# 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

parameter in the network, it uses these gradients to update each parameter with a Gradient Descent step.

Unfortunately, gradients often get smaller and smaller as the algorithm progresses down to the lower layers. As a result, the Gradient Descent update leaves the lower layer connection weights virtually unchanged, and training never converges to a good solution. This is called the *vanishing gradients* problem. In some cases, the opposite can happen: the gradients can grow bigger and bigger, so many layers get insanely large weight updates and the algorithm diverges. This is the *exploding gradients* problem, which is mostly encountered in recurrent neural networks (see Chapter 14). More generally, deep neural networks suffer from unstable gradients; different layers may learn at widely different speeds.

Although this unfortunate behavior has been empirically observed for quite a while (it was one of the reasons why deep neural networks were mostly abandoned for a long time), it is only around 2010 that significant progress was made in understanding it. A paper titled "Understanding the Difficulty of Training Deep Feedforward Neural Networks" by Xavier Glorot and Yoshua Bengio[1] found a few suspects, including the combination of the popular logistic sigmoid activation function and the weight initialization technique that was most popular at the time, namely random initialization using a normal distribution with a mean of 0 and a standard deviation of 1. In short, they showed that with this activation function and this initialization scheme, the variance of the outputs of each layer is much greater than the variance of its inputs. Going forward in the network, the variance keeps increasing after each layer until the activation function saturates at the top layers. This is actually made worse by the fact that the logistic function has a mean of 0.5, not 0 (the hyperbolic tangent function has a mean of 0 and behaves slightly better than the logistic function in deep networks).

Looking at the logistic activation function (see Figure 11-1), you can see that when inputs become large (negative or positive), the function saturates at 0 or 1, with a derivative extremely close to 0. Thus when backpropagation kicks in, it has virtually no gradient to propagate back through the network, and what little gradient exists keeps getting diluted as backpropagation progresses down through the top layers, so there is really nothing left for the lower layers.

1 "Understanding the Difficulty of Training Deep Feedforward Neural Networks," X. Glorot, Y Bengio (2010).

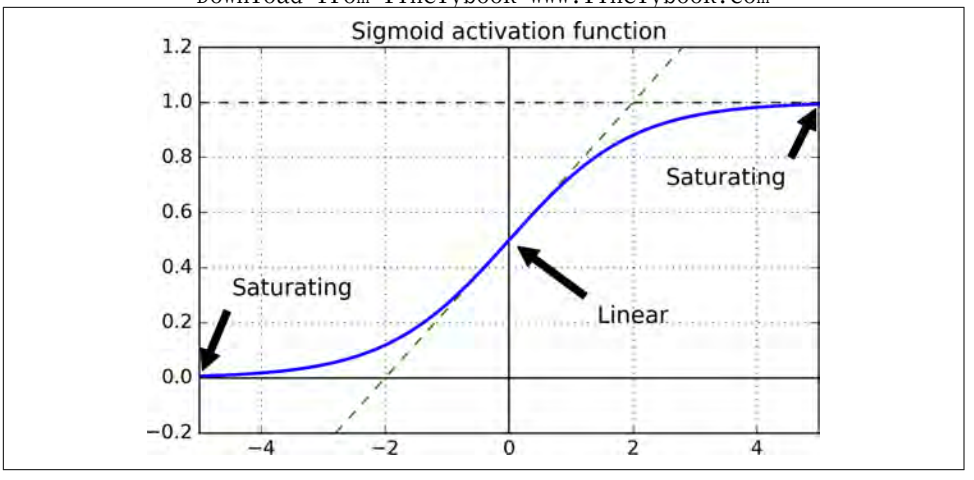


*Figure 11-1. Logistic activation function saturation*

## Xavier and He Initialization

In their paper, Glorot and Bengio propose a way to significantly alleviate this problem. We need the signal to flow properly in both directions: in the forward direction when making predictions, and in the reverse direction when backpropagating gradients. We don't want the signal to die out, nor do we want it to explode and saturate. For the signal to flow properly, the authors argue that we need the variance of the outputs of each layer to be equal to the variance of its inputs,[2] and we also need the gradients to have equal variance before and after flowing through a layer in the reverse direction (please check out the paper if you are interested in the mathematical details). It is actually not possible to guarantee both unless the layer has an equal number of input and output connections, but they proposed a good compromise that has proven to work very well in practice: the connection weights must be initialized randomly as described in Equation 11-1, where $n_{\text{inputs}}$ and $n_{\text{outputs}}$ are the number of input and output connections for the layer whose weights are being initialized (also called *fan-in* and *fan-out*). This initialization strategy is often called *Xavier initialization* (after the author's first name), or sometimes *Glorot initialization*.

2 Here's an analogy: if you set a microphone amplifier's knob too close to zero, people won't hear your voice, but if you set it too close to the max, your voice will be saturated and people won't understand what you are saying. Now imagine a chain of such amplifiers: they all need to be set properly in order for your voice to come out loud and clear at the end of the chain. Your voice has to come out of each amplifier at the same amplitude as it came in.