19/03/2020

The Backpropagation Algorithm

The algorithm starts by constructing a network with the required input units (dependent on the task at hand), desired number of hidden layers, desired number of nodes in each hidden layer, and the output units.

The feedforward step is performed first (as done in the last class). For each input sample, the output is noted. Since the output may have more than one node, the error associated with the input sample d can be written as

$$E_d(\vec{w}) = \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2$$

You may visualize this as

output layer

- node 1
- node 2
- ⋮
- node 10

k

Say, we were solving the digit recognition problem. Say we input a [1] and the output is

$$\begin{bmatrix} 0.98 \\ 0.02 \\ 0.03 \\ 0.15 \\ 0.05 \\ 0.01 \\ 0.70 \\ 0.20 \\ 0.30 \\ 0.05 \end{bmatrix} \quad \begin{bmatrix} o_1 \\ o_2 \\ o_3 \\ \vdots \\ o_{10} \end{bmatrix} \; o_k$$

and the desired output is

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ \vdots \\ t_{10} \end{bmatrix} \; t_k$$

So, the error of this sample is

$$E_d(\vec{w}) = \frac{1}{2} \sum_{k \in \text{output}} (t_k - o_k)$$

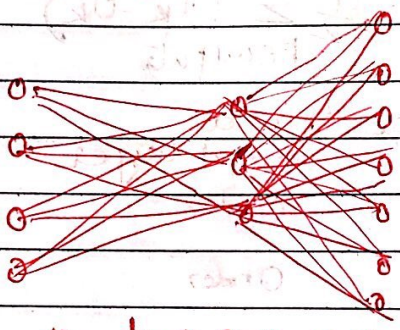
If we were to find the total error due to ALL the input samples, it would be

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2$$

where the inner summation is for each sample and outer summation is over ALL D samples.

Intuition.

Say a network has an input layer, one hidden layer and an output layer.



During the first step of choosing some random weights, we feed the input data forward and get some outputs. We also are aware of the target values that we desire.
So in essence we are aware of the error at the output layer, when a particular sample is input. We desire to somehow update the weight matrix that is between input layer and the hidden layer, as well as the weight matrix between the hidden layer and the output layer.
In the lab question given to you, I had given both these matrices to you.

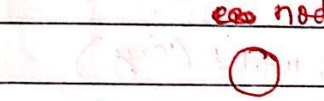
We shall again apply the stochastic gradient rule. (We did for training a linear unit).
For each training example d, every weight, w_{ji} is updated by the following Δw_{ji}

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$ (Negative to the steepest ascent)

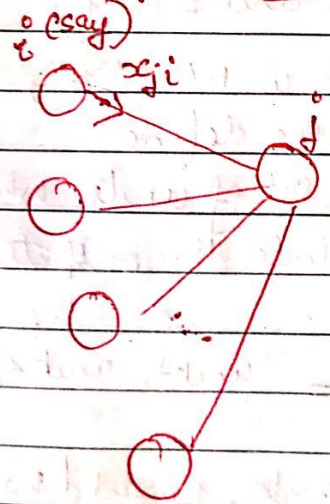
where E_d is the error on the training sample d,

$$E_d(\vec{w}) = \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2$$

Some notations that you need to know are in your book. Diagramatically they are as follows—

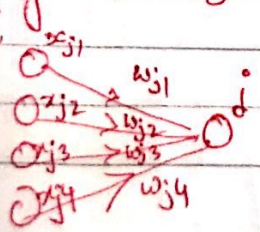


This node j may be anywhere in one of the hidden layers or in the output layer.



x_{ji} = i^{th} input to node j
w_{ji} = weight from i^{th} input to node j
$net_j = \sum_i w_{ji} x_{ji}$

You may visualize net_j as follows—



$w_{j1}x_{j1} + w_{j2}x_{j2} + w_{j3}x_{j3} + w_{j4}x_{j4}$

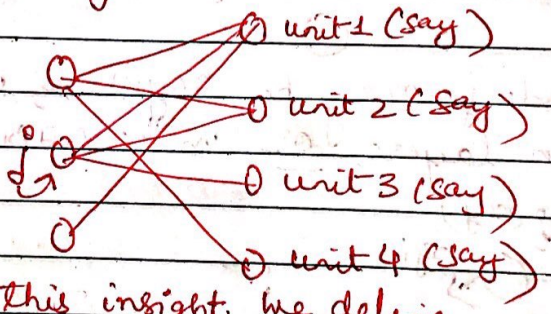
So net_j is the weighted sum of inputs to node j from the previous layer

o_j = Output computed by unit j
which is in effect
$$\sigma(net_j) = \sigma\left(\sum_i w_{ji} x_{ji}\right)$$

t_j = target output for unit j

σ = Sigmoid function (any squashing function)

Now, though not mentioned earlier, you should now understand that it is not necessary that all node of one layer are connected to all nodes of the next layer. Depending on the need and architecture, one layer may be partially connected to the next layer.



So, with this insight we define
Downstream(j) = the set of units whose immediate inputs include the outputs of unit j

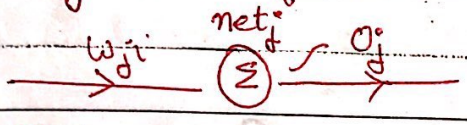
So Downstream(j) = {unit1, unit2, unit3}

In a fully connected network, j would have been connected to EACH node in the next layer.

To implement the stochastic gradient descent rule i.e $\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$ — ★

we need the value of $\frac{\partial E_d}{\partial w_{ji}}$

In a very basic form



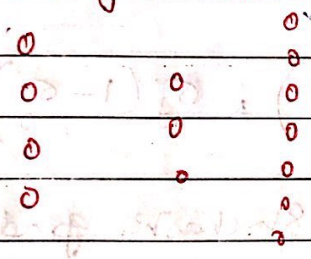
Using chain Rule

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ji}}$$

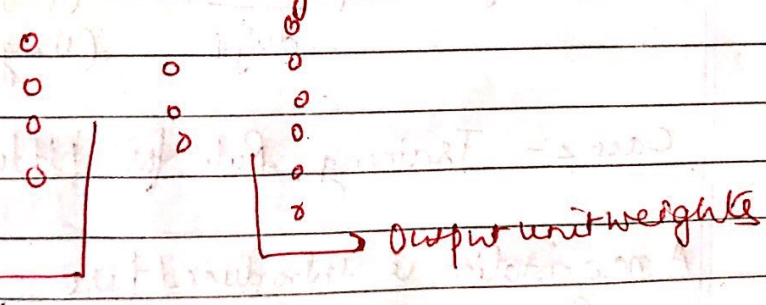
$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} \cdot x_{ji} \quad \leftarrow \text{To find} \quad -(1)$$

This value is essential for updating all the weights.

For a three layer architecture



There are 2 sets of weights
- Output unit weights
- Hidden unit weights



In a higher number of hidden layers, we will have more hidden unit weight matrices.

The training rule for Output unit weights & the training rule of Hidden Unit weights is different. So we shall handle it with 2 cases.

Case 1 - Training Rule for output Unit Weights

? $\frac{\partial E_d}{\partial net_j}$ ← To find

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \frac{\partial o_j}{\partial net_j} \quad (2) \text{ (Chain Rule Again)}$$

$$= \frac{\partial}{\partial o_j}\left[\frac{1}{2}\sum_{k \in \text{outputs}} (t_k - o_k)^2\right] * \frac{\partial}{\partial net_j}\left[\sigma(net_j)\right]$$

$= $ this derivative will be non zero only when $k = j$

$$= -(t_j - o_j) * \sigma_j(1-\sigma_j)$$

$\sigma' = \sigma(1-\sigma)$

Substituting in the Stochastic gradient descent rule equation ✱

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} = \eta(t_j - o_j)\sigma_j(1-\sigma_j) \cdot x_{ji}$$

(Using (1) and (2))

Case 2 - Training Rule for hidden Unit Weights

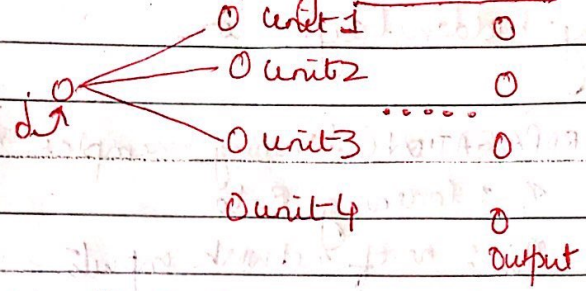
A new notation is introduced here

δ_i = error term associated with unit i

$$= -\frac{\partial E}{\partial net_i}$$

$\frac{\partial E_d}{\partial net_j}$ ← To find

Since we are in a hidden layer, each unit influences the set of units immediately downstream of unit j.



Since net_j effects E_d through downstream(j)

Therefore

$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in Downstream(j)} \frac{\partial E_d}{\partial net_k} \cdot \frac{\partial net_k}{\partial net_j} \quad \text{(Chain Rule)}$$

$$= \sum_k -\delta_k \frac{\partial net_k}{\partial net_j}$$

$$= \sum_k -\delta_k \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j} \quad \text{(Chain Rule again)}$$

$$= \sum_k -\delta_k w_{kj} \frac{\partial o_j}{\partial net_j}$$

$\because net_k = x_{k1} w_{k1} + x_{k2} w_{k2} + \cdots + w_{kj} \cdot w_{kj}$

$= o_1 w_{k1} + o_2 w_{k2} + \cdots o_j w_{kj}$

$$= \sum_k -\delta_k w_{kj} o_j (1-o_j)$$

$$= -o_j(1-o_j) \sum_k \delta_k w_{kj}$$

Also, as defined

$$\delta_j = -\frac{\partial E_d}{\partial net_j} = o_j(1-o_j) \sum_{k \in Downstream(j)} \delta_k w_{kj}$$

$$\Delta w_{ji} = \eta \delta_j x_{ji}$$

Having obtained the gradient descent weight update rule for output unit weights and for hidden unit weights, the complete Algorithm can be written as follows (Pg 98, TM). This algo is for 2 layers of Sigmoid units and can be easily extended to many hidden layers too.

BACKPROPAGATION(training_examples, η, n_{in}, n_{out}, n_{hidden})

η : learning Rate
n_{in} : no. of network inputs
n_{hidden} : no. of units in hidden layer
x_{ji} : input from unit i to unit j
w_{ji} : weight from unit i to unit j

A. Create a FF network with n_{in} inputs, n_{hidden} hidden units and n_{out} output units

B. Initialize all network weights to small random numbers

C. Until termination do

- For each $(\vec{x}, \vec{t})$ training example

Propagate the input forward through the network

1. Input $\vec{x}$ and compute o_u of every unit u

Propagate the errors backward

2. For each output unit k, calculate error term

$$\delta_k \leftarrow o_k(1-o_k)(t_k-o_k)$$

3. For each hidden unit h, calculate error term

$$\delta_h \leftarrow o_h(1-o_h)\sum_{k \in outputs} w_{kh}\delta_k$$

4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

where

$$\Delta w_{ji} = \eta \delta_j x_{ji}$$

Assignment

Understand this much. In the last lab exercise involving only FF, the 2 matrices were provided to you. On implementing, you would have realized that the weights provided to you were optimum and that the actual output was same as target output for all the training samples.

I want you to do the following now:-

Take the input & the target output as given in the questio

Randomly initialize both the weight matrices (say between -0.05 and +0.05)

Feed the inputs, observe the outputs

Check the error

In case of error, backpropagate (BP) the error and update the weights

Repeat the FF and BP till you get a satisfactory model.

Rajasekhar
19/03/2020