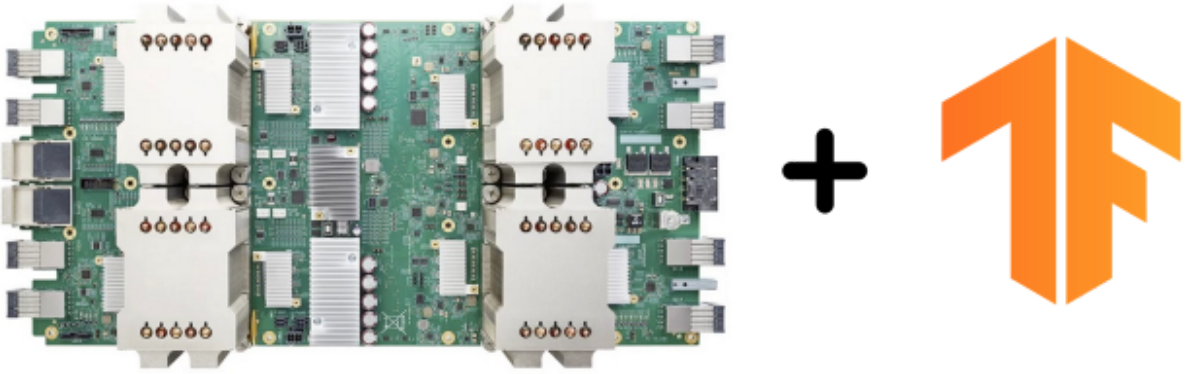
Training Your Models on Cloud TPUs in 4 Easy Steps on Google Colab

I trained an Neural Machine Translation(NMT) model on a TPU and now feel like a Wizard…

Rishabh Anand Follow
Jul 22, 2019 · 8 min read



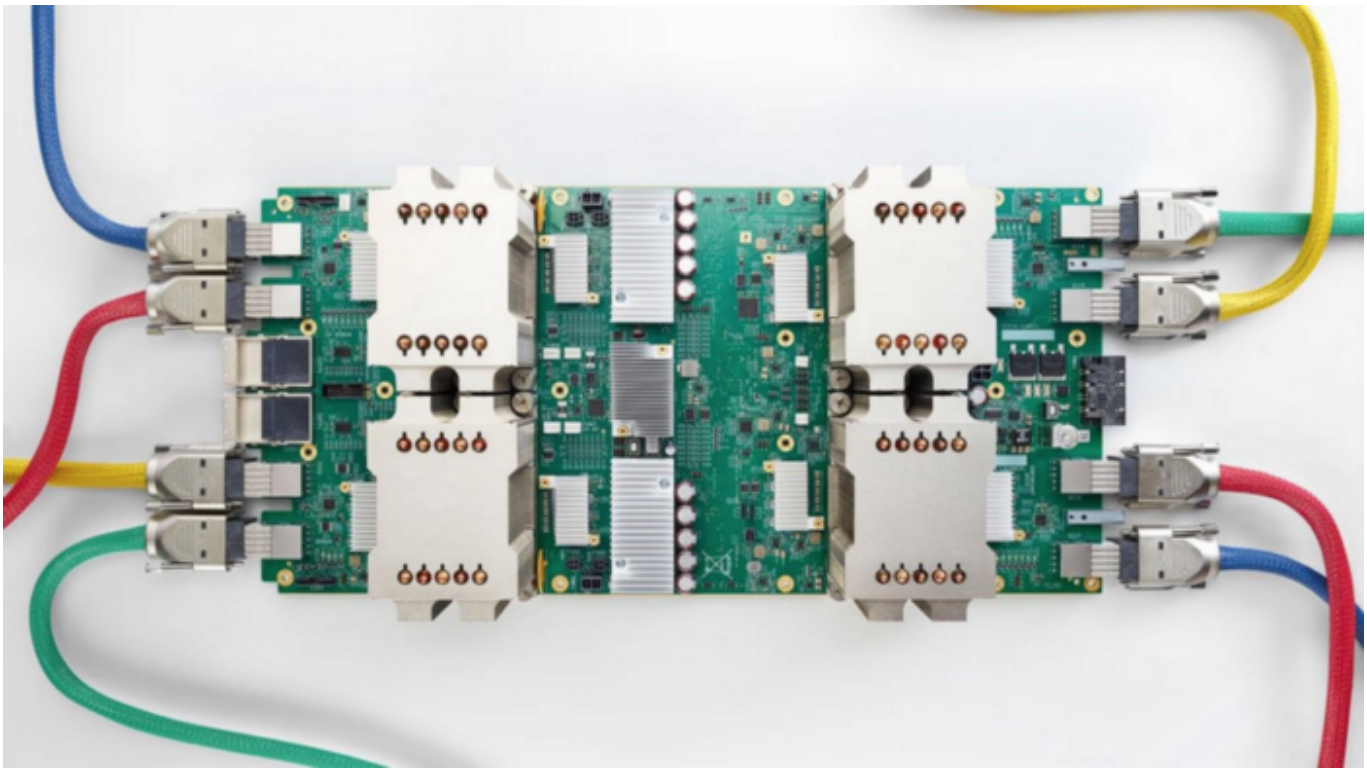
You have a plain old TensorFlow model that's too computationally expensive to train on your standard-issue work laptop. I get it. I've been there too, and if I'm being honest, seeing my laptop crash twice in a row after trying to train a model on it is painful to watch.

In this article, I'll be breaking down the steps on how to train any model on a TPU in the cloud using Google Colab. After this, you'll never want to touch your clunky CPU ever again, believe me.

TL;DR: *This article shows you how easy it is to train any TensorFlow model on a TPU with very few changes to your code.*

What's a TPU?

The Tensor Processing Unit (TPU) is an accelerator — custom-made by Google Brain hardware engineers — that specializes in training deep and computationally expensive ML models.



This is what a TPU looks like. It has 4 humongous heat sinks that sit on top of a board with some insanely advanced and powerful circuitry.

Let's put things into perspective just to give you an idea of how awesome and powerful a TPU is. A standard MacBook Pro Intel CPU can perform a few operations per clock cycle. A standard off-the-shelf GPU can perform tens of thousands of operations per cycle. A state-of-the-art TPU can perform hundreds of thousands of operations per cycle (sometimes up to 128K OPS).

To understand the scale, imagine using these devices to print a book. A CPU can print character-by-character. A GPU can print a few words at a time. A TPU? Well, it can print a whole page at a time. That's some amazing speed and power that we now have at our disposal; shoutout to Google for giving everyone access to high-performance hardware.

If you're interested in the inner workings of a TPU and what makes it so amazing, go check out the Google Cloud blog article where they discuss everything from hardware to software here. You can find information regarding mechanisms, hardware specs, benefits, limits, constraints, and much more to help you with your projects!

Preprocessing MNIST

For the sake of this tutorial, I'll be running through a quick and easy MNIST model. Do note that this model can be whatever you want it to be. To better help you visualize what's going on, I've chosen good old MNIST (again, a dataset of your choosing).

Let's first begin by extracting and preprocessing our dataset. This shouldn't be much of a problem:

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# This can be any dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape([x_train.shape[0], 784])
x_test = x_test.reshape([x_test.shape[0], 784])
x_train = x_train / 255
x_test = x_test / 255

y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

print (x_train.shape, y_train.shape)
print (x_test.shape, y_test.shape)

# >> (60000, 784), (60000, 10)
# >> (10000, 784), (10000, 10)
```

I've tried to make this tutorial as comprehensive as possible so that you can train your models at unfathomable speeds and feel like you're on top of the world. There are 4 steps we will be taking to train our model on a TPU:

1. Connect to an available TPU instance
2. Initialize a distributed training strategy
3. Build our model under the said strategy
4. Train the model and feel like a *Wizard*

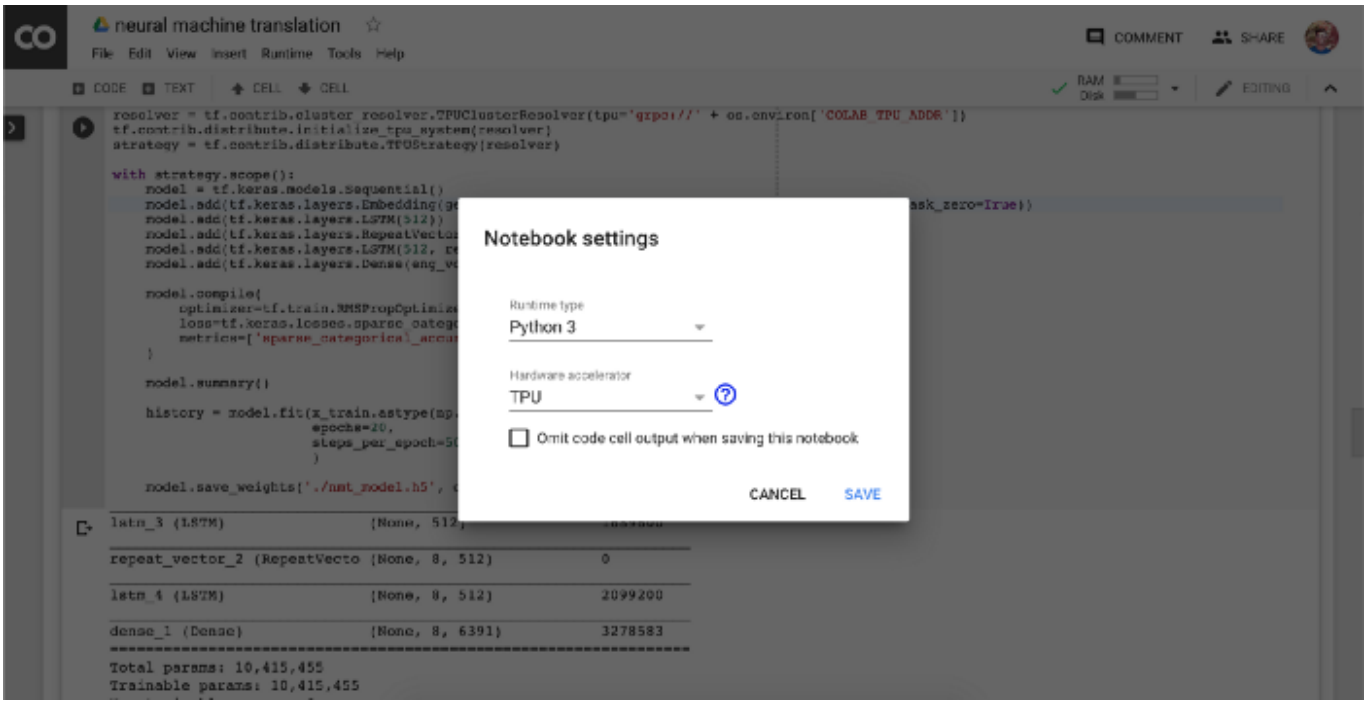
Let's dive right into it!

***Note:** Feel free to preprocess your datasets using* `tf.data` *or convert them into* `TFRecords` *if that catches your fancy. These steps are not necessary but may be useful in handling out-of-memory instances in the case that you are working with very large datasets.*

Connecting to a TPU

When I was messing around with TPUs on Colab, connecting to one was the most tedious. It took quite a few hours of searching online and looking through tutorials, but I was finally able to get it done.

We first need to connect our Colab notebook running locally to the TPU runtime. To change runtime, simply click the `Runtime` tab in the navigation bar. A dropdown menu will be displayed from which you can select the `Change runtime type` option. A popup window will show up where you can select the `TPU` option from the dropdown selector.



The popup window will look something like this. Switch the runtime from CPU to TPU.

To connect to a TPU instance, we need to create a `TPUClusterResolver` that takes in the available device and provisions one for us:

This should connect our current Colab session to an available TPU instance. Let's move on to initializing the strategy and finally invoking the TPU.

Note: *If you are experiencing difficulties in connecting to a TPU instance even after several attempts and re-runs, you can change the runtime type to* `GPU` *instead. The code still compiles and runs efficiently.*

Initializing a Distributed Training Strategy

Now that we've changed the runtime and have acquired a list of available TPUs, we need to invoke the TPU by creating a distributed training strategy — a wrapper around the model that makes it compatible with multi-core training.

The `resolver` gives us access to the TPU such that we can finally build a parallelly-distributed pipeline on it. This is a necessary step because a TPU is a distributed training processor and is not single-core like a traditional CPU. With this strategy method, jumping on board a TPU is very simple! It should give you a list of available instances:

```
Running on TPU  ['10.5.99.242:8470']
Number of accelerators:  8
```

We have 8 devices at our disposal

Building our model under the distributed training strategy

With a training strategy now at hand, we can proceed to build our model using that strategy as such:

This looks like a lot of jargon, but all it does is take a regular `tf.keras` model (that is typically run on a CPU) and places it on the TPU and automatically distributes it across all the available TPU cores.

Training our model

This is the exciting part of the process. We can finally train our model on a Cloud TPU for free. With so much power in our hands, let's make good use of it and train on MNIST (I know…very anti-climatic).

Ensure that the number of instances is perfectly divisible by the `steps_per_epoch` parameter so that all the instances are used during training. For example, we have 60000 instances in our training set. 60000 is divisible by 50 so that means all our instances are fed into the model without any *leftovers*. If you hit run, it should start training on a TPU instance after a short while:

```
Epoch 1/20
50/50 [==============================] - 7s 142ms/step - loss: 4.0048 - sparse_categorical_accuracy: 0.9133
Epoch 2/20
50/50 [==============================] - 1s 19ms/step - loss: 0.3821 - sparse_categorical_accuracy: 0.9320
Epoch 3/20
50/50 [==============================] - 1s 19ms/step - loss: 0.1847 - sparse_categorical_accuracy: 0.9320
Epoch 4/20
50/50 [==============================] - 1s 19ms/step - loss: 0.1447 - sparse_categorical_accuracy: 0.9429
Epoch 5/20
50/50 [==============================] - 1s 19ms/step - loss: 0.1057 - sparse_categorical_accuracy: 0.9593
Epoch 6/20
50/50 [==============================] - 1s 19ms/step - loss: 0.0993 - sparse_categorical_accuracy: 0.9612
Epoch 7/20
50/50 [==============================] - 1s 19ms/step - loss: 0.1015 - sparse_categorical_accuracy: 0.9621
```

```
Epoch 8/20
50/50 [==============================] - 1s 19ms/step - loss: 0.1019 - sparse_categorical_accuracy: 0.9624
Epoch 9/20
50/50 [==============================] - 1s 19ms/step - loss: 0.1036 - sparse_categorical_accuracy: 0.9629
Epoch 10/20
50/50 [==============================] - 1s 18ms/step - loss: 0.0897 - sparse_categorical_accuracy: 0.9632
Epoch 11/20
50/50 [==============================] - 1s 19ms/step - loss: 0.0903 - sparse_categorical_accuracy: 0.9636
Epoch 12/20
50/50 [==============================] - 1s 19ms/step - loss: 0.0888 - sparse_categorical_accuracy: 0.9639
Epoch 13/20
50/50 [==============================] - 1s 19ms/step - loss: 0.0835 - sparse_categorical_accuracy: 0.9639
Epoch 14/20
50/50 [==============================] - 1s 18ms/step - loss: 0.0882 - sparse_categorical_accuracy: 0.9641
Epoch 15/20
50/50 [==============================] - 1s 19ms/step - loss: 0.0889 - sparse_categorical_accuracy: 0.9645
Epoch 16/20
```

Here's the TPU training cycle for the Neural Machine Translation model I built on Colab.

A final round-up

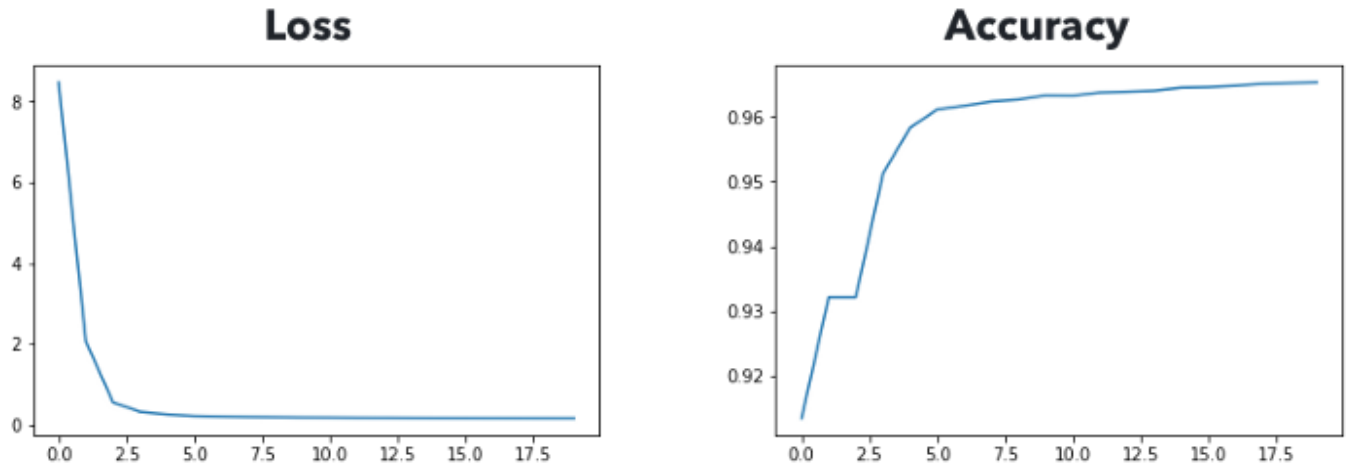
With that, training should commence soon. Colab will boot up a TPU and upload the model architecture on it. You should soon see the classic Keras progress bar style layout in the terminal output. Congrats! You've successfully just used a TPU.

Retrospectively, I always thought training models on TPUs were a thing only ML Wizards — those with years of experience and skill — could handle. TPU training was always that one huge thing out of my grasp simply because of the hype around it (making it seem very difficult to use). Never in my wildest dreams did I think it'd be as simple as adding less than 10 lines of code to my preexisting models.

After reading this article, I want you to know that anyone can train on accelerated hardware regardless of experience. This is just one of many steps undertaken by Google AI to democratize Machine Learning and Artificial Intelligence amongst the masses.

Some observations about TPU training

The Neural Machine Translation model I had written (for the TensorFlow Docs) took me less than a minute to train on a TPU. I was astonished because the same model took more than 4 hours to train on a CPU (which probably explains why my laptop crashed and ran out of memory twice).



My NMT model trained superbly on a TPU! The metrics show that the training was smooth and I didn't encounter any speedbumps along the way.

My model hit a very high accuracy — higher than what I achieved training on a CPU. With performance and speed, the Cloud TPU is second to none when it comes to quality training!

Note: *You can find my code here. Have fun!*

In a nutshell

Google always comes bearing gifts when it launches new Machine Learning toys that we can tinker and play around with. The TPU processor is certainly a boon to the ML community as it plays a major role in the democratization of AI — it gives everyone a chance to use accelerated hardware regardless of demographic. With TPUs, research and experimentation are hitting all-time highs and people are now engaging in Machine Learning like never before!

I hope this article has helped you train your models in a much more efficient and elegant manner. If you have any questions about the use of TPUs or want to chat in general about Tech and ML, feel free drop them down below in the comments or catch me on Twitter or LinkedIn. I usually reply within a day.

Until then, I'll catch you in the next one!

Article by

Rishabh Anand