Deep Learning

Model Optimization and Performance Improvement

Learning Objectives

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

- Optimize a deep learning model to get the most accurate results
- Evaluate various optimization algorithms, such as SGD, Momentum, AdaGrad, and Adadelta
- Implement optimization algorithms
- Apply batch normalization

Learning Objectives

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

- Apply regularization techniques such as dropout and early stopping
- Analyze the issues of vanishing and exploding gradients
- Distinguish between interpretability and explainability

Business Scenario

XYZ Corp. is an online retail company that has been experiencing high cart abandonment rates on its website. It wants to optimize its deep learning-based product recommendation system to improve customer retention rates.

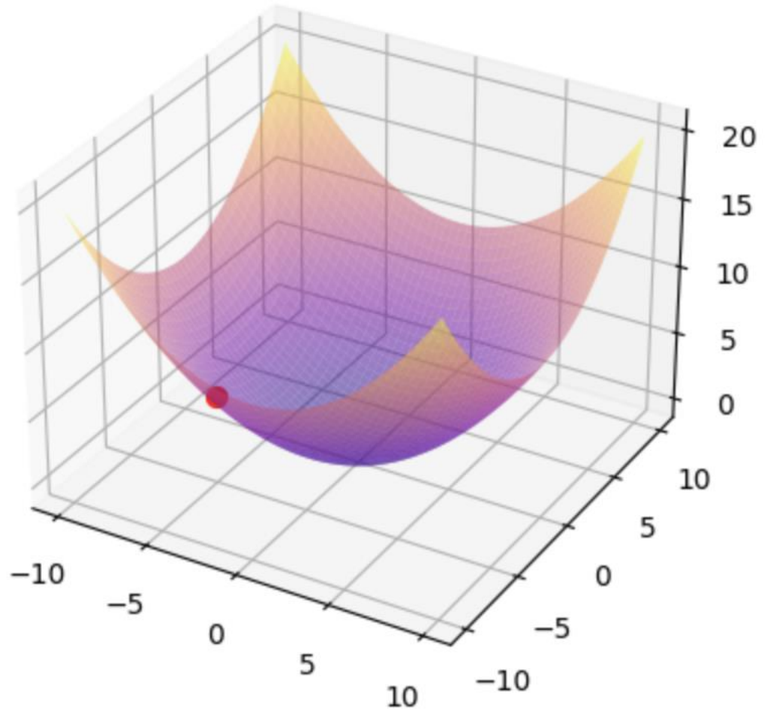
XYZ partners with a machine learning solutions provider that recommends utilizing the Adadelta optimization algorithm to improve the convergence speed of the neural network model. The provider also suggests implementing batch normalization to expedite the training process and mitigate overfitting. XYZ incorporates dropout and early stopping techniques to prevent overfitting and enhance the model's generalization capability.

Upon training and deploying the model, the company observes a substantial reduction in cart abandonment rates, resulting in increased sales revenue and improved customer satisfaction.

Introduction to Optimization Algorithms

What Is Optimizer?

Algorithms or methods that reduce losses by changing neural network attributes such as weights and learning rates.



Optimizers establish a connection between the loss function and model parameters, modifying the model based on the loss function's output.

They manipulate the neural network weights, molding and refining the model to achieve the highest possible accuracy.

Example of an Optimizer

The loss function serves as a roadmap for the optimizer to indicate if it has taken the correct or wrong path.

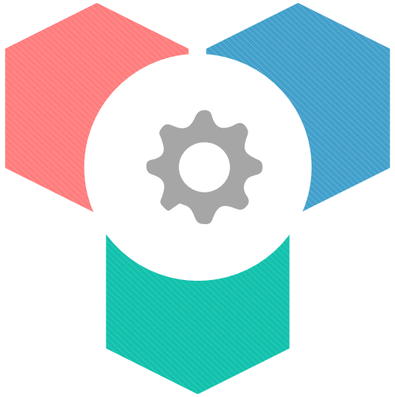
Consider a hiker attempting to descend a mountain, with a blindfold.

It's impossible for the hiker to determine which way to travel, but they can tell if they are going down (making progress) or up (losing progress).

If the hiker continues downhill, they will eventually reach the bottom.

What Is Optimization?

The process of minimizing or maximizing any mathematical expression involves finding the values that result in the lowest or highest outcome. This makes the predictions as accurate and optimized as possible.



The parameters of a model are adjusted and changed during the training phase to minimize the loss function.

Example: Optimization

Building bridges involves balancing weight capacity, cost, safety, and material use. This equilibrium is referred to as optimization.

- Aim for maximum load capacity given design and usage
- Minimize material usage without compromising integrity
- Prioritize cost-effectiveness
- Uphold stringent safety standards

Example: Optimization

The optimization of warehouse location and placement in logistics aims to minimize shipping costs and improve efficiency.

- Choose locations near transport routes and customers
- Consider supplier proximity for efficient inbound logistics
- Plan warehouse layout for optimal space and goods movement

Optimization Algorithms

In order to minimize the loss function, optimization algorithms are used to iteratively change the model's parameters during training. In deep learning, optimization algorithms are used to optimize cost function **J.**

The equation is represented as follows:

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

Optimization Algorithms: Equation

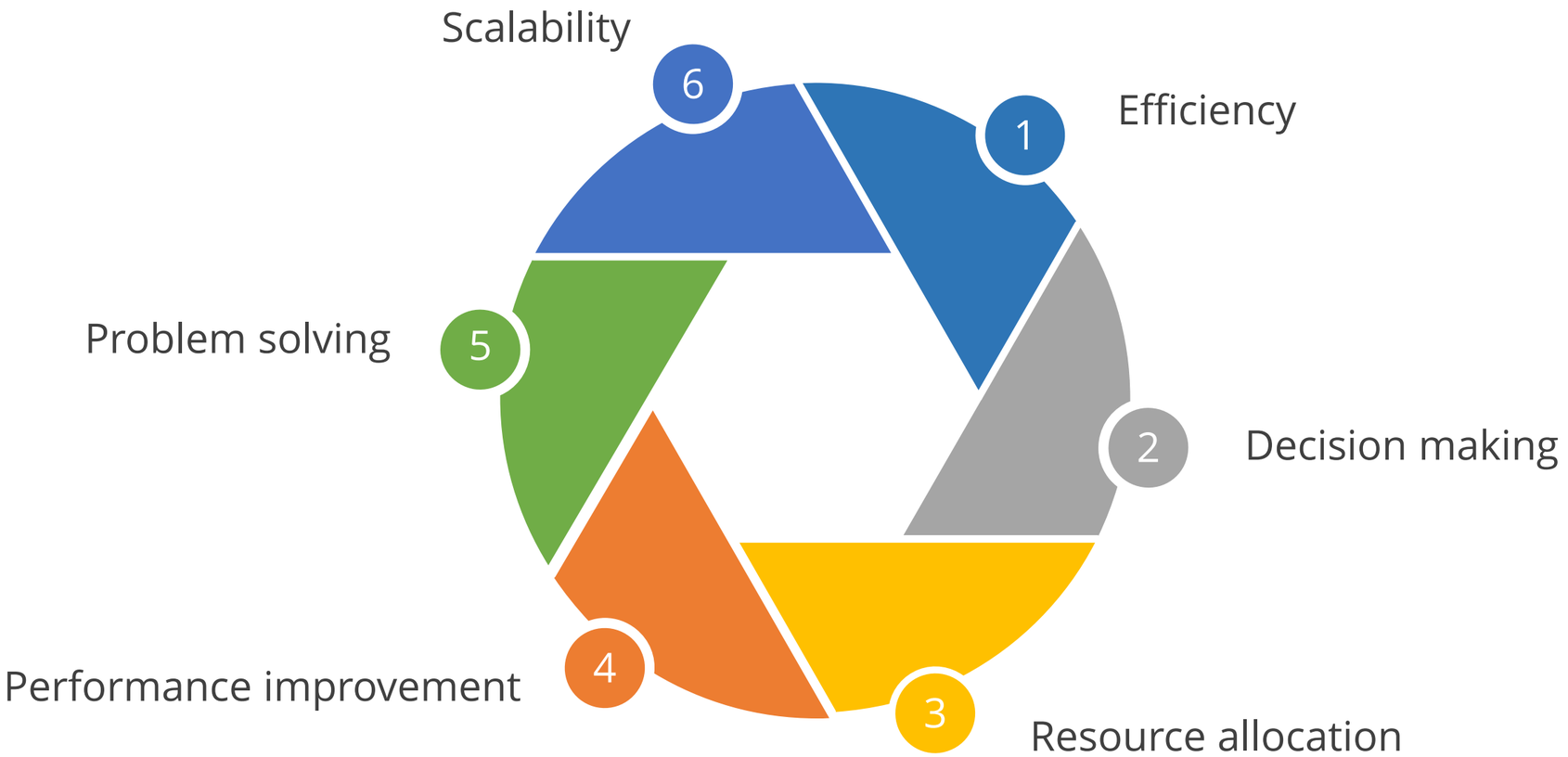
Equation represents it as follows:

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

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

Importance of Optimization Algorithms

The following list of factors illustrates the significance of optimization algorithms:



Types of Optimizers

The types of optimizers are:

GD (Gradient Descent)	Finding the global minima for a convex optimization problem
SGD (Stochastic Gradient Descent)	A faster learning rate is required
Momentum	There are chances of having local minima
AdaGrad	Dealing with sparse data (NLP or computer vision)

Types of Optimizers

The types of optimizers are:

AdaDelta	An exponentially weighted average of gradients
RMSprop	Faster convergence is required
Adam	The minimum oscillation in global minima

Introduction to Stochastic Gradient Descent

Discussion

Discussion: Stochastic Gradient Descent

Duration: 10 minutes

- What is Stochastic Gradient Descent?
- What are the advantages of Stochastic Gradient Descent?

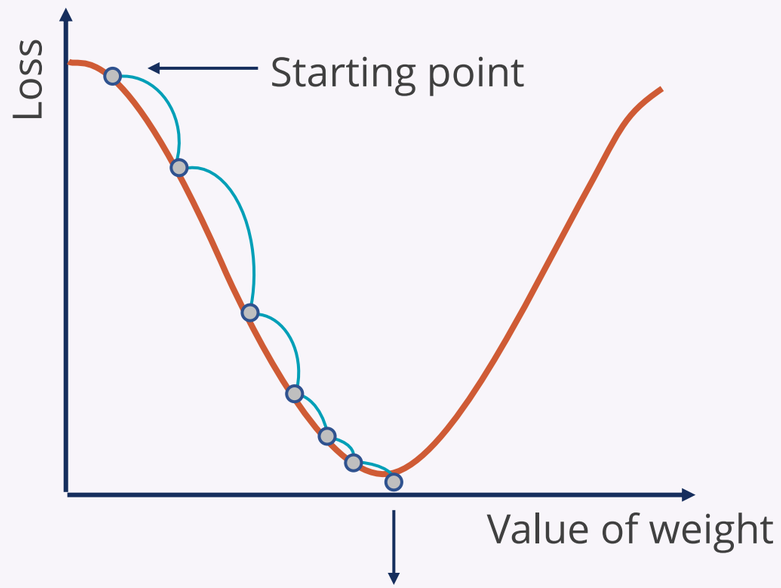
What Is Gradient Descent?

One of the primary concepts in deep learning is optimizing a cost function.

Gradient descent is a typical optimization algorithm and the foundation of training deep learning models.

Gradient Descent

This technique involves calculating how much the model's error changes with changes to its parameters.



The point of convergence is where the cost function is at its minimum.

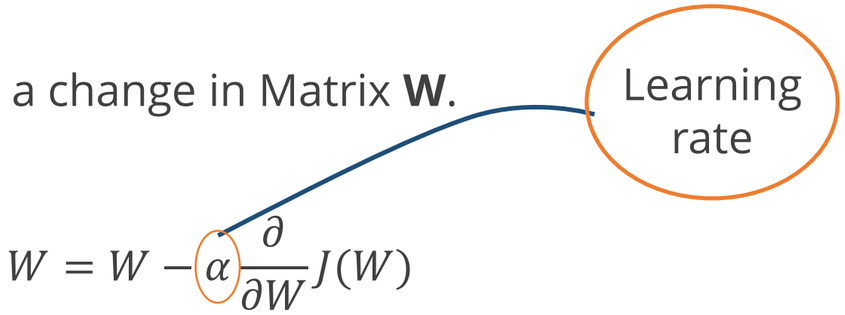
It modifies those parameters in the way that will most drastically decrease the error.

The optimal weights for a model are not known before training but can be found through trial and error using the loss function.

Gradient Descent

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

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



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

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

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

Gradient Descent

In gradient descent, the parameter W is updated iteratively to minimize the loss function $J(W)$.

Update formula:

1. If the slope of the partial derivative with respect to W is positive:
$W = W - \alpha \, \partial/\partial W \, J(W)$

2. If the slope of the partial derivative with respect to W is negative:
$W = W - \alpha \, \partial/\partial W \, J(W)$

Here, α represents the learning rate, a positive scalar controlling the step size.

In both cases, the goal is to adjust W in a direction that reduces $J(W)$ and approaches the minimum loss function during the training process.

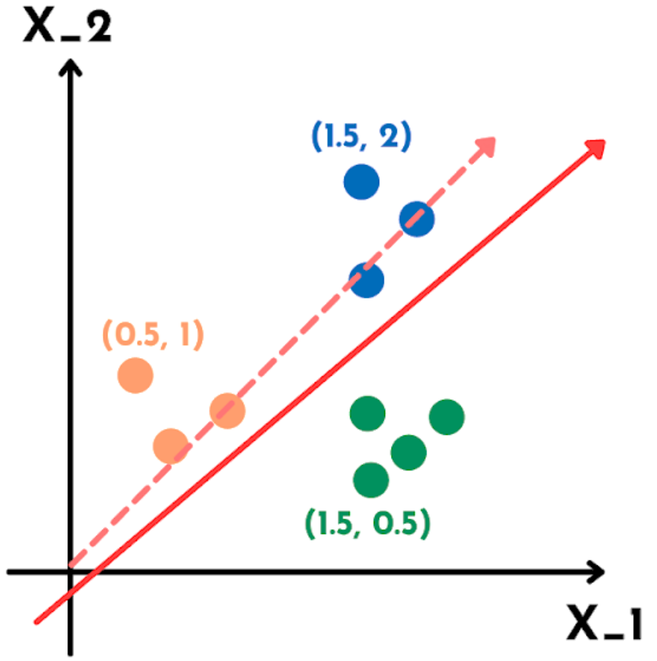
Gradient Descent

The dataset is shuffled for each iteration to avoid biasing the descent and improve the randomness of the mini-batch selection process.

Steps to be followed:

Given equation: $X_2 = m(X_1) + b$
let, m = 1 and b = 0

1. Compute the gradients:
$d(loss)/d(m) = -2°[X_2 - m(X_1) - b]*X_1 = M$
$d(loss)/d(b) = -2°[X_2 - m(X-1) - b] = B$



Gradient Descent

2. Update the parameters:
m" = m - L(M) and
b" = b - L(B)
where L = learning rate

3. Repeat the above steps for several iterations, and adjust the parameters based on the gradients and the learning rate.

After several iterations, the equation can be represented as:
X_2 = mf(X_1) + bf
where mf and bf are the optimal slopes and intercepts obtained using GD.

Stochastic Gradient Descent

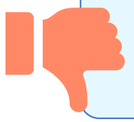
Stochastic gradient descent (SGD) updates parameters by evaluating the loss and gradient on mini-batches of data, enabling efficient iterative optimization in deep learning.

SGD, a variant of the gradient descent algorithm, is used to accelerate model learning in deep learning.

Stochastic Gradient Descent

The term **stochastic** alludes to the random nature of the algorithm.

SGD randomly picks one sample from a dataset to approximate the loss and the gradient and updates the parameters.

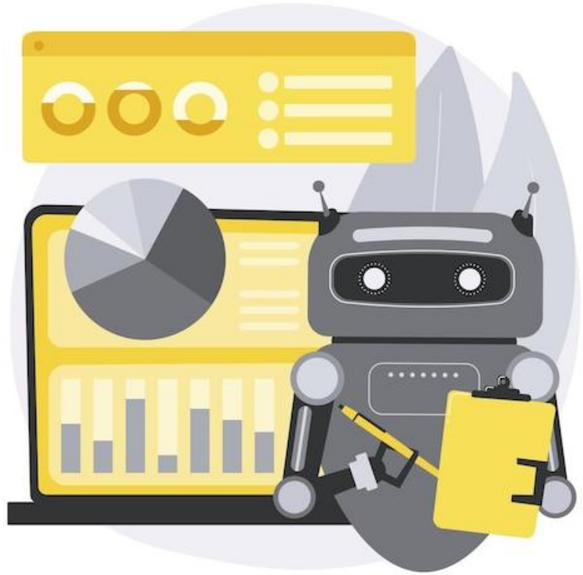


The resulting **descent** can be noisy and might take more iterations.

It is computationally less expensive than vanilla gradient descent.

Stochastic Gradient Descent

Optimal convergence in SGD can be achieved by systematically decreasing the learning rate over time, allowing the algorithm to settle closer to the global minimum.

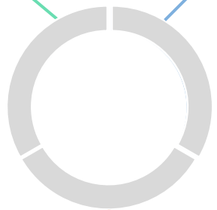


A common practice is to set the learning rate value to 0.01.

Key Features of Stochastic Gradient Descent

Deep learning optimization uses mini-batches of data to find optimized weights.

SGD shuffles the data points within each mini-batch for improved generalization.



The optimizer aims to iteratively update the weights to find the optimal solution, considering factors like mini-batch randomness and noise in the data.

Advantages of Stochastic Gradient Descent

It offers a range of advantages, including:

The descent need not be redone when a new datapoint is introduced.

The algorithm can continue from the latest equation and update the parameters to suit the new data.

While SGD is highly random and liable to noise, it is a highly efficient algorithm and can provide optimal solutions under the right conditions.

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

It is a combination of vanilla GD and SGD, which distribute the training data in its entirety in mini-batches.

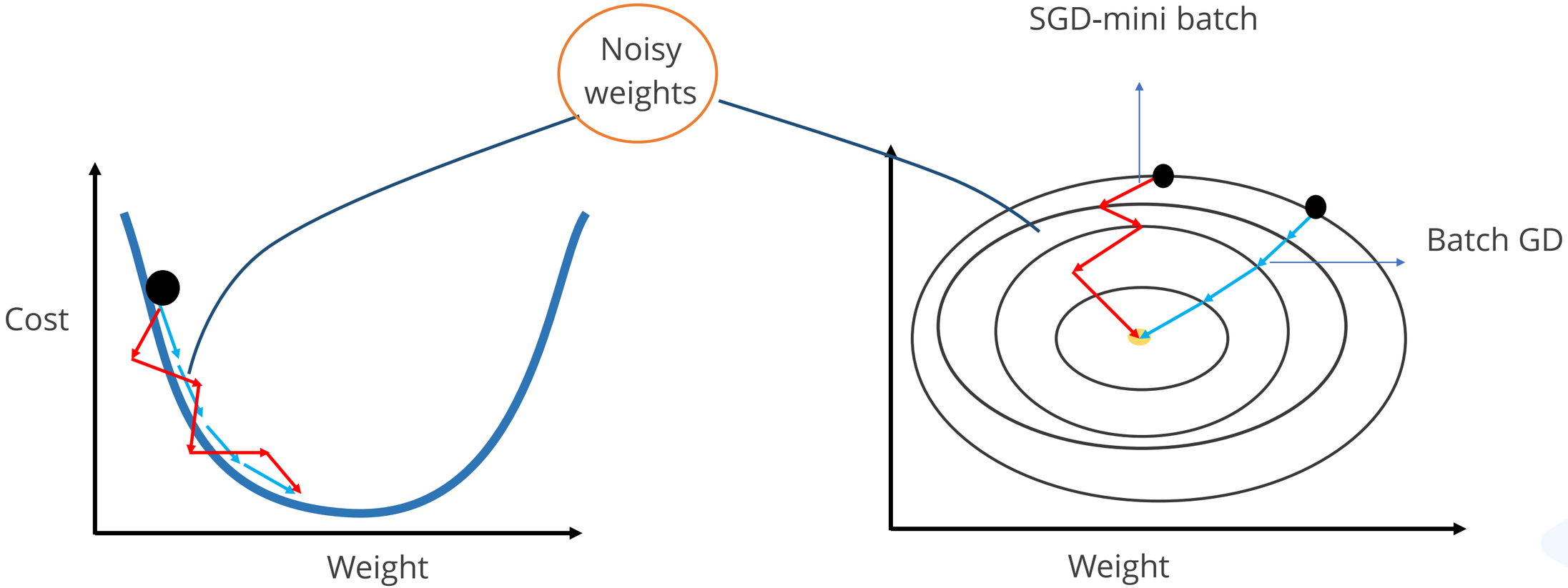
It divides the training data into small batches so that the network can easily be trained on the data.

The mathematical formulation is the same as vanilla GD, but the training occurs batch-wise.

For example, a training set of 400 examples can be divided into 10 batches of 40 training examples each. In this example, the weight evaluation equation will be iterated 10 times the number of batches.

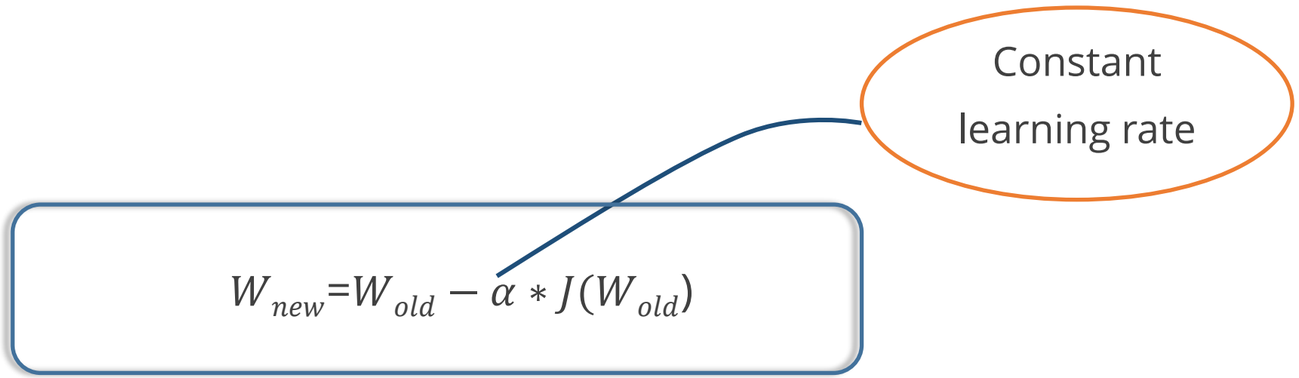
GD vs. SGD-Mini Batch

GD is computationally expensive, but it converges to the global minimum smoothly. In contrast, SGD creates more noisy weight, which, in turn, takes more time to reach the global minimum.



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 drawback of optimizers with a constant learning rate throughout all epochs.



Assisted Practices

Let’s understand the concept of SGD using Jupyter Notebooks.

- 6.04_Implementation of SGD

Note: Please refer to the Reference Material section to download the notebook files corresponding to each mentioned topic

ASSISTED PRACTICE

Discussion: Stochastic Gradient Descent

Duration: 10 minutes

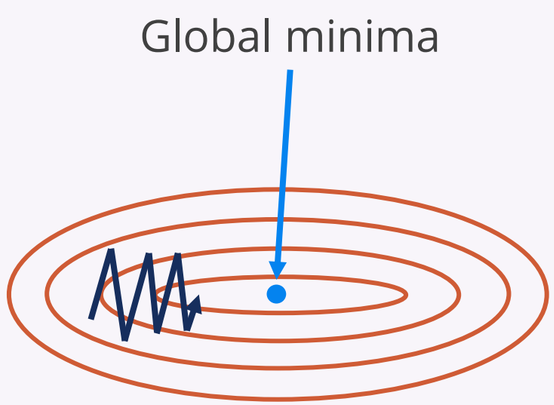
- What is Stochastic Gradient Descent?
  Answer: Stochastic Gradient Descent is an optimization algorithm used to train machine learning models. It updates model parameters incrementally by using mini-batches of training examples..

- What are the advantages of Stochastic Gradient Descent?
  Answer: Stochastic Gradient Descent offers advantages such as efficiency, the capability for online learning, and the potential for better generalization.

Introduction to Momentum

Momentum

It improves convergence, stability, and helps overcome local optima by introducing inertia and consistent direction during weight updates.



At iteration t,

$W_t = W_{t-1} - L * (d(loss)/d(W_{t-1}))$

Where,

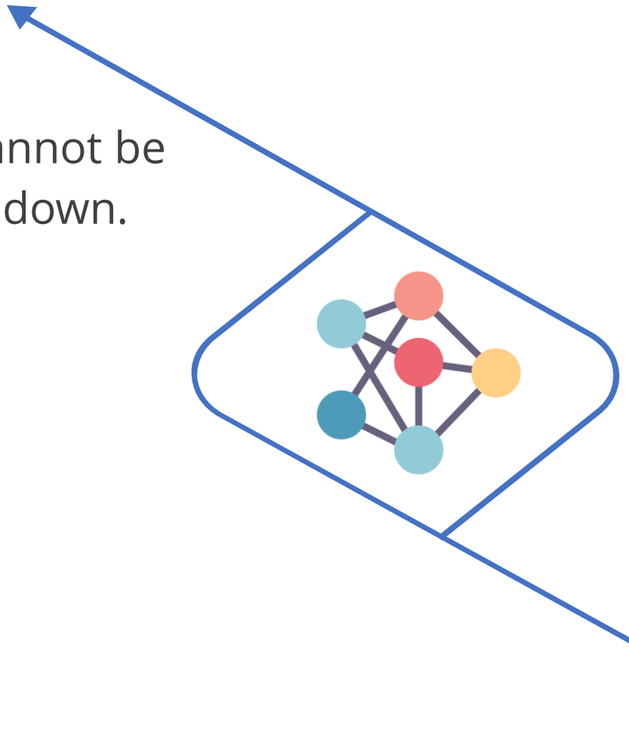
W_t is the updated value of the parameter
W_t-1 is the previous value
L is the learning rate (typically 0.01)

During model training, algorithm parameters are initialized randomly and then updated iteratively, progressively getting closer to the optimal value of the function.

Momentum

The algorithm has an assigned learning rate to prevent complete divergence.

Since the learning rate value cannot be very large, the process slows down.

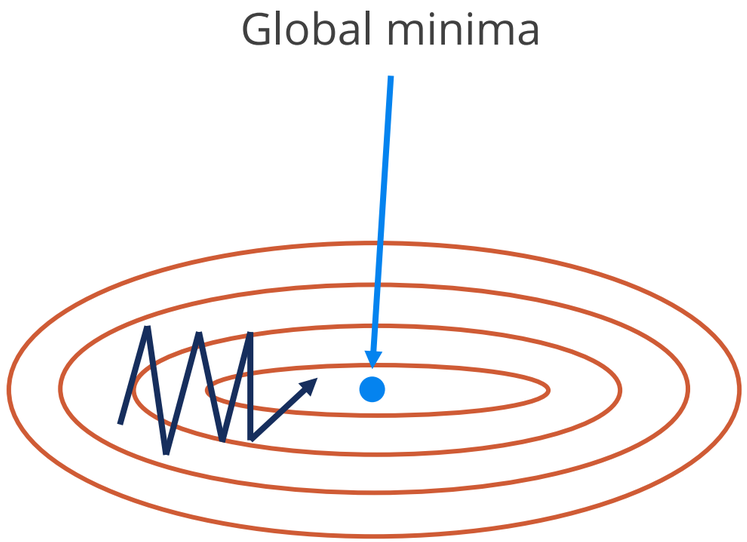


If the algorithm encounters a plateau, it may be deceived, thinking that it has reached the minimum.

To resolve these issues and correct the learning rate effect, the concept of momentum is used.

Momentum

Although it can easily handle smaller datasets, momentum is typically used in the vast, noisy datasets of neural networks.



The only disadvantage to using momentum is that it adds further complexity to the algorithm.

Momentum

The objective is to reach the global minimum swiftly.

At iteration t,

$$M_t = PM_{t-1} - (1-P)\frac{(\mathrm{d}(loss))}{d(w_{t-1})}$$

$$w_t = w_{t-1} - LM_t$$

Where,

M_t: Updated value of momentum

M_{t-1}: Previous value

P: Momentum hyperparameter (typically between 0.5 and 0.9)

W_t: Updated value of the parameter

W_{t-1}: Previous value

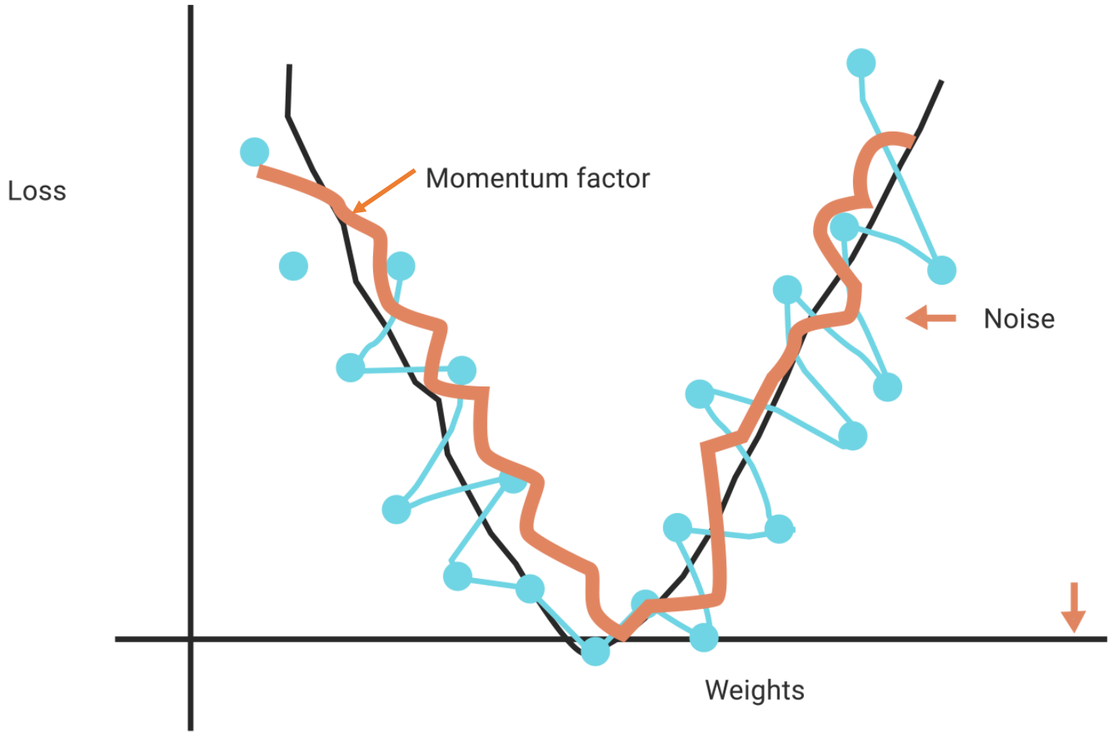
L: Learning rate (typically 0.01)

The descent must be steep initially and then slow down after a certain point to avoid overshooting the minima.

SGD with Momentum

- SGD with momentum is an advanced optimization algorithm that uses a moving average to update the trainable parameters.

- SGD with momentum is well-suited to overcoming the noisy weights of SGD.

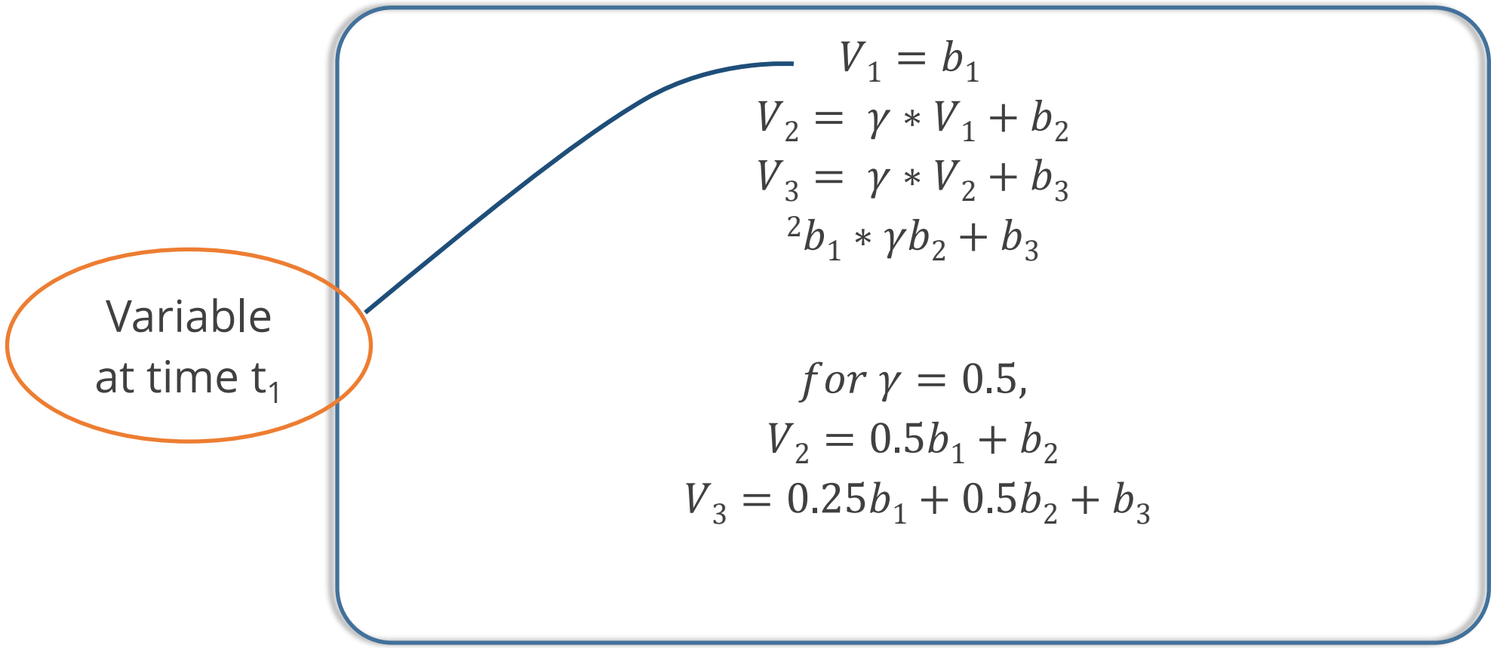


Source: https://i0.wp.com/www.datasciencelearner.com/wp-content/uploads/2021/01/SGD-with-Momentum.png?ssl=1

SGD with Momentum

- A moving average is a calculation to analyze data points by creating a series of averages of different subsets of the full dataset.
- For example, for times t1, t2, and t3 the corresponding data points are b1, b2, and b3.

According to the moving average:



Here,
γ: Decay factor

SGD with Momentum

- Apply a moving average to the weighted evaluation formula.
- In most cases, gamma = 0.9 works well.

$$J(W_{old}) = \frac{\partial L}{\partial W_{old}}$$

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

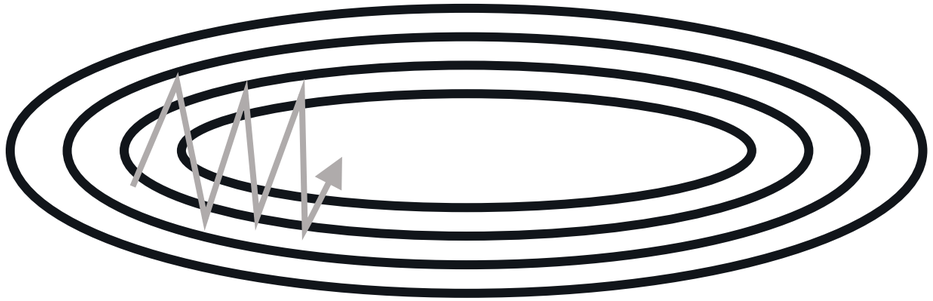
$$= W_{old} - [\gamma V_{t\_1} + \alpha * J(W)]$$

Momentum

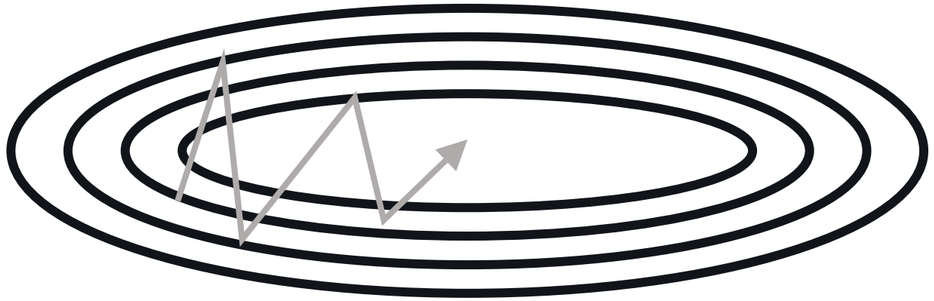
$$Vt_{\_1} = 1_{*} J(W)_t + \gamma * J(W)_{t\_1} + \gamma^2 * J(W)_{t\_2} \ldots\ldots$$

SGD With and Without Momentum

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



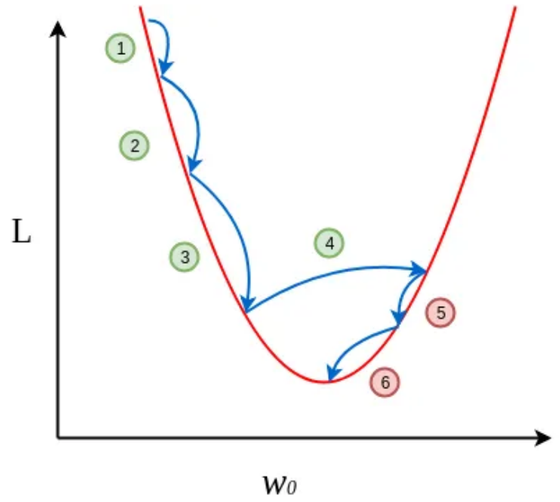
SGD with momentum



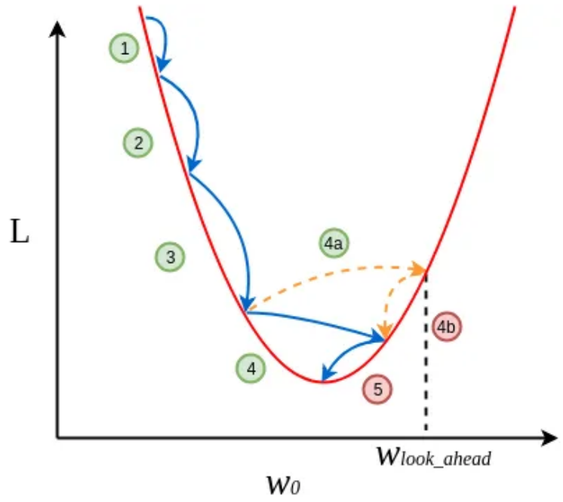
SGD without momentum

Nesterov Accelerated Gradient (NAG)

NAG combines momentum-based updates with lookahead adjustments for faster convergence compared to traditional SGD with or without momentum.



(a) Momentum-Based Gradient Descent



(b) Nesterov Accelerated Gradient Descent

$$\bigcirc \Longrightarrow \frac{\partial L}{\partial w_0} = \frac{Negative(-)}{Positive(+)}$$

$$\bigcirc \Longrightarrow \frac{\partial L}{\partial w_0} = \frac{Negative(-)}{Negative(-)}$$

*Source: https://miro.medium.com/v2/resize:fit:1400/format:webp/1*6MEi74EMyPERHlAX-x2Slw.png*

Nesterov Accelerated Gradient (NAG)

The key aspects of how NAG works compared to the traditional momentum method are:

Interim parameters

NAG considers interim parameters in velocity updates to avoid bad losses.

Interim velocity weight

NAG calculates an interim velocity weight, refining the update precision.

Gradient computation

NAG differs in gradient computation, using lookahead to estimate future positions accurately.

NAG vs. SGD with Momentum

Characteristics of NAG and SGD with the momentum method may resemble each other or differ significantly under various scenarios:

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

Assisted Practices

Let’s understand the concept of momentum using Jupyter Notebooks.

- 6.05_Implementation of Momentum

Note: Please refer to the Reference Material section to download the notebook files corresponding to each mentioned topic

ASSISTED PRACTICE

Introduction to AdaGrad

Discussion

Discussion: AdaGrad

Duration: 10 minutes

- What is AdaGrad?
- How does AdaGrad work?

Adaptive Gradient (AdaGrad)

Adagrad is an optimization algorithm that adjusts the learning rate for each parameter based on historical gradients, improving convergence and performance.

- Accumulates gradients over time from the entire training dataset
- Scales the learning rate individually for each parameter based on their historical gradients
- Is effective for handling sparse data and varying parameter importance

AdaGrad

Adaptive gradient (AdaGrad) optimization iteratively updates different learning rates for each parameter without manual tuning.

At iteration t,

$$G_t = \Sigma_{i=1}^{t} \left[\frac{\mathrm{d}(\mathrm{loss})}{\mathrm{d}(w_i)}\right]^2$$

$$L_t = \frac{L_{t-1}}{\sqrt{G_t + E}}$$

$$w_t = w_{t-1} - L_t \left(\frac{\mathrm{d}(loss)}{\mathrm{d}(w_{t-1})}\right)$$

Where,

G_t: Sum of the squares of gradients over time

L_t: Updated value of the learning rate

E: Small positive number

W_t: Updated value of the parameter

AdaGrad

AdaGrad updates the learning rate in each iteration to reach the global minimum swiftly.

It calculates the custom learning rate (step size) for one parameter as follows:

$$\frac{Step\ size}{1\times10^{-8}+\sqrt{s(t)}} = cust\ step\ size(t+1)$$

Where,

cust step size(t+1): Estimated step size for an input variable at a certain point

Step size is the beginning step size

$1x10^{-8}$ is the small constant value

sqrt() is the square root operation

s(t)is the sum of the squared partial derivatives for the input variable

AdaGrad

The AdaGrad optimizer is commonly used in machine learning and deep learning algorithms to adaptively adjust the learning rate during the training process.

The mathematical formulation for AdaGrad optimizer:

$$W_t = W_{t\_1} - \alpha * {}_t {}_* J(W_{t\_1})$$

$$\alpha * t = \frac{\alpha}{\sqrt{\beta t + \varepsilon}}$$

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

Initial learning rate

AdaGrad

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

$\boldsymbol{\varepsilon}$ is considered in the formula because if $\boldsymbol{\beta t}$ 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 the $\boldsymbol{\beta t}$ value can sometimes turn out to be very large.

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

Advantages of AdaGrad

Some of the advantages of AdaGrad are:

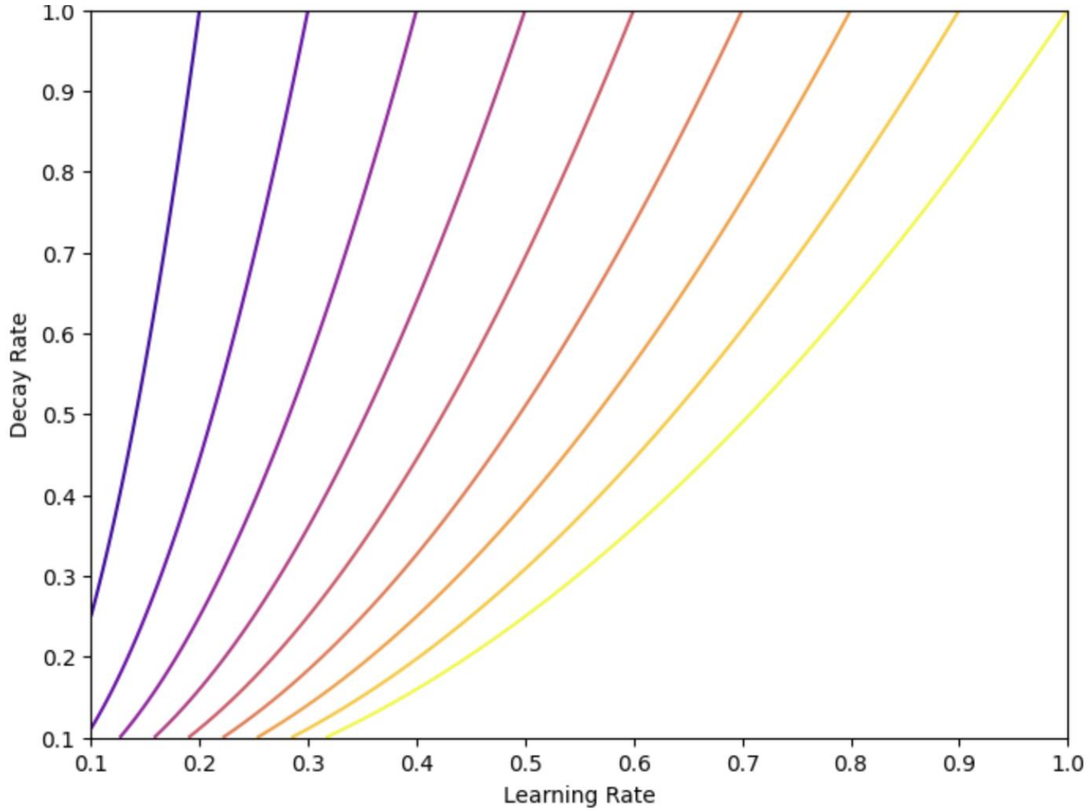
01 It eliminates the need to manually tune the learning rate.

02 It facilitates faster convergence and is more reliable.

03 It is not very sensitive to the size of the master step.

Disadvantages of AdaGrad

The main disadvantage is that the learning rate value might decrease rapidly, leading to slower convergence.



To resolve this, a decay factor is added to AdaGrad using root mean square propagation to maintain the sum of squares in the denominator at a manageable size.

Assisted Practices

Let’s understand the concept of AdaGrad using Jupyter Notebooks.

- 6.06_Implementation of AdaGrad

Note: Please refer to the Reference Material section to download the notebook files corresponding to each mentioned topic

ASSISTED PRACTICE

Discussion: AdaGrad

Duration: 10 minutes

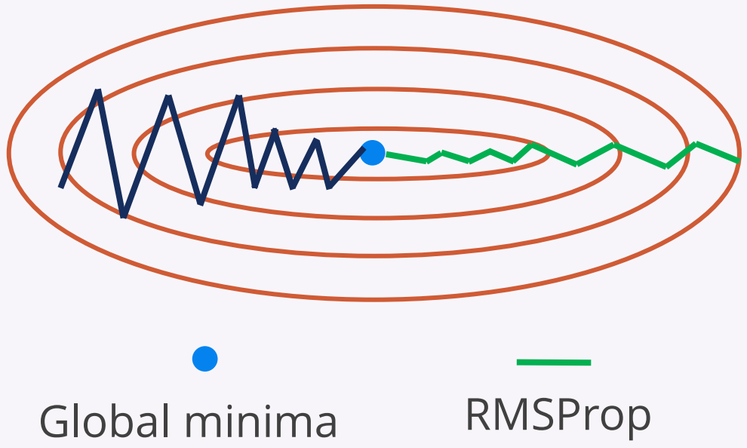
- What is AdaGrad?
  Answer: AdaGrad is an optimization algorithm used in machine learning that adapts the learning rate of each parameter based on their historical gradients.

- How does AdaGrad work?
  Answer: AdaGrad works by adjusting the learning rate for each parameter individually. It divides the learning rate by a sum of the squared gradients of that parameter, giving larger updates for infrequent parameters and smaller updates for frequent parameters.

Introduction to RMSProp

What Is RMSProp?

The Root Mean Square Propagation (RMSProp) optimizer is a momentum-based version of the gradient descent technique.



It limits the oscillations in the vertical plane.

It boosts the learning rate, allowing the algorithm to take greater horizontal steps to converge faster.

RMSProp and gradient descent differ in the way the gradients are calculated.

RMSProp: Equation

The RMSProp and gradient descent with momentum are determined in the following way:

$$v_{dw} = \beta * v_{dw} + (1 - \beta) * dw$$

$$v_{db} = \beta * v_{db} + (1 - \beta) * db$$

$$W = W - \alpha * vdw$$

$$b = b - \alpha * vdb$$

Here, vdw: Velocity for weights

vdb: Velocity for biases

β: Momentum coefficient

dw: Current gradient of weights

db: Current gradient of biases

W: Weights

b: Biases

RMSProp: Equation

Beta is the measure of momentum and is commonly set to 0.9.

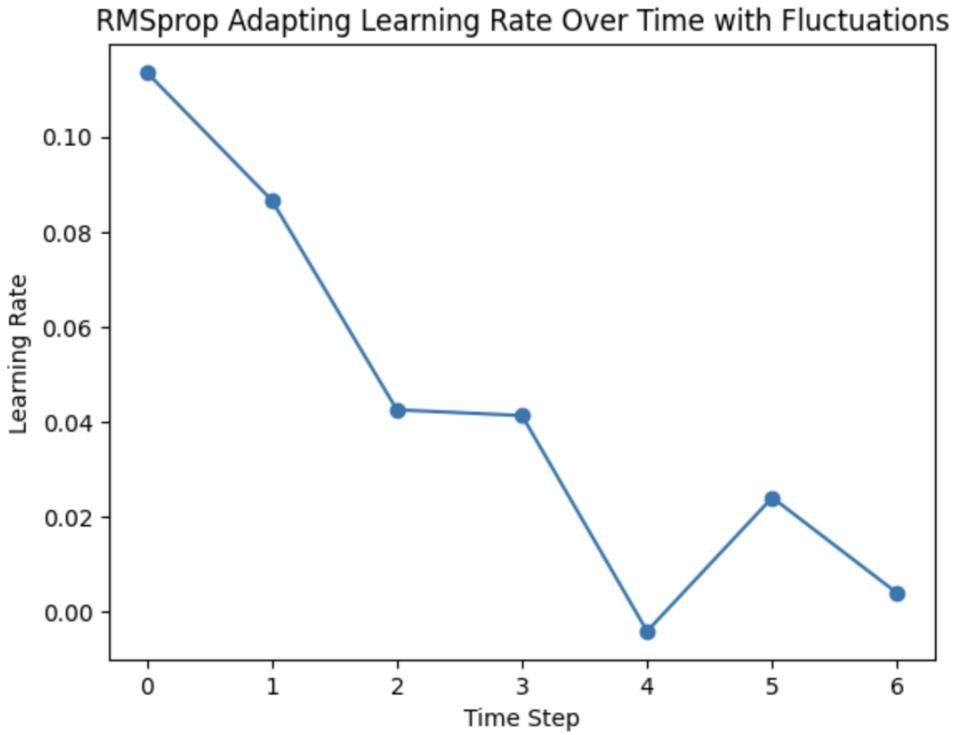
To normalize the gradient, RMSProp uses a moving average of squared gradients.

This normalization equalizes the step size or momentum.

Normalization lowers the step size for high gradients to avoid exploding and raises it for minor gradients to avoid vanishing.

Root Mean Square Propagation (RMSProp)

RMSProp treats the learning rate as an adaptive parameter rather than a hyperparameter.



This indicates that the rate of learning fluctuates with time.

Root Mean Square Propagation (RMSProp)

RMSprop was developed to mitigate the drawbacks of AdaGrad.

The formula for RMSprop is as follows:

$$W_t = W_{t\_1} - \alpha * t * J(W_{t\_1})$$

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

$$W_{avg_{(t)}} = \gamma * W_{avg_{(t\_1)}} + (1 - \gamma) J(W)^2$$

Where,

W_t: Updated weights at time step t

W_{t-1}: Previous weights from the previous time step

$\alpha * t$: Adaptive learning rate at time step t

α: Initial learning rate (constant)

t: Current time step

$Wavg(t)$: Accumulated squared gradients at time step t

γ: Decay factor

$J(W)$: Cost

ε: Constant

RMSProp: Syntax

The Python code for implementing RMSprop is as follows:

Syntax:-

```
def RMSprop(index, beta, db, dw,vdw, vdb, alpha);
vdw = beta*vdw + (1 – beta)*dw**2
Vdb =(beta*vdb + (1-beta)*db**2

Model.layers[index].weight-=(alpha/(np.sqrt(vdw)+1e-8))*dw
Model.layers[index].bias-=(alpha/(np.sqrt(vdb)+1e-8))*db
```

Here, epsilon ε= e-8 is added for weight and bias to maintain numerical stability.

Assisted Practices

Let’s understand the concept of RMSProp using Jupyter Notebooks.

- 6.07_Implementation of RMSProp

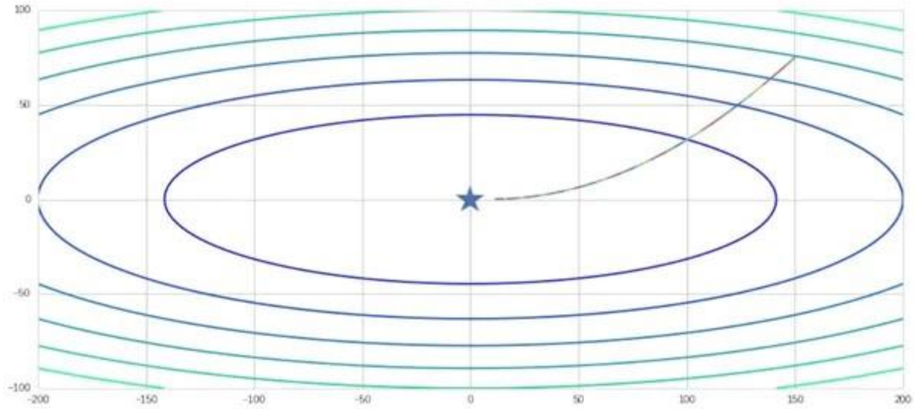
Note: Please refer to the Reference Material section to download the notebook files corresponding to each mentioned topic

ASSISTED PRACTICE

Introduction to Adadelta

Adadelta

Adadelta is an optimization algorithm used to train neural networks. It builds on AdaGrad and RMSProp, altering the custom step size calculation.



This approach removes the need for an initial learning rate hyperparameter.

-

*Source: https://miro.medium.com/v2/resize:fit:640/format:webp/0*7oiBl24fy1ZA7Qv9.png*

Adadelta: Selecting the Right Algorithm

To achieve optimal performance, choose the algorithm that aligns with the behavior of the data variables in the specific application.

Understand the data

- Analyze data variables' behavior and characteristics
- Identify challenges such as sparsity or noisy gradients

Choose the appropriate algorithm

- Recognize the strengths and weaknesses of various algorithms
- Use adaptive learning rate methods without manual tuning

Adadelta: Selecting the Right Algorithm

Align with application

- Assess the alignment of Adadelta's features with data behavior
- Ensure the compatibility of the algorithm with the specific task

Optimize performance

- Fine-tune hyperparameters, including the decay factor
- Evaluate and adjust continuously to align with data variables

Adadelta: Equation

Consider the parameter ρ, where the leaky updates are stored in the state variable as below:

$$S_t = \rho S_{t-1} + (1- \rho)g_t^2$$

Here,

S_t: Current EWMA (Exponentially Weighted Moving Average) of squared gradients

ρ: Decay factor for the contribution of the previous value

S_{t-1}: Previous EWMA of squared gradients

(1-ρ): Weight for the current squared gradient

g_t^2: Squared gradient at the current step

Adadelta: Equation

The algorithm performs the update with the rescaled gradient:

$$X_t = X_{t-1} - g_t'$$

Where

$$g_t' = \frac{\sqrt{\Delta X_{t-1} + \epsilon}}{\sqrt{S_t + \epsilon}} g_t$$

Here, ΔX_{t-1} = Leaky average of the squared rescaled gradients.

g_t' = The rescaled gradient at time *t*

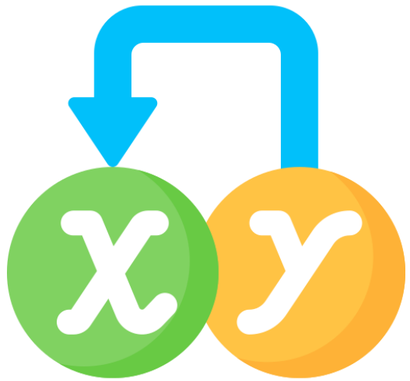
$\Delta X_0 = 0$ is updated at each step with the rescaled gradient as shown:

$$\Delta X_t = \rho \Delta X_{t-1} + (1 - \rho) g_t'^2$$

ϵ is in the range of 10^{-5} to maintain numerical stability.

Adadelta: Equation

In the algorithm, Adadelta uses two state variables:



- $\mathbf{s}_t$ to store a leaky average of the second moment of the gradient
- $\mathbf{\Delta x}_t$ to store a leaky average of the second moment of the change of parameters in the model itself

Adadelta: Syntax

It can be used with Keras in the following way:

Syntax:-

```
tf.keras.optimizers.Adadelta(
 Learning_rate=0.001, rho=0.95, epsilon=1e-07, name="Adadelta", **kwargs
)
```

The learning rate can be set to higher values for better performance.

ρ is the decay rate and can be a tensor or a floating point value.

ε maintains numerical stability to offset cases like dividing by zero and rounding off issues.

Summary of Adadelta

To summarize:

Adadelta requires two state variables to store the second moments of the gradient and the change.

It has no learning rate parameter.

It uses leaky averages to estimate the appropriate statistics.

Assisted Practices

Let’s understand the concept of Adadelta using Jupyter Notebooks.

- 6.08_Implementation of Adadelta

Note: Please refer to the Reference Material section to download the notebook files corresponding to each mentioned topic

ASSISTED PRACTICE

Adam Optimizer

What Is Adam Optimizer?

The Adam optimizer is a popular algorithm in deep learning, optimizing stochastic objectives using adaptive estimates. It efficiently handles sparse gradients and noisy problems with low resources.

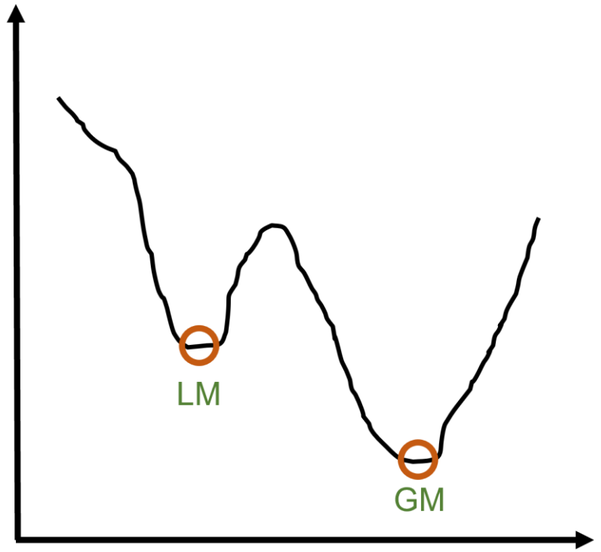
It effectively handles large-scale problems with extensive data or numerous parameters.

It doesn't require excessive memory, but memory use mainly depends on the model architecture.

Adam combines momentum and RMSProp concepts for effective parameter space navigation and fast convergence

Adam Optimizer

With Adam, limit the pace of gradient descent to minimize oscillation.



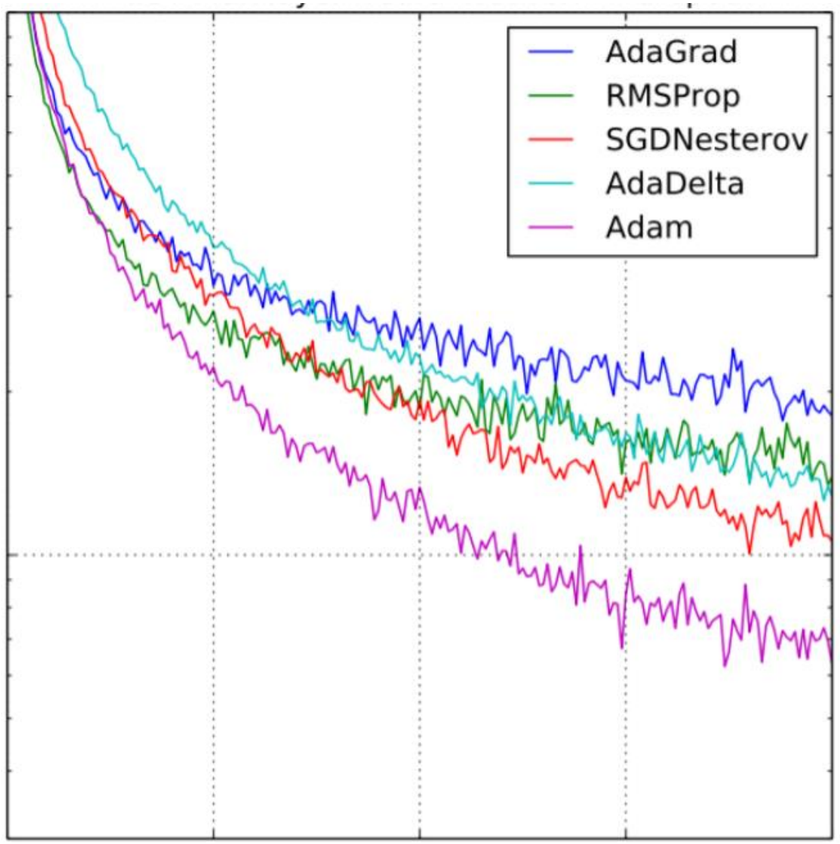
GM - Global Minimum
LM - Local Minimum

- It takes large enough steps (step size) to avoid local minimum barriers.
- As a result, it integrates the qualities of the preceding strategies to efficiently obtain the global minimum.
- It outperforms the optimization algorithm by a large margin to provide an optimized gradient descent.

Source: https://media.geeksforgeeks.org/wp-content/uploads/20200909192459/LMGM.png

Adam Optimizer

Comparing the Adam optimizer with other optimizers for the MNIST dataset yields the following:



Adam performed more efficiently than other optimizers.

Adam Optimizer

It keeps an exponentially decaying average of the past gradient $\mathbf{m_t}$.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$V_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

m_t: Momentum of gradients

β_1: Decay rate for momentum

g_t: Current gradient value

v_t: Moving average of squared gradients

β_2: Decay rate for v_t

Adam Optimizer

m_t and v_t are the estimates of the first and second moment of the gradients, respectively.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{\upsilon}_t = \frac{\upsilon_t}{1 - \beta_1^t}$$

The updated parameters are used just like in Adadelta and RMSProp, which yields the following Adam update rule:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{\upsilon}_t} + \epsilon} \hat{m}_t.$$

Adam Optimizer

The parameters used are:

To prevent zero division, a tiny positive constant, ϵ (10^{-8}), is added.

$\beta 1$ and $\beta 2$ = decay rates of the average of gradients ($\beta 1 = 0.9$ and $\beta 2 = 0.999$ recommended)

η = learning rate (0.001)

Adam Optimizer

The Adam optimization algorithm combines momentum and adaptive learning rates, enabling efficient handling of sparse gradients and providing optimized learning performance.

Updated momentum:

$$Mi(t) = momentum * Mi(t-1) + (1 - momentum) * gradient$$

Where,

$Mi(t)$: Momentum vector at time t

$momentum$: Hyperparameter determining gradient's contribution

$gradient$: Gradient of the loss function

$Wi(t)$: Model's weights at time t

$learning\_rate$: Hyperparameter controlling step size in gradient descent

Adam Optimizer

The Adam optimization algorithm utilizes the following formulas for efficient training in machine learning models:

Update mean (first moment)

$$M_t = \beta_1 * M_{t-1} + (1-\beta_1) * G_t$$

Update variance (second moment)

$$V_t = \beta_2 * V_{t-1} + (1-\beta_2) * (G_t)^2$$

Update weights

$$W_t = W_{t-1} - \alpha * M_t / (\sqrt{V_t} + \varepsilon)$$

Assisted Practices

Let’s understand the concept of Adam using Jupyter Notebooks.

- 6.09_Implementation of Adam

Note: Please refer to the Reference Material section to download the notebook files corresponding to each mentioned topic

ASSISTED PRACTICE

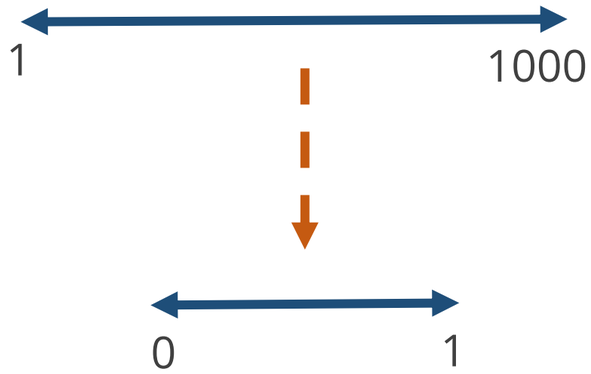
Batch Normalization

Data Preprocessing

In preprocessing, the data is generally normalized or standardized.

Normalization

A typical normalization involves 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 divide the difference by the standard deviation.

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

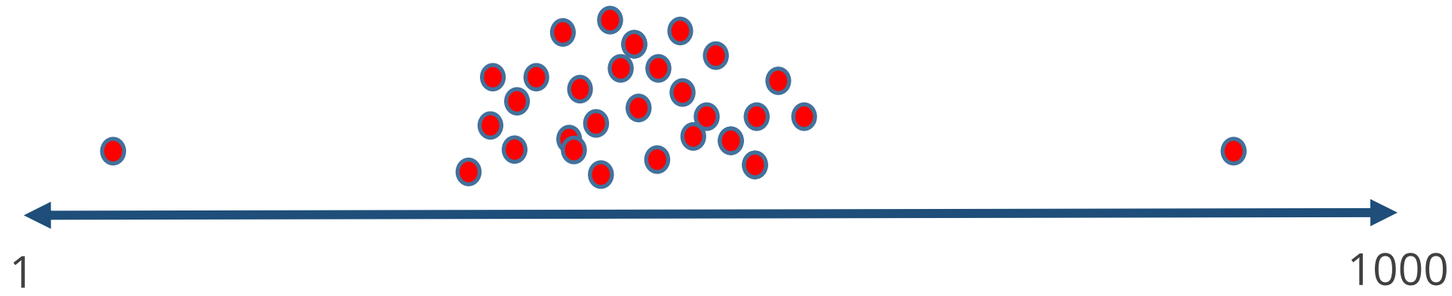
X = Data points

m = Mean

σ = Standard Deviation

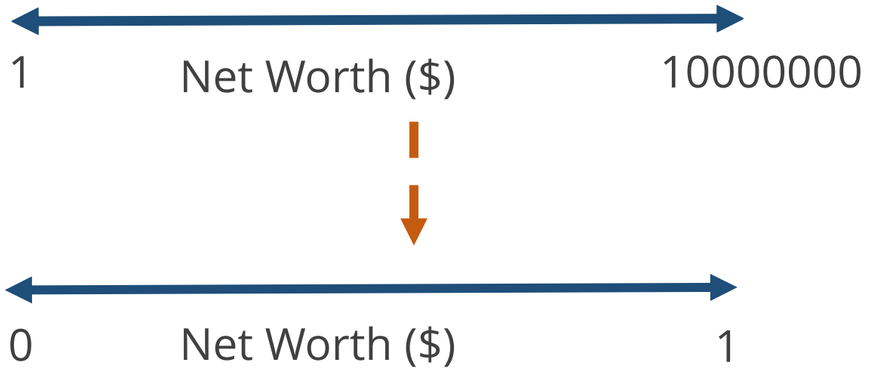
Why Data Preprocessing?

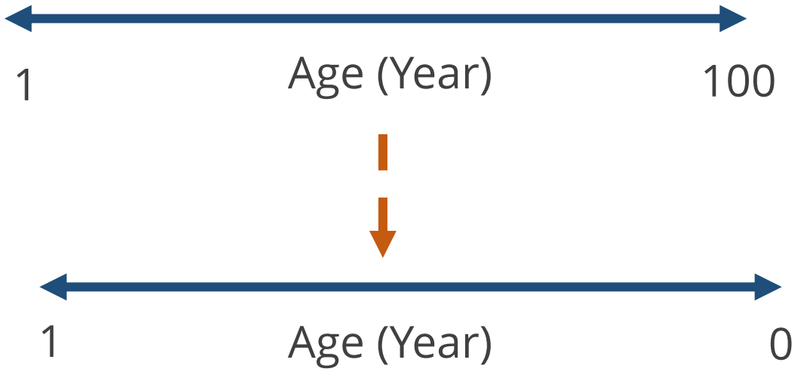
Data points can either be high or low. This leads to cascading effects in the network. Therefore, data preprocessing is needed.



Why Data Preprocessing?

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





Normalization Techniques

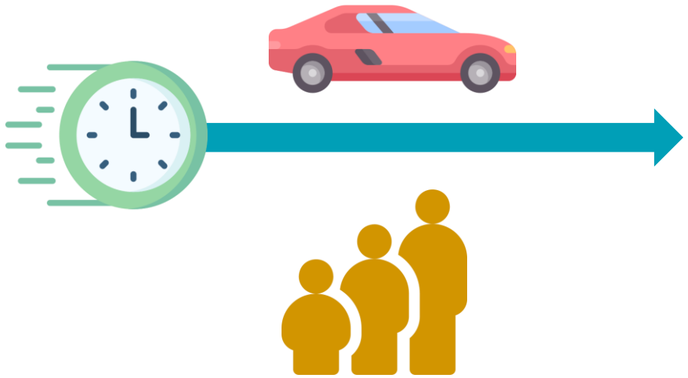
During the preprocessing procedure, before training a neural network, the input is either normalized or standardized.

During the data preparation stage, the data is prepared for use in training.

This exercise alters the data so that all data points are on the same scale.

Normalization Techniques: Example

Each of the characteristics in every sample differs significantly.



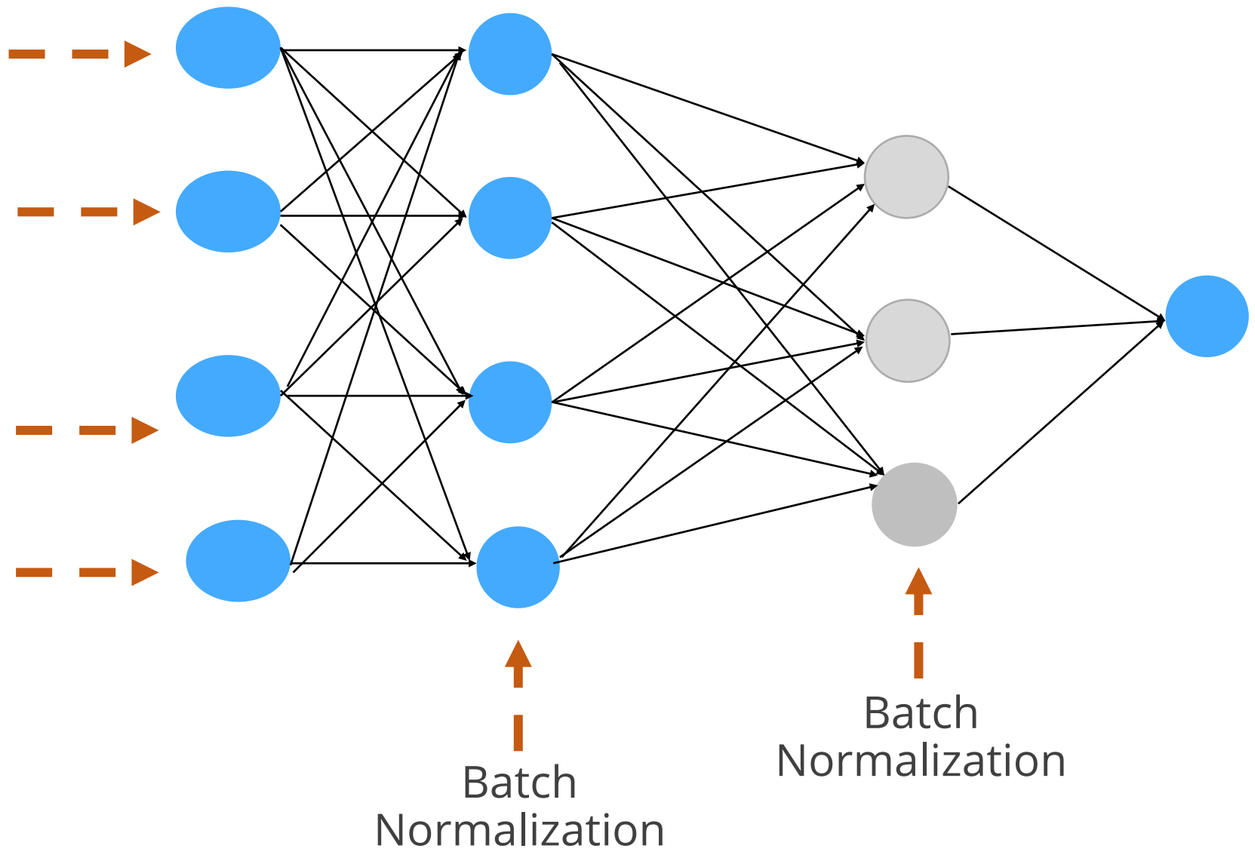
Example

If one characteristic relates to an individual's age and the other to the individual’s miles driven in a car in the last five years, these two data points, age and miles traveled, will not be on the same scale.

Batch Normalization Implementation

Batch Normalization

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



Batch Normalization Process

The batch normalization process involves the following:

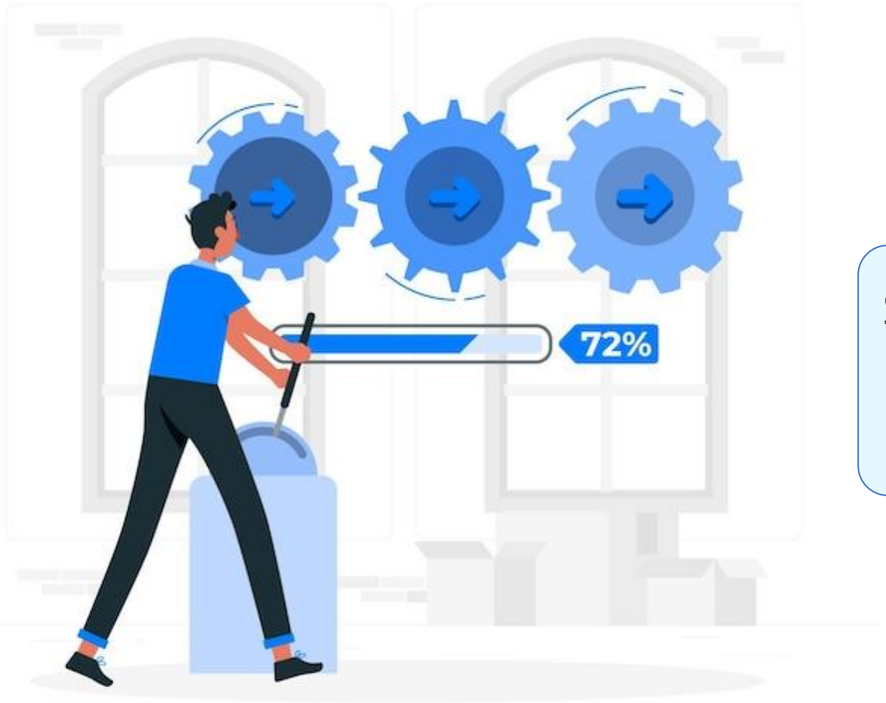
Steps	Expression	Description
1	$Z = \frac{\mathrm{X} - m}{\sigma}$	Normalize output x from the activation function
2	$Z * \mathrm{g}$	Multiply normalized output z by the arbitrary parameter g
3	$(Z * \mathrm{g})$+b	Add an arbitrary b to the resulting product (z * g)

m: Mean of the mini-batch, **s:** Standard deviation (or variance) of the mini-batch

g: Scale parameter (or gamma), **b:** Shift parameter (or beta)

Batch Normalization Process

The data is given a new standard deviation and mean with two arbitrarily chosen, trainable parameters, g and b.



Since normalization is included in the gradient process, the weights inside the network do not become imbalanced.

Thus, batch norms can accelerate training and reduce the ability of outlying heavy weights to sway the training process.

Batch Normalization Process

It normalizes output data from the model's activation functions for specific layers.

Therefore, the data flowing in and the data within the model are both normalized.

The entire process occurs on a per-batch basis and is thus called batch norm.

Implementing Batch Normalization Using Keras

- Batch normalization is an additional imported library.
- Initialize batch normalization after the ReLu activation function.

Syntax:-

```
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')])
```

Implementing Batch Normalization Using Keras

The batches are determined by the batch size that is set when the model is trained.

Syntax:-

```
import tensorflow as tf

model = tf.keras.Sequential([
 tf.keras.layers.Dense(16, activation='relu'),
 tf.keras.layers.Dense(32, activation='relu'),
 tf.keras.layers.BatchNormalization(axis=1),
 tf.keras.layers.Dense(2, activation= 'softmax')])
```

This is how a batch norm can be added to a model using Keras.

Components of Batch Normalization

The model has the following:

Input layer
Depends on the number of features in the dataset.

Hidden layer 1 with 16 nodes
Dense layer
Batch normalization (optional)
ReLU activation

Hidden layer 2 with 32 nodes
Dense layer
Batch normalization (optional)
ReLU activation

Output layer with 2 nodes
Dense layer
Softmax activation

Components of Batch Normalization

There is a batch normalization layer between the last hidden layer and the output layer.

```
tf.keras.layers.BatchNormalization(axis=1)
```

This is how it is specified in Keras, with a batch normalization object after the layer for which the activation output needs normalization.

Components of Batch Normalization

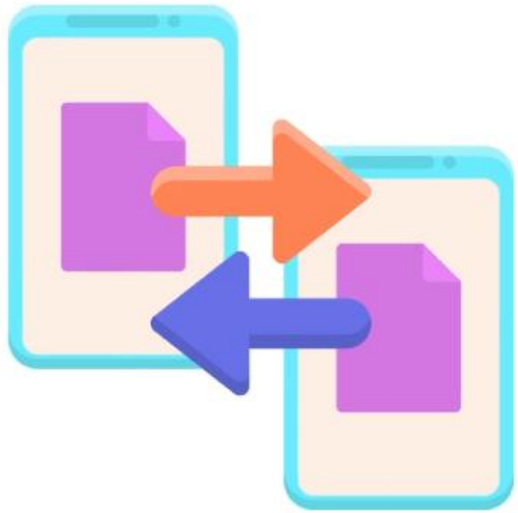
The axis parameter specifies the axis of the data to be normalized and is often the feature axis.

Many other parameters can also be specified and adjusted, including beta_initializer and gamma_initializer.

The beta_initializer and gamma_initializer parameters are commonly used in neural network architectures, especially in normalization layers like Batch Normalization.

Applying Batch Norm to a Layer

When a batch norm is applied to a layer, it first normalizes the output from the activation function that goes as input to the next layer.

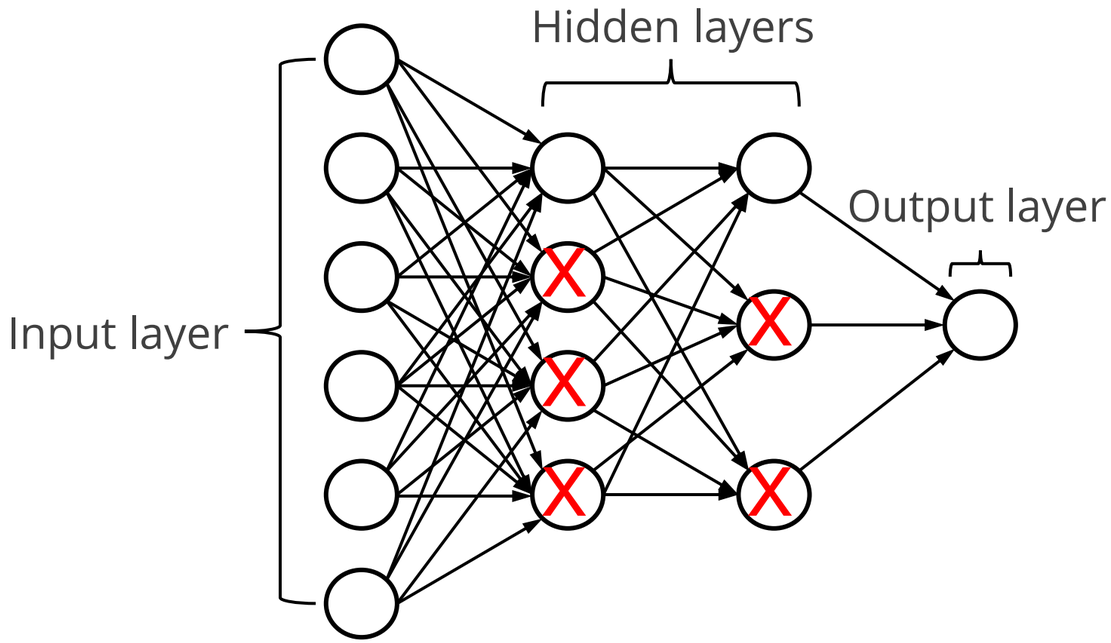


The purpose of batch normalization is to normalize this output, making it more standardized and easier to train.

Regularization

Regularization

Regularization is a technique that makes slight modifications to the learning algorithm so that the model generalizes more effectively.



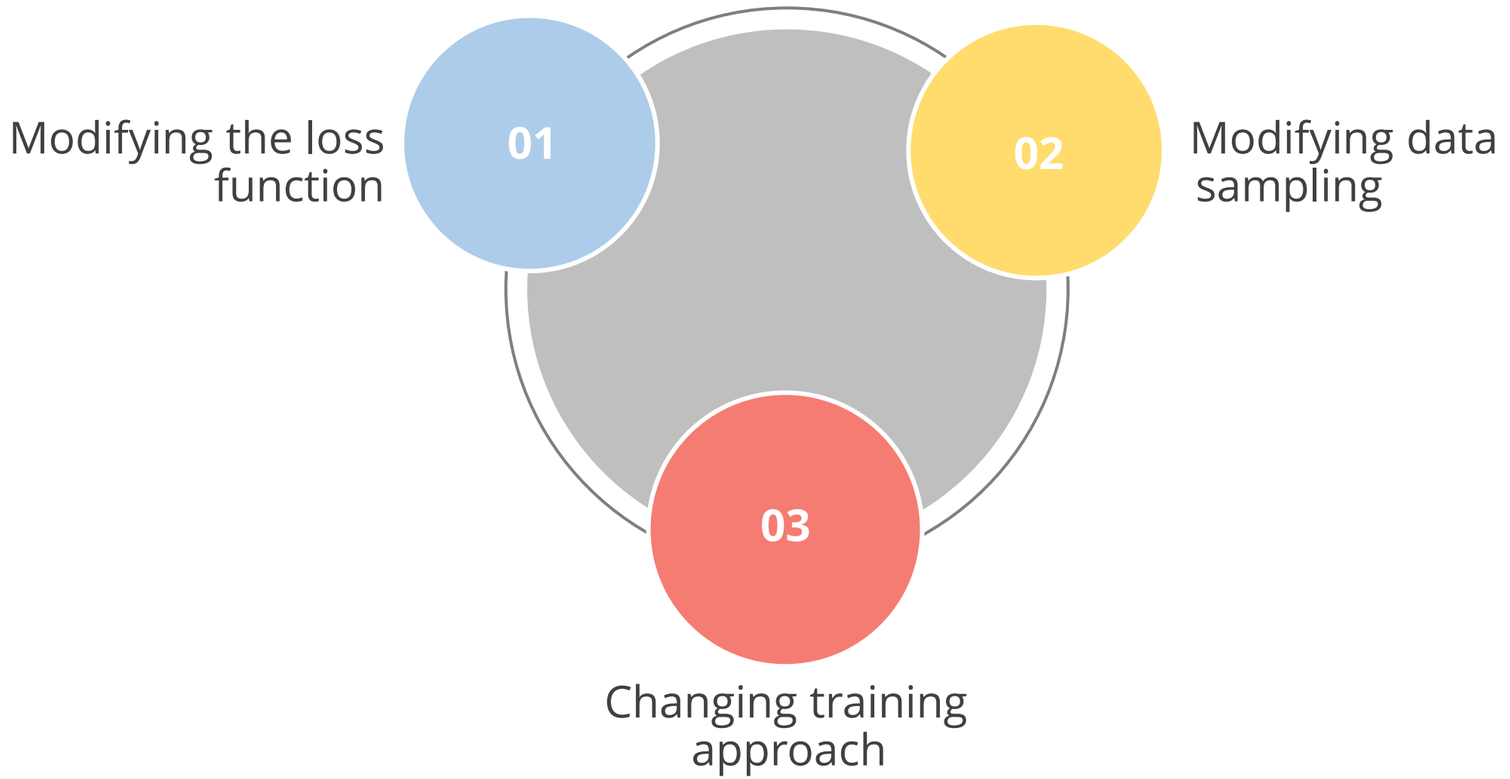
Regularization helps reduce error by fitting a function appropriately to the given training set and avoiding overfitting.

Why Regularization?

In machine learning, regularization is frequently employed as a solution to the overfitting problem.

Types of Regularization

The various forms of regularization employed in machine learning models are:



Modifying the Loss Function

There are two components of modifying the loss function in the context of regularization strategies:

- In regularization strategies, the loss function is adjusted to directly consider the norm of the learned parameters or the output distribution to improve the model.
- Regularization itself involves modifying the loss function to penalize large weight values.

Modifying the Loss Function: Strategies

The various strategies for modifying the loss function are discussed below:

01

L2 Regularization

- More weights make the model more complicated, increasing the chances of overfitting.

02

L1 Regularization

- L1 regularization introduces weight sparsity and decreases more weights to zero rather than lowering the average magnitude of all weights.

03

Entropy

- Entropy measures the uncertainty in a probability distribution. The greater the distribution's uncertainty, the greater the entropy.

Modifying Data Sampling

The two aspects related to modifying data sampling in regularization strategies are:

- These regularization approaches attempt to alter the available input to provide a fair depiction of the real input distribution.
- These strategies are beneficial for overcoming overfitting caused by the limited size of the available dataset.

Modifying Data Sampling: Strategies

The two strategies for modifying the data sampling are:

01

Data augmentation

- Create extra data from existing data by randomly cropping, dilating, rotating and adding a little noise

02

K-fold cross-validation

- Separate the data into k groups, train with (k-1) groups, and test with the remaining k group
- Experiment with all the possible k combinations

Change Training Approach

There are two components of the change training approach in the context of regularization strategies:

Algorithm modification

Adding regularization terms to the learning algorithm to prevent overfitting

Data augmentation

Increasing dataset size and diversity through modifications of existing data to improve the model's generalization ability

Change Training Approach: Strategies

The various strategies for changing training approach are:

01

Injecting noise

- It improves generalization, prevents overfitting, and is widely used in the deep learning industry to enhance model performance on unseen data.

02

Dropout

- It guarantees that the neural network learns a more stable collection of characteristics that function as well with random node sections.

Dropout and Early Stopping

Dropout Layer

The Dropout layer is a regularization technique used to prevent overfitting in neural networks. Following are the steps to be performed for dropout:

Step 1: Choose a dropout rate, typically between 0.2 and 0.5.

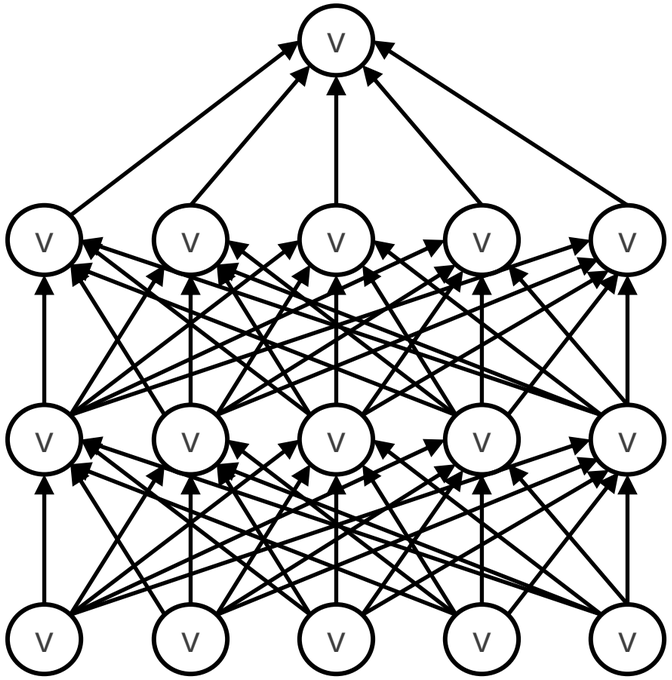
Step 2: Apply dropout during the forward pass by randomly setting a fraction of neurons to zero with the chosen dropout rate.

Step 3: Scale the remaining activations by dividing them by (1 - dropout rate).

Step 4: Complete the forward pass through the network and update weights based on gradients during backpropagation.

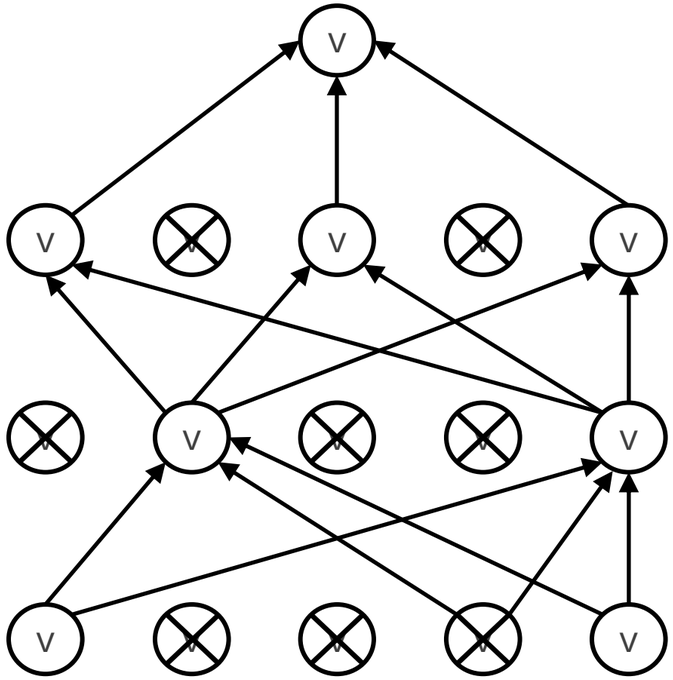
Dropout Layer

Example: A typical neural network contains two hidden layers.



(a) Standard neutral net

This figure depicts a standard neural network that is fully connected, with no dropout applied. Each node is connected to every other node in the next layer.



(b) After applying dropout

This figure depicts a thinned version of the previous network, achieved by applying dropout. Some nodes are turned off (dropped out), resulting in a thinned network.

How to Use Dropout

In a neural network, dropout is implemented with most types of layers, including:

Convolutional layers

Dense fully connected layers

Recurrent layers

How to Use Dropout

Best practices for using dropout are as follows:

Use a value of 0.5 to maintain the output of every node in a hidden layer

Use a value close to 1.0, such as 0.8, to retain inputs from the visible layer

Dropout regularization reduces overfitting by randomly deactivating neurons during training, promoting the learning of robust features and improving generalization.

Implementing Dropout Using Keras

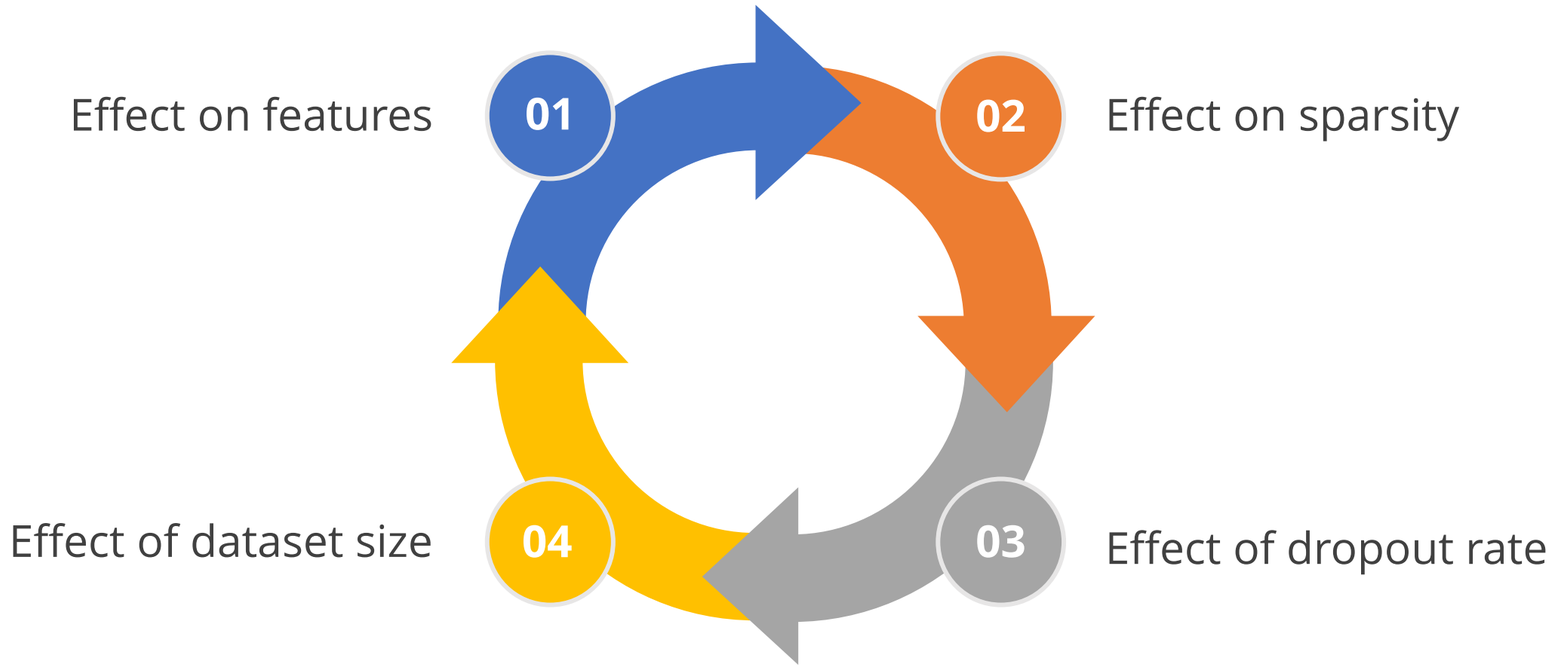
To make the usual predictions, scale the weights based on desired dropout rate before the network is finalized.

Syntax:-

```
) # Example of output size for a 10-class classification
model = tf.keras.Sequential([
 tf.keras.layers.Flatten(),
 tf.keras.layers.Dense(1024, activation=tf.nn.relu),
 tf.keras.layers.Dropout(0.2),
 tf.keras.layers.Dense(1024, activation=tf.nn.relu),
 tf.keras.layers.Dropout(0.2),
 tf.keras.layers.Dense(1024, activation=tf.nn.relu),
 tf.keras.layers.Dropout(0.2),
 tf.keras.layers.Dense(10, activation=tf.nn.softmax
])
```

Features of Dropout Layer

Dropout layer has the following features:



Features of Dropout Layer

Effect on features

The characteristics depicted in Figure (a) have co-adapted to create acceptable reconstructions. In Figure (b), the concealed units appear to identify edges, strokes, and dots in various sections of the image.

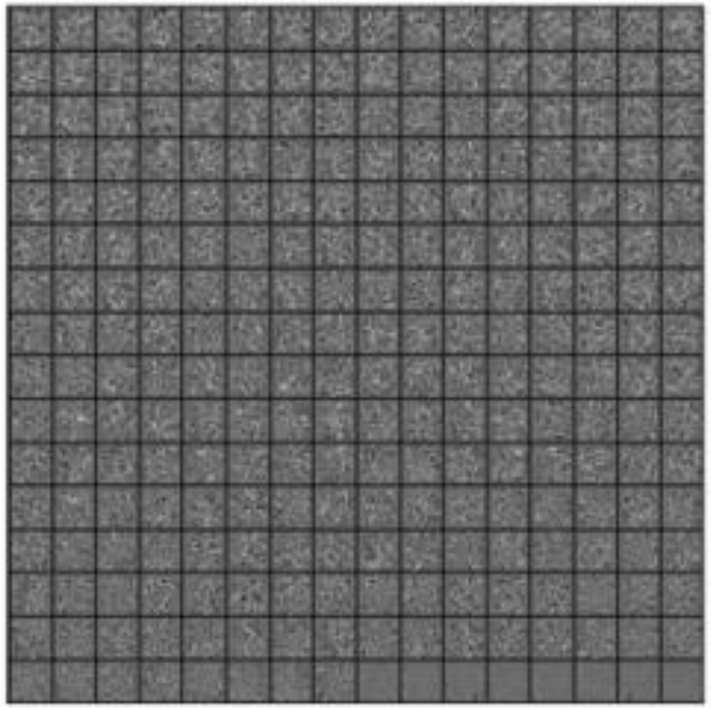


Figure (a) Without dropout

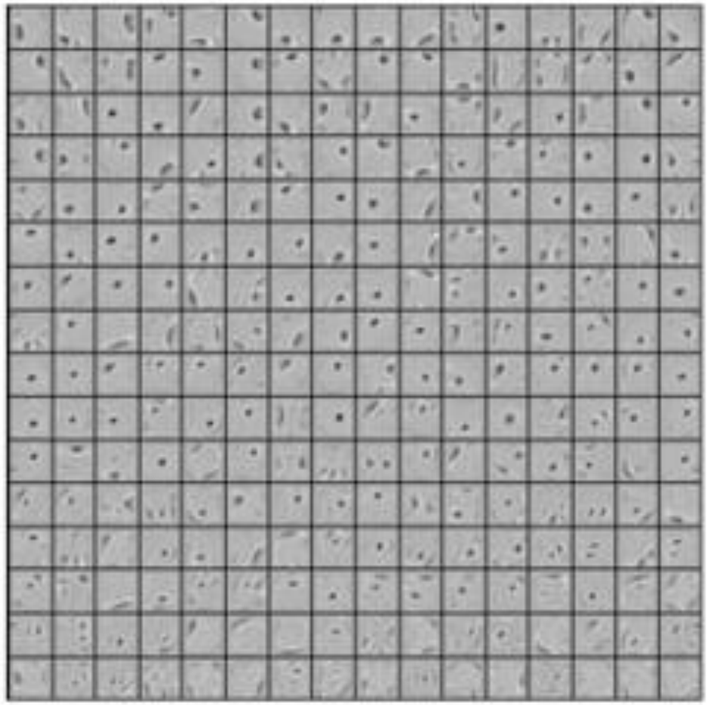


Figure (b) Drop with p = 0.5

Source: https://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

Features of Dropout Layer

Effect on sparsity

The activations of the hidden units become sparse as a side consequence of doing dropout. The histogram of mean activations in Figure (a) shows that most units have a mean activation of around 2.0, whereas in Figure (b) it is 0.7.

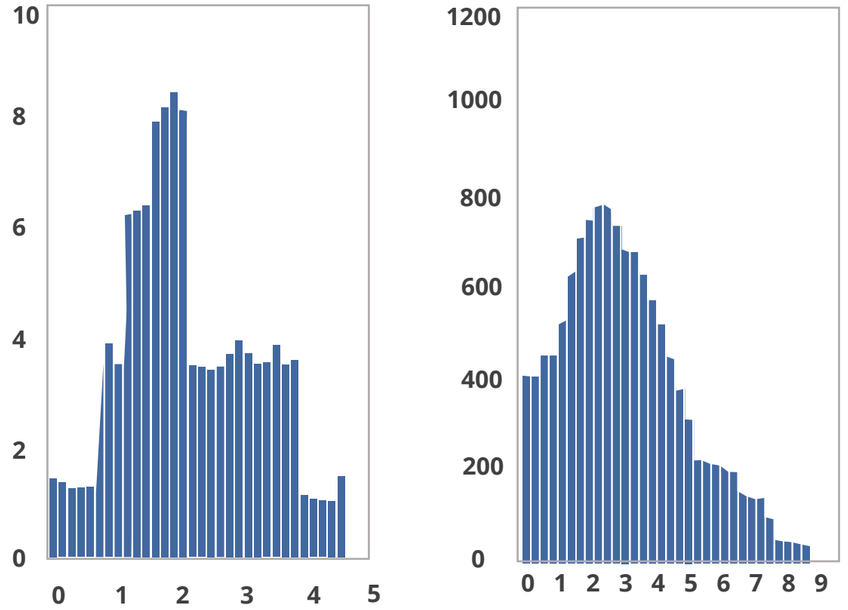


Figure (a) Without dropout

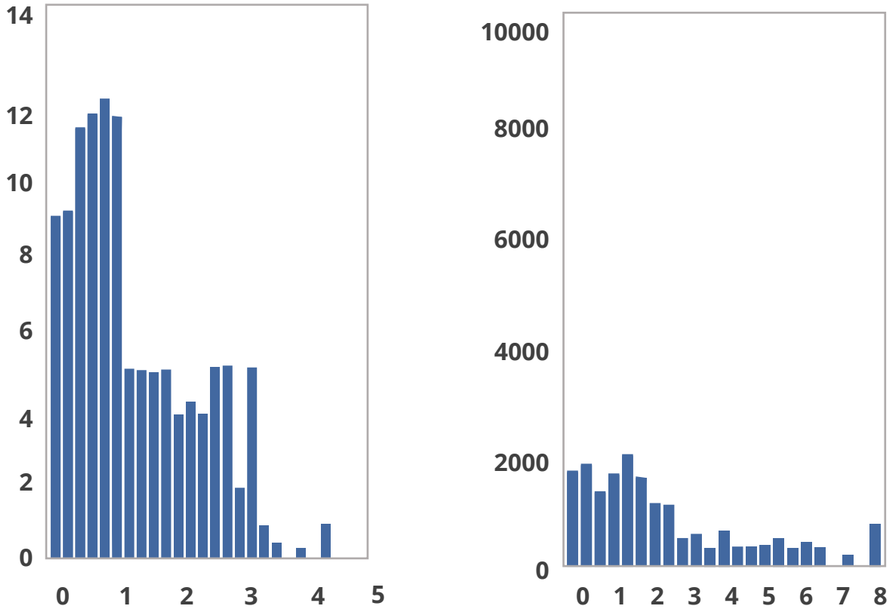


Figure (b) Dropout with p = 0.5

Source: https://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

Features of Dropout Layer

Effect of dropout rate

In Figure (a), the number of concealed units is fixed, but in figure (b), the number of hidden units is adjusted so that the estimated number of hidden units preserved after dropout remains constant.

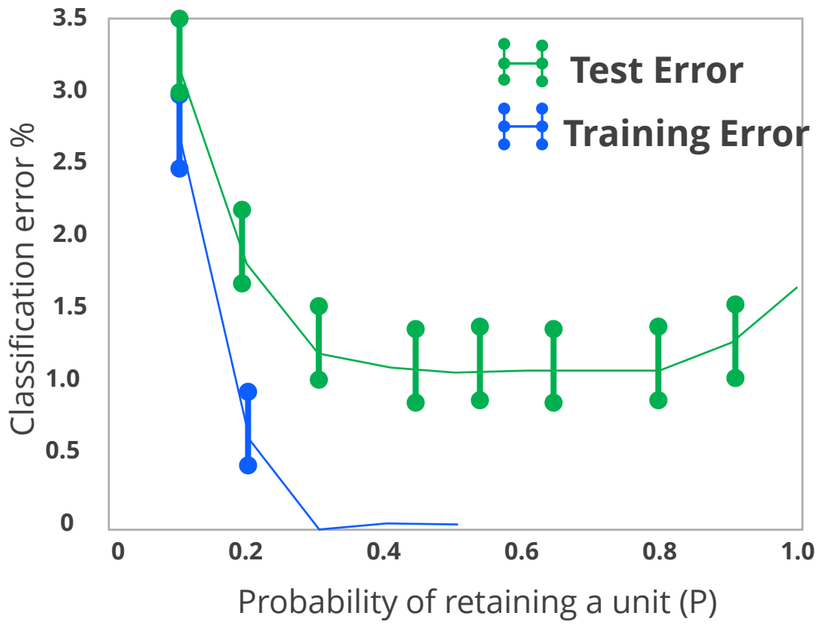


Figure (a) Keeping *n* fixed

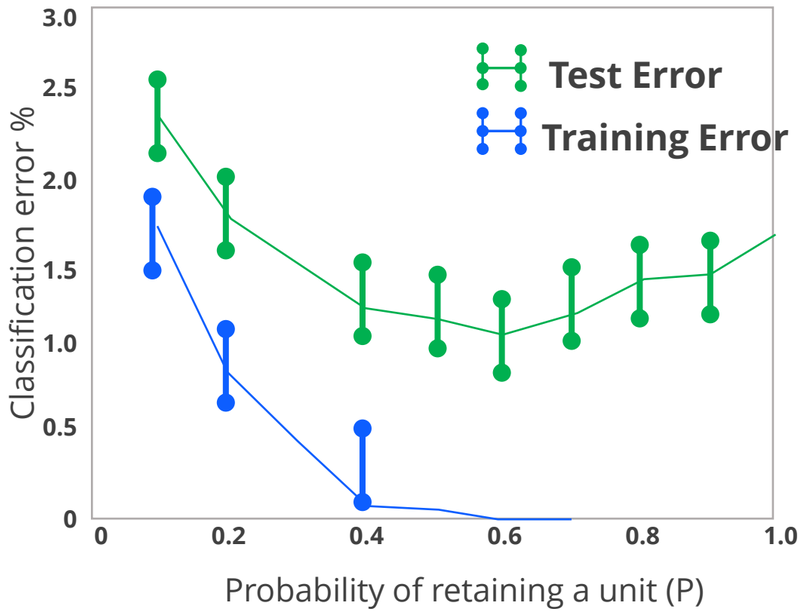


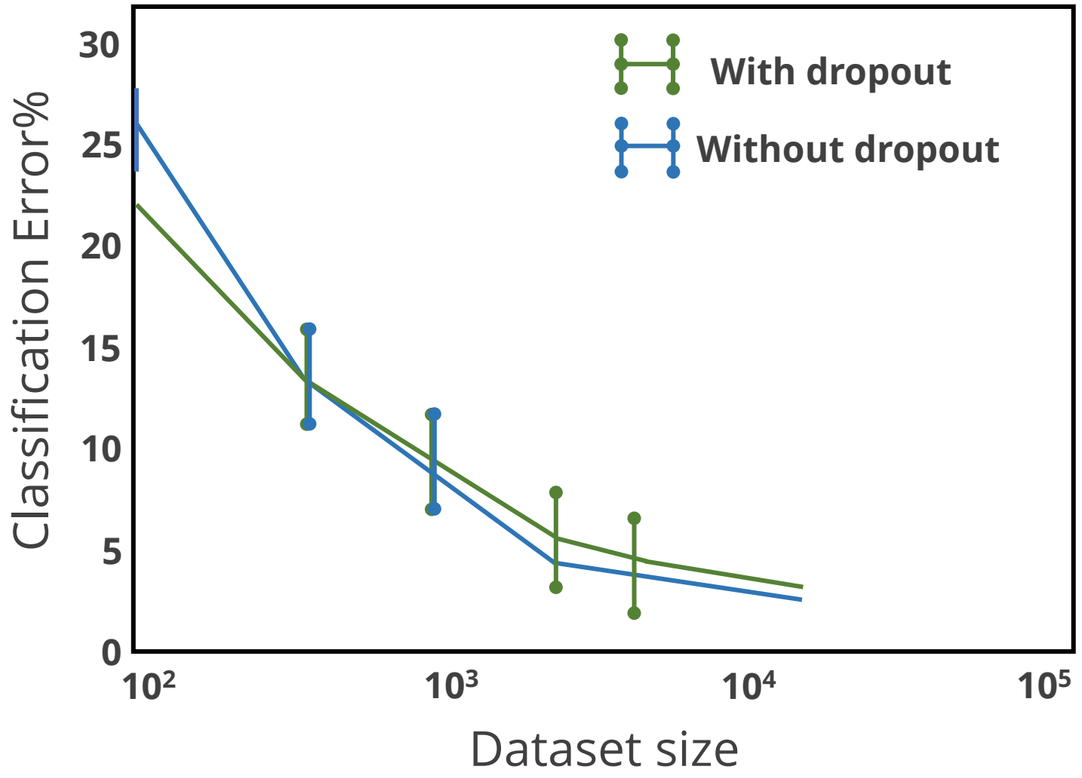
Figure (b) Keeping *pn* fixed

Source: https://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

Features of Dropout Layer

Effect of dataset size

The effect of changing the dataset size is apparent when dropout is used with feedforward networks. In the figure, the network was selected randomly from the MNIST training set of sizes 100, 500, 1K, 5K, 10K, and 50K.



Source: https://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

Dropout and Early Stopping

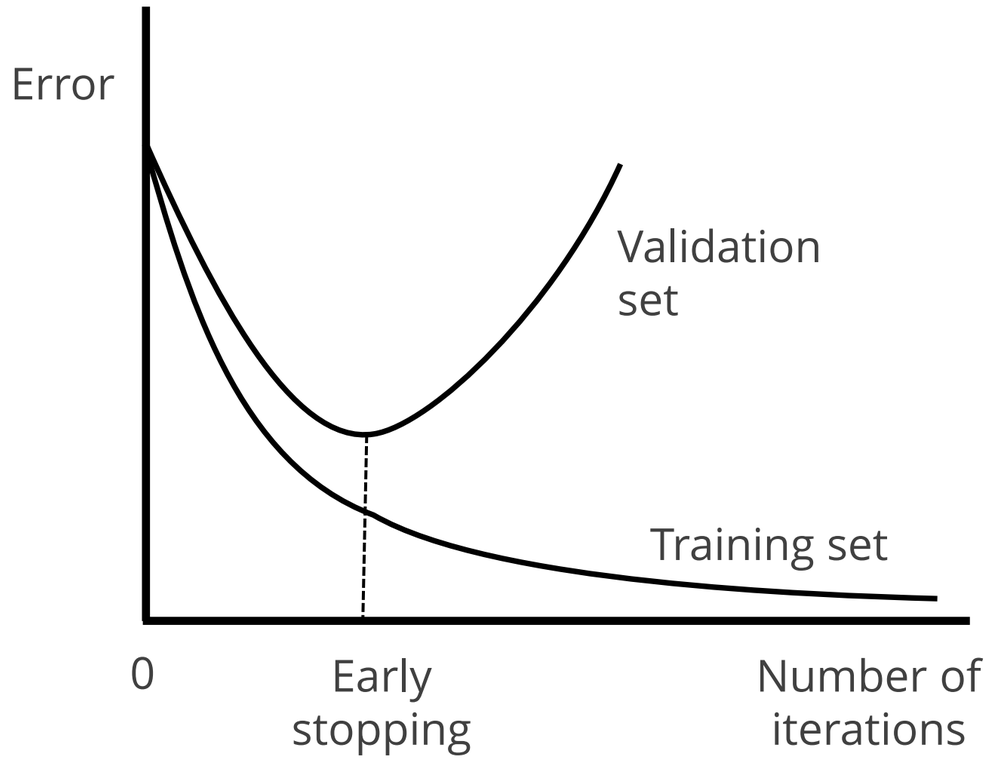
Early stopping is a regularization strategy that reduces overfitting.

Following is the difference between dropout and early stopping:

Dropout	Early stopping
• It randomly drops out nodes during training. • It improves generalization error in deep neural networks, is computationally inexpensive, and is effective.	• It allows one to specify an arbitrary number of training epochs. • The training stops once the model's performance on a holdout validation dataset stops improving.

Early Stopping

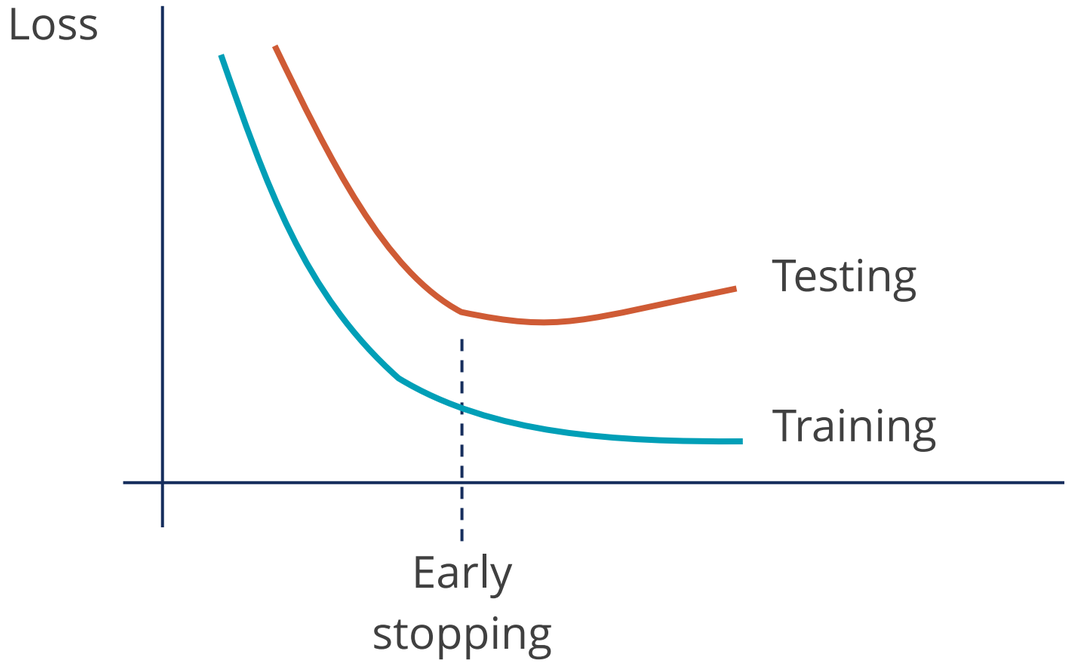
Early stopping is a regularization technique for deep neural networks that stops training when parameter updates no longer begin to yield improvements on a validation set.



By limiting the optimization approach to a smaller amount of parameter space, it functions as a regularized technique.

Early Stopping in Keras

Keras has a callback named EarlyStopping that stops training early.



It lets one define the performance metric to track and a trigger to end the training.

Early Stopping in Keras

Keras stops training when there's no more improvement in the chosen measure.

The first time with no improvement isn't always the best time to stop.

The model might not get better or might get worse before improving again.

Early Stopping in Keras

The patience argument in Keras adds a delay to the trigger for early stopping equal to the number of epochs where no improvement is desired.

This parameter allows the model more time to potentially recover and improve before stopping the training process.

The required patience value varies depending on the model type and difficulty.

Implementing Early Stopping Using Keras

Users may invoke various arguments to configure Earlystopping.

To stop training, use the **monitor** option to indicate the performance metric to track.

Syntax:-

```
callback = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=3)

model = tf.keras.Sequential([
 tf.keras.layers.Flatten(),
 tf.keras.layers.Dense(1024, activation=tf.nn.relu),
 tf.keras.layers.Dropout(0.2),
 tf.keras.layers.Dense(1024, activation=tf.nn.relu),
 tf.keras.layers.Dropout(0.2),
 tf.keras.layers.Dense(1024, activation=tf.nn.relu),
 tf.keras.layers.Dropout(0.2),
 tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer=tf.keras.optimizers.Adam(),
 loss='sparse_categorical_crossentropy',
 metrics=['accuracy'])

history = model.fit(x_train, y_train, epochs=50, validation_data=(x_test, y_test), callbacks=[callback])
```

Assisted Practices

Let’s understand the concept of Dropout using Jupyter Notebooks.

- 6.12_Implementation of Dropout

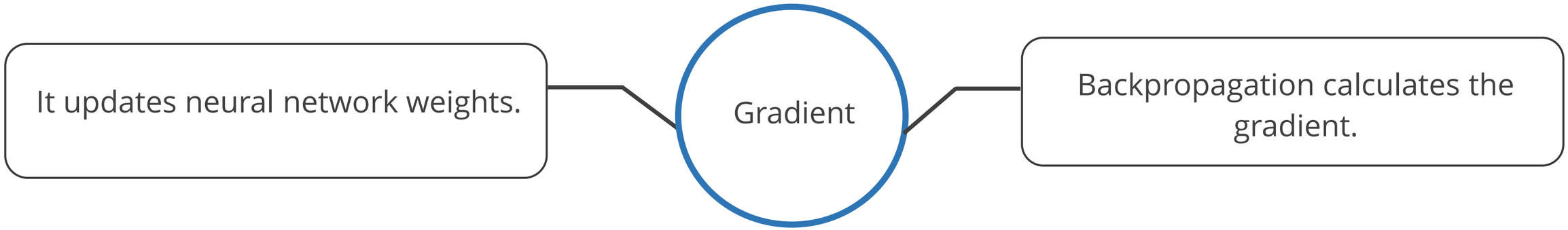
Note: Please refer to the Reference Material section to download the notebook files corresponding to each mentioned topic

ASSISTED PRACTICE

Vanishing Gradient

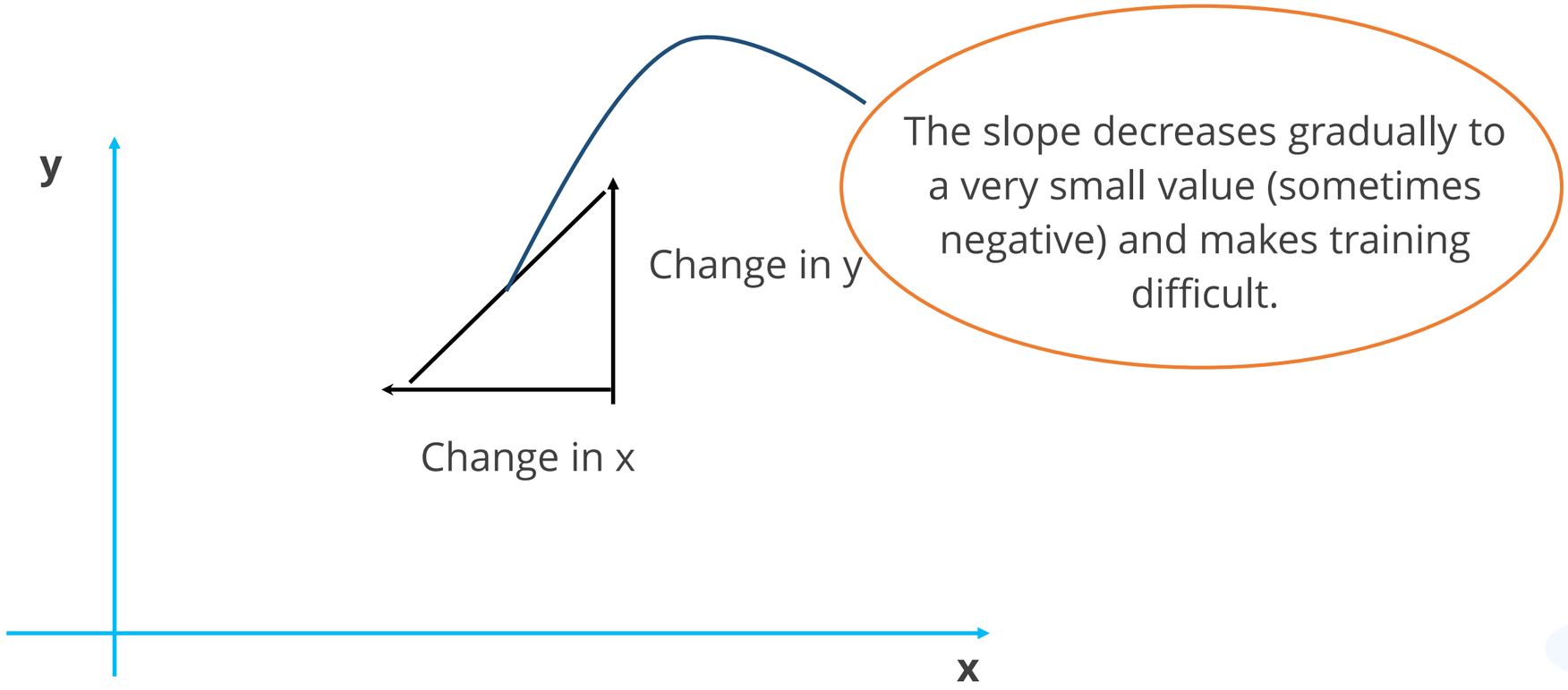
What Is Gradient?

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



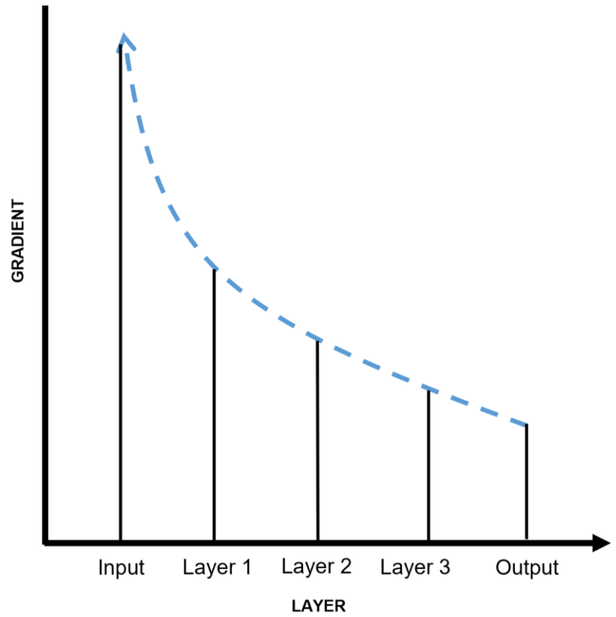
What Is Vanishing Gradient?

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



Vanishing Gradients

In vanishing gradients, the gradients become smaller as the backpropagation method progresses backward from the output layer to the input layer.



Finally, the lower layer weights remain unchanged, and the gradient descent never reaches the optima.

Vanishing Gradient Issue

The vanishing gradient issue refers to the challenge of updating the weights in the previous layers of a neural network during training.

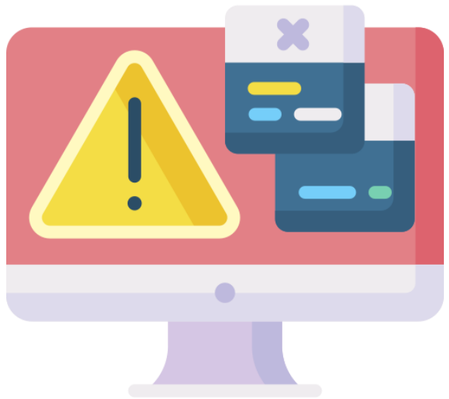
It occurs when the gradients become very small as they propagate backward, resulting in little or no update to the weights.

This problem is a difficulty with weights in the network's previous layers during training.

Therefore, the network may struggle to learn complex patterns and exhibit reduced performance.

Cause of Vanishing Gradient Issue

The gradient of loss for any given weight is the product of some derivatives dependent on the components further down the network.



An earlier weight in the network requires more terms in the product.

A tiny product is used to update the weight by multiplying it by the learning rate, a tiny integer ranging between .01 and .0001.

This small outcome is subtracted from the weight to obtain its updated value.

How to Prevent Vanishing Gradient?

- Residual Networks (ResNets) use shortcut connections to effectively address the vanishing gradient problem and enable training of very deep neural networks.
- GPUs play a crucial role in mitigating the vanishing gradient issue through parallel processing, faster training, and more frequent weight updates during backpropagation.
- Choosing the right activation function, like rectified linear (ReLU), helps prevent the vanishing gradient problem by avoiding input compression into small ranges.
- The switch from CPUs to GPUs has significantly improved the feasibility of standard backpropagation, even for low-cost models, due to their faster compilation times.

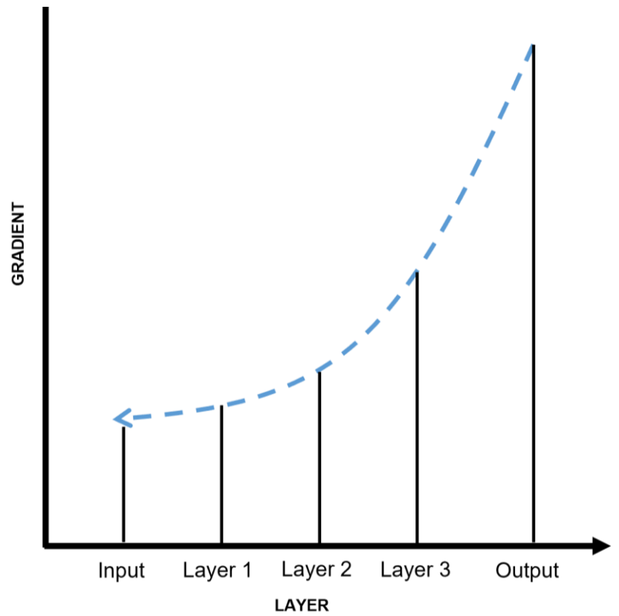
Exploding Gradient

What Is Exploding Gradient?

> The issue of exploding gradients arises when there is a significant accumulation of error gradients, leading to excessively large weight updates in a neural network during the training process.

Exploding Gradients

When the gradients grow larger with the advancement of the backpropagation method, massive weight updates occur, causing the gradient descent to diverge.



This is the exploding gradients problem.

How to Fix Exploding Gradients?

The vanishing gradient problem can be fixed by redesigning the neural network to have fewer layers and mini-batch sizes.

Using Long Short-Term Memory (LSTM) networks reduces exploding gradients.

Gradient clipping limits gradient size to effectively fix exploding gradients.

Hyperparameter Tuning

What Are Parameters and Hyperparameters?

Parameters

- They are found during model training.
- These internal variables are adjusted to make predictions based on input data.
- For example, in K-means clustering, the positions of the centroids are learned during training. These positions are the model's parameters.

Hyperparameters

- They are determined before training.
- These are configurations or settings that govern the learning process but aren't learned from the data.
- For example, the value of K in K-means clustering is decided before creating the model. This value, representing the number of clusters, is a hyperparameter.

Hyperparameters of Deep Learning Models

Learning rate	The most important hyperparameter that helps the model get an optimized result
Number of hidden units	A classic hyperparameter that specifies the representational capacity of a model
Convolutional kernel width	It determines the size of the filters used in a convolutional neural network, which influences the receptive field and the capacity of the model.
Mini-batch size	It affects the training process, training speed, and number of iterations in a deep learning model
Number of epochs	It is partly responsible for the weight optimization in a neural network

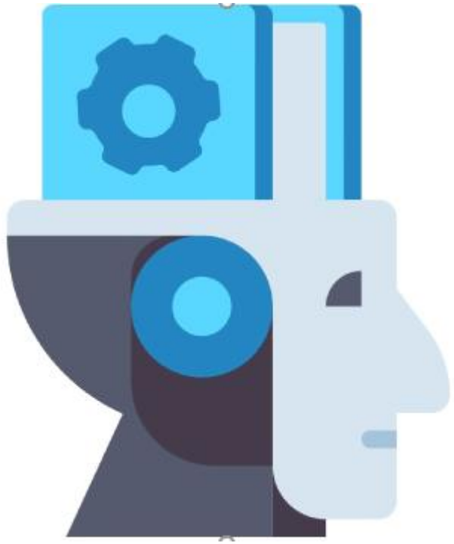
Hyperparameter Tuning

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

Hyperparameter Tuning

Hyperparameter tuning involves finding the optimal values for hyperparameters, which are manually set parameters in a machine-learning model.

The goal of iteratively adjusting hyperparameter values and evaluating model performance is to enhance the model's effectiveness and generalization.



Grid search, random search, and Bayesian optimization are popular hyperparameter tuning techniques.

Hyperparameter Tuning

Unlike parameters, hyperparameters are not estimated directly from training data.

Hyperparameters define the model's complexity and learning efficiency.

Set hyperparameters before training to control the model's behavior.

Tune hyperparameters correctly to improve performance and efficiency.

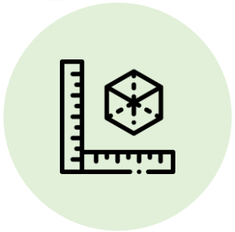
Hyperparameter Tuning

Some hyperparameters, like the momentum hyperparameter, have specific values that are typically used.

Generalizing the values of hyperparameters might not be practical for real-world data.

Therefore, finding the best set of hyperparameters is a quintessential search problem.

How to Tune the Hyperparameters?



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

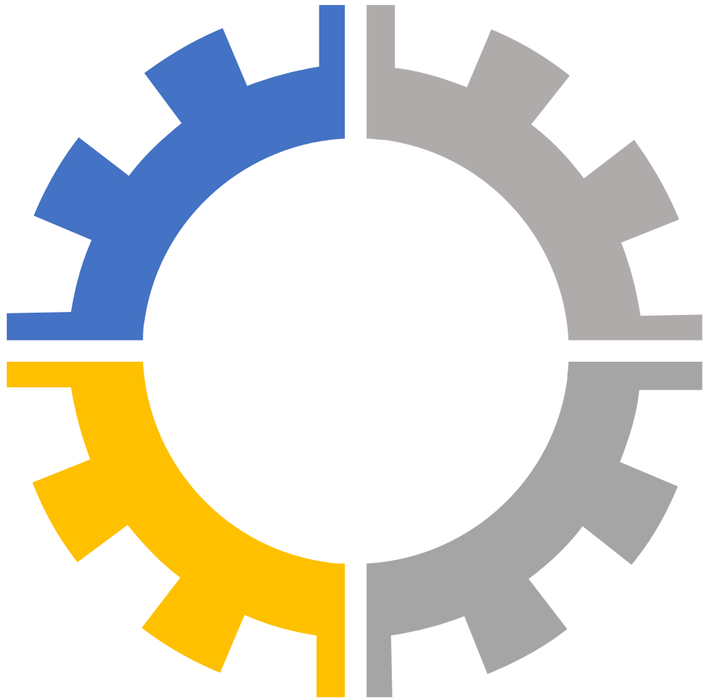
Understand the training process
Know the training process and how exactly it can be influenced.

Perform a grid search
Search over the defined range of hyperparameters using grid or random search methods.

Hyperparameter: Examples

Some examples of hyperparameters are:

Regularization factor in regression



Learning rates and momentum hyperparameters in gradient descent

The value of K in K-nearest neighbors and K-means clustering

Number of hidden layers in a neural network

Selection of Hyperparameters

There are two approaches to selecting hyperparameters:

Manual hyperparameter tuning

This involves manually selecting and tuning hyperparameters based on intuition, experience, and trial and error.

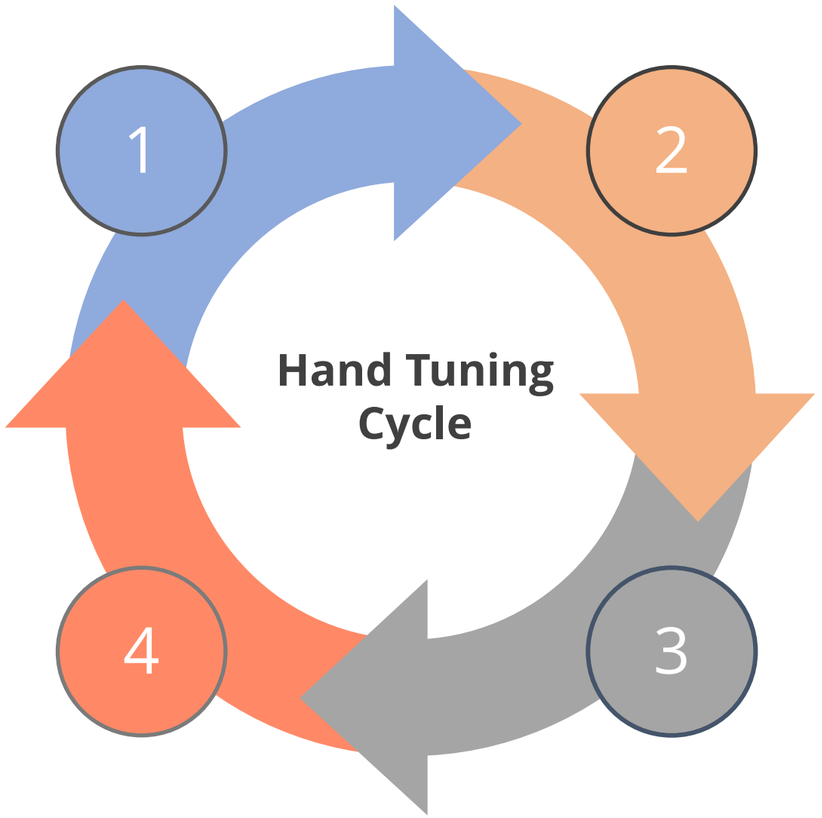
Automatic hyperparameter tuning

Use algorithms to systematically search the hyperparameter space to find the optimal values.

Manual Tuning Approach

The following example illustrates that manual tuning is not efficient, as even a fivefold increase in the neurons resulted in an improvement in accuracy of only 4%.

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



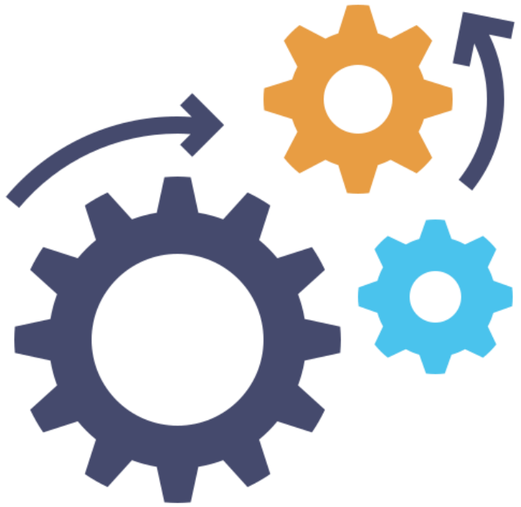
The number of neurons has been increased to 100 with one additional layer, and the resulting accuracy is 84%.

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

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

Automatic Hyperparameter Tuning

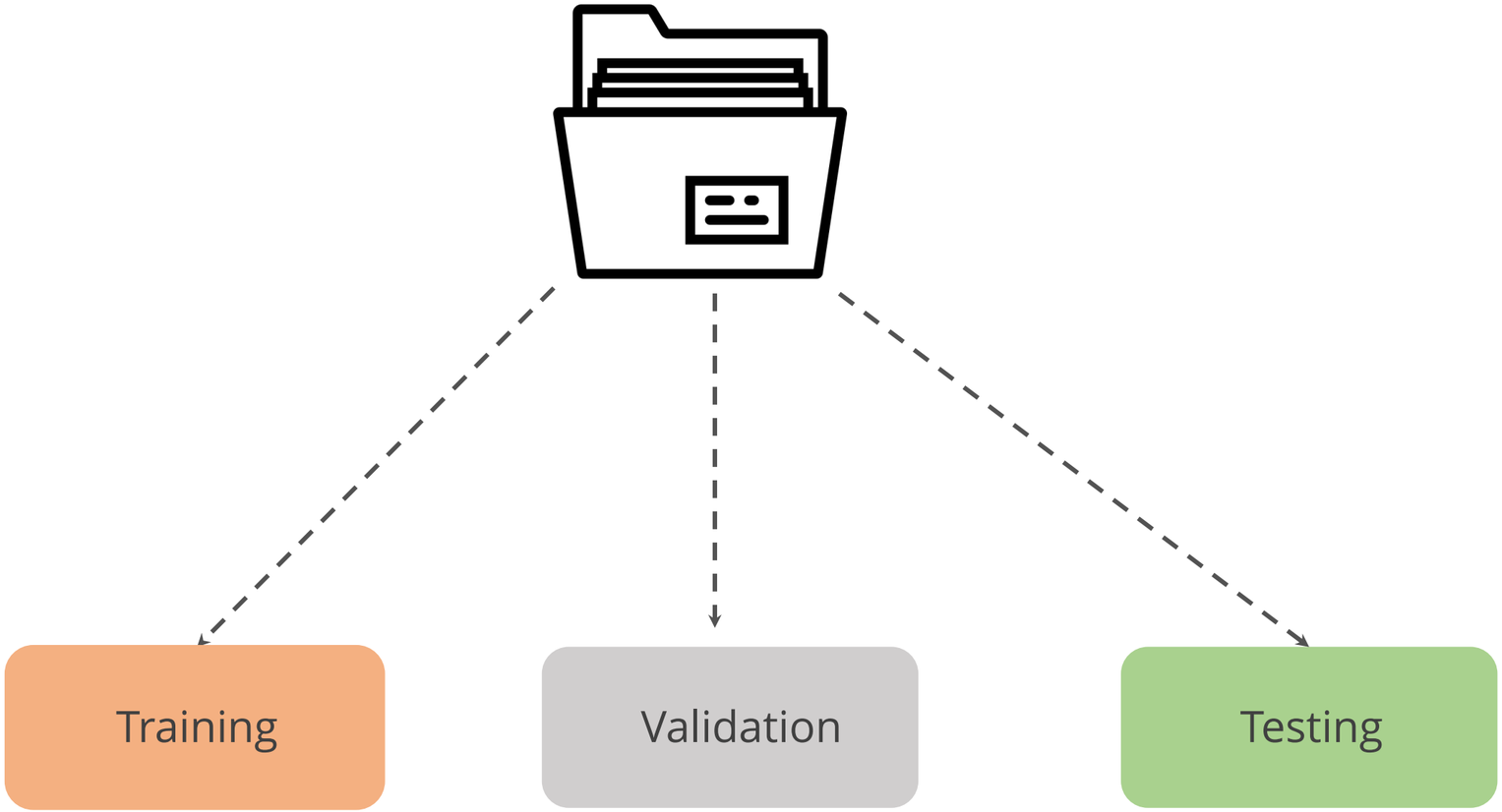
The automatic selection approach is preferred over the manual approach, as the latter is a very rigorous method. The automatic approach is the process of tuning the hyperparameters with the help of algorithms. They are as follows:



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

Data Partitioning for Hyperparameter Selection

There are three categories of available data for the purpose of selecting the hyperparameters.



Data Partitioning for Hyperparameter Selection

The training set is used to train the model with different hyperparameter combinations.

Training

Validation

Testing

Data Partitioning for Hyperparameter Selection

The validation set is used to identify which parameters minimize error.

Training

Validation

Testing

Data Partitioning for Hyperparameter Selection

The test set is used to assess performance with selected parameters.

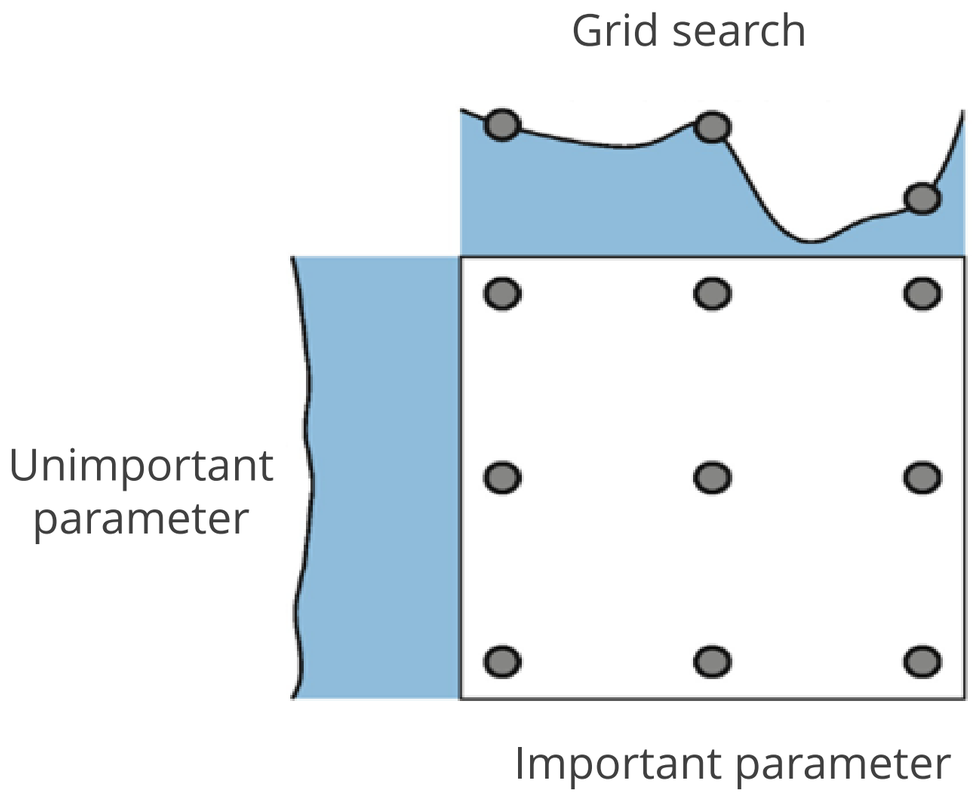
Training

Validation

Testing

Hyperparameter Tuning Techniques: Grid Search

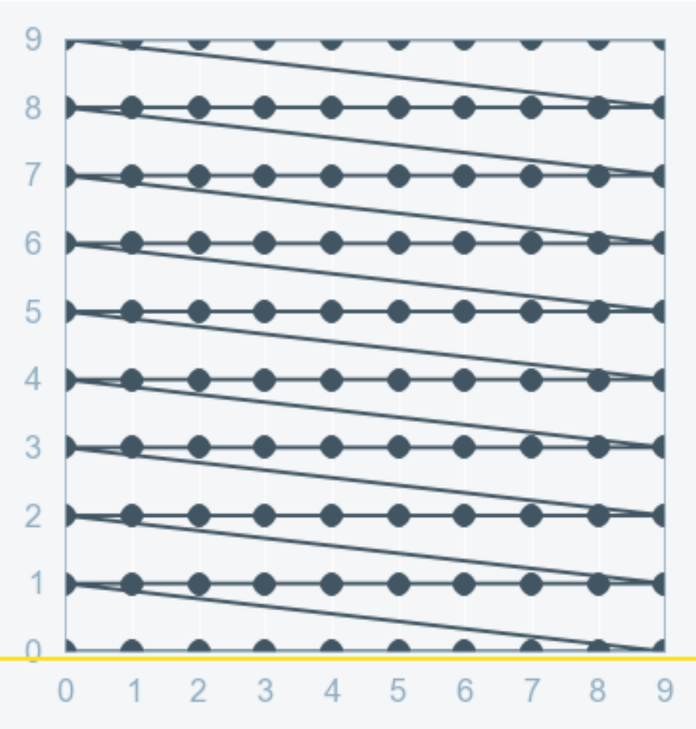
Iterating over given hyperparameters using cross-validation is called **Grid Search**.



Source: https://i.stack.imgur.com/u9BMB.png

Hyperparameter Tuning Techniques: Grid Search

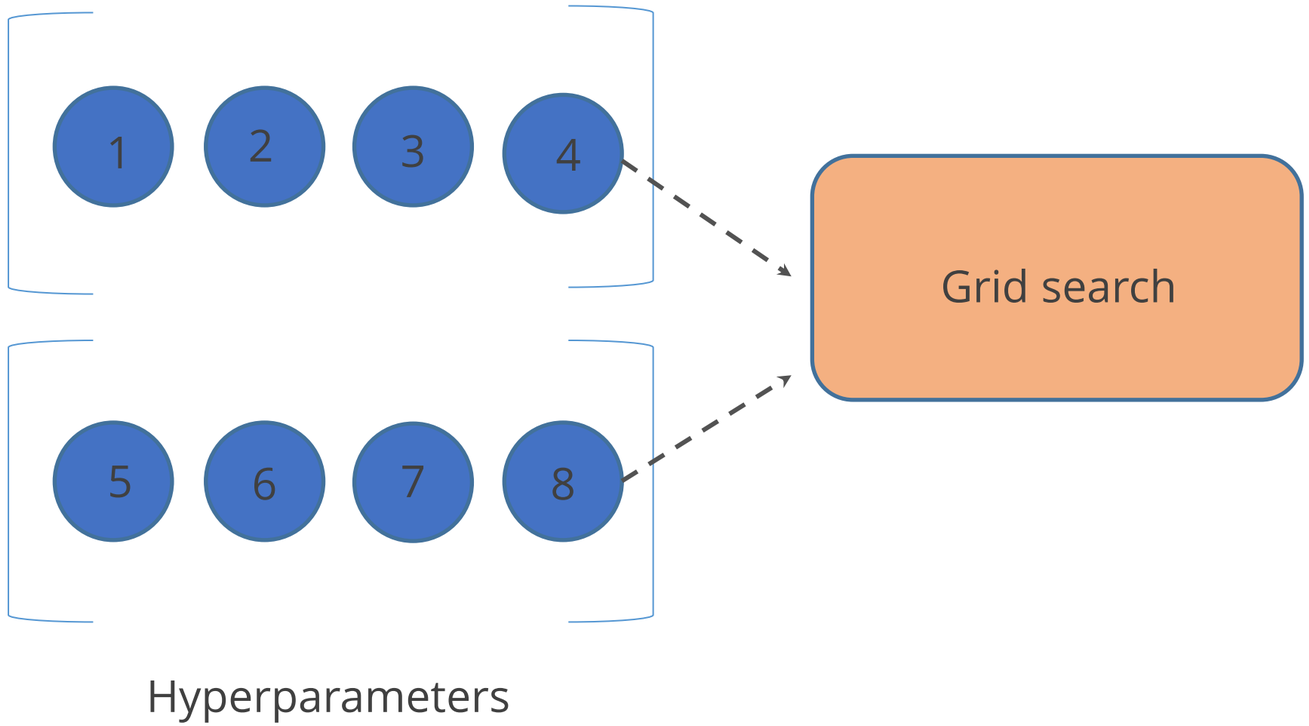
The process of grid search for hyperparameter tuning is as follows:



- **Parameter grid construction**: Arrange all potential hyperparameter combinations in a grid layout
- **Matrix conversion**: Represent each unique hyperparameter combination in a matrix for systematic processing
- **Performance evaluation**: Train and assess models for each distinct set of parameters, typically using a validation set
- **Best model identification**: Choose the model with the highest performance score (based on metrics like accuracy, precision, and various others) as the grid search's optimal outcome

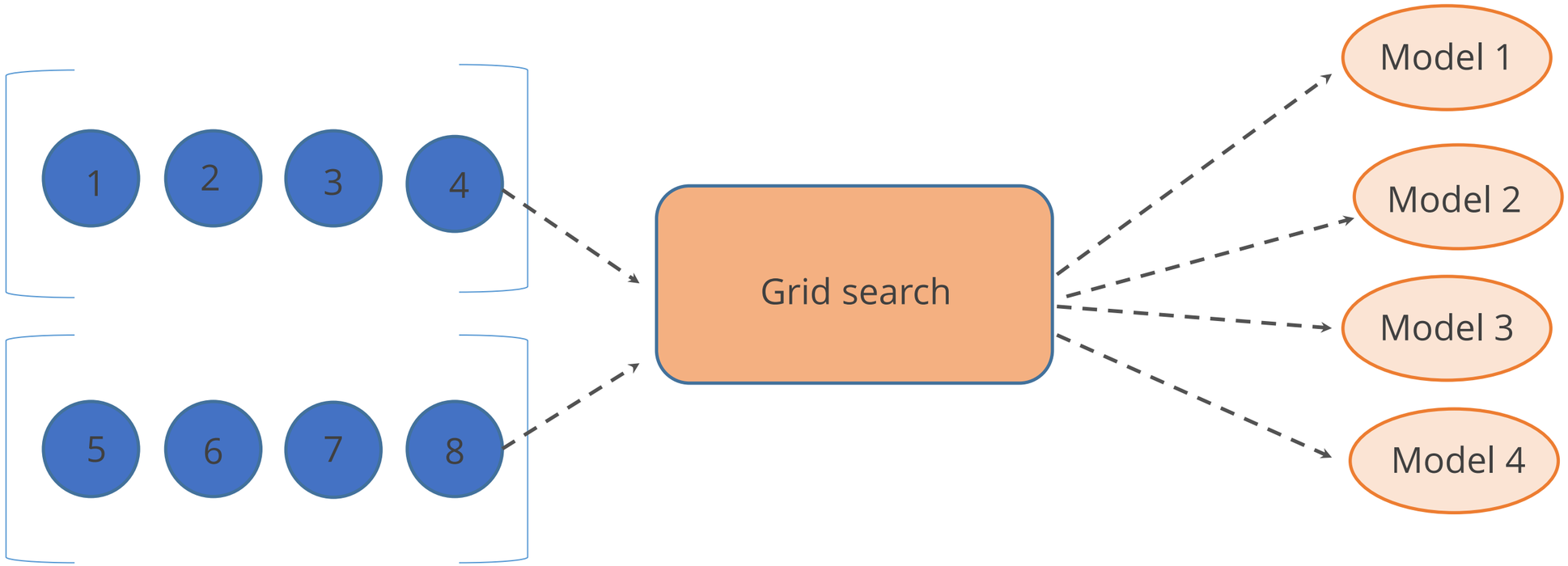
Hyperparameter Tuning Techniques: Grid Search

Of the eight given hyperparameters, grid search takes different hyperparameter values.



Hyperparameter Tuning Techniques: Grid Search

Next, it creates four models using the available hyperparameters.



Grid search selects the model with the lowest error as the most efficient and finalizes those hyperparameters.

Hyperparameter Tuning Techniques: GridSearchCV

The GridSearchCV algorithm extends grid search by adding cross-validation, evaluating all hyperparameter combinations with different data splits.

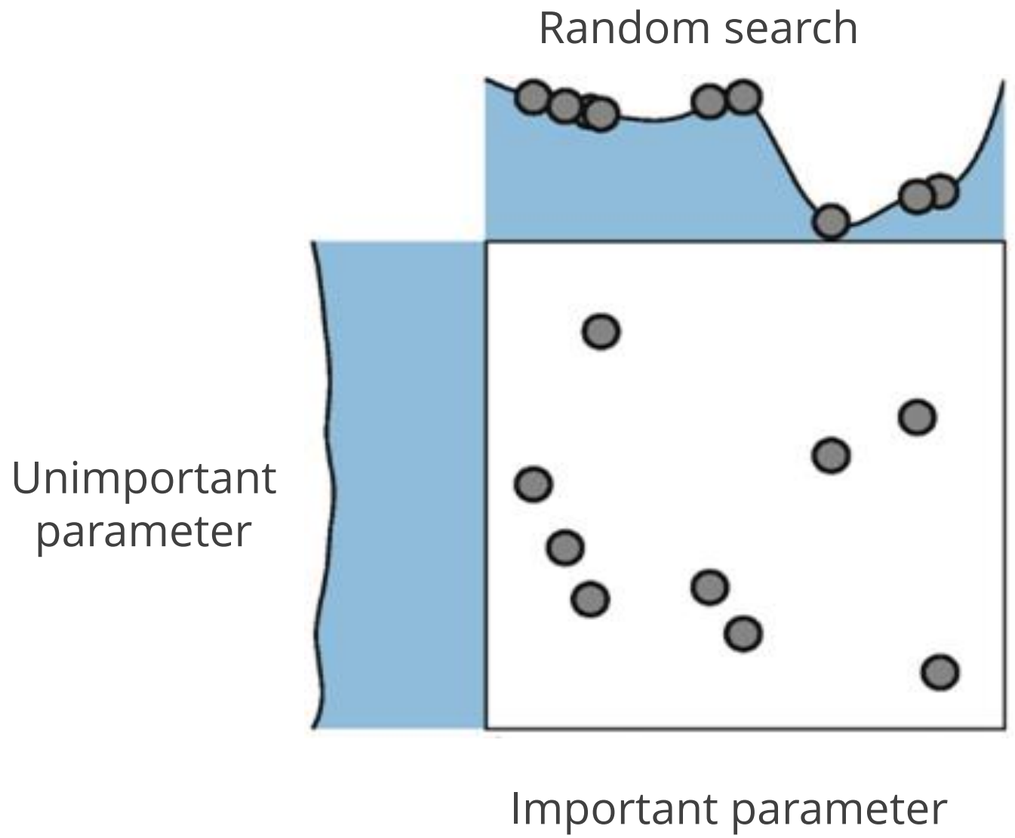
- Construct numerous versions of the machine learning model, exploring all possible combinations of specified hyperparameters.

- Incorporate cross-validation and fit the model across every possible set of hyperparameters, ensuring a more thorough evaluation.

This thorough approach offers better model validation but at the cost of higher computational effort.

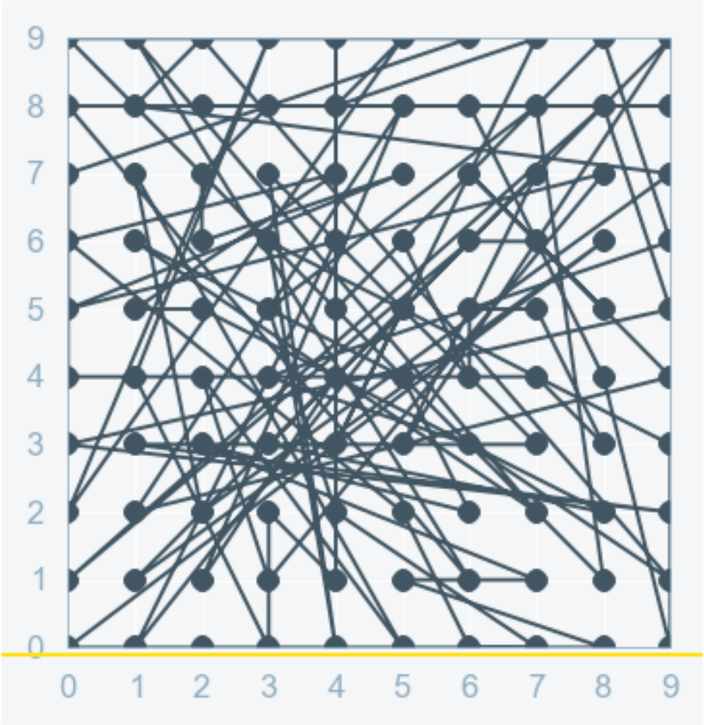
Hyperparameter Tuning Techniques: Random Search

Random search is commonly used as a hyperparameter tuning method for functions that are non-differentiable or discontinuous, including those with complex, nonlinear behavior.



Hyperparameter Tuning Techniques: Random Search

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



Hyperparameter Tuning Techniques: Random Search

The advantages of using random search for hyperparameter tuning are as follows:

Saves time

More efficient than manual or grid search

Has the drawback of producing high variance during the optimization process

Hyperparameter Tuning Techniques: RandomSearchCV

RandomSearchCV is an advanced version of random search, efficiently sampling hyperparameter combinations to reduce computational burden in machine learning tuning.

- It explores a set number of randomly chosen hyperparameter combinations, constructing multiple versions of the model while reducing the computational burden.
- Incorporate cross-validation to assess model performance, similar to GridSearchCV but with a more efficient search process.
- This balanced approach often leads to nearly as good results as GridSearchCV, but with significantly reduced computational effort

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.

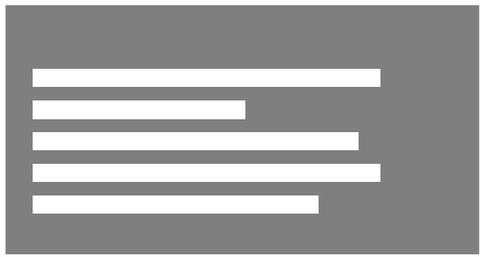
Evolutionary Optimization

Evolutionary algorithms mimic natural evolution to optimize solutions.

They use processes like selection, crossover, mutation, and replacement to explore and adapt solutions.

These algorithms can be applied to find optimal hyperparameters in various models.

They are especially useful in black box functions with noise, aiding in global optimization.



Evolutionary Optimization

Steps for 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 through 4 until the algorithm gives a constant good outcome

Bayesian Optimization

Uses a machine learning framework to predict optimal hyperparameters

Finds optimal hyperparameters from the results of previously built models with different hyperparameter configurations through the Gaussian process

Inherently studies the trend in each dataset, which is not possible for a human

Interpretability

What Is Interpretability?

> Interpretability is the degree of a 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 ensure it explains its decisions less like a machine so is easily trustable for humans

When Is Interpretability Not Needed?

Interpretability is not used in these situations:

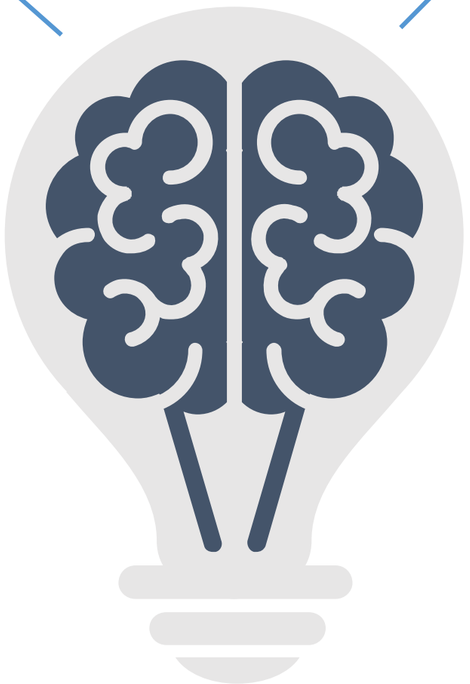
- For an insignificant model
- For a well-studied and researched problem
- For scenarios where people or the program might manipulate the model

Classification of Interpretability Methods

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

Intrinsic or Post-Hoc

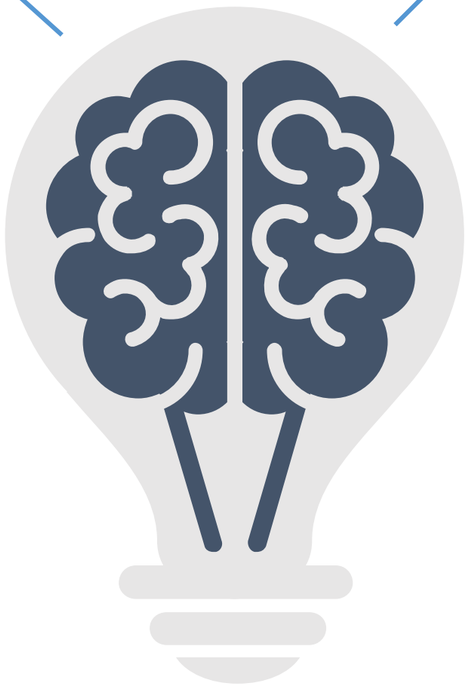
Achieves interpretability by simplifying the machine learning model and analyzes the method after training



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

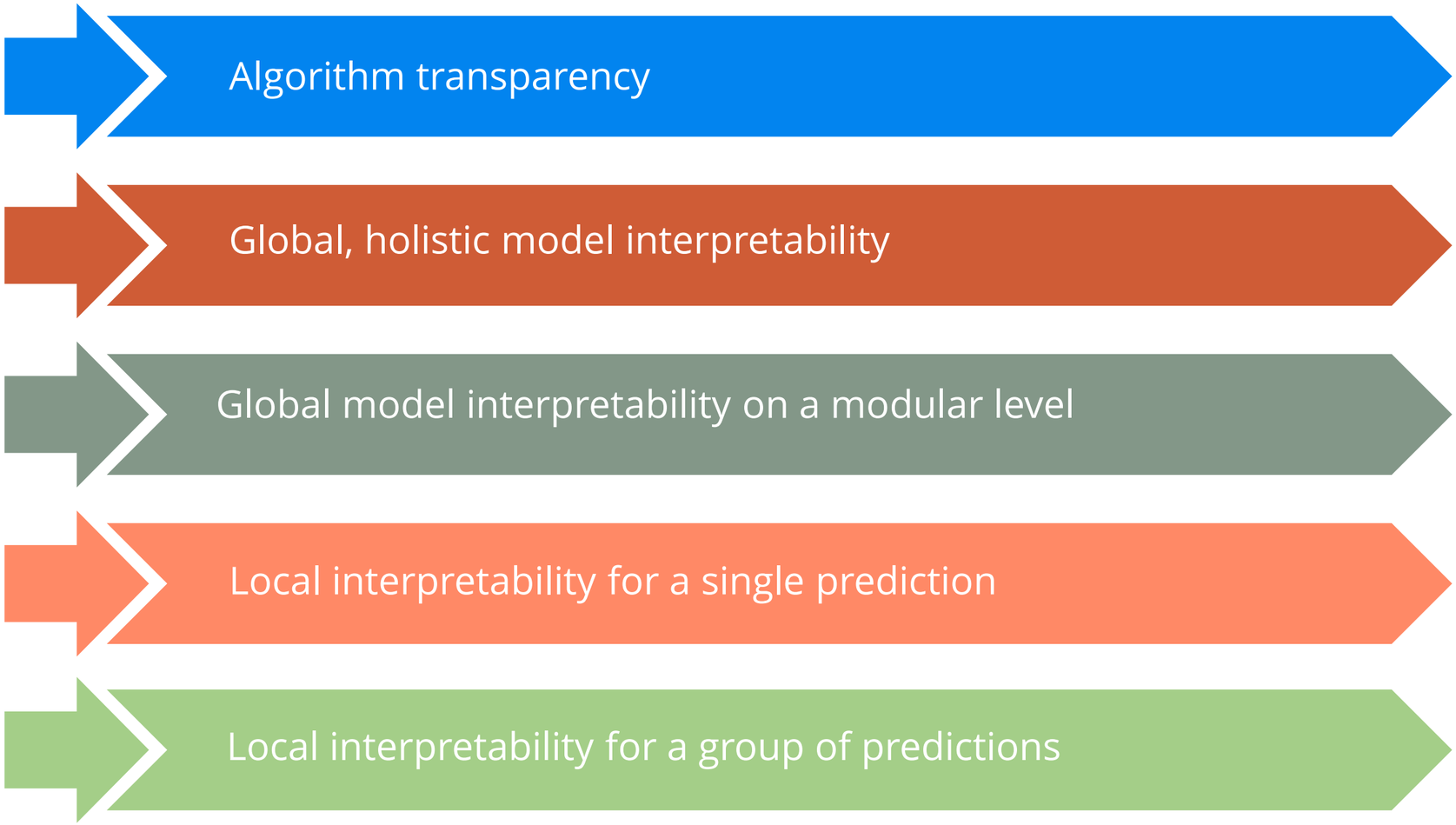
Model-Specific or Model-Agnostic

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

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

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

Global, Holistic Model Interpretability

Helps to understand the distribution of the 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 decisions with a holistic view of features

Global Model Interpretability on a Modular Level

Can be used when there is difficulty achieving global model interpretability

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

Local Interpretability for a Single Prediction

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

The accuracy of local interpretability is more important than the prediction of global interpretability

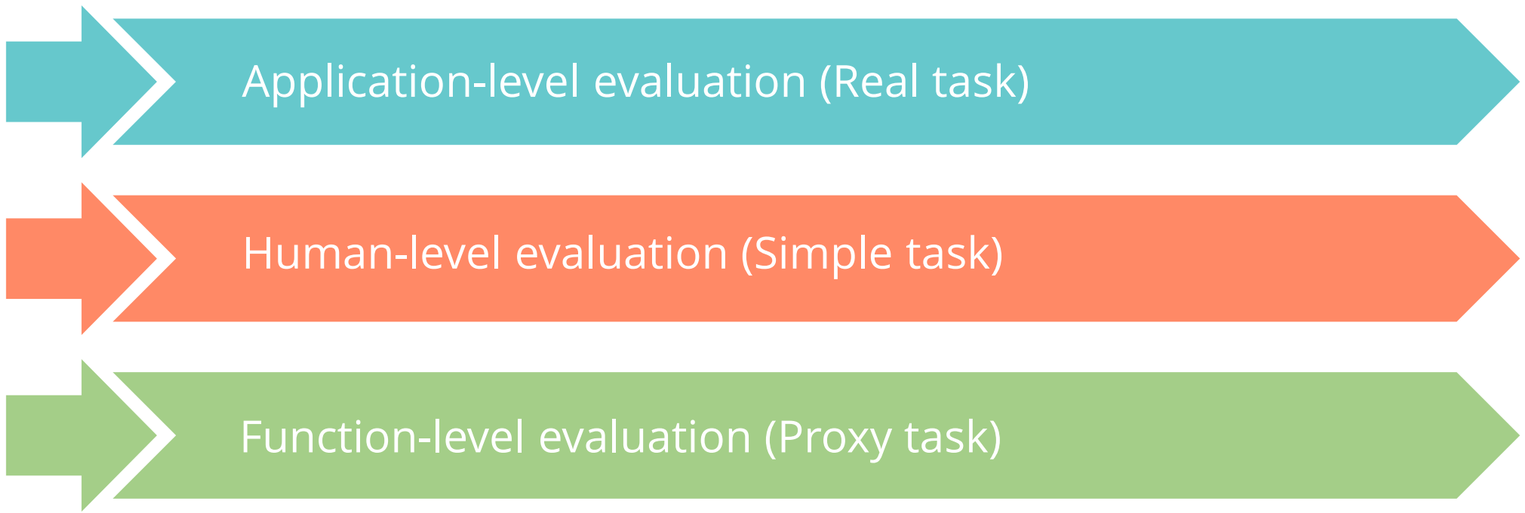
Local Interpretability

Applies global methods to 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)

Application-level or real task evaluation is the assessment of interpretability as an outcome 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 that work as explanation methods

Effectiveness of Explanation

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

Effectiveness of Explanation

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

Key Takeaways

- Optimization algorithms are used to change the attributes of the neural network to reduce losses.
- The standard gradient descent algorithm updates the parameters by evaluating the loss and gradient over the entire training dataset, leading to optimal solutions.
- Adaptive gradient (AdaGrad) optimization iteratively updates different learning rates for each parameter without manual tuning.
- Adadelta alters the custom step size calculation and removes the need for an initial learning rate hyperparameter.

Key Takeaways

- Batch normalization is used to normalize output data from the model's activation functions for particular layers.
- Dropout and early stopping are the regularization strategies that reduce overfitting.
- Interpretability is the degree of a human’s ability to predict the model’s result consistently.

Knowledge Check

Knowledge Check

1

What are optimizers in deep learning?

A. Algorithms or methods used to change the attributes of the neural network, such as weights and learning rate, to reduce losses.

B. Algorithms or methods used to generate synthetic data for training.

C. Algorithms or methods used to evaluate the accuracy of the model.

D. Algorithms or methods used to initialize the weights of the model.

Knowledge Check

1

What are optimizers in deep learning?

A. Algorithms or methods used to change the attributes of the neural network, such as weights and learning rate, to reduce losses.

B. Algorithms or methods used to generate synthetic data for training.

C. Algorithms or methods used to evaluate the accuracy of the model.

D. Algorithms or methods used to initialize the weights of the model.

The correct answer is **A**

Optimizers are algorithms or methods used to change the attributes of the neural network, such as weights and learning rate, to reduce losses.

Knowledge Check

2

What is the role of the loss function in optimization?

A. It determines the size of the model.

B. It serves as a roadmap for the optimizer to indicate if it has taken the correct or the wrong path.

C. It sets the learning rate of the optimizer.

D. It determines the number of epochs needed to train the model.

Knowledge Check 2

What is the role of the loss function in optimization?

A. It determines the size of the model.

B. It serves as a roadmap for the optimizer to indicate if it has taken the correct or the wrong path.

C. It sets the learning rate of the optimizer.

D. It determines the number of epochs needed to train the model.

The correct answer is **B**

The loss function serves as a roadmap for the optimizer to indicate if it has taken the correct or the wrong path.

Knowledge Check

3

What are hyperparameters in a machine learning model?

A. The parameters that are learned by the model during training

B. The parameters that are used to evaluate the model after training

C. The parameters that are set manually before training the model

D. The parameters that are used to test the model's accuracy

Knowledge Check

3

What are hyperparameters in a machine learning model?

A. The parameters that are learned by the model during training

B. The parameters that are used to evaluate the model after training

C. The parameters that are set manually before training the model

D. The parameters that are used to test the model's accuracy

The correct answer is **C**

Hyperparameters are the parameters that are set manually before training the model.

Lesson-End Project: Hyperparameter Tuning with MNIST Dataset

Problem statement: 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 the MNIST dataset.

Objective:

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

Access: Click on the **Lab** tab on the left side of the LMS panel. Copy the generated username and password. Click on the **Launch Lab** button. On the new page, enter the username and password you copied earlier into the respective fields. Click **Login** to start your lab session. A full-fledged Jupyter lab opens, which you can use for your hands-on practice and projects.

LESSON-END PROJECT

Thank You!