## LEARNING PYTORCH:

1. Learn about Pytorch. Install Pytorch. Uses of tools like Jupyter Notebook, Google Colab.

- PyTorch is an open source machine learning (ML) framework based on the Python programming language and the Torch library. Torch is an open source ML library used for creating deep neural networks and is written in the Lua scripting language. It's one of the preferred platforms for deep learning research. The framework is built to speed up the process between research prototyping and deployment.
- The PyTorch framework supports over 200 different mathematical operations. PyTorch's popularity continues to rise, as it simplifies the creation of artificial neural network models. PyTorch is mainly used by data scientists for research and artificial intelligence (AI) applications. PyTorch is released under a modified BSD license.

PyTorch provides the following key **features**:

- **Tensor computation:** Similar to NumPy array -- an open source library of Python that adds support for large, multidimensional arrays -- tensors are generic n-dimensional arrays used for arbitrary numeric computation and are accelerated by graphics processing units. These multidimensional structures can be operated on and manipulated with application program interfaces (APIs).
- **TorchScript:** This is the production environment of PyTorch that enables users to seamlessly transition between modes. TorchScript optimizes functionality, speed, ease of use and flexibility.
- **Dynamic graph computation:** This feature lets users change network behavior on the fly, rather than waiting for all the code to be executed.
- **Automatic differentiation**: This technique is used for creating and training neural networks. It numerically computes the derivative of a function by making backward passes in neural networks.
- **Python support:** Because PyTorch is based on Python, it can be used with popular libraries and packages such as NumPy, SciPy, Numba and Cynthon.
- **Variable:** The variable is enclosed outside the tensor to hold the gradient. It represents a node in a computational graph.

- **Parameter:** Parameters are wrapped around a variable. They're used when a parameter needs to be used as a tensor, which isn't possible when using a variable.
- **Module:** Modules represent neural networks and are the building blocks of stateful computation. A module can contain other modules and parameters.
- **Functions:** These are the relationships between two variables. Functions don't have memory to store any state or buffer and have no memory of their own.

## PyTorch benefits:

Using PyTorch can provide the following benefits:

- Offers developers an **easy-to-learn**, simple-to-code structure that's based on Python.
- Enables **easy debugging** with popular Python tools.
- Offers scalability and is well-supported on major cloud platforms.
- Provides a small community focused on open source.
- Exports learning models to the Open Neural Network Exchange (ONNX) standard format.
- **Offers a user-friendly interface.**
- Provides a C++ front-end interface option.
- Includes a rich set of **powerful APIs** that extend the PyTorch library.

### PyTorch vs. TensorFlow:

- PyTorch is often compared to TensorFlow, a deep machine learning framework developed by Google. Because TensorFlow has been around longer, it has a larger community of developers and more documentation.

- However, PyTorch does have advantages over TensorFlow. PyTorch dynamically defines computational graphs, unlike the static approach of TensorFlow. Dynamic graphs can be manipulated in real time. Additionally, TensorFlow has a steeper learning curve, as PyTorch is based on intuitive Python.

- TensorFlow may be better suited for projects that require production models and scalability, as it was created with the intention of being production ready. However, PyTorch is easier and lighter to work with, making it a good option for creating prototypes quickly and conducting research.

| Feature | Jupyter Notebook | Google Colab |
|---|---|---|
| Environment | Runs locally on a user's machine | Cloud-based environment accessible from any device |
| Collaboration | Limited | Real-time collaboration with multiple users on a single notebook |
| Computing Power | Depends on the user's local machine | Access to high-end CPUs, GPUs, and TPUs through Google Cloud |
| Pre-installed Libraries | Some pre-installed libraries with Anaconda distribution | Many popular libraries pre-installed |
| Integration with External Services | Possible through external packages | Built-in integration with Google Drive, Google Sheets, and GitHub |
| Cost | Free and open-source | Free with limitations, paid version available |

- Jupyter Notebook and Google Colab are both great tools for data science and machine learning.
- Jupyter Notebook is best suited for those who need full control over their environment and who want to work locally.
- Google Colab, on the other hand, is best suited for those who need to collaborate with others in real-time or who need access to a GPU or TPU.

2. Vector, Matrix, Tensor, Gradient descent, Numpy, Torch, TorchVision.

**Vectors:**

- Vectors are a **one-dimensional tensor**, and to manipulate them several operations available. Vector operations are of different types such as

mathematical operation, dot product, and linspace. Vectors play a vital role in deep learning.

- In deep learning neural network, we generate random point with the help of vectors or one-dimensional tensor.

**Matrix:**

Matrices and vectors are special cases of torch.Tensors, where their dimension is 2 and 1 respectively.

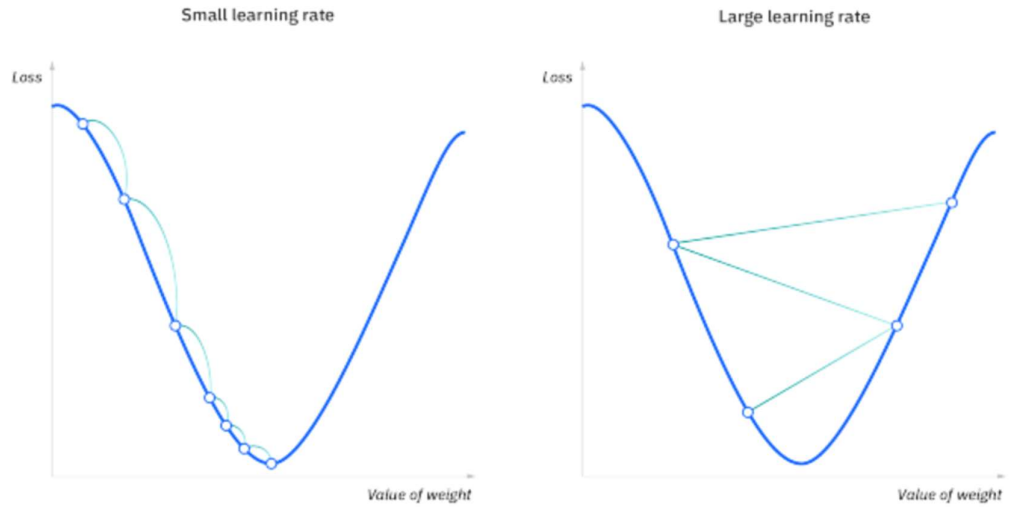
**Tensors:**

Tensors are a specialized data structure that are very similar to arrays and matrices. In PyTorch, we use tensors to encode the inputs and outputs of a model, as well as the model's parameters.

**Gradient descent:**

- Gradient descent is an optimization algorithm which is commonly-used to train machine learning models and neural networks. Training data helps these models learn over time, and the cost function within gradient descent specifically acts as a barometer, gauging its accuracy with each iteration of parameter updates. Until the function is close to or equal to zero, the model will continue to adjust its parameters to yield the smallest possible error. Once machine learning models are optimized for accuracy, they can be powerful tools for artificial intelligence (AI) and computer science applications.
- The goal of gradient descent is to minimize the cost function, or the error between predicted and actual y. In order to do this, it requires two data points—a direction and a learning rate. These factors determine the partial derivative calculations of future iterations, allowing it to gradually arrive at the local or global minimum (i.e. point of convergence).
- Learning rate (also referred to as step size or the alpha) is the size of the steps that are taken to reach the minimum. This is typically a small value, and it is evaluated and updated based on the behavior of the cost function. High learning rates result in larger steps but risks overshooting the minimum. Conversely, a low learning rate has small step sizes. While it has the advantage of more precision, the number of iterations compromises overall efficiency as this takes more time and computations to reach the minimum.

- The cost (or loss) function measures the difference, or error, between actual y and predicted y at its current position. This improves the machine learning model's efficacy by providing feedback to the model so that it can adjust the parameters to minimize the error and find the local or global minimum. It continuously iterates, moving along the direction of steepest descent (or the negative gradient) until the cost function is close to or at zero. At this point, the model will stop learning. Additionally, while the terms, cost function and loss function, are considered synonymous, there is a slight difference between them. It's worth noting that a loss function refers to the error of one training example, while a cost function calculates the average error across an entire training set.




## Types of gradient descent

There are three types of gradient descent learning algorithms: batch gradient descent, stochastic gradient descent and mini-batch gradient descent.

## Batch gradient descent

Batch gradient descent sums the error for each point in a training set, updating the model only after all training examples have been evaluated. This process referred to as a training epoch.

While this batching provides computation efficiency, it can still have a long processing time for large training datasets as it still needs to store all of the data into memory. Batch gradient descent also usually produces a stable error

gradient and convergence, but sometimes that convergence point isn't the most ideal, finding the local minimum versus the global one.

**Stochastic gradient descent**

Stochastic gradient descent (SGD) runs a training epoch for each example within the dataset and it updates each training example's parameters one at a time. Since you only need to hold one training example, they are easier to store in memory. While these frequent updates can offer more detail and speed, it can result in losses in computational efficiency when compared to batch gradient descent. Its frequent updates can result in noisy gradients, but this can also be helpful in escaping the local minimum and finding the global one.

**Mini-batch gradient descent**

Mini-batch gradient descent combines concepts from both batch gradient descent and stochastic gradient descent. It splits the training dataset into small batch sizes and performs updates on each of those batches. This approach strikes a balance between the computational efficiency of batch gradient descent and the speed of stochastic gradient descent

**Challenges with gradient descent**

While gradient descent is the most common approach for optimization problems, it does come with its own set of challenges. Some of them include:

**Local minima and saddle points**

For convex problems, gradient descent can find the global minimum with ease, but as nonconvex problems emerge, gradient descent can struggle to find the global minimum, where the model achieves the best results.

Recall that when the slope of the cost function is at or close to zero, the model stops learning. A few scenarios beyond the global minimum can also yield this slope, which are local minima and saddle points. Local minima mimic the shape of a global minimum, where the slope of the cost function increases on either

side of the current point. However, with saddle points, the negative gradient only exists on one side of the point, reaching a local maximum on one side and a local minimum on the other. Its name inspired by that of a horse's saddle.

Noisy gradients can help the gradient escape local minimums and saddle points.

**Vanishing and Exploding Gradients**

In deeper neural networks, particular recurrent neural networks, we can also encounter two other problems when the model is trained with gradient descent and backpropagation.

**Vanishing gradients**: This occurs when the gradient is too small. As we move backwards during backpropagation, the gradient continues to become smaller, causing the earlier layers in the network to learn more slowly than later layers. When this happens, the weight parameters update until they become insignificant—i.e. 0—resulting in an algorithm that is no longer learning.

**Exploding gradients**: This happens when the gradient is too large, creating an unstable model. In this case, the model weights will grow too large, and they will eventually be represented as NaN. One solution to this issue is to leverage a dimensionality reduction technique, which can help to minimize complexity within the model.

**Numpy:**

NumPy is a Python library used for working with arrays.

It also has functions for working in domain of linear algebra, fourier transform and matrices.

NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.

**Torch:**

Torch is an open source ML library used for creating deep neural networks and is written in the Lua scripting language. It's one of the preferred platforms for

deep learning research. The framework is built to speed up the process between research prototyping and deployment.

**TorchVision:**

The torchvision package consists of popular datasets, model architectures, and common image transformations for computer vision.

## 3. Process of vectorization

Vectorization is the process of converting textual data into numerical vectors and is a process that is usually applied once the text is cleaned. It can help improve the execution speed and reduce the training time of your code

**Vectorization techniques**

There are three major methods for performing vectorization on text data:

1. CountVectorizer

2. TF-IDF

3. Word2Vec

### 1. CountVectorizer

CountVectorizer is one of the simplest techniques that is used for converting text into vectors. It starts by tokenizing the document into a list of tokens (words). It selects the unique tokens from the token list and creates a vocabulary of words. Finally, a sparse matrix is created containing the frequency of words, where each row represents different sentences and each column represents unique words.

### 2.TF-IDF:

TF-IDF or Term Frequency–Inverse Document Frequency, is a statistical measure that tells how relevant a word is to a document. It combines two metrics — term frequency and inverse document frequency — to produce a relevance score.

### 3.Word2Vec

Word2Vec is a word embedding technique that makes use of neural networks to convert words into corresponding vectors in a way that semantically similar vectors are close to each other in N-dimensional space, where N refers to the dimensions of the vector. This technique was first implemented by Tomas Mikolov at Google back in 2013.

## 4. Linear regression model.

A linear regression model describes the relationship between a dependent variable, y, and one or more independent variables, X. The dependent variable is also called the response variable. Independent variables are also called explanatory or predictor variables. Continuous predictor variables are also called covariates, and categorical predictor variables are also called factors. The matrix X of observations on predictor variables is usually called the design matrix.

## 5. Dataset, Weight, Bias. Parameters Vs HyperParameters.

**Dataset:**

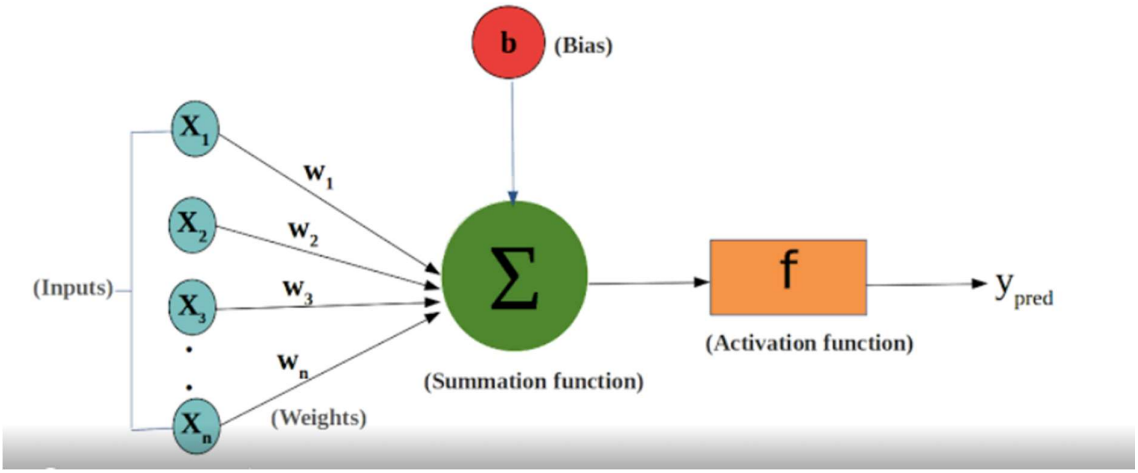
A Dataset is a set or collection of data. This set is normally presented in a tabular pattern. Every column describes a particular variable. And each row corresponds to a given member of the data set, as per the given question.

**Weight and bias:**

Weights and biases are parameters in neural networks that help identify machine learning data. They are also known as learnable parameters in some machine learning models.

Weights and biases are two of the most important concepts in a neural network. They determine how a neural network moves data through the network, which is called forward propagation.

Weights are real values associated with each input. They represent the strength of the connection between the input neuron and the output neuron.



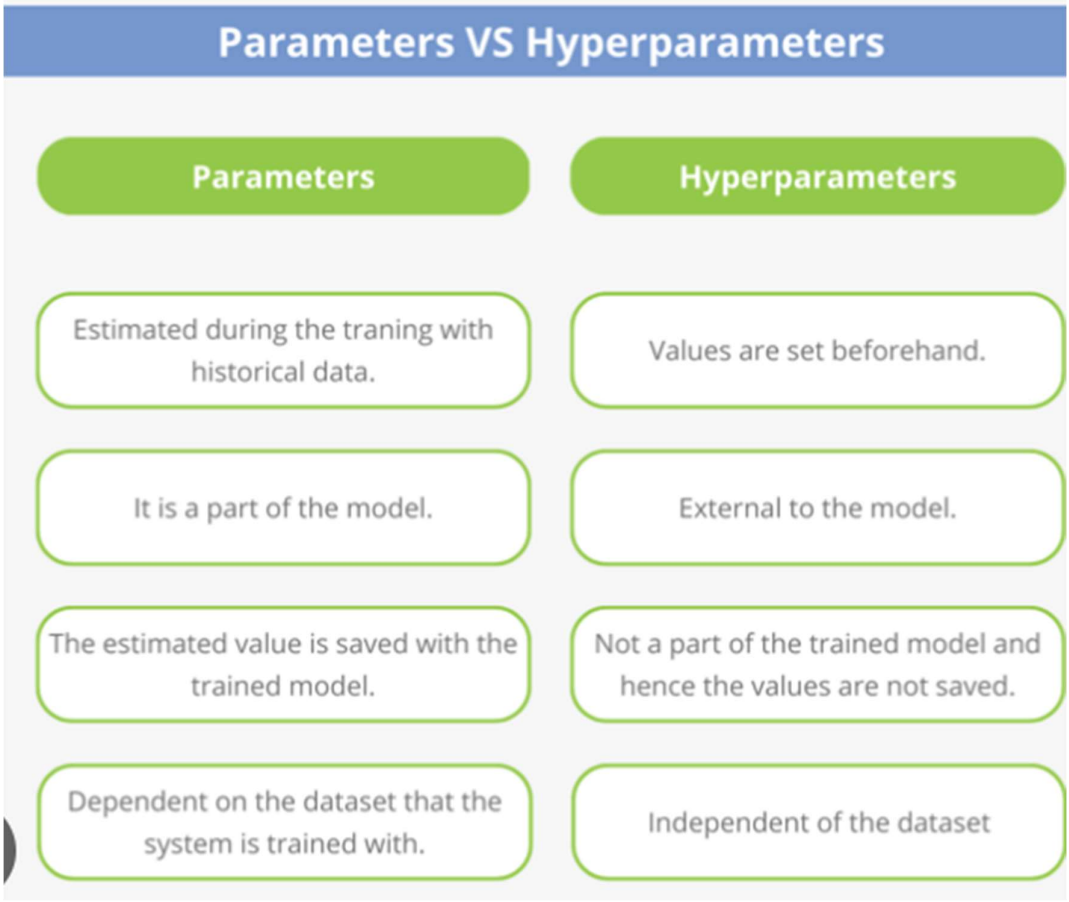

Bias is a term added to the input layer to give the perceptron more flexibility in modeling complex patterns in the input data.

**Parameters vs HyperParameters:**

Parameters are the values that are learned by a model during training, whereas hyperparameters are those that are set before the model is trained. During training, the parameters of a model are determined by the data. These values are used to make predictions based on new data.

Model parameters are configuration variables that are internal to the model, and a model learns them on its own. For example, W Weights or Coefficients of independent variables in the Linear regression model. Weights or Coefficients of independent variables SVM, weight, and biases of a neural network, cluster centroid in clustering.

Some examples of Hyperparameters are the learning rate for training a neural network, K in the KNN algorithm, etc.




6. Generating prediction using linear regression model.

**Linear regression** is a method we can use to quantify the relationship between one or more predictor variables and a response variable.

One of the most common reasons for fitting a regression model is to use the model to predict the values of new observations.

We use the following steps to make predictions with a regression model:

- **Step 1:** Collect the data.
- **Step 2:** Fit a regression model to the data.
- **Step 3:** Verify that the model fits the data well.
- **Step 4:** Use the fitted regression equation to predict the values of new observations.

7. **Loss function Vs Cost function.**

The terms cost function & loss function are analogous.

**Loss function:** Used when we refer to the error for a single training example.

**Cost function:** Used to refer to an average of the loss functions over an entire training data.

8. **Mean squared error.**

The Mean Squared Error measures how close a regression line is to a set of data points. It is a risk function corresponding to the expected value of the squared error loss. Mean square error is calculated by taking the average, specifically the mean, of errors squared from data as it relates to a function.

9. **Forward Vs Backward function.**

Feedforward means moving forward with provided input and weights (assumed in 1st run) till the output. And, backward propagation , as a name suggests, is moving from output to input. In BP, we reassign weights based on the loss and then forward propagation runs. These two process are independent for the training of your model.

10. **Relationship between gradient descent and weights and biases.**

Gradient descent is an iterative optimization algorithm used in machine learning to minimize a loss function. The loss function describes how well the model will perform given the current set of parameters (weights and biases), and gradient descent is used to find the best set of parameters.

11. **Epoch, Benefits of increasing epoch.**

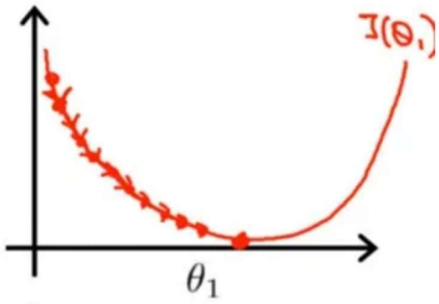
Adjusting the number of epochs in a neural network training process can significantly impact its performance. More epochs can help the model learn complex patterns, but too many may lead to overfitting.

## 12.Learning rate.

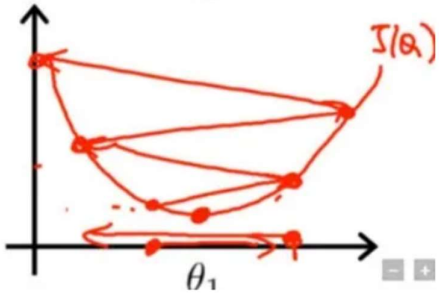
In machine learning and statistics, the learning rate is a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function.

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

If α is too small, gradient descent can be slow.




If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.




## 13.Optimizer in Pytorch

An optimizer is an algorithm or function that adapts the neural network's attributes, like learning rate and weights. Hence, it assists in improving the accuracy and reduces the total loss. But it is a daunting task to choose the appropriate weights for the model

torch.optim is a package implementing various optimization algorithms. Most commonly used methods are already supported, and the interface is general enough, so that more sophisticated ones can also be easily integrated in the future.