even months for training to complete. In the next chapter we will discuss how to use distributed TensorFlow to train and run models across many servers and GPUs.

Exercises

1. Is it okay to initialize all the weights to the same value as long as that value is selected randomly using He initialization?
2. Is it okay to initialize the bias terms to 0?
3. Name three advantages of the ELU activation function over ReLU.
4. In which cases would you want to use each of the following activation functions: ELU, leaky ReLU (and its variants), ReLU, tanh, logistic, and softmax?
5. What may happen if you set the `momentum` hyperparameter too close to 1 (e.g., 0.99999) when using a `MomentumOptimizer`?
6. Name three ways you can produce a sparse model.
7. Does dropout slow down training? Does it slow down inference (i.e., making predictions on new instances)?
8. Deep Learning.
    a. Build a DNN with five hidden layers of 100 neurons each, He initialization, and the ELU activation function.
    b. Using Adam optimization and early stopping, try training it on MNIST but only on digits 0 to 4, as we will use transfer learning for digits 5 to 9 in the next exercise. You will need a softmax output layer with five neurons, and as always make sure to save checkpoints at regular intervals and save the final model so you can reuse it later.
    c. Tune the hyperparameters using cross-validation and see what precision you can achieve.
    d. Now try adding Batch Normalization and compare the learning curves: is it converging faster than before? Does it produce a better model?
    e. Is the model overfitting the training set? Try adding dropout to every layer and try again. Does it help?
9. Transfer learning.
    a. Create a new DNN that reuses all the pretrained hidden layers of the previous model, freezes them, and replaces the softmax output layer with a fresh new one.
    b. Train this new DNN on digits 5 to 9, using only 100 images per digit, and time how long it takes. Despite this small number of examples, can you achieve high precision?

   c. Try caching the frozen layers, and train the model again: how much faster is it now?

   d. Try again reusing just four hidden layers instead of five. Can you achieve a higher precision?

   e. Now unfreeze the top two hidden layers and continue training: can you get the model to perform even better?

10. Pretraining on an auxiliary task.

    a. In this exercise you will build a DNN that compares two MNIST digit images and predicts whether they represent the same digit or not. Then you will reuse the lower layers of this network to train an MNIST classifier using very little training data. Start by building two DNNs (let's call them DNN A and B), both similar to the one you built earlier but without the output layer: each DNN should have five hidden layers of 100 neurons each, He initialization, and ELU activation. Next, add a single output layer on top of both DNNs. You should use TensorFlow's `concat()` function with `axis=1` to concatenate the outputs of both DNNs along the horizontal axis, then feed the result to the output layer. This output layer should contain a single neuron using the logistic activation function.

    b. Split the MNIST training set in two sets: split #1 should containing 55,000 images, and split #2 should contain contain 5,000 images. Create a function that generates a training batch where each instance is a pair of MNIST images picked from split #1. Half of the training instances should be pairs of images that belong to the same class, while the other half should be images from different classes. For each pair, the training label should be 0 if the images are from the same class, or 1 if they are from different classes.

    c. Train the DNN on this training set. For each image pair, you can simultaneously feed the first image to DNN A and the second image to DNN B. The whole network will gradually learn to tell whether two images belong to the same class or not.

    d. Now create a new DNN by reusing and freezing the hidden layers of DNN A and adding a softmax output layer on with 10 neurons. Train this network on split #2 and see if you can achieve high performance despite having only 500 images per class.

Solutions to these exercises are available in Appendix A.

CHAPTER 12
Distributing TensorFlow Across Devices and Servers

In Chapter 11 we discussed several techniques that can considerably speed up training: better weight initialization, Batch Normalization, sophisticated optimizers, and so on. However, even with all of these techniques, training a large neural network on a single machine with a single CPU can take days or even weeks.

In this chapter we will see how to use TensorFlow to distribute computations across multiple devices (CPUs and GPUs) and run them in parallel (see Figure 12-1). First we will distribute computations across multiple devices on just one machine, then on multiple devices across multiple machines.

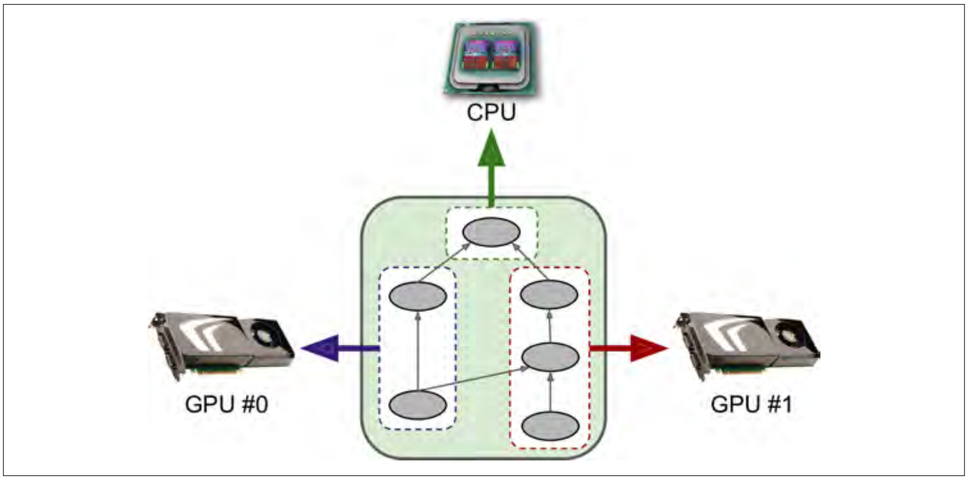

Figure 12-1. Executing a TensorFlow graph across multiple devices in parallel