Dropout Layers

When studying for an exam, it is common practice for students to use past papers and sample questions to improve their knowledge of the subject material. Some students try to memorize the answers to these questions, but then come unstuck in the exam because they haven't truly understood the subject matter. The best students use the practice material to further their general understanding, so that they are still able to answer correctly when faced with new questions that they haven't seen before.

The same principle holds for machine learning. Any successful machine learning algorithm must ensure that it generalizes to unseen data, rather than simply *remembering* the training dataset. If an algorithm performs well on the training dataset, but not the test dataset, we say that it is suffering from *overfitting*. To counteract this problem, we use *regularization* techniques, which ensure that the model is penalized if it starts to overfit.

There are many ways to regularize a machine learning algorithm, but for deep learning, one of the most common is by using *dropout* layers. This idea was introduced by Geoffrey Hinton in 2012 and presented in a 2014 paper by Srivastava et al.[7]

Dropout layers are very simple. During training, each dropout layer chooses a random set of units from the preceding layer and sets their output to zero, as shown in Figure 2-16.

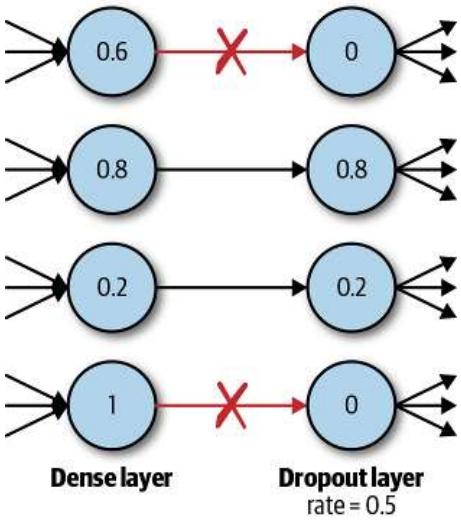

Figure 2-16. A dropout layer

7 Nitish Srivastava et al., "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research* 15 (2014): 1929–1958, *http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf*.

Incredibly, this simple addition drastically reduces overfitting, by ensuring that the network doesn't become overdependent on certain units or groups of units that, in effect, just remember observations from the training set. If we use dropout layers, the network cannot rely too much on any one unit and therefore knowledge is more evenly spread across the whole network. This makes the model much better at generalizing to unseen data, because the network has been trained to produce accurate predictions even under unfamiliar conditions, such as those caused by dropping random units. There are no weights to learn within a dropout layer, as the units to drop are decided stochastically. At test time, the dropout layer doesn't drop any units, so that the full network is used to make predictions.

Returning to our analogy, it's a bit like a math student practicing past papers with a random selection of key formulae missing from their formula book. This way, they learn how to answer questions through an understanding of the core principles, rather than always looking up the formulae in the same places in the book. When it comes to test time, they will find it much easier to answer questions that they have never seen before, due to their ability to generalize beyond the training material.

The `Dropout` layer in Keras implements this functionality, with the `rate` parameter specifying the proportion of units to drop from the preceding layer:

```
Dropout(rate = 0.25)
```

`Dropout` layers are used most commonly after `Dense` layers since these are most prone to overfitting due to the higher number of weights, though you can also use them after convolutional layers.

Batch normalization also has been shown to reduce overfitting, and therefore many modern deep learning architectures don't use dropout at all, and rely solely on batch normalization for regularization. As with most deep learning principles, there is no golden rule that applies in every situation—the only way to know for sure what's best is to test different architectures and see which performs best on a holdout set of data.

Putting It All Together

You've now seen three new Keras layer types: `Conv2D`, `BatchNormalization`, and `Dropout`. Let's put these pieces together into a new deep learning architecture and see how it performs on the CIFAR-10 dataset.

You can run the following example in the Jupyter notebook in the book repository called *02_03_deep_learning_conv_neural_network.ipynb*.

The model architecture we shall test is shown here: