Weight Initializers

1) We can use weight initializers in every hidden layer of an artificial neural network (ANN). In fact, it is generally recommended to use a weight initializer in all layers of an ANN, as it can help to improve the convergence of the network and prevent it from overfitting.

2) Weight initialization is important for both the input and hidden layers of a neural network. It helps improve the convergence speed and stability of the training process.

3) Weight initializers are used in all layers of a neural network, not just the input layer. The best weight initialization practice depends on the activation function used in the neural network. However, there is no hard and fast rule that says which weight initializers should be used with which activation functions. Experimentation is key to finding the best weight initializers for your particular problem.

4) There is no rule that says which weight initializers should be used with which activation functions. It is important to experiment with different weight initializers to see which ones work best for your particular problem.

5) Proper weight initialization can help the neural network learn from the data in a more efficient way. It can also help prevent the network from converging to a local minimum.

6) There are many different weight initialization techniques available. The best technique to use depends on the specific task and the architecture of the ANN. It is important to experiment with different weight initialization techniques to see which ones work best for your particular task.

7) For neural networks with ReLU activation functions, it is common to use a He initialization. He initialization ensures that the weights are initialized in a way that prevents the vanishing gradient problem.

8) He initialization is a popular weight initialization technique for neural networks with ReLU activation functions. It ensures that the weights are initialized in a way that prevents the vanishing gradient problem.

9) Weight initialization is an important step in training a neural network. By initializing the weights properly, we can help the network to learn more quickly and efficiently.

10) For very deep neural networks, it may be beneficial to use different initializers at different layers. This is because the deeper layers of the network will need to learn more complex features, and using a different initializer can help to prevent the network from becoming too unstable.

11) For shallow neural networks, it may be sufficient to use the same initializer in every layer. This is because the shallower layers of the network will not need to learn as complex features, and using the same initializer can help to ensure that the network converges more quickly.

Q) Which weight initializer is used when?

(I) Xavier / Glorot Initialization: Used when we use Sigmoid or Tanh Activation Function.

(II) He Initialization: Used with ReLU.

Note: We have 2 types of both `normal` and `uniform`. Perform trail and test and then select better one.

Neurons

Q) How to specify number of neurons in each layer?

(I) Input layer: The number of neurons in the input layer is equal to the number of independent features in the dataset.

(II) Hidden layer: The number of neurons in the hidden layers is typically between the number of neurons in the input layer and the number of neurons in the output layer. However, there is no hard and fast rule for this, and it is often necessary to experiment with different configurations to find the best results.

(III) Output layer: In the case of regression and binary classification, the number of neurons in the output layer is one. In the case of multi-class classification and image data, the number of neurons in the output layer depends on the number of categories in the dependent variable.

Activation Function

Q) Which activation function is applied in which layer and when?

(I) Regression: ReLU + Linear

(II) Binary Classification: ReLU + Sigmoid

(III) Multi-Class Classification: ReLU + SoftMax

\* Note: There is no such rule that always use these combinations. Perform trial and error to choose better combination than this.

Dropout Regularization

Here are some guidelines for using dropout in ANNs:

1) Smaller layers: Use a lower dropout percentage, such as 0.1 or 0.2.

2) Larger layers: Use a higher dropout percentage, such as 0.3 or 0.4.

3) More complex tasks: Use a higher dropout percentage.

4) Less training data: Use a higher dropout percentage.

Batch Normalization

Best Practice: -

(I) Apply Batch Normalization after Every Hidden Layer (or Almost All)

(II) Avoid Batch Normalization in the Output Layer

Here are the steps of how batch normalization works in ANNs:

1. Input data (features) are fed into the layer.
2. The layer applies a linear transformation (weights and biases) to the input data.
3. Batch normalization is applied to the output of the linear transformation. This ensures that the output of the layer has a mean of 0 and a standard deviation of 1.
4. The output of the batch normalization layer is passed through an activation function, producing the activations of the layer.

Q) Batch normalization is used ANN and when with code and best practice to apply batch normalization?

**(I) Yes, batch normalization can be used in artificial neural networks (ANNs).** In fact, batch normalization was originally introduced for feedforward ANNs and has since been widely adopted in various neural network architectures.

(II) Batch normalization converts the values of the input to the range of (0, 1). This helps to ensure that the input to the activation function is on a similar scale, which can help to improve the stability of training and the convergence of the network.

(III) If we apply batch normalization after the activation function, the output of the activation function will also be converted to the range of (0, 1). This can lead to the problem of vanishing gradients, which occurs when the gradients of the activation function become very small. This can make it difficult for the network to learn, as the updates to the weights will be very small.

