Backpropagation Neural Network

Michael Holuša

2020-02-28

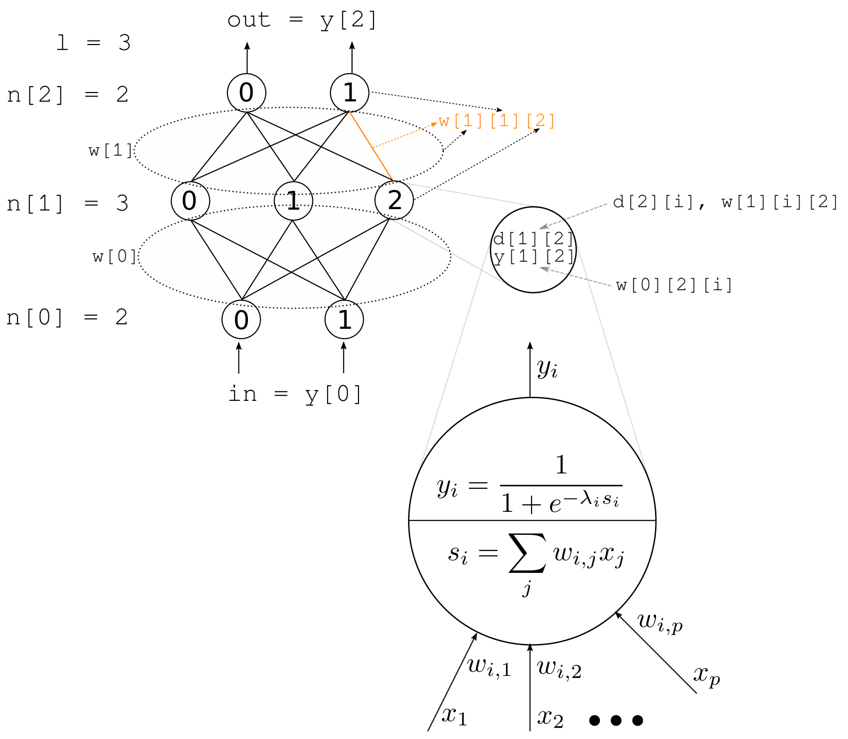

Figure 1: Neural Network Model

The potential of neuron i in a layer k is defined as the weighted sum of all inputs to the neuron, i.e.

$$s_{k,i} = \sum_j w_{k-1,i,j}\, y_{k-1,j}\ , \tag{1}$$

where $w_{k-1,i,j}$ is the weight between a neuron i from the k-th layer and a neuron j from the $(k-1)$-st layer, $y_{k-1,j}$ is the input to the neuron i (output of the neurons from the previous layer). If this potential is higher than a threshold value, the neuron is excited to the value 1, otherwise it is inhibited to the value 0. The value of the excitation depends on an activation function. For this purpose, we use the sigmoid function. Let $y_{k,i}$ be the output of the neuron i in the layer k. The output (the excitation value) is defined as

$$y_{k,i} = \frac{1}{1+e^{-\lambda s_{k,i}}}\ . \tag{2}$$

The error of the network is given by the values of y in the output layer comparing to the expected values. Let us assume that we have a neural network with 3 layers - input layer, one hidden layer, and output layer (as shown in Figure 1. The output layer contains two neurons, therefore, the expected output on a given input is represented by two values. Let us now assume that we want to classify the objects into two classes (square and rectangle) using our features $F1$, $F2$. Let the output of first neuron in the output layer $y_{2,0}$ be the probability that the features represent the square object. Similarly, let $y_{2,1}$ be the probability that the features represent the rectangular object. For the square, the expected values are $y_{2,0} = 1$, $y_{2,1} = 0$. For the rectangle, it is $y_{2,0} = 0$, $y_{2,1} = 1$. The error of the network is given by the difference between the expected and the real output. Let t_i be the expected output of the neuron i. Then the error of the network is defined as

$$E = \frac{1}{2} \sum_i (t_i - y_{2,i})^2 \, . \tag{3}$$

The goal is to minimize this error. This minimization is done via changing the weights between the neurons. For this purpose, the back-propagation is used, which is an algorithm minimizing the error by a gradient method. The weights are iteratively changing until the error of the network is lower than a chosen threshold (or until a number of iterations is reached). The computation of the new weights is different for the output layer and for the hidden layers (the derivation of the formulas is in [1]). For the output layer, the error of the neuron is computed as

$$d_{k,i} = (t_i - y_{k,i}) \, \lambda \, y_{k,i} \, (1 - y_{k,i}) \, . \tag{4}$$

For the hidden layer, the error of a neuron i in a layer k is influenced by the errors of the neurons j from the layer $(k+1)$. Therefore, the error is computed as

$$d_{k,i} = \Big(\sum_j d_{k+1,j} \, w_{k,j,i} \Big) \lambda \, y_{k,i} \, (1 - y_{k,i}) \, . \tag{5}$$

The weights in the network are then changed by

$$\Delta w_{k,i,j} = \eta \, d_{k+1,i} \, y_{k,i} \, . \tag{6}$$

Your task

The neural network is prepared in the template files **backprop.cpp** and **backprop.h**. All the variables are prepared and are called in the same way as in this text (and also as in Figure 1. Your task is to create the methods **feedforward** and **backpropagation**. The **feedforward** method takes the input vector (**y[0]**), passes it through the network and sets the output (**y[2]**). You will use Eqs. (1), (2). In the **backpropagation** method, compute the error of the network using Eq. (3), and set the new weights between the neurons using Eqs.

(4), (5), (6). Train your neural network to classify the objects (square, rectangle, star, circle) using the features $F1$-$F3$ obtained from the training image. Afterwards, test your neural network to classify the objects in the testing image.