Training Multi-Layer Neural Networks - the Back-Propagation Method

(c) Marcin Sydow

Plan

- training single neuron with continuous activation function
- training 1-layer of continuous neurons
- training multi-layer network - back-propagation method

Training single neuron with continuous activation function

Reminder: neuron with continuous activation function

- sigmoid unipolar activation function:

$$f(net) = \frac{1}{1 + e^{-net}}$$

- sigmoid bipolar activation function:

$$f(net) = \frac{2}{1 + e^{-net}} - 1$$

where:

$$net = w^T x$$

Error of continuous neuron

Let's define the following error measure for a single neuron:

$$E = \frac{1}{2}(d - y)^2 = \frac{1}{2}(d - f(w^T x))^2$$

where:

- d - desired output (continuous)
- y - actual output (continuous) (y = f(net))

(coefficient 1/2 is selected for simplification of subsequent computations)

Training goal: minimisation of the error

We wish to modify the weight vector w so that the error E is minimised.

Gradient Method: the direction of maximum descent of the function (towards the minimum) is opposite to the gradient vector (of partial derivatives of the error as the function of weight vector)

$$\nabla E(w) = \frac{\partial E}{\partial w}$$

$$\nabla E(w) = -(d - y)f'(net)(\frac{\partial net}{\partial w_1}, \ldots, \frac{\partial net}{\partial w_p})^T =$$

$$= -(d - y)f'(net)x$$

Derivatives of sigmoid functions

Let's observe that:

- for unipolar sigmoid function:

$$f'(net) = f(net)(f(net) - 1) = y(y - 1)$$

- for bipolar sigmoid function:

$$f'(net) = \frac{1}{2}(1 - f^2(net)) = \frac{1}{2}(1 - y^2)$$

Thus, the derivative of f can be easily expressed in terms of itself.

(Now, we can understand why such particular form of activation function was selected)

Learning rule for a continuous neuron

To sum up, the weights of a continuous neuron are modified as follows:

- unipolar:

$$w_{new} = w_{old} + \eta(d - y)y(1 - y)x$$

- bipolar:

$$w_{new} = w_{old} + \frac{1}{2}\eta(d - y)(1 - y^2)x$$

where: η is the *learning rate* coefficient

Notice a remarkable analogy to the delta rule for discrete perceptron

Training 1-layer of neurons with continuous activation function

1-layer of neurons with continuous activation

Assume, the network has J inputs and K continuous neurons.

Let's introduce the following denotations:

- input vector: $y^T = (y_1, \ldots, y_J)$
- output vector: $z^T = (z_1, \ldots, z_K)$
- weight matrix: $W = [w_{kj}]$ (w_{kj}: k-th neuron, j-th weight)
- matrix of activation funtions: $\Gamma = diag[f(\cdot)]$ (size: $K \times K$)

Computing the output vector is as follows:

$$z = \Gamma[Wy]$$

Training 1-layer of neurons with continuous activation funtions

Let's introduce additional denotations:

- desired output vector: $d^T = (d_1, \ldots, d_K)$
- output error for a single input vector:

$$E = \frac{1}{2}\sum_{k=1}^{K}(d_k - z_k)^2 = \frac{1}{2}||d - z||^2$$

Again, the gradient method will be applied (as in the case of a single neuron)

Modification of a single weight is as follows:

$$\Delta w_{kj} = -\eta \frac{\partial E}{\partial w_{kj}}$$

Training 1-layer NN, cont.

Thus, we obtain:

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial net_k} \frac{\partial net_k}{\partial w_{kj}}$$

error signal delta of the k-th neuron of the last layer:

$$\delta_{zk} = -\frac{\partial E}{\partial net_k} = (d_k - z_k) z_k (1 - z_k)$$

$$\delta_{zk} = -\frac{\partial E}{\partial net_k} = \frac{1}{2}(d_k - z_k)(1 - z_k)^2$$

Notice that: $\frac{\partial net_k}{\partial w_{kj}} = y_j$

We get the following matrix weight modification formula:

$$W_{new} = W_{old} + \eta \delta_z y^T$$

Algorithm for training 1-layer NN

- select η, E_{max}, initialise random weights W, $E = 0$
- for each case from the training set:
  - compute the output z
  - mofify the weight of the k-th neuron (unipolar/bipolar):

$$w_k \leftarrow w_k + \eta(d_k - z_k)z_k(1 - z_k)y$$

$$w_k \leftarrow w_k + \frac{1}{2}\eta(d_k - z_k)(1 - z_k^2)y$$

  - accumulate the error:

$$E \leftarrow E + \frac{1}{2}\sum_{k=1}^{K}(d_k - z_k)^2$$

- if all training cases where considered and $E < E_{max}$ then complete the training phase. Else, reset the error E and repeat training on the whole training set.

Training multi-layer network - Back-propagation method

Multi-layer network

1-layer NN can split the input space into linearly separable regions.

Each next layer can further transform the space.

As the result, multi-layer network is a universal tool that theoretically can arbitrarily well approximate any transformation of the input space into output space.

Training of a multi-layer network

We will illustrate training of a multi-layer network on a 2-layer example. To this end we will prepend one additional layer in the front of the output layer and will demonstrate how to train it.

Each layer except the output one is called "hidden", since it is not known what is the "correct" output vector of such a layer.

A method for training multi-layer networks was discovered not earlier than in 70s and it was applied since 80. of the XX-th century. It is known as the **back-propagation method**, since the weights are modified "backwards", starting from the last layer.

The method can be naturally extended from 2 layers to any number of hidden layers.

2-layer neural network

Let's introduce the following denotations:

- input vector: $x^T = (x_1, \ldots, x_I)$
- weight matrix of the 1st layer: $V = [v_{ji}]$
  (v_{ji}: j-th neuron, i-th weight)
- output vector of the first layer (input to the 2nd layer):
  $y^T = (y_1, \ldots, y_J)$
- output vector of the 2nd layer (final output):
  $z^T = (z_1, \ldots, z_K)$
- weight matrix of the 2nd layer: $W = [w_{kj}]$
  (w_{kj}: k-th neuron, j-th weight)
- activation function operator: $\Gamma = diag[f(\cdot)]$
  (size: $J \times J$ lub $K \times K$)

Computing the final output can be expressed in matrix form as:

$$z = \Gamma[Wy] = \Gamma[W\Gamma[Vx]]$$

Training multi-layer network

Back-propagation method:

After computing the output vector z, the weights are modified starting from the last layer towards the first one (backwards)

Earlier, it was demonstrated how to modify the weights of the last layer.

After modifying the weights of the last layer, the weights of the first layer are modified.

We again apply the gradient method to modify the weights of the first (hidden) layer:

$$\Delta v_{ji} = -\eta \frac{\partial E}{\partial v_{ji}}$$

Backpropagation method, cont.

By analogy, the weight matrix V is modified as follows:

$$V_{new} = V_{old} + \eta \delta_y x^T$$

where, δ_y denotes *error signal vector* of the hidden layer:

$$\delta_y^T = (\delta_{y_1}, \ldots, \delta_{y_J})$$

Error signal of the hidden layer is computed as follows:

$$\delta_{yj} = -\frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial net_j} = -\frac{\partial E}{\partial y_j} \cdot f'(net_j) = \sum_{k=1}^{K} \delta_{zk} w_{kj} \cdot f_j'(net_j)$$

Algorithm for training multi-layer neural network

- select η, E_{max}, initialise randomly the weights W and V, $E = 0$
- for each case from the training set:
  - compute output vectors y and z
  - accumulate the error: $E \leftarrow E + \frac{1}{2}\sum_{k=1}^{K}(d_k - z_k)^2$
  - compute the error signals (for the last and first layer):
    unipolar:
    $\delta_{zk} = (d_k - z_k)z_k(1 - z_k),\ \delta_{yj} = y_j(1 - y_j)\sum_{k=1}^{K}\delta_{zk}w_{kj}$
    bipolar:
    $\delta_{zk} = \frac{1}{2}(d_k - z_k)(1 - z_k^2),\ \delta_{yj} = \frac{1}{2}(1 - y_j^2)\sum_{k=1}^{K}\delta_{zk}w_{kj}$
  - modify the weights of the last layer:
    $w_{kj} \leftarrow w_{kj} + \eta\delta_{zk}y_j$
  - modify the weights of the first (hidden) layer:
    $v_{ji} \leftarrow v_{ji} + \eta\delta_{yj}x_i$
- if all the cases from the training set were presented and $E < E_{max}$ then complete the training phase. Else, reset the error E and repeat training on the whole training set

Summary

- training single neuron with continuous activation function
- training 1-layer of continuous neurons
- training multi-layer network - back-propagation method

Thank you for attention.