MACHINE LEARNING CS60050

Instructor: Dr. Aritra Hazra
Department of Computer Science Engineering
Indian Institute of Technology, Kharagpur
Scribed by: Abhishek Kumar ; 20CS91R02

Notes for class on 10th February 2021

1 First Slide

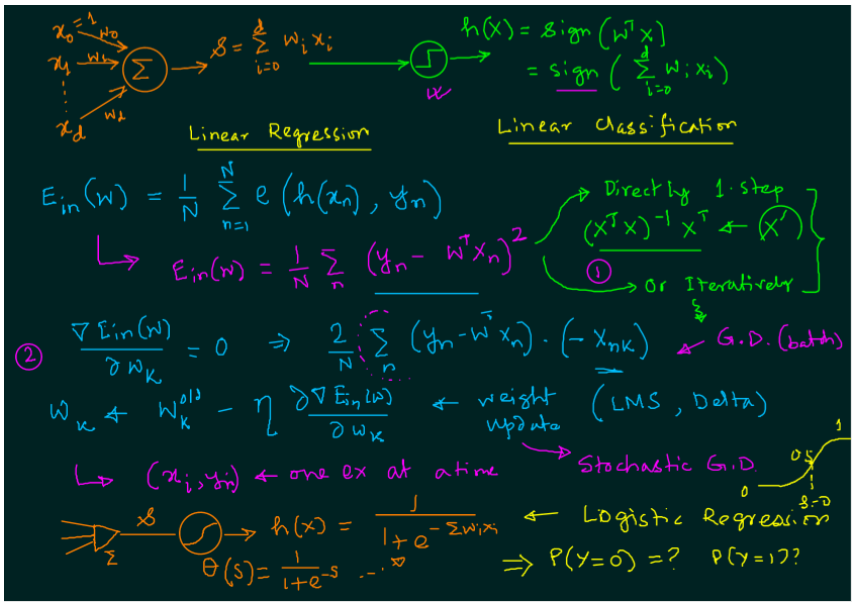

Figure 1: First Slide .

We have a mandatory x_0 attribute that is equal to 1 and we have a set of attributes from x_0 to x_d, where d is the dimensionality. We provide the output sum which is nothing but summation of i from 0 to d. Weight w_0 to w_d are the weights with respect to each attribute. This is what we call **Linear Regression**.

Further we went on and included a new thing that is a threshold value.

But Why we did it?

To know the class i.e two class classification called **linear classification**.

With respect to weights which are free variable in our case. We try to minimise the average error which we call the error being produced by N number of training examples, where each of $h(x_n)$ gives hypothesis and actual training examples output from unknown function y_n which is error.

In case of linear regression we got the error

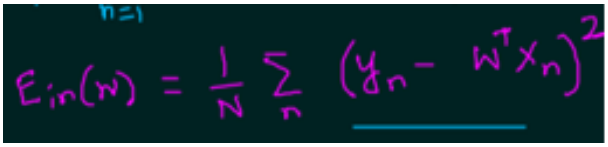

Figure 2: Slide 1.

Now it can be solved using two ways:

1. Direct Step

   The closed-form solution may (should) be preferred for "smaller" datasets – if computing (a "costly") matrix inverse is not a concern. For very large datasets, or datasets where the inverse of $X^T X$ may not exist (the matrix is non-invertible or singular, e.g., in case of perfect multicollinearity), the GD or SGD approaches are to be preferred. The linear function (linear regression model) is defined as:

   $$y = w_0 x_0 + w_1 x_1 + ... + w_m x_m = \sum_{j=0}^{m} W^T X$$

   where y is the response variable, x is an m-dimensional sample vector, and w is the weight vector (vector of coefficients). Note that w0 represents the y-axis intercept of the model and therefore x0=1. Using the closed-form solution (normal equation), we compute the weights of the model as follows:

   $$w = (X^T X)^{-1} X^T Y$$

2. Gradient Descent Approach

   - **Batch Gradient Descent**: In Batch Gradient Descent, all the training data is taken into consideration to take a single step. We take the average of the gradients of all the training examples and then use that mean gradient to update our parameters. So that's just one step of gradient descent in one epoch.
   - **Stochastic gradient descent**: The word 'stochastic' means a system or a process that is linked with a random probability. Hence, in Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration.

2 Second slide

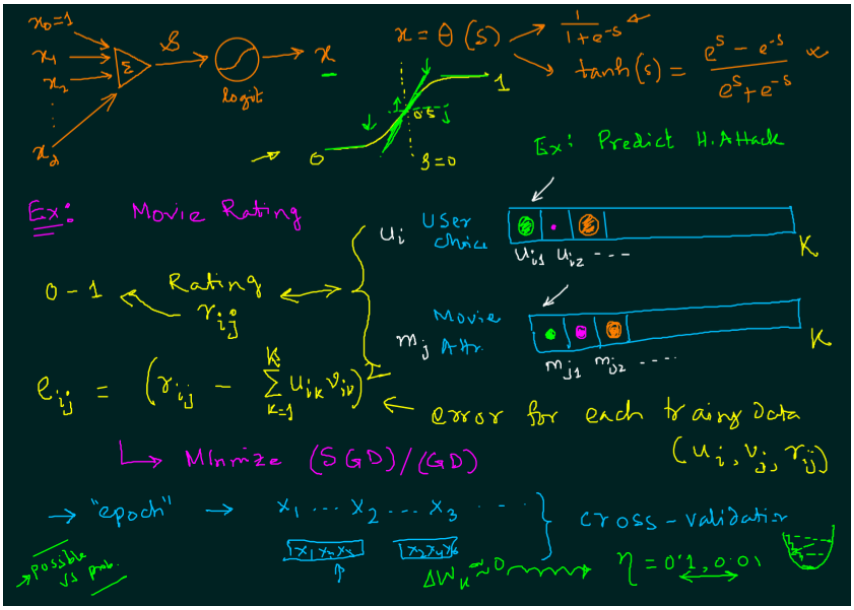

Figure 3: Slide 2.

Logistic regression: In logistic regression, the output of the hypothesis is nothing but the probability of output being 0 or 1. Mostly, Logistic Regression is used when the dependent variable (target) is categorical. We get the two values because of the nature of the curve and the threshold value. For theta of S, we can take **sigmoid** function or **tanh** function .

1. **Sigmoid function**: This function looks like smooth curve. It is linear in both the ends and in the middle portion it looks somewhat linear. In classification, we take decisions based on whether value is greater than 0.5 or less than 0.5.

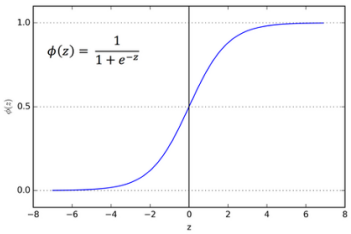

Figure 4: Sigmoid Function.

2. **Hyperbolic tangent function(tanh)**: tanh is also like logistic sigmoid but better. The range of the tanh function is from (-1 to 1). tanh is also sigmoidal (s - shaped). The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph.

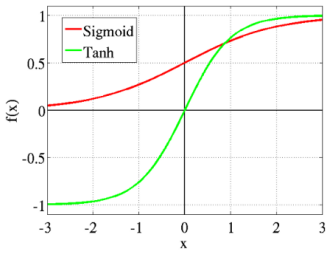

Figure 5: tanh Function.

Predicting heart attacks is an application where logistic regression can be used as after some age we have a higher chance of heart attack that is value towards 1 and less or no chance of heart attack that is value towards 0.

Another popular example which we discussed earlier is the movie rating. Where we have user attributes

$$u_{i1}u_{i2}, ...u_{ik}$$

and movie attributes as

$$v_{j1}, v_{j2}, ...v_{jk}$$

Based on the pair we get the movie ratings between 0 and 1. We have eij which is the error for each training dataset.

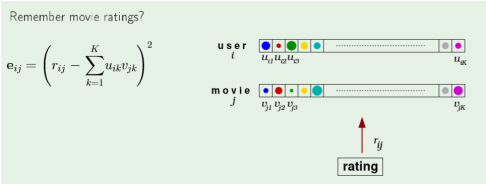

Figure 6: movie rating example

Question: How we can we be sure that SGD takes on example at a time ?
Answer: For this we have a term called 'epoch'. Cross validation

3 Third Slide

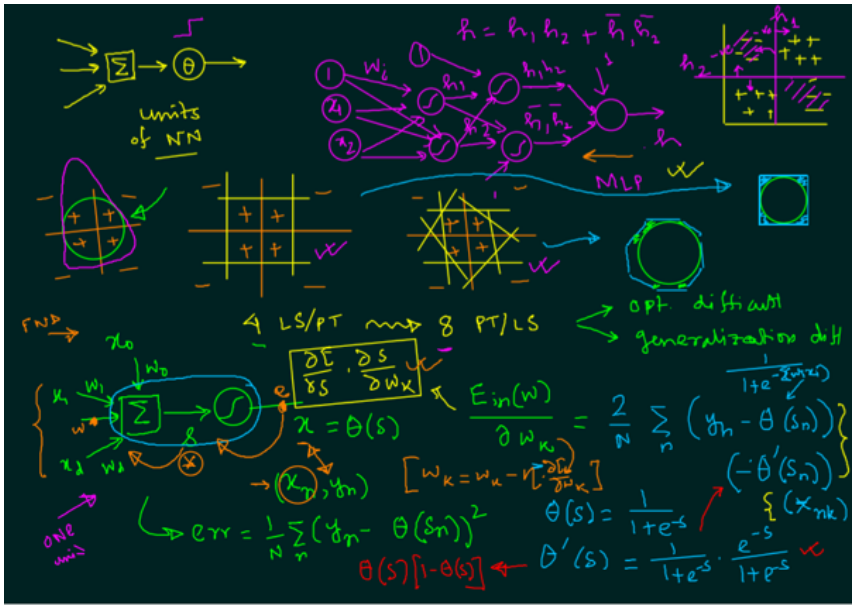

Figure 7: Third slide

Following figure below, represents the units of neural network where we take summation follow it up with our logistic and given some input.

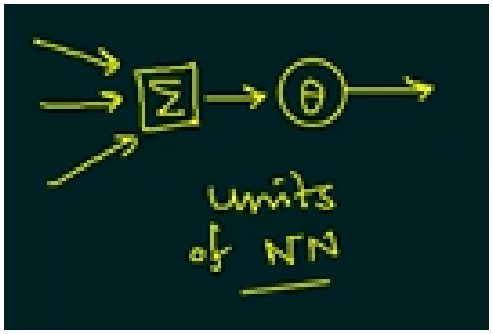

Figure 8: Units of Neural Network

When we talk about the multilayer perceptron, which can differentiate the following negative and positive symbols. You can easily see that one single line can not differentiate them, so we need atleast two different lines. In previous class we used h1 and h2 for differentiating the symbols as you can see in the below figure clearly.

Figure 9:

We have a question in last class where we have data points as shown in figure below, so how can we find the target concept?

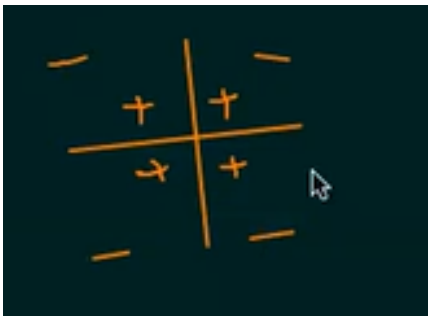

Figure 10:

Now we have to concept using the neural network or multilayer perceptron. We can do that using 4 perceptrons, 8 perceptrons or 16 perceptrons. As we use more perceptrons, it makes the neural network more deeper and complex which can be seen in figure below. In every case we get some error but with more perceptron optimization and generalization becomes very difficult.

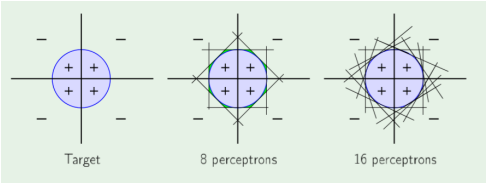

Figure 11:

Now we need to see what will be our weights. In single level the error will be:

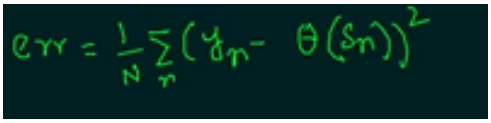

Figure 12:

4 Fourth Slide

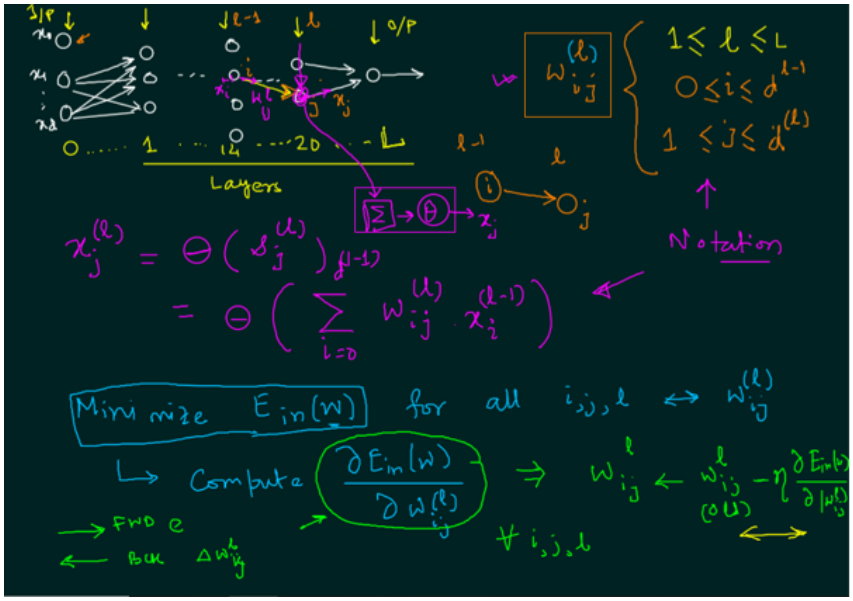

Figure 13:

In Multi layer we have different layers which gives an output. Input layers are x0, x1 upto xd. We have l number of layers staring from 0 where first is the input, middle one's are the hidden layers and last is the output layer. We have a layer variable 1¡= l¡=L. Weights are bound to the layer as it moves to like one node i to other j will be having weight wlij J boundary is the number of output that is there in layer L upto 1. Each node is blown up as Sum unit followed by theta unit as shown below

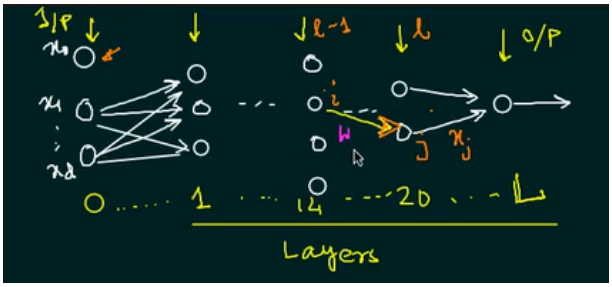

Figure 14:

The bottle neck is, We need to compute the following for every i,j and l. Forward pass for error and backward pass for delta updation.How to compute we will see in next slide.

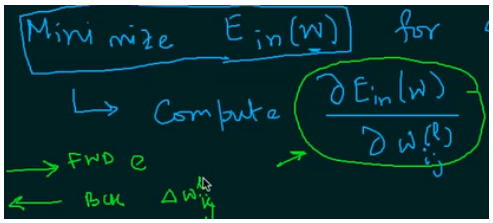

Figure 15:

5 Fifth Slide

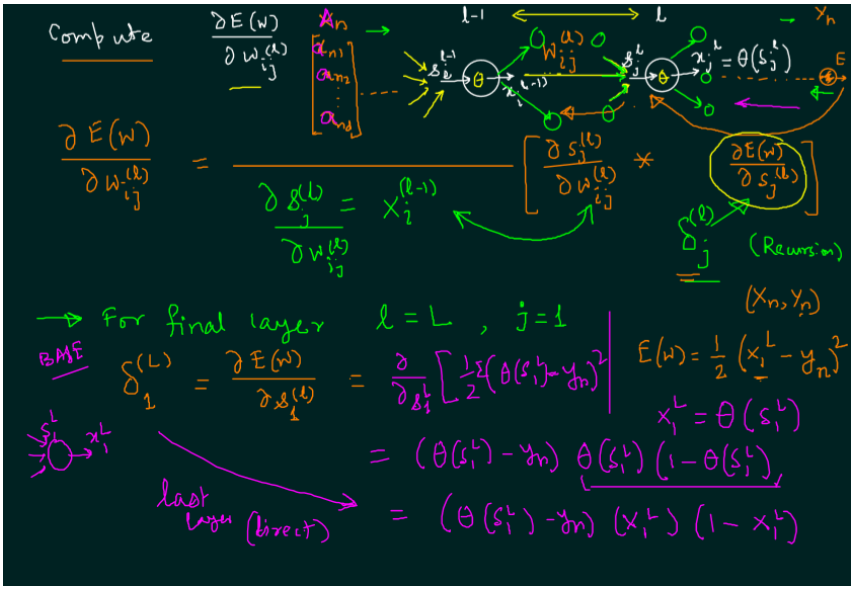

Figure 16: Fifth slide

For computing $\frac{\delta e(w)}{\delta w_{ij}^l}$ we will take the reference of the following slide below.

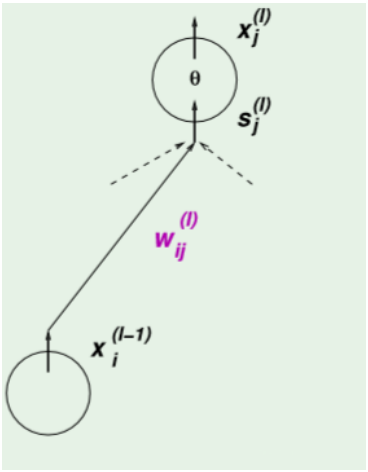

Figure 17:

A trick for efficient computation:

$$\frac{\delta e(w)}{\delta w_{ij}^l} = \frac{\delta e(w)}{\delta s_j^l} * \frac{s_j^l}{\delta w_{ij}^l}$$

We have $\frac{s_j^l}{\delta w_{ij}^l} = x_i^{l-1}$ We only need $\frac{\delta e(w)}{\delta s_j^l} = \delta_j^l$

δ for final layer

δ_j^l For the final layer l=L and j=1

$$\delta_1^l = \frac{\delta e(w)}{\delta s_1^l}$$

$$e(w) = (x_1^L - y_n)^2$$

$$x_1^L = \theta(s_1^L)$$

$$\theta(S) = 1 - \theta^2(s)$$

Back propagation of δ :

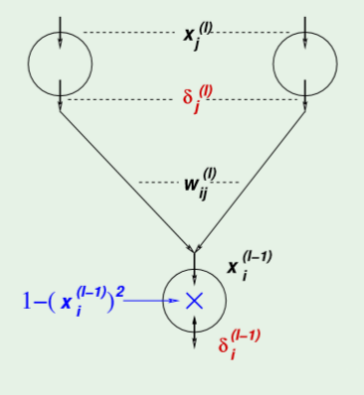

Figure 18:

$$\delta_i^{l-1} = \frac{\theta e(w)}{\theta s_i^{l-1}}$$

$$= \sum_{j=1}^{d^l} \frac{\delta e(w)}{\delta s_j^l} * \frac{\delta s_j^L}{\delta x_i^{l-1}} * \frac{\delta x_i^{l-1}}{\delta s_i^{l-1}}$$

$$= \sum_{j=1}^{d^l} \delta_j^l * w_i{}_j^l * \theta(s_i^{l-1})$$

$$\delta_i^{l-1} = (1 - (x_i^{l-1})^2) \sum_{j=1}^{d^l} w_i{}_j^l \theta_l^l$$

6 Sixth Slide

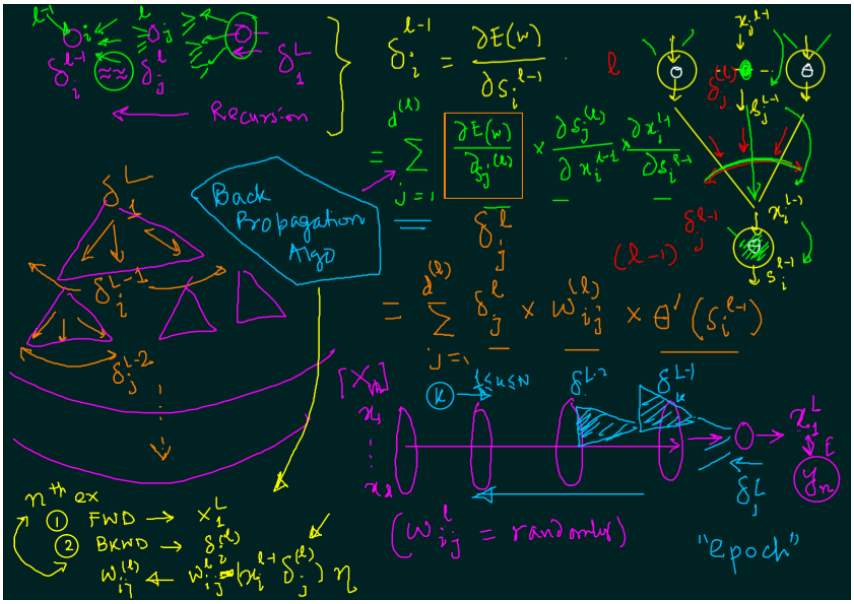

Figure 19:

Backpropagation Algorithm

1. **Initialize all weights w_{ij}^l at random.**
2. **for t = 0,1,2,... do**
3. **Pick n 1,2,··· ,N**
4. **Forward: compute all x_j^l**
5. **Backward: compute all δ_j^l**
6. **Update the weights: $w_{ij}^l < -w_{ij}^l - nx_i^{l-1}\delta_j^l$**
7. **Iterate to the next stop until it is time to stop.**
8. **Return the final weights w_{ij}^l**

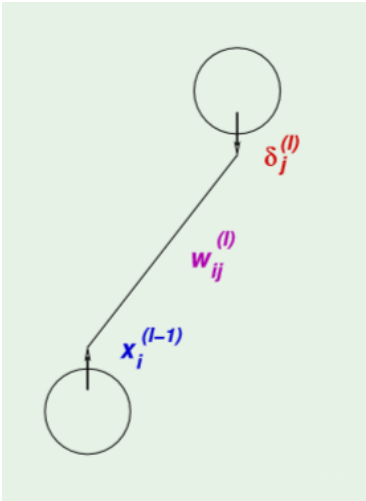

Figure 20: