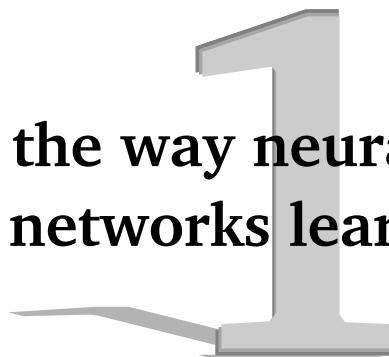


# Improving the way neural networks learn



When a golf player is first learning to play golf, they usually spend most of their time developing a basic swing. Only gradually do they develop other shots, learning to chip, draw and fade the ball, building on and modifying their basic swing. In a similar way, up to now we've focused on understanding the backpropagation algorithm. It's our "basic swing", the foundation for learning in most work on neural networks. In this chapter I explain a suite of techniques which can be used to improve on our vanilla implementation of backpropagation, and so improve the way our networks learn.

The techniques we'll develop in this chapter include: a better choice of cost function, known as the cross-entropy cost function; four so-called "regularization" methods (L1 and L2 regularization, dropout, and artificial expansion of the training data), which make our networks better at generalizing beyond the training data; a better method for initializing the weights in the network; and a set of heuristics to help choose good hyper-parameters for the network. I'll also overview several other techniques in less depth. The discussions are largely independent of one another, and so you may jump ahead if you wish. We'll also implement many of the techniques in running code, and use them to improve the results obtained on the handwriting classification problem studied in Chapter 1.

Of course, we're only covering a few of the many, many techniques which have been developed for use in neural nets. The philosophy is that the best entree to the plethora of available techniques is in-depth study of a few of the most important. Mastering those important techniques is not just useful in its own right, but will also deepen your understanding of what problems can arise when you use neural networks. That will leave you well prepared to quickly pick up other techniques, as you need them.

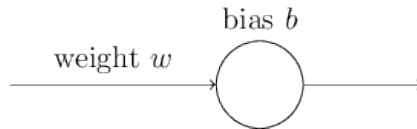
## 1.1 The cross-entropy cost function

Most of us find it unpleasant to be wrong. Soon after beginning to learn the piano I gave my first performance before an audience. I was nervous, and began playing the piece an octave

## 1

too low. I got confused, and couldn't continue until someone pointed out my error. I was very embarrassed. Yet while unpleasant, we also learn quickly when we're decisively wrong. You can bet that the next time I played before an audience I played in the correct octave! By contrast, we learn more slowly when our errors are less well-defined.

Ideally, we hope and expect that our neural networks will learn fast from their errors. Is this what happens in practice? To answer this question, let's look at a toy example. The example involves a neuron with just one input:



We'll train this neuron to do something ridiculously easy: take the input 1 to the output 0. Of course, this is such a trivial task that we could easily figure out an appropriate weight and bias by hand, without using a learning algorithm. However, it turns out to be illuminating to use gradient descent to attempt to learn a weight and bias. So let's take a look at how the neuron learns.