****

**Improve the Performance of the Machine Learning models using Tensorflow Profiler**

Machine Learning can be thought of the study and design of algorithms which makes use of various statistical and mathematical concepts and models that our modern computer system uses to perform complex and intelligent tasks without any explicit instructions. We can also think machine learning as the study of patterns from huge amount of data which helps us in inferencing.

For the first few years the idea was very simple and very close to what we explained above , however since more and more researches happened in the field of Artificial Intelligence , the complexity , intelligence and effectiveness of the ML models increased exponentially which is also supported by the large amount of data every system generates now days. We can now see the application of machine learning methods in almost all the fields including [computer vision](https://en.wikipedia.org/wiki/Computer_vision), where it is difficult or infeasible to develop a conventional algorithm for effectively performing the task.

Now as the adaptability of machine learning increases it also creates the demand for more effectiveness, less computation and less latency in the inference. That brings one of the most important aspects called “**Real time inferencing** “which requires the study of each and every process with the idea of improvisation. One such effective process or method is called **Profiling** the processes. This technique helps us to understand the effectiveness/ineffectiveness of each of the process running under the hood. The Data scientist now have the job of tuning each process as per the latency need of the system/Business Process.

Let’s relate it to some real-life example to understand the importance of it. Say you are the CEO of a multi-national company. Your Job at some point in time is to evaluate the performance of your first line executives. How will you do that? The effectiveness of their performance, isn’t it? And what will you do when someone is not doing the work as per the expectation? or if some of your executives making it slow? You will find, replace or try a different executive or you may even try to shuffle the responsibilities between the executives. That’s exactly the idea behind profiling each process. This helps us to find the most effective and ineffective process, replace those with better options and achieve the state-of-the-art performance with the same setup and resources or at least try to go as close as possible.

TensorFlow has a built-in profiler that allows you to record runtime of each operations with very little effort. Then you can visualize the profile result in Tensor Board’s Profile Plugin. This helps the user to understand the processes which are taking more time than usual, the processes which are ideal, the processes which are over burdened etc. The Data Scientist can carefully shuffle or parallelize the processes to minimize the time taken by process heavy operations. Let’s jump into an example and see what it exactly means. We won’t run through the complete code here but focus only on the codes which are required for profiler. The user may go through the complete code by visiting <https://www.tensorflow.org/tensorboard/tensorboard_profiling_keras>.

**from tensorflow.python.eager.profiler import Profiler**

The Profiler must be imported to the notebook( If you are not familiar with the eager execution, then please see some tutorials on the same).This exercise can be performed using both GPU and TPU. So please ensure that you select the “Runtime Type” appropriately.

Before we run the exercise it’s important to see that tensorflow can see the GPU/TPU (example shows for gpu)

device\_name = tf.test.gpu\_device\_name()

if not tf.test.is\_gpu\_available():

  raise SystemError('GPU device not found')

print('Found GPU at: {}'.format(device\_name))

This article assumes that you are familiar with building models using the tensorflow high level api Keras or tensor estimators. Now pick any of the model that you have built (if you don’t have one handy then you can use the one mentioned in the link given above). The objective here is to understand each process and operation that the model is performing. Now in order to visualize we need to take the help of “Tensorboard” which is the visualization dashboard of tensorflow. If you are not familiar with Tensorboard then I would suggest you to spend some time going through the link (<https://www.tensorflow.org/tensorboard/>) to get a basic understanding of the same.

Now as we need to visualize the results so let’s tell our program to capture the statistics of the model training in the log directly which then can be visualized using tensor board.The following lines of code will help to do it.

log\_dir="logs/profile/" + datetime.now().strftime("%Y%m%d-%H%M%S")

tensorboard\_callback = tf.keras.callbacks.TensorBoard(log\_dir=log\_dir, histogram\_freq=1, profile\_batch = 3)

The profile batch parameter gives the user the flexibility to define the batch to be monitored. Now as we defined the **tensorboard\_callback** in our notebook our next step is to tell our model to capture the statistics every time we do an operation in training. How do we do that? it’s very simple and exactly similar to the way we interrupt the model for capturing the learning rate, early stopping etc. Yes, you guessed it correct, it’s the callback that we are talking about here.

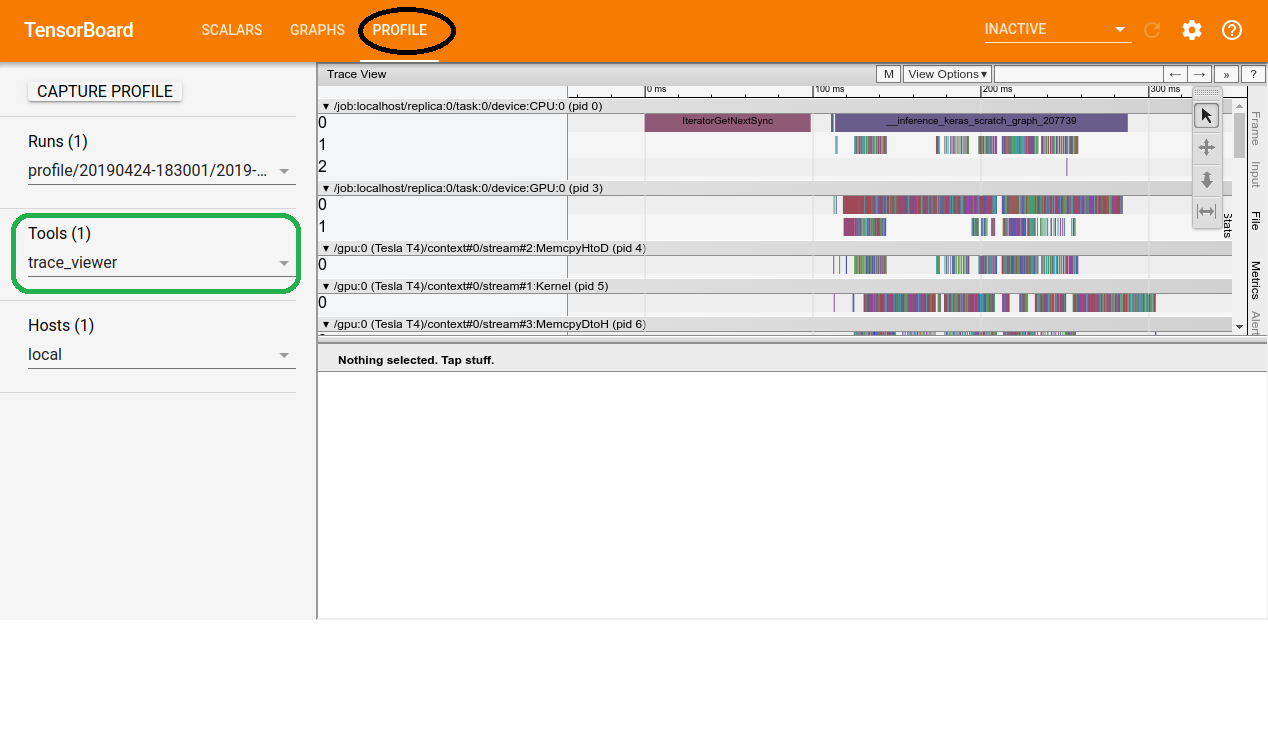
model.fit(train\_data,

          steps\_per\_epoch=20,

          epochs=5,

**callbacks=[tensorboard\_callback])**

Once the model finish training , we opened tensorboard and see how each process behaved during training. For that we need to analyze the “Profile “tab in the tensorboard dashboard.



You must be wondering how to understand this image. The page displays a timeline of different events that happened on the CPU and the accelerator during the collection period. Each event is identified using the process identifier(pid).

The Trace Viewer (It’s a tool which helps us to trace different events and that why we choose it for our visualization) shows multiple event groups on the vertical axis. Each event group has multiple horizontal tracks, filled with trace events. The track is basically an event timeline for events executed on a thread or a GPU stream. Individual events are the colored, rectangular blocks on the timeline tracks. Time moves from left to right.

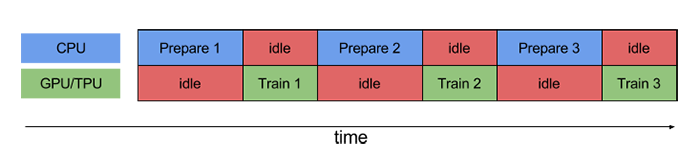
Let’s dig more into each of the trace events and see how they worked. We can do so by selecting single/multiple events and zooming it to the millisecond level.

* **CPU:** CPU events are under event group named */host:CPU*. Each track represents a thread on CPU. E.g. input pipeline events, GPU op scheduling events, CPU ops execution events, etc.
* **GPU:** GPU events are under event groups prefixed by */device:GPU*:. Except *stream:all*, each event group represents one stream on GPU. *stream::all* aggregates all events on one GPU. E.g. Memory copy events, Kernel execution events, etc.
* **TensorFlow Runtime:** Runtime events are under event groups prefixed by */job:.* Runtime events represent the TensorFlow ops invoked by python program. E.g. function execution events, etc.

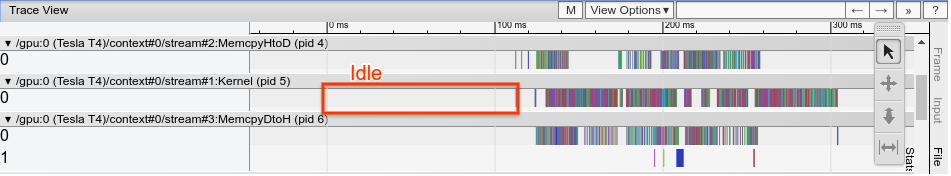
Let’s see the following table and see how CPU and GPU behaves at different stages of training after the notebook allocates the CPU and GPU for the specific task to complete.

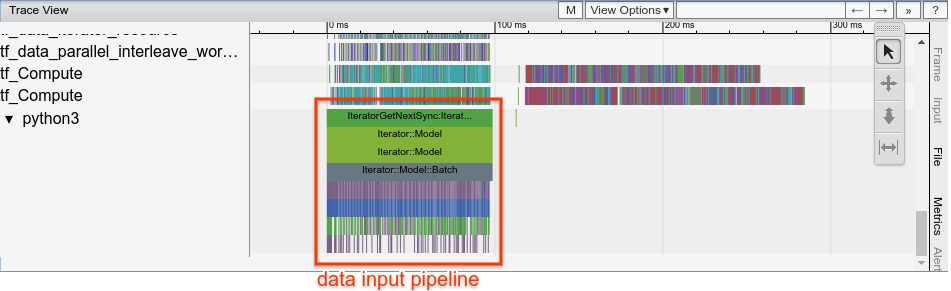
|  |  |
| --- | --- |
| **CPU** | **GPU** |
| Imports all libraries | Nothing |
| Reads the data into memory | Nothing |
| Performs preprocessing work (cutting, shuffling etc..) | Nothing |
| Nothing | Convolution Operation |

while the CPU is preparing the data, the accelerator is sitting idle. Conversely, while the accelerator is training the model, the CPU is sitting idle. The training step time is thus the sum of both CPU pre-processing time and the accelerator training time.



Isn’t it a pure wastage of processing time? Let’s see if the dashboard also confirms our understanding.





The above two diagrams show that the GPU is idle the first half of the time while the CPU is busy and the CPU is idle when GPU processes.

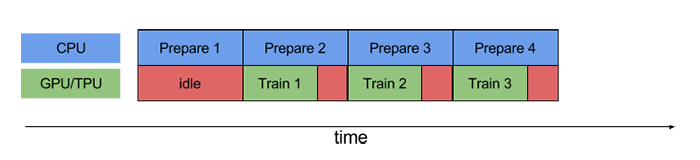
Now we understand the problem, but what’s the solution? Isn’t it a great idea to engage both CPU and GPU at the same time? We know that the model learns by taking input data in batches irrespective of the batch size. So why not the CPU engage in preparing the data for the next batch while the GPU is busy processing the current batch? That way both CPU and GPU can work in parallel and save a lot of time. But the obvious question here is why isn’t it happening now? and who is obstructing this from happening?



In TensorFlow runtime, there is a big block named *Iterator::GetNextSync*, which is a blocking call to get the next batch from data input pipeline and it blocks the training step as well.

To make it happen tensorflow has something called Data Prefetching. It creates a buffer which can fetch the elements from the dataset. Like other Dataset methods, prefetch operates on the elements of the input dataset.

**Pipelining** overlaps the preprocessing and model execution of a training step. While the accelerator is performing training step N, the CPU is preparing the data for step N+1. Doing so reduces the step time to the maximum (as opposed to the sum) of the training and the time it takes to extract and transform the data.



The **tf.data** API provides a software pipelining mechanism through the **tf.data.Dataset.prefetch** transformation, which can be used to decouple the time when data is produced from the time when data is consumed. In particular, the transformation uses a background thread and an internal buffer to prefetch elements from the input dataset ahead of the time they are requested. The number of elements to prefetch should be equal to (or possibly greater than) the number of batches consumed by a single training step. Note that the prefetch transformation provides benefits any time there is an opportunity to overlap the work of a "producer" with the work of a "consumer."

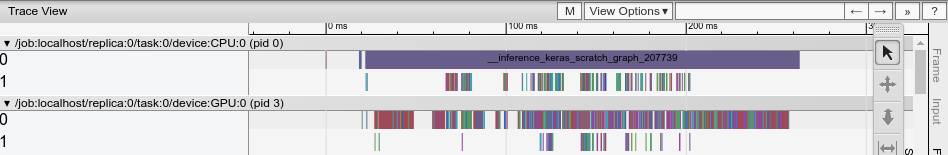
**Prefetch(2)** will prefetch two elements (2 examples)

**examples.batch(20).prefetch(2)** will prefetch 2 elements (2 batches, of 20 examples each).

This is how we can specify the prefetch option for training data

train\_data = train\_data.prefetch(buffer\_size=tf.data.experimental.AUTOTUNE)

The above line of code will prefecth data in the n-1 step.Now let’s run the model again and see if it works.



Did we observe? Now the Iterator::GetNextSync is not seen. That means the data is being prefetched.

There are some additional ways for profiling manually. Those are Profiler API and Profiler Service.

**Profiler API**

# Context manager APIs  
with tf.python.eager.profiler.Profiler('logdir\_path'):  
  # do your training here  
  pass  
  
  
# Function APIs  
tf.python.eager.profiler.start()  
# do your training here  
profiler\_result = tf.python.eager.profiler.stop()  
tf.python.eager.profiler.save('logdir\_path', profiler\_result)

**Profiler Service**

# This API will start a gRPC server with your TensorFlow job which can receive  
# on-demand profiling request.  
tf.python.eager.profiler.start\_profiler\_server(6009)  
  
# Your TensorFlow program here

So, can we assume that now the model performs the best? Even though this model performs better than the before but the improvement is around 1/6th of the overall time per step. The key towards making the process more effective is by creating an efficient input pipeline. The [tf.data](https://www.tensorflow.org/api_docs/python/tf/data) API helps to build flexible and efficient input pipelines.

The Typical Tensorflow pipeline can be seen as an ETL Process.

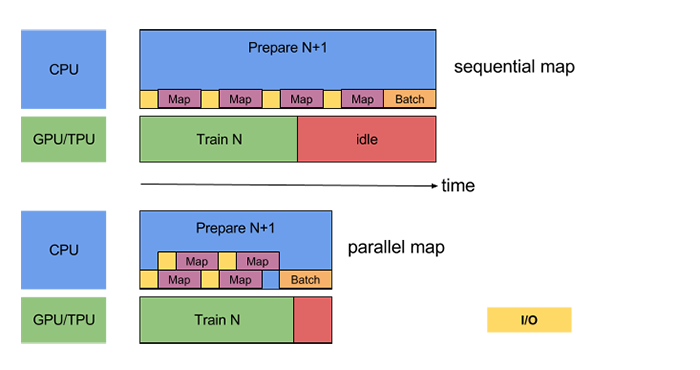
1. **Extract**: Read data from memory (NumPy) or persistent storage -- either local (HDD or SSD) or remote (e.g. [GCS](https://cloud.google.com/storage/) or [HDFS](https://en.wikipedia.org/wiki/Apache_Hadoop#Hadoop_distributed_file_system)).
2. **Transform**: Use CPU to parse and perform preprocessing operations on the data such as shuffling, batching, and domain specific transformations such as image decompression and augmentation, text vectorization, or video temporal sampling.
3. **Load**: Load the transformed data onto the accelerator device(s) (e.g. GPU(s) or TPU(s)) that execute the machine learning model.

In order to process this tensorflow has a file system names as TFRecord files. TFRecord comes very handy when we think about serialize your data over a network and store it in a set of files .The files can now be read linearly. This also helps in caching any data-preprocessing that ultimately helps in the performance. TFRecord format stores data in a sequence of binary records. (Please visit <https://www.tensorflow.org/tutorials/load_data/tfrecord> if you want to know more about how to read/write data into TFRecord).

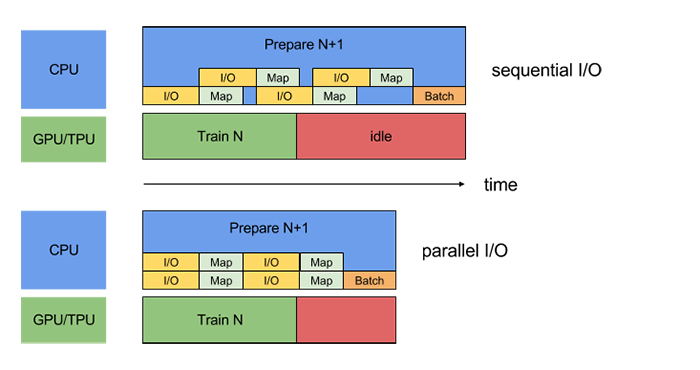
Now once we have the data in TF Record The input pipeline is represented as a [tf.data.Dataset](https://www.tensorflow.org/api_docs/python/tf/data/Dataset) which can passed to high-level TensorFlow API such as [tf.keras](https://www.tensorflow.org/api_docs/python/tf/keras).

Now let’s recap the naïve synchronous implementation. To perform a training step, you must first extract and transform the training data and then feed it to a model running on an accelerator. We already saw this above .

Again when preparing a batch, input elements may need to be pre-processed. To this end, the [tf.data](https://www.tensorflow.org/api_docs/python/tf/data) API offers the [tf.data.Dataset.map](https://www.tensorflow.org/api_docs/python/tf/data/Dataset#map) transformation, which applies a user-defined function (for example, parse\_fn from the running example) to each element of the input dataset which can also be applied to multiple CPU’s.



Data Extraction involves some overhead. For example, Time-to-first-byte and Read throughput from a remote storage, or the encryption and decryption for both local and remote storage. Tensorflow provides [tf.data.Dataset.interleave](https://www.tensorflow.org/api_docs/python/tf/data/Dataset#interleave) which can be used to parallelize the data extraction step. Here is an example of 2 parallel calls.



There are some other Performance considerations present which can have significant impact on the time taken for training the model. Data Scientists are also expected to follow the Best Practices mentioned in the tensorflow documentation for better performance. I would strongly recommend to go through the link to have a much better understanding of different performance measures.( <https://www.tensorflow.org/guide/data_performance>). We are not covering every aspect here as the intention of this article is to give a brief idea about many different methods used for performance improvements. We will leave it to the reader to explore in detail. Tensorflow developer summits also published videos on each of these topics.

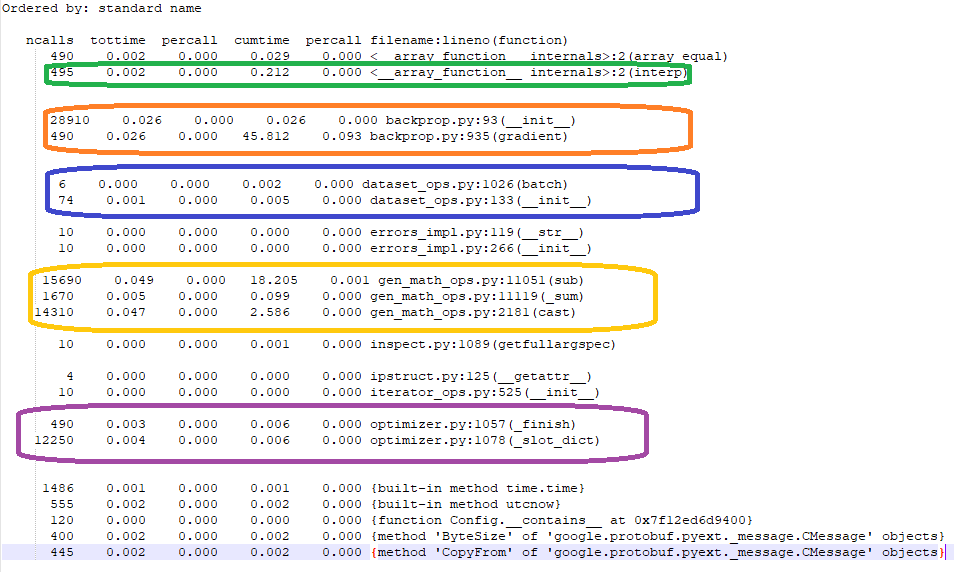
The field of computer vision has improved a lot over the past few years. Lot of new ideas and innovations evolved since 2015 which are really taking this study lot deeper. To summarize every data scientist currently focusing more towards improving the performance of the model in record time. So, as a machine learning enthusiast each one of us should start shifting our focus more towards the same or at least strongly believe that performance in very little time is the key of the hour.

Before we conclude this article, I want to give a very high level view of “CProfile” which is one of the very useful module in python.

This makes us know where the program is spending too much time and what to do inorder to optimize it. It is better to optimize the code inorder to increase the efficiency of a program. So, perform some standard tests to ensure optimization and we can improve the program inorder to increase the efficiency. Python includes a built in module called cProfile which is used to measure the execution time of a program.cProfiler module provides all information about how long the program is executing and how many times the function get called in a program.

I tried to implement Cprofile in one of my tensorflow code. Before I show you the output of the Cprofile let me give you a brief about the program which I tried to execute .The Program is a image classification program which takes cifar10 dataset and performs many convolution function, Batch Normalization, Image normalization etc. As stated above Cprofile prints each and every process , however for readability I will put few lines of the output.

Let me show you how the output of Cprofile looks like.



The above log shows that every function call, every iteration and every mathematical operation is being captured for displaying the amount of time it takes. This is very much helpful for the Data Scientists for performance tuning.