Supervised learning in multilayer feedforward networks - "backpropagation"

Bruno A. Olshausen

May 6, 1998

Abstract

This handout describes the backpropagation learning rule for multilayer feedforward networks composed of McCulloch-Pitts type neurons. These methods were originally worked out by Rumelhart, Hinton, and Williams in the mid-1980's. The derivation of a learning rule for multilayer networks represented a critical breakthrough in the field, because it allows for complex input-output relationships to be learned that could not be achieved by a single-stage network due to the limitation of linear separability.

Let us consider a 2-stage network composed of sum-and-sigmoid neurons, as shown in Figure 1. Both the neurons in the middle layer, y_i, and in the output, z_i, compute

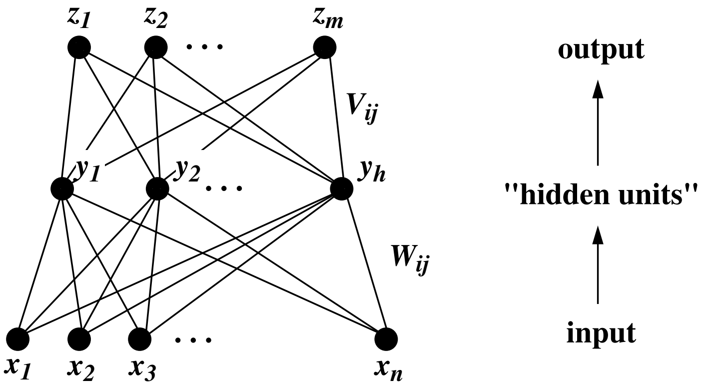

Figure 1: A multilayer, feedforward network composed of sum-and-sigmoid (i.e., McCulloch-Pitts) neurons.

their response by taking a weighted sum of the signals from the layer below, and then passing the sum through a sigmoidal nonlinearity:

$$y_i = \sigma(\sum_j W_{ij} x_j + W_{i0}) \quad (1)$$

$$z_i = \sigma(\sum_j V_{ij} y_j + V_{i0})\,. \quad (2)$$

W_{ij} denotes the weight from the j-th input unit to the i-th hidden unit, and V_{ij} denotes the weight from the j-th hidden unit to the i-th output unit. The function $\sigma(x)$ is the by now familiar sigmoid function, $\sigma(x) = \frac{1}{1+e^{-x}}$. The units in the middle layer are termed *hidden units* because they are not directly visible to either the input or the output. Note that the non-linearities in the hidden layer are crucial. Without them, the entire network could be collapsed to a single-stage linear network.

To help see where such a network architecture would be useful, let us consider a pattern discrimination problem that is linearly inseparable, and therefore not solvable by a single-stage network. Such a problem is shown in Figure 2. When a single neuron

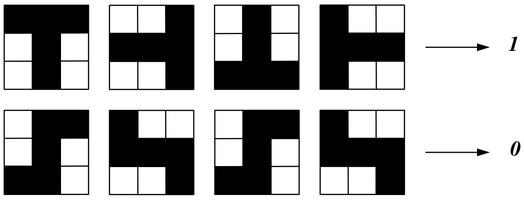

Figure 2: Discrimination of 'T' vs. 'S'.

is trained on this ensemble, it is unable to learn the desired input-output mapping. One solution we might think of is to construct a two-stage network, where the first stage consists of "feature detectors" that look for three pixels being on in any one of the four margins (Fig. 3), and the second stage pools across these detectors in order to signal the presence of a 'T'. Note that the process of "detecting" is non-linear—i.e.,

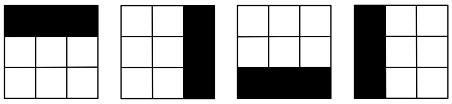

Figure 3: The weights for a set of "feature detectors" used in the first stage of a two-stage network for signaling the presence of a 'T'.

either the feature is there or it isn't. This process may be subserved by the sigmoid function by simply setting the bias for each unit in Figure 3 so that a net input value of three produces a 1 but a net input of two or less produces a 0. One way to think of the operation of this network is that the first stage transforms the input values into a new representation in the hidden layer that now makes the problem linearly separable. The problem of *learning*, then, is to find the proper weights for both the first and second stage that allow the problem to be solved.

As before, we may formulate the problem of learning in terms of gradient descent. The function to be minimized is simply the square of the difference between the actual and desired output value:

$$E = \sum_{\alpha} E^{\alpha} \qquad (3)$$

$$E^\alpha = \sum_i [z_i^\alpha - z_i(\mathbf{x}^\alpha)]^2 \tag{4}$$

where α denotes the training example, z_i^α is the desired response, and $z_i(\mathbf{x}^\alpha)$ is the computed response for the i-th output unit. The learning rule for the weights in the top-stage is similar to what we derived last time for the single-stage network:

$$\begin{aligned} \Delta V_{ij} &\propto -\frac{\partial E^\alpha}{\partial V_{ij}} && (5) \\ &= 2\,[z_i^\alpha - z_i(\mathbf{x}^\alpha)]\ \frac{\partial z_i(\mathbf{x}^\alpha)}{\partial V_{ij}} && (6) \\ &= 2\,[z_i^\alpha - z_i(\mathbf{x}^\alpha)]\ \sigma'(net_{z_i})\,y_j && (7) \end{aligned}$$

where net_{z_i} denotes the total input to z_i from below, i.e.,

$$net_{z_i} = \sum_j V_{ij} y_j + V_{i0}\,. \tag{8}$$

By making a slight change of notation we can re-write Equation 7 more compactly. Let us define as the "modified error" of output z_i:

$$\delta_{z_i} = [z_i^\alpha - z_i(\mathbf{x}^\alpha)]\ \sigma'(net_{z_i})\,. \tag{9}$$

This is simply the error on output i times the derivative of the sigmoid function evaluated at the current input level for unit i. Thus, our learning rule for V_{ij} (Eq. 7) then becomes

$$\Delta V_{ij} \propto 2\,\delta_{z_i}\,y_j\,. \tag{10}$$

This says that we change V_{ij} proportional to the "modified error" on output i, δ_{z_i}, times the current value of hidden unit j, y_j. This is simple enough, no?

Now let us derive the learning rule for the weights in the bottom stage. Again, using gradient descent we get:

$$\begin{aligned} \Delta W_{kl} &\propto -\frac{\partial E^\alpha}{\partial W_{kl}} && (11) \\ &= 2\sum_i [z_i^\alpha - z_i(\mathbf{x}^\alpha)]\ \frac{\partial z_i(\mathbf{x}^\alpha)}{\partial W_{kl}}\,. && (12) \end{aligned}$$

This time, z_i depends indirectly on W_{kl} through the value of hidden unit k. So, let's write out the derivative on the right side of Equation 12 in two parts:

$$\begin{aligned} \frac{\partial z_i(\mathbf{x}^\alpha)}{\partial W_{kl}} &= \frac{\partial z_i(\mathbf{x}^\alpha)}{\partial y_k} \times \frac{\partial y_k}{\partial W_{kl}} && (13) \\ &= \sigma'(net_{z_i}) V_{ik} \times \sigma'(net_{y_k}) x_l\,. && (14) \end{aligned}$$

Thus, the learning rule for W_{kl} is

$$\Delta W_{kl} \propto 2\sum_i [z_i^\alpha - z_i(\mathbf{x}^\alpha)]\ \sigma'(net_{z_i}) V_{ik}\,\sigma'(net_{y_k}) x_l\,. \tag{15}$$

At first sight, this looks like a mess. But let's compactify it a little as before. By using the definition of δ_{z_i} in Equation 9, and in addition defining the "modified error" for hidden unit k to be

$$\delta_{y_k} = \sigma'(net_{y_k}) \sum_i \delta_{z_i} V_{ik} \tag{16}$$

then our learning rule for W_{kl} becomes

$$\Delta W_{kl} \propto 2\, \delta_{y_k}\, x_l\,. \tag{17}$$

Thus, the learning rule for W_{kl} is not unlike that for V_{ij}. We simply change W_{kl} proportional to the "modified error" for hidden unit k, δ_{y_k}, times the value of input l, x_l. In this case, though, δ_{y_k} is obtained by "backpropagating" the δ_{z_i}'s in the layer above through the weights V_{ik}.

The problem of devising a learning rule for multilayer networks stumped neural network researchers for nearly a quarter century. One of the reasons for this is that the non-linearities used by early researchers where typically "hard" threshold units (i.e., a sigmoid with high λ) which did not allow for taking a derivative. If one simply allows for smooth non-linearities that can be differentiated, then deriving a learning rule is pretty straightforward.

The backpropagation algorithm is a powerful tool for learning complex input-output mappings. In fact, with enough hidden units, any conceivable function could be theoretically implemented by a multilayer network. However, the backpropagation algorithm is not always guaranteed to find you the "correct" solution from a limited set of training data. A major limitation of the backpropagation learning algorithm is that it is prone to get stuck in *local minima*. This is because the technique of gradient descent is blind to the global characteristics of the error surface; it simply moves downhill from where it currently is. An additional problem that plagues learning with backpropagation in multilayer networks is the problem of *generalization*. Usually it is desired in supervised learning problems for the network to be able to generalize what it has learned from the training examples so that it responds appropriately to input patterns it has not seen before. But if there are too many hidden units, then the network will usually end up overfitting the training data and thus give inappropriate responses to input patterns that were not part of the training set. These two problems of avoiding local minima and generalization are the focus of many current research efforts.

A final problem of backpropagation, from a neurobiological standpoint, is its biological implausibility. The algorithm depends on having a "teacher" that allows the network to compute an error for each output, and then these errors must be percolated backward through the network to provide appropriate learning signals at synapses deep down in the system. While this is not impossible, it is nevertheless difficult to conceive of real neurons operating in this manner.