Exercises

1. Draw an ANN using the original artificial neurons (like the ones in Figure 10-3) that computes $A \oplus B$ (where $\oplus$ represents the XOR operation). Hint: $A \oplus B = (A \wedge \neg B) \vee (\neg A \wedge B)$.
2. Why is it generally preferable to use a Logistic Regression classifier rather than a classical Perceptron (i.e., a single layer of linear threshold units trained using the Perceptron training algorithm)? How can you tweak a Perceptron to make it equivalent to a Logistic Regression classifier?
3. Why was the logistic activation function a key ingredient in training the first MLPs?
4. Name three popular activation functions. Can you draw them?
5. Suppose you have an MLP composed of one input layer with 10 passthrough neurons, followed by one hidden layer with 50 artificial neurons, and finally one output layer with 3 artificial neurons. All artificial neurons use the ReLU activation function.
    - What is the shape of the input matrix $\mathbf{X}$?
    - What about the shape of the hidden layer's weight vector $\mathbf{W}_h$, and the shape of its bias vector $\mathbf{b}_h$?
    - What is the shape of the output layer's weight vector $\mathbf{W}_o$, and its bias vector $\mathbf{b}_o$?
    - What is the shape of the network's output matrix $\mathbf{Y}$?
    - Write the equation that computes the network's output matrix $\mathbf{Y}$ as a function of $\mathbf{X}$, $\mathbf{W}_h$, $\mathbf{b}_h$, $\mathbf{W}_o$ and $\mathbf{b}_o$.
6. How many neurons do you need in the output layer if you want to classify email into spam or ham? What activation function should you use in the output layer? If instead you want to tackle MNIST, how many neurons do you need in the output layer, using what activation function? Answer the same questions for getting your network to predict housing prices as in Chapter 2.
7. What is backpropagation and how does it work? What is the difference between backpropagation and reverse-mode autodiff?
8. Can you list all the hyperparameters you can tweak in an MLP? If the MLP overfits the training data, how could you tweak these hyperparameters to try to solve the problem?
9. Train a deep MLP on the MNIST dataset and see if you can get over 98% precision. Just like in the last exercise of Chapter 9, try adding all the bells and whistles

(i.e., save checkpoints, restore the last checkpoint in case of an interruption, add summaries, plot learning curves using TensorBoard, and so on).

Solutions to these exercises are available in Appendix A.

CHAPTER 11
Training Deep Neural Nets

In Chapter 10 we introduced artificial neural networks and trained our first deep neural network. But it was a very shallow DNN, with only two hidden layers. What if you need to tackle a very complex problem, such as detecting hundreds of types of objects in high-resolution images? You may need to train a much deeper DNN, perhaps with (say) 10 layers, each containing hundreds of neurons, connected by hundreds of thousands of connections. This would not be a walk in the park:

- First, you would be faced with the tricky *vanishing gradients* problem (or the related *exploding gradients* problem) that affects deep neural networks and makes lower layers very hard to train.
- Second, with such a large network, training would be extremely slow.
- Third, a model with millions of parameters would severely risk overfitting the training set.

In this chapter, we will go through each of these problems in turn and present techniques to solve them. We will start by explaining the vanishing gradients problem and exploring some of the most popular solutions to this problem. Next we will look at various optimizers that can speed up training large models tremendously compared to plain Gradient Descent. Finally, we will go through a few popular regularization techniques for large neural networks.

With these tools, you will be able to train very deep nets: welcome to Deep Learning!

Vanishing/Exploding Gradients Problems

As we discussed in Chapter 10, the backpropagation algorithm works by going from the output layer to the input layer, propagating the error gradient on the way. Once the algorithm has computed the gradient of the cost function with regards to each