simplilearn | PURDUE UNIVERSITY

Deep Learning with Keras with TensorFlow

Deep Neural Net Optimization, Tuning, and Interpretability

simplilearn

Learning Objectives

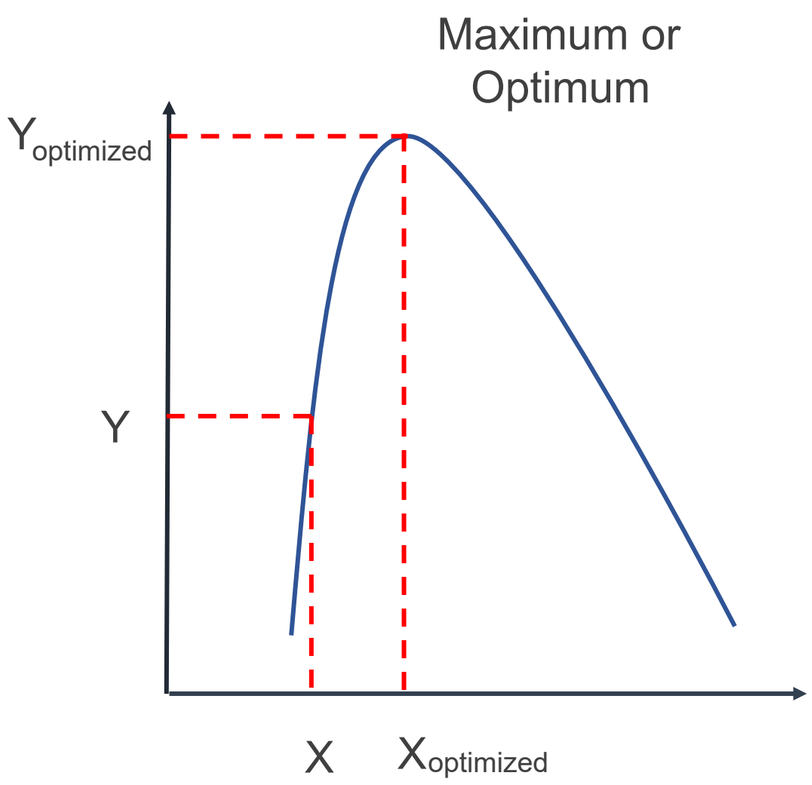
By the end of this lesson, you will be able to:

- Explain algorithms of optimization
- Perform batch normalization
- Describe hyperparameter tuning and its significance
- Explain interpretability in deep learning

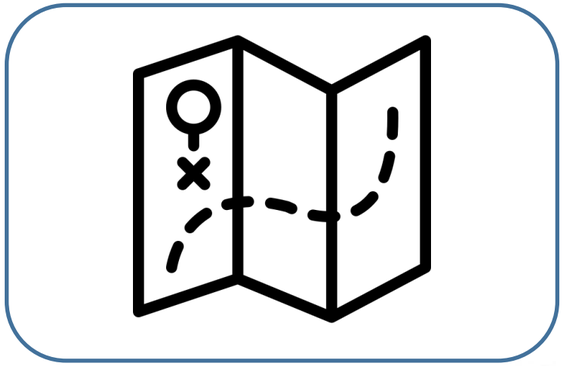
Optimization

What Is Optimization?

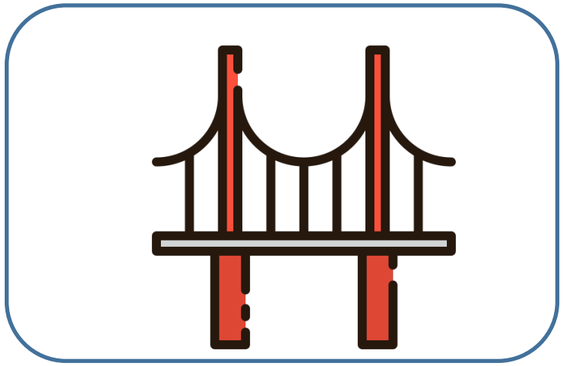
Optimization is choosing input to obtain the best possible output.



Warehouse Location → Warehouse Placement → Minimize Shipment



Design → Bridge Construction → To Carry Maximum Load



Optimization Algorithm

Optimization Algorithms

- Algorithms which are used to solve optimization problems are called optimization algorithm. In deep learning, optimization algorithms are used to optimize cost function J.

$$J(W, b) = \sum_{i=1}^{m} L(y'^{i}, y^{i})$$

Optimization Algorithms

$$J(W,b) = \sum_{i=1}^{m} L(y'^{i}, y^{i})$$

- The value of cost function ***J*** is the mean of the loss ***L*** between the predicted value ***y'*** and actual value ***y***.
- The value ***y'*** is obtained during the forward propagation step and makes use of the weights ***W*** and biases ***b*** of the network.
- With the help of optimization algorithms, we minimize the value of Cost Function ***J*** by updating the values of the trainable parameters ***W*** and ***b***

Types of Optimizers

- GD
- SGD
- SGD with Minibatch
- SGD with Momentum
- Nesterov Accelerated Gradient (NAG)
- Adaptive Gradient (Adagrad)
- Root Mean Square Propagation (RMSprop)
- Adaptive Momentum Estimation (Adam)

Gradient Descent (GD)

- GD is used to minimize the cost function **J** and obtain the optimal weight **W** and bias **b**.

- This equation represents change in Matrix **W**.

Learning Rate

$$W = W - \alpha \frac{\partial}{\partial W} J(W)$$

- This equation represents change in bias **b**.

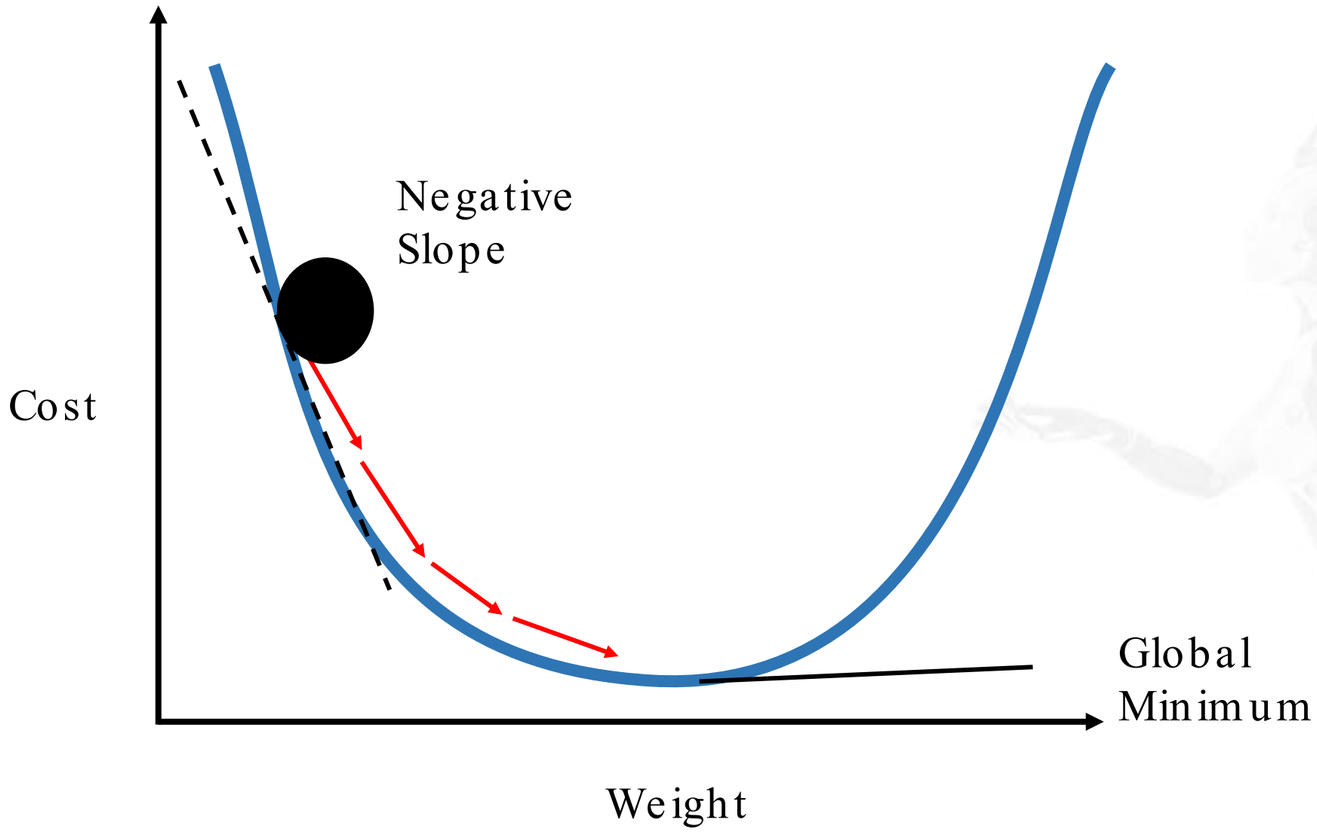
$$b = b - \alpha \frac{\partial}{\partial b} J(b)$$

- The change in values is determined by learning rate and derivatives of J with respect to **W** and **b**. This process is repeated until ***J*** has been minimized.

Gradient Descent

If the slope, the partial derivative with respect to **W** is negative at a point, then the **W** increases to achieve the global minimum.

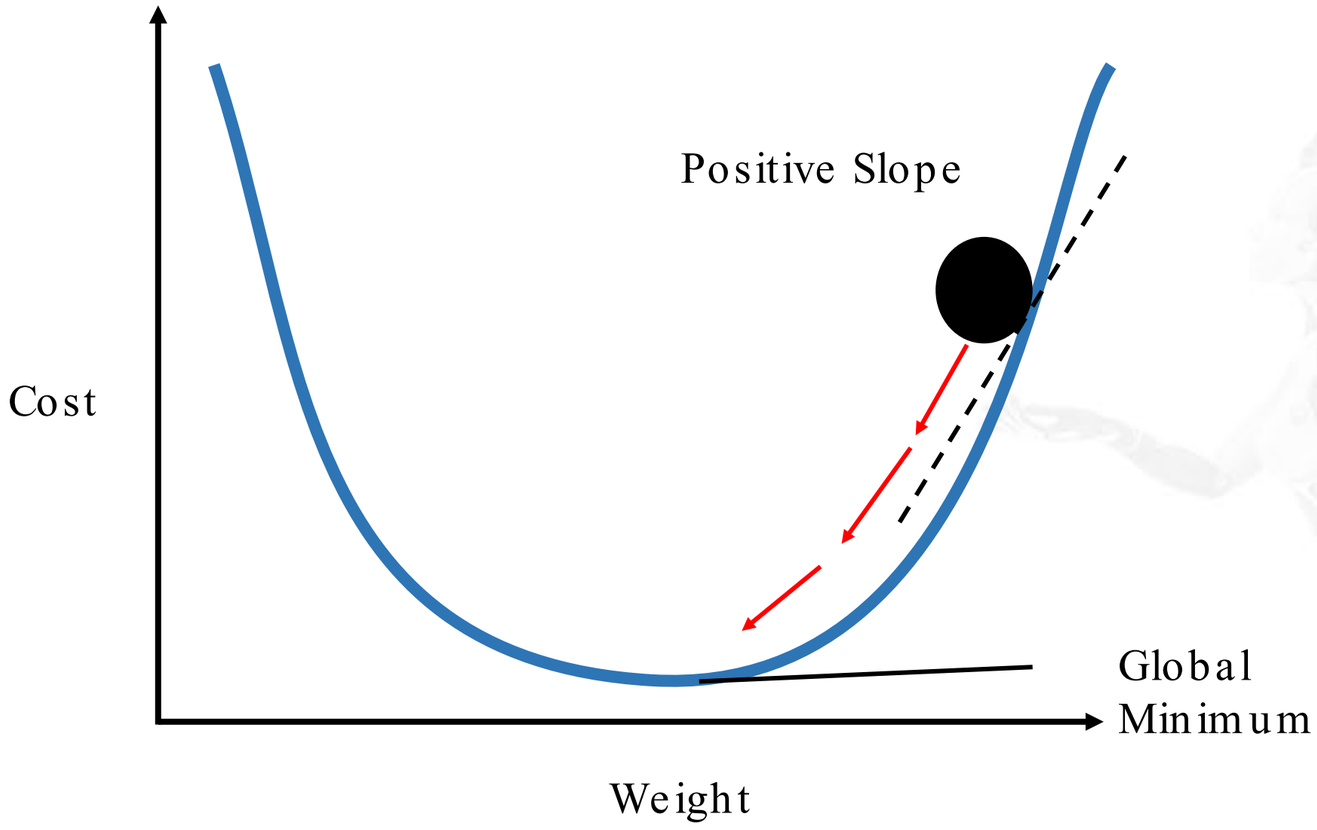
$$W = W + \alpha \frac{\partial}{\partial W} J(W)$$



Gradient Descent

If the slope, the partial derivative with respect to **W** is positive at a point, then the **W** decreases to achieve the global minimum.

$$W = W - \alpha \frac{\partial}{\partial W} J(W)$$



Stochastic Gradient Descent (SGD)

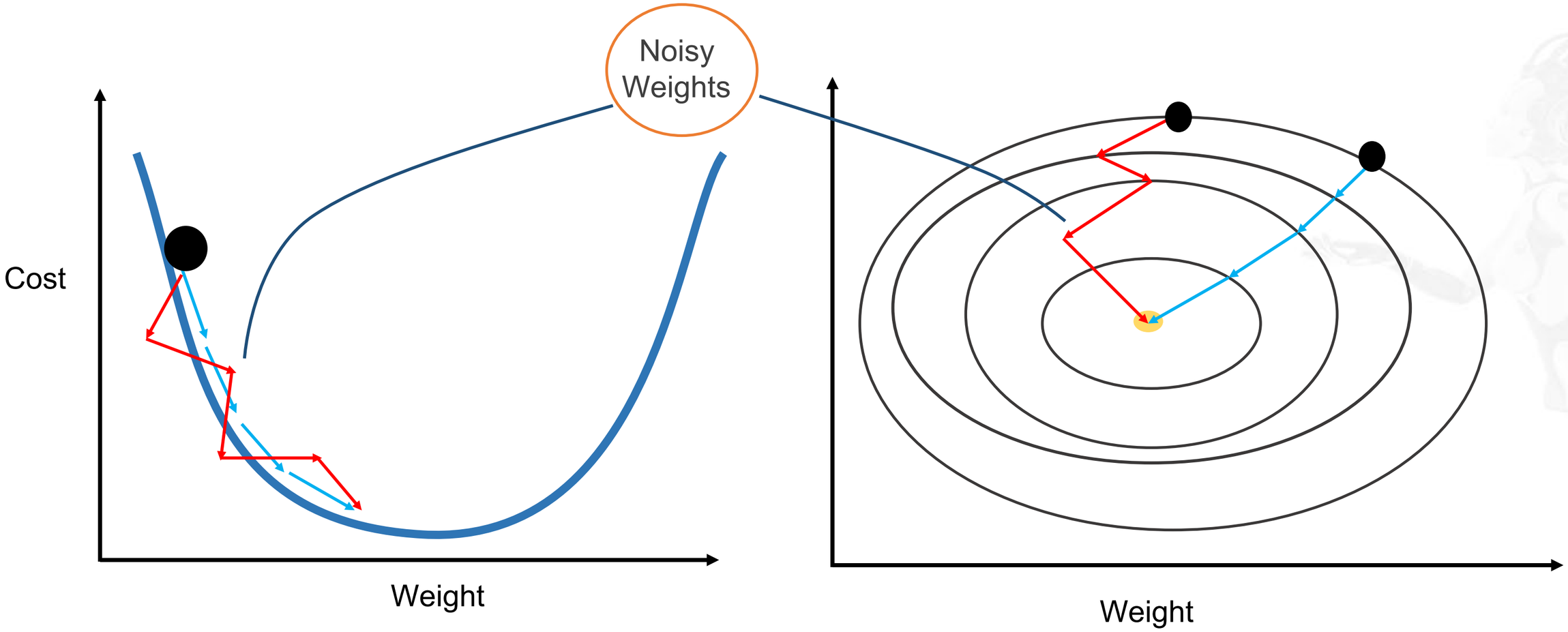
- Single data points are taken to find the optimized weights.
- The mathematical formulation for the weight evaluation for SGD is same as GD, but the data points are shuffled before using them for optimization.
- Random data points go into the optimizer and result into random weights, that is the resulted weights are noisy.

Stochastic Gradient Descent -Mini Batch (SGD -Mini Batch)

- Is combination of vanilla GD and SGD which distributes the whole training data in small mini -batches
- Divides the training data into small batches, so that the network can easily be trained on the data
- The mathematical formulation is same as vanilla GD, but the training occurs batch wise
- For example, training set has 400 training examples which are divided into 10 batches with each batch containing 40 training examples. Thus, the weight evaluation equation will be iterated over 10 times (number of batches).

GD vs. SGD-Mini Batch

GD is computationally expansive, but it converges into global minimum smoothly. On the other hand, in SGD, there is more noisy weight created which takes more time to reach the global minimum.



SGD with Momentum

- SGD with momentum or just momentum is an advanced optimization algorithm that uses moving average to update the trainable parameters.
- SGD with momentum is a very suitable method to overcome the noisy weights of SGD.

SGD with Momentum

- Moving average is a calculation to analyze data points by creating a series of averages of different subsets of the full data set formulation. For example, for time t_1, t_2, t_3 you have corresponding data points b_1, b_2, b_3.
- According to the moving average:

$$V_1 = b_1$$

$$V_2 = \gamma \times V_1 + b_2$$

$$V_3 = \gamma \times V_2 + b_3 = \gamma^2 b_1 + \gamma b_2 + b_3$$

for $\gamma = 0.5$, $V_2 = 0.5b_1 + b_2$

$$V_3 = 0.25b_1 + 0.5b_2 + b_3$$

- Variable at time t_1

SGD with Momentum

- Apply moving average in the weight evaluation formulation.

$$W_{new} = W_{old} - \alpha \times J(W_{old})$$
$$= W_{old} - [\gamma V_{t-1} + \alpha \times J(W)]$$

Momentum

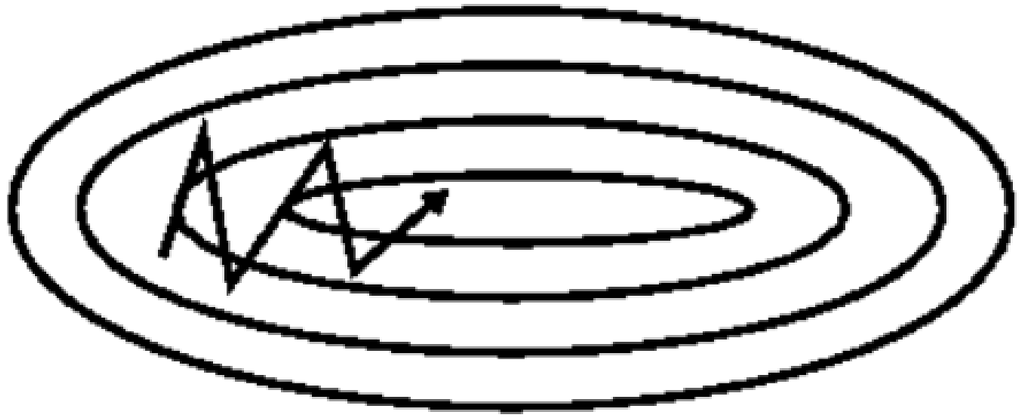
$$V_{t-1} = 1 \times J(W)_t + \gamma \times J(W)_{t-1} + \gamma \times J(W)_{t-1} + \gamma^2 \times J(W)_{t-2} \text{ - - - - - -}$$

- It is observed that $\gamma = 0.9$ works well in most cases.

SGD with and without Momentum

SGD with momentum clearly shows that momentum makes the steps smooth and less noisy.

SGD without momentum



SGD with momentum

Nesterov Accelerated Gradient (NAG)

- In NAG, interim parameters are observed if the velocity update leads to bad loss.
- In NAG, an interim velocity weight is calculated which is further used to calculate the weight with the help of a velocity factor.
- The difference between momentum method and NAG is in the gradient computation phase.

NAG vs. SGD with Momentum

Both methods give distinct output when the learning rate η is reasonably large. In such a case, NAG allows larger decay rate α than SGD with momentum method while preventing oscillations. The theorem also shows that both SGD with momentum and NAG become equal when η is small.

Learning Rate of GD, SGD, and SGD -Mini Batch

- The learning rate in weight initialization using the optimizers GD, SGD, and SGD-mini batch remains constant. This is a draw back of the optimizers with constant learning rate throughout all epochs.

Constant Learning Rate

$$W_{new} = W_{old} - \alpha \times J(W_{old})$$

- The features of the datasets used in ANNs are divided into two major types, one is sparse and another is dense. In dense, most of the values are nonzero, whereas in sparse it is completely the opposite. In that case, using a constant learning rate throughout all the epochs for one neural network is not an efficient approach.

Adaptive Gradient (AdaGrad)

- AdaGrad optimizer uses different learning rate for each neuron in each hidden layer throughout all epochs.

- The mathematical formulation for AdaGrad optimizer:

$$W_t = W_{t-1} - \alpha^*{}_t \times J(W_{t-1})$$

$$\alpha^*{}_t = \frac{\alpha}{\sqrt{\beta_t + \varepsilon}}$$

$$\beta t = \sum_{i=1}^{t} J(W_i)^{\,2}$$

Initial Learning Rate

AdaGrad

- As the epochs increase, the learning rates decrease which results into gradual decrease in weight.

- ε is considered in the formula because if βt becomes zero, then the weight will be constant.

$$\alpha^*{}_t = \frac{\alpha}{\sqrt{\beta_t + \varepsilon}}$$

Small Nonzero Value

- The only drawback with AdaGrad is that βt value sometimes can turn out to be very large.

$$\beta t = \sum_{i=1}^{t} J(W_i)^2$$

Root Mean Square Propagation (RMSprop)

- RMSprop is developed to take care of the drawback of AdaGrad.

- The formula for RMSprop is as follows:

$$W_t = W_{t-1} - \alpha^*{}_t \times J(W_{t-1})$$

$$\alpha^*{}_t = \frac{\alpha}{\sqrt{W_{avg(t)} + \varepsilon}}$$

$$W_{avg(t)} = \gamma \times W_{avg(t-1)} + (1 - \gamma) J(W)^2$$

Adaptive Momentum Estimation (Adam)

- Adam is the most cutting edge and advanced optimizer at present.
- It is the combination of RMSprop and momentum method.
- The formula for Adam is as follows:

$$W_t = W_{t-1} - \alpha^*{}_t \times J(W_{t-1})$$

$$\alpha^*{}_t = \frac{\alpha}{\sqrt{W_{avg(t)} + \varepsilon}} \times M_{i(t)}$$

$$W_{avg(t)} = \gamma \times W_{avg(t-1)} + (1 - \gamma) J(W)^2$$

Adam

$$W_t = W_{t-1} - \alpha^*{}_t \times J(W_{t-1})$$

$$\alpha^*{}_t = \frac{\alpha}{\sqrt{W_{avg(t)} + \varepsilon}} \times M_{i(t)}$$

To introduce momentum

$$W_{avg(t)} = \gamma \times W_{avg(t-1)} + (1 - \gamma) J(W)^2$$

- In this formula, there is one additional term to introduce, i.e. momentum $M_{i(t)}$

$$M_{i(t)} = momentum \times M_{i(t-1)} + (1 - momentum) \times J(W_{i(t)})$$

$$M_{i(0)} = 0$$

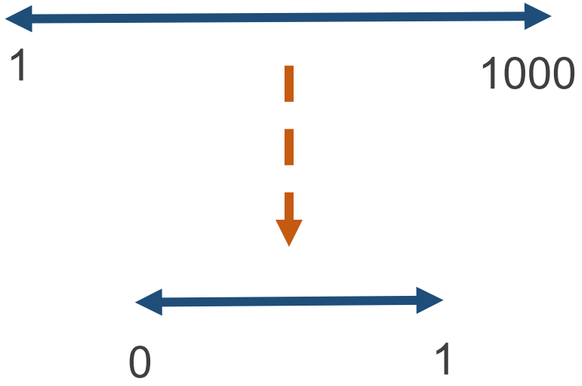
Batch Normalization

Data Preprocessing

In preprocessing, the data is generally normalized or standardized.

Normalization

A typical normalization is scaling down a large range of data into a smaller range.



Standardization

A typical standardization is to subtract the mean of all the data points from each data point and then dividing the difference by the standard deviation.

$$Z = \frac{X - m}{\sigma}$$

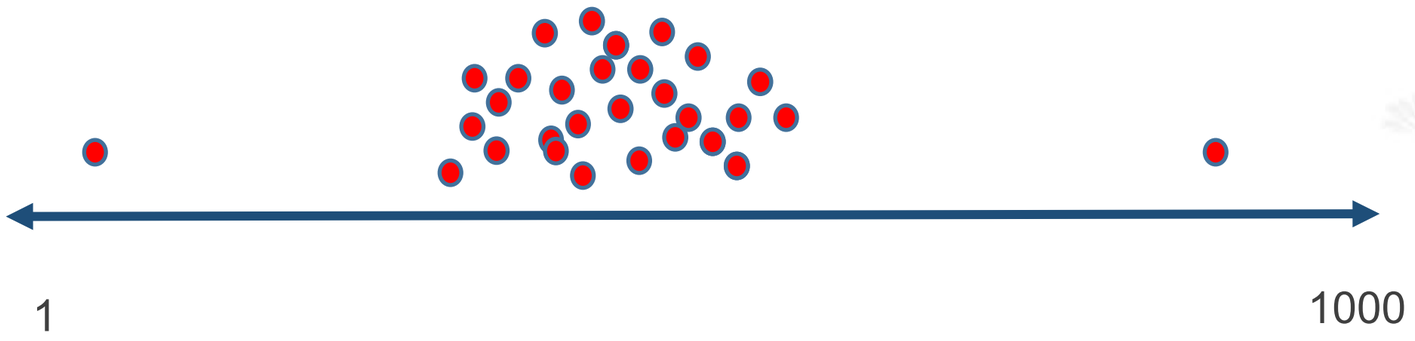
X = Data points

m = Mean

σ = Standard Deviation

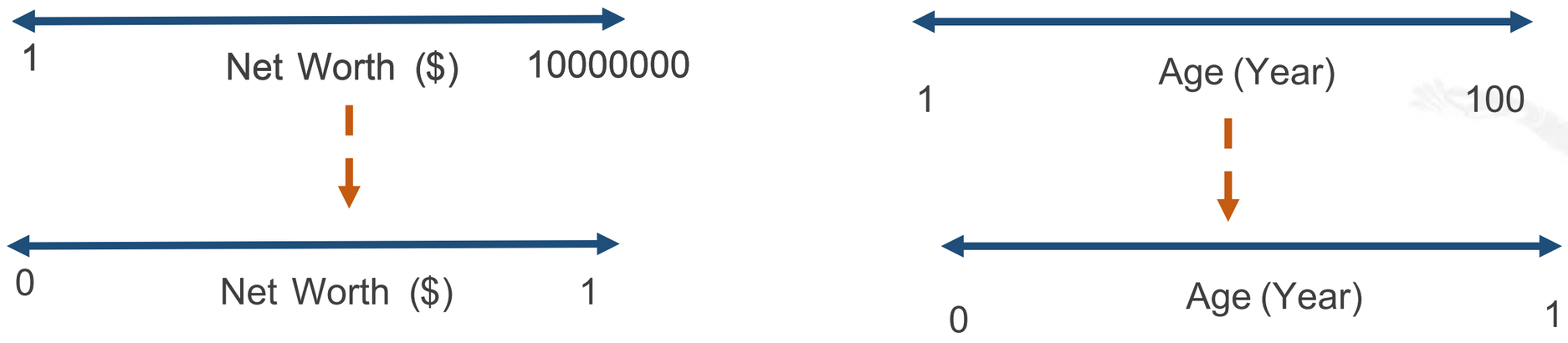
Why Data Preprocessing?

Data points can either be high or low. This leads to cascading of the network. Therefore, data preprocessing is needed.



Why Data Preprocessing?

When there are multiple features each with different range of data points, the non -processed data creates instability. It further cascades through the neural network layers. Scaling the different ranges to a standard range leads to stability and brings in better results.

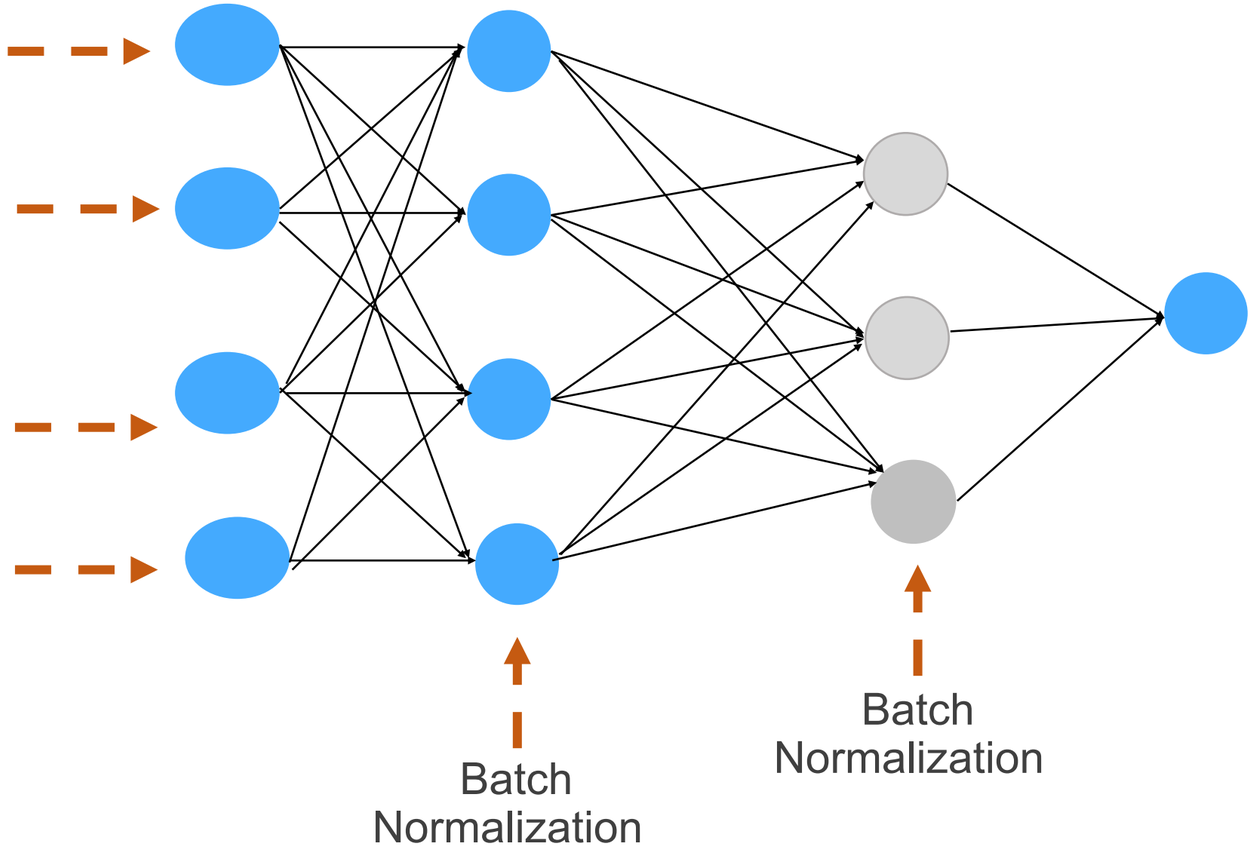


Batch Normalization

- Weights of the neural network get updated during the training period, in each epoch. Suppose, weight assigned to a neuron suddenly become large, it cascades through all the layers which further causes instability.

- Normalization of data before feeding into network is not enough, the outputs from the neurons should also be normalized. This is where batch normalization comes into the picture.

Batch Normalization



Batch Normalization

- It normalizes the output from activation function before passing it to the next layer as input.

$$Z = \frac{X - m}{\sigma}$$

- Normalized output is multiplied with arbitrary parameter g.

$$(Z \times g)$$

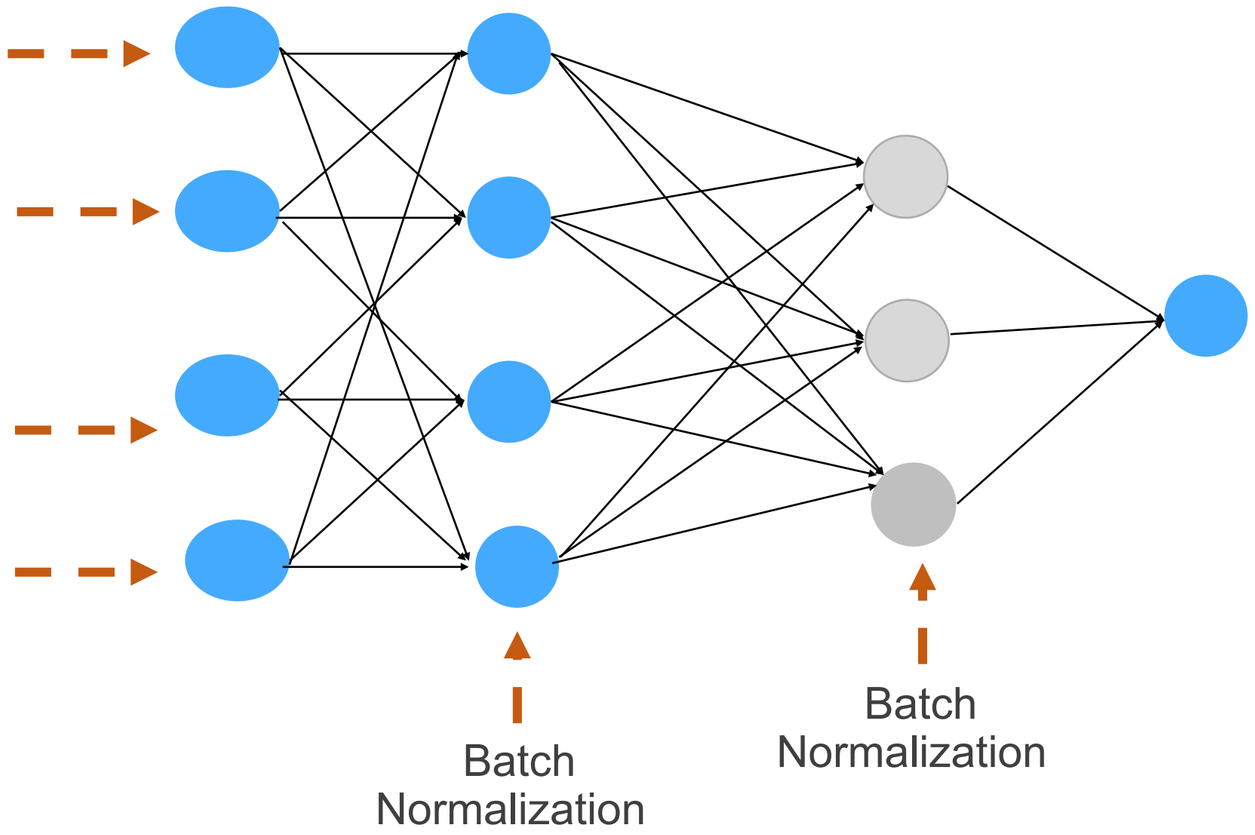
- One more arbitrary parameter b is also added to the output after multiplication.

$$(Z \times g) + b$$

Batch Normalization

- m, s, g, b are all trainable values i.e. the mean, standard deviation along with arbitrary values are optimized in the training process.

- Large weights are no more a concern, as normalization is applied for every layer's output per batch, that is why it is called batch normalization.



Implementing Batch Normalization Using Keras

- Batch normalization is implemented using the deep learning framework, Keras.
- One additional library **BatchNormalization** is imported.
- The batch normalization is initialized after the ReLU activation function.

```
from keras.models import Sequential
from keras.layers import Dense, Activation, BatchNormalization

model = Sequential([
    Dense(16, input_shape=(1,5), activation='relu'),
    Dense(32, activation='relu'),
    BatchNormalization(axis=1),
    Dense(2, activation='softmax')])
```

Batch Normalization with Keras

Problem Statement : Batch normalization allows each layer of a network to learn by itself a little bit more independently of other layers. You are supposed to increase the model performance by implementing batch normalization optimization technique.

Objective: Build a MLP model to demonstrate the effect of Batch Normalization using Keras.

Access: Click the Practice Labs tab on the left panel. Now, click on the START LAB button and wait while the lab prepares itself. Then, click on the LAUNCH LAB button. A full -fledged Jupyter lab opens, which you can use for your hands -on practice and projects.

ASSISTED PRACTICE

Vanishing Gradient

What Is Gradient?

Gradient refers to the derivative of loss with respect to weight.

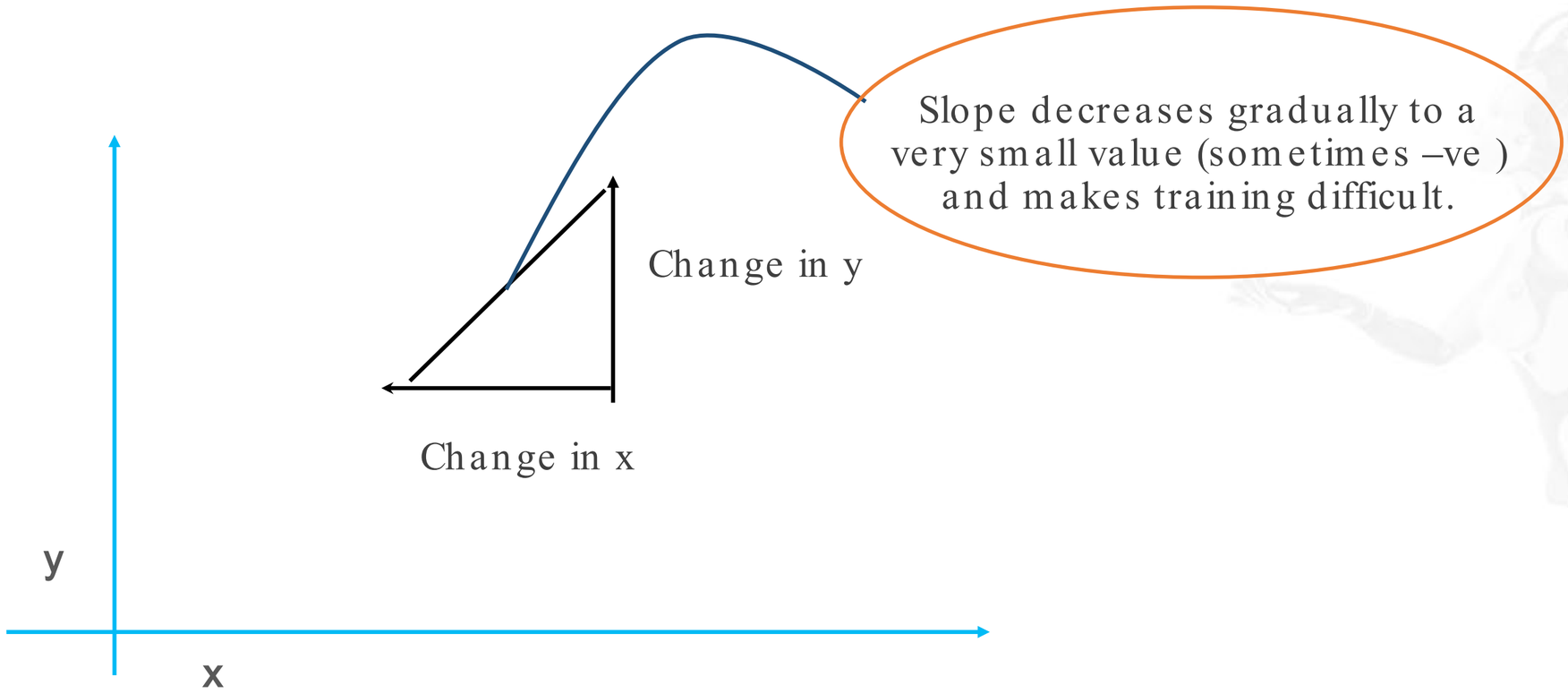
Gradient

It is calculated during the process of back propagation.

It is used to update weights of the neural networks.

What Is Vanishing Gradient?

When gradient becomes very small, subtracting it from the weight doesn't change the previous weight. Therefore, model stops learning. This problem of neural network is called vanishing gradient.



Why Does Vanishing Gradient Occur?

- Vanishing gradient occurs depending on the choice of activation function.
- Activation function like Sigmoid or Tanh crushes its output into a very small numerical range.
- For example, Sigmoid maps the output into 0 to 1 range. As a result, there are large regions of the input space, even a large change in input will only produce a small change in the output. Therefore, gradient becomes small or vanishing gradient occurs.

How to Prevent Vanishing Gradient?

- Vanishing gradient problem can be avoided by using activation function which doesn't have the property of crushing input into very small number range. A popular choice is rectified linear which maps x to max (0, x).
- Switching from CPUs to GPUs with faster compilation time has made standard back propagation method feasible, where the cost of the model is very less.
- Use of residual network can avoid problem of vanishing gradient by grouping many short neural networks together.

Exploding Gradient

What Is Exploding Gradient?

> Exploding gradients are a problem, where large error gradients accumulate and result in very large updates to neural network model weights during training.

How to Fix Exploding Gradients?

- Vanishing gradient problem can be fixed by redesigning neural network to have fewer layers and mini batch sizes.
- Using Long Short -Term Memory (LSTM) networks reduce exploding of gradients.
- Gradient clipping is a method by which gradient size can be limited. It is an effective way to fix exploding gradient.

Hyperparameter Tuning

What Is Parameter?

> Parameters are found while training the model. For example, in K -mean clustering, the number of centroids is a model parameter.

What Is Hyperparameter?

> Hyperparameters are found before the training. A classic example of hyperparameter is the value of K in K-mean clustering which is decided before creating the model.

Hyperparameters of Deep Learning Model

Learning Rate	The most important hyperparameter that helps the model to get an optimized result
Number of Hidden Units	A classic hyperparameter that specifies the representational capacity of a model
Convolutional Kernel Width	It influences the capacity of a model by influencing the number of model parameters in a convolutional neural network
Mini -Batch Size	It affects the training process, training speed, and number of iterations in a deep learning model
Number of Epochs	It is responsible for the optimized weight initialization in a neural network up to some extent

How to Tune the Hyperparameters?

> Hyperparameter tuning is choosing a set of optimal hyperparameters for a learning algorithm.

How to Tune the Hyperparameters?

Choose the parameters wisely

Select the most influential parameters, as it is not possible to tune all of them.

Understand the training process

Know the training process and how exactly it can be influenced.

Selection of Hyperparameters

Hyperparameters can be selected through two approaches:

- Manual
- Automatic

Selection of Hyperparameters

Technically, both of the selection approaches are viable. The real -world hyperparameter optimization is an intersection of the two.

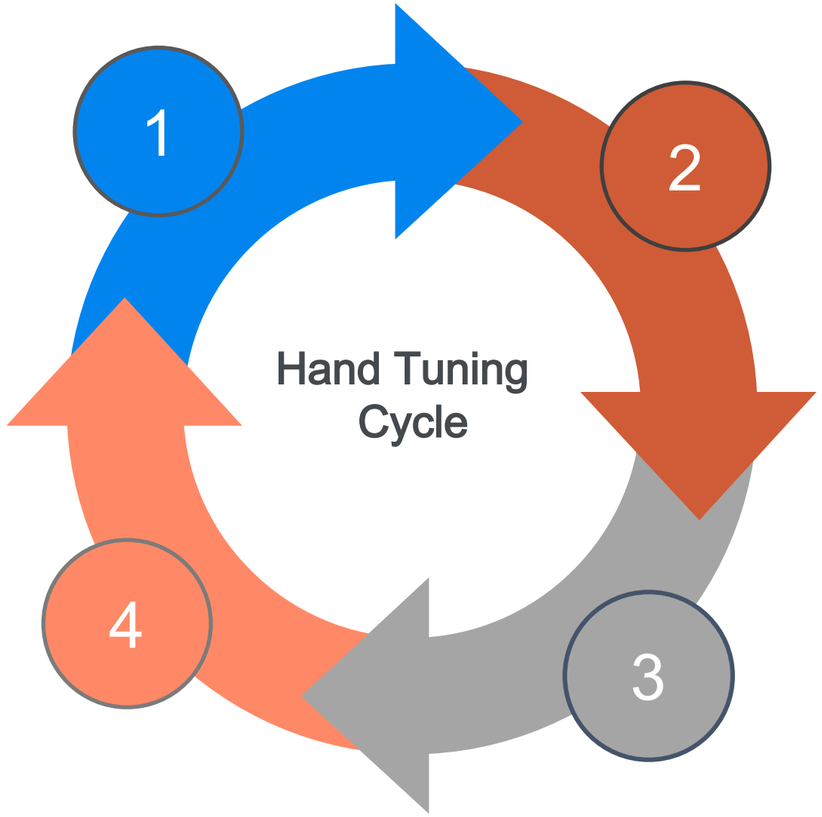
Manual Tuning

> The change made manually in hyperparameter at a time after each evaluation of neural network is called manual tuning.

Manual Tuning Approach

From the following example, it can be concluded that manual tuning is not an efficient way as even five times increase in the neuron resulted only 4% increase in accuracy.

There is one layer of MLP with 50 neurons and the accuracy of the model is 82%.



Now, we have increased the number of neurons to 100 with one additional layer and the resulting accuracy is 84%.

Similarly, the neurons are increased to 250 with five layers, but the accuracy is increased to 86% only.

The number of layers are increased to three and the resulting accuracy is 85%.

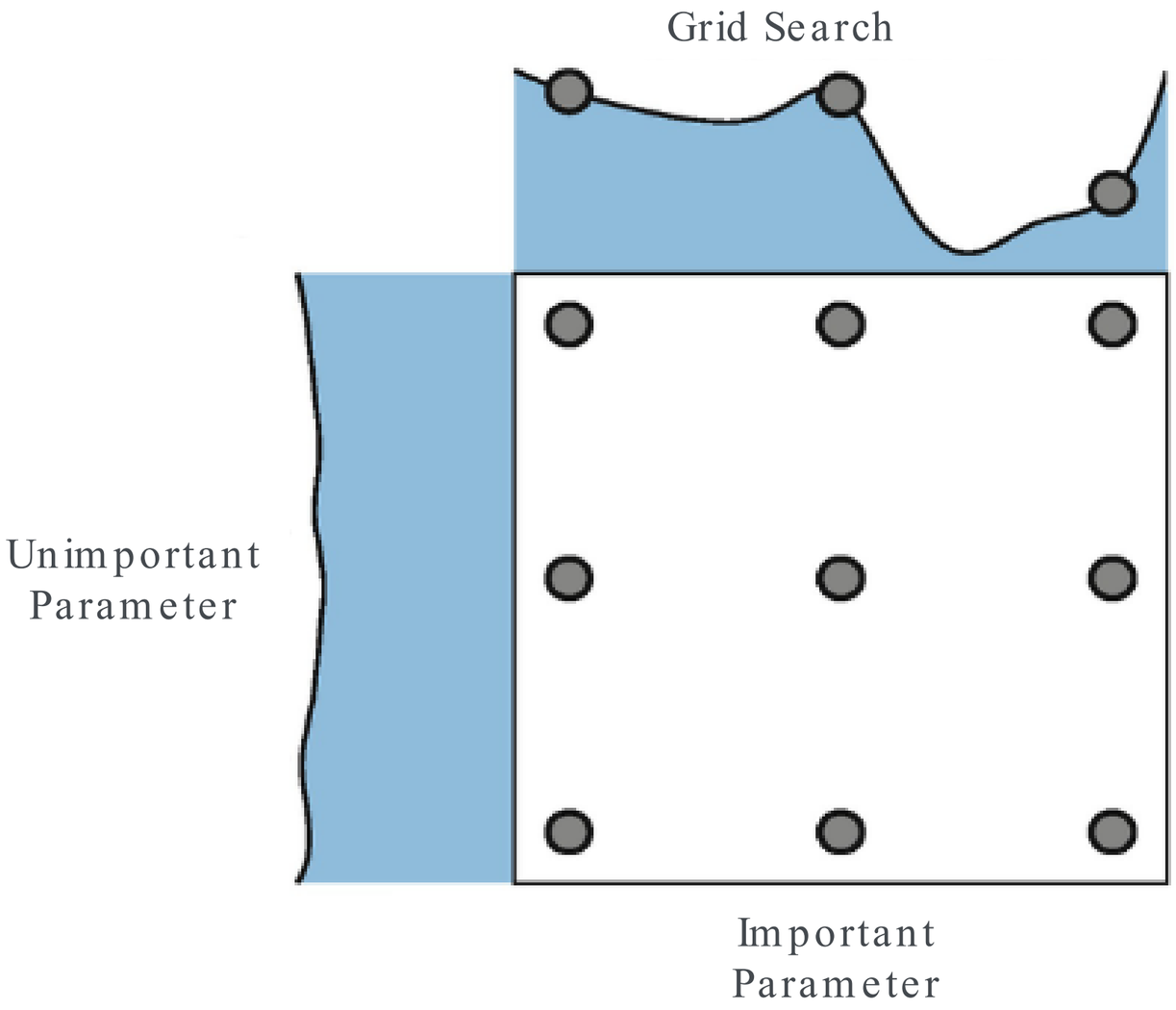
Automatic Hyperparameter Tuning

Automatic selection approach is preferred over the manual approach as the latter is a very rigorous method. Automatic approach is the process of tuning the hyperparameters with the help of algorithms. They are as follows:

- Grid Search
- Random Search
- Gradient -Based Tuning
- Evolutionary Optimization
- Bayesian Optimization

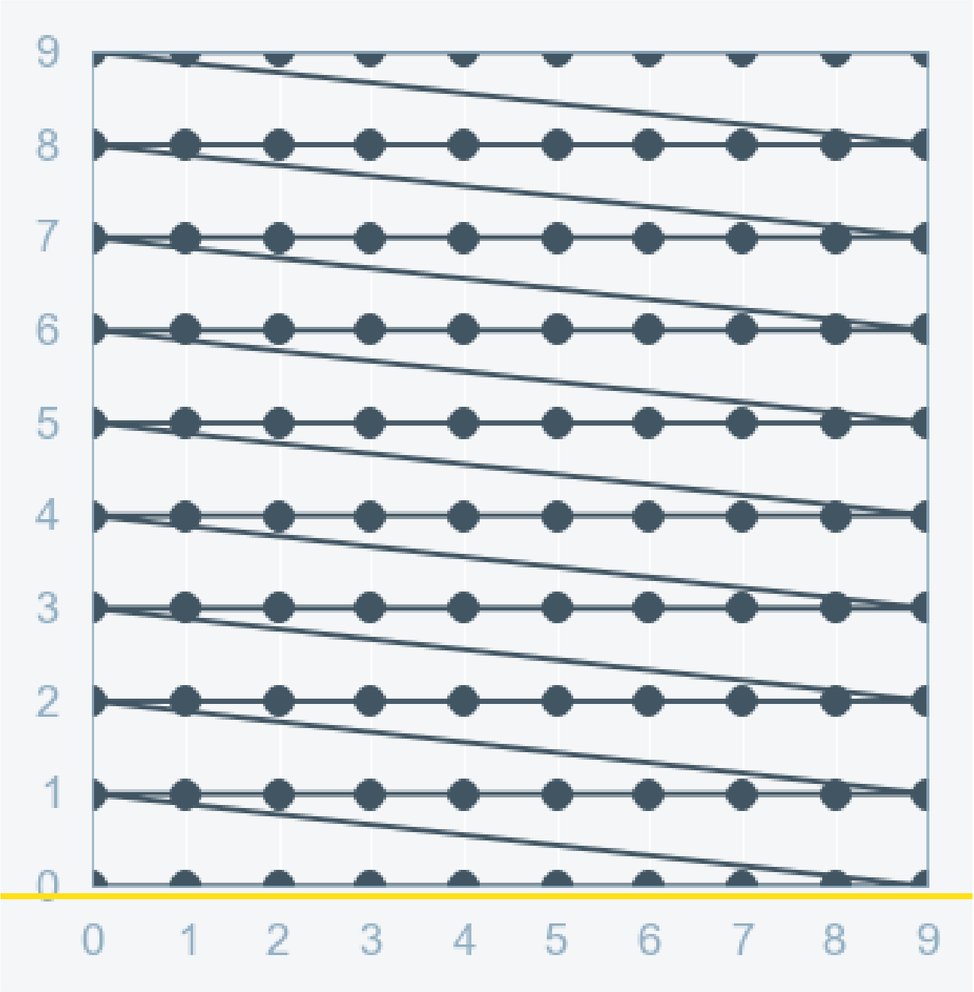
Grid Search

Iterating over given hyperparameters using cross validation is called **Grid Search** .



Grid Search

- Pattern is similar to a grid
- Values are put in the form of a matrix
- Each set of parameters is taken into models and accuracy is noted
- Models with all combinations are evaluated, whichever gives the highest accuracy is declared the best

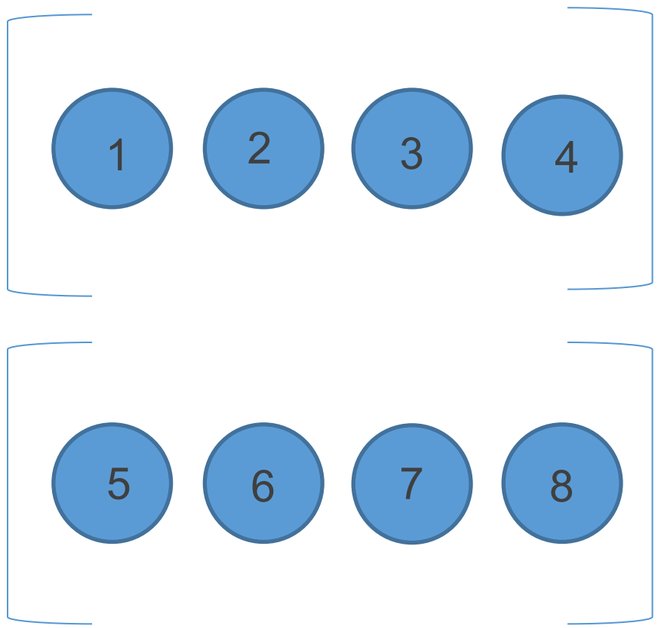


Types of Optimizers

- Grid Search
- Random Search
- Gradient -Based Tuning
- Evolutionary Optimization
- Bayesian Optimization

Grid Search

Grid search takes different values of hyperparameter separately.

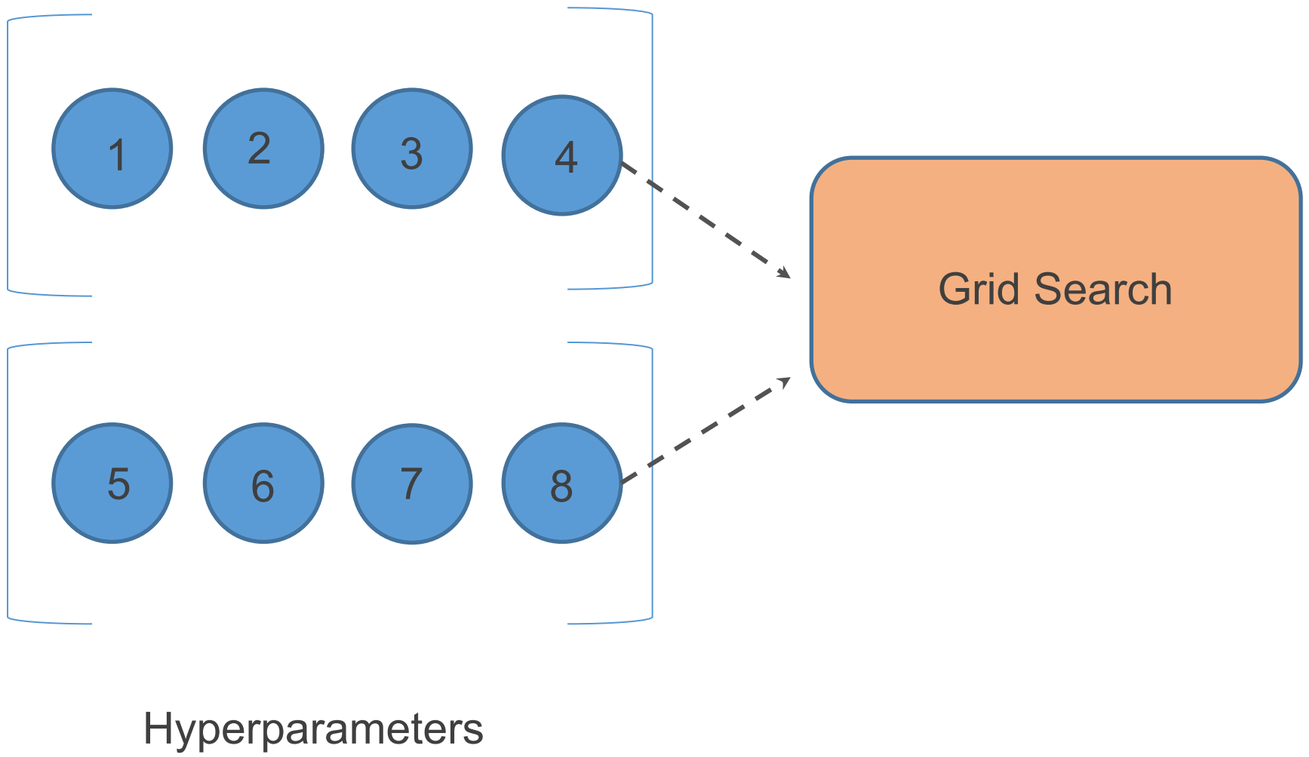


Hyperparameters

Grid Search

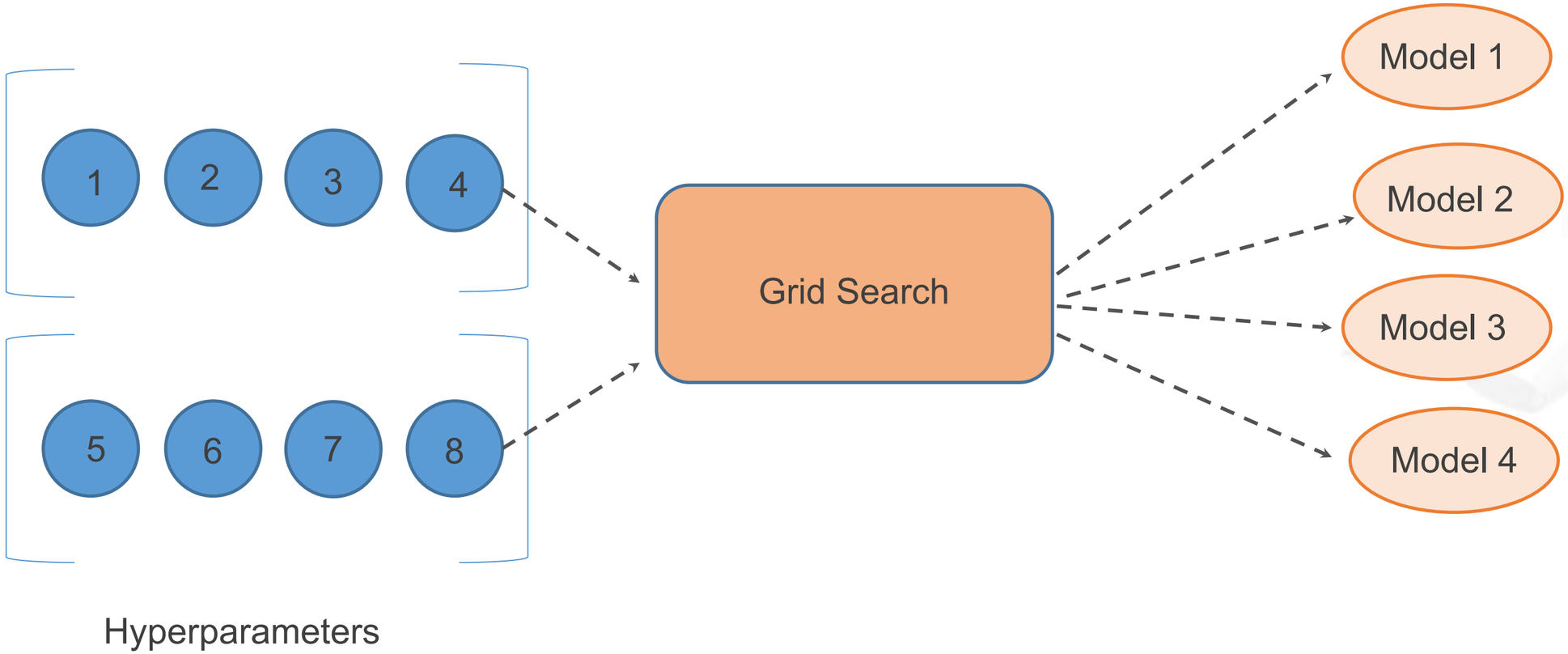
Grid Search

Eight different hyperparameters are given, grid search takes different hyperparameter values.



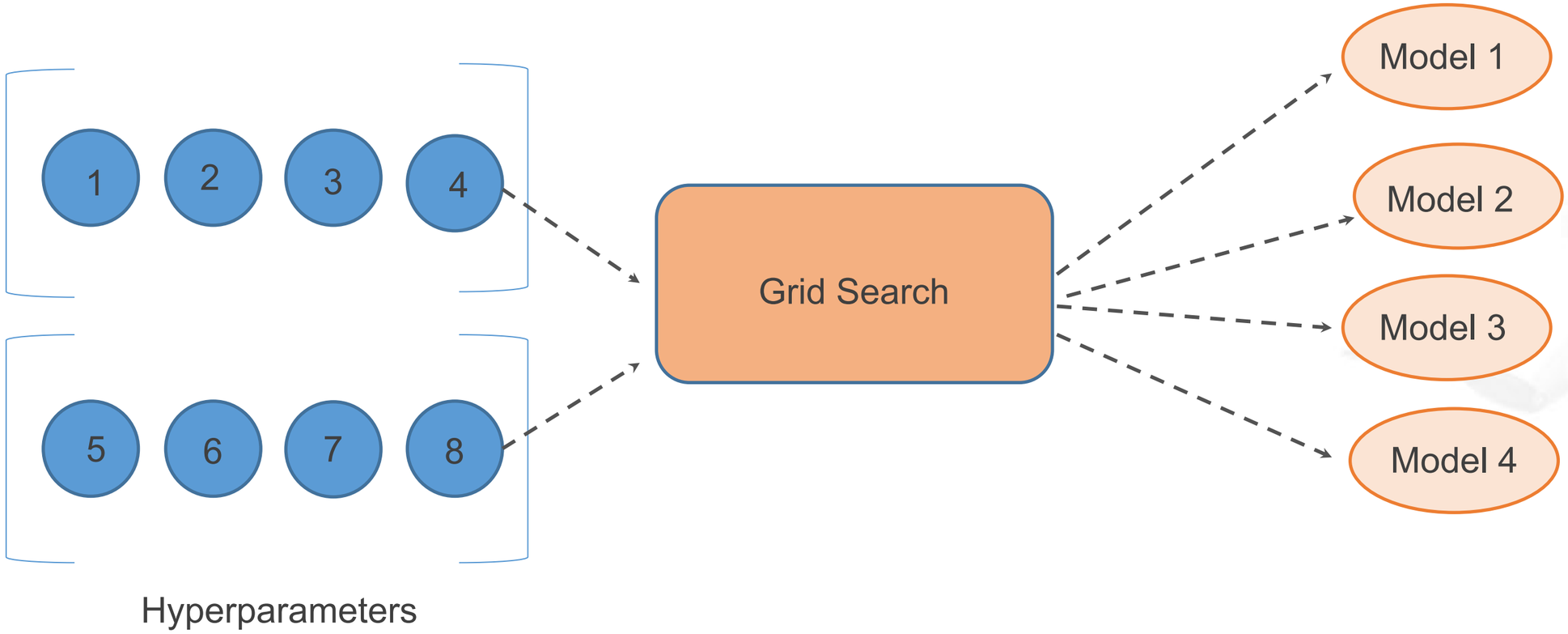
Grid Search

Four models are created with the available hyperparameters.



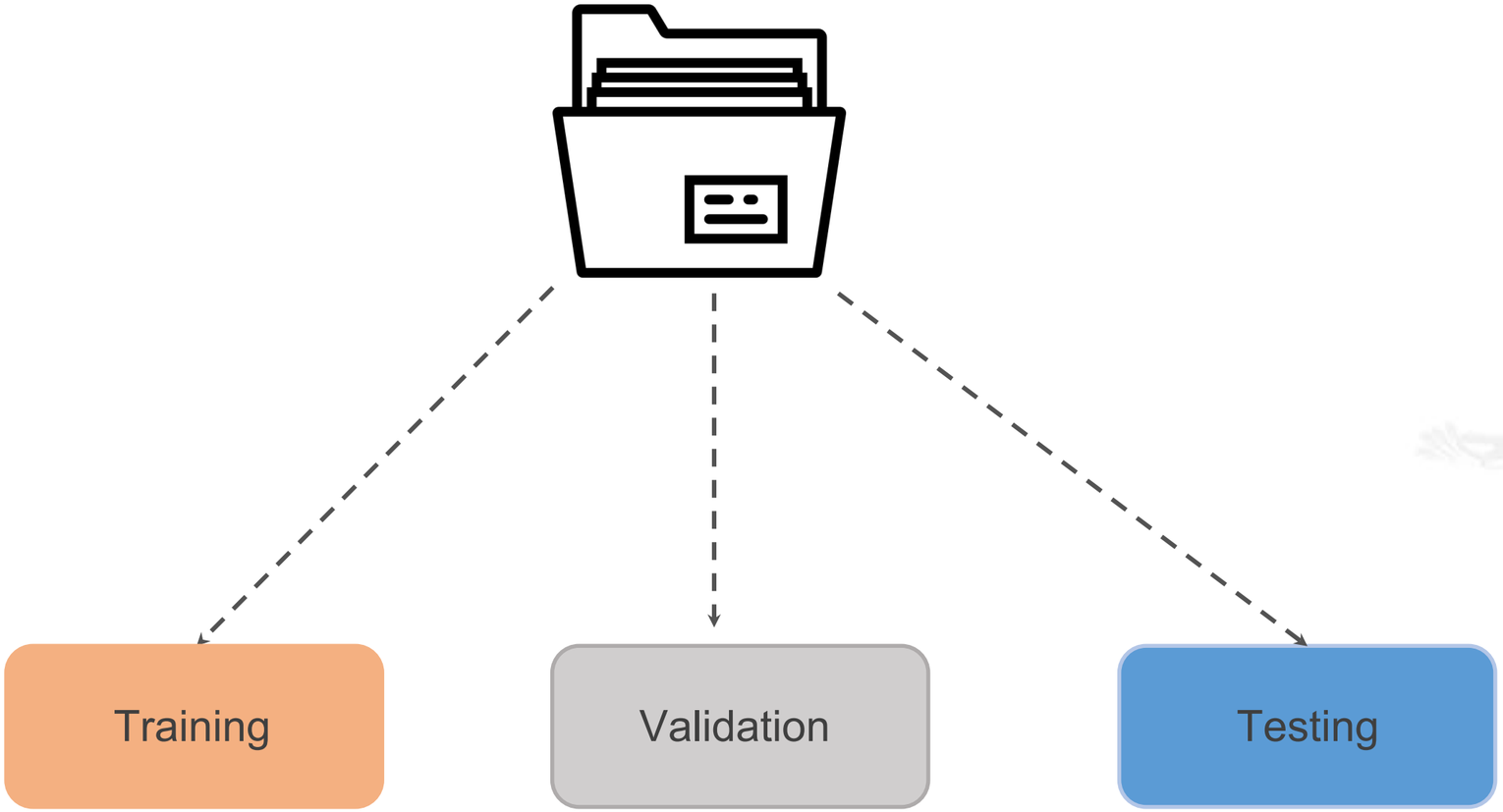
Grid Search

The model with the lowest error will be selected as the most efficient model and the hyperparameters used in the model are finalized.



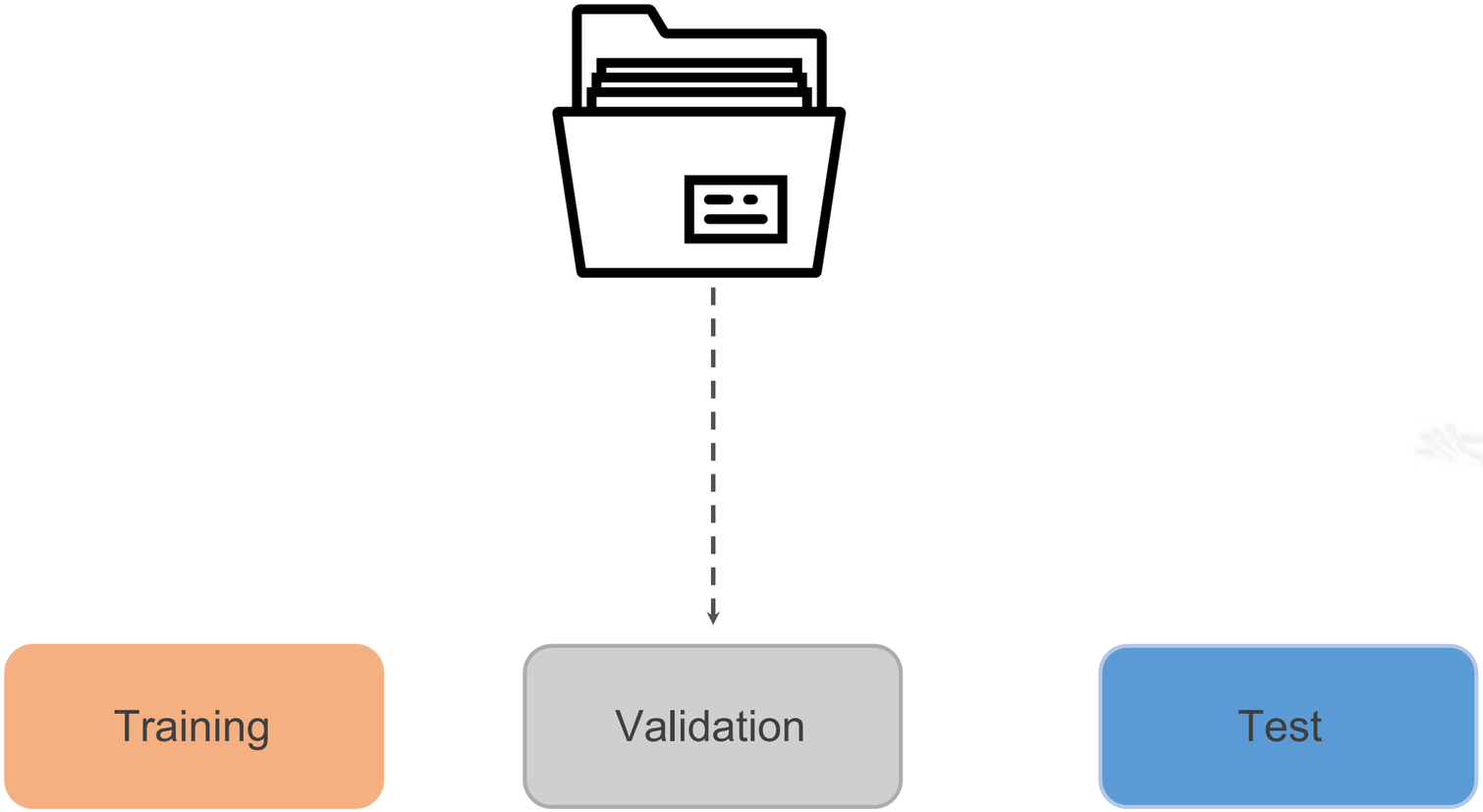
Grid Search

To select the hyperparameters, the given data is divided into three different parts.



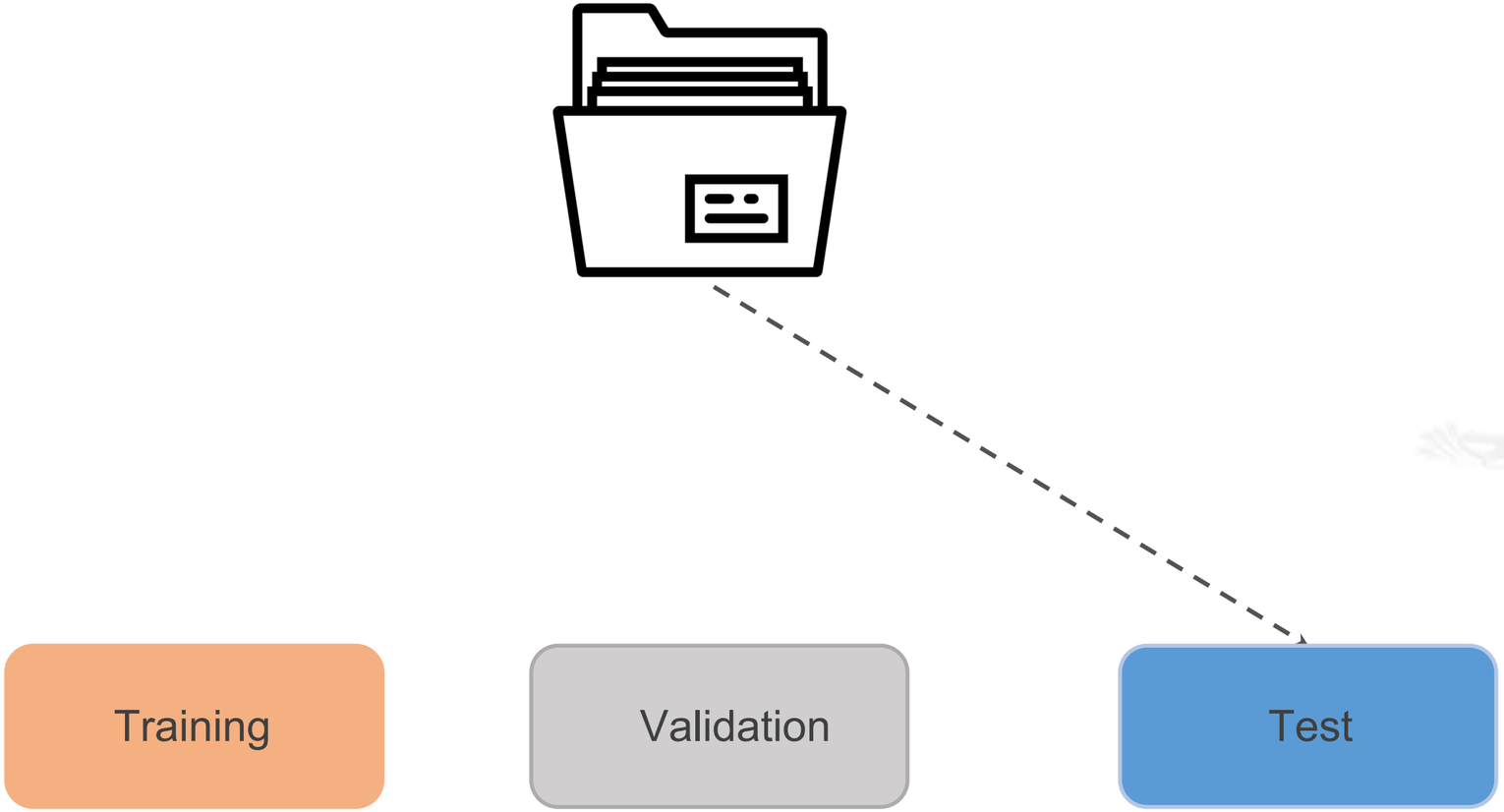
Grid Search

Select the hyperparameter that minimizes the error in the validation set.



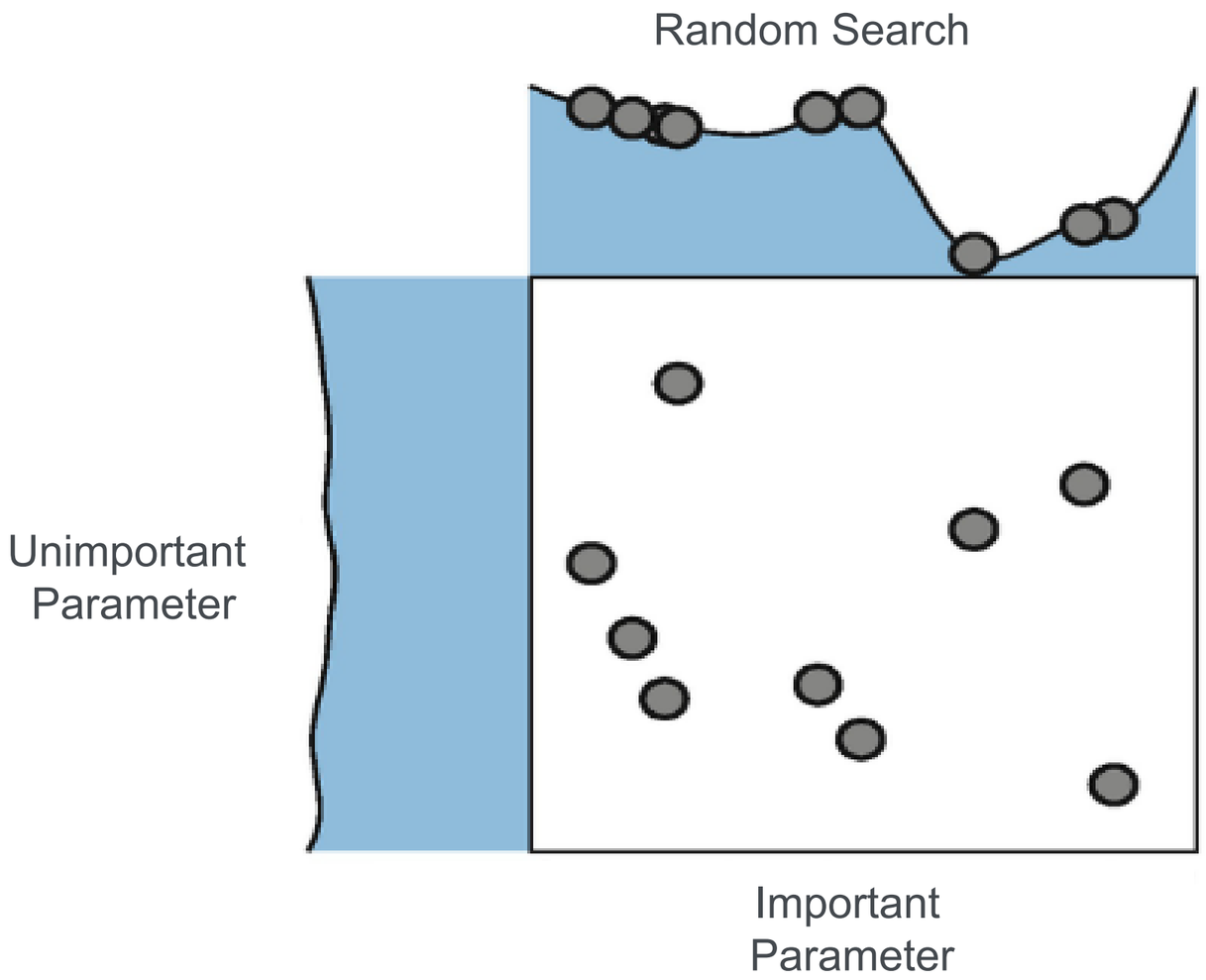
Grid Search

Finally, the model is tested with the selected hyperparameters to assess its performance.



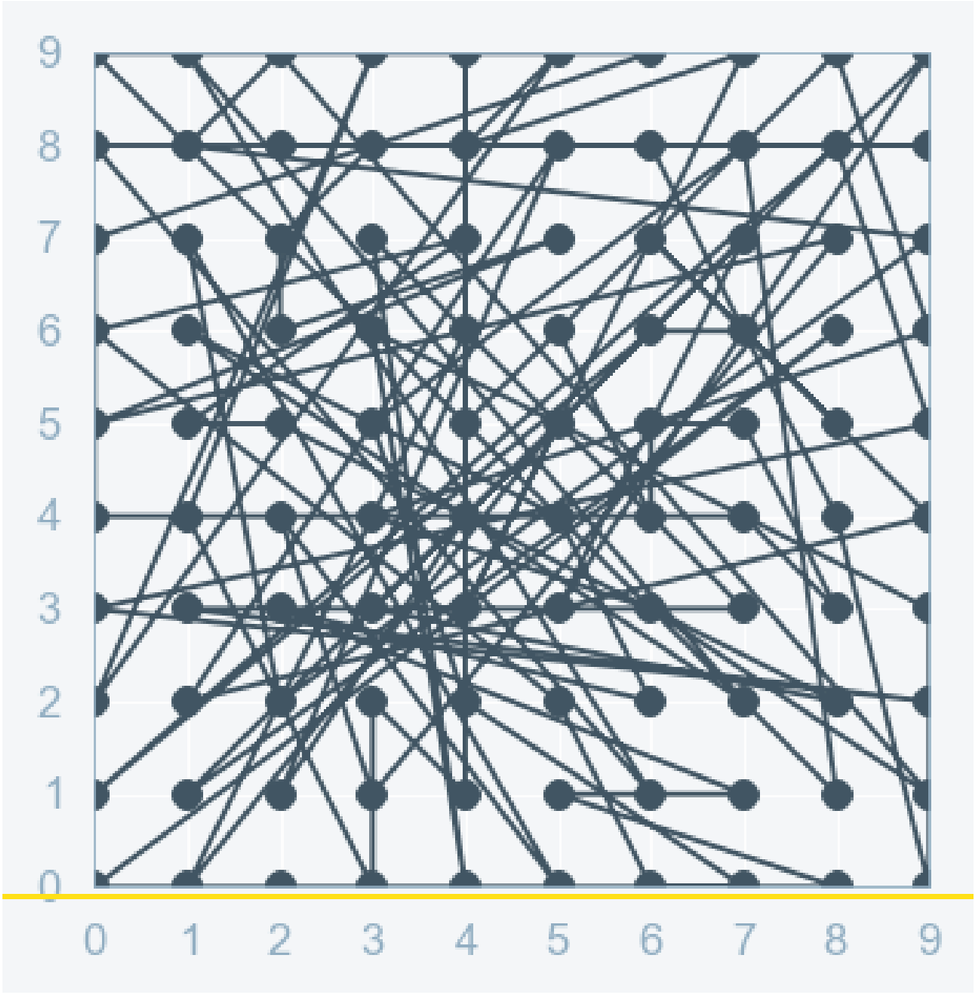
Random Search

Random search is an optimization method used on functions that are not differentiable or continuous.



Random Search

- Produces random value at each instance
- Covers every combination of instances
- Considers random combination of parameters at every iteration
- Finds the optimized parameter through performance of models



Random Search

Saves time

More efficient than manual or grid search

Has a drawback of producing high variance is during

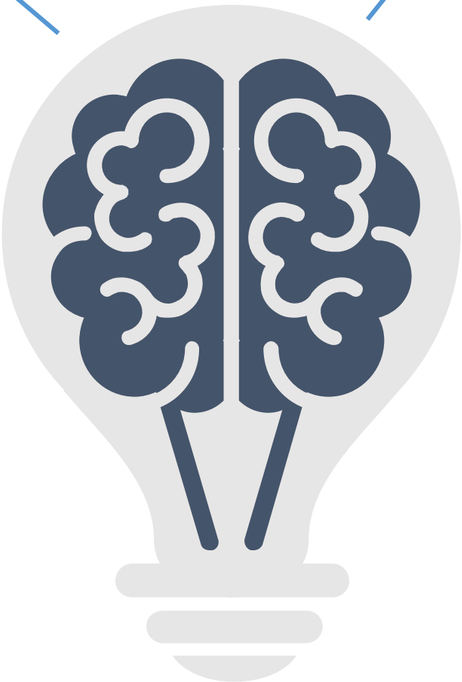
Gradient -Based Tuning

Gradient -based tuning is used for algorithms, where it is possible to compute the hyperparameter with respect to the gradient and optimization of the hyperparameter is done by the gradient descent.

Evolutionary Optimization

Uses evolutionary algorithm to find optimal hyperparameters

Used in black box functions with noises for global optimization



Evolutionary Optimization

Steps for the evolutionary optimization:

1. Create multiple number of solutions
2. Determine the hyperparameter tuple and obtain their fitness function
3. Rank hyperparameter tuple according to their relative fitness
4. Replace the worst performing hyperparameter with new pairs
5. Repeat steps 2 to 4 until the algorithm gives a constant good outcome

Bayesian Optimization

- Uses machine learning framework to predict optimal hyperparameters
- Finds optimal hyperparameters from the result of previously built models with different hyperparameter configuration through the Gaussian process
- Has the inherent property to study the trend in given data set which is not possible for a human

Interpretability

What Is Interpretability?

> Interpretability is the degree of human's ability to predict the model's result consistently.

Importance of Interpretability

Fairness	To ensure that predictions are unbiased
Privacy	To ensure that sensitive information in the data is well -protected
Reliability	To ensure that small changes in the data do not lead to big changes in the prediction
Causality	To check that all the relationships picked up are causal
Trust	To make it easily trustable for humans as it explains its decisions unlike the machine

When Is Interpretability Not Needed?

For an insignificant model

For a well-studied and researched problem

For scenarios where people or the program might manipulate the model

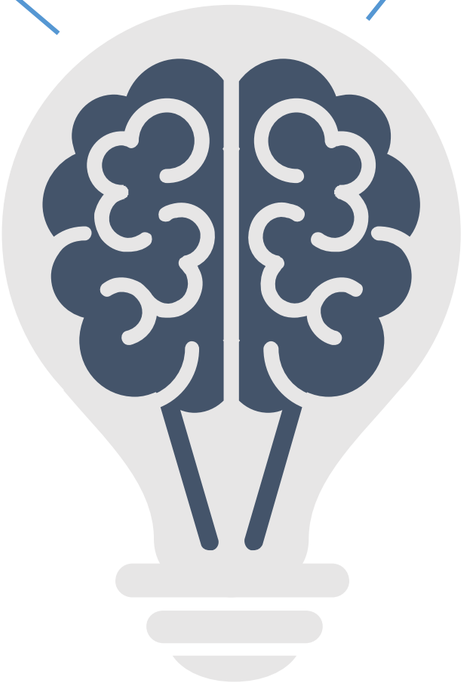
Classification of Interpretability Methods

- Intrinsic or Post Hoc
- Model -Specific or Model -Agnostic

Intrinsic or Post Hoc

Achieves interpretability by simplifying the machine learning model and analyzes the method after the training

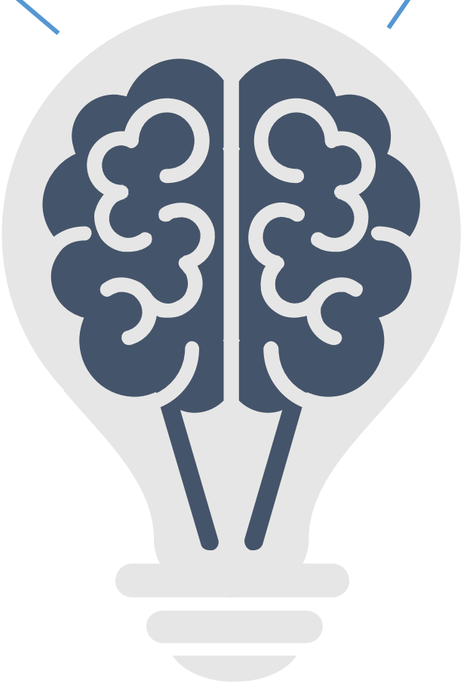
Refers to models that are considered interpretable due to their simple structure



Model -Specific or Model -Agnostic

Can be used on any model and are applied after the model has been trained

Works by analyzing feature input and output pairs



Scope of Interpretability

- Algorithm Transparency
- Global, Holistic Model Interpretability
- Global Model Interpretability on a Modular Level
- Local Interpretability for a Single Prediction
- Local Interpretability for a Group of Predictions

Algorithm Transparency

Deals with how the algorithm learns a model and the types of relationships from the given data

Requires the knowledge of the algorithm and not of the data or trained model

Global, Holistic Model Interpretability

Helps to understand the distribution of target outcome based on the features

Requires the output of a trained model, knowledge of the algorithm used in the model and the given data

Deals with the understanding of how the model makes decision with a holistic view of features

Global Model Interpretability on a Modular Level

Can be used when there is difficulty in achieving global model interpretability

Can be understood through the effects of parameters and features on the predictions on an average

Local Interpretability for a Single Prediction

Examines a single instance of a model and its prediction for the specific input

Accuracy of local interpretability is more than prediction of global interpretability

Local Interpretability

Applies global methods on a group of instances considering the group as a complete dataset

Uses individual explanation methods on each instance and aggregates the entire group of instances

Evaluation of Interpretability

Doshi-Velez and Kim (2017) propose three main levels for the evaluation of interpretability:

- Application-Level Evaluation (Real Task)
- Human-Level Evaluation (Simple Task)
- Function-Level Evaluation (Proxy Task)

Application -Level Evaluation (Real Task)

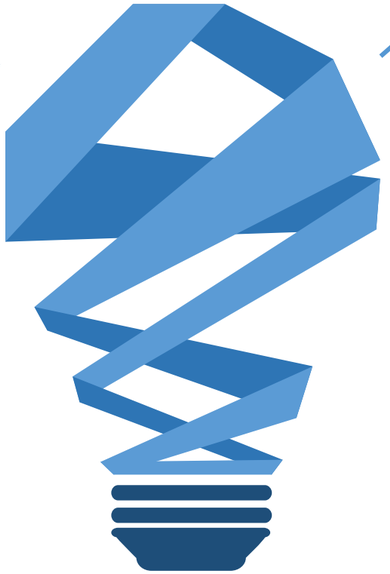
Is assessment of the outcome of the interpretability by domain experts

Requires a good experimental setup and an understanding of quality assessment

Human -Level Evaluation (Simple Task)

Is a simplified version of application -level evaluation

Is an inexpensive method as the evaluation does not require technical expertise



Function -Level Evaluation (Proxy Task)

Does not require human expertise

Is generally performed after human -level evaluation which leads to enhanced results

Explanation in Interpretability

- Relates different parameters of a dataset to the predicted model in an understandable way
- Is generated by algorithms which work as explanation methods

Properties of Explanation Methods

These properties measure the effectiveness of the explanation method:

Expressive Power	A language structure generated from the explanation method
Translucency	Describes how the model relies on its parameters
Probability	Describes the explanation method suitable for the range of models
Algorithmic Complexity	To check that all the relationships picked up are causal

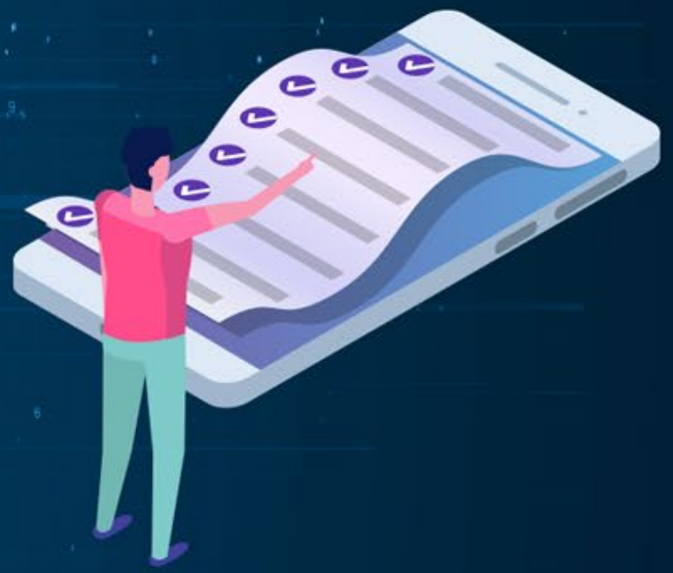
Properties of Individual Explanations

Accuracy	Assesses the accuracy of prediction of the unseen data
Fidelity	Checks how well the explanation approximates the prediction
Consistency	Helps to differentiate among models trained on same data set with same procedure
Stability	Highlights the similar parameters in a fixed model
Comprehensibility	Helps in making the explanation understandable

Key Takeaways

Now, you are able to:

- Explain algorithms of optimization
- Perform batch normalization
- Describe hyperparameter tuning and its significance
- Explain interpretability in deep learning



Knowledge Check

Knowledge Check 1	Which of the following optimization algorithms use moving average?

a. Gradient Descent

b. Stochastic Gradient Descent

c. Stochastic Gradient Descent with Mini Batch

d. Stochastic Gradient Descent with Momentum

Knowledge Check 1	Which of the following optimization algorithms use moving average?

a. Gradient Descent

b. Stochastic Gradient Descent

c. Stochastic Gradient Descent with Mini Batch

d. Stochastic Gradient Descent with Momentum

The correct answer is **b**

Stochastic gradient descent with momentum optimization algorithm uses moving average.

Knowledge Check 2	Which of the following optimization functions update learning rate along with the weights?

a. Gradient Descent

b. Stochastic Gradient Descent

c. Stochastic Gradient Descent with Mini Batch

d. AdaGrad

Knowledge Check 2	Which of the following optimization functions update learning rate along with the weights?

a. Gradient Descent

b. Stochastic Gradient Descent

c. Stochastic Gradient Descent with Mini Batch

d. AdaGrad

The correct answer is **d**

AdaGrad optimization algorithm updates learning rate along with optimization.

Knowledge Check 3	Which of the following are the most widespread optimizers in deep learning?

a. Adam

b. AdaGrad

c. AdaDelta

d. RMSProp

Knowledge Check 3	Which of the following are the most widespread optimizers in deep learning?

a. Adam

b. AdaGrad

c. AdaDelta

d. RMSProp

The correct answer is **a**

At present, Adam is the most widespread optimizer in deep learning.

Knowledge Check 4	“When large error gradients get accumulated, it results in sudden change in weight of the neural network.” What is this called?

a. Gradient Exploding

b. Gradient Cliffing

c. Gradient Converging

d. None of the above

Knowledge Check 4	"When large error gradients get accumulated, it results in sudden change in weight of the neural network." What is this called?

a. Gradient Exploding

b. Gradient Cliffing

c. Gradient Converging

d. None of the above

The correct answer is **a**

Gradient exploding is the sudden change in weight of the neural network when large error gradients get accumulated.

Hyperparameter Tuning with MNIST Data Set

Problem Scenario: A classification model has been made using inbuilt optimizers of deep learning frameworks but the model is not giving the desired output. You are facing the same problem and you have to perform the tuning with MNIST data set.

Objective: Build a single -layer dense neural network perform hyperparameter tuning using random search with keras -tuner.

Access: Click the Practice Labs tab on the left panel. Now, click on the START LAB button and wait while the lab prepares itself. Then, click on the LAUNCH LAB button. A full -fledged Jupyter lab opens, which you can use for your hands -on practice and projects.

LESSON-END PROJECT

Thank You