TDA231 - Algorithms for Machine Learning & Inference

Chalmers University of Technology

LP3– 2017-02-20

By:
Karabo Ikaneng, 891024-8239, kara.ikaneng@gmail.com
Elias Svensson, 920406-3052, elias.svensson.1992@gmail.com

Goal

Feed-forward neural networks

1 Theoretical problems

1.1 Topological properties, 2 points

a

In order to be able to perform backpropagation, the network has to fulfill the criteria for a feed forward netowrk. The only neural network that fulfils this is in (a).

b

The cost function of the neural network must be differentiable with respect the weights of the network in order to use backpropogation. The network must also satisfy the criteria of the feedforward neural network, which is information only propagates in one direction, from the input nodes, through hidden nodes (if any) and finally to the output nodes. This is from left to right when observing the given figures. There must also be no cycles or loops in the network.

1.2 Committee, 2 points

Yes this is possible since the function has a linear activation, the derivation is given below.

$$g(w_1, w_2) = \frac{1}{2}\mathbf{w_1}^{\mathbf{T}}\mathbf{x} + \frac{\mathbf{1}}{\mathbf{2}}\mathbf{w_2}^{\mathbf{T}}\mathbf{x} = \frac{\mathbf{1}}{\mathbf{2}}\mathbf{x}(\mathbf{w_1}^{\mathbf{T}} + \mathbf{w_2}^{\mathbf{T}}) \tag{1}$$

Where:

$$w_3(w_1, w_2) = \mathbf{w_1}^{\mathbf{T}} + \mathbf{w_2}^{\mathbf{T}}$$

Therefore from equation 1

$$g(w_1, w_2) = \frac{1}{2}\mathbf{w_3}^{\mathbf{T}}$$

1.3 Backpropagation, shallow network, 2 points

The gradient decent, with learning rate α is given as:

$$\Delta w_i = -\alpha \frac{\partial E(w_i)}{\partial w_i}$$

Where:

$$\alpha \frac{\partial E(w_i)}{\partial w_i} = \frac{1}{2} \frac{\partial E}{\partial w_i} (h_w(x) - y)^2$$

$$\alpha \frac{\partial E(w_i)}{\partial w_i} = 2.\frac{1}{2} \frac{\partial E}{\partial w_i} (h_w(x) - y)^2 \frac{\partial E}{\partial w_i} (-\mathbf{w_1}^{\mathbf{T}} \mathbf{x})$$

Therefore:

$$\Delta w_i = \alpha (h_w(x) - \mathbf{w_1}^{\mathbf{T}} \mathbf{x}) \mathbf{x}$$

1.4 Backpropagation, 4 points

a

$$\frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial z_k} = \frac{\partial E}{\partial y_k} g'(z_k) \tag{1}$$

b

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial z_j} = \frac{\partial E}{\partial y_j} g'(z_j) \tag{2}$$

$$\frac{\partial E}{\partial y_j} = \frac{\partial E}{\partial z_k} \frac{\partial z_k}{\partial y_j} = \frac{\partial E}{\partial z_k} \left[\frac{\partial}{\partial y_j} (\sum w_{jk} y_j) \right] = \left[\frac{\partial E}{\partial z_k} \right] \sum w_{jk} y_j \tag{3}$$

Therefore substituting (3) into (2) and $\frac{\partial E}{\partial y_k}$

$$\frac{\partial E}{\partial z_j} = \left[\frac{\partial E}{\partial z_k} \sum w_{jk} y_j \right] g'(z_j) \tag{4}$$

c

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial z_k} \frac{\partial z_k}{\partial w_{jk}} \tag{5}$$

Substitute (1) into into (5) and the definition of z_k

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial y_k} g'(z_k) \left[\frac{\partial}{\partial w_{jk}} (\sum w_{jk} y_j) \right] \tag{6}$$

Therefore:

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial y_k} g'(z_k)\delta_k y_j \tag{7}$$

d

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}} \tag{8}$$

Therefore substituting (4) into 8 and the definition of z_j

$$\frac{\partial E}{\partial w_{ij}} = \left[\frac{\partial E}{\partial z_k}\sum w_{jk} y_j\right] g'(z_j) \left[\frac{\partial}{\partial w_{ij}}(\sum w_{ij} y_i)\right] \tag{9}$$

Therefore:

$$\frac{\partial E}{\partial w_{ij}} = \left[\frac{\partial E}{\partial z_k}\sum w_{jk} y_j\right] g'(z_j)\delta_j y_i \tag{10}$$

2 Practical Problems

2.1 Backpropagation on paper, 3 points

The cross-entropy error gradient with respect to the Softmax input (z) can be considered as given,

$$\frac{\partial E^{Classification}}{\partial z} = \text{prediction} - \text{target} \tag{11}$$

The total error is

$$E = E^{Classification} + \alpha E^{Weightdecay} \tag{12}$$

The partial derivative of the error function with respect to a weight is

$$\frac{\partial E}{\partial w} = \frac{\partial E^{Classification}}{\partial w} + \alpha \frac{\partial E^{Weightdecay}}{\partial w} \tag{13}$$

The partial derivative of the error function with respect to weight w_{jk} is

$$E^{Classification} = \frac{1}{2} \sum_{k \in K} (a_k - t_k)^2 \tag{14}$$

$$E^{Weightdecay} = \alpha \sum_j \sum_k ||w_{ij}||^2 \tag{15}$$

$$E = \frac{1}{2} \sum_{k \in K} (a_k - t_k)^2 + \alpha \sum_j \sum_k ||w_{ij}||^2 \tag{16}$$

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial z_k} \frac{\partial z_k}{\partial w_{jk}} + \alpha \frac{\partial}{\partial w_{jk}} ||w_{jk}||^2 = (a_k - t_k) g'_k(z_k) a_j + \alpha 2 w_{jk} = \delta_k a_j + \alpha 2 w_{jk} \tag{17}$$

where

$$\delta_k = (a_k - t_k) g'_k(z_k) \tag{18}$$

The partial derivative of the error function with respect to weight w_{ij} is

$$\frac{\partial E^{Classification}}{\partial w_{ij}} = \sum_{k \in K} (a_k - t_k) g'_k(z_k) w_{jk} g'_j(z_j) a_i = \tag{19}$$

$$g'_j(z_j) a_i \sum_{k \in K} (a_k - t_k) g'_k(z_k) w_{jk} = a_i g'_j(z_j) \sum_{k \in K} \delta_k w_{jk} = \delta_j a_i$$

where

$$\delta_j = g'_j(z_j) \sum_{k \in K} \delta_k w_{jk} \tag{20}$$

$$\frac{\partial E^{Weightdecay}}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} ||w_{ij}||^2 = 2\alpha w_{ij} \tag{21}$$

$$\frac{\partial E}{\partial w_{ij}} = \delta_j a_i + 2\alpha w_{ij} \tag{22}$$

2.2 Backpropagation, 2 points

Running net(0.1, 7, 10, 0, 0, false, 4), the cost on the training data is 2.768381.

Code added to function res = grad(model, data, wd_coefficient):

```
    %% TODO - Write code here ---------------
    % Calculate number of inputs
    [number_inputs, number_classes] = size(data.inputs);

    % Calculate error classifier for wjk
    error_classifier_wjk = (class_prob-data.targets) * hid_output' ...
        / number_classes;

    % Calculate error weightdecay for wjk
    error_weightdecay_wjk = model.hid_to_class * wd_coefficient;

    % Calculate total error for wjk
    error_total_wjk = error_classifier_wjk + error_weightdecay_wjk;

    % Return error to res
    res.hid_to_class = error_total_wjk;

    % Calculate error classifier for wij
    error_classifier_wij = (((class_prob - data.targets)' * model.hid_to_class)' ...
        .* (hid_output - hid_output.^2)) * data.inputs' / number_classes;

    % Calculate error weightdecay for wij
    error_weightdecay_wij = model.input_to_hid * wd_coefficient;

    % Calculate total error for wij
    error_total_wij = error_classifier_wij + error_weightdecay_wij;

    % Return error to res
    res.input_to_hid = error_total_wij;
    % ---------------------------------------
```

2.3 Optimization, 2 points

a) The best run is with momentum = 0.9.

b) The best learning rate out of the 14 runs is: $\alpha = 0.2$.

2.4 Generalization, 3 points

a) The cost on the validation data is 0.430185.

b) The cost on the validation data is 0.334505.

c) The best run is with wd_coefficient = 0.001. The classification cost (i.e. without weight decay) on the validation data is 0.287910.

d) The best number of hidden units is 30. The cost on the validation data is 0.317077.

e) The best number of hidden units is 37. The cost on the validation data is 0.265165.

f) Running net(0.001, 37, 1000, 0.2, 0.9, true, 100), the classification error rate on the test data is 0.073222.