Deep Learning Lab 3 Report: High-Performance Computing of Deep Learning

Zhang Shanshan

Course 2022 - 2023 Q1
Barcelona, January 12, 2023

Contents

1 Introduction

The Power9 cluster is a heterogeneous CPU-GPU cluster in Barcelona Supercomputer Center with its main computational power provided by NVIDIA GPUs. In this lab, we are going to utilize Power9 cluster to test the performance of our algorithms. The main components of the hardware are listed below.

- **CPU** 2 × IBM Power9 8335-GTH @ 2.4GHz
    1. 3.0 GHz on turbo
    2. 20 cores and 4 threads/core, total 160 threads per node
- **Memory** 512GB of main memory distributed in 16 dimms × 32GB @ 2666MHz
- 4 × GPU NVIDIA V100(Volta) with 16GB HBM2.

By following the guidelines for this lab, we divide our report into four parts w.r.t four required experiments. We re-implement the code of all of the experiments by tensorflow 2 and python 3.8, in the later parts, if not specified, we follow this software configuration.

2 Experiment1: Gradient Descent

In this part, we are required to find the linear model $y = Wx + b$ that better fits a set of points. Our implementation includes deployments of different optimizers and implementation of training at different orders of magnitude of learning rate. For optimizer, we implement basic **Gradient Descent Optimizer**. For learning rate, we have tried **0.0001, 0.001, 0.01, 0.1**. For all the settings, we run for 1000 epochs since the training process can be finished very quickly. Here we list our results of 1000 epochs as below.

Learning Rate	Weight	Bias	MSE
0.1	-1	1	2.1672e-13
0.01	-0.9938	0.9819	5.5029e-05
0.001	-0.9085	0.7310	0.0121
0.0001	-0.8524	0.6572	0.0279

Table 1: Gradient Descent Optimizer Result

By the way, we also implemented the linear regression model from the sklearn library, the training process was finished in only several seconds and gives us a really nice result, it should be mentioned that the score in the last column is the coefficient of determination of the prediction which is calculated by the formula $(1-\frac{u}{v})$, where u is is the residual sum of squares $((y\_true-y\_pred)^2).sum()$ and v is the total sum of squares $((y\_true-y\_true.mean())^2).sum()$. The best possible score is 1.0.

Model	Weight	Bias	Score
Linear Regressor	-1	1	1

Table 2: Result of Linear Model of Sklearn

3 Experiment 2: Single Layer Neural Network

In this task, we load the MNIST dataset from the official API of TensorFlow datasets directly, flatten the 2D input image with 28×28 pixels to a 1D tensor (size of 784 after flattening), and add the classification layer with 10 neurons on top. Base on our past experience, Adam should be the most widely used optimizer in deep learning approaches, so we simply work on it to test the performance of different learning rates, **0.1, 0.01, 0.001, 0.0001** as shown in figure 1. It could be easy to figure out if we reach the best result by setting the learning rate to 0.001, so in the later part we will keep this rate for our implementation.

Figure 1: Training Result of different learning rates by using Adam

With respect to optimizers, we implemented 10 kinds of them including **stochastic gradient descent (SGD)**, **SGD with Momentum**, **SGD with Momentum and Nesterov**, **AdaGrad**, **Root Mean Square Propagation (RMSProp)**, **Adam**, **Adamax**, **Adadelta**, **Adadelta**, **Nadam**, **Adam with weight decay (AdamW)** with the learning rate = 0.001.

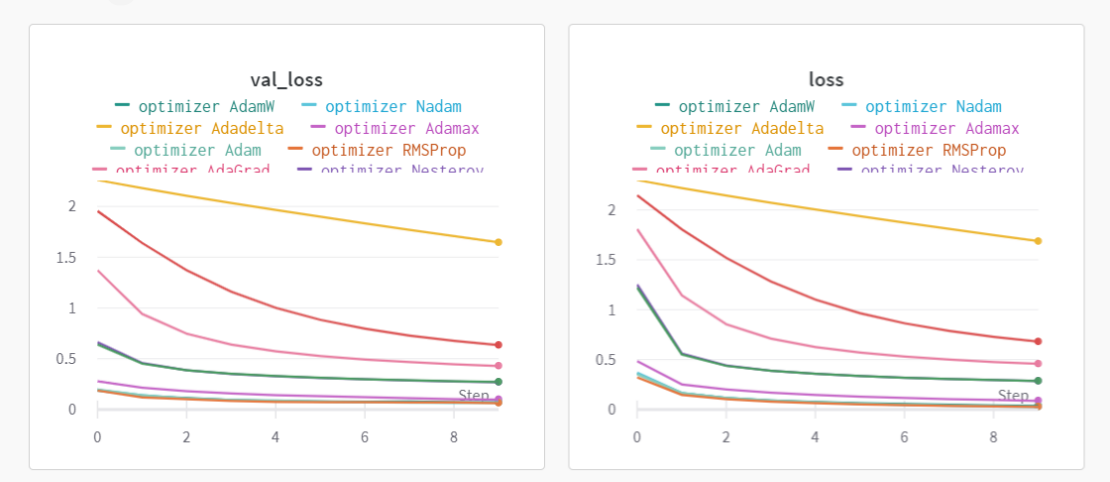

Figure 2: Training and validation loss of different optimizers

It is obvious that the optimizers of **Adagrad**, **SGD**, **Adadelta**, **SGD with Momentum** give us kind of bad results, but if you take a look at the SGD momentum optimizer, we can conclude that adding the momentum term really makes sense. By simply adding momentum we improve the validation accuracy from 85.58% to 92.44%. For the performance of optimizers of **AdamW**, **Nadam**, **Adam**, **RMSProp**, we have not really found much

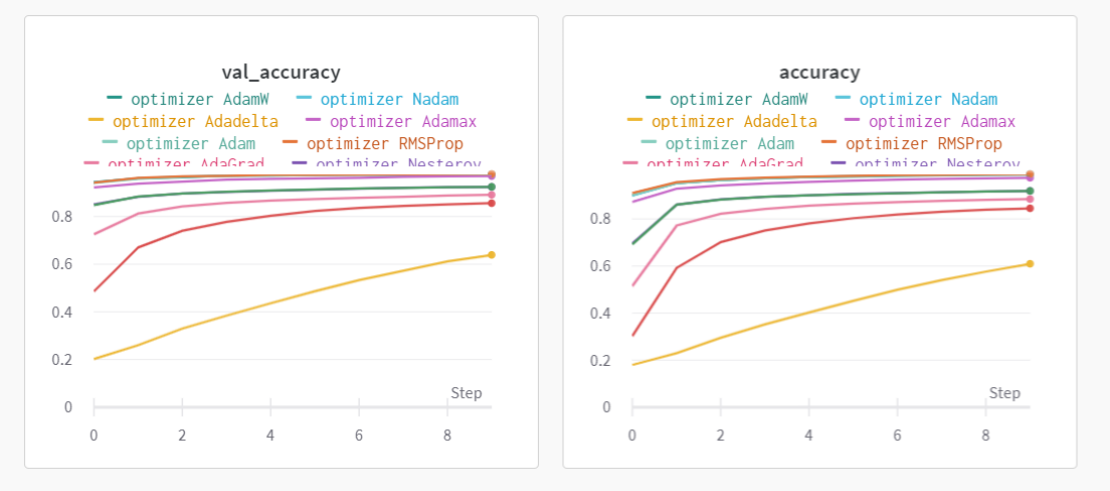

Figure 3: Training and validation accuracy of different optimizers

Acc	SGD	Momentum	Nesterov	AdaGrad	RMSProp	Adam	Adamax	Adadelta	Nadam	AdamW
Val	0.8558	0.9244	0.9241	0.8910	0.9785	0.9782	0.9694	0.6390	0.9759	0.9775
Train	0.8452	0.9189	0.9194	0.8844	0.9912	0.9909	0.9749	0.6095	0.9913	0.9891

Table 3: Summarization of Linear Model of Sklearn

difference based on their curves. Based on our experience with deep learning, **AdamW** has been proven to be the best optimizer in many cases and will be adopted for the future Convolutional Neural Network part.

4 Experiment 3: Multiple Layer Neural Network

According to the lab guide, we build 2 layers of convolutional layer followed by a max-pooling layer, then train the model on different **batch size (= 32, 64, 128)** and different learning rates. We reach the performance, **val_acc=0.9936** with the setting of **batch size = 128, epochs=10, learning rate = 0.001**. Actually, the top performances of different batch size are more or less the same, but it should be noted that **with batch size = 128, we got the most reliable validation accuracy**. As we can see, just like we have observed in Experiment 2, **learning rate = 0.001** produces the best result.

Figure 4: Training and validation accuracy of different learning rate

Figure 5: Training and validation loss of different optimizers

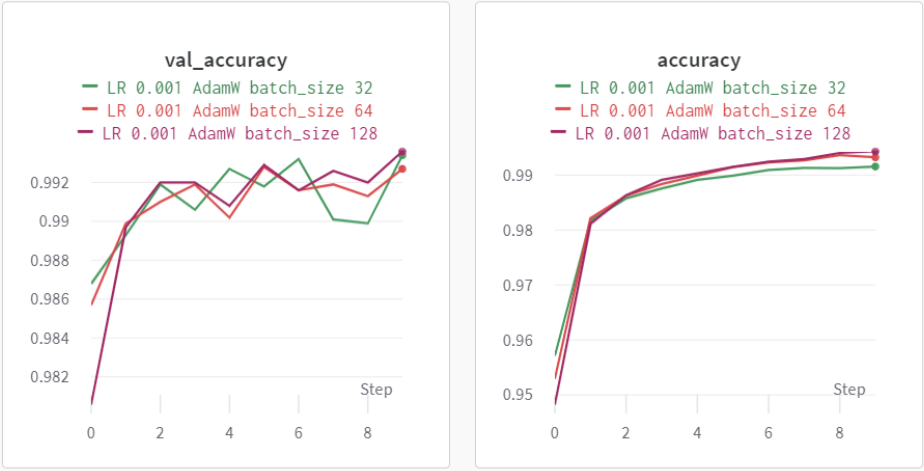

Figure 6: Training and validation accuracy of different batch size

5 Experiment 4: Multiple Layer Neural Network by Multiple GPUs

5.1 Executing by 1 GPU

In experiment 3 it is executed all by GPU we rent online, its configuration is as follows

- **CPU** 16 vCPU Intel(R) Xeon(R) Platinum 8350C CPU @ 2.60GHz
- **GPU** RTX 3090(24GB)

So in this part, there is no difference from Experiment 3.

5.2 Executing by 2 GPUs

My execution file is blocked in BSC queues, so eventually, we rent our GPU clusters which support renting multiple GPUs. Here we use $2RTX3090$ to train the same model in 1 GPU. we tried different batch sizes including very large ones, **[32, 64, 128, 256, 512, 1024, 2048, 4096, 8192]**, since we really want to take benefit of our powerful hardware, and we got the following conclusions.

- **For large batch size, 2048, 4096, 8192**, they hurt the performance at the beginning of training, such as the first 4 to 5 epochs.
- **But all the batch sizes reach the performance at last** as shown in figure 7.
- **Smaller batch size** will results in a longer training time, which means the batch size of 32 leads to the longest training time (212 seconds). When we apply a very large batch size, the training process can be finished in 30 to 40 seconds as shown in figure 8 and figure 9
- For other batch sizes ($<= 1024$), we got more or less the same performance.

Figure 7: Validation loss and accuracy of different batch sizes

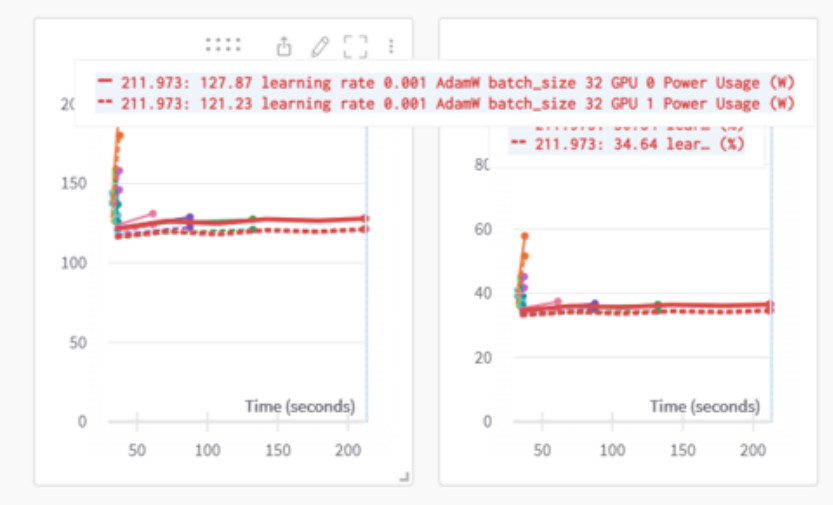

Figure 8: Training time of batch size 32

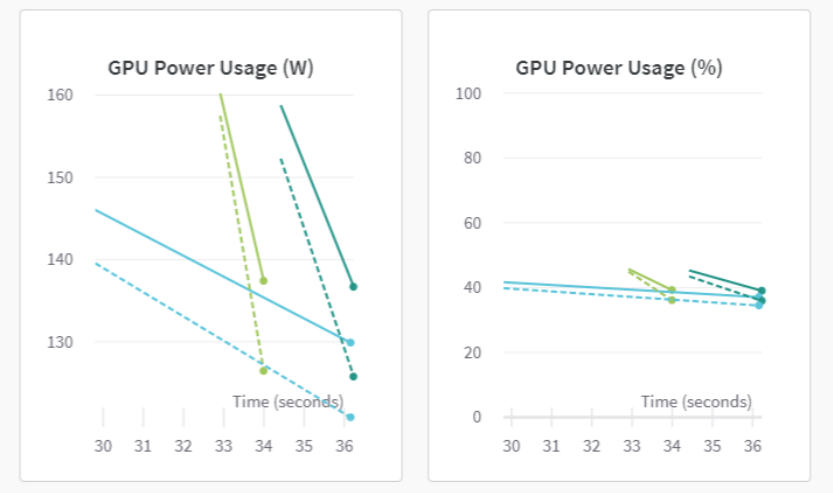

Figure 9: GPU usage time of batch size 8192, 4096, 2048

5.3 Executing by 4 GPUs

In this part, we train our model at a batch size of **128, 512, 2048, 8192** and got the following conclusions.

- **For large batch size, 2048, 4096, 8192**, they hurt the performance at the beginning of training, such as the first 4 to 5 epochs. **But all the batch sizes reach the performance at last** as shown in figure 10.
- **It is surprising to see that our workload is distributed evenly among all 4 GPUs** as shown in figure 11
- The performances by using different numbers of GPUs are more or less the same as shown in figure 12.

Figure 10: Performance of batch size 8192, 2048, 512, 128

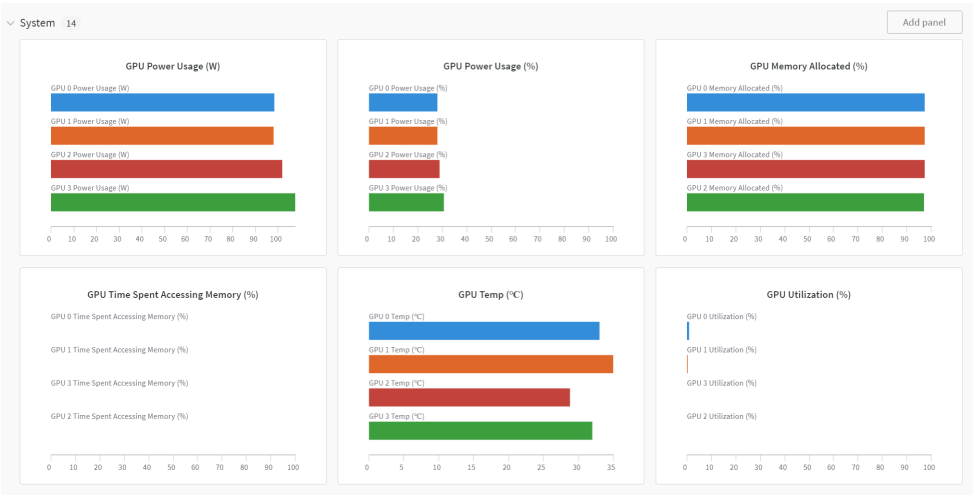

Figure 11: GPU Power and Memory allocation

Figure 12: Performances by using 1 GPU, 2 GPUs, 4GPUs

6 Conclusion

- **Experiment 1** By Gradient Descent optimizer, we get very close to the ground truth after 1000 epochs with learning rate = 0.1, and the linear model of sklearn also fits the ground truth perfectly.
- **Experiment 2** By trying different learning rates, learning rate = 0.001 is significantly better than the others, but for optimizers, we got pretty nice but quite similar results with **AdamW**, **Nadam**, **Adam**, **RMSProp**.
- **Experiment 3** By training our model on an RTX 3090 GPU, we got the validation accuracy of 0.9936 with batch size = 128 and learning rate = 0.001.
- Experiment 4 For a single GPU we follow the result of Experiment 3, for multiple GPUs, if the training hyperparameters are same, we more or less got the same validation accuracy. Batch size can both hurt the performance and impact the training time.

.