DS JULY 2022 Batch
Module 19 – Deep Learning part 2

Topics

- Regularization
- Optimizers
- Hyperparameters and tuning of the same

Regularization

Regularization

- What is a deep learning algorithm? Obviously, we know it includes a model but is not just that, isn't it? Using a pseudo-match nomenclature, we can define a deep learning algorithm with the following equation:
  - **DL(x)= Model(x) + Cost_Function(Model(x)) + Input_Data_Set (x) + Optimization(Cost_Function(x))**
- Using this conceptual equation, we can represent any deep learning algorithm as a function of an input data set, a cost function, a deep neural network model and an optimization process. In the context of this article, we are focusing on the optimization processes.
- What those those processes so challenging in deep learning systems? One word: size. Deep neural networks include a large number of layers and hidden units that can also include many nodes. That level of complexity directly translates into millions of interconnected nodes which makes for an absolute optimization nightmare.
- When thinking about improving a deep learning model, you should focus the efforts in two main areas:
  - a) Reducing the cost function.
  - b) Reducing the generalization error.

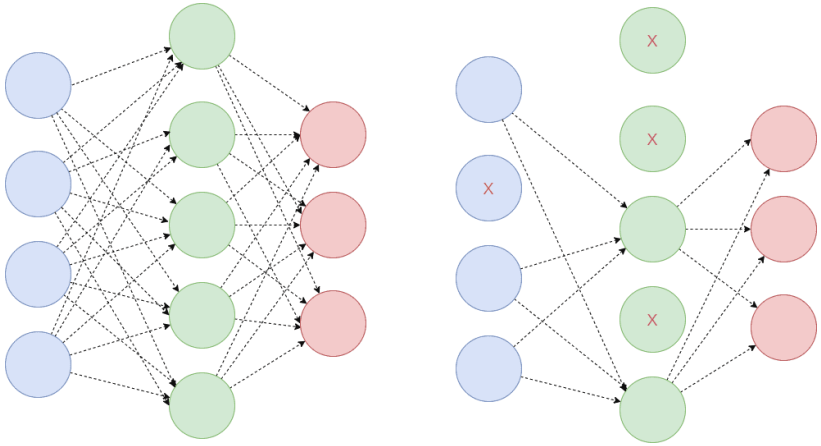
Regularization

- **Regularization**
- The role of regularization is to modify a deep learning model to perform well with inputs outside the training dataset. Specifically, regularization focuses on reducing the test or generalization error without affecting the initial training error.
- Regularization, in the context of neural networks, is a process of preventing a learning model from getting overfitted over training data. It involves a mechanism to reduce generalization errors of the learning model.

- L1
- L2
- Dropout
- Early stopping

Regularization : Dropout

- **Dropout**
- A high number of nodes in each layer of a neural network capture the complex features of data along with noise which leads to overfitting of a learning model. In this regularization technique, the nodes are randomly selected and dropped to make a learning model more generalized so that it will perform better on newly arrived data
-



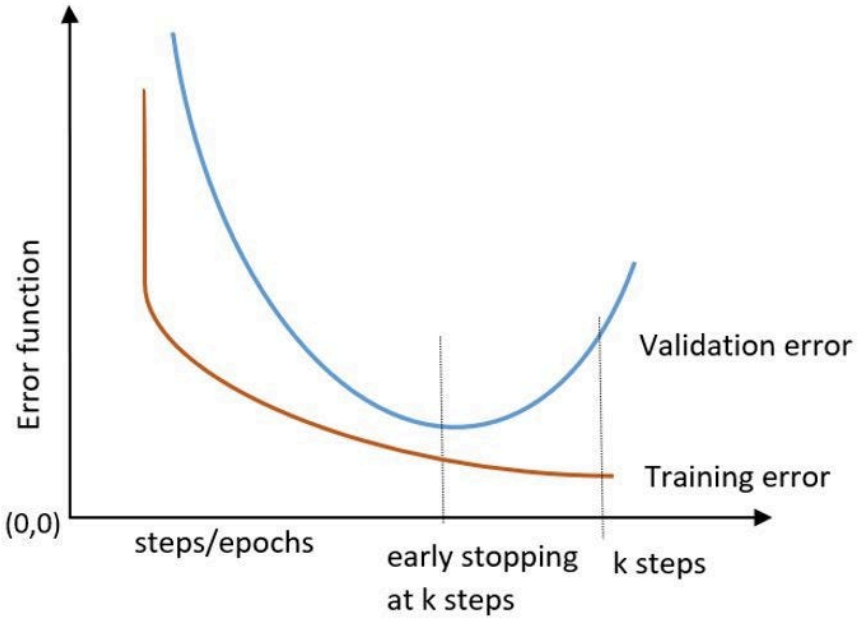
- A high number of nodes in each layer of a neural network capture the complex features of data along with noise which leads to overfitting of a learning model. In this regularization technique, the nodes are randomly selected and dropped to make a learning model more generalized so that it will perform better on newly arrived data.

Regularization : Dropout

- **Dropout**
- During training, some number of layer outputs are randomly ignored (dropped out) with probability p.
- During test time, all units are present, but they have been scaled down by p. This is happening because after dropout, the next layers will receive lower values. In the test phase though, we are keeping all units so the values will be a lot higher than expected. That's why we need to scale them down.
- By using dropout, the same layer will alter its connectivity and will search for alternative paths to convey the information in the next layer. As a result, each update to a layer during training is performed with a different "view" of the configured layer. Conceptually, it approximates training a large number of neural networks with different architectures in parallel.
- "Dropping" values means temporarily removing them from the network for the current forward pass, along with all its incoming and outgoing connections. Dropout has the effect of making the training process noisy.

Regularization : Early Stopping

- **Early stopping**
- Early stopping is a type of regularization that's used to avoid the overfitting of a learning model. A high number of training iterations or epochs leads to overfitting of the learning model over a dataset whereas a smaller number of epochs results in underfitting of the learning model.
- In early stopping, a neural network is simultaneously trained on training data and validated on testing data just to figure out how many iterations are needed for better convergence of the learning model before its validation error starts increasing as shown in the figure.



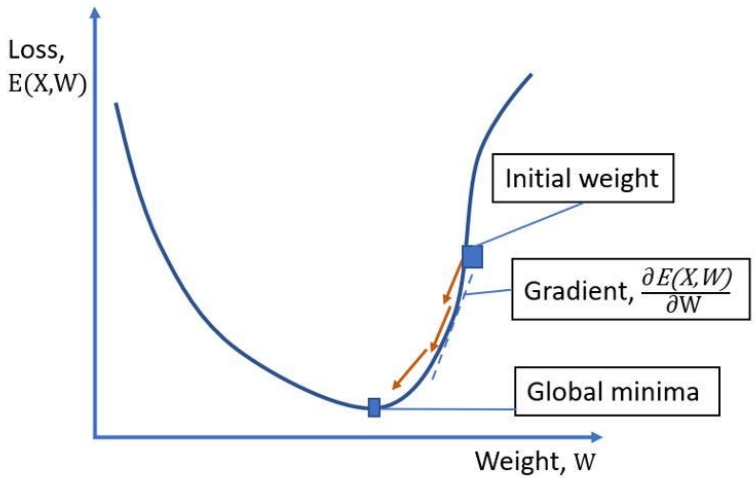
Optimizers

Optimization

- It is important to understand different optimization techniques like Batch Gradient Descent, Stochastic Gradient Descent, Mini-batch Gradient Descent, AdaGrad, AdaDelta for better convergence of the neural network.
- Optimization techniques help in better convergence of a neural network by optimizing the gradient of the error function. There are many variants of gradient descent, which differentiate each other based on how much data is being processed to calculate the gradient of the error function (objective function).
- Before proceeding, there are a few terms that you should be familiar with.
- **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, which 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.

Optimization

- **Batch Gradient Descent (Vanilla Gradient Descent)**
- Batch Gradient Descent is the most commonly used optimization algorithm for improving model performance. It uses the first-order approximation for finding minima for the objective function in an iterative manner. It uses a learning algorithm that updates the weights or parameters proportional to the negative of the gradient of the objective function for connections between layers of neurons in a multi-layered network.
- In other words, it tries to find out the optimal set of weights or parameters for the connections between different nodes of neural layers that will give a minimal error for the predictability. In the batch gradient descent, a loss function, as shown above in the diagram, is calculated by the aggregation of squared differences between correct response, y and calculated response, ŷ of all individual N data points, or observations in a dataset.



Optimization

- **Stochastic Gradient Descent**
- It's a variant of gradient descent for model optimization. It considers only one datapoint or observation of a dataset in every iteration or epoch for calculation of loss function, E(X, W) as shown in the following equation,

$$Loss\ Function, E(X, W) = \underbrace{(y_i - \hat{y}_i)^2}_{Squared\ Error}$$

- It gives a similar performance like batch gradient descent but takes more time to converge due to frequent updates. Weights have high variance and make loss function to variate to different range of values.

Optimization

- **Mini Batch Gradient Descent**
- Mini-batch gradient considers mini-batch of datapoints or observations of the dataset rather than considering all datapoints(as in batch gradient descent) or single datapoints(as in stochastic gradient descent), for calculation of loss function, over a mini-batch. It accordingly makes updations in the weights of connections between layers of neurons in the neural network.

$$Loss\ Function, E(X,W) = \underbrace{\frac{1}{K}\sum_{i=1}^{K}(y_i - \hat{y}_i)^2}_{Mean\ Squared\ Error}$$

- In the above formula, K represents the size of the batch of data points of the dataset, N, taken into consideration for the calculation of loss function, over a K data points in every iteration or epoch. The mini-batch gradient descent variant helps to converge faster than stochastic gradient descent due to comparatively less frequent updates in the weights or parameters of neural network with lower variance in the loss function.

Optimization

- **Adagrad**
- Adagrad (Adaptive Gradient) is a gradient-based optimization algorithm in which the learning rate updates with respect to the weights. It sets a low learning rate for the weights of connections between different layers of neurons in the neural network with commonly occurring features of data while setting it to a higher learning rate for the weights or parameters associated with uncommon features of data.
- Adagrad typically adapts different learning rates for every weight or parameter at every iteration or epoch based on their association with sparse or dense features. The weight updation formula in the adagrad is represented as follows,

$$W_t = W_{t-1} - \eta'_t * \frac{\partial E(X, W)}{\partial W}$$

Optimization

- **Adam Deep Learning Optimizer**
- The name adam is derived from adaptive moment estimation. This optimization algorithm is a further extension of stochastic gradient descent to update network weights during training. Unlike maintaining a single learning rate through training in SGD, Adam optimizer updates the learning rate for each network weight individually.
- The Adam optimizers inherit the features of both Adagrad and RMS prop algorithms. In adam, instead of adapting learning rates based upon the first moment(mean) as in RMS Prop, it also uses the second moment of the gradients.
- We mean the uncentred variance by the second moment of the gradients(we don't subtract the mean).
- The adam optimizer has several benefits, due to which it is used widely. It is adapted as a benchmark for deep learning papers and recommended as a default optimization algorithm. Moreover, the algorithm is straightforward to implement, has a faster running time, low memory requirements, and requires less tuning than any other optimization algorithm.

Hyper Parameter Tuning

Hyper parameters

- Learning rate – α
- Momentum – β
- Adam's hyperparameter – $\beta 1$, $\beta 2$, ε
- Number of hidden layers
- Number of hidden units for different layers
- Learning rate decay
- Mini-batch size


Thank you!