power than computer chips and smart designs should aim to learn from the brain's architecture.

A number of projects have built large spike train chips attempting to expand upon this core thesis. IBM's TrueNorth project has succeeded in building spike train processors with millions of "neurons" and demonstrated that this hardware can perform basic image recognition with significantly lower power requirements than existing chip designs. However, despite these successes, it is not clear how to translate modern deep architectures onto spike train chips. Without the ability to "compile" TensorFlow models onto spike train hardware, it's unlikely that such projects will see widespread adoption in the near future.

Distributed Deep Network Training

In the previous section, we surveyed a variety of hardware options for training deep networks. However, most organizations will likely only have access to CPUs and perhaps GPUs. Luckily, it's possible to perform *distributed training* of deep networks, where multiple CPUs or GPUs are used to train models faster and more effectively. Figure 9-5 illustrates the two major paradigms for training deep networks with multiple CPUs/GPUs, namely data parallel and model parallel training. You will learn about these methods in more detail in the next two sections.

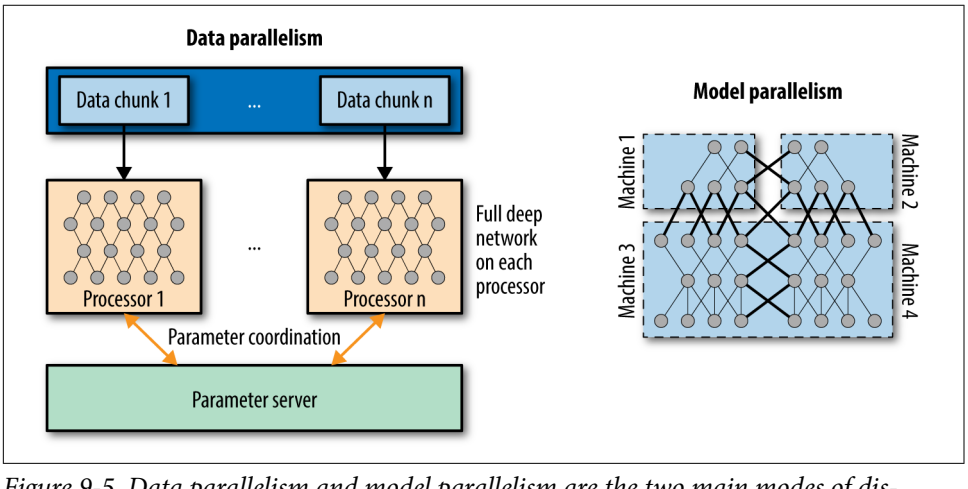

Figure 9-5. Data parallelism and model parallelism are the two main modes of distributed training of deep architectures. Data parallel training splits large datasets across multiple computing nodes, while model parallel training splits large models across multiple nodes. The next two sections will cover these two methods in greater depth.

Data Parallelism

Data parallelism is the most common type of multinode deep network training. Data parallel models split large datasets onto different machines. Most nodes are workers and have access to a fraction of the total data used to train the network. Each worker node has a complete copy of the model being trained. One node is designated as the supervisor that gathers updated weights from the workers at regular intervals and pushes averaged versions of the weights out to worker nodes. Note that you've already seen a data parallel example in this book; the A3C implementation presented in Chapter 8 is a simple example of data parallel deep network training.

As a historical note, Google's predecessor to TensorFlow, DistBelief, was based on data parallel training on CPU servers. This system was capable of achieving distributed CPU speeds (using 32–128 nodes) that matched or exceeded GPU training speeds. Figure 9-6 illustrates the data parallel training method implemented by DistBelief. However, the success of systems like DistBelief tends to depend on the presence of high throughput network interconnects that can allow for rapid model parameter sharing. Many organizations lack the network infrastructure that enables effective multinode data parallel CPU training. However, as the A3C example demonstrates, it is possible to perform data parallel training on a single node, using different CPU cores. For modern servers, it is also possible to perform data parallel training using multiple GPUs stocked within a single server, as we will show you later.

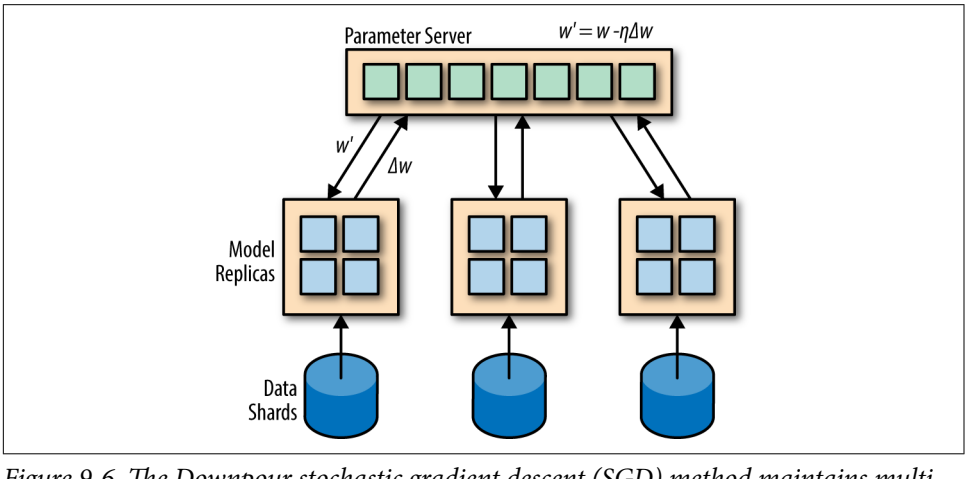

Figure 9-6. The Downpour stochastic gradient descent (SGD) method maintains multiple replicas of the model and trains them on different subsets of a dataset. The learned weights from these shards are periodically synced to global weights stored on a parameter server.