- Reading inputs efficiently using readers, queue runners, and coordinators

Now let's use all of this to parallelize neural networks!

Parallelizing Neural Networks on a TensorFlow Cluster

In this section, first we will look at how to parallelize several neural networks by simply placing each one on a different device. Then we will look at the much trickier problem of training a single neural network across multiple devices and servers.

One Neural Network per Device

The most trivial way to train and run neural networks on a TensorFlow cluster is to take the exact same code you would use for a single device on a single machine, and specify the master server's address when creating the session. That's it—you're done! Your code will be running on the server's default device. You can change the device that will run your graph simply by putting your code's construction phase within a device block.

By running several client sessions in parallel (in different threads or different processes), connecting them to different servers, and configuring them to use different devices, you can quite easily train or run many neural networks in parallel, across all devices and all machines in your cluster (see Figure 12-11). The speedup is almost linear.[4] Training 100 neural networks across 50 servers with 2 GPUs each will not take much longer than training just 1 neural network on 1 GPU.

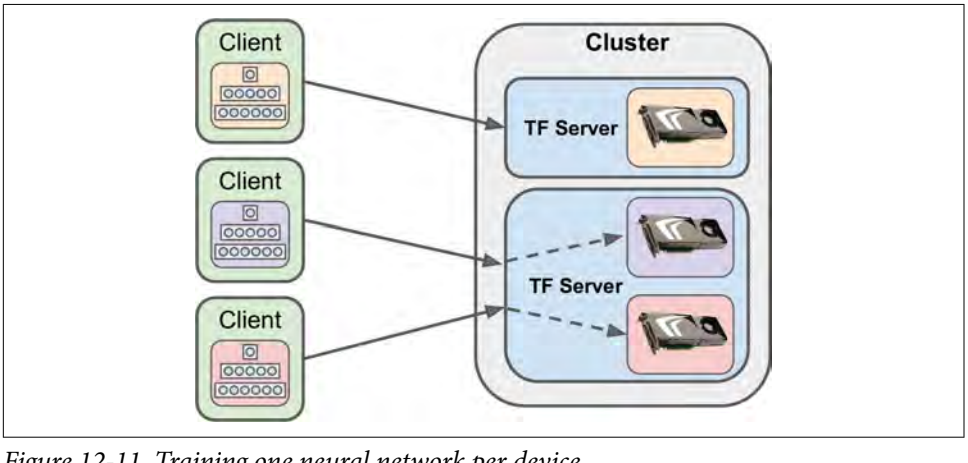

Figure 12-11. Training one neural network per device

4 Not 100% linear if you wait for all devices to finish, since the total time will be the time taken by the slowest device.

This solution is perfect for hyperparameter tuning: each device in the cluster will train a different model with its own set of hyperparameters. The more computing power you have, the larger the hyperparameter space you can explore.

It also works perfectly if you host a web service that receives a large number of *queries per second* (QPS) and you need your neural network to make a prediction for each query. Simply replicate the neural network across all devices on the cluster and dispatch queries across all devices. By adding more servers you can handle an unlimited number of QPS (however, this will not reduce the time it takes to process a single request since it will still have to wait for a neural network to make a prediction).

Another option is to serve your neural networks using *TensorFlow Serving*. It is an open source system, released by Google in February 2016, designed to serve a high volume of queries to Machine Learning models (typically built with TensorFlow). It handles model versioning, so you can easily deploy a new version of your network to production, or experiment with various algorithms without interrupting your service, and it can sustain a heavy load by adding more servers. For more details, check out *https://tensorflow.github.io/serving/*.

In-Graph Versus Between-Graph Replication

You can also parallelize the training of a large ensemble of neural networks by simply placing every neural network on a different device (ensembles were introduced in Chapter 7). However, once you want to *run* the ensemble, you will need to aggregate the individual predictions made by each neural network to produce the ensemble's prediction, and this requires a bit of coordination.

There are two major approaches to handling a neural network ensemble (or any other graph that contains large chunks of independent computations):

- You can create one big graph, containing every neural network, each pinned to a different device, plus the computations needed to aggregate the individual predictions from all the neural networks (see Figure 12-12). Then you just create one session to any server in the cluster and let it take care of everything (including waiting for all individual predictions to be available before aggregating them). This approach is called *in-graph replication*.