Project 3: Machine Learning Using AWS SageMaker

Zeehan Rahman

Training the MNIST model on local machine:

Firstly, I installed jupyter notebook and pytorch framework on my system. Then I followed a tutorial of training models in local machine using the link: https://towardsdatascience.com/handwritten-digit-mnist-pytorch-977b5338e627

It took 3.70 minutes for job to be completed in my local machine.

Attached is the PYNB file "RunModelLocal" of the steps executed by me to training the model

Training the MNIST model in a distributed fashion:

I used amazon sagemaker to train the model in a distributed fashion. However, I experienced a decent amount of learning curve to get accustomed to the sagemaker and the basics of machine learning.

Firstly, the project description with the intimidating terms like "softmax" and "loss function" motivated me to learn more about machine learning. I believed having a sound understanding of the concept would enable me to execute the task in a more methodical and approachable way.

Reading numerous articles about the topic and professor's recommended website http://rasbt.github.io/mlxtend/user_guide/classifier/SoftmaxRegression/, I was able to decipher some part of the projects. However, I still did not understand terms such as "Activation Functions" or "Back Propagations" or why we use gradient descent to find the ideal value of a given weight where the loss function would be minimum.

To my rescue, StatQuestVideos: **https://www.youtube.com/c/joshstarmer** proved to be significantly helpful. Through the series solely dedicated to machine learning for beginners, I realized there are various types of loss functions, including cross-entropy that uses the output of the SoftMax to calculate the total loss. Terms like argmax "a sort of function that is used to convert the outputs the highest probability (not between 0 and 1 though) for the potential correctly recognized class. I also learnt that SoftMax converts the output of the argmax into probabilistic values (between 0 and 1). More importantly, I learnt that all the loss functions like

cross entropy or squared residuals have one thing in common: there is only one point (say P) where the slope of the tangent to the loss function is equal to 0 (and therefore we use gradient descent that calculates the derivative of these loss functions). The point P corresponds to the lowest value of y (output of the loss function). Therefore, gradient descent purpose is to find that point P where the slope equals to 0, and this is where the output (y value) of the loss function would be at its minimum.

Now, having been frequented with the basics of machine learning, I moved ahead to get accustomed to sagemaker. There was a bit of learning curve as this was my first time using the service on top of the newly acquainted machine learning framework: Pytorch. After extensive research, I found some decent ways to fulfill the task of training the MNIST model in sagemaker.

These were:

pytorch_mnist: https://github.com/aws/amazon-sagemaker-examples/tree/main/sagemaker-python-sdk/pytorch_mnist

During the training of the dataset, an error occurred "Dataset not found or corrupted. You can use download=True to download it". After some research I found a potential solution https://discuss.pytorch.org/t/cant-load-dataset-using-dataloader/34831. However, the potential solution did not work in my case.

```
  File "/opt/conda/lib/python3.6/site-packages/torchvision/datasets/mnist.py", line 83, in __init__
    ' You can use download=True to download it')
RuntimeError: Dataset not found. You can use download=True to download it, exit code: 1
```

Using Horovod: https://sagemaker-examples.readthedocs.io/en/latest/sagemaker-python-sdk/pytorch_horovod_mnist/pytorch_mnist_horovod.html

This way involved using GPUs to train the model. However, when trying to execute estimator.fit() an error occurred "*Resource Limit Exceeded when calling the CreateTrainingJob operation: The account -level service limit 'ml.p2.xlarge'" for training job usage is 0 instances, with current utilization of 0 instances and a request delta of 2 instances. Please contact AWS support to request an increase for this limit*". Therefore, I contacted AWS to increase my limit. However, with the help of professor, I realized that AWS is preventing me to use the instance "ml.p2.xlarge" as it is too powerful (and therefore restricted to me) and unnecessary for such a small dataset.

Therefore, I switched to smaller instance type “ml.c5.2xlarge”, the same one that I tried using in the previous method. However, upon solving the error, I ran into another problem during the training of the model “horovod module not found”. Therefore, I downloaded the horovod module in the beginning of the jupyter notebook by using the line “pip install horovod”. After solving the error, and restarting the kernel, the error persisted. To fix these errors might lead me into a rabbit hole and waste a huge chunk of my time, I switched gears and looked for another way.

Distributed data parallel MNIST training with PyTorch and SageMaker distributed: https://github.com/aws/amazon-sagemaker-examples/blob/main/training/distributed_training/pytorch/data_parallel/mnist/pytorch_smdataparallel_mnist_demo.ipynb

This involved using the smdistributed library to train the model. However, in the file train_pytorch_smdataparallel_mnist.py, line 21 import “smdistributed.dataparallel.torch.distributed as dist” caused an error “No module named smdistributed”.

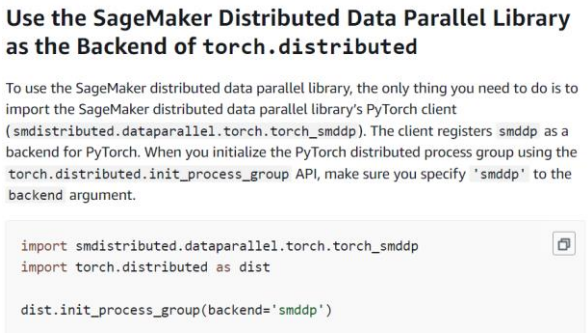
```
    ----

UnexpectedStatusException: Error for Training job pytorch-smdataparallel-mnist-2022-05-20-00-41-58-795: Failed. Reason: A
lgorithmError: ExecuteUserScriptError:
Command "/opt/conda/bin/python3.6 train_pytorch_smdataparallel_mnist.py"
Traceback (most recent call last):
  File "train_pytorch_smdataparallel_mnist.py", line 22, in <module>
    import smdistributed.dataparallel.torch.torch_smddp
ModuleNotFoundError: No module named 'smdistributed', exit code: 1
```

I tried importing the module, but I was not able to find a way to achieve it. With numerous research I found a potential cause to the problem:

> **Important**
>
> Because the SageMaker distributed data parallelism library v1.4.0 and later works as a backend of PyTorch distributed, the following smdistributed APIs for the PyTorch distributed package are deprecated.
>
> - `smdistributed.dataparallel.torch.distributed` is deprecated. Use the torch.distributed package instead.
> - `smdistributed.dataparallel.torch.parallel.DistributedDataParallel` is deprecated. Use the torch.nn.parallel.DistributedDataParallel API instead.

With some revival of the lost hope, I followed the simple instructions to modify the above pyfile using the following instructions:



Use the SageMaker Distributed Data Parallel Library as the Backend of `torch.distributed`

To use the SageMaker distributed data parallel library, the only thing you need to do is to import the SageMaker distributed data parallel library's PyTorch client (`smdistributed.dataparallel.torch.torch_smddp`). The client registers `smddp` as a backend for PyTorch. When you initialize the PyTorch distributed process group using the `torch.distributed.init_process_group` API, make sure you specify `'smddp'` to the `backend` argument.

```
import smdistributed.dataparallel.torch.torch_smddp
import torch.distributed as dist

dist.init_process_group(backend='smddp')
```

Once pass through the error, I stumbled into another familiar error:

```
ResourceLimitExceeded: An error occurred (ResourceLimitExceeded) when calling the CreateTrainingJob operation: The account-l
evel service limit 'ml.p3dn.24xlarge for training job usage' is 0 Instances, with current utilization of 0 Instances and a r
equest delta of 2 Instances. Please contact AWS support to request an increase for this limit.
```

And in this case, my option was limited to using only the 3 instances (all unavailable to me) supported by the smdistributed library:

`smdistributed.dataparallel` supports model training on SageMaker with the following instance types only. For best performance, it is recommended you use an instance type that supports Amazon Elastic Fabric Adapter (ml.p3dn.24xlarge and ml.p4d.24xlarge).

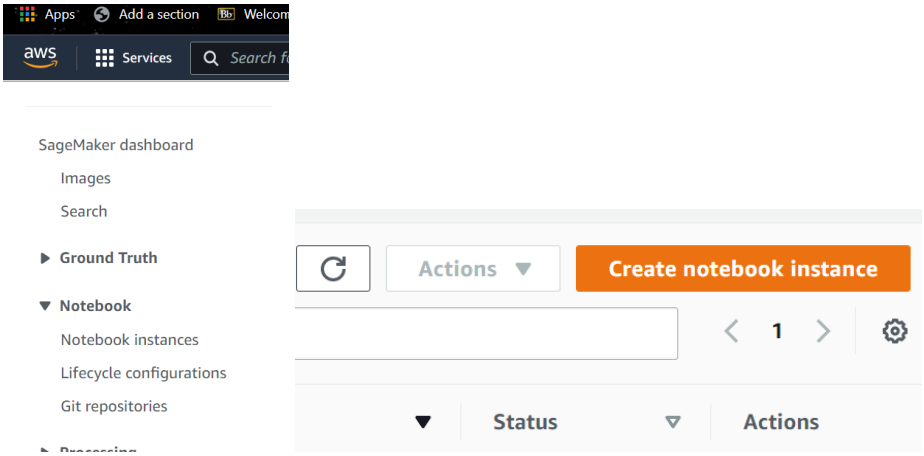
1. ml.p3.16xlarge
2. ml.p3dn.24xlarge [Recommended]
3. ml.p4d.24xlarge [Recommended]

Having gotten back no reply from the AWS team to increase my limit. I tried to think of some other ways.

With some days gone by thinking for a simple fix (thank you the extension of the due date), I decided to set the kernel to amazonneipytorch36 in jupyter notebook instead of using pytorch36. This time the 1st way worked out!!! I realized the error in the 1st way was caused by using an incorrect (specific to this method) environment “pytorch36”.

Here is a brief step of how I finally got to train the model:

1. Created a Jupyter Notebook instance. The notebook instance ran on the VM: ml.t2.medium that is included in the aws free tier



2. Afterwards, I followed the same simple instructions as before with the kernel set as amazonneipytorch36. I ran the model using the instance “ml.m5.large” and setting the instance count to only one.
3. The training was successful with an accuracy of about 90%. However, I had no means to figure out the total utilization of the CPU or identify any potential bottlenecks.

This led me to disregard the result and find a way to track the usage of system resources used in training the model. After some research I got to know about the sagemaker debugger profiling report. A feature that collects all the monitoring and profiling rules and gathers them in a downloadable detailed report.

To activate the profiler report and start collecting records for any potential bottlenecks or over utilization of any resources, I inputted the following syntax right before calling the estimator

```
profiler_config=ProfilerConfig(
    system_monitor_interval_millis=1000,
    framework_profile_params=FrameworkProfile()
)

debugger_hook_config=DebuggerHookConfig()
rules=[
    ProfilerRule.sagemaker(rule_configs.ProfilerReport()),
    ProfilerRule.sagemaker(rule_configs.BatchSize()),
    ProfilerRule.sagemaker(rule_configs.CPUBottleneck()),
    ProfilerRule.sagemaker(rule_configs.OverallSystemUsage()),
    ProfilerRule.sagemaker(rule_configs.LowGPUUtilization()),
    Rule.sagemaker(rule_configs.loss_not_decreasing()),
]
```

And in the estimator, I included the following (in bold):

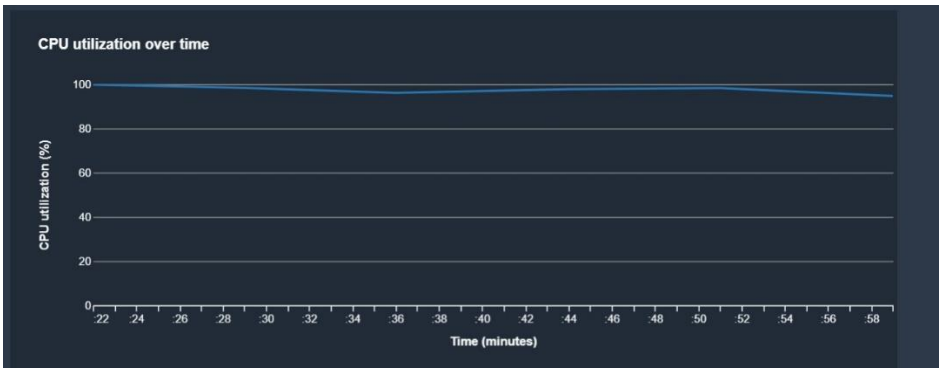
```
estimator=PyTorch(
    entry_point='mnist.py',
    role=role,
    py_version='py36',
    framework_version='1.8.0',
    instance_count=2,
    instance_type='ml.m5.large',
    hyperparameters={
        'epochs': 1,
        'backend': 'gloo'
    },
    # Debugger-specific parameters
    profiler_config=profiler_config,
    debugger_hook_config=debugger_hook_config,
    rules=rules,

)
```

Afterwards, I ran the training again with this time keeping track of system usage reports. However, one of the learning curves was to discover where to check for the reports. After some research I got to know about Sagemaker Studio: one of the ways to view and track system usage resources. For this I created a user in the studio that ran on `ml.m5.4xlarge`**.** After opening the studio app, I followed the simple steps using the link: https://docs.aws.amazon.com/sagemaker/latest/dg/debugger-on-studio-insights.html.

1st attempt:

Here is a graph of the CPU utilization of the instance: that I used to train the model:



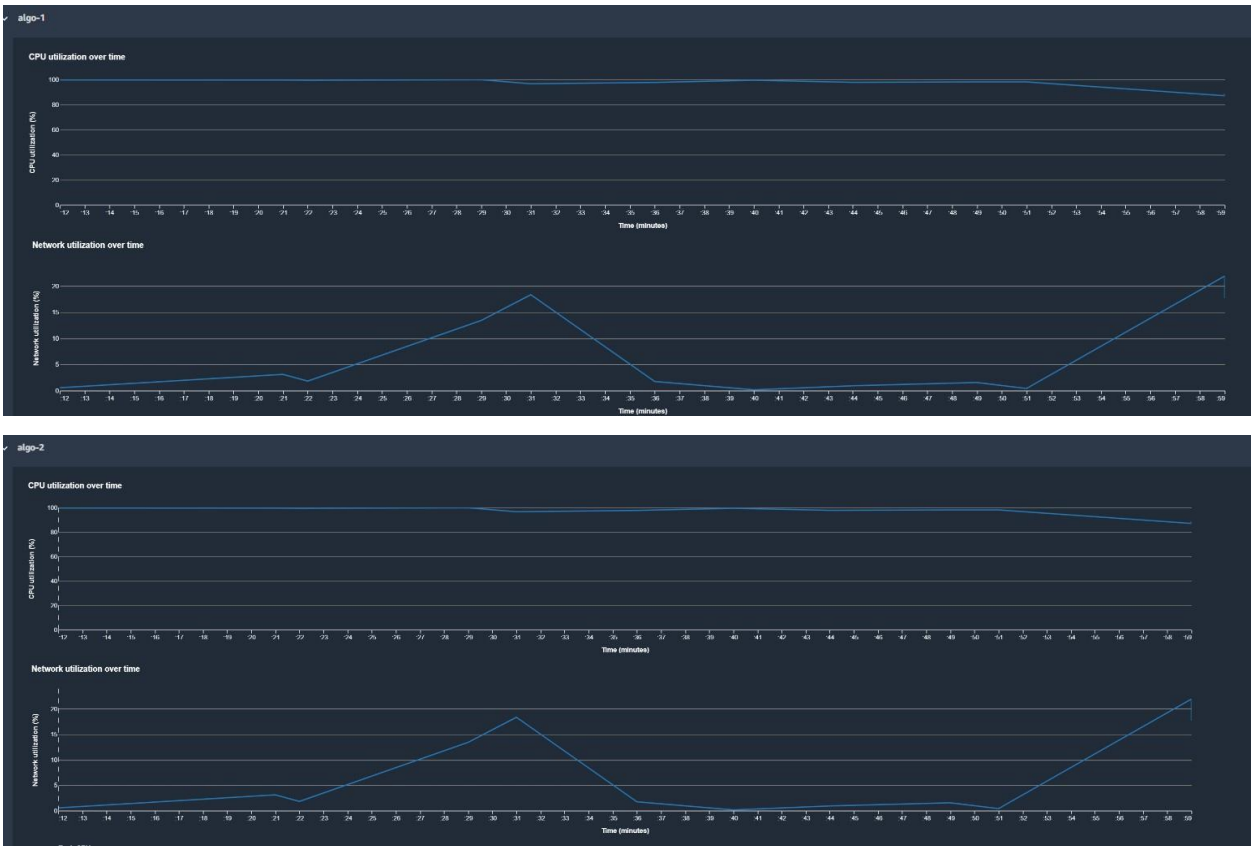
My observations: Training the model with batch size 64 and an instance ml.m5.large (with 2 CPU and 8gb of memory) causes over utilization of the CPU. Also, the total training time is 38 seconds. I comprehended that using more or powerful instance might lower the training time.

My next plan: To mitigate the over utilization of the CPU, I decided to use two instances of the same type and the same batch size. I believed that involving two instances would speed up the job.

The comprehensive report is attached with the name as "1stAttempReport".

2nd attempt:

Here is a graph of the sytem utilization and the network usage of the instances: that I used to train the model:



My observations: As we notice the CPU utilization remains high and the network usage also have some peeks at time = 31, 59. I am almost sure the peak in the network usage graph is

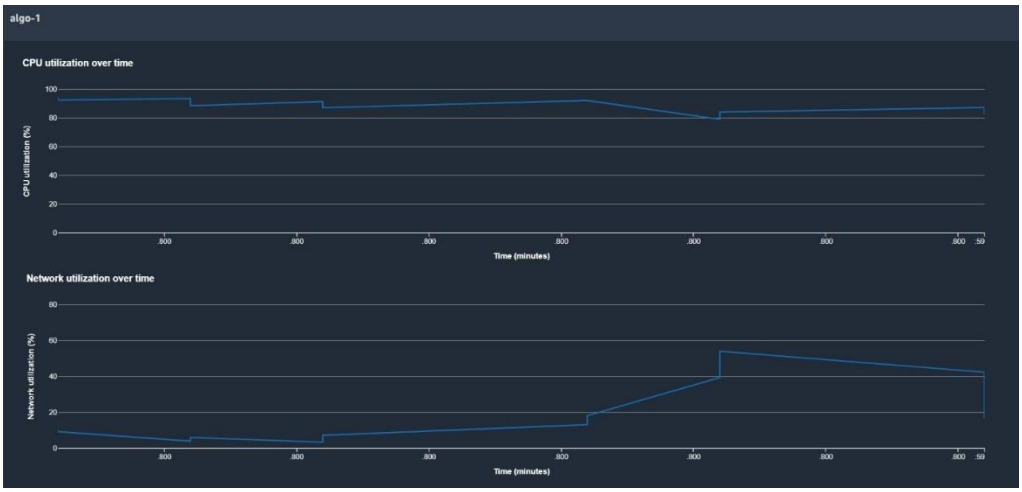
caused by the communications of the instances with the master node at the end of every epoch. Also, according to the profiling report the training time is 48 seconds that is longer than the previous attempt.

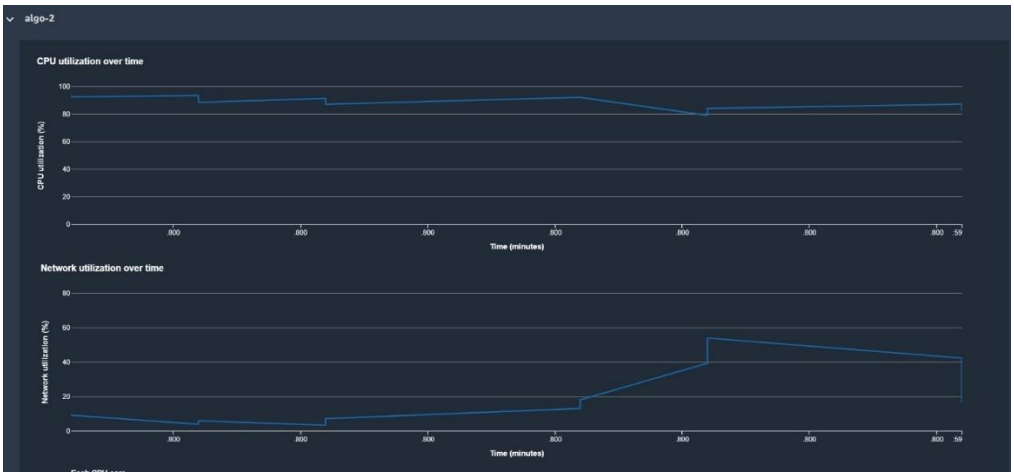
My next plan: As the CPU utilization is unchanged, I decided to use a more powerful instance.

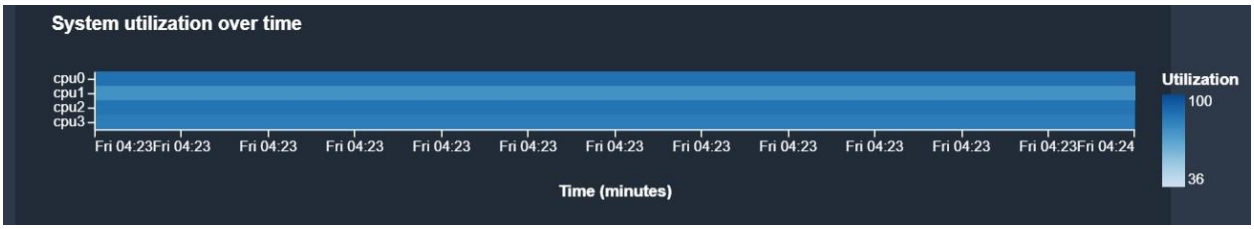
PYNB file and the comprehensive report are attached with the name as "PYNB_Attemp_2" and "1stAttempReport" respectively

3rd attempt:

Here is a graph of the system utilization of the instance: that I used to train the model:







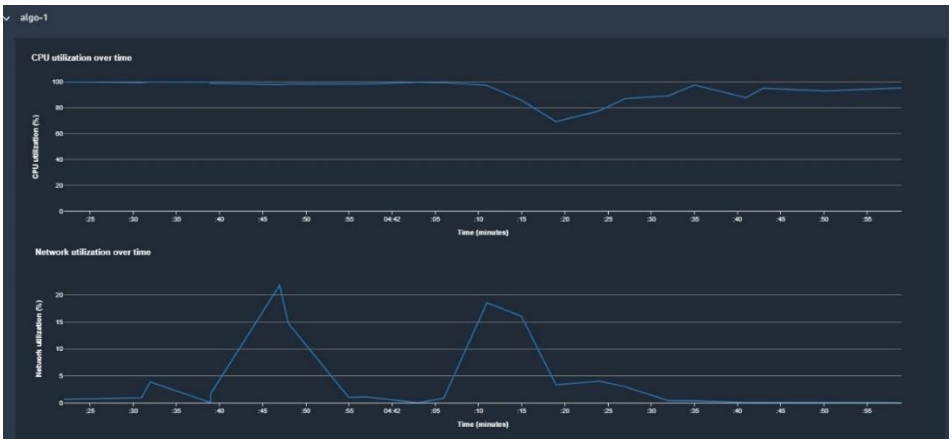
My observations: As we can see, using two instance of type “ml.c5.xlarge” with 4 CPUs and 8gb of memory leads us to significant improvements in the performance. Firstly, the CPU are not totally utilized (remain under 100%). Secondly, the network usage has tolerable peaks. Thirdly, the overall system utilization is impressive. Moreover, the training time has been reduced to an impressive 8 seconds.

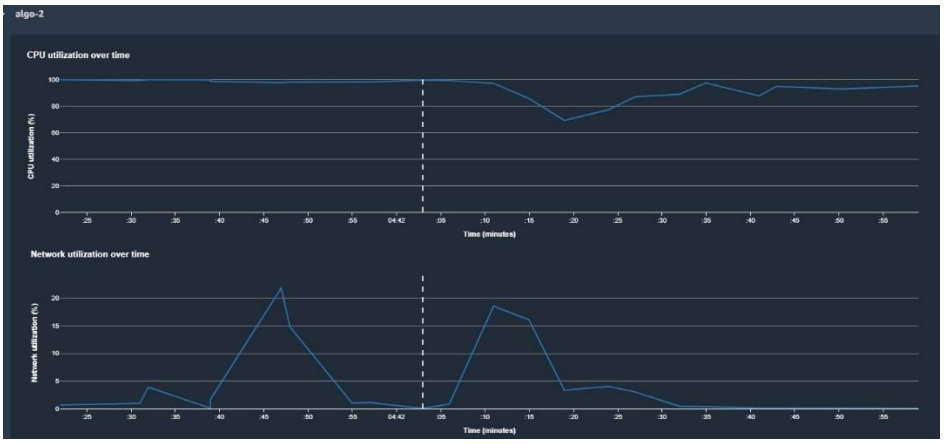
My next plan: Motivated by the greedy algorithm, I seek to improve the training performance even further while also using a cheaper instance as ml.c5.xlarge cost $ 0.204 an hour.

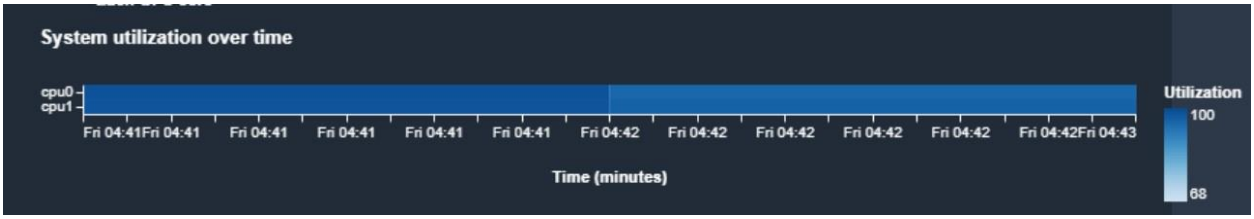
PYNB file and the comprehensive report are attached with the name as “PYNB_Attemp_3” and “3rdAttempReport” respectively

4th attempt:

Here is a graph of the system utilization of the instance: that I used to train the model:







My observations: This time I went back to the same instance “ml.m5.large” as it costs $0.115 an hour (half than ml.c5.xlarge) with instance count set to 2. However, this time I tried playing with the batch size. I reduced the batch size from 64 to 32 hoping that my CPUs will have to do less work (but run longer (more epochs to converge)) at every step, preventing low utilization of the CPUs. I did keep in mind that reducing the batch size to half will also increase in the number of communications between the workers and the master. However, I was eager to see the results.

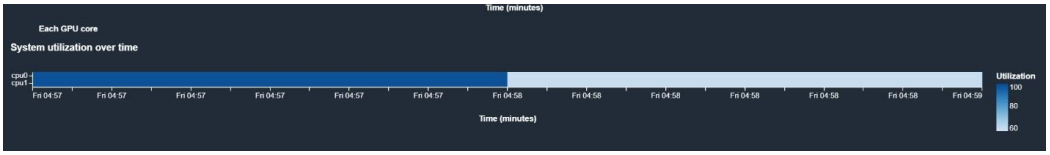
However, there was no improvements as we can see from the graph above. The CPU utilization and the overall system usage resources remain high. Moreover, during communications the network usage are extremely high. On top of that, job duration has increased to 98 seconds.

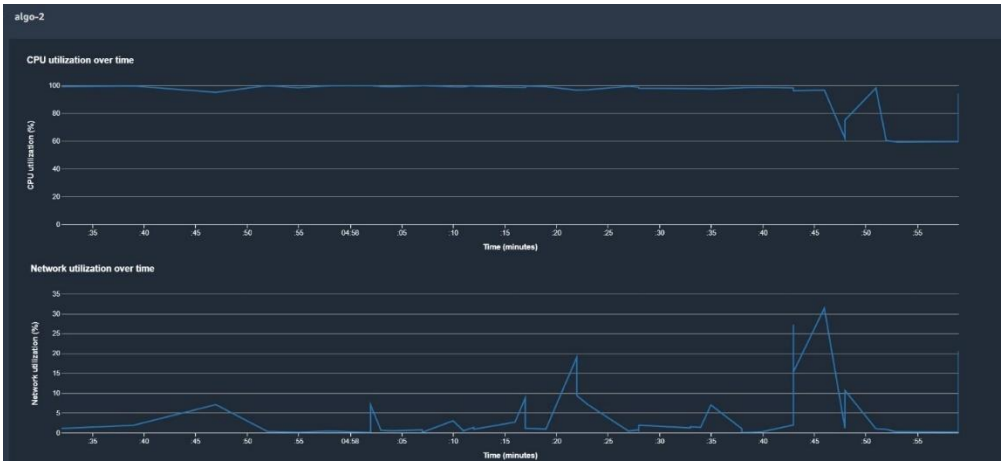
My next plan: Double the instance count (4) with batch size 32

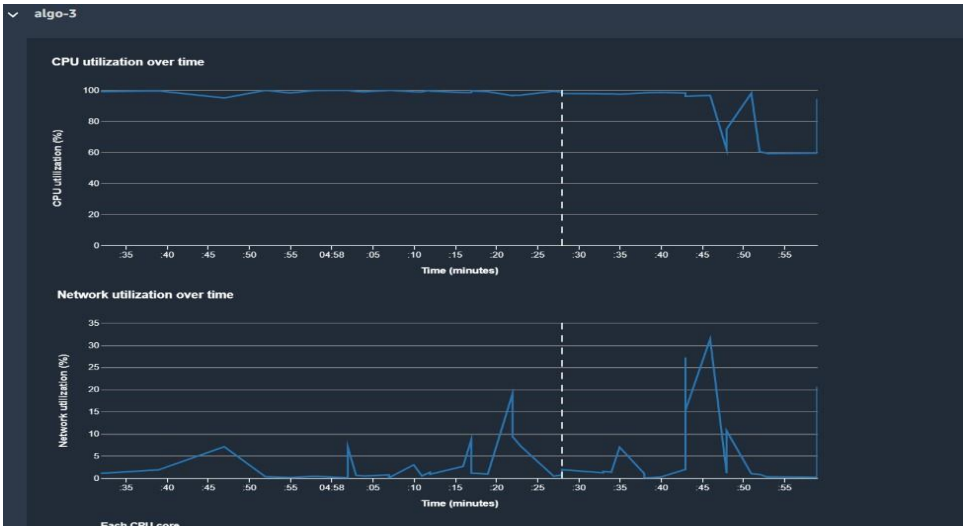
PYNB file and the comprehensive report are attached with the name as "PYNB_Attemp_4" and "4thAttempReport" respectively

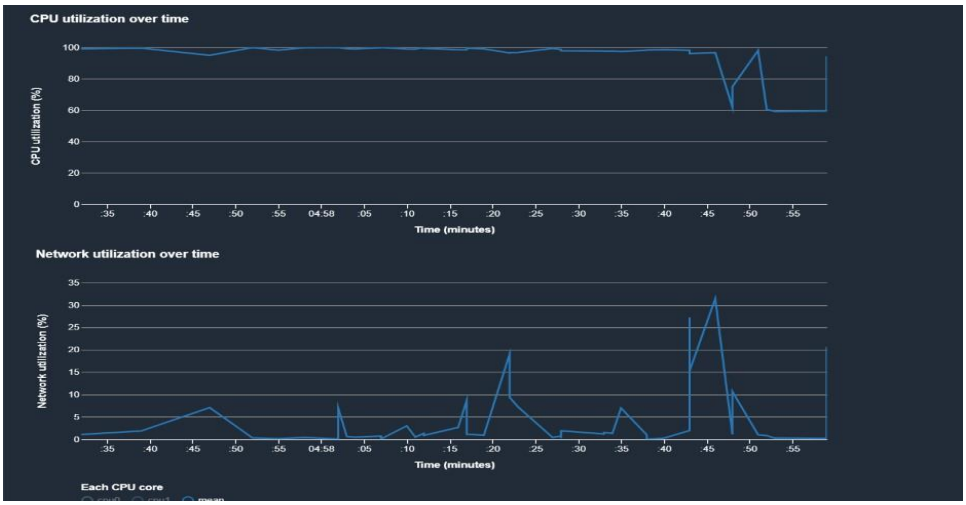
5th attempt:

Here is a graph of the utilization of the instance: that I used to train the model:









My observations: The CPU utilization does not improve. Moreover, due to doubling the instance count, the number of communications increases. This also explains the increase in the number of peaks in the network usage graph.

PYNB file and the comprehensive report are attached with the name as "PYNB_Attemp_5" and "5thAttempReport" respectively

My final observation: I conclude that using a instance similar to ml.m5. large that only has 2 CPUs and 8gb of memory is unlikely to be the best choice in training the MNIST model. Also, using powerful instance (in my case: ml.c5.xlarge) led to significant performance (adequate usage of CPUs, overall system resources and training duration of 8 seconds).

Other References:

Enabling the debugger profling report:

https://docs.aws.amazon.com/sagemaker/latest/dg/debugger-configure-framework-profiling.html

https://docs.aws.amazon.com/sagemaker/latest/dg/debugger-built-in-rules.html#built-in-rules-profiling

https://docs.aws.amazon.com/sagemaker/latest/dg/use-debugger-built-in-rules.html

https://docs.aws.amazon.com/sagemaker/latest/dg/debugger-profiling-report.html

https://docs.aws.amazon.com/sagemaker/latest/dg/debugger-on-studio-insights-controllers.html

https://docs.aws.amazon.com/sagemaker/latest/dg/batch-transform.html

Pricings:

https://aws.amazon.com/sagemaker/pricing/