Deep Learning

Note: Unless otherwise noted all references including images are from the required textbook, Machine Learning: A Probabilistic Perspective by Kevin P. Murphy.

Deep Learning

Many of the models we have covered have a simple two-layer architecture $\mathbf{x} \rightarrow \mathbf{y}$. However, when we look at the brain, we seem many levels of processing. It is believed that each level is learning features or representations at increasing levels of abstraction. For example, the standard model of the visual cortex (Hubel and Wiesel 1962; Serre et al. 2005; Ranzato et al. 2007) suggests that the brain first extracts edges, then patches, then surfaces, then objects, etc.

Deep Learning

This observation has inspired a recent trend in machine learning known as deep learning (see e.g., deeplearning.net), which attempts to replicate this kind of architecture in a computer. Note the idea can be applied to non-vision problems as well, such as speech and language.

Deep Generative Models

Deep models often have millions of parameters. Acquiring enough labeled data to train such models is difficult, despite crowd sourcing sites. In simple settings, such as hand-written character recognition, it is possible to generate lots of labeled data by making modified copies of a small manually labeled training set, but it seems unlikely that this approach will scale to complex scenes.

Deep Generative Models

To overcome the problem of needing labeled training data, we will focus on unsupervised learning. The most natural way to perform this is to use generative models. Three kinds of deep generative models are directed, undirected, and mixed models.

Deep Directed Networks

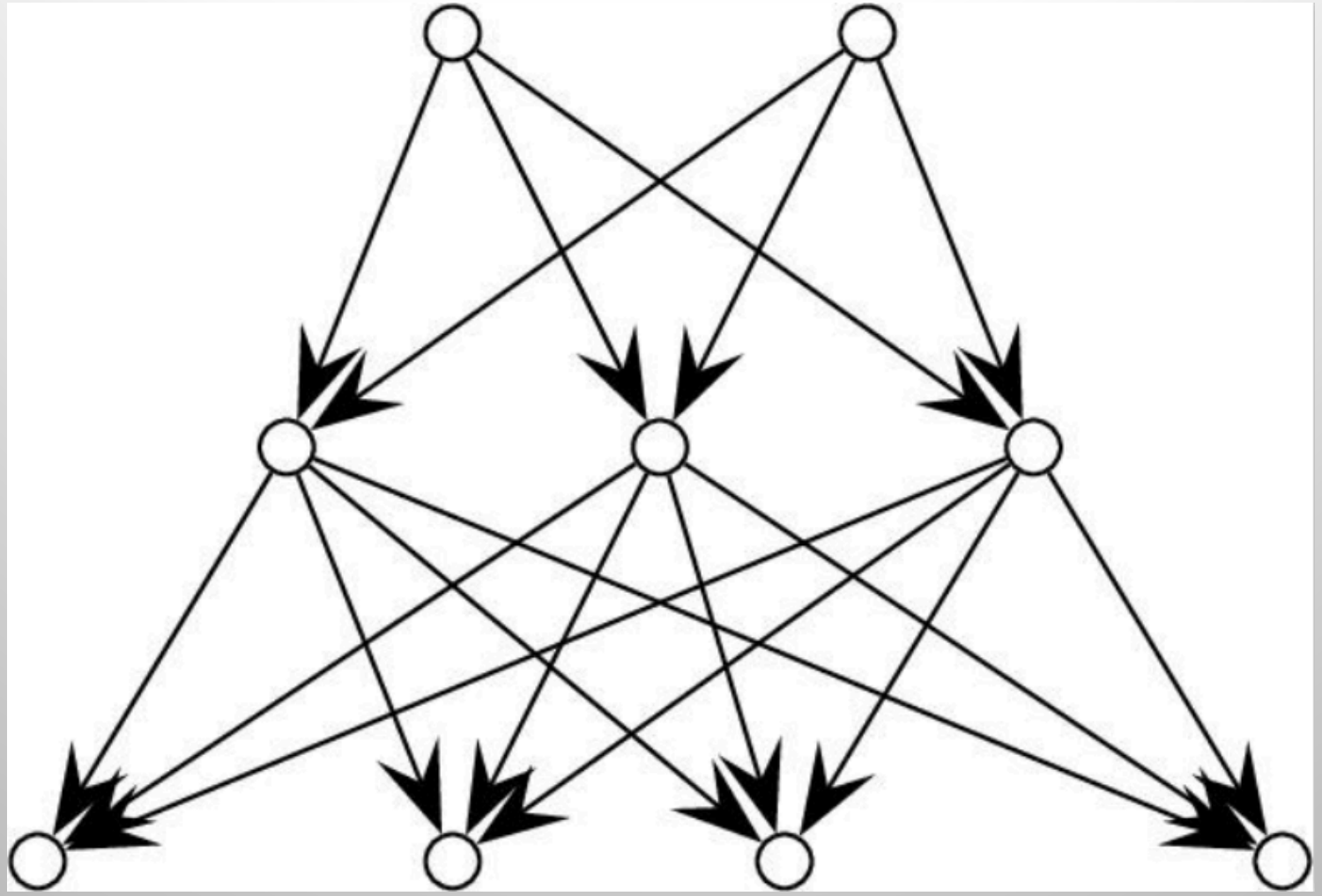
Perhaps the most natural way to build a deep generative model is to construct a deep directed graphical model. The bottom level contains the observed pixels (or whatever the data is), and the remaining layers are hidden. Just 3 layers are assumed for notational simplicity.

Deep Directed Networks

The number and size of layers is usually chosen by hand, although non-parametric Bayesian methods (Adams et al. 2010) or boosting (Chen et al. 2010) to infer the model structure can also be used.

We shall call models of this form **deep directed networks** or **DDN**s. If all the nodes are binary, and all CPDs are logistic functions, this is called a sigmoid belief net (Neal 1992).

Deep Directed Networks

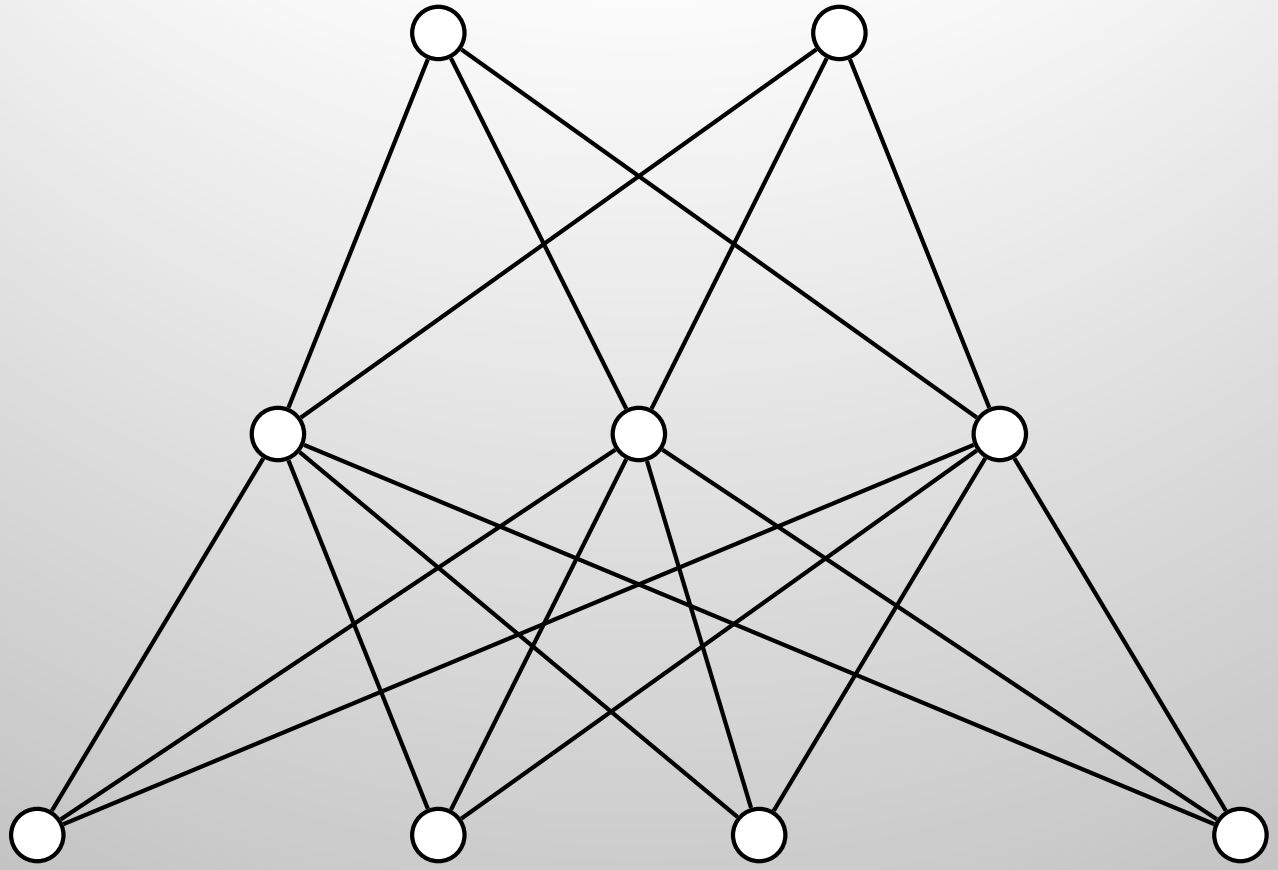


A directed deep multi-layer graphical model

Deep Boltzman Machines

A natural alternative to a directed model is to construct a deep undirected model. For example, we can stack a series of Restricted Boltzman Machines (RBMs) on top of each other. This is known as a **deep Boltzmann machine** or **DBM** (Salakhutdinov and Hinton 2009).

Deep Boltzman Machines

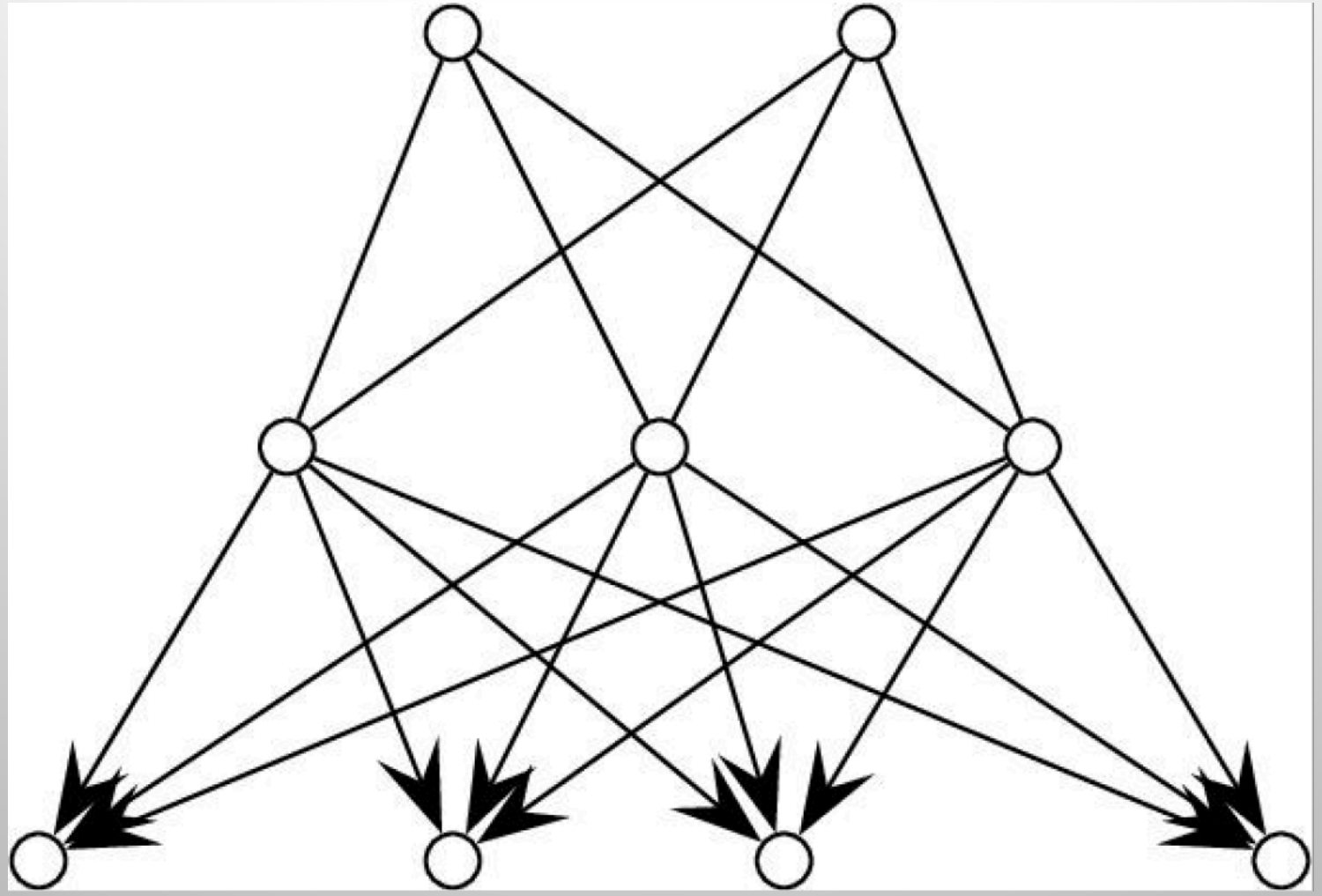


An undirected deep multi-layer graphical model

Deep Belief Networks

An interesting compromise is to use a model that is partially directed and partially undirected. In particular, suppose we construct a layered model which has directed arrows, except at the top, where there is an undirected graph. This model is known as a **deep belief network** (Hinton et al. 2006) or **DBN**.

Deep Belief Networks



A mixed directed-undirected deep multi-layer graphical model

Deep Neural Networks

Given that DBNs are often only used in a feed-forward, or bottom-up, mode, they are effectively acting like neural networks. In view of this, it is natural to dispense with the generative story and try to fit deep neural networks directly. The resulting training methods are often simpler to implement, and can be faster.

Deep Multi-Layer Perceptrons

Many decision problems can be reduced to classification, e.g., predict which object (if any) is present in an image patch, or predict which phoneme is present in a given acoustic feature vector. We can solve such problems by creating a deep feedforward neural network or multilayer perceptron (MLP), and then fitting the parameters using backpropagation and gradient descent.

Deep Multi-Layer Perceptrons

Unfortunately, this method does not work very well. One problem is that the gradient becomes weaker the further we move away from the data; this is known as the “vanishing gradient” problem (Bengio and Frasconi 1995). A related problem is that there can be large plateaus in the error surface, which cause simple first-order gradient-based methods to get stuck (Glorot and Bengio 2010).

Deep Multi-Layer Perceptrons

Consequently early attempts to learn deep neural networks proved unsuccessful. Recently there has been some progress, due to the adoption of GPUs (Ciresan et al. 2010) and second-order optimization algorithms (Martens 2010). Nevertheless, such models remain difficult to train.

Deep Multi-Layer Perceptrons

Next we will discuss a way to initialize the parameters using unsupervised learning; this is called **generative pre-training**. The advantage of performing unsupervised learning first is that the model is forced to model a high-dimensional response, namely the input feature vector, rather than just predicting a scalar response. This helps backpropagation find local minima with good generalization properties (Erhan et al. 2010; Glorotand Bengio 2010).

Deep Auto-encoders

An auto-encoder is a kind of unsupervised neural network that is used for dimensionality reduction and feature discovery. More precisely, an auto-encoder is a feedforward neural network that is trained to predict the input itself. The system can minimize the reconstruction error by ensuring the hidden units capture the most relevant aspects of the data.

Deep Auto-encoders

Suppose the system has one hidden layer, so the model has the form $\mathbf{v} \rightarrow \mathbf{h} \rightarrow \mathbf{v}$. If all the functions are linear, auto-encoders are equivalent to Principal Component Analysis (PCA). However, by using nonlinear activation functions, we can discover nonlinear representations of the data.

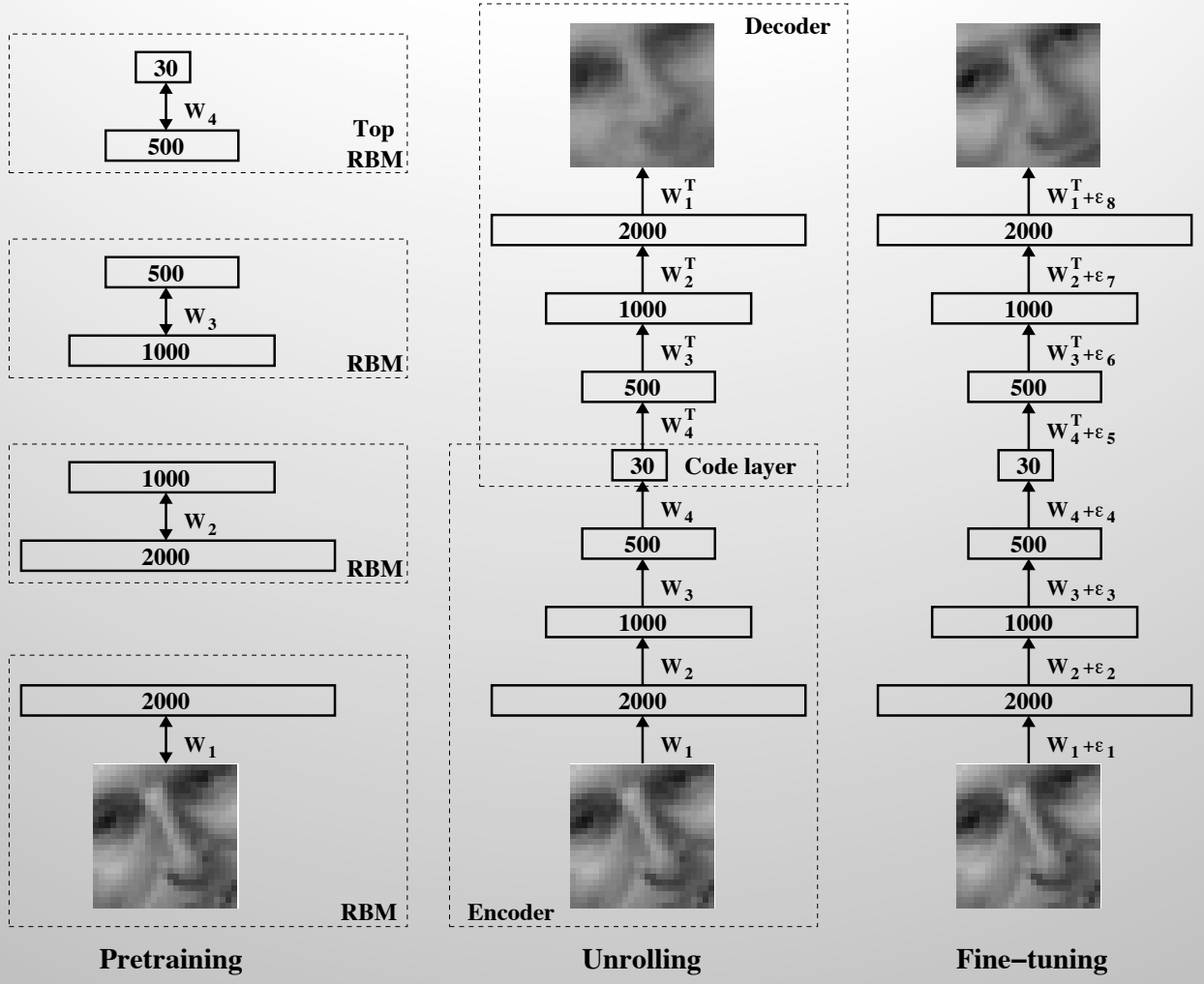
Deep Auto-encoders

More powerful representations can be learned by using deep auto-encoders. Unfortunately training such models using back-propagation does not work well, because the gradient signal becomes too small as it passes back through multiple layers, and the learning algorithm often gets stuck in poor local minima.

Deep Auto-encoders

One solution to this problem is to greedily train a series of RBMs and to use these to initialize an auto-encoder. The whole system can then be fine-tuned using backpropagation in the usual fashion. This approach, first suggested by Hinton and Salakhutdinov (2006) works much better than trying to fit the deep auto-encoder directly starting with random weights.

Deep Auto-encoders



Training a deep auto-encoder

Applications of Deep Networks

- Handwritten digit classification using DBNs
- Data visualization and feature discovery using deep auto-encoders
- Information retrieval using deep auto-encoders (semantic hashing)
- Learning audio features using 1d convolutional DBNs
- Learning image features using 2d convolutional DBNs