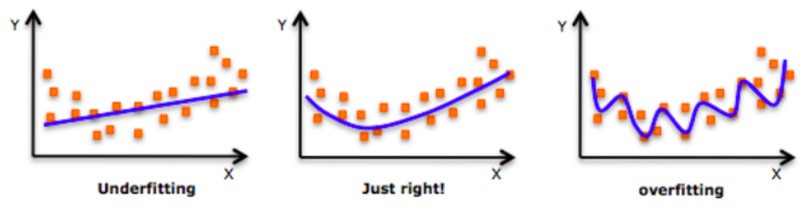
Day 4

Agenda :

1. Regularization
    1. Dropout
    2. L1 and L2 Regularization
2. Weight initialization techniques
    1. Zero Initialization (Initialized all weights to 0)
    2. Random Initialization (Initialized weights randomly)
    3. With Zero mean
    4. Uniform Distribution
    5. Xavier/Glorot Init(2010)
    6. He-Init (2015)
3. Optimizers:-
    1. Gradient Descent
    2. Stochastic Gradient Descent
    3. Mini-Batch Gradient Descent
    4. Adagrad
    5. AdaDelta
    6. Adam
4. Batch Normalization
5. Loss Functions for Neural Networks
6. Softmax
7. How to train MLP (steps)
8. project on Churn dataset.

Regularization

-- Regularization refers to a set of different techniques that lower the complexity of a neural network model during training, and thus prevent the overfitting.

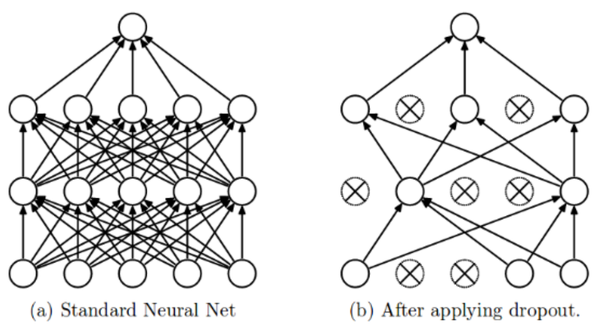


-- Regularization is a technique which makes slight modifications to the learning algorithm such that the model generalizes better. This in turn improves the model's performance on the unseen data as well.

Different Regularization Techniques in Deep Learning

Dropout :

-- Nitish Shrivastava intern under Jeffery Hinton
— Dropout: A Simple Way to Prevent Neural Networks from Overfitting.



(a) Standard Neural Net (b) After applying dropout.

The term "dropout" refers to dropping out the nodes (input and hidden layer) in a neural network. All the forward and backwards connections with a dropped node are temporarily removed, thus creating a new network architecture out of the parent network. The nodes are dropped by a dropout probability of p.

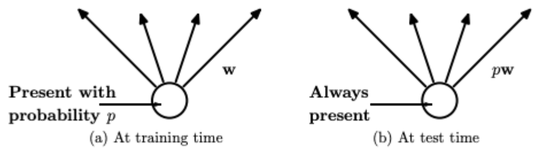
-- We randomized the neurons.

-- At training time only P % neuron will be present like they have probability of P and connected to next layer with weight W at test time the unit is always present and multiplied by P.

-- 0 <=P<= 1

P = 1 / 1-Rd

Rd - Dropout rate, = #dropout N/W / #totral N/W



(a) At training time (b) At test time

- Weight is disconnected / Neurons are disconnected only at training time, not test time. Test time weights are multiplied by the P so the total weight for each neuron will be = W*P

- P is a hyperparameter sometimes.

L2 & L1 regularization

-- L1 and L2 are the most common types of regularization. These update the general cost function by adding another term known as the regularization term.

Cost function = Loss (say, binary cross entropy) + Regularization term

In L2, we have:

$$Cost\,function = Loss + \lambda * \sum \|w\|^2$$

In L1, we have:

$$Cost\,function = Loss + \lambda * \sum \|w\|$$

Weight Initialization

Why Weight Initialization?

-- Its main objective is to prevent layer activation outputs from exploding or vanishing gradients during the forward propagation.

-- If we initialized the weights correctly, then our objective i.e, optimization of loss function will be achieved in the least time

Different Weight Initialization Techniques

1. Zero Initialization (Initialized all weights to 0)

if we initi W =0 ---------- Very bad Idea

---------- All neurons compute the same
---------- Same gradient update

We have to make asymmetry so all neuron should work differently (like weight should be diff)

2. Random Initialization (Initialized weights randomly)

-- we can initiate the W randomly but it's shouldn't be zero. by doing this we can get asymmetry in network.
--
-- "What happens if the weights initialized randomly can be very high or very low?"

Cons : if we initiate W very larger then expanding gradient problem can occurs

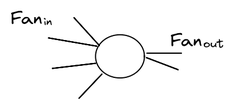
3. With Zero mean

-- Weight should be small (not too small)
-- Not all zeros
-- With good variance ---var(W(ki j))

$W_{ij}^{k} = N(0, var)$ Var should be small

4. Uniform Distribution

$$W_{ij}^{k} \longrightarrow unif\left[\frac{-1}{\sqrt{Fan_{in}}}, \frac{+1}{\sqrt{Fan_{out}}}\right]$$



-- We take value from above range
-- it work well with sigmoid function.

5. Xavier/Glorot Init(2010)

(a) $W_{ij}^{k} \longrightarrow N(0, var_{ij})$

$$Var_{ij} = \frac{\sqrt{2}}{\sqrt{Fan_{in} + Fan_{out}}}$$

(b) $W_{ij}^{k} = Unif\left[\frac{-\sqrt{6}}{S}, \frac{+\sqrt{6}}{S}\right]$

$$S = \sqrt{Fan_{in} + Fan_{out}}$$

-- It work well with sigmoid activation function

6. He - Init (2015)

(a) $W_{ij} \rightarrow N(0, var_{ij})$ $\quad Var_{ij} = \sqrt{\frac{2}{Fan_{in}}}$

(b) $W_{ij}^{k} = Unif\left[\frac{-\sqrt{6}}{S}, \frac{+\sqrt{6}}{S}\right]$

$S = \sqrt{Fan_{in}}$

-- It work well with ReLu

Optimizers

Epoch – The number of times the algorithm runs on the whole training dataset.

Sample – A single row of a dataset.

Batch – It denotes the number of samples to be taken to for updating the model parameters.

Learning rate – It is a parameter that provides the model a scale of how much model weights should be updated.

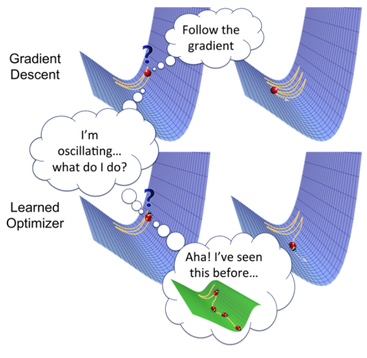
Cost Function/Loss Function – A cost function is used to calculate the cost that is the difference between the predicted value and the actual value.

Weights/ Bias – The learnable parameters in a model that controls the signal between two neurons.

An optimizer is a function or an algorithm that modifies the attributes of the neural network, such as weights and learning rates.
Thus, it helps in reducing the overall loss and improving accuracy. The problem of choosing the right weights for the model is a daunting task, as a deep learning model generally consists of millions of parameters

Different Types :

1. Gradient Descent
2. Stochastic Gradient Descent
3. Mini-Batch Gradient Descent
4. Adagrad
5. AdaDelta
6. Adam



4. Adagrad :

In G.D and SG.D we were taking the learning rate (n=0.01) same for each weight.

$W_t = W_{t-1} - n g_t$ $\quad g = \left(\frac{dt}{dw}\right)_{t-1}$

for Adaptive ---

$W_t = W_{t-1} - n'_t g_t$ $\quad \eta'_t = \frac{\eta}{sqrt(\alpha_t + \epsilon)}$

$\alpha_t = \sum_{i=1}^{t} g_i^2$

summing positive gradient

As t↑ α_t ↑ η'_t ↓

- Learning rate is increasing or decreasing adaptively.
- No need of manually training n.

Prob-

1. Computationally expensive as a need to calculate the second order derivative.
2. The learning rate is always decreasing resulting in slow training.

5. AdaDelta

-- It is an extension of AdaGrad which tends to remove the decaying learning Rate problem of it.

-- Instead of accumulating all previously squared gradients, Adadelta limits the window of accumulated past gradients to some fixed size w.

$W_t = W_{t-1} - n'_t g_t$ $\quad \eta'_t = \frac{\eta}{\sqrt{eda_t + \epsilon}}$

-- eda - exponential decaying average

$eda_t = \gamma(eda)_{t-1} + (1-\gamma) * g^2_{t-1}$

$\gamma = 0.95$

$eda_t = 0.95*(eda)_{t-1} + 0.05*g^2_{t-1}$

Advantages:
-- Now the learning rate does not decay and the training does not stop.

Disadvantages:
-- Computationally expensive.

6. Adam (Adaptive moment estimation)

-- Adam (Adaptive Moment Estimation) works with momentums of first and second order.

-- The intuition behind the Adam is that we don't want to roll so fast just because we can jump over the minimum, we want to decrease the velocity a little bit for a careful search.

M(t) and V(t) are values of the first moment which is the Mean and the second moment which is the uncentered variance of the gradients respectively.

$M_t = B_1 * M_{t-1} + (1-B) * g_t$ $\quad 0 <= B_1 <= 1$

$V_t = B_2 * V_{t-2} + (1-B_2) * g_t^2$ $\quad 0 <= B_2 <= 1$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}.$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}.$$

First and second order of momentum

$$W_t = W_{t-1} - \frac{n * m_t}{\sqrt{V_t} + \epsilon}$$

The values for β1 is 0.9 , 0.999 for β2, and (10 x exp(-8)) for 'ϵ'.

Advantages:

1. The method is too fast and converges rapidly.
2. Rectifies vanishing learning rate, and high variance.

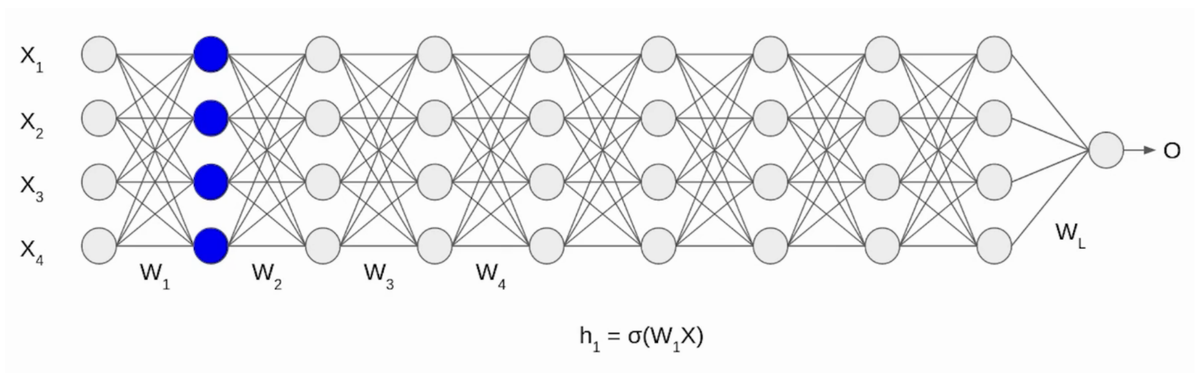
Disadvantages:

1. Computationally costly.

Adam is the best optimizer. If one wants to train the neural network in less time and more efficiently than Adam is the optimizer.

Batch Normalization

Initially, our inputs X1, X2, X3, X4 are in normalized form as they are coming from the pre-processing stage. When the input passes through the first layer, it transforms, as a sigmoid function applied over the dot product of input X and the weight matrix W.



--- As we did Normalization in machine learning to make less variance data
--- Sometimes in neuron network after so many computation and gradient multiplications data become large so variance coefficient get increased.
--- To overcome this issue we make a batch between the layers to normalize the data(input) in between the hidden layers.

Normalize the data in between the layers

Loss Functions for Neural Networks

-- The Loss Function is one of the important components of Neural Networks.

-- Loss is nothing but a prediction error of Neural Net. And the method to calculate the loss is called Loss Function.

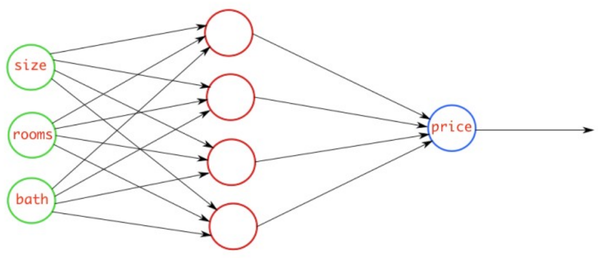
1. Mean Squared Error :

MSE loss is used for regression tasks. As the name suggests, this loss is calculated by taking the mean of squared differences between actual(target) and predicted values.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2$$

Example :

For Example, we have a neural network which takes house data and predicts house price. In this case, you can use the MSE loss. Basically, in the case where the output is a real number, you should use this loss function.



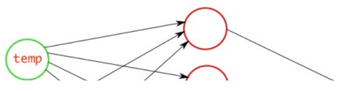
2. Binary Crossentropy:

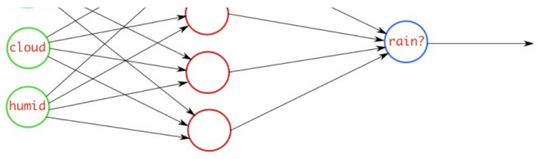
BCE loss is used for the binary classification tasks. If you are using BCE loss function, you just need one output node to classify the data into two classes. The output value should be passed through a sigmoid activation function and the range of output is (0 – 1).

$$CE\ Loss = \frac{1}{n}\sum_{i=1}^{N} - (y_i \cdot log(p_i) + (1 - y_i) \cdot log(1 - p_i)$$

Example:

-- We can take an example of rain prediction.

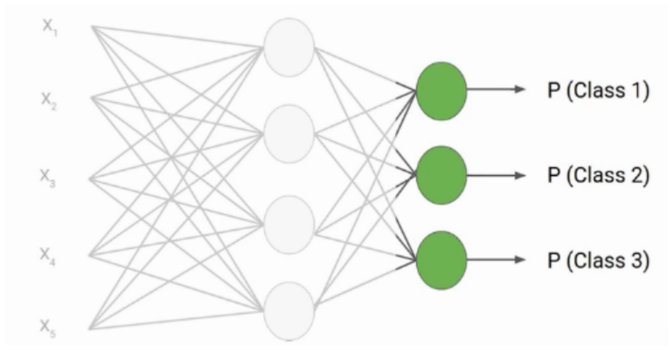




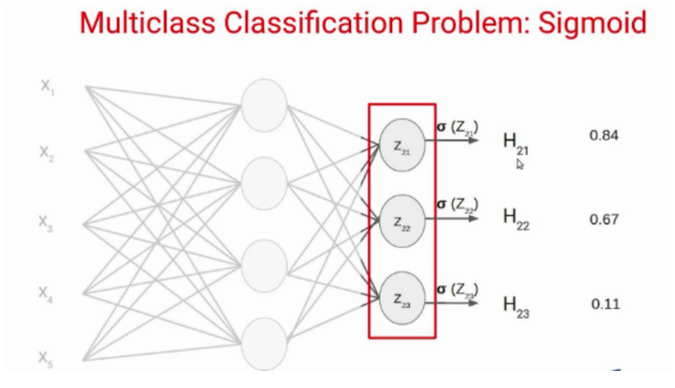
SoftMax

-- It is often used as the last activation function of a neural network to normalize the output of a network to a probability distribution over predicted output classes.

Feature X1	Feature X2	Feature X3	Feature X4	Feature X5	Target
1	22	569	35	0	Class 1
1	7	351	75	1	Class 2
1	45	451	542	1	Class 2
1	5	572	8	0	Class 1
0	22	565	44	1	Class 3
0	24	243	546	1	Class 3
1	78	953	42	0	Class 2



Why Not Sigmoid?



There are two problems in this case

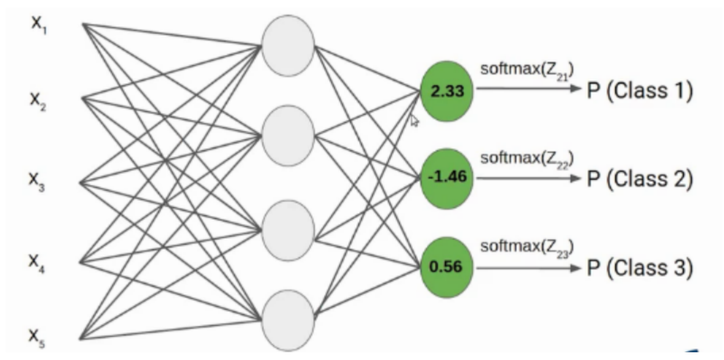
1. First, if we apply a thresh-hold of say 0.5, this network says the input data point belongs to two classes.
2. Secondly, these probability values are independent of each other.

That means the probability that the data point belongs to class 1 does not take into account the probability of the other two classes.

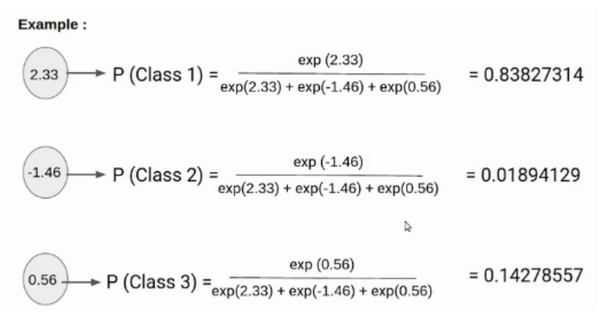
Back to the SoftMax

$$softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)}$$

Here, the Z represents the values from the neurons of the output layer. The exponential acts as the non-linear function.



These are the probability values that a data point belonging to the respective classes.
Note that, the sum of the probabilities, in this case, is equal to 1.



How to Train MLP :

Steps:-
1. Preprocess - Data Normalization (Minmaxscaler, Standardscaler)
2. Weight init - Xavier/Glorot - Sigmoid/tanh
   - He- ReLu
   - Gaussian (small variance)
3. Choose activation function - ReLu(2018-recomm), Leakey ReLu
4. Batch Normalization - especially for Deep MLP (Later Layer)
   - Dropout

5. Optimizer - Adam (Adaptive, Fast)

6. Hyperparameter - Architechture -- #layer
                                    # Neuron
                                -- Dropout rate
                                -- Adam: B_1 , B_2 , alpha

7. Loss Function - 2- Class classification - Log loss
                 - Regression -   Squere loss
                 - K- classs - Multiclass losses

8. Plots (Train loss wrt epoches)