Building Blocks of Neural Networks

Discussion Question

1. What are the different types of optimizations?
2. Why is the need for advanced optimizers?
3. What is weight initialization in Neural Networks and why do we need it?
   a. What happens when weight is initialized with W=0?
   b. What happens when weights are initialized randomly ?
4. How to prevent Overfitting in Neural Networks?
5. When to use Dropout and Batch Normalization?
6. What are the different types of Hyperparameters in Neural Networks?

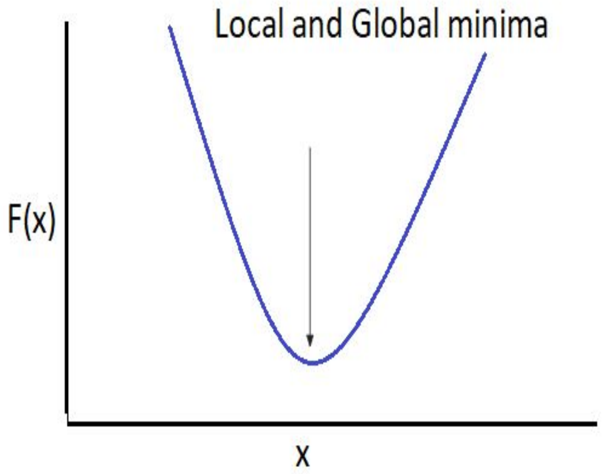
What are the different types of optimizations?

There are two types of optimizations.

1. Convex optimization
2. Non-Convex optimization

Convex Optimization

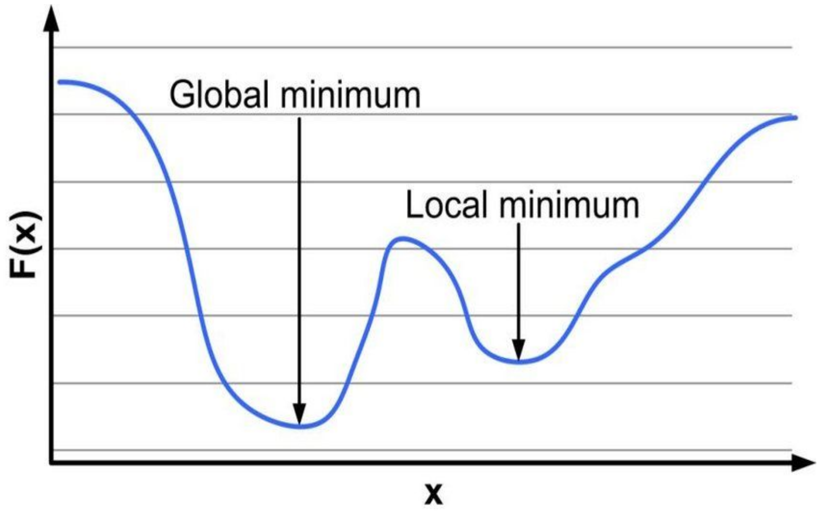
Convex optimization involves the function that has only one minima, corresponding to Global optimum (minima or maxima). There is no concept of local optima for convex optimization problems.



What are the different types of optimizations?(Contd)

Non-Convex Optimization

Non-convex optimization involves a function that has multiple optima values, out of which only one is the global optima. Depending on the loss function, it can be very difficult to locate global optima.



https://www.researchgate.net/figure/example-of-non-convex-continuous-function_fig9_284419329

Why is the need for advanced optimizers?

In most real-life problems, we only have Non-convex functions but it is really hard to find Global minima in these functions, now using Gradient Descent as an optimization algorithm for solving non-convex functions. There are two problems associated with it:

1. It converges to minima slowly
2. It gets stuck in the local optima(minima)

Below are the different types of Gradient Descent or Optimizers that can solve the above problems

1. Stochastic Gradient Descent
2. Mini Batch Gradient Descent
3. Stochastic Gradient Descent with Momentum
4. RMSprop
5. Adam

What is weight initialization in Neural Networks and why do we need it?

- Weight Initialization is a procedure to assign weights to a neural network with some small random values during the training process in a neural network model.

- The purpose of using the weight initialization technique is to prevent the neural network from exploding or vanishing gradient during the forward and backward propagation. If either occurs, the neural network will take a longer time to converge.

What is weight initialization in Neural Networks and why do we need it?(Contd)

What happens when weight is initialized with W=0?

- This makes your model equivalent to a linear model.
- When you set all weights to 0, the derivative with respect to loss function is the same for every weight (w) in every layer, thus, all the weights have the same values in the subsequent iteration.
- This makes the hidden units symmetric and continues for all the n iterations you run.
- Thus setting weights to zero makes your network no better than a linear model.

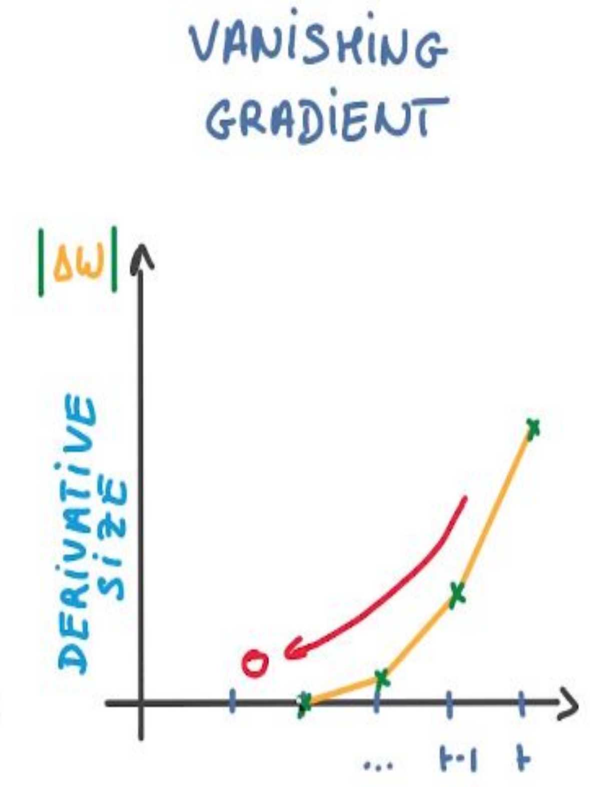
What happens when weights are initialized randomly?

Initializing weights randomly, following standard normal distribution (np.random.randn(n, n-1) in Python) while working with a (deep) neural network can potentially lead to 2 issues — **vanishing gradients** or **exploding gradients**.

What is weight initialization in Neural Networks and why do we need it?(Contd)

Vanishing Gradient Descent

- In the case of deep networks, for any activation function, abs(dW) will get smaller and smaller as we go backward with every layer during backpropagation.
- The earlier layers are the slowest to train in such a case. The weight update is minor and results in slower convergence.
- This makes the optimization of the loss function slow. In the worst case, this may completely stop the neural network from training further.

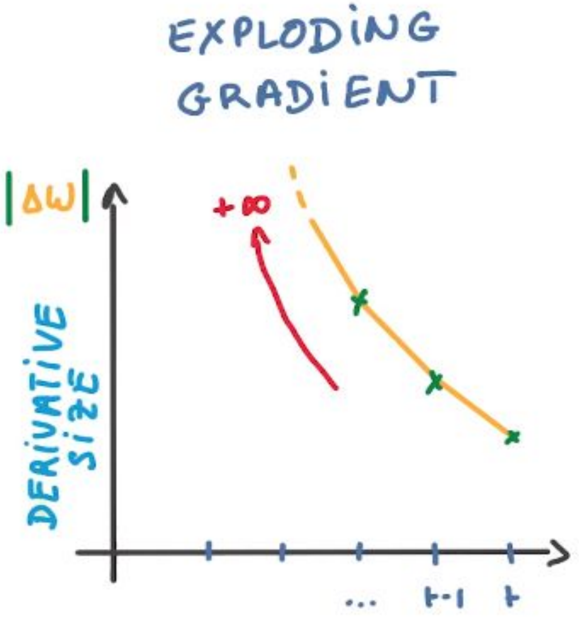


https://medium.com/@ayushch612/vanishing-gradient-and-exploding-gradient-problems-7737c0aa535f

What is weight initialization in Neural Networks and why do we need it?(Contd)

Exploding Gradient Descent

- This is the exact opposite of vanishing gradients. Let us consider having non-negative and large weights and small activations A (as can be the case for sigmoid(z)).
- This may result in oscillating around the minima or even overshooting the optimum again and again and the model will never learn!
- Another impact of exploding gradients is that huge values of the gradients may cause number overflow resulting in incorrect computations or introductions of NaN's. This might also lead to the loss of taking the value NaN.



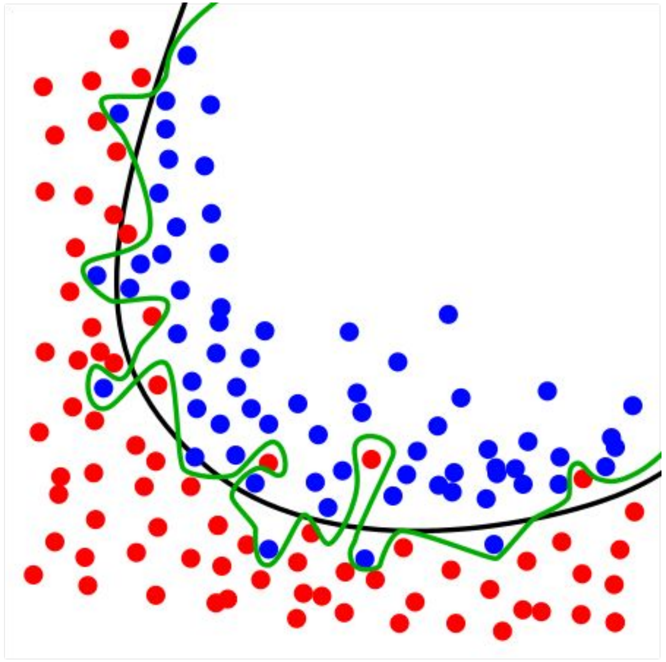
https://medium.com/@ayushch612/vanishing-gradient-and-exploding-gradient-problems-7737c0aa535f

How to prevent Overfitting in Neural Networks?

- Neural networks are prone to overfitting because of the larger number of parameters.
- Neural Network is able to model higher order and complex functions which makes it more prone to overfitting .

Different types of Regularizations to prevent overfitting are as follows:

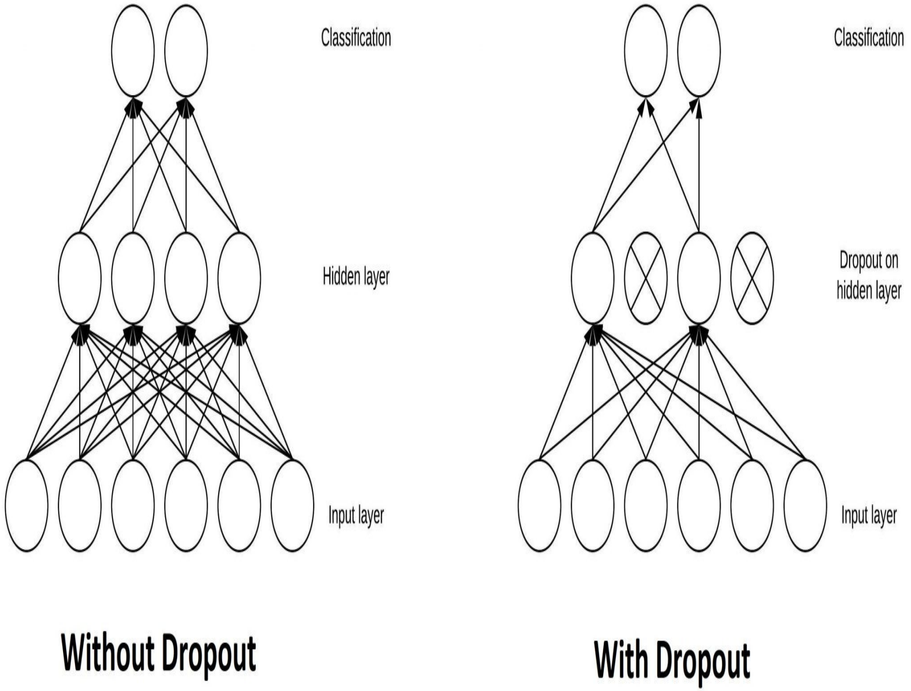
1. Dropout
2. Batch Normalization
3. L2 Regularization
4. Early Stopping


https://en.wikipedia.org/wiki/Overfitting

How to prevent Overfitting in Neural Networks?(Contd)

Dropout

- During training, dropout randomly sets some neurons to zero in the forward pass.
- A fully connected layer occupies most of the parameters, and hence, neurons develop interdependence amongst each other during training which curbs the individual power of each neuron, leading to over-fitting of the training data.
- A dropout is an approach to regularization in neural networks which helps to reduce interdependent learning amongst the neurons.



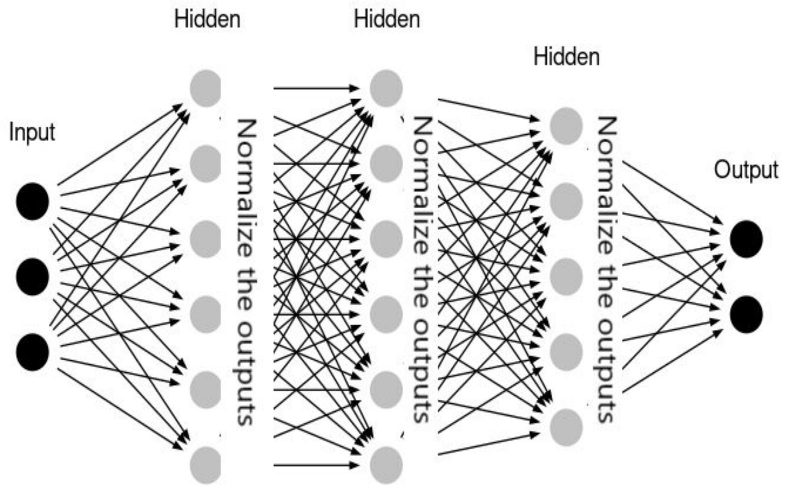
https://www.baeldung.com/cs/ml-relu-dropout-layers

How to prevent Overfitting in Neural Networks?(Contd)

Batch Normalization

Batch normalization is a technique for improving the performance and stability of neural networks. The idea is to normalize the inputs of each layer in such a way that they have a mean of zero and a standard deviation of one.

Due to these normalization “layers” between the fully connected layers, the range of input distribution of each layer stays the same, no matter the changes in the previous layer.



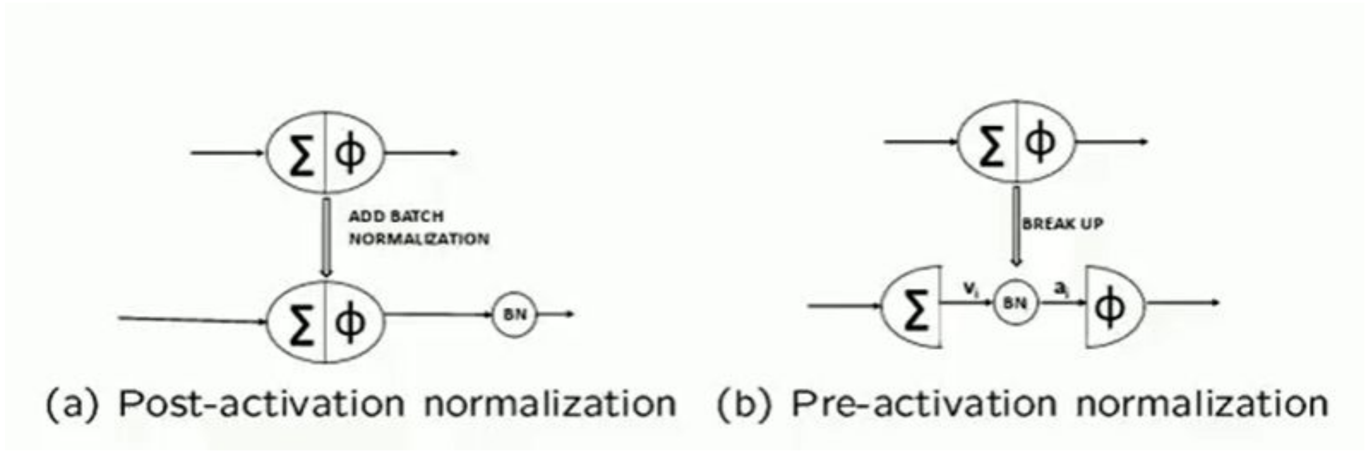
https://csmoon-ml.com/index.php/2019/04/05/batch-normalization/

How to prevent Overfitting in Neural Networks?(Contd)

Batch Normalization

There are usually two stages in which Batch Normalization is performed:

1. Before the Activation function
2. After the Activation function



Σ
Φ
ADD BATCH NORMALIZATION
BN
BREAK UP
v_i
a_i
(a) Post-activation normalization
(b) Pre-activation normalization

Φ - Activation Function

Σ - Summation of Inputs and Weights

BN - Batch Normalization

https://iq.opengenus.org/batch-normalization/

How to prevent Overfitting in Neural Networks?(Contd)

L2 Regularization

Due to the addition of the regularization term, the weights decrease because it assumes that a neural network with a smaller weight leads to a simpler model. Therefore, it will also reduce the overfitting problem to quite an extent.

$$W_{i,j} - n\frac{\Delta(L)}{\Delta W_{ij}}$$

Fig (a)

Weight updation equation with regularization term is given in fig (b)

$$W_{ij} * (1-nl) - n\left[\frac{\Delta L}{\Delta W_{ij}}\right]$$

Fig (b)

n - Learning Rate
l - Regularization Parameter
(1-nl) - Weight Decay

If the learning rate is high, then difference in (1-nl) will be low and weights will go down.

If the learning rate is low, then difference in (1-nl) will be high and weights will be high.

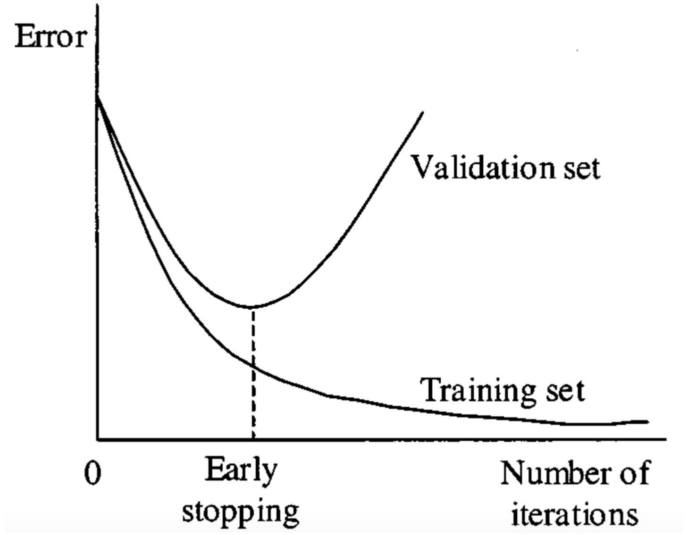
How to prevent Overfitting in Neural Networks?(Contd)

Early Stopping

Early stopping is a technique similar to cross-validation, where a part of training data is kept as the validation data. When the performance of the validation data starts worsening, the model will immediately stop the training.

We usually reduce overfitting by observing the training/validation accuracy gap during model training and then stopping at the right point.

In the given fig, the model will stop training at the dotted line since after that model will start overfitting on the training data.



https://paperswithcode.com/method/early-stopping

What are the Guidelines to use Dropout and Batch Normalization?

Dropout

- The choice of which units(Neurons) to be dropped during training is random.
- What should be the Dropout ratio in a neural network is a hyperparameter. The dropout ratio (p) is between 0<=p<=1. Whenever a deep neural network is overfitting, the (p) value should be high.
- A good value for the dropout ratio in the hidden layer is 0.5-0.8.
- At test time all neurons are always ON and weights in the neural networks will be scaled with the Dropout ratio. Default Dropout ratio in python is 0.5

Batch Normalization

- The Batch norm should be used in the deeper network.
- Batch Normalization (BN) often leads to a worse performance when It is combined with dropout in both theoretical and statistical aspects.(Not in all cases)
- Using batch norm depends on the problem. It should be used when the absolute scale of the features does not matter.

What are the different types of Hyperparameters in Neural Networks?

Hyperparameters are the ones which we will be giving the model that helps in the learning process.

Example:- In Decision Tree and Random Forest Max_Depth, Min_sample, Min_sample_leaf, max_leaf_node, and criterion are the hyperparameters given to DT and RF

Hyperparameters in Neural Networks

- Activation Function
- Number of Neurons in each layer
- Number of Hidden Layers
- Optimizer
- Learning Rate
- Loss/ Cost Function
- Regularization Techniques

greatlearning
Power Ahead

Happy Learning !