Lecture Outline

Reading: Chapter 5 Neural Networks

This lecture covers

- Review of neural networks and notation
- Error function
- Steepest descent algorithm
- Error backpropagation and algorithm
- Example

5.2 Network Training

Overall Network and Notation

The overall network function is given by

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

where the activation is given by

$$a_j^{(1)} = \sum_{i=0}^{D} w_{ji}^{(1)} x_i, \tag{2}$$

hidden unit is given by

$$z_j = h\left(a_j^{(1)}\right), \tag{3}$$

and output activation is given by

$$a_k^{(2)} = \sum_{j=0}^{M} w_{kj}^{(2)} z_j. \tag{4}$$

The function σ maps output unit activations to network output y_k.

Error Function

Given a training set comprising a set of input vectors $\{\mathbf{x}_n\}$ where $n = 1, \ldots, N$, together with a corresponding set of target vectors $\{\mathbf{t}_n\}$, we seek to minimize the error function

$$\begin{aligned} E(\mathbf{w}) &= \sum_{n=1}^{N} E_n(\mathbf{w}) \\ &= \frac{1}{2}\sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n\|^2. \end{aligned} \tag{5}$$

through proper solution of $\mathbf{w} = \left\{w_{ji}^{(1)}, w_{kj}^{(2)}\right\}$.

Given the complicated nature of (1), there is clearly no hope of finding an analytical solution to the equation $\nabla E(\mathbf{w}) = 0$. Further complicating the problem, $E(\mathbf{w})$ may have many local minima. For successful application of neural networks, it may not be necessary to find the global minimum but it may be necessary to compare several local minima in order to find a sufficiently good solution.

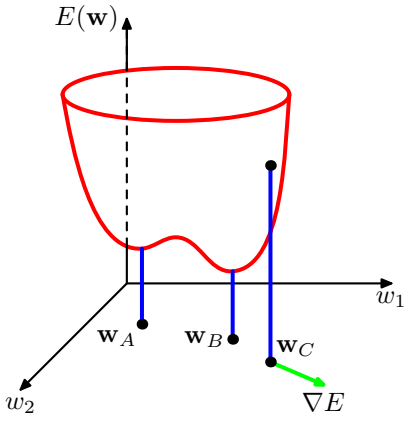

Figure 5:

5.2.3 Use of Gradient Information

The simplest approach to finding a solution for $\mathbf{w}$, i.e. local minima of $E(\mathbf{w})$ is to use gradient information in choosing the weight update. To do this we compute a small step in the direction of the negative gradient, so that

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)}) \tag{6}$$

where the parameter $\eta > 0$ is known as the *learning rate* or step-size. After each such update, the gradient is re-evaluated, using the whole training set, for the new weight vector and the process repeated. Note that the error function is defined with respect to a training set, and so after each step requires that the entire training set is processed in order to evaluate $\nabla E(\mathbf{w})$.

It is possible to evaluate the gradient of the error function efficiently by means of the *backpropagation* procedure. The use of this gradient information can lead to significant improvements in the speed with which a solution, $\mathbf{w}$ can be located.

5.3 Error Backpropagation

Introduction

Our goal is to find an efficient technique for finding $\mathbf{w}$ such that $\nabla E(\mathbf{w}) = 0$, i.e. a local minimum of the error function. We shall see that this can be achieved using a local message passing scheme in which information is sent alternately forwards and backwards through the network and is known as *error backpropagation*. There are two distinct stages in the iterative procedure for numerically finding our solution: in the first stage, the derivatives of the error function with respect to the weights are evaluated (backpropagation) and in the second stage, the derivatives are used to compute adjustments to the weights.

5.3.1 Example: Gradient of Error Function for a Linear Model

To begin, we review the gradient of the squared-error function for a linear model. Consider first a simple linear model in which the outputs y_k are linear combinations of input variables x_i,

$$y_k = \sum_{i=0}^{D} w_{ki} x_i \tag{7}$$

i.e. $\mathbf{y} = [y_1, \ldots, y_K]^T$ and $\mathbf{x} = [1, x_1, \ldots, x_D]^T$. The squared-error function, for a particular input $\mathbf{x}_n$ takes the form

$$E_n = \frac{1}{2} \sum_{k=1}^{K} (y_{nk} - t_{nk})^2; \quad y_{nk} = y_k(\mathbf{x}_n, \mathbf{w}) \tag{8}$$

where t_{nk} is the kth element of the associated target $\mathbf{t}_n$. As we have seen, the gradient of a squared-error function is of the form

$$\begin{aligned} \frac{\partial E_n}{\partial w_{ki}} &= (y_{nk} - t_{nk}) x_{ni} \\ &= \delta_{nk} x_{ni} \end{aligned} \tag{9}$$

where $\delta_{nk} = y_{nk} - t_{nk}$ is the 'error signal' associated with the kth output end of the link w_{ki} and x_{ni} is associated with the ith input end of the link. This is a "local" computation since it involves only the error at the output of the link and the input of the link.

Figure 6: Gradient of squared-error for a linear model

Gradient of Error Function for a Two-Layer Neural Network

In the two-layer feedforward network, the activation and output unit activation are given by

$$a_j^{(1)} = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \quad \text{and} \quad a_k^{(2)} = \sum_{j=0}^{M} w_{kj}^{(2)} z_j = \sum_{j=0}^{M} w_{kj}^{(2)} h\left(a_j^{(1)}\right) \tag{10}$$

where x_i sends a connection to activation $a_j^{(1)}$ and $w_{ji}^{(1)}$ is the weight associated with that connection in the first layer and z_j sends a connection to activation $a_k^{(2)}$ and $w_{kj}^{(2)}$ is the weight associated with the connection in the second layer.

We have to find $\left\{w_{ji}^{(1)}, w_{kj}^{(2)}\right\}$ that minimizes $E = \sum E_n$ for a given $\{\mathbf{x}_n, \mathbf{t}_n\}$. In this derivation, we assume a linear output activation function so that $y_k = a_k^{(2)}$.

First: From (9), the gradient with respect to the second layer weights is simply given by

$$\begin{aligned} \frac{\partial E_n}{\partial w_{kj}^{(2)}} &= (a_{nk}^{(2)} - t_{nk}) z_{nj} \\ &= \delta_{nk}^{(2)} z_{nj}. \end{aligned} \tag{11}$$

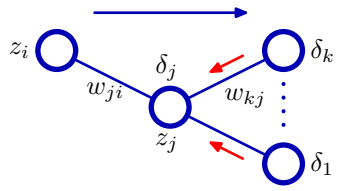

Figure 7: Gradients for first and second layers

We see that $\delta_{nk}^{(2)} = a_{nk}^{(2)} - t_{nk}$ is the 'error signal' associated with the kth output end of the link $w_{kj}^{(2)}$ and z_{nj} is associated with the jth input end of the link.

Second: In order to determine the gradient with respect to the first layer weights, we have to evaluate $\delta_{nj}^{(1)}$ for the hidden units, i.e. outputs for the first layer. Using the chain rule we have

$$\begin{aligned} \frac{\partial E_n}{\partial w_{ji}^{(1)}} &= \frac{\partial E_n}{\partial a_j^{(1)}} \frac{\partial a_j^{(1)}}{\partial w_{ji}^{(1)}} \\ &= \delta_{nj}^{(1)} x_{ni} \end{aligned} \tag{12}$$

where

$$\begin{aligned} \delta_{nj}^{(1)} &\equiv \frac{\partial E_n}{\partial a_j^{(1)}} \\ &= \sum_k \frac{\partial E_n}{\partial a_k^{(2)}} \frac{\partial a_k^{(2)}}{\partial a_j^{(1)}} \end{aligned} \tag{13}$$

where the sum runs over all output activation units k to which activation unit j sends connections. We note that

$$\frac{\partial E_n}{\partial a_k^{(2)}} = \delta_{nk}^{(2)} \tag{14}$$

By using the activation $z_j = h\left(a_j^{(1)}\right)$ in (10), we have the *backpropagation* formula

$$\delta_{nj}^{(1)} = h'\left(a_{nj}^{(1)}\right) \sum_k w_{kj}^{(2)} \delta_{nk}^{(2)} \tag{15}$$

which tells us that the value of $\delta^{(1)}$ for a particular hidden unit can be obtained by propagating the $\delta^{(2)}$'s backwards.

Error Backpropagation Algorithm

1. Initialize $w_{ji}^{(1)}$ and $w_{kj}^{(2)}$ for $0 \le i \le D$, $0 \le j \le M$, and $1 \le k \le K$.
2. Apply an input vector $\mathbf{x}_n$ to the network and forward propagate to find the activations $a_{nj}^{(1)} = \sum_{i=0}^{D} w_{ji}^{(1)} x_i$ of the hidden units and activations $a_{nk}^{(2)} = \sum_{j=0}^{M} w_{kj}^{(2)} z_{nj}$ of the output units where $z_{nj} = h\left(a_{nj}^{(1)}\right)$.
3. Compute $\delta_{nk}^{(2)} = a_{nk}^{(2)} - t_{nk}$
4. Backpropagate $\delta_{nk}^{(2)}$ to obtain $\delta_{nj}^{(1)} = h'\left(a_{nj}^{(1)}\right) \sum_k w_{kj}^{(2)} \delta_{nk}^{(2)}$

5. Evaluate $\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_{nj}^{(1)} x_{ni}$ and $\frac{\partial E_n}{\partial w_{ki}^{(2)}} = \delta_{nk}^{(2)} z_{nj}$ obtain the required derivatives

Note that this will give $\nabla E(\mathbf{w})$ which can then be used in (6).

5.3.2 Example

Consider a two-layer network in which the output units have a linear activation function so that $y_k = a_k$ while the hidden units have sigmoidal activation functions given by

$$h(a) = \tanh(a) \tag{16}$$

where

$$\tanh(a) \equiv \frac{e^a - e^{-a}}{e^a + e^{-a}} \tag{17}$$

and $h'(a) = 1 - h(a)^2$.

1. Initialize $w_{ji}^{(1)}$ and $w_{kj}^{(2)}$ for $0 \leq i \leq D$, $0 \leq j \leq M$, and $1 \leq k \leq K$.
2. For each $\mathbf{x}_n$, we perform a forward propagation using

$$\begin{aligned} a_{nj}^{(1)} &= \sum_{i=0}^{D} w_{ji}^{(1)} x_{ni} \\ z_{nj} &= \tanh\left(a_{nj}^{(1)}\right) \\ y_{nk} &= \sum_{j=0}^{M} w_{kj}^{(2)} z_{nj} \end{aligned} \tag{18}$$

3. Compute

$$\delta_{nk}^{(2)} = y_{nk} - t_{nk} \tag{19}$$

4. Backpropagate $\delta_{nk}^{(2)}$ to obtain the δ's for the hidden units using

$$\delta_{nj}^{(1)} = (1 - z_{nj}^2) \sum_{k=1}^{K} w_{kj}^{(2)} \delta_{nk}^{(2)}. \tag{20}$$

5. Compute

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_{nj}^{(1)} x_{ni} \quad \text{and} \quad \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_{nk}^{(2)} z_{nj}. \tag{21}$$

6. Using (21), update weights

$$\mathbf{w}^{(1)} \leftarrow \mathbf{w}^{(1)} - \eta \nabla E(\mathbf{w}^{(1)}) \quad \text{and} \quad \mathbf{w}^{(2)} \leftarrow \mathbf{w}^{(2)} - \eta \nabla E(\mathbf{w}^{(2)}) \tag{22}$$

7. Goto Step 2 until convergence of $\mathbf{w}^{(1)}$, $\mathbf{w}^{(2)}$ or a fixed number of iterations has elapsed.

In the test stage, we forward propagate through the neural network using $\mathbf{w}^{(1)}$ and $\mathbf{w}^{(2)}$ from training.