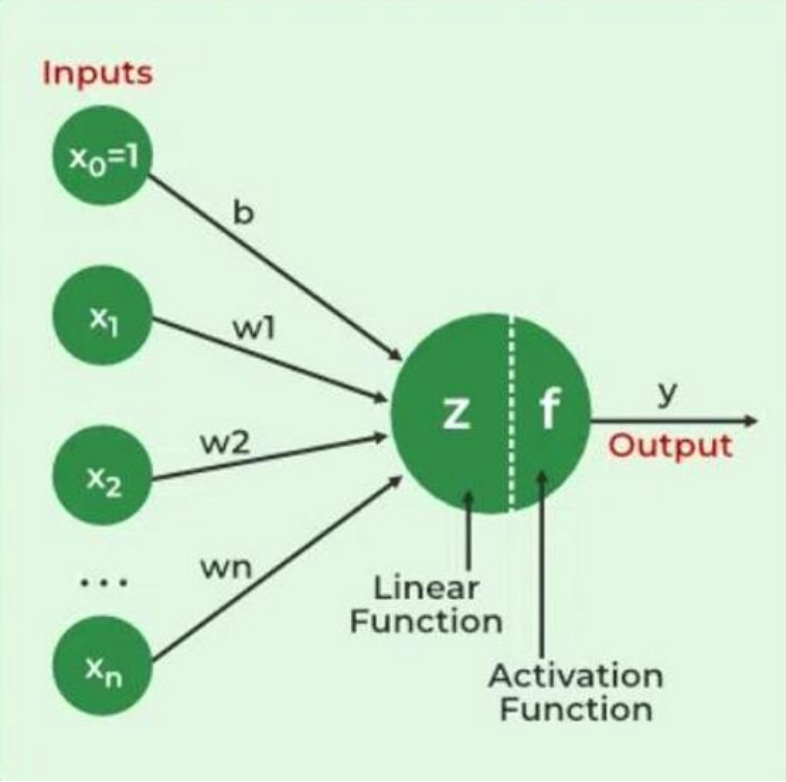
Neural Networks and Bias Variance Tradeoff and Optimizers

Neural Networks

Neural networks extract identifying features from data, lacking pre-programmed understanding. Network components include neurons, connections, weights, biases, propagation functions, and a learning rule. Neurons receive inputs, governed by thresholds and activation functions. Connections involve weights and biases regulating information transfer. Learning, adjusting weights and biases, occurs in three stages: input computation, output generation, and iterative refinement enhancing the network's proficiency in diverse tasks



Neural Networks has a lot of applications in todays world. It is used in natural language processing and self-driving automobiles to automating decision-making processes and increasing efficiency in numerous industries. The development of artificial intelligence is largely dependent on neural networks, which also drive innovation and influence the direction of technology.

Working of a neural Network

The input layer provides the necessary inputs to the neurons in the next layer and these inputs are multiplied by the weights and then passed through various layers in the neural network. If it is a binary classifier then the last layer will have a binary activation function making the necessary predictions. Two stages of the basic process are called backpropagation and forward propagation.

Forward Propagation

• Input Layer: Each feature in the input layer is represented by a node on the network, which receives input data.

• Weights and Connections: The weight of each neuronal connection indicates how strong the connection is. Throughout training, these weights are changed.

• Hidden Layers: Each hidden layer neuron processes inputs by multiplying them by weights, adding them up, and then passing them through an activation function. By doing this, non-linearity is introduced, enabling the network to recognize intricate patterns.

• Output: The final result is produced by repeating the process until the output layer is reached.

Backpropagation

• Loss Calculation: The network's output is evaluated against the real goal values, and a loss function is used to compute the difference. For a regression problem, the Mean Squared Error (MSE) is commonly used as the cost function. Loss Function: MSE = 1 $n\sum$ (yi − yi) 2

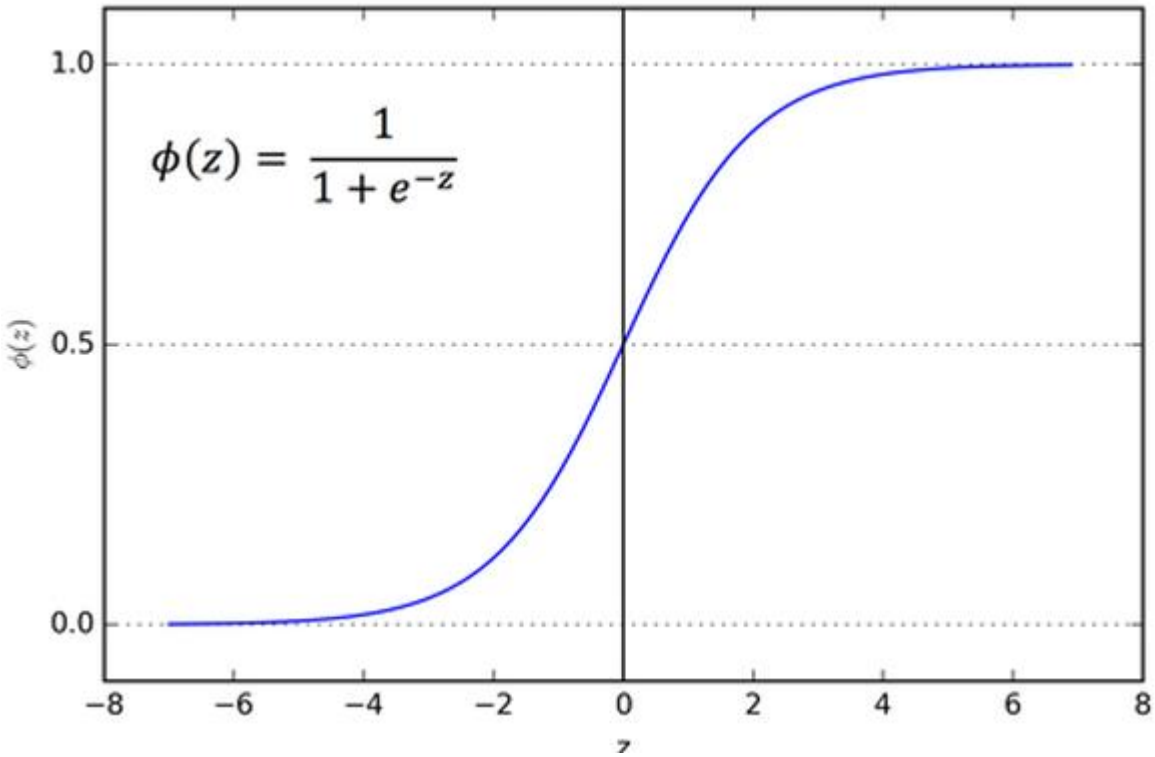
Gradient Descent: Gradient descent is then used by the network to reduce the loss. To lower the inaccuracy, weights are changed based on the derivative of the loss with respect to each weight.

• Adjusting weights: The weights are adjusted at each connection by applying this iterative process, or backpropagation, backward across the network.

• Training: During training with different data samples, the entire process of forward propagation, loss calculation, and backpropagation is done iteratively, enabling the network to adapt and learn patterns from the data. Activation Functions: Model non-linearity is introduced by activation functions like the rectified linear unit (ReLU) or sigmoid. Their decision on whether to "fire" a neuron is based on the whole weighted input.

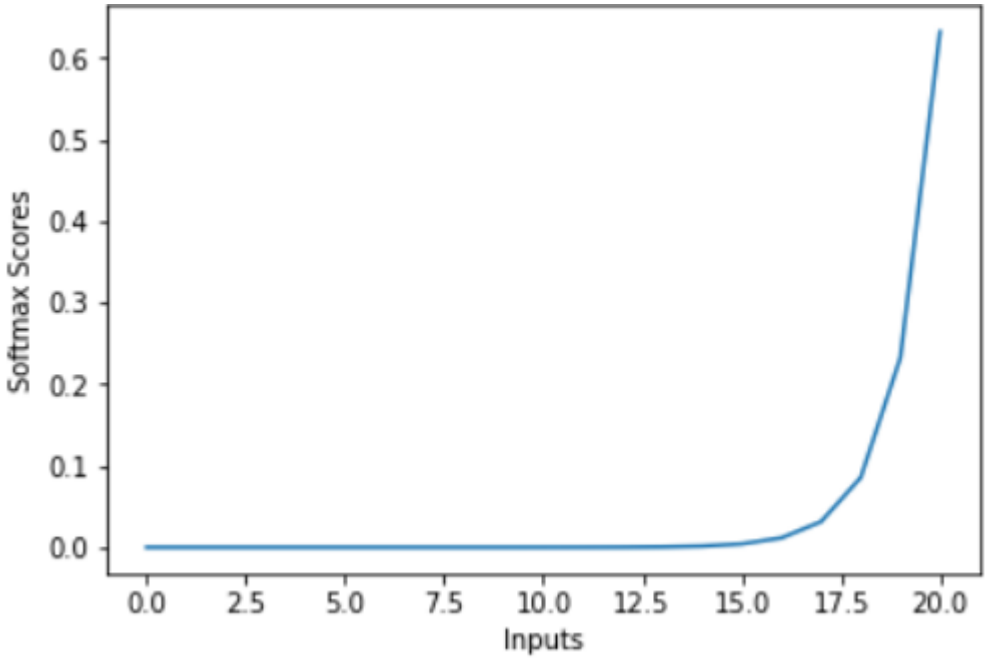
ACTIVATION FUNCTIONS

1) Sigmoid Activation function



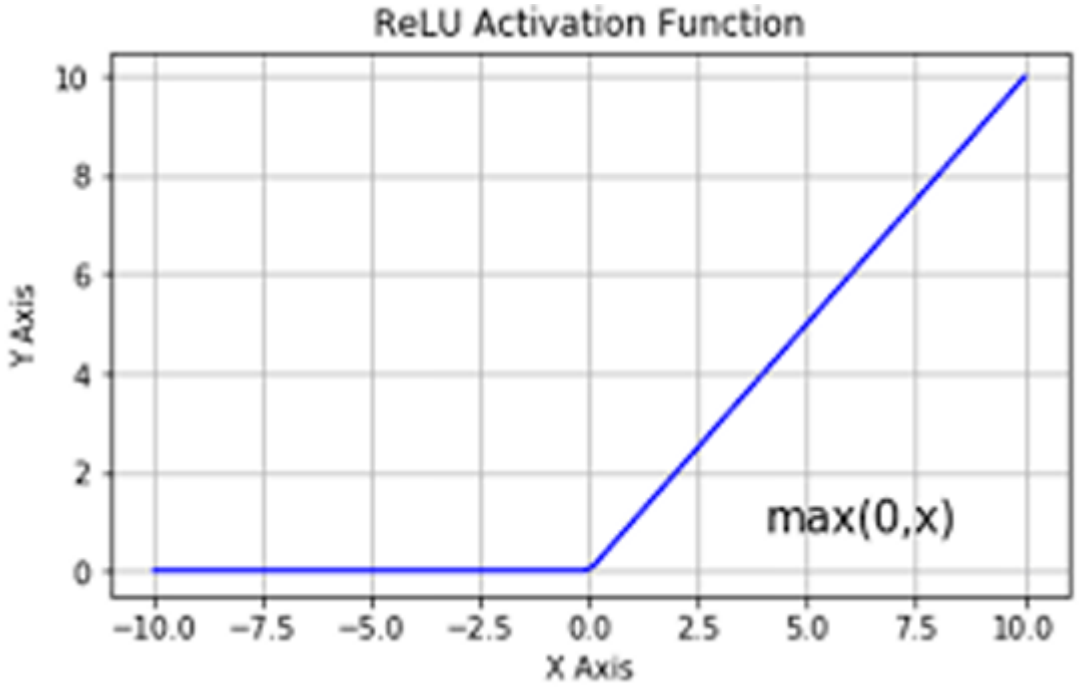
The main reason why we use sigmoid function is because it exists between (0 to 1). Therefore, it is especially used for models where we have to predict the probability as an output. Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice. The function is differentiable. That means, we can find the slope of the sigmoid curve at any two points. The function is monotonic but function's derivative is not. The logistic sigmoid function can cause a neural network to get stuck at the training time. The SoftMax function is a more generalized logistic activation function which is used for multiclass classification.

2) SoftMax



The SoftMax function is a type of sigmoid function used to handle multi class classification problems. It is nonlinear in nature. It is commonly used in the output layer of image classification problems, and other classifiers where the objective is to attain probabilities to define the class of each input. Hence different activation functions are optimal for different applications.

3) ReLU (Rectified Linear Unit) Activation Function



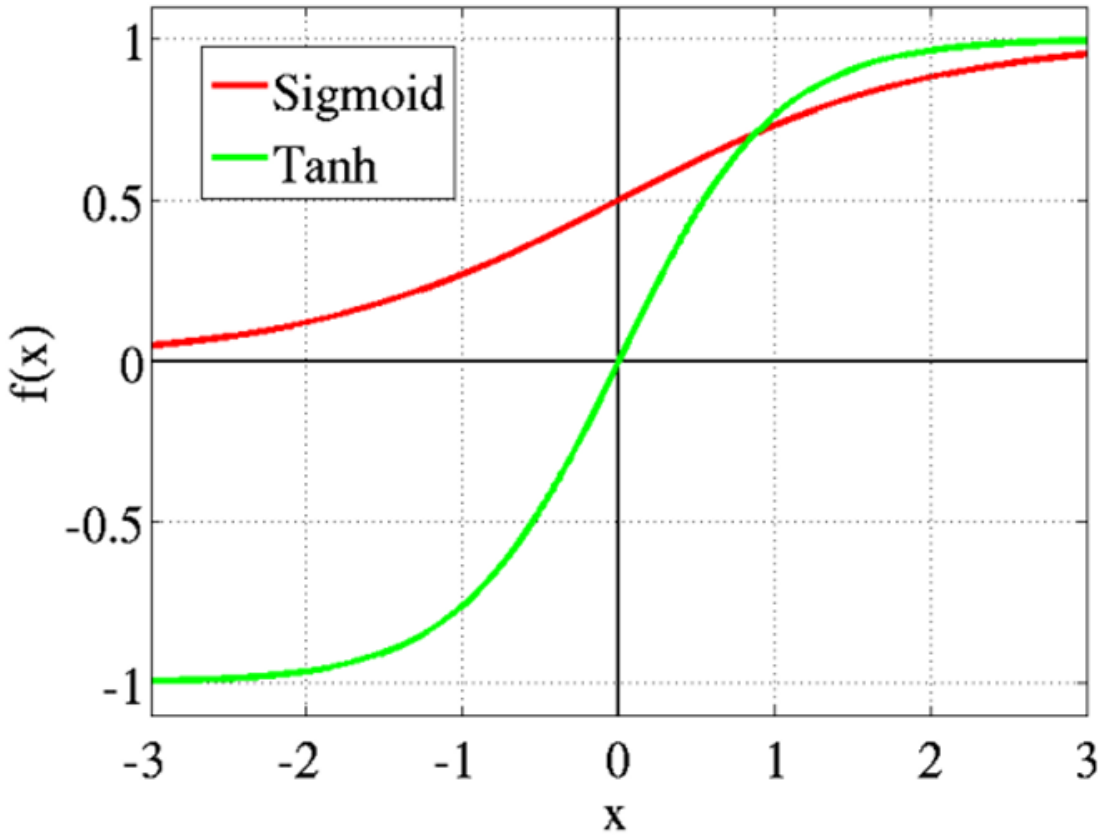
The ReLU is the most used activation function in the world right now.Since, it is used in almost all the convolutional neural networks or deep learning. As you

can see, the ReLU is half rectified (from bottom). f(z) is zero when z is less than zero and f(z) is equal to z when z is above or equal to zero.

Range: [0 to infinity)

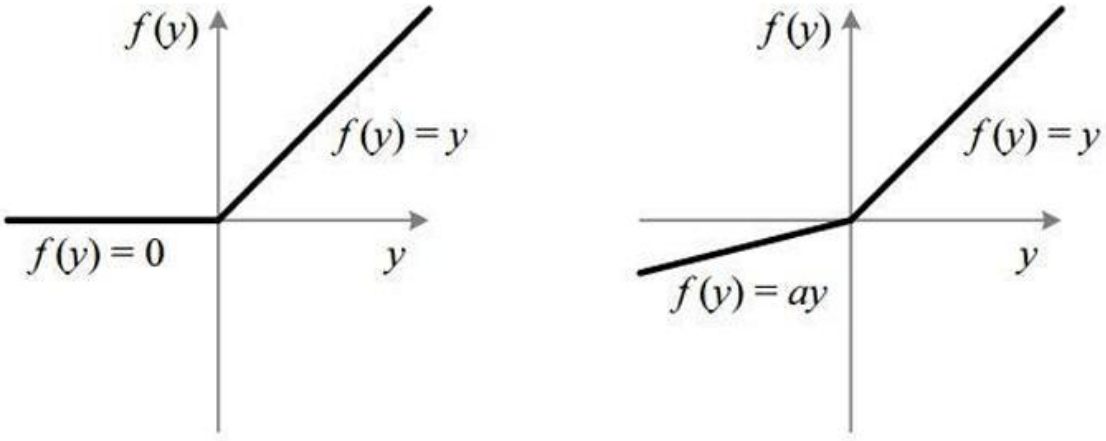
The function and its derivative both are monotonic. But the issue is that all the negative values become zero immediately which decreases the ability of the model to fit or train from the data properly. That means any negative input given to the ReLU activation function turns the value into zero immediately in the graph, which in turns affects the resulting graph by not mapping the negative values appropriately.

4) Tanh or hyperbolic tangent Activation Function



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. The function is differentiable.The function is monotonic while its derivative is not monotonic.The tanh function is mainly used classification between two classes.

5) Leaky ReLU



The leak helps to increase the range of the ReLU function. Usually, the value of a is 0.01 or so.When a is not 0.01 then it is called Randomized ReLU.Therefore the range of the Leaky ReLU is (-infinity to infinity).Both Leaky and Randomized ReLU functions are monotonic in nature. Also, their derivatives also monotonic in nature.

Hyperparameter tuning techniques

1) GridSeach: Grid search can be considered as a “brute force” approach to hyperparameter optimization. We fit the model using all possible combinations after creating a grid of potential discrete hyperparameter values. We log each set’s model performance and then choose the combination that produces the best results. This approach is called GridSearch, because it searches for the best set of hyperparameters from a grid of hyperparameters values. It is a very exhaustive approach that can identify the ideal hyperparameter combination is grid search. But the slowness is a disadvantage. It often takes a lot of processing power and time to fit the model with every potential combination, which might not be available.
   For example: if we want to set two hyperparameters C and Alpha of the Logistic Regression Classifier model, with different sets of values. The grid search technique will construct many versions of the model with all possible combinations of hyperparameters and will return the best one. As in the image, for C = [0.1, 0.2, 0.3, 0.4, 0.5] and Alpha = [0.1, 0.2, 0.3, 0.4]. For a

combination of C=0.3 and Alpha=0.2, the performance score comes out to be 0.726(Highest), therefore it is selected.

C \ Alpha	0.1	0.2	0.3	0.4
0.5	0.701	0.703	0.697	0.696
0.4	0.699	0.702	0.698	0.702
0.3	0.721	0.726	0.713	0.703
0.2	0.706	0.705	0.704	0.701
0.1	0.698	0.692	0.688	0.675

2) Randomized Search

As the name suggests, the random search method selects values at random as opposed to the grid search method's use of a predetermined set of numbers. Every iteration, random search attempts a different set of hyperparameters and logs the model's performance. It returns the combination that provided the best outcome after several iterations. This approach reduces unnecessary computation. RandomizedSearch solves the drawbacks of GridSearch, as it goes through only a fixed number of hyperparameter settings. It moves within the grid in a random fashion to find the best set of hyperparameters. The advantage is that, in most cases, a random search will produce a comparable result faster than a grid search.

Regularization Techniques

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.

1) L1 Regularization

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum \|w\|$$

L1 regularization works by adding a penalty term to the standard loss function (usually the sum of squared errors) that is proportional to the absolute values of the coefficients of the features. Mathematically, it

adds the sum of the absolute values of the coefficients multiplied by a constant (alpha) to the loss function. This additional penalty encourages the model to shrink the coefficients of less important features towards zero, effectively performing feature selection by setting some coefficients to exactly zero. As a result, L1 regularization can yield sparse models where only a subset of features are considered important. Overall, L1 regularization helps in controlling model complexity and can lead to better generalization performance, especially when dealing with high-dimensional data or datasets with many irrelevant features.

2) L2 regularization
   A regression model that uses the L2 regularization technique is called Ridge regression. Ridge regression adds the “squared magnitude” of the coefficient as a penalty term to the loss function(L). similar to L1 regularization, L2 regularization involves adding a penalty term to the standard loss function (usually the sum of squared errors). However, instead of penalizing the absolute values of the coefficients like L1 regularization does, L2 regularization penalizes the square of the coefficients. Mathematically, L2 regularization adds the sum of the squared coefficients multiplied by a constant (alpha) to the loss function. This additional penalty encourages the model to distribute the weight of the coefficients more evenly across all features, as it penalizes large coefficients but doesn't force them to be exactly zero. As a result, L2 regularization typically leads to smoother coefficient estimates compared to L1 regularization.

3) Dropout regularization
   Dropout is a regularization technique which involves randomly ignoring or “dropping out” some layer outputs during training, used in deep neural networks to prevent overfitting. Dropout is implemented per-layer in various types of layers like dense fully connected, convolutional, and recurrent layers, excluding the output layer. The dropout probability specifies the chance of dropping outputs, with different probabilities for input and hidden layers that prevents any one neuron from becoming too specialized or overly dependent on the presence of specific features in the training data. During training, dropout randomly deactivates a chosen proportion of neurons (and their connections) within a layer. This essentially temporarily removes them from the network. The deactivated neurons are chosen at random for each training iteration. This randomness is crucial for preventing overfitting. To account for the deactivated neurons, the outputs of the remaining active neurons are

scaled up by a factor equal to the probability of keeping a neuron active (e.g., if 50% are dropped, the remaining ones are multiplied by 2).

4) Data augmentation
   Data augmentation is the process of increasing the amount and diversity of data. We do not collect new data, rather we transform the already present data. I will be talking specifically about image data augmentation in this article. So we will look at various ways to transform and augment the image data. The most commonly used operations are- Rotation Shearing Zooming Cropping Flipping Changing the brightness level Data augmentation is an integral process in deep learning, as in deep learning we need large amounts of data and in some cases it is not feasible to collect thousands or millions of images, so data augmentation comes to the rescue. It helps us to increase the size of the dataset and introduce variability in the dataset.

   Optimization algorithms

   1) Gradient Descent
      Gradient Descent is a fundamental optimization algorithm in machine learning used to minimize the cost or loss function during model training. It iteratively adjusts model parameters by moving in the direction of the steepest decrease in the cost function. The algorithm calculates gradients, representing the partial derivatives of the cost function concerning each parameter. These gradients guide the updates, ensuring convergence towards the optimal parameter values that yield the lowest possible cost.

   2) Mini batch Gradient Descent
      Mini batch gradient Descent is used when the dataset is too large and applying gradient descent on a very large data set will be computationally expensive. In this process, the dataset is divided into a number of batches of size (m) and gradient descent is applied on each batch. This is a much faster method then applying gradient descent on the whole dataset.

   3) Momentum
      An Adaptive Optimization Algorithm uses exponentially weighted averages of gradients over previous iterations to stabilize the convergence, resulting in quicker optimization. For example, in most real-world applications of Deep Neural Networks, the training is carried out on noisy data. It is, therefore, necessary to reduce the

effect of noise when the data are fed in batches during Optimization. This problem can be tackled using Exponentially Weighted Averages (or Exponentially Weighted Moving Averages).

4) RMSprop

RMSprop was proposed by the University of Toronto's Geoffrey Hinton. The intuition is to apply an exponentially weighted average method to the second moment of the gradients (dW2). This algorithm works by exponentially decaying the learning rate every time the squared gradient is less than a certain threshold. This helps reduce the learning rate more quickly when the gradients become small. In this way, RMSProp is able to smoothly adjust the learning rate for each of the parameters in the network, providing a better performance than regular Gradient Descent alone.

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta)\left(\frac{\delta C}{\delta w}\right)^2$$

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{E[g^2]_t}} \frac{\delta C}{\delta w}$$

E[g] — moving average of squared gradients. dC/dw — gradient of the cost function with respect to the weight.

n — learning rate. Beta — moving average parameter (good default value — 0.9)

5) Adam optimizer

Adam optimizer, short for Adaptive Moment Estimation optimizer, is an optimization algorithm commonly used in deep learning. It is an extension of the stochastic gradient descent (SGD) algorithm and is designed to update the weights of a neural network during training. The name "Adam" is derived from "adaptive moment estimation," highlighting its ability to adaptively adjust the learning rate for each network weight individually. Unlike SGD, which maintains a single learning rate throughout training, Adam optimizer dynamically computes individual learning rates based on the past gradients and their second moments. The creators of Adam optimizer incorporated the beneficial features of other optimization

algorithms such as AdaGrad and RMSProp. Similar to RMSProp, Adam optimizer considers the second moment of the gradients, but unlike RMSProp, it calculates the uncentered variance of the gradients (without subtracting the mean). By incorporating both the first moment (mean) and second moment (uncentered variance) of the gradients, Adam optimizer achieves an adaptive learning rate that can efficiently navigate the optimization landscape during training. This adaptivity helps in faster convergence and improved performance of the neural network. In summary, Adam optimizer is an optimization algorithm that extends SGD by dynamically adjusting learning rates based on individual weights. It combines the features of AdaGrad and RMSProp to provide efficient and adaptive updates to the network weights during deep learning training.

Adam optimizer formula

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \left[\frac{\delta L}{\delta w_t} \right] v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left[\frac{\delta L}{\delta w_t} \right]^2$$

Here B1 and B2 represent the decay rate of the average of the gradients.

Bias

It refers to how far off the models predictions are from the true value resulting from a very simple model. It's a measure of how far off your models predictions are on average from the underlying relationship in the data.

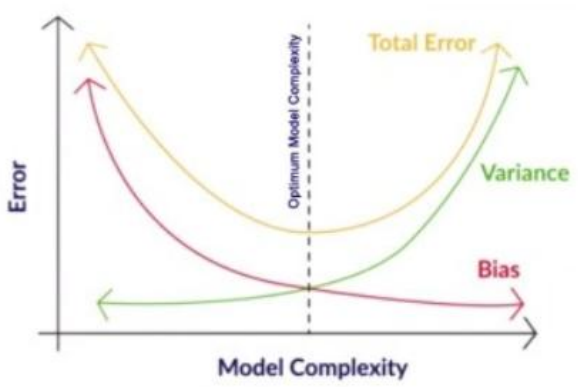
High Bias = Underfitting. To fix high bias we usually increase model complexity

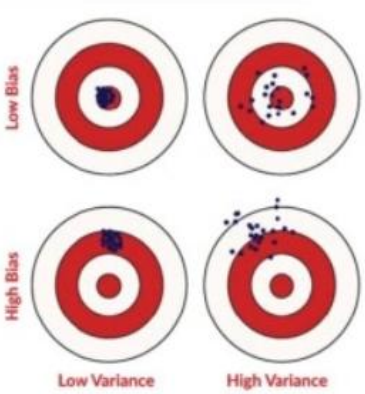
Variance

It refers to how much the models predictions change when the dataset is changed. A model with high variance captures noise also thus performing poorly on unseen data. High variance = Overfitting. To fix high variance we reduce model complexity by using regularization, reducing the number of features or just gathering more data

Bias Variance Trade off

You cannot minimize both simultaneously. A balanced should be maintained


Error
Optimum Model Complexity
Total Error
Variance
Bias
Model Complexity


Low Bias
High Bias
Low Variance
High Variance