Code Performance and Scaling

Machine Learning

MLP vs. CNN

The performance and scaling of multilayer perceptron (MLP) and Convolutional neural networks (CNN) for images classification is tested in this report. The summary of machine learning (ML) cases for benchmark is shown in Table 1.

**Table 1.** A summary of ML cases for benchmark using TensorFlow

|  |  |  |
| --- | --- | --- |
| **ML cases** | **Data sets** | **Clusters** |
| Multilayer perceptron (MLP) classification | Images | Bridges2 GPU-AI |
| Convolutional neural networks (CNN) classification | Images | Bridges2 GPU-AI |

**Chart, bar chart

Description automatically generated**

**Figure 1**

**Figure 1** compares the training speed of MLP and CNN on my local CPU (Intel i5-1038NG7) and Bridges2 GPU-AI (NVIDIA Tesla V100 32 GB). The speed comparison between GPU and CPU by two machine learning methods is shown in Figure 1. It is obvious that the ML code runs more efficiently on GPUs than CPU. The training time by GPU is got from report of HW1 and HW2. The result of training time by my local CPU is attached in the appendix. All the calculations didn’t include the first epoch, because the training time of the first epoch doesn’t stable which includes the time of loading data.

Chart, bar chart

Description automatically generated

**Figure 2**

**Figure 2** shows the computational time per epoch of the MLP model on different numbers of GPUs (1, 2, 4, 8). Three different batch sizes (256, 512, and 1024) are tested. All the results of training time are included in the homework folder. When more GPUs and bigger batch size is used, the calculation speed should be quicker. However, the conclusion cannot be clarified in my test. There is a guess that it might because the GPUs capability I requested are different every time.

Chart, line chart

Description automatically generated

**Figure 3** is the speedup image of performance of different GPUs. There is no clear conclusion can be got from this experiment. There are two guessed reasons: the first one is that different type of GPU capability. The second is that I use MLP as a ML method to test the GPU performance, the method might be too simple to be tested.

**Challenges**: I met lots of unexcepted problems in this project. The most challenge one is that I have been waiting in the queue for so long time (more than 5.5 hours), and “Disk quota exceeded” error always happen. There is another problem that annoys me a lot is that I use the same .py file and just request different number of GPUs, some jobs completed with no error and give the result, some completed with no error but don’t give me the result, some failed but with no error mentioned in the script file. I attached all the output situations I met in the homework folder. The last thing annoyed me most is that my running result doesn’t make a lot of sense to me. Sometimes, more GPUs and bigger batch size is used, the running time even become longer. There is another wired thing is that I cannot use VS code to login remotely to the server, but one of students in our class can login it by VS code. I don’t think it’s the problem of the laptop, because I changed 3 devices including MacBook laptop, windows laptop, and lab computer to try to login the server by VS code, it all failed. When I use FileZilla to transfer the revised code to the server, it doesn’t allow me to transfer it (even I selected overwrite, the code file just doesn’t be renewed). It only allows me to download files from the server. Those unexpected and wired things almost make me give up this project, but thanks for the homework extend permission.

**Appendix**

**Training time of MLP by local CPU:**

299/299 - 79s - loss: 1.8847 - accuracy: 0.8991 - val\_loss: 0.0010 - val\_accuracy: 1.0000

Epoch 2/10

299/299 - 50s - loss: 5.5711e-04 - accuracy: 1.0000 - val\_loss: 4.7078e-04 - val\_accuracy: 1.0000

Epoch 3/10

299/299 - 48s - loss: 2.3913e-04 - accuracy: 1.0000 - val\_loss: 2.2011e-04 - val\_accuracy: 1.0000

Epoch 4/10

299/299 - 47s - loss: 1.2612e-04 - accuracy: 1.0000 - val\_loss: 1.2098e-04 - val\_accuracy: 1.0000

Epoch 5/10

299/299 - 47s - loss: 7.1406e-05 - accuracy: 1.0000 - val\_loss: 8.3540e-05 - val\_accuracy: 1.0000

Epoch 6/10

299/299 - 47s - loss: 4.6005e-05 - accuracy: 1.0000 - val\_loss: 7.1658e-05 - val\_accuracy: 1.0000

Epoch 7/10

299/299 - 47s - loss: 2.9104e-05 - accuracy: 1.0000 - val\_loss: 4.9417e-05 - val\_accuracy: 1.0000

Epoch 8/10

299/299 - 47s - loss: 1.9913e-05 - accuracy: 1.0000 - val\_loss: 5.1366e-05 - val\_accuracy: 1.0000

Epoch 9/10

299/299 - 47s - loss: 1.4245e-05 - accuracy: 1.0000 - val\_loss: 3.4981e-05 - val\_accuracy: 1.0000

Epoch 10/10

299/299 - 49s - loss: 1.0508e-05 - accuracy: 1.0000 - val\_loss: 2.3929e-05 - val\_accuracy: 1.0000

Eclapse time: 510.28358817100525s

**Training time of CNN by local CPU:**

299/299 - 482s - loss: 0.4363 - accuracy: 0.9589 - val\_loss: 0.0031 - val\_accuracy: 0.9996

Epoch 2/10

299/299 - 434s - loss: 1.2134e-04 - accuracy: 1.0000 - val\_loss: 0.0010 - val\_accuracy: 0.9996

Epoch 3/10

299/299 - 447s - loss: 3.6739e-05 - accuracy: 1.0000 - val\_loss: 5.8497e-04 - val\_accuracy: 0.9996

Epoch 4/10

299/299 - 438s - loss: 2.2527e-05 - accuracy: 1.0000 - val\_loss: 2.5207e-04 - val\_accuracy: 1.0000

Epoch 5/10

299/299 - 437s - loss: 1.1684e-05 - accuracy: 1.0000 - val\_loss: 1.6041e-04 - val\_accuracy: 1.0000

Epoch 6/10

299/299 - 429s - loss: 0.3985 - accuracy: 0.9723 - val\_loss: 0.6923 - val\_accuracy: 0.5092

Epoch 7/10

299/299 - 423s - loss: 0.7253 - accuracy: 0.5175 - val\_loss: 0.6930 - val\_accuracy: 0.5092

Epoch 8/10

299/299 - 425s - loss: 0.7041 - accuracy: 0.5160 - val\_loss: 0.6930 - val\_accuracy: 0.5092

Epoch 9/10

299/299 - 431s - loss: 0.6931 - accuracy: 0.5092 - val\_loss: 0.6930 - val\_accuracy: 0.5092

Epoch 10/10

299/299 - 436s - loss: 0.6931 - accuracy: 0.5092 - val\_loss: 0.6930 - val\_accuracy: 0.5092

Restoring model weights from the end of the best epoch.

Epoch 00010: early stopping

Eclapse time: 4381.370968818665s