Lecture Outline

Reading: Chapter 5 Neural Networks

This lecture covers

- Review of neural networks
- Invariances
- Convolutional neural networks

Review of Neural Networks

The basic structure of the 2-layer neural network is illustrated in Fig. 1 below.

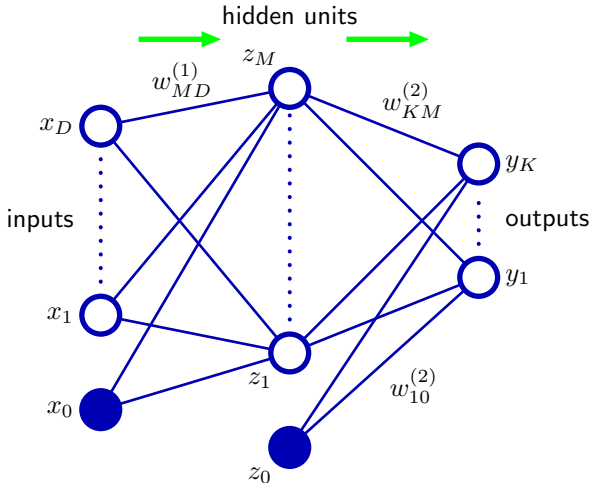

Figure 1:

Forward propagating through the network we see

1. Inputs x_i are linearly combined to form *activations*

$$a_j = \sum_{i=1}^{D} w^{(1)}_{ji} x_i + w^{(1)}_{j0} \tag{1}$$

2. Activations are transformed to hidden units

$$z_j = h(a_j) \tag{2}$$

where the nonlinear functions $h(\cdot)$ are generally chosen to be sigmoidal functions such as the logistic sigmoid or the 'tanh' function.

3. The hidden units are linearly combined to give *output unit activations*

$$a_k = \sum_{j=1}^{M} w^{(2)}_{kj} z_j + w^{(2)}_{k0} \tag{3}$$

4. The output unit activations are transformed using an appropriate activation function to give a set of network outputs

$$y_k = \sigma(a_k) \tag{4}$$

The choice of activation function is determined by the nature of the data. For standard regression problems, the activation function is the identity function $\sigma(a) = a$ so that $y_k = a_k$. For multiple binary classification problems, each output unit activation is transformed using a logistic sigmoid function so that where

$$\sigma(a) = \frac{1}{1 + \exp(-a)}. \tag{5}$$

For multiclass problems, a softmax activation function is used

$$\sigma(a_k) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}. \tag{6}$$

We can combine these various stages to give the overall network function that, for sigmoidal output unit activation functions, takes the form

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right) \tag{7}$$

where the biases are included in the weight parameters and $x_0 = 1$ and $z_0 = 1$.

For a given set of training data (inputs and targets), the weights of the neural network are obtained through a gradient-descent approach where the gradient is computed via an efficient back propagation technique. Once the neural network is trained, forward propagation of the test input through the network yields estimated targets.

5.5.3 Invariances

In many applications of pattern recognition, it is known that predictions should be unchanged, or *invariant*, under one or more transformations of the input variables. For example, in the classification of objects in 2D images, such as handwritten digits, a particular object should be assigned the same classification irrespective of its position within the image (*translation invariance*) or of its size (*scale invariance*). Such transformations produce significant changes in the raw data, expressed in terms of the intensities at each of the pixels in the image, and yet should give rise to the same output from the classification system. Similarly in speech recognition, small levels of nonlinear warping along the time axis, which preserve temporal ordering, should not change the interpretation of the signal.

If sufficiently large numbers of training patterns are available, then an adaptive model such as a neural network can learn the invariance, at least approximately. This involves including within the training set a sufficiently large number of examples of the effects of the various transformations. Thus, for translation invariance in an image, the training set should include examples of objects at many different positions.

This approach may be impractical, however, if the number of training examples is limited, or if there are several invariants (because the number of combinations of transformations grows exponentially with the number of such transformations). We therefore seek alternative approaches for encouraging an adaptive model to exhibit the required invariances. These can be broadly divided into four categories:

1. The training set is augmented using replicas of the training patterns, transformed according to the desired invariances. For instance in our digit recognition example, we could make multiple copies of each example in which the digit is shifted to a different position in each image.

2. A regularization term is added to the error function that penalizes changes in the model output when the input is transformed. This leads to the technique of *tangent propagation*.

3. Invariance is built into the pre-processing by extracting features that are invariant under the required transformations. Any subsequent regression or classification system that uses such features as inputs will necessarily also respect these invariances.

4. The final option is to build the invariance properties into the structure of a neural network. One way to achieve this is through the use of local receptive fields and shared weights, as discussed in the context of *convolutional neural networks*.

5.5.6 Convolutional neural networks

One approach to creating models that are invariant to certain transformation of the inputs is to build the invariance properties into the structure of the neural network. This is the basis for the *convolutional neural network*, which has been widely applied to computer vision and automatic speech recognition (ASR).

Consider the specific task of recognizing handwritten digits. Each input image comprises a set of pixel intensity values, and the desired output is a posterior probability distribution over the ten digits. We know that the identity of the digit is invariant under translations and scaling as well as (small rotations). Furthermore, the network must also exhibit invariance to more subtle transformations such as elastic deformations. One simple approach would be to treat the image as the input to a fully connected network. Given a sufficiently large training set, such a network could in principle yield a good solution to this problem and would learn the appropriate invariances by example.

However, this approach ignores a key property of images, which is that nearby pixels are more strongly correlated than more distant pixels. This property is also key in speech: nearby samples are more strongly correlated than more distant samples. Many of the modern approaches to computer vision exploit this property by extracting *local* features that depend on small subregions of the image. Information from such features can then be merged in later stages of processing in order to detect higher-order features and ultimately to yield information about the image as a whole.

These notions are incorporated into convolutional neural networks through three mechanisms: (i) local receptive fields, (ii) weight sharing, and (iii) subsampling. The structure of a convolutional network is illustrated in Fig. 17.

In the convolutional layer (hidden layer) the units are organized into planes, each of which is called a *feature map*. Units in a feature map each take inputs only from a (i) small subregion of the image, and all of the units in a feature map are constrained to (ii) share the same weight values. This is different than before where each of the M hidden unit accepts all D inputs and is associated with a weight connecting the d-th input with the m-th hidden unit. With *weight sharing*, the weights do not depend upon the position of the training pattern.

For instance, a feature map might consist of 100 units arranged in a 10×10 grid, with each unit taking inputs from a 5×5 pixel patch of the image. The whole feature map therefore has 25 adjustable weight parameters plus one adjustable bias parameter.

Input values from a patch are linearly combined using the weights and the bias, and the result transformed by a sigmoidal nonlinearity. If we think of the units as feature detectors, then all of the units in a feature map detect the same pattern but at different locations in the input image. Due to the weight sharing, the evaluation of the activations of these units is equivalent to a convolution of the image pixel intensities with a 'kernel' comprising the weight parameters.

If the input image is shifted, the activations of the feature map will be shifted by the same amount but will otherwise be unchanged. This provides the basis for the (approximate) invariance of the network outputs to translations and distortions of the input image. Because we will typically need to detect multiple features

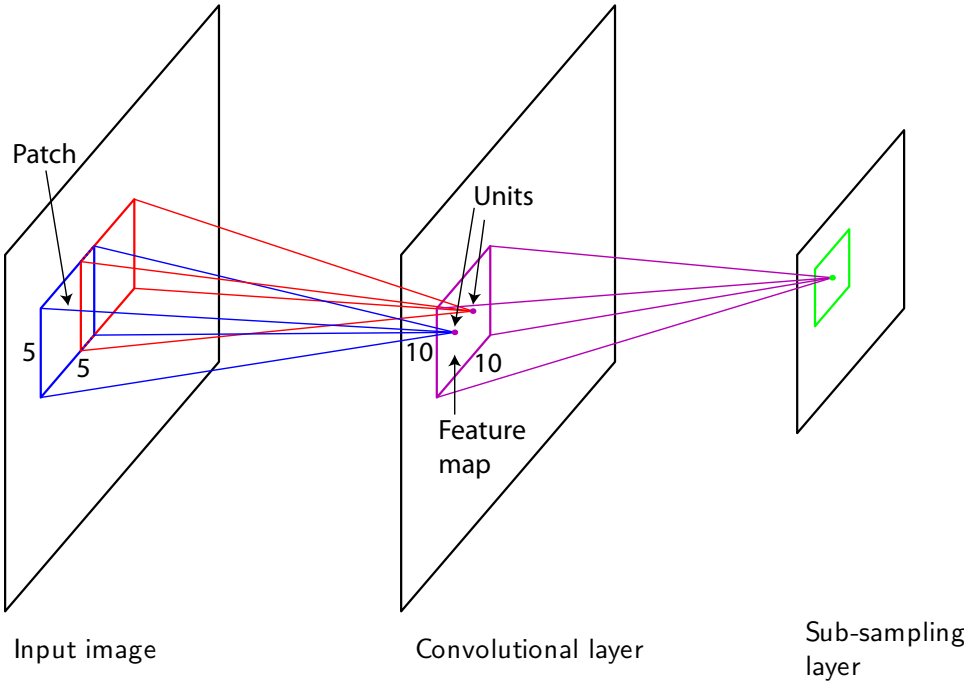

Figure 17:

in order to build an effective model, there will generally be multiple feature maps in the convolutional layer, each having its own set of weight and bias parameters.

The outputs of the convolutional units form the inputs to the (iii) subsampling layer of the network. For each feature map in the convolutional layer, there is a plane of units in the subsampling layer and each unit takes inputs from a small receptive field in the corresponding feature map of the convolutional layer. These units perform subsampling. For instance, each subsampling unit might take inputs from a 2×2 unit region in the corresponding feature map and would compute the average of those inputs, multiplied by an adaptive weight with the addition of an adaptive bias parameter, and then transformed using a sigmoidal nonlinear activation function.

The receptive fields are chosen to be contiguous and nonoverlapping so that there are half the number of rows and columns in the subsampling layer compared with the convolutional layer. In this way, the response of a unit in the subsampling layer will be relatively insensitive to small shifts of the image in the corresponding regions of the input space.

Final Remarks

At the time of publication of the PRML textbook, CNNs were successful in image recognition tasks. However, with recent research and the availability of low-cost GPU computing, CNNs with many layers and other architectural features have become very prominent in machine learning and in particular deep learning. Unfortunately, the PRML textbook does not provide adequate treatment of CNNs. Therefore, please review one of the many tutorials on CNNs including

http://machinelearningmastery.com/crash-course-convolutional-neural-networks/