Report HW3

Armand Ghaffarpour, 9101103738, armandg@student.chalmers.se
Ryan Damarputra Widjaja, 9002205616, ryand@student.chalmers.se

7th May 2018

1 Theoretical Problems

1.1 Topological Properties

a. Network (1) and Network (3) can be trained using backpropagation algorithm

b. The network must be a feed-forward neural network. That is, they don't contain any cycles between the layers

1.2 Committee

Answer: Yes.
Proof

Given that the prediction from the two networks can be described as :

$$y_B = \frac{1}{2}\mathbf{w}_1^T\mathbf{x} + \frac{1}{2}\mathbf{w}_2^T\mathbf{x}$$

We can arrange the equation above as :

$$y_B = (\frac{1}{2}\mathbf{w}_1^T + \frac{1}{2}\mathbf{w}_2^T)\mathbf{x}$$

Now if we want to use a single feed forward network with no hidden layers, and assuming that w_3 is the weight for our network, then the prediction formula will become :

$$y = \mathbf{w}_3^T\mathbf{x}$$

This function is similar to Brian's function. Now let $\mathbf{w}_3$ be :

$$\mathbf{w}_3 = f(\mathbf{w}_1, \mathbf{w}_2) = (\frac{1}{2}\mathbf{w}_1^T + \frac{1}{2}\mathbf{w}_2^T)$$

Then our function becomes :

$$y = \mathbf{w}_3^T\mathbf{x}$$

$$y = (\frac{1}{2}\mathbf{w}_1^T + \frac{1}{2}\mathbf{w}_2^T)\mathbf{x}$$

So we get: $y_B = y \quad \square$

1.3 Backpropagation - shallow network

We know the following:

$$y = \mathbf{w}^T\mathbf{x}$$

$$w_i := w_i - \lambda\frac{\partial E}{\partial w_i}$$

$$E = \frac{1}{2}(t-y)^2$$

We calculate the following:

$$\frac{\partial E}{\partial w_i} = \frac{2}{2}(t - \mathbf{w}^T\mathbf{x}) * (-x_i) = x_i(\mathbf{w}^T x - t)$$

Answer: $w_i := w_i - \lambda(x_i(\mathbf{w}^T x - t))$

1.4 Backpropagation

Before we write the answer, we will explain the notation that we use. We will use the letter i, j, k for identifying the layer, with layer i as the input layer, layer j as the hidden layer, and k as the output layer. For the weights, we will use this notation : $w_{ij}^{(jk)}$, which means this is the weight of the channel from layer j to layer k, from the i-th node in the layer j to the j-th node in the layer k. Lastly, for $z_i^{(j)}$ means the i-th node in the layer j.

Answers:

a.

$$\frac{\partial E}{\partial z_i^{(k)}} = \frac{\partial E}{\partial y_i^{(k)}} * \frac{\partial y_i^{(k)}}{\partial z_i^{(k)}}$$

b.

$$\frac{\partial E}{\partial z_i^{(j)}} = \frac{\partial E}{\partial y_i^{(j)}} * \frac{\partial y_i^{(j)}}{\partial z_i^{(j)}}$$

where:

$$\frac{\partial E}{\partial y_i^{(j)}} = \sum_a \left(\frac{\partial E}{\partial z_a^{(k)}} * \frac{\partial z_a^{(k)}}{\partial y_i^{(j)}} \right)$$

c.

$$\frac{\partial E}{\partial w_{ij}^{(jk)}} = \frac{\partial E}{\partial y_i^{(k)}} * \frac{\partial y_i^{(k)}}{\partial z_i^{(k)}} * \frac{\partial z_i^{(k)}}{\partial w_{ij}^{(jk)}} = \frac{\partial E}{\partial z_i^{(k)}} * y_j^{(j)}$$

d.

$$\frac{\partial E}{\partial w_{ij}^{(ij)}} = \frac{\partial E}{\partial z_i^{(j)}} * \frac{\partial z_i^{(j)}}{\partial w_{ij}^{(ij)}} = \frac{\partial E}{\partial z_i^{(j)}} * y_i^{(i)}$$

2 Practical Problems

2.1 Backpropagation on paper

The error formula is $E = E^{Classification} + E^{weightdecay}$, so naturally we will carry the derivation of the Error formula all the way from the output layer to the input layer. We will denote the $E^{Classification}$ as E^C and $E^{weightdecay}$ as E^{WD}

Firstly, we will clarify some notation that we will be using for the derivation. We will use the letter i, j, k for identifying the layer, with layer i as the input layer, layer j as the hidden layer, and k as the output layer. For the weights, we will use this notation : $w_{ij}^{(jk)}$, which means this is the weight of the channel from layer j to layer k, from the i-th node in the layer j to the j-th node in the layer k. Lastly, for $z_i^{(j)}$ means the i-th node in the layer j.

For clarity, the functions involved in the neural network are similar to what the Problem 1.4 describes :

$$z_i^{(i)} = \text{this is the input data}$$

$$z_i^{(j)} = \sum_i w_{ij}^{(ij)} z_i^{(i)}$$

$$y_i^{(j)} = \sigma(z_i^{(j)})$$

$$z_i^{(k)} = \sum_i w_{ij}^{(jk)} y_i^{(j)}$$

$$y_i^{(k)} = softmax(z_i^{(k)})$$

First, we will start with the starting incoming gradient of the classification error $\frac{\partial E^C}{\partial y_i^{(k)}}$. By chain rule, we can calculate the $\frac{\partial E^C}{\partial z_i^{(k)}}$ as follows :

$$\frac{\partial E^C}{\partial z_i^{(k)}} = \frac{\partial E^C}{\partial y_i^{(k)}} \frac{\partial y_i^{(k)}}{\partial z_i^{(k)}} = prediction - target$$

The $prediction - target$ is already given in the description. Next in the hidden layer, we can calculate $\frac{\partial E^C}{\partial y_i^{(j)}}$ as follows :

$$\frac{\partial E^C}{\partial y_i^{(j)}} = \sum_a^{10} \frac{\partial E^C}{\partial z_a^{(k)}} \frac{\partial z_a^{(k)}}{\partial y_i^{(j)}} = \sum_a^{10} \frac{\partial E^C}{\partial z_a^{(k)}} w_{ia}^{(jk)}$$

Here we have the sum because for every node in the hidden layer, they are connected to 10 output nodes (because as stated in the description, the output layer are 10-way softmax)

For the $\frac{\partial E^C}{\partial w_{ij}^{(jk)}}$, we can compute it as follows :

$$\frac{\partial E^C}{\partial w_{ij}^{(jk)}} = \frac{\partial E^C}{\partial z_i^{(k)}} \frac{\partial z_i^{(k)}}{\partial w_{ij}^{(jk)}} = \frac{\partial E^C}{\partial z_i^{(k)}} y_i^{(j)}$$

Still in the hidden layer, we can calculate $\frac{\partial E^C}{\partial z_i^{(j)}}$ as follows :

$$\frac{\partial E^C}{\partial z_i^{(j)}} = \frac{\partial E^C}{\partial y_i^{(j)}} \frac{\partial y_i^{(j)}}{\partial z_i^{(j)}} = \frac{\partial E^C}{\partial y_i^{(j)}} (\sigma(z_i^{(j)}) - (\sigma(z_i^{(j)}))^2)$$

Finally in the input layer, based on the formula above, we can calculate $\frac{\partial E^C}{\partial w_{ij}^{(ij)}}$ as follows :

$$\frac{\partial E^C}{\partial w_{ij}^{(ij)}} = \frac{\partial E^C}{\partial z_i^{(j)}} \frac{\partial z_i^{(j)}}{\partial w_{ij}^{(ij)}} = \frac{\partial E^C}{\partial z_i^{(j)}} z_i^{(i)}$$

Since the Error function is $E = E^{Classification} + E^{WeightDecay}$, that means :

$$\frac{\partial E}{\partial w_{ij}^{(jk)}} = \frac{\partial E^C}{\partial w_{ij}^{(jk)}} + \frac{\partial E^{WD}}{\partial w_{ij}^{(jk)}}$$

$$\frac{\partial E}{\partial w_{ij}^{(ij)}} = \frac{\partial E^C}{\partial w_{ij}^{(ij)}} + \frac{\partial E^{WD}}{\partial w_{ij}^{(ij)}}$$

The E^{WD} functions are already given in the MATLAB as follows :

$$E^{WD(ij)} = \frac{\alpha}{2} \sum (w_{ij}^{(ij)})^2$$

$$E^{WD(jk)} = \frac{\alpha}{2} \sum (w_{ij}^{(jk)})^2$$

By deriving both E^{WD} function with respects to the weights, the error function derivation becomes :

$$\frac{\partial E}{\partial w_{ij}^{(jk)}} = \frac{\partial E^C}{\partial w_{ij}^{(jk)}} + \alpha * w_{ij}^{(jk)}$$

$$\frac{\partial E}{\partial w_{ij}^{(ij)}} = \frac{\partial E^C}{\partial w_{ij}^{(ij)}} + \alpha * w_{ij}^{(ij)}$$

2.2 Backpropagation

This is the implementation that we did in MATLAB :

```
d_EClass_by_d_zk = class_prob - data.targets;
hid_output_transpose = transpose(hid_output);
d_EClass_by_d_hid_to_class = (d_EClass_by_d_zk * hid_output_transpose);

d_EClass_by_d_hid_node = transpose(class_prob-data.targets) * model.hid_to_class;
d_logistic_by_d_zj = logistic(hid_input) - (logistic(hid_input)^2);

d_EClass_by_d_zj = transpose(d_EClass_by_d_hid_node) .* d_logistic_by_d_zj;
d_EClass_by_d_input_to_hid= d_EClass_by_d_zj * transpose(data.inputs);

d_Ewd_input_to_hid = wd_coefficient * model.input_to_hid;
d_Ewd_hid_to_class = wd_coefficient * model.hid_to_class;

[m,n] = size(data.inputs);

d_E_by_d_input_to_hid = (d_EClass_by_d_input_to_hid/n) + d_Ewd_input_to_hid;
d_E_by_d_hid_to_class = (d_EClass_by_d_hid_to_class/n) + d_Ewd_hid_to_class;

res.input_to_hid = d_E_by_d_input_to_hid;
res.hid_to_class = d_E_by_d_hid_to_class;
```

Here we are using matrix multiplication, so the sums in the Problem 2.1 are already taken care of by matrix multiplication, and we did some transpose transformation in order to make the inner dimensions of each matrix agree with each other when we are performing matrix multiplication.

However since the classification loss takes the mean of every loss, we will divide our ∂E^C value by the number of the data in order to get the mean.

With these implementation, we have a training cost of 2.768381

2.3 Optimization

a. The best run is a run with a momentum of 0.9

b. The learning rate of this best run is 0.2

2.4 Generalization

a. The validation data cost for this run is 0.430185

b. With early stopping, the validation data cost for this run is 0.334505, with validation cost was at its lowest after 161 iterations

c. If we are looking for the best classification cost without weight decay, then the best one is 0.287910. This run uses 0.001 as the weight decay coefficient.

d. With 30 hidden layers, we have the lowest validation data for this experiment : 0.317077

e. The run with 37 hidden layers combined with early stopping gives us the best validation cost for this experiment : 0.265165

f. Here, we are using 37 hidden layers, 0.001 weight decay, and combine that with early stopping (net(0.001, 37, 1000, 0.35, 0.9,true, 100)). This gives us 0.071778 as the test data classification error rate value