Let's suppose that $f(\theta, x)$ is a function that represents a deep fully connected network. Here x is the inputs to the fully connected network and θ is the learnable weights. Then the backpropagation algorithm simply computes $\frac{\partial f}{\partial \theta}$. The practical complexities arise in implementing backpropagation for all possible functions f that arise in practice. Luckily for us, TensorFlow takes care of this already!

Universal Convergence Theorem

The preceding discussion has touched on the ideas that deep fully connected networks are powerful approximations. McCulloch and Pitts showed that logical networks can code (almost) any Boolean function. Rosenblatt's Perceptron was the continuous analog of McCulloch and Pitt's logical functions, but was shown to be fundamentally limited by Minsky and Papert. Multilayer perceptrons looked to solve the limitations of simple perceptrons and empirically seemed capable of learning complex functions. However, it wasn't theoretically clear whether this empirical ability had undiscovered limitations. In 1989, George Cybenko demonstrated that multilayer perceptrons were capable of representing arbitrary functions. This demonstration provided a considerable boost to the claims of generality for fully connected networks as a learning architecture, partially explaining their continued popularity.

However, if both backpropagation and fully connected network theory were understood in the late 1980s, why didn't "deep" learning become more popular earlier? A large part of this failure was due to computational limitations; learning fully connected networks took an exorbitant amount of computing power. In addition, deep networks were very difficult to train due to lack of understanding about good hyperparameters. As a result, alternative learning algorithms such as SVMs that had lower computational requirements became more popular. The recent surge in popularity in deep learning is partly due to the increased availability of better computing hardware that enables faster computing, and partly due to increased understanding of good training regimens that enable stable learning.

Is Universal Approximation That Surprising?

Universal approximation properties are more common in mathematics than one might expect. For example, the Stone-Weierstrass theorem proves that any continuous function on a closed interval can be a suitable polynomial function. Loosening our criteria further, Taylor series and Fourier series themselves offer some universal approximation capabilities (within their domains of convergence). The fact that universal convergence is fairly common in mathematics provides partial justification for the empirical observation that there are many slight variants of fully connected networks that seem to share a universal approximation property.

Universal Approximation Doesn't Mean Universal Learning!

A critical subtlety exists in the universal approximation theorem. The fact that a fully connected network can represent any function doesn't mean that backpropagation can learn any function! One of the major limitations of backpropagation is that there is no guarantee the fully connected network "converges"; that is, finds the best available solution of a learning problem. This critical theoretical gap has left generations of computer scientists queasy with neural networks. Even today, many academics will prefer to work with alternative algorithms that have stronger theoretical guarantees.

Empirical research has yielded many practical tricks that allow backpropagation to find good solutions for problems. We will go into many of these tricks in significant depth in the remainder of this chapter. For the practicing data scientist, the universal approximation theorem isn't something to take too seriously. It's reassuring, but the art of deep learning lies in mastering the practical hacks that make learning work.

Why Deep Networks?

A subtlety in the universal approximation theorem is that it in fact holds true for fully connected networks with only one fully connected layer. What then is the use of "deep" learning with multiple fully connected layers? It turns out that this question is still quite controversial in academic and practical circles.

In practice, it seems that deeper networks can sometimes learn richer models on large datasets. (This is only a rule of thumb, however; every practitioner has a bevy of examples where deep fully connected networks don't do well.) This observation has led researchers to hypothesize that deeper networks can represent complex functions "more efficiently." That is, a deeper network may be able to learn more complex functions than shallower networks with the same number of neurons. For example, the ResNet architecture mentioned briefly in the first chapter, with 130 layers, seems to outperform its shallower competitors such as AlexNet. In general, for a fixed neuron budget, stacking deeper leads to better results.

A number of erroneous "proofs" for this "fact" have been given in the literature, but all of them have holes. It seems the question of depth versus width touches on profound concepts in complexity theory (which studies the minimal amount of resources required to solve given computational problems). At present day, it looks like theoretically demonstrating (or disproving) the superiority of deep networks is far outside the ability of our mathematicians.