```
neuron  1 : w0 =  0.40 , w1 = -0.58 , w2 = -0.56
neuron  2 : w0 = -0.43 , w1 =  1.01 , w2 =  0.89
----------------
x1 = -1.0 , x2 = -1.0 , y = 0.4255
x1 = -1.0 , x2 =  1.0 , y = 0.6291
x1 =  1.0 , x2 = -1.0 , y = 0.6258
x1 =  1.0 , x2 =  1.0 , y = 0.4990
```

The last four lines show the predicted output `y` for each `x1`, `x2` combination, and we see that it implements the XOR function, since the output is greater than 0.5 when only one of the inputs is positive, which is exactly the XOR function.

Just as for the example that described backpropagation, we provide a spreadsheet that includes the mechanics of backpropagation for solving this XOR problem so that you can gain some insight through hands-on experimentation.

We did it! We have now reached the point in neural network research that was state-of-the art in 1986!

This was two years after the release of the first *Terminator* movie where a thinking machine was traveling back in time. Meanwhile, the research community is solving XOR, and we can conclude that more complicated AI was still science fiction at the time.

Network Architectures

Before moving on to solving a more complex classification problem in Chapter 4, we want to introduce the concept of network architectures. *Network architecture* is simply a name for how multiple units/neurons are connected when we build more complex networks.

Three key architectures are fundamental in most contemporary neural network applications:

- *Fully connected feedforward network.* We introduced this type of network when we solved the XOR problem. We learn more about fully connected feedforward networks in the next couple of chapters. As previously mentioned, there are no backward connections (also known as *loops* or *cycles*) in a feedforward network.

- *Convolutional neural network (CNN).* The key property of convolutional networks is that individual neurons do not have their own unique weights; they use the same weights as other neurons in the same layer. This is a property known as *weight sharing*. From a connectivity perspective, a CNN is similar to the fully connected feedforward network, but it has considerably fewer connections than a fully connected network. Instead of being fully connected, it is sparsely connected. CNNs have been shown to excel on image classification problems and therefore represent an important class of neural networks. The feature identifier described in Chapter 2 that recognized a pattern in a 3×3 patch of an image plays a central role in CNNs.

- *Recurrent neural network (RNN).* As opposed to the feedforward network, the RNN has backward connections; that is, it is not a directed acyclic graph (DAG), because it contains cycles. We have not yet shown any examples of recurrent connections, but they are studied in more detail in Chapter 9, "Predicting Time Sequences with Recurrent Neural Networks."

Figure 3-10 shows illustrations of the three network types.

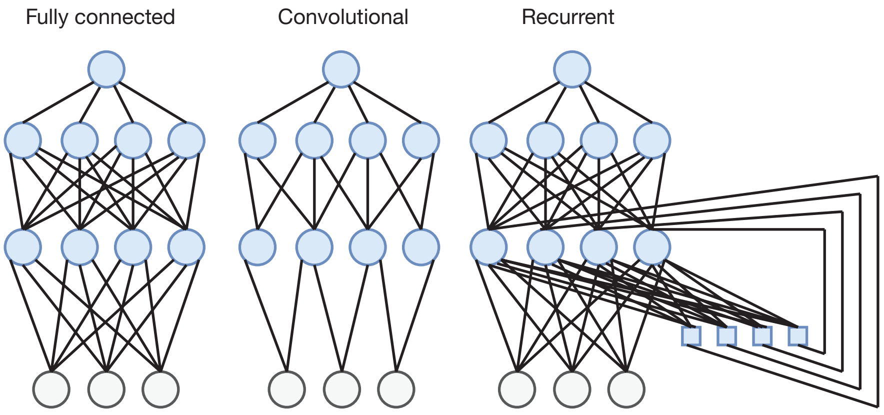

Figure 3-10 Three types of network architectures. Neurons in a convolutional network do not have unique weights but use the same weights (weight sharing) as other neurons in the same layer (not shown in the figure).

We discuss these architectures in more detail in later chapters. For now, it is helpful to know that the CNN has fewer connections than the fully connected network, whereas the RNN has more connections and some additional elements (the squares in the figure) needed to feed back the output to the input.

Fully connected, convolutional, and **recurrent** networks are three key network architectures. More complex networks often consist of combinations of these three architectures.

It is often the case that networks are hybrids of these three architectures. For example, some layers of a CNN are often fully connected, but the network is still considered to be a CNN. Similarly, some layers of an RNN might not have any cycles, as seen in Figure 3-10. Finally, you can build a network from a combination of fully connected layers, convolutional layers, and recurrent layers to tap into properties of each type of architecture.

Concluding Remarks on Backpropagation

This chapter contained a lot of mathematical formulas, and you might have found it challenging to get through. However, there is no need to worry even if you did not read it all in detail. The rest of the book is less heavy on the formulas, and there will be more focus on different network architectures with a lot of programming examples.

Taking a step back, it is worth considering the overall effect of all the equations. We started with randomly initialized weights for the network. We then ran an example through it and hoped that its output value would match the ground truth. Needless to say, with weights selected at random, this is typically not the case. The next step was therefore to identify in what direction and by how much to modify each weight to make the network perform better. To do this, we needed to know how sensitive the output was to a change in each weight. This sensitivity is simply the definition of a partial derivative of the output with respect to the weight. That is, all in all, we needed to calculate a partial derivative corresponding to each weight. The backpropagation algorithm is a mechanical and efficient way of doing this.

In Chapter 4, we extend our multilevel network to be able to handle the case of multiple outputs. That will be the last time in this book that we implement the backpropagation algorithm in detail. After that, we move on to using a DL framework, which implements the details under the hood.