bibliography


Appendix 1: Final Project Description

This is a deep learning research project involving in comparing the performance of the different hardware platforms. The aim of the project is to choose an appropriate benchmark to analyze the performance between various devices in training deep learning workloads. The running time of deep learning models in different devices can be used to compare the computing performance of CPU and GPU.

This project involves five main tasks:

1. Construction of ParaDnn benchmark: ParaDnn is a tool that can generate parameterized end-to-end models to run on target platforms. Fully connected networks (FC) and recurrent neural network were generated by ParaDnn. TensorFlow framework was used in this task to build these deep learning models.

2. Construction of convolutional neuralnet work based on Keras: A convolutional neural network was built in this task based on Keras, and the dataset for training the model is MNIST (Mixed National Institute of Standards and Technology database), which is comprised of a training set of 60,000 images and a test set of 10,000 images.

3. Results analysis: Analyze the performance by comparing the running time of workloads on different hardware platforms.
Appendix 2: Independent Study Contract

INDEPENDENT STUDY CONTRACT
PROJECTS

SECTION A (Students and Supervisors)

UNIT: 

SURNAME: 
FIRST NAME: 

PROJECT SUPERVISOR: 

FORMAL SUPERVISOR: 

COURSE CODE, TITLE AND UNITS: 

COMMENCING SEMESTER: 

PROJECT TITLE: 
Analyzing Machine Learning Workloads on Contemporary Processors

LEARNING OBJECTIVES:
An improved knowledge and working experience of Deep Learning methods, algorithm and software, plus that of contemporary processors, will be obtained. In the use of performance evaluation including tools will also be acquired.

PROJECT DESCRIPTION:
Machine Learning workloads are becoming increasingly more prevalent and compute intensive. They can run on modern computer processors and accelerators such as GPUs, as well as custom or semi-custom devices such as Tensor Processing Units and Qualcomm's Snapdragon DSP core.

This project will involve the benchmarking and performance analysis of various ML with an emphasis on Deep Learning workloads, on a selection of processors, including modern x86-64 processors, GPUs and custom devices. The goals will be to first measure the dominant features to the workloads and their characteristics (e.g., memory working set, intensity) and to evaluate the effectiveness of the different classes of algorithms or processing them.

Research School of Computer Science

Page updated Jan 2018
ASSESSMENT (as per the project course's rules and pages, with any differences noted below).

<table>
<thead>
<tr>
<th>Assessed project components</th>
<th>% of mark</th>
<th>Due date</th>
<th>Evaluated by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report title: (e.g. research report, software documentation)</td>
<td>70</td>
<td></td>
<td>examiner</td>
</tr>
<tr>
<td>Assessed tasks: (e.g. software, data analysis, model...)</td>
<td>20</td>
<td></td>
<td>examiner</td>
</tr>
<tr>
<td>Final submission</td>
<td>100</td>
<td></td>
<td>examiner</td>
</tr>
</tbody>
</table>

MEETING DATES (IF KNOWN):

STUDENT DECLARATION: I agree to fulfill the above-defined contract.

Signature: 
Date: 04/06/2017

SECTION B (Supervisor):

I am willing to supervise and support this project. I have checked the student's academic record and believe this product can complete the project. I authorize the following examiner, and have obtained their consent to review the report (see signature below or attached email).

Signature: 
Date: 21/06/19

Comments: 
Name: V. H. Nguyen 
Signature: Research School of Computer Science 
[Linked to human and email, signature may be subject to change as requested by the supervisor or course coordinator]

REQUIRED DEPARTMENT RESOURCES:

SECTION C (Course coordinator approval):

Signature: 
Date: 11/08/19

Research School of Computer Science 
Form updated Jan 2018
Appendix 3: READ ME

1. ParaDnn

ParaDnn is a tool that generates parameterized deep neural network models. It provides large “end-to-end” models covering current and future applications, and parameterizing the models to explore a much larger design space of DNN model attributes.

- **Requirements:**
  python >= 3.7.6
  1.13 >= TensorFlow >= 1.6

- **Environment:**
  pip3 install --user --upgrade tensorflow==1.13.1
  pip3 install --upgrade google-api-python-client
  pip3 install --upgrade oauth2client
  pip3 install notebook
  pip3 install seaborn
  pip3 install matplotlib
  pip3 install sklearn

- **Run on CPUs and GPUs**
  To run FC and RNN models on CPUs, do
  cd paradnn/
  bash run/fc_cpu.sh
  bash run/rnn_cpu.sh
  You can also modify the hyperparameter ranges in paradnn/run/fc_cpu.sh and rnn_cpu.sh.

  To run FC and RNN models on GPUs, do
  bash run/fc_gpu.sh
  bash run/rnn_gpu.sh
  You can also modify the hyperparameter ranges in paradnn/run/fc_gpu.sh and rnn_gpu.sh.

  Collect performance data from execution logs
  cd ../scripts
  python3 get_perf.py

  Run the analysis tools
  cd scripts/plotting
  jupyter notebook

2. Keras

We use Keras to build a convolutional neural network.
Modify the n_gpus to change the number of GPUs.
model = to_multi_gpu(model, n_gpus=2)

To run CNN models on GPUs, do
python3 mnist_cnn.py