CHAPTER 4

Beyond Gradient Descent

The Challenges with Gradient Descent

The fundamental ideas behind neural networks have existed for decades, but it wasn't until recently that neural network-based learning models have become mainstream. Our fascination with neural networks has everything to do with their expressiveness, a quality we've unlocked by creating networks with many layers. As we have discussed in previous chapters, deep neural networks are able to crack problems that were previously deemed intractable. Training deep neural networks end to end, however, is fraught with difficult challenges that took many technological innovations to unravel, including massive labeled datasets (ImageNet, CIFAR, etc.), better hardware in the form of GPU acceleration, and several algorithmic discoveries.

For several years, researchers resorted to layer-wise greedy pre-training in order to grapple with the complex error surfaces presented by deep learning models.[1] These time-intensive strategies would try to find more accurate initializations for the model's parameters one layer at a time before using mini-batch gradient descent to converge to the optimal parameter settings. More recently, however, breakthroughs in optimization methods have enabled us to directly train models in an end-to-end fashion.

In this chapter, we will discuss several of these breakthroughs. The next couple of sections will focus primarily on local minima and whether they pose hurdles for successfully training deep models. In subsequent sections, we will further explore the nonconvex error surfaces induced by deep models, why vanilla mini-batch gradient descent falls short, and how modern nonconvex optimizers overcome these pitfalls.

1 Bengio, Yoshua, et al. "Greedy Layer-Wise Training of Deep Networks." *Advances in Neural Information Processing Systems* 19 (2007): 153.

Local Minima in the Error Surfaces of Deep Networks

The primary challenge in optimizing deep learning models is that we are forced to use minimal local information to infer the global structure of the error surface. This is a hard problem because there is usually very little correspondence between local and global structure. Take the following analogy as an example.

Let's assume you're an ant on the continental United States. You're dropped randomly on the map, and your goal is to find the lowest point on this surface. How do you do it? If all you can observe is your immediate surroundings, this seems like an intractable problem. If the surface of the US was bowl shaped (or mathematically speaking, convex) and we were smart about our learning rate, we could use the gradient descent algorithm to eventually find the bottom of the bowl. But the surface of the US is extremely complex, that is to say, is a nonconvex surface, which means that even if we find a valley (a local minimum), we have no idea if it's the lowest valley on the map (the global minimum). In Chapter 2, we talked about how a mini-batch version of gradient descent can help navigate a troublesome error surface when there are spurious regions of magnitude zero gradients. But as we can see in Figure 4-1, even a stochastic error surface won't save us from a deep local minimum.

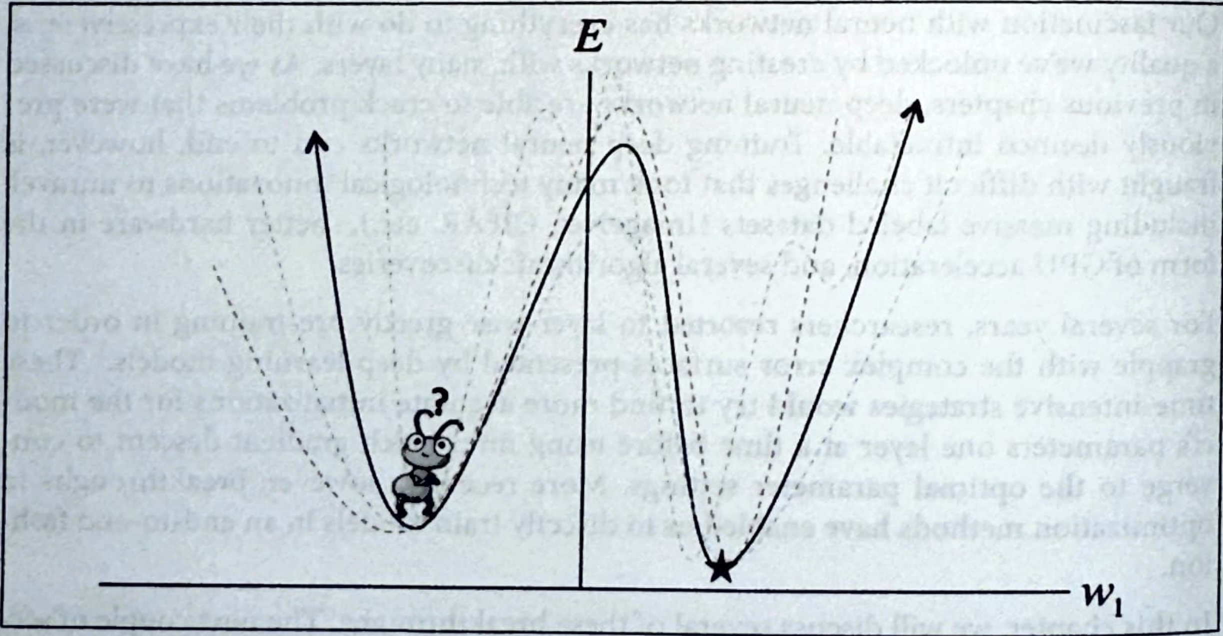

Figure 4-1. Mini-batch gradient descent may aid in escaping shallow local minima, but often fails when dealing with deep local minima, as shown

Now comes the critical question. Theoretically, local minima pose a significant issue. But in practice, how common are local minima in the error surfaces of deep networks? And in which scenarios are they actually problematic for training? In the following two sections, we'll pick apart common misconceptions about local minima.

Model Identifiability

The first source of local minima is tied to a concept commonly referred to as *model identifiability*. One observation about deep neural networks is that their error surfaces are guaranteed to have a large—and in some cases, an infinite—number of local minima. There are two major reasons this observation is true.

The first is that within a layer of a fully-connected feed-forward neural network, any rearrangement of neurons will still give you the same final output at the end of the network. We illustrate this using a simple three-neuron layer in Figure 4-2. As a result, within a layer with n neurons, there are $n!$ ways to rearrange parameters. And for a deep network with l layers, each with n neurons, we have a total of $n!^l$ equivalent configurations.

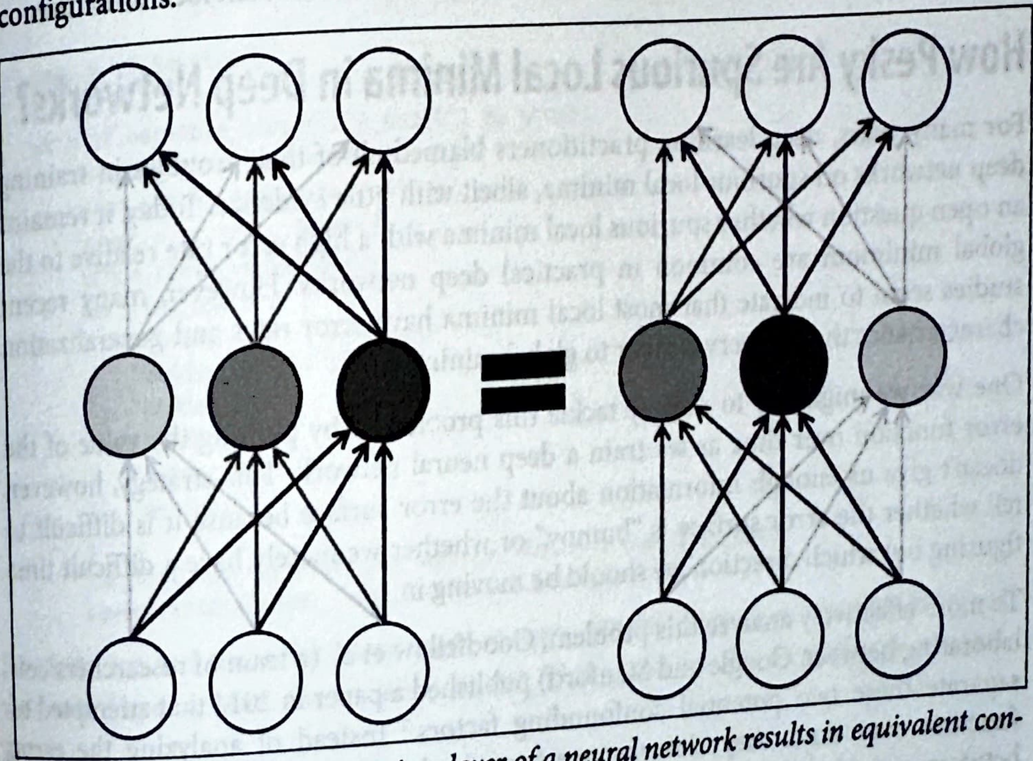

Figure 4-2. Rearranging neurons in a layer of a neural network results in equivalent configurations due to symmetry

In addition to the symmetries of neuron rearrangements, non-identifiability is present in other forms in certain kinds of neural networks. For example, there is an infinite number of equivalent configurations that for an individual ReLU neuron result in equivalent networks. Because an ReLU uses a piecewise linear function, we are free to multiply all of the incoming weights by any nonzero constant k while scaling all of the outgoing weights by $\frac{1}{k}$ without changing the behavior of the network. We leave the justification for this statement as an exercise for the active reader.