Homework set 3 TDA231 Algorithms for Machine Learning & Inference

Kevin Jaquier 921119T334
Sandra Viknander 9003012482

May 2018

1 Theoretical problems

1.1 Topological properties

- Backpropagation can be used on networks 1 and 3.
- Network 2 and 4 the regular version of backpropagation can not be used sins you have recursion in the network. However you can use a slightly modified version, backpropagation trough time where you keep feed the same pattern to the network until you have a stable output. Depending on the inputs you may or may not get stable outputs from these two networks which is required for backpropagation trough time.
- The basic backpropagation algorithm requires a non-recurrent neural network, or more generally, a directed and acyclic computational graph. Though there are extensions that allows the algorithm to be performed on recurrent networks too as described for network 2 and 4. However the output must be stable trough time for this version of backpropagation to work.

1.2 Commitee

We use the fact that $a^T b$ is equal to the dot product $a \cdot b$.

$$
\begin{aligned}
y &= \frac{1}{2}(w_1^T x + w_2^T x) \\
&= \frac{1}{2}(x^T w_1 + x^T w_2) && \text{(commutativity)} \\
&= \frac{1}{2} x^T (w_1 + w_2) && \text{(distributivity)} \\
&= \frac{1}{2}(w_1 + w_2)^T x && \text{(commutativity)} \\
&= w_3^T x && \text{with } w_3 = \frac{1}{2}(w_1 + w_2)
\end{aligned}
$$

1.3 Backpropagation - shallow network

Each iteration, the algorithm performs the following operation:

$$w_i \leftarrow w_i - \lambda \frac{\partial E}{\partial w_i}$$

Given $E = \frac{1}{2}(t-y)^2$ and $y = w^T x$, we have:

$$\begin{aligned} E &= \frac{1}{2}(t - w^T x)^2 \\ \frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}(t - w^T x)^2 \\ &= \frac{1}{2}2(t - w^T x)\frac{\partial}{\partial w_i}(t - w^T x) \\ &= (t - w^T x)(0 - x_i) \end{aligned}$$

Thus the algorithms will perform:

$$w_i \leftarrow w_i + \lambda x_i (t - w^T x)$$

1.4 Backpropagation

lets start with denoting $\frac{\partial E}{\partial y_k} = C$ for simplicity in writing.

1.4.1 a)

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

1.4.2 b)

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial z_k}\frac{\partial z_k}{\partial y_j}\frac{\partial y_j}{\partial z_j} = C * g'(z_k)\sum_j \omega_{jk} g'(z_j) \tag{2}$$

1.4.3 c)

$$\frac{\partial E}{\partial \omega_{ij}} = \frac{\partial E}{\partial z_k}\frac{\partial z_k}{\partial \omega_{jk}} = \frac{\partial E}{\partial z_k} y_j \tag{3}$$

lets define $\delta_k = \frac{\partial E}{\partial y_k} g'(z_k)$

1.4.4 d)

$$\frac{\partial E}{\partial \omega_{jk}} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial \omega_{ij}} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{z_k}\frac{\partial z_k}{\partial y_j}\frac{\partial y_j}{\partial \omega_{ij}} = \frac{\partial E}{\partial z_j} y_i = Cg'(z_k)\sum_j \omega_{jk} g'(z_j) y_i \tag{4}$$

now we can define $\delta_j = \delta_k * \sum_j \omega_{jk} g'(z_j)$

2 Practical problems

2.1 Backpropagation on paper

We start out by writing down the the functions for the different outputs of the neurons just as in the exercise 1.4. we can see these functions in equation 5 and the network architecture can be seen in figure 1. Then we recall the expressions for ω_{jk} and ω_{ij} in equations 3 and 4 noting that we only need to compute the $g'(z_k)$ and $f'(z_j)$ were $g(x)$ and $f(x)$ is the soft-max function and logistic function respectively. These respective derivatives are calculated in equation 6. We now only need to calculate the gradient term for the weight decay, sins the weight decay term is just the L^2 norm of the weights times a constant α it is easily computed as in equation 7. Putting all of this together we can calculate the gradients that we are going to use in our back propagation and is shown in equation 8. There is one thing that we have missed though we are not using the classification error instead we are using the cross entropy error. Thus we need to calculate the derivative of the cross entropy which is done in equation 9. Now setting y_k as the soft max output and the x_k as the target ξ_k we get equation 10 which also give us $\frac{\partial E_{Classification}}{\partial z_k}$ for equation 8.

$$\begin{aligned} y_i &= z_i \\ z_j &= \sum_i \omega_{ij} z_i \\ y_j &= \frac{1}{1+e^{-kz_j}} \qquad k \in \mathbf{N} \\ z_k &= \sum_j \omega_{jk} y_j \\ y_k &= \frac{e^{z_k}}{\sum_n e^{z_{k_n}}} \end{aligned} \tag{5}$$

$$\begin{aligned} g(x_i) = \frac{e^{x_i}}{\sum_n e^{x_n}} \qquad &\rightarrow \qquad g'(x_i) = \begin{cases} i = j & g(x_i)(1-g(x_j)) \\ i \neq j & -g(x_i)g(x_j) \end{cases} \\ f(x) = \frac{1}{1+e^{-kx}} \qquad &\rightarrow \qquad f'(x) = \frac{ke^{kx}}{(1+e^{kx})^2} \end{aligned} \tag{6}$$

$$\frac{\partial}{\partial \omega}\alpha\frac{1}{2}||\omega||_2^2 = \alpha||\omega|| \tag{7}$$

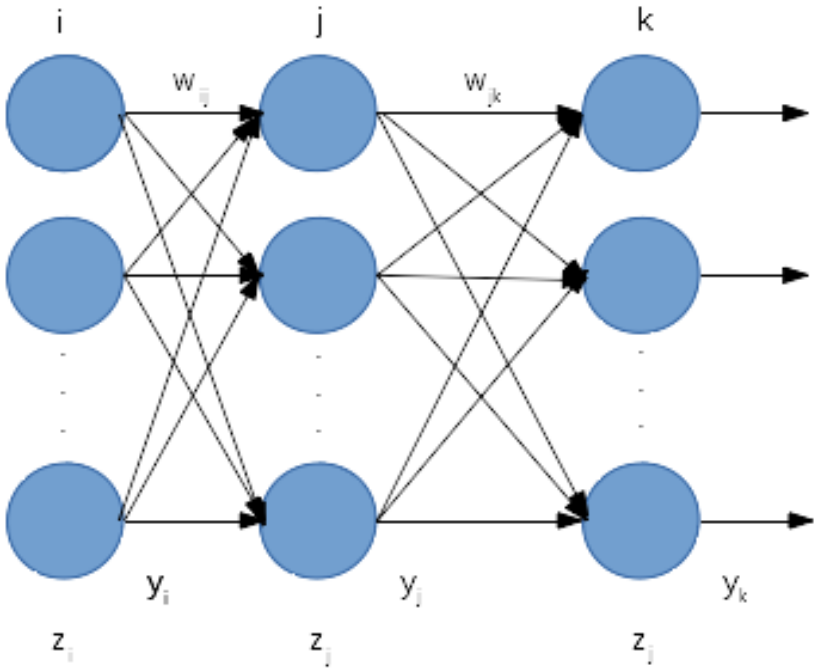

Figure 1

$$\begin{aligned}
\frac{\partial E}{\partial \omega_{jk}} &= \frac{\partial E_{Classification}}{\partial z_k} y_j + \alpha||\omega_{jk}|| : & \delta_k &= \frac{\partial E_{Classification}}{\partial z_k} \\
\frac{\partial E}{\partial \omega_{ij}} &= \delta_k \sum_j \omega_{ij} \frac{k e^{k z_j}}{(1+e^{k z_j})^2} y_i + \alpha||\omega_{ij}|| : & \delta_j &= \delta_k \sum_j \omega_{jk} \frac{k e^{k z_j}}{(1+e^{k z_j})^2}
\end{aligned} \tag{8}$$

$$\begin{aligned}
H(x,y) &= -\sum_k x_k Log(y_k) \\
\frac{\partial H(x,y)}{\partial n_i} &= -\sum_k x_k \frac{\partial Log(y_k)}{\partial y_k} * \frac{\partial y_k}{\partial n_i} = & &: Chainrule \\
& -\sum_k x_k \frac{1}{y_k} * \frac{\partial y_k}{\partial n_i}
\end{aligned} \tag{9}$$

$$\begin{aligned}
& -\xi_i(1-O_i) - \sum_{k \neq i} \xi_k \frac{1}{O_k}(-O_k O_i) = \\
& O_i \sum_k \xi_k - \xi_i = & & \sum_k \xi_k = 1 \\
& O_i - \xi_i
\end{aligned} \tag{10}$$

2.2 Programming - backpropagtion

2.2.1 Code

```
n_samples = size(data.inputs, 2);
n_input = size(data.inputs, 1);
n_hid = size(model.hid_to_class, 2);
n_out = size(model.hid_to_class, 1);

% W_jk gradient calculated in steps
err = class_prob - data.targets;
hid_to_class_cost = zeros(n_out, n_hid, n_samples);
for s = 1:n_samples
  hid_to_class_cost(:,:,s) = err(:,s) * hid_output(:,s)';
end

% W_ij gradient calculated in steps
d_logistic = hid_output .* (1 - hid_output);
temp = err' * model.hid_to_class;
delta_j = d_logistic .* temp';
input_to_hid_cost = zeros(n_hid, n_input, n_samples);
for s = 1:n_samples
    input_to_hid_cost(:,:,s) = delta_j(:,s) * data.inputs(:,s)';
end

% Putting everything together
input_to_hid_l2 =  wd_coefficient .* model.input_to_hid;
hid_to_class_l2 =  wd_coefficient .* model.hid_to_class;
res.input_to_hid = mean(input_to_hid_cost, 3) + input_to_hid_l2;
res.hid_to_class = mean(hid_to_class_cost, 3) + hid_to_class_l2;
```

2.2.2 Problem 2.2

```
net(0.1, 7, 10, 0, 0, false, 4)
```

Training data cost: 2.768381481704748

2.3 Experimentation

2.3.1 Optimization (Problem 2.3)

The best run was with momentum (0.9). The learning rate was 0.2 and training data cost 1.08342860306662.

2.3.2 Generalization (Problem 2.4)

Part a)

```
net(0, 200, 1000, 0.35, 0.9,false, 100)
```

Validation data cost: 0.43019

Part b)

```
net(0, 200, 1000, 0.35, 0.9,true, 100)
```

Validation data cost: 0.3345

Part c)

```
net(var, 200, 1000, 0.35, 0.9,false, 100)
```

Weight decay: var	Validation data cost:
10	22.6128
1	2.3026
0.0001	0.3483
0.001	0.2879
5	2.3026

Table 1: Table with five different values of the weight decay and the respective validation cost associated with each value

The lowest validation cost were 0.2879 and obtained whit a weight decay of 0.001 as seen in table 1

Part d)

```
net(0, var, 1000, 0.35, 0.9,false, 100)
```

Number of neurons in hidden layer: var	Validation data cost:
10	0.4217
30	0.3171
100	0.3686
130	0.3976
200	0.4302

Table 2: Table with five different numbers of hidden neurons in the hidden layer and the respective validation cost associated with each value

The lowest validation cost were 0.3171 and obtained whit a hidden layer with 30 neurons as seen in table 3

Part e)

```
net(0, var, 1000, 0.35, 0.9,true, 100)
```

Number of neurons in hidden layer: var	Validation data cost:
18	0.3061
37	0.2652
83	0.3112
113	0.3137
236	0.3438

Table 3: Table with five different numbers of hidden neurons in the hidden layer and the respective validation cost associated with each value

The lowest validation cost when using early stopping were 0.2652 and obtained whit a hidden layer with 37 neurons as seen in table 3

Part f)

```
net(0.001, 37, 1000, 0.35, 0.9,true, 100)
```

combining the best parameters as found in the previous tests yield a Test data cost of 0.2571 and a error rate of 0.0732 on the test set which is a good result.