Measurement of machine learning performance with different condition and hyperparameter

Thesis

Presented in Partial Fulfillment of the Requirements for the Degree Master of Science in the Graduate School of The Ohio State University

By

Jiaqi Yin, M.S

Graduate Program in Electrical and Computer Engineering

The Ohio State University

2020

Thesis Committee

Xiaorui Wang, Advisor

Irem Eryilmaz

Abstract

In this article, we tested the performance of three major machine learning frameworks under deep learning: TensorFlow, PyTorch, and MXnet.The experimental environment of the whole test was measured by GPU GTX 1060 and CPU Inter i7-8650.During the whole experiment, we mainly measured the following indicators: memory utilization, CPU utilization, GPU utilization, GPU memory utilization, accuracy and algorithm performance. In this paper, we compare the training performance of CPU/GPU; under different batch sizes, various indicators were measured respectively, and the accuracy under different batch size and learning rate were measured.

Vita

2018.....B.S. Electrical Engineering, Harbin Engineering University

2018 to present.....Department of Electrical and Computer Engineering, The Ohio State University

Fields of Study

Major Field: Electrical and Computer Engineering

Table of Contents

List of Tables

List of Figure

Chapter 1. Introduction

There is no doubt that deep learning plays an important role in the field of computing. A series of machine learning frameworks such as TensorFlow, MXnet, PyTorch have emerged. Developers can use the parallel computing capabilities of the GPU to perform routine calculations and speed up the speed of machine learning. Although these deep learning frameworks are widely used, their architecture and internal differences make them different in their performance. In terms of cost, time, it is necessary to measure a series of indicators such as training performance under different frameworks to choose the appropriate machine learning framework.

The release of machine learning frameworks has promoted the prosperity of deep learning, and GPU versions under different learning frameworks have been continuously released and updated. Our goal is to evaluate machine learning performance under various conditions and hyperparameters. We will evaluate a set of indicators including GPU utilization, memory utilization, training latency, accuracy.

Following contents is our test platform: Windows 10, GPU GTX 1060, CPU Inter i7-8650, memory 16GB

After measurement, we got several insights:

•GPU can improve training speed nearly 5 times.

•The larger the batch size, the larger the memory consumption, the larger the GPU usage; the faster the training speed, but if batch size is too large, the accuracy of the model will decrease.

•Under PyTorch_1.4.0 framework, batch size has little impact on the performance of PC. Possibly because PyTorch_1.4.0 uses some strategies to alleviate the batch size fluctuation.

•The initial learning rate has an optimal value. Too large leads to the model being divergent. Too small leads to slow convergence.

•We also discuss the selection method of batch size, learning rate, and how GPU memory was occupied.

Chapter 2. Background

2.1 TensorFlow,PyTorch and MXnet

MXnet：

Apache MXnet is an open source deep learning software framework for training and deploying deep neural networks. MXnet is extensible, allows rapid model training, and supports flexible programming models and multiple programming languages (including C ++, Python, Julia, Matlab , JavaScript, Go, R, Scala, Perl, and Wolfram Language)[1]. The MXnet library can be extended to multiple GPUs and multiple machines and is portable. MXnet is supported by public cloud providers Amazon Cloud Computing Services (AWS) and Microsoft Azure. Amazon has selected MXnet as AWS's preferred deep learning framework.

PyTorch：

PyTorch is a Python version of torch, a neural network framework open sourced by Facebook, specifically for GPU-accelerated deep neural network (DNN) programming. Torch is a classic tensor library that operates on multidimensional matrix data. Learning and other mathematically intensive applications have a wide range of applications. Unlike

TensorFlow's static computation graphs, PyTorch's computation graphs are dynamic and can be changed in real-time based on computational needs[2].

TensorFlow：

TensorFlow is an open source software library for machine learning for various language understanding tasks. It is currently used by over 50 teams to research and used to produce many Google commercial products such as speech recognition, Gmail, Google Photos, and search.

TensorFlow was originally developed by the Google Brain team for Google research and production, and was released under the Apache 2.0 open source license on November 9, 2015[3].

A comparison of some of the basic attributes of the four frameworks is shown in Table 1:

	TensorFlow	MXnet	PyTorch
supported language	C++/Python	Python/C++/R	Python
supported hardware	CPU/GPU/Mobile	CPU/GPU/Mobile	CPU/GPU
Imperative programming	√	√	√
Symbolic programming	√	√	×

Table 1 Comparison of frameworks

Most of the Python code are imperative programming, for example,

```
import numpy as np
```

```
a = np.ones(10)
b = np.ones(10) * 2
c = b * a
d = c + 1
```

When the program executes to c=b * a, the code begins the corresponding numeric calculation.

Rewrite the example with symbolic programs:

```
import numpy as np
A = Variable('A')
B = Variable('B')
C = B * A
D = C + Constant(1)
f = compile(D)
d = f(A=np.ones(10), B=np.ones(10)*2)
```

In symbolic-type program, you need to first give a function definition (which can be very complicated). When we define this function, we don't really do numerical calculations. A numeric placeholder is used in the definition of such a function, which is not compiled until the actual input is given.

Symbolic programming is more efficient in both memory and speed, and the same code typically takes up less memory. Although PyTorch adopts symbolic programming, in practice, PyTorch is no less efficient than TensorFlow or MXnet due to other optimizations

2.2 GTX-1060

NVIDIA GeForce 10 series is a graphics processor series developed and launched by NVIDIA, which is used to replace NVIDIA GeForce 900 series graphics processors. The new series uses Pascal microarchitecture to replace the previous Maxwell microarchitecture and uses TSMC's 16nm and Samsung's 14-nanometer fin multi-gate transistor process (Fin-FET).

Nvidia has announced that the highest specifications of Pascal GPUs use four sets of 4-Hi HBM2 video memory, and high-end consumer graphics card memory up to 16GB. The high-performance computing GPU also equipped with shared memory technology and NVLink bus.2[4].

	Cuda Cores	Base Clock （MHz）	Boost Clock （MHz）	Memory Data Rate
NVIDIA TITAN X	3584	1417	1531	10Gbps
GeForce GTX 1080	2560	1556	1733	10Gbps
GeForce GTX 1070	2048	1442	1645	8Gbps
GeForce GTX 1060	1280	1506	1708	8Gbps
GeForce GTX 1050Ti	768	1290	1392	7Gbps
GeForce GTX 1050	640	1354	1455	7Gbps
NVIDIA TITAN X	memory data rate	total video memory	memory bus width	memory band width(GB/s)
GeForce GTX 1080	10Gbps	12GB GDDR5X	384-Bit	480
GeForce GTX 1070	10Gbps	8GB GDDR5X	256-Bit	320
GeForce GTX 1060	8Gbps	8GB GDDR5X	256-Bit	265
GeForce GTX 1050Ti	8Gbps	8GB GDDR5X	192-Bit	192
GeForce GTX 1050	7Gbps	4 GB GDDR5	128-Bit	112

Table 2 Nvidia mainstream graphics card parameters [5]

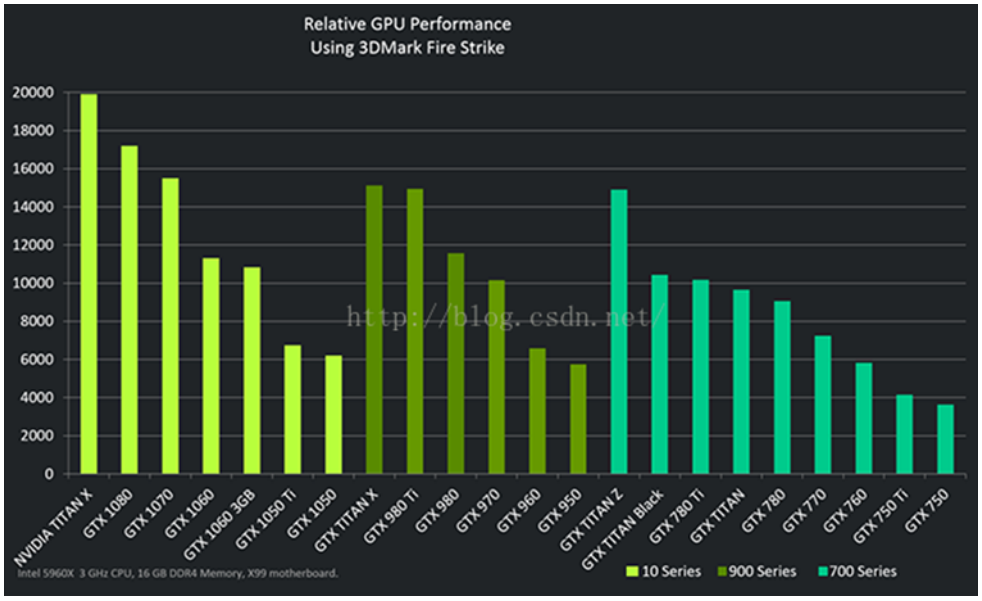

Figure 1 Nvidia mainstream graphics card performance

2.3 Environment

In this section, we will introduce the test environment of the test platform. We will also introduce the benchmark indicators used in our testing. In order to ensure that our tests are repeatable, we will write down our test environment in detail as much as possible.

Test environment:

We have installed GTX 1060 on our mainstream test computer. I think our test environment is representative for most readers.

CPU	Inter(R) Core(TM) i7-8650U CPU @ 1.90GHz
GPU	Nvidia GTX 1060
OS	Windows 10
Disk	SAMSUNG MZFLW512HMJP-000MV 512GB
Memory	16G DDR3 Speed:1867MHz
Motherboard	Microsoft (7th/8th Generation Intel Processor Family I/O - 9D4E)

Table 3 Environment

Table 4 shows the framework versions and drivers we installed. For all frameworks we use fp32 precision by default.

TensorFlow	TensorFlow-2.0.0
PyTorch	PyTorch-1.4.0
MXnet	MXnet-1.5.0
CUDA	10.0.0
cuDNN	7.6.5

Table 4 Framework versions and drivers

Third parties such as MLPerf have obtained detailed training performance results in multiple GPUs. In this article, we will only introduce a series of experiments on GTX 1060. For ML practitioners, this report will visually introduce the performance under this GPU [6].

We have built the evaluation experiments to different CIFAR10 datasets. Regarding the evaluation metrics, we provide CPU utilization percentage, GPU utilization percentage, GPU memory utilization percentage, CPU memory utilization percentage, accuracy and training speed. In this case, you can have a comprehensive impression to these results. These utilization indicators are finally expressed as average values. Data are recorded every 5 seconds, and the average usage is calculated based on the recorded data after the experiment.

Chapter 3 CPU/GPU performance comparison

3.1 Experimental results

After the configuration and experimental stages, we have obtained the following data results in Table 5:

batch size 128 training	TensorFlow 2.0.0		PyTorch 1.4.0		MXnet 1.5.0	
CPU Usage	30.67%	73.49%	7.44%	64.49%	25.38%	59.16%
GPU Usage	26.00%		13.00%		34.83%	
CPU Memory Usage (MiB)	1999.930	3249.181	1724.223	859.468	2495.153	1574.410
GPU Memory usage (MiB)	641.000		502.642		961.211	
Performance (steps/sec)	16.148	3.328	11.917	4.192	32.22	3.65

Table 5 GPU/CPU comparison results

3.2 Result analysis

We will show our data in the form of charts so that we can compare them in different depth learning frameworks.

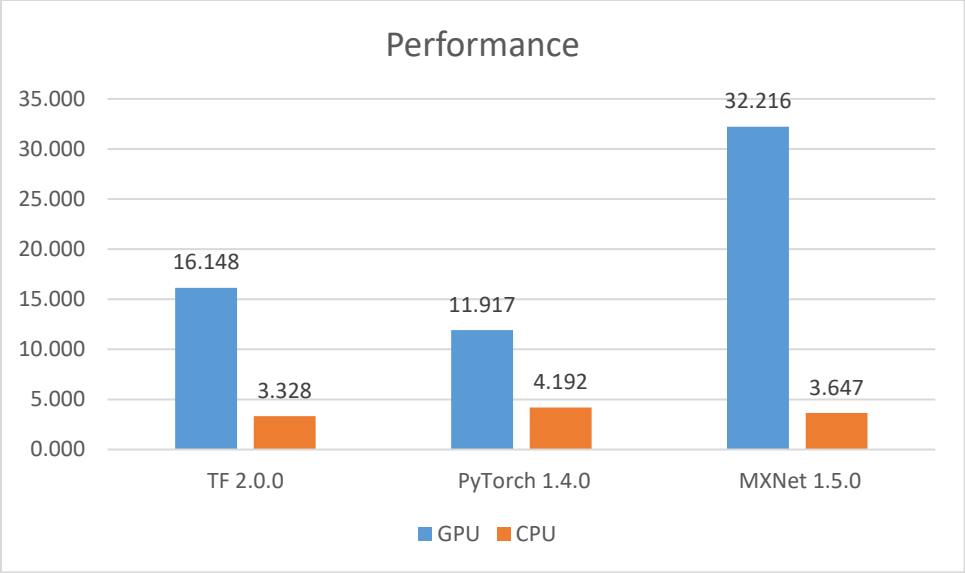

Figure 2 GPU/CPU Train Performance

Figure 2 shows the comparison of CPU/GPU performance. From the perspective of the three frameworks, the prediction speed of PyTorch 1.4.0 is approximately the same as that of TF 2.0.0. With the help of GPU, the speed of MXnet 1.5.0 is higher than rest of the other two machine learning frameworks. Perhaps it is because MXnet's GPU memory optimization is better.

In addition, by contrast, we find that for PyTorch 1.4.0 and TF 2.0.0, GPU can increase the speed of training by about 3 times. For MXnet, the running speed has increased from 3.65 steps/sec to 32.33 steps/sec.

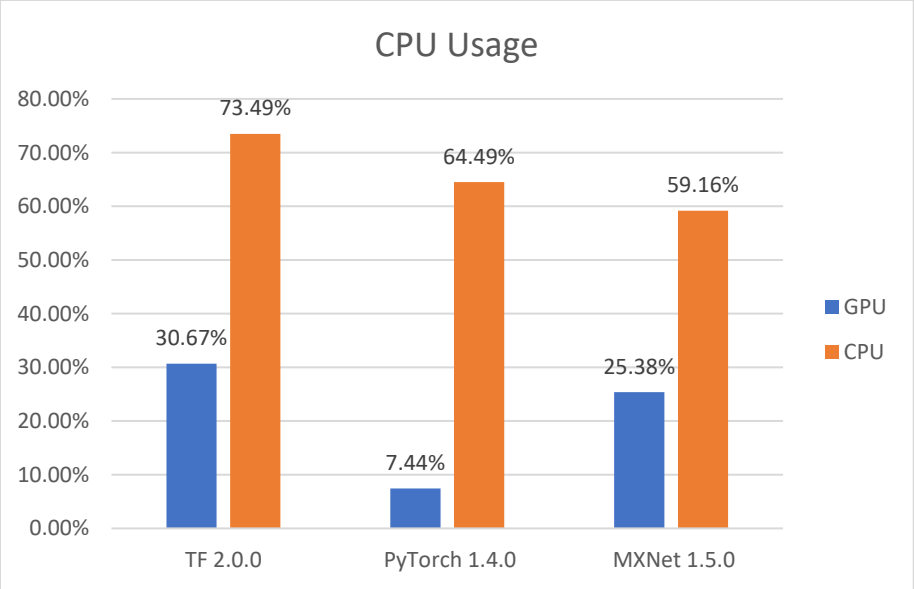

Figure 3 GPU/CPU-CPU Usage

In the three frameworks, the usage of TF 2.0.0 is the highest. When GPU training is started, the usage of CPU decreases to 30.67%. When CPU training is used to perform the task, the CPU usage reaches 73.49%. When using GPU for training, PyTorch 1.4.0 has very low CPU usage, accounting for only 7.44%. When the CPU is used for training alone, the CPU usage increased significantly, reaching 64.5%.

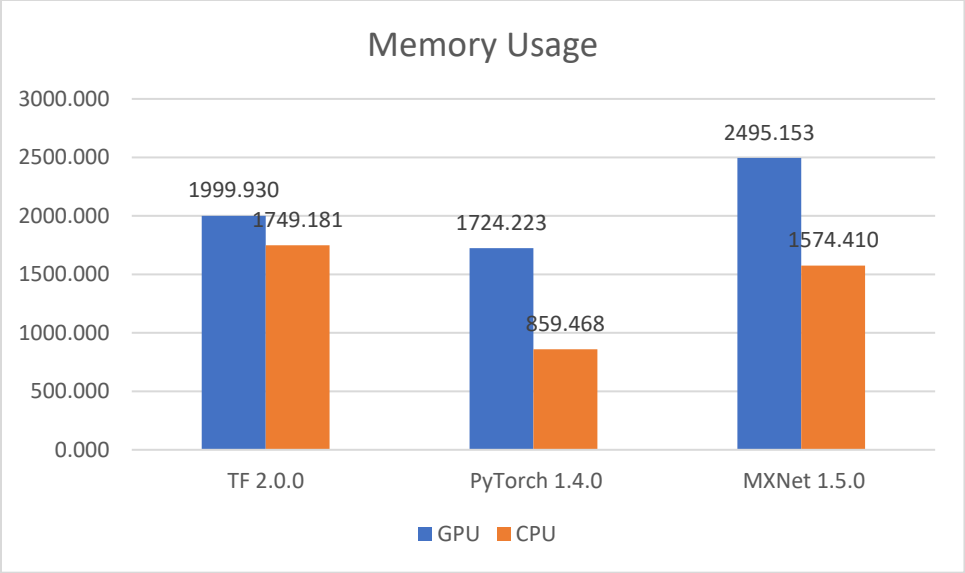

Figure 4 GPU/CPU-GPU Usage

We have made statistics on computer memory. Surprisingly, after using the CPU for training, the computer memory actually decreases. Possibly because that when using CPU to train, the data stored in memory has machine learning parameters (RNN parameters of each layer), batch data. But when training with GPU, except 2 items above, some of the data exchanged with the GPU may also be stored in memory.

It is worth noting that all of our experiments are open source codes. Some code may have specific performance optimization, which may lead to different results.

3.3 Insights

In Table 5, we can get some interesting findings:

- In general, TensorFlow2.0.0 needs more resources when running the same code, for example, tensorlow2.0.0 has a high CPU usage and memory usage.
- The prediction speed of PyTorch 1.4.0 is approximately the same as that of TF 2.0.0. When using GPU, MXnet_1.5.0 has the fastest training speed.
- After using the CPU for training, the computer memory decreases. A possible reason is that data exchanged with the GPU may also be stored in memory.
- The layers that occupy GPU memory are usually: conv2d layer, linear layer, batchnorm layer and embedding layer. And the ones that do not occupy the memory are: activation layer Relu, Pooling layer and Dropout level.
- The way to reduce memory consumption including: reduce size of input image, reduce batch, reduce the number of input images each time, more use of downsampling, pooling layer and optimize from the deep learning framework.

Chapter 4 Experimental data under different batch size

4.1 Experimental results

In this experiment, we only used GPU for training. Test data under different batch size: CPU utilization, GPU utilization, CPU memory, GPU memory and training speed. After the configuration and experimental phases, we obtained the following CPU data results in Table 6, Table 7 and Table 8:

TensorFlow_2.0.0	Train with GPU				
Batches	4	8	16	32	64
CPU Usage	23.63%	23.63%	25.03%	24.81%	26.85%
GPU Usage	5.65%	5.65%	29.01%	30.46%	36.17%
CPU Memory Usage (MiB)	1536.943	1987.022	1860.327	1963.994	2026.420
GPU Memory usage (MiB)	694.274	665.804	649.137	620.909	694.563
performance(steps/sec)	0.025	0.310	1.151	3.509	9.846
batches	128	256	512	1024	

Continued

Table 6 TensorFlow_2.0.0 measurement

Table 6 Continued

CPU Usage	30.67%	24.88%	22.45%	22.14%	
GPU Usage	26.00%	48.00%	61.15%	65.00%	
CPU Mem Usage(MiB)	1999.930	2071.256	2221.523	2221.523	
GPU Mem usage(MiB)	641.000	541.333	535.063	535.063	
performance(steps/sec)	16.148	62.710			

PyTorch 1.4.0					
batches	4	8	16	32	64
CPU Usage	12.27%	11.61%	10.88%	10.22%	9.22%
GPU Usage	9.59%	9.98%	11.46%	11.00%	13.29%
CPU Memory Usage(MiB)	1641.762	1663.916	1675.642	1647.161	1707.757
GPU Memory usage(MiB)	520.222	516.784	509.296	530.000	495.250
performance(steps/sec)	1.511	2.644	4.940	6.977	11.380
batches	128	256	512	1024	
CPU Usage	7.44%	6.30%	5.51%	5.22%	
GPU Usage	13.00%	9.91%	11.00%	9.67%	
CPU Memory Usage(MiB)	1724.223	1609.242	1614.986	1620.946	
GPU Memory usage(MiB)	502.642	503.478	525.818	553.524	
performance(steps/sec)	11.917	12.714	11.812	11.812	

Table 7 TensorFlow_2.0.0 measurement

MXnet 1.5.0					
batches	4	8	16	32	64
CPU Usage	28.17%	26.34%	23.87%	22.94%	20.87%
GPU Usage	22.13%	24.65%	30.22%	33.14%	34.91%
CPU Memory Usage(MiB)	2513.146	2479.037	2482.092	2487.127	2503.613
GPU Memory usage(MiB)	502.452	505.479	536.672	606.953	720.811
performance(steps/sec)	0.031	0.063	0.203	0.563	1.527
batches	128	256	512	1024	
CPU Usage	25.38%	18.02%	16.84%	17.21%	
GPU Usage	34.83%	38.33%	46.09%	46.09%	
CPU Memory Usage(MiB)	2495.153	2485.639	2491.920	2511.472	
GPU Memory usage(MiB)	961.211	1401.378	1952.543	2238.897	
performance(steps/sec)	3.222	7.067	14.317	29.435	

Table 8 MXnet_1.5.0 measurement

4.2 Result analysis

We will show our data in the form of charts so that we can compare them in different deep learning frameworks.

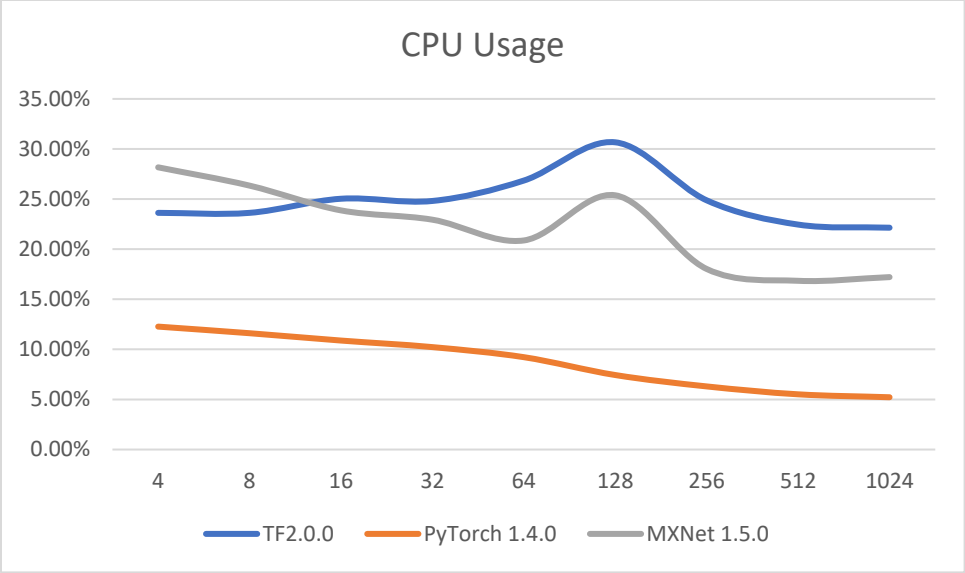

Figure 5 CPU Usage under 3 frameworks

Figure 5 shows the usage of CPU under the three frameworks. Basically under all batch size, the usage of TF 2.0.0 is the highest and its CPU usage is over 25%.

It is interesting that it seems that in the three frameworks, with the increase of batch size, the CPU usage is decreasing.

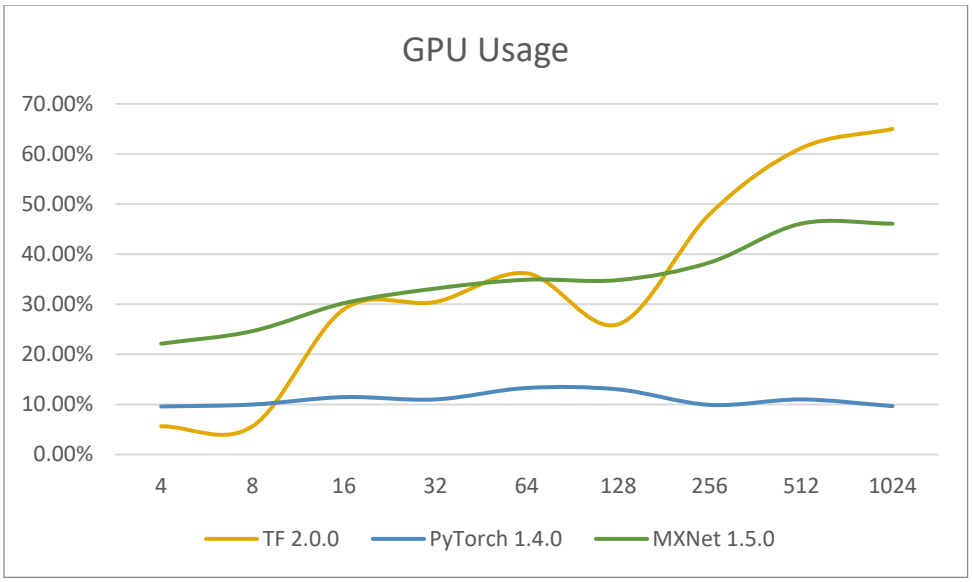

Figure 6 GPU Usage under 3 frameworks

Figure 6 shows the usage of GPU under the three frameworks.

We can find that when GPU is used and batchs_size is relatively small, the GPU usage of TF2.0.0 is the lowest, and MXnet is the highest. When batch size is relatively large, the GPU usage rate of TF 2.0.0 is the highest, and the PyTorch_1.4.0 is the lowest.

In addition, we can see that with the increase of batch size, the usage of GPU will also increase significantly.

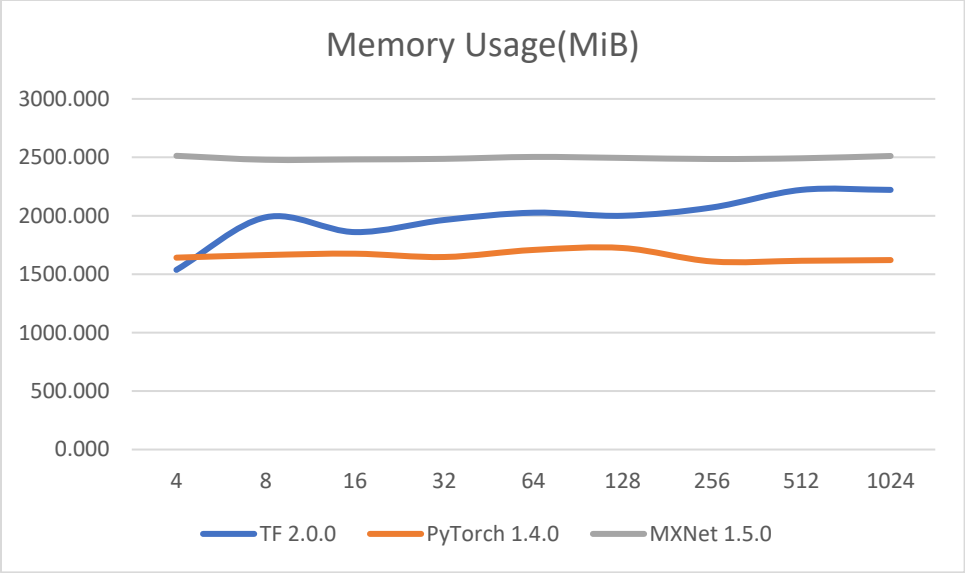

Figure 7 Memory under 3 frameworks

Figure 7 shows the memory space occupied under the three frameworks.

We can see that MXNET1.5.0 occupies the most memory resources, about 2500MB, and PyTorch 1.4.0 occupies the least memory resources, about 1650MB.

Besides, with the increase of batch size, the memory occupied by PyTorch 1.4.0 also increased slightly. The memory space of the rest two frameworks remain unchanged.

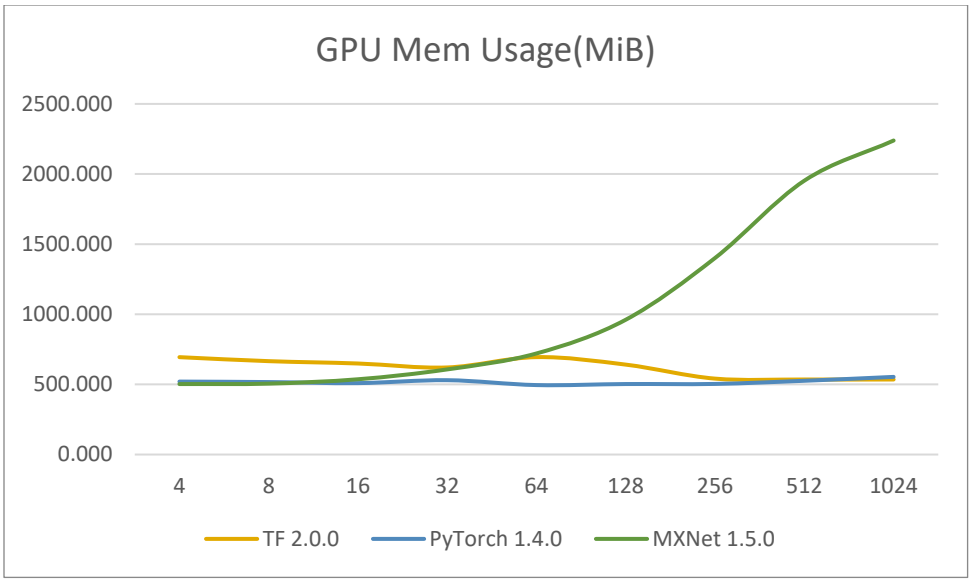

Figure 8 GPU Memory under 3 frameworks

Figure 8 shows the occupied memory space under the three frameworks.

We can find that the memory space occupied by MXnet_1.5.0 increases significantly with the increase of batch size, while the remaining two will hardly change. Generally speaking, MXnet_1.5.0 occupies the highest memory space. One possible reason is that MXnet_1.5.0 stores all the batch size data in the GPU memory and releases them after entire program were finished. On the other hand, TensorFlow_2.0.0 and PyTorch_1.4.0 process data in batches. After processing one epoches, they will be deleted from the memory.

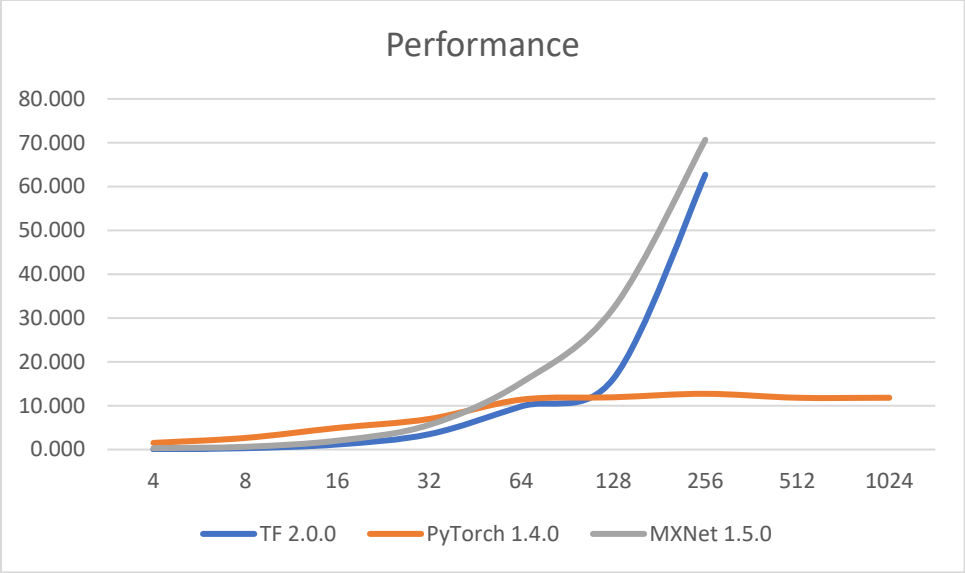

Figure 9 Performance under 3 frameworks

Figure 9 shows training speed under the three frameworks.

When batch size is relatively small, the training speed of PyTorch_1.4.0 is faster. When batch size increases, the training speed of MXnet_1.5.0 and TensorFlow_2.0.0 is obviously improved. When batch size is greater than 64, the training speed of MXnet_1.5.0 is the fastest.

The training speed represents the number of steps that can be trained per second. When batch size is too large, the accuracy of training epoch will decrease.

4.3 Insights

In the Table 6, Table 7 and Table 8, we can get some interesting findings:

- Large batch size may cannot improve training speed under some condition. When you make full use of the computing resources, the improve of performance is pretty limited [7].

  Assumes that all GPU processing units have been fully used:Increasing batch size can increase the speed, but it is very limited; small batch size can slow down the gradient oscillation. To achieve same accuraty, it needs less iteration times and converges faster, but each iteration takes more time.

  Large batch size (for example, batch size becomes the total number of samples) reduces the number of optimizations an epoch can perform, thus requiring more time to converge.
- Generally speaking, the larger the batch size, the larger the memory consumption. Memory consumption is not only related to the parameters of the model itself, but also each batch data.
- Although it is not effective in every case, a simple way to increase GPU usage is to increase batch size. In general, the GPU parallelly computes the gradient of the batch size. Therefore, as long as there is enough memory to accommodate the entire, large batch size can increase the GPU usage and training speed.
- With the increase of batch size, the training speed growth of PyTorch_1.4.0 is the slowest, and its GPU usage and the used memory are almost unchanged. A

reasonable guess is that PyTorch_1.4.0 has used a way to distribute strategy, which can reduce GPU usage, CPU memory and training speed when batch is too large and vice versa.

- In the case of small batch size, the effect of PyTorch_1.4.0 is better.If the batch size is large, MXnet_1.5.0 works better.

Chapter 5 Learning rate and batch size influence on training accuracy

5.1 Experimental results

Because different frameworks have little difference in training accuracy, we only tested TensorFlow_2.0.0.

Table 9 and Table 10 are the experimental results:

learning rate	0.001	0.005	1.00e-02	3.00e-02	5.00e-02	7.00e-02
accuracy after 10 epoches	47.81%	65.40%	69.60%	68.28%	64.03%	58.68%
time to achieve 60% accuracy	202.54	64.92	39.25	27.49	38.24	144.64
epoches to achieve 60%	24	7	4	3	4	16

Table 9 Accuracy under different learning rate

batches		8	16	32
accuracy after 10 epoches training	32.55%	45.83%	63.06%	68.59%
batches	128	256	512.000	1024
accuracy after 10 epoches training	69.77%	64.82%	57.28%	51.02%

Table 10 Accuracy under different batch size

5.2 Result analysis

We will show our data in the form of charts so that we can compare them in different conditions.

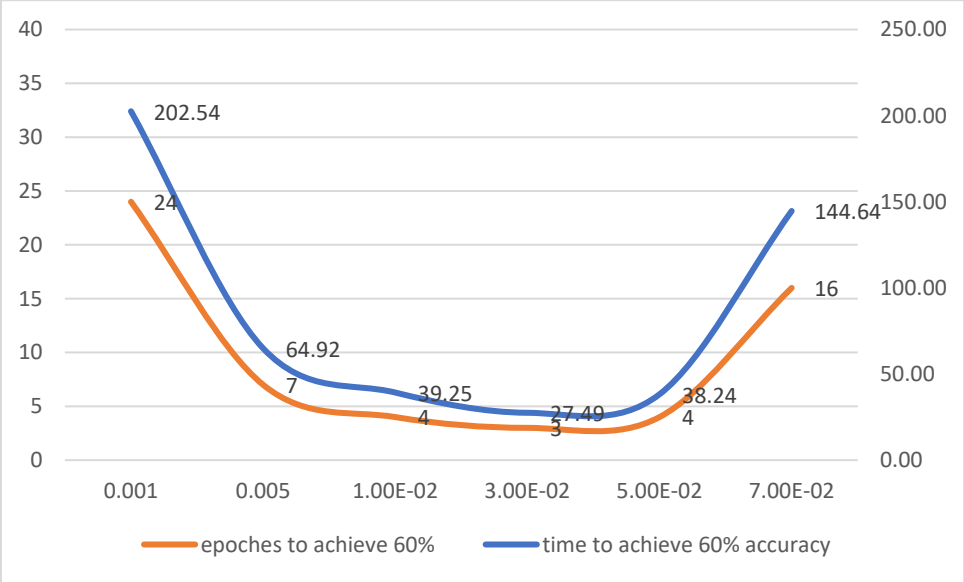

Figure 10 Training speed under different learning rate

Figure 10 shows the effect of different learning rate on training speed.

We can find that the two curves highly coincides, thus different learning rate has little effect on the training time of each epoch. When learning rate is in the interval from 5e-3 to 5e-2, the speed of training is the fastest. When learning rate increases, training speed increases first and then decreases.

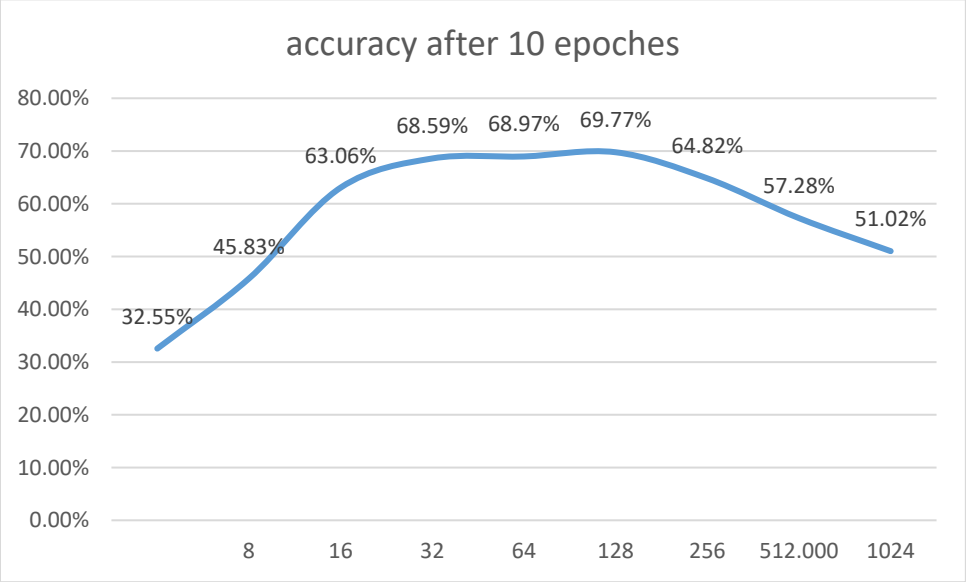

Figure 11 Accuracy after 10 epoch

Figure 11 shows the effect of different learning rate on training accuracy.

With the increase of learning rate, it can be seen from the graph that the training accuracy increases first and then decreases.

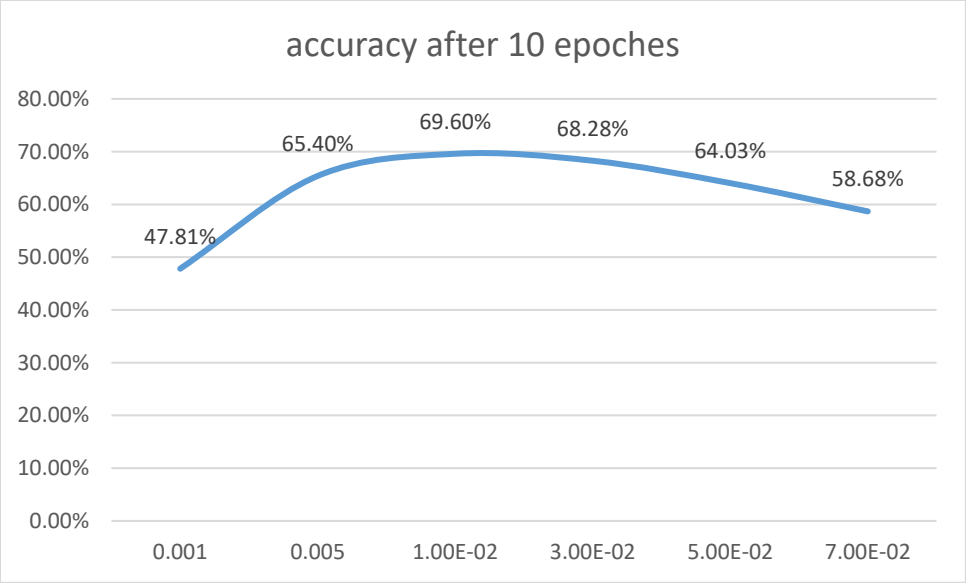

Figure 12 Accuracy after 10 epoch

Figure 12 shows the effect of different batch size on training accuracy.

With the increase of learning rate, it can be seen from the graph that the training accuracy increases first and then decreases.

5.3 Insights

In Table 9 and Table 10, we can get some interesting findings:

- Large batch size reduces training time and improves stability, but if batch size is too large, the accuracy of the model will decrease.

  Without considering BN, batch size determines the time required to complete each epoch and the smoothness of the gradient between each iteration in deep learning training process. Batch size only affects the time it takes to complete

each epoch, not a decision. The root cause is still CPU, GPU performance. If the bottleneck is in the CPU, the bigger the batch size is, sometimes the slower the computation.

If batch size is too small, training speed is small, gradient oscillation is serious and training process is hard to convergence. If the batch size is too large, the gradient of different batches is the same, so it is easy to fall into local minimum value.

- The initial learning rate have an optimal value. Too large leads to the model being divergent. Too small leads to slow convergence.

  The most popular deep learning model is optimized by stochastic gradient descent algorithm. The principle of stochastic gradient descent algorithm is as follows: $w_{t+1} = w_t - \varepsilon\frac{1}{n}\sum_{x\in B}\nabla l(x,w_t)$. N is the batch size and the ϵ is the learning rate. Besides the gradient itself, these two factors directly determine the updating of the weight of the model. From the perspective of model optimization, they are the most important parameters affecting the model convergence.

  Usually, we all need to set a appropriate learning rate. To achieve the minimum value of a convex function, learning rate should satisfy the following conditions:$\sum_{i=1}^{\infty}\epsilon_i = \infty$, $\sum_{i=1}^{\infty}{\epsilon_i}^2 < \infty$. The first formula determines that no matter how far the initial state is from the optimal state, it can always converge. The second formula guarantee the convergence stability. In essence, various

adaptive learning rate algorithms are constantly adjusting the learning rate at all times.

The initial learning rate have an optimal value. Too large leads to the model being divergent. Too small leads to slow convergence. Figure 13 shows the situation of convergence of models under different learning rates.

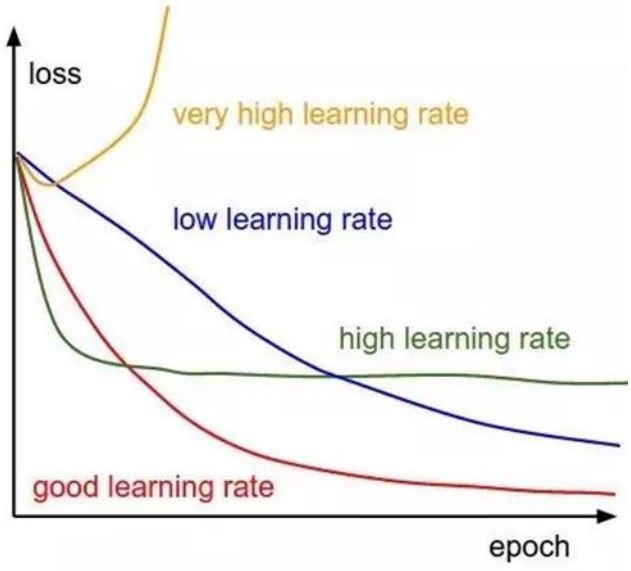

Figure 13 Loss under different learning rate

- In practice, we can only determine the initial learning rate by trying. we can first set the learning rate to 0.01, then observe the trend of training cost. If cost is decreasing, you can gradually increase the learning rate and try 0.1,1.0. If cost is increasing, you have to reduce the learning rate. Try 0.001,0.0001. After several attempts, we can finally determine the appropriate value of learning rate.

- The recommended batch size selection strategy is as follows:

  Consider the GPU memory / CPU memory / computing resource constraints, and find a maximum allowed value. Selected batch size can not be greater than the maximum value.

  Consider the balance of data sets, and try to make batch contain all kinds of samples.

  Observe convergence stability through the loss change curve. Extreme instability can increase batch size or reduce learning rate; if too stable, batch size. can be decreased.

  At the end of training, we can consider increasing batch size and decreasing learning rate gradually, so that the learning result is closer to the local minimum.

Chapter 6 Other factors affecting model performance

Under the same machine learning platform, different tasks. For example, different tasks such as computer vision and natural language processing, the performance of different machine learning frameworks is undoubtedly different.

In addition, half-precision, single-precision, and mixed-precision will undoubtedly have an impact on deep learning performance. It can be expected that without sacrificing model accuracy, a mixed-precision training model has higher performance than a single-precision training model. Generally, half-precision training and inference consume less GPU utilization. For NLP(Nature Language Processing) tasks, it can be proved that deep learning models can be trained with mixed precision without loss of accuracy, while speeding up training. Mixed precision may be An important technique, it can reduce arithmetic operations, thereby reducing the requirements of the GPU.

This report only analyzes a small part of various combinations of software and hardware. Besides, the TensorRT architecture can greatly accelerate the training and inference speed of machine learning. SAP's deep learning team claims that Tesla V100 GPU can further improve the speed of machine learning training. In the future work, more models, frameworks, and evaluation work on machine learning software and hardware have yet to be completed [8].

Chapter 7 Conclusion

In this report, we tested the performance of CIFar 10 classification under three popular machine learning frameworks. The results show that compared to the CPU, the GTX 1060 GPU can bring great growth in model training and inference; in addition, we have also observed the performance of different models in utilizing the DL architecture. Even for different tasks under the same framework, device, Models, as well as data sets and code optimization methods can lead to performance fluctuations in deep learning frameworks. For machine learning developers, when they combine the right machine learning framework, software, hardware, and machine learning tasks, these performances are critical. In this paper, we also discussed the effects of batch size and learning rate on the model and their selection strategies

Bibliography

[1] Wikipedia (2020). Apache_MXnet

https://en.wikipedia.org/wiki/Apache_MXnet

[2] Wikipedia (2020). PyTorch

https://en.wikipedia.org/wiki/PyTorch

[3] Wikipedia (2020). TensorFlow

https://en.wikipedia.org/wiki/TensorFlow

[4]https://lambdalabs.com/blog/best-gpu-tensorflow-2080-ti-vs-v100-vs-titan-v-vs-1080-ti-benchmark/

[5] https://www.nvidia.com/en-us/geforce/products/10series/compare-graphics-cards/

[6] https://mlperf.org/training-results-0-5

[7] Goyal, Priya, et al. "Accurate, large minibatch sgd: Training imagenet in 1 hour." arXiv preprint arXiv:1706.02677 (2017).

[8] https://syncedreview.com/2019/04/23/TensorFlow-PyTorch-or-MXnet-a-comprehensive-evaluation-on-nlp-cv-tasks-with-titan-rtx/#respond