Deep Learning

1 The Human Visual Cortex

In the primate vision system, there seems to be a hierarchy of neurons within the visual cortex

- simple cells respond to spots of light;
- then there are cells responding to edges;
- then cells that respond to edges at particular orientations;
- and so on, up to complex shapes.

The representation of a complex object is distributed across many neurons. This is robust and economical. The intermediate features are reused in the representation of many objects.

There are perhaps as many as 8 layers in the visual cortex alone. On that basis, it seems reasonable to try additional hidden layers in artificial neural networks.

2 Deep Architectures

Early attempts to have more than 2 or 3 hidden layers were generally unsuccessful. The learning tended to get stuck in local minima. A possible explanation is that gradients defined at the output layer become too diffuse to train ever earlier layers: the blame assignment at the earlier layers is just not discriminating enough.

However, matters started to change around 2006 with the beginnings of what is now known as deep learning. Deep learners mostly share these characteristics:

- Deep architectures, e.g. neural networks with many hidden layers
- Pre-training layer by layer in an unsupervised way:
  - we show each layer some unlabelled examples;
  - weights are adjusted to achieve a compact representation of the examples.
- Fine-tuning using back-prop and some labelled examples. However, at this point the back-prop already has better-than-random weights obtained from the pre-training.

For concreteness, we will study one deep learner in a little more detail.
3 Stacked Autoencoders

An autoencoder typically has an input layer, a single hidden layer, and an output layer with the same number of TLUs as the input layer:

The idea is to train it using back-prop on a set of unlabelled examples. But back-prop needs labelled examples: in other words, it needs the target values. But we will set the target values to be equal to the input values. We want it to output the same values as we input. For every example, we want $s_i = \text{output}_i$ for all $i > 0$.

(One small technicality: since the outputs fall between 0 and 1, we'll have to normalize the inputs by making sure they are all between 0 and 1.)

This seems a bit pointless until you notice that in the diagram above there is a limited number of hidden units — fewer than there are input units. So, the network will have to discover structure in the data. It will have to use the hidden units to detect features in the inputs that in combination allow it to reconstruct the inputs.

For example, suppose we have a training set of $10 \times 10$ grayscale images, so there are 100 input units (ignoring the extra unit) and 100 output units. And suppose there are only 50 hidden units. The network must learn a compressed representation of the inputs in terms of 50 features. Assuming that the images are not just random (e.g. assuming they are images of real objects), then there ought to be structure and patterns (e.g. correlations) that it can exploit.

So far, our explanation has relied on there being fewer hidden units than there are input units. But, in fact, this isn’t necessary. We can allow lots of units in the hidden layer (more even than in the input layer, if we want), and still get the autoencoder to discover structure and patterns in the data — provided we impose some other constraint on the autoencoder. For example, suppose we constrain the TLUs in the hidden layer so that most of the time their output is
close to zero. If most of the time, a unit’s output is zero (or close to it), then that’s a bit like not having the unit in the hidden layer at all! So this too would force the network to discover a compact representation. An autoencoder that uses this idea is called a **sparse autoencoder**.

We won’t look at the maths for the sparsity constraint. Suffice to say that it involves penalizing hidden units whose average output is too far from 0 and searching for weights that minimizes the sum of both the error and the penalties.

Now consider **stacked autoencoders**, i.e. a network made up of (many) layers of (sparse) autoencoders. How would we train it?

**Pre-training.** We use a large set of unlabelled examples and we use *greedy layer-wise training*. What this means is that we take the input layer and the first hidden layer and we hook these up to an output layer with the same number of units as the input layer. We present the examples and use the back-prop algorithm to learn the weights between the input layer and the hidden layer and between the hidden layer and the output layer.

Then we repeat this but this time we use the first and second hidden layers and we hook them up to an output layer that has the same number of units as the first hidden layer. Our examples this time are the activations of the first layer that we get from our original examples. We use back-prop to learn the weights between the first hidden layer and the second hidden layer, and between the second hidden layer and the output layer.

Then we repeat again using the second and third hidden layers. And so on, through the layers.

At this point, the weights on the wires that enter all the hidden units have been learned.

**Fine-tuning.** Finally, we use the labelled examples to train the full network by the back-prop algorithm. Why is this less likely to encounter the problems that people encountered in early efforts at training deep architectures? Because we are not using random initial weights. Weights that enable the hidden units to detect features have already been set. So the final back-prop is now responsible only for some minor adjustments.

4 Remarks

Stacked autoencoders are just one example of deep architectures — one of the simplest. There are many others, of which Deep Boltzmann Machines and Convolutional Nets are perhaps the most talked about.
Recently, deep architectures and deep learning have scored some notable successes in recognition tasks and data mining competitions, some of which we will discuss in the lecture. The main players in the field have been snapped up by Google, Facebook, Microsoft, Baidu, . . .