UNIT - I

Machine Learning Basics : Learning Algorithms, Capacity, Overfitting and Underfitting, Hyperparameters and Validation Sets, Estimators, Bias and Variance, Maximum Likelihood Estimation, Bayesian Statistics, Supervised Learning Algorithms, Unsupervised Learning Algorithms, Stochastic Gradient Descent, Building a Machine Learning Algorithm, Challenges Motivating Deep Learning.

Deep Feedforward Networks : Learning XOR, Gradient-Based Learning, Hidden Units, Architecture Design, Back-Propagation and Other Differentiation Algorithms.

Machine Learning Basics :

Machine learning is programming computers to optimize a performance criterion using example data or past experience.
Machine learning is programming computers to optimize a performance criterion using example data or past experience. We have a model defined up to some parameters, and learning is the execution of a computer program to optimize the parameters of the model using the training data or past experience.
In 1997, Tom Mitchell gave a "well-posed" mathematical and relational definition that "A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.
Example : Dog Vs Cat Classification
Task : Calssify images as Cat vs Dog
Performance : Accuracy
Experience : Set of images with correct labels
(Machine Learning Algorithm takes Feature as Input and produce output as Labels)

Techniques of ML (Learning Algorithm):

1. **Supervised Learning**
   a) Classification
   b) Regression
2. **Unsupervised Learning**
   a) Clustring
   b) Association
3. **Reinforcement Learning**

Learning Algorithms :

Learning algorithms are the methods used by machines to learn from data. They can be broadly classified into three types based on the type of learning they perform:

1. Supervised Learning

A training set of examples with the correct responses (targets) is provided and, based on this training set, the algorithm generalizes to respond correctly to all possible inputs.This is also called learning from exemplars.
Example:-
Prediction of future cases: Use the rule to predict the output for future inputs
Knowledge extraction: The rule is easy to understand
Compression: The rule is simpler than the data it explains
Outlier detection: Exceptions that are not covered by the rule, e.g., fraud

- **Classification**

classification is a predictive modeling problem where the class label is anticipated for a specific example of input data.

Example: Credit scoring
Differentiating between low-risk and high-risk customers from their income and savings

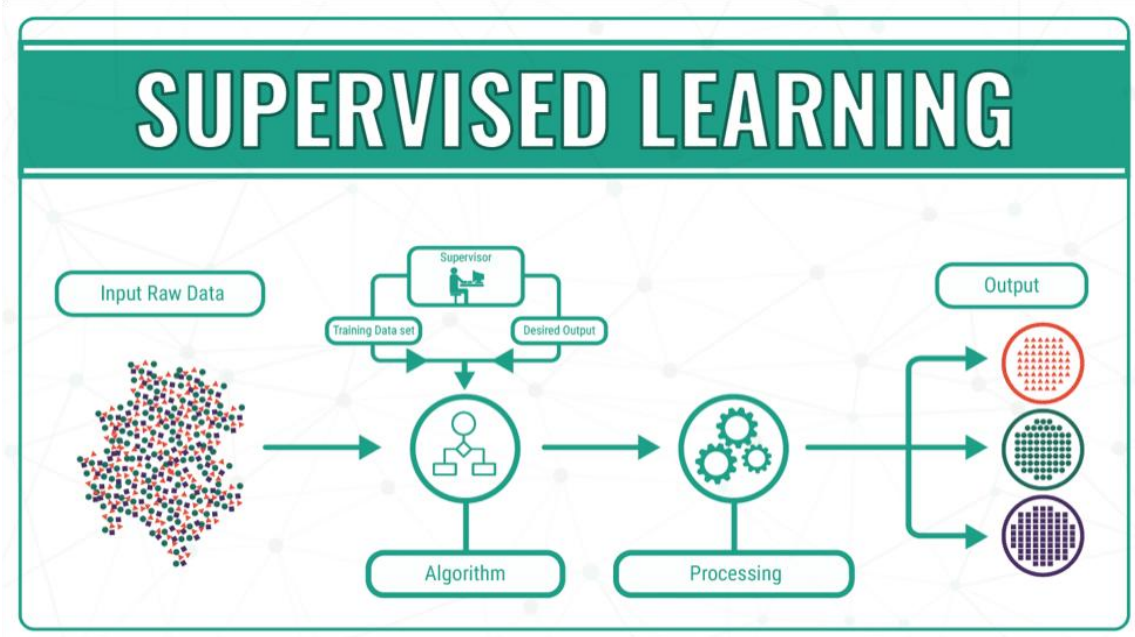
Classification Algorithms can be further divided into the Mainly two category:

1. Linear Models

- Logistic Regression
- Support Vector Machines

2. Non-linear Models

- K-Nearest Neighbours
- Kernel SVM
- Naïve Bayes
- Decision Tree Classification
- Random Forest Classification



Classification: Applications-

Face recognition: Pose, lighting, occlusion (glasses, beard), make-up, hair style

Character recognition: Different handwriting styles**.**

Speech recognition: Temporal dependency**.**

Medical diagnosis: From symptoms to illnesses

Biometric: Recognition/authentication using physical and/or behavioral characteristics: Face, iris, sig

● Regression

Problems where the output is a number are regression problems.Regression implies statistical tool used to identify the nature of relationship existing between a dependent variable and a set of independent variable.

Example: The output is the price of the car.

Price of a used car

x : car attributes

y : price

$y = g(x|\theta)$

Where g ( ) model, θ parameters

Types of regression:

- Linear Regression

- Logistic Regression
- Polynomial Regression
- Support Vector Regression
- Decision Tree Regression
- Random Forest Regression

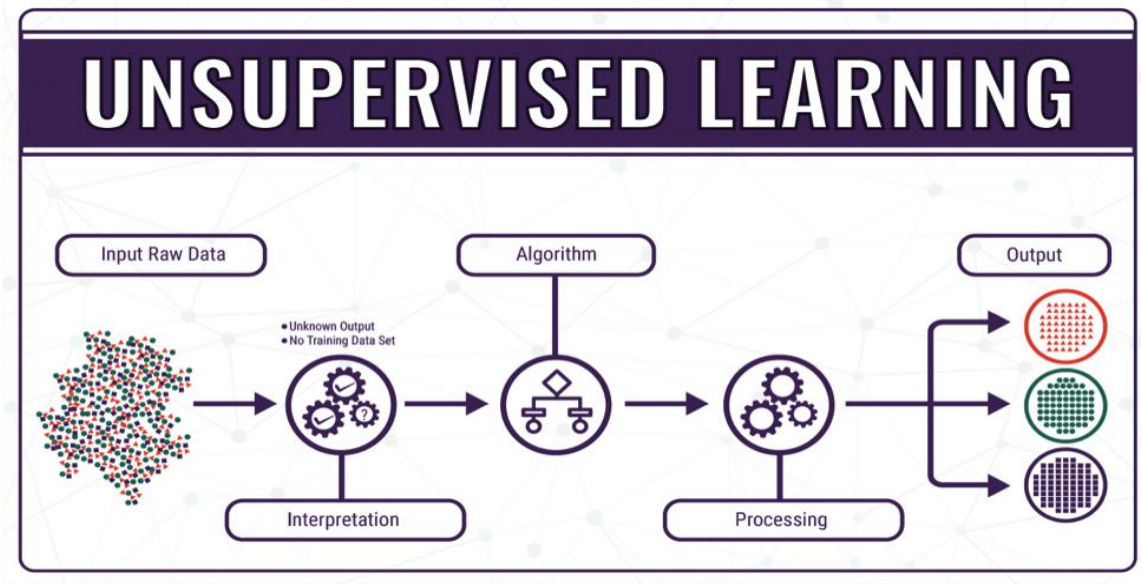
Regression : Applications-
Regression has a wide range of real-life applications.
- Financial forecasting (like house price estimates, or stock prices)
- Sales and promotions forecasting
- Testing automobiles
- Weather analysis and prediction
- Time series forecasting

2. Unsupervised Learning:

Learning useful structure without labeled classes, optimization criterion, feedback signal, or any other information beyond the raw data.

- Learning "what normally happens"
- No output
- Clustering: Grouping similar instances

Applications
- **Customer segmentation in CRM:** (Customer Relationship Management)
- **Image compression:** Color quantization
- **Bio-informatics:** Learning motifs (Distinctive repeating features)



Clustering :

It is a method of grouping the objects into clusters on the basis of similarities.

Example:-
- In Identification of Cancer Cells
- In Search Engines

- In Biology
- Customer Segmentation

Types of Clustering :

- **Density-Based Clustering**
- **Partitioning Clustering** (K-Means Clustering algorithm)
- **Distribution Model-Based Clustering** (Expectation-Maximization Clustering algorithm (Gaussian Mixture Models (GMM)))
- **Hierarchical Clustering** (Agglomerative Hierarchical algorithm)

● Learning Associations :

Associative learning is a style of learning that happens when two unrelated elements become connected in our brains thr3ough a process known as conditioning.

Basket analysis:

P (Y | X) probability that somebody who buys X also buys
Y where X and Y are products/services.

Example: P (bread | milk) = 0.7

Then, we can define the rule : 70 percent of customers who buy milk also buy bread.

● Reinforcement Learning :

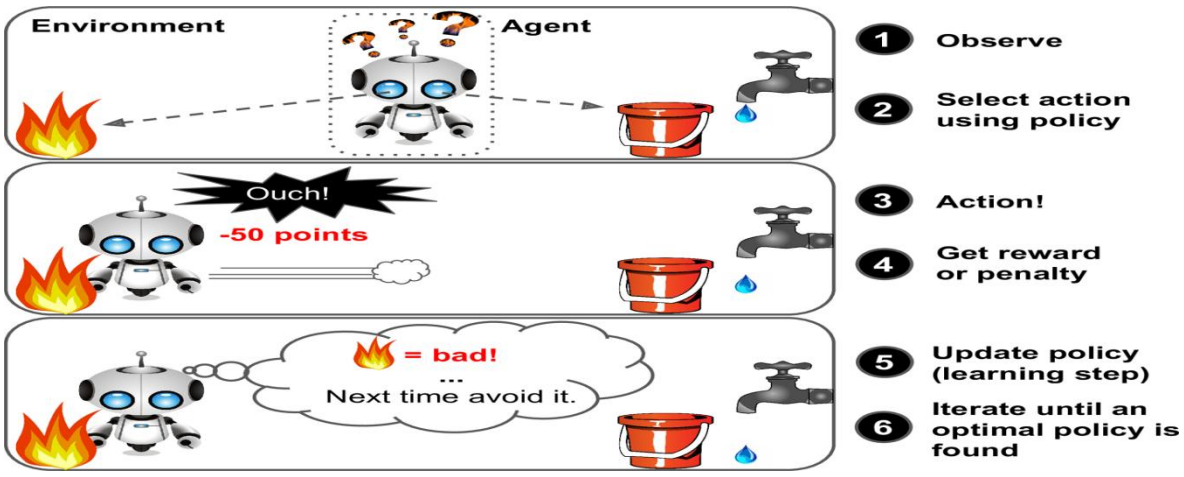
The machine learning program should be able to assess the goodness of policies and learn from past good action sequences to be able to generate a policy. Such learning methods are called reinforcement learning algorithms.

Key concepts include:

- **Agent**: The learner or decision-maker.
- **Environment**: The system with which the agent interacts.
- **Policy**: A strategy used by the agent to decide actions.
- **Reward Signal**: Feedback from the environment.

Applications :

- Game playing
- Robot in a maze
- Multiple agents, partial observability



- ## Capacity :

The capacity of a machine learning model refers to its ability to fit a wide variety of functions. It is influenced by the model's complexity, which includes factors such as the number of parameters and the structure of the model. Here are some key concepts related to capacity:

- **Underfitting:** When a model is too simple to capture the underlying pattern of the data, resulting in poor performance on both the training and test data.
- **Overfitting:** When a model is too complex and captures the noise in the training data, leading to poor generalization to new, unseen data.
- **Bias-Variance Trade-off:** A fundamental concept in machine learning where a balance must be struck between bias (error due to overly simplistic assumptions) and variance (error due to too much complexity). A model with high bias may underfit, while a model with high variance may overfit.

Hyper-parameter :

Most machine learning algorithms have several settings that we can use to control the behavior of the learning algorithm. These settings are called **hyperparameters.** The values of hyperparameters are not adapted by the learning algorithm itself.
Sometimes a setting is chosen to be a hyperparameter that the learning algorithm does not learn because it is difficult to optimize. More frequently, we do not learn the hyperparameter because it is not

appropriate to learn that hyperparameter on the training set.

A model **parameter** is a configuration variable that is **internal** to the model and whose value **can be estimated from data**.
Example: coefficients in logistic regression/linear regression, weights in a neural network, support vectors in SVM.

A model **hyperparameter** is a configuration that is **external** to the model and whose value **cannot be estimated from data**.
Example: max_depth in Decision Tree, learning rate in a neural network, C and sigma in SVM.

Basically, the goal of hyperparameter tuning is to get the optimal model's performance.
There are many methods to perform hyperparameter tuning.

1. **Grid Search**
2. **Random Search**
3. **Coarse to Fine Search etc…**

- ## Grid Search:

Basically, this is simply brute force where we have to test all of the possible combinations. Grid search is popular hyperparameter tuning technique, especially when the hyperparameter space is relatively small and discrete.

Grid Search Strategy:

- Construct a grid of all possible combinations of hyperparameters.
- For each combination, train a model using the specified hyperparameters.
- Evaluate the performance of each model using cross-validation or a separate validation set.
- Select the hyperparameters that yield the best performance according to your chosen evaluation metric.

Pros:

- Able to test all combinations within the hyperparameter space.
- Super simple to implement

Cons:

- Curse of dimensionality.
- Possible to miss the better hyperparameter combination outside the hyperparameter space

● Random Search:

In random search, you randomly select sets of hyperparameters from a defined search space and evaluate their performance using cross-validation. This approach is effective because it doesn't rely on assumptions about the relationships between hyperparameters and model performance.
This method is suited best when you do not know the suitable hyperparameter space (most often this is the case).

Random Search Strategy:

- Define the hyperparameter space and sampling method.
- Randomly sample a set of hyperparameters.
- Train and evaluate the model using the sampled hyperparameters.
- Repeat the above steps for a predefined number of iterations or until computational resources are exhausted.

Pros:

- Great for discovery and getting hyperparameter combinations that you would not have guessed intuitively.

Cons:

- Often requires more time to execute until getting the best combinations

● Coarse to Fine Search:

The coarse-to-fine strategy involves starting with a broad search over a wide range of hyperparameters and then narrowing down the search space to focus on areas that seem promising based on the results obtained during the initial coarse search.
Example: Let's consider a scenario where we are tuning hyperparameters for a support vector machine (SVM) classifier using the popular Iris dataset.
Coarse-to-Fine Strategy:

1. Perform Random Search on the initial hyperparameter space
2. Find promising area
3. Perform Grid/Random search in the smaller area

4. Continue until optimal score obtained or maximum iterations reached

Pros:

- Utilizes the advantages of Grid and Random Search.
- Spending more time on search spaces that are giving the good result

Cons:

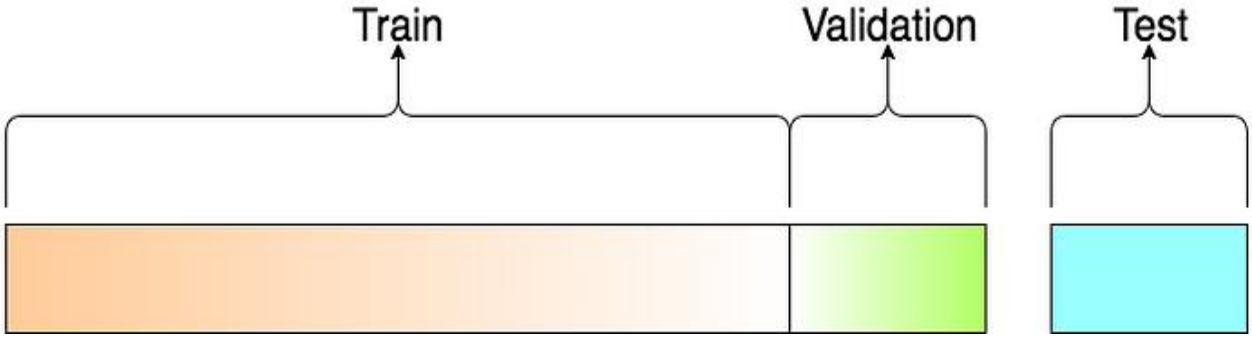
- Harder to implement since no package supporting this feature yet

Validation set:

Training Set: The sample of data used to fit the model. The actual dataset that we use to train the model (weights and biases in the case of a Neural Network). The model sees and learns from this data.

Validation Dataset: The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. The evaluation becomes more biased as skill on the validation dataset is incorporated into the model configuration.

Test Dataset: The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset.



Estimators

In machine learning, an *estimator* refers to any algorithm, model, or statistical method that estimates some underlying relationship or pattern within data. The term is used broadly and can refer to a wide range of methods, from simple linear regression to complex deep learning models.

Types of Estimators

Parametric Estimators:

1. Assume a specific form for the model, defined by a finite number of parameters.
2. Examples:
    1. **Linear Regression:** Estimates coefficients for a linear relationship between input features and the target variable.
    2. **Logistic Regression:** Estimates parameters for predicting binary outcomes.

Non-Parametric Estimators:

1. Do not assume a specific model form and instead rely on the data to determine the model's shape.
2. Examples:
    1. **k-Nearest Neighbors (k-NN):** Estimates the target value based on the closest data points.
    2. **Decision Trees:** Create a model that predicts the target variable by splitting the data based on feature values.

Ensemble Estimators:

1. Combine multiple models to improve prediction accuracy and robustness.
2. Examples:
    1. **Random Forest:** An ensemble of decision trees where each tree is trained on a subset of the data.
    2. **Gradient Boosting Machines (GBM):** Build a sequence of models, each correcting errors made by the previous one.

How Estimators Work

- **Training:** Estimators learn from data by adjusting their internal parameters to minimize some loss function or maximize a likelihood function. For example, in linear regression, the estimator adjusts the slope and intercept to minimize the sum of squared errors.
- **Prediction:** Once trained, an estimator can make predictions on new, unseen data. The quality of these predictions depends on how well the estimator has captured the underlying patterns in the training data.

Common Functions Associated with Estimators

1. **Fit (**fit**):** Trains the estimator on the data.
2. **Predict (**predict**):** Uses the trained estimator to make predictions.
3. **Score (**score**):** Evaluates the performance of the estimator, often by comparing predictions to actual outcomes.

Overfitting and Underfitting :

Overfitting and underfitting are two common issues encountered in deep learning (DL) models, as well as in other machine learning approaches.

● Overfitting:

When a model performs very well for training data but has poor performance with test data (new data), it is known as overfitting.

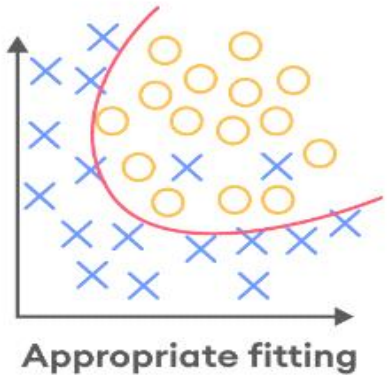
A statistical model is said to be overfitted when the model does not make accurate predictions on testing data. When a model gets trained with so much data, it starts learning from the noise and inaccurate data entries in our data set, and when testing with test data results in High variance.

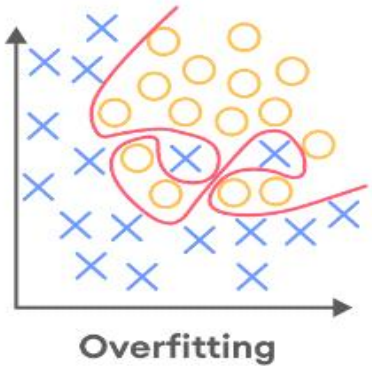
Reasons for Overfitting:

1. High variance and low bias.
2. The model is too complex.
3. The size of the training data.

Techniques to Reduce Overfitting:

1. Increase training data.
2. Reduce model complexity.
3. Early stopping during the training phase (have an eye over the loss over the training period as soon as loss begins to increase stop training).
4. Ridge Regularization and Lasso Regularization.
5. Use dropout for neural networks to tackle overfitting.





● Underfitting :

Underfitting occurs when a model is not able to make accurate predictions based on training data and hence, doesn't have the capacity to generalize well on new data.

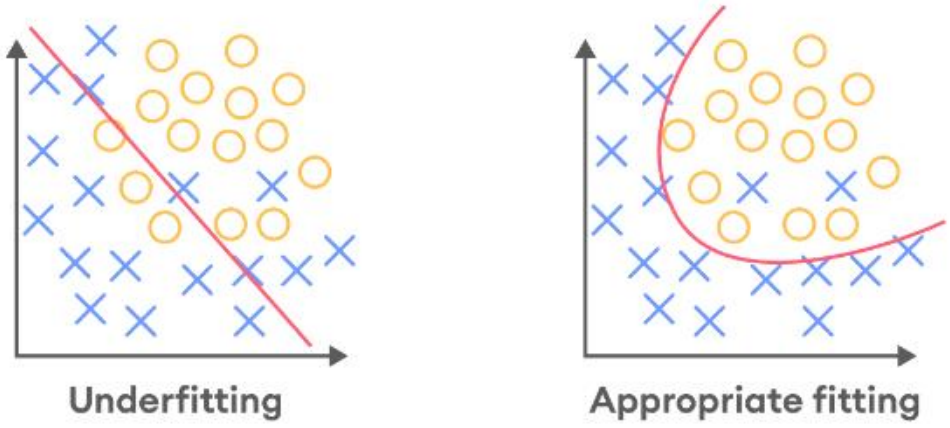
A statistical model is said to have underfitting when a model is too simple to capture data complexities. It represents the inability of the model to learn the training data effectively result in poor performance both on the training and testing data.

Reasons for Underfitting

1. The model is too simple, So it may be not capable to represent the complexities in the data.
2. The input features which is used to train the model is not the adequate representations of underlying factors influencing the target variable.
3. The size of the training dataset used is not enough.
4. Excessive regularization are used to prevent the overfitting, which constraint the model to capture the data well.
5. Features are not scaled.

Techniques to Reduce Underfitting

1. Increase model complexity.
2. Increase the number of features, performing feature engineering.
3. Remove noise from the data.
4. Increase the number of epochs or increase the duration of training to get better results.



Bias vs Variance

Bias and variance are two key components that you must consider when developing any good, accurate machine learning model.

Bias: Bias refers to the error introduced by approximating a real-world problem with a simplified model. It represents the difference between the average prediction of the model and the true value being predicted.
These differences between actual or expected values and the predicted values are known as error or bias error or error due to bias. Bias is a systematic error that occurs due to wrong assumptions in the machine learning process.

$$\text{Bias}(\hat{Y}) = E(\hat{Y}) - Y$$

where $EY^{\wedge}$ is the expected value of the estimator $Y^{\wedge}$. It is the measurement of the model that how well it fits the data.

- **Low Bias:** Low bias value means fewer assumptions are taken to build the target function. In this case, the model will closely match the training dataset.
- **High Bias:** High bias value means more assumptions are taken to build the target function. In this case, the model will not match the training dataset closely.

Ways to reduce high bias in Machine Learning

- **Use a more complex model:** One of the main reasons for high bias is the very simplified model. it will not be able to capture the complexity of the data.In such cases, we can make our mode more complex by increasing the number of hidden layers in the case of a deep neural network. Or we can use a more complex model like Polynomial regression for non-linear datasets, CNN for image processing.
- **Increase the number of features:** By adding more features to train the dataset will increase the complexity of the model.
- **Reduce** Regularization **of the model:** Regularization techniques such as L1 or L2 regularization can help to prevent overfitting and improve the generalization ability of the model.

Variance: Variance refers to the variability of model predictions for a given input data point. It measures how much the predictions for a given point differ across different training instances.

Variance is the measure of spread in data from its mean position. In machine learning variance is the amount by which the performance of a predictive model changes when it is trained on different subsets of the training data.

$$\text{Variance} = E[(\hat{Y} - E[\hat{Y}])^2]$$

where $E[Y^{\wedge}]$ is the expected value of the predicted values. Here expected value is averaged over all the training data.

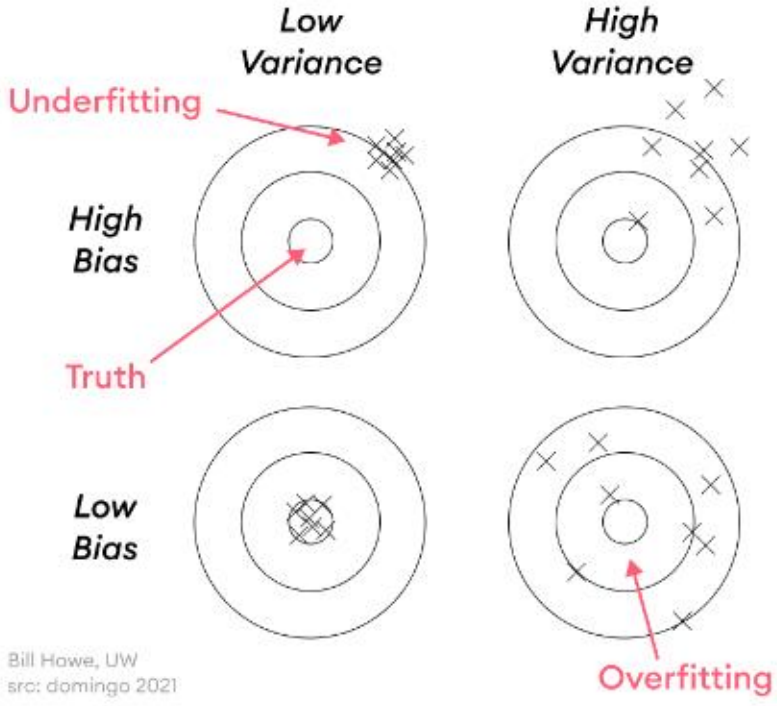
Ways to Reduce the reduce Variance in Machine Learning:

- **Cross-validation:** By splitting the data into training and testing sets multiple times, cross-validation can help identify if a model is overfitting or underfitting and can be used to tune hyperparameters to reduce variance.
- **Feature selection:** By choosing the only relevant feature will decrease the model's complexity. and it can reduce the variance error.
- **Regularization:** We can use L1 or L2 regularization to reduce variance in machine learning models
- **Ensemble methods:** It will combine multiple models to improve generalization performance. Bagging, boosting, and stacking are common ensemble methods.
- **Simplifying the model:** Reducing the complexity of the model, such as decreasing the number of parameters or layers in a neural network, can also help reduce variance and improve generalization performance.
- **Early stopping:** Early stopping is a technique used to prevent overfitting by

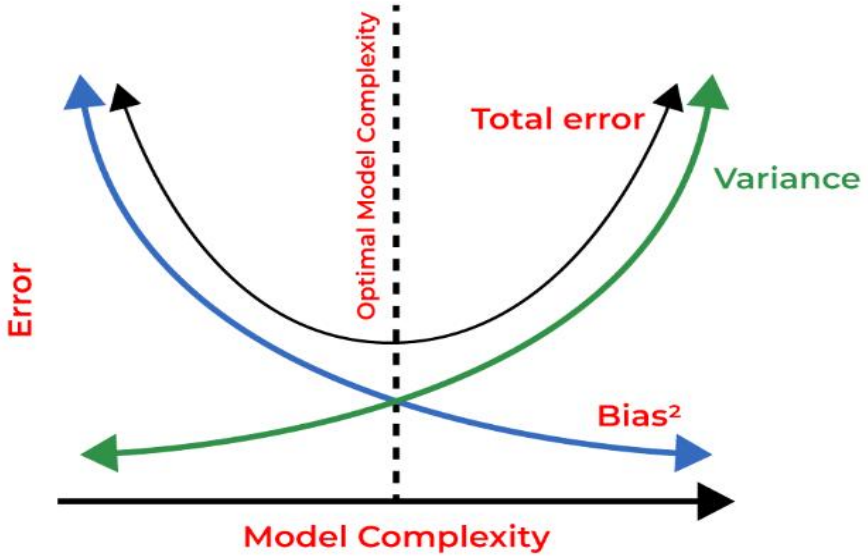
stopping the training of the deep learning model when the performance on the validation set stops improving.

Different Combinations of Bias-Variance
There can be four combinations between bias and variance.

- **High Bias, Low Variance:** A model with high bias and low variance is said to be underfitting.
- **High Variance, Low Bias:** A model with high variance and low bias is said to be overfitting.
- **High-Bias, High-Variance:** A model has both high bias and high variance, which means that the model is not able to capture the underlying patterns in the data (high bias) and is also too sensitive to changes in the training data (high variance). As a result, the model will produce inconsistent and inaccurate predictions on average.
- **Low Bias, Low Variance:** A model that has low bias and low variance means that the model is able to capture the underlying patterns in the data (low bias) and is not too sensitive to changes in the training data (low variance). This is the ideal scenario for a machine learning model, as it is able to generalize well to new, unseen data and produce consistent and accurate predictions. But in practice, it's not possible



Bias Variance Trade-off : If the model is very simple with fewer parameters, it may have low variance and high bias. Whereas, if the model has a large number of parameters, it will have high variance and low bias. So, it is required to make a balance between bias and variance errors, and this balance between the bias error and variance error is known as the **Bias-Variance trade-off.**



Maximum Likelihood Estimation

Maximum Likelihood Estimation (MLE) is a method used in statistics to estimate the parameters of a probabilistic model. The core idea behind MLE is to find the parameter values that maximize the likelihood of the observed data given the model.
Once parameters are estimated from the sample, the whole distribution is known. We estimate the parameters of the distribution from the given sample, plug in these estimates to the assumed model, and get an estimated distribution, which we then use to make a decision.
"A statistic is any value that is calculated from a given sample." In statistical inference, we make a decision using the information provided by a sample.
The method we use to estimate the parameters of a distribution is **maximum likelihood estimation.**

Key Concepts:
Likelihood Function:

1. The likelihood function measures the probability of the observed data given a set of parameters.

    1. For a dataset $X=\{x1,x2,\ldots,xn\}$ and a model with parameters θ, the likelihood function $L(\theta)$ is defined as:

$$L(\theta)=P(X|\theta)=\prod_{i=1}^{n} P(x_i \mid \theta)$$

    2. This function treats the data as fixed and the parameters as variables.

Log-Likelihood:

- Often, the logarithm of the likelihood function is used because it simplifies the mathematics (turns products into sums) and is computationally more stable.
- The log-likelihood function is:

$$\mathbf{logL(\theta) = {}^{n}\sum_{i=1} logP(x_i \mid \theta)}$$

Maximization:

- The goal of MLE is to find the parameter values θ that maximize the likelihood (or log-likelihood) function. This is often done by taking the derivative of the log-likelihood with respect to θ, setting it to zero, and solving for θ.
- In practice, especially for complex models, optimization algorithms like gradient ascent or more sophisticated methods like Expectation-Maximization (EM) are used to find the maximum likelihood estimates.

Bayesian Statistics

Bayesian statistics is a paradigm in statistics and machine learning that uses Bayes' theorem to update the probability of a hypothesis as more evidence or data becomes available. Unlike frequentist approaches, which provide point estimates of parameters, Bayesian methods incorporate prior beliefs and provide a probabilistic framework for model inference and prediction.

Bayes' Theorem (Bayes' Rule): We often find ourselves in a situation where we know P(y | x) and need to know P(x | y). Fortunately, if we also know P(x), we can compute the desired quantity using Bayes' rule:

$$P(\mathrm{x} \mid \mathrm{y}) = \frac{P(\mathrm{x})P(\mathrm{y} \mid \mathrm{x})}{P(\mathrm{y})}.$$

Note that while P (y) appears in the formula, it is usually feasible to compute **P(y) =** $\sum_x$ **P(y | x)P(x)**, so we do not need to begin with knowledge of P(y). Bayes' rule is straightforward to derive from the definition of conditional pro bability.

Bayes' Theorem Derivation: Now, let's derive Bayes' Theorem using the definition of conditional probability.

1. Start with the definition of conditional probability: **P(x ∣ y)= P(x∩y) / P(y)**
2. Rearrange to express the joint probability **P(x∩y)** in terms of conditional probability: **P(x∩y)=P(x ∣ y)· P(y)**
3. Similarly, express **P(y∩x)** in terms of conditional probability: **P(y∩x)=P(y ∣ x)· P(x)**
4. Since **P(x∩y)=P(y∩x),** we can set these equal: **P(x ∣ y)· P(y)=P(y ∣ x)· P(x)**
5. Solve for **P(x ∣ y)** to get Bayes' Theorem: **P(x ∣ y)=P(x)P(y ∣ x)/P(y)**

Prior Probability: P(x) is called the prior.

Class likelihood: p(y|x) is called the class likelihood and is the conditional probability

Evidence : p(y), the evidence, is the Marginal probability (The probability of an event irrespective of the outcomes of other random variables)

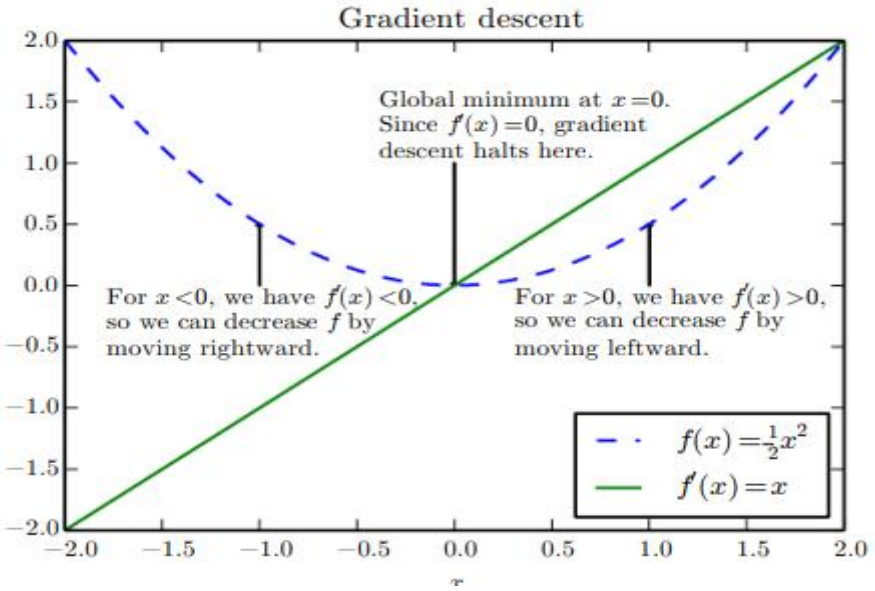
posterior = prior × likelihood / evidence

Bayes' Theorem is a fundamental principle in probability theory that relates conditional probabilities. It is named after Reverend Thomas Bayes, an 18th-century statistician and theologian. The term **"Bayes' Rule"** is often used interchangeably with **Bayes' Theorem,** and both refer to the same mathematical formula.

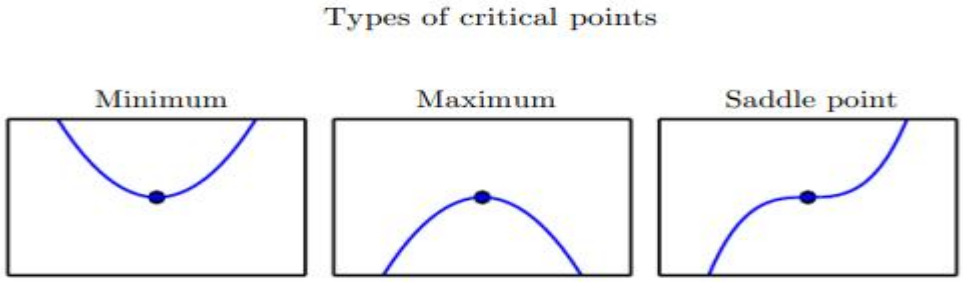
Stochastic Gradient Descent

Gradient Descent is an optimization algorithm for finding a local minimum of a differentiable function. Gradient descent in machine learning is simply used to find the values of a function's parameters (coefficients) that minimize a cost function as far as possible.

$$X^* = X - \textit{learning-rate}\ f'(x)$$



A **local minimum** is a point where f(x) is lower than at all neighboring points, so it is no longer possible to decrease f(x) by making infinitesimal steps. A **local maximum** is a point where f(x) is higher than at all neighboring points, so it is not possible to increase f(x) by making infinitesimal steps. Some critical points are neither maxima nor minima. These are known as **saddle points.**



Types of Gradient Descent:

1. Batch Gradient Descent: Batch gradient descent **(BGD)** is used to find the error for each point in the training set and update the model after evaluating all training examples. This procedure is known as the **training epoch.** In simple words, it is a greedy approach where we have to sum over all examples for each update.

2. Stochastic gradient descent: Stochastic gradient descent **(SGD)** is a type of gradient descent that runs one training example per iteration. As it requires only one training example at a time, hence it is easier to store in allocated memory.

3. MiniBatch Gradient Descent: Mini Batch gradient descent **(MGD)** is the combination of both batch gradient descent and stochastic gradient descent. It divides the training datasets into small batch sizes then performs the updates on those batches separately. Splitting training datasets into smaller batches make a balance to maintain the computational efficiency of batch gradient descent and speed of stochastic gradient descent.

Stochastic Gradient Descent :

Stochastic Gradient Descent (SGD) is a variant of the **Gradient Descent** algorithm that is used for optimizing machine learning models. It addresses the computational inefficiency of traditional Gradient Descent methods when dealing with large datasets in machine learning projects.

Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to fitting linear classifiers and regressors under convex loss functions such as (linear) Support Vector Machines and Logistic Regression.

In SGD, instead of using the entire dataset for each iteration, only a single random training example (or a small batch) is selected to calculate the gradient and update the model parameters. This random selection introduces randomness into the optimization process, hence the term "stochastic" in stochastic Gradient Descent.

$$\theta_{t+1} = \theta_t - \alpha \nabla J(\theta_t)$$

where θ are the model parameters, α is the learning rate, and $J(\theta)$ is the loss function.

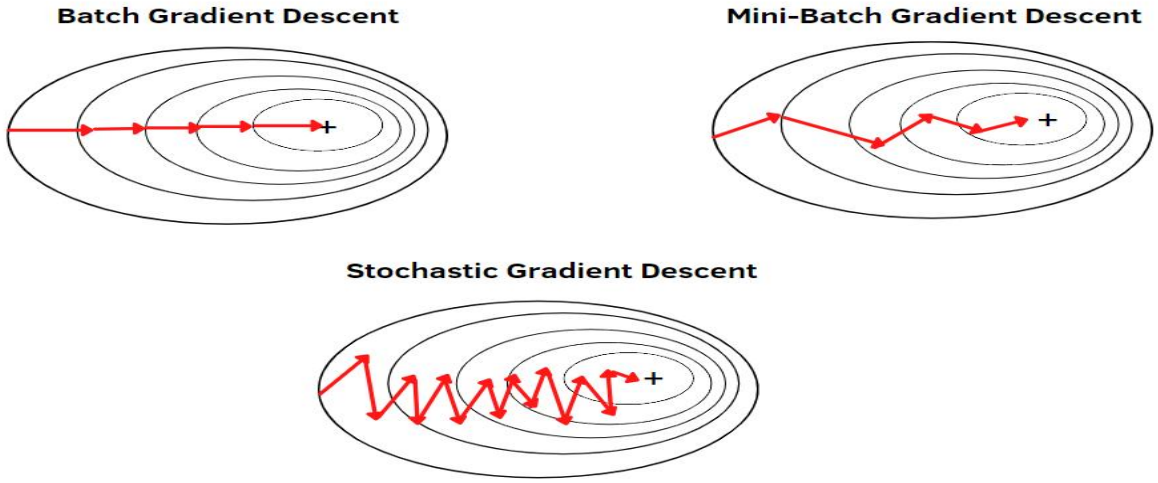
Stochastic Gradient Descent Algorithm

- **Initialization**: Randomly initialize the parameters of the model.
- **Set Parameters**: Determine the number of iterations and the learning rate (alpha) for updating the parameters.
- **Stochastic Gradient Descent Loop**: Repeat the following steps until the model converges or reaches the maximum number of iterations:
  1. Shuffle the training dataset to introduce randomness.
  2. Iterate over each training example (or a small batch) in the shuffled order.
  3. Compute the gradient of the cost function with respect to the model parameters using the current training example (or batch).

4. Update the model parameters by taking a step in the direction of the negative gradient, scaled the learning rate.
5. Evaluate the convergence criteria, such as the difference in the cost function between iterations of the gradient.

- **Return Optimized Parameters**: Once the convergence criteria are met or the maximum number of iterations is reached, return the optimized model parameters.

.LOL9O



In SGD, since only one sample from the dataset is chosen at random for each iteration, the path taken by the algorithm to reach the minima is usually noisier than your typical Gradient Descent algorithm.

Building a Machine Learning Algorithm :

This comprehensive guide will take you through the process of building a machine-learning model

Step 1: Data Collection for Machine Learning

Data collection is a crucial step in the creation of a machine learning model, as it lays the foundation for building accurate models. In this phase of machine learning model development, relevant data is gathered from various sources to train the machine learning model and enable it to make accurate predictions.

Step 2: Preprocessing and Preparing Your Data

Preprocessing and preparing data is an important step that involves transforming raw data into a format that is suitable for training and testing for our models. This phase aims to clean i.e. remove null values, and garbage values, and normalize and preprocess the data to achieve greater accuracy and performance of our machine learning models.

Step 3: Selecting the Right Machine Learning Model

Selecting the right **machine learning model** plays a pivotal role in building of successful model, with the presence of numerous algorithms and techniques available easily, choosing the most suitable model for a given problem significantly impacts the accuracy and performance of the model.

Step 4: Training Your Machine Learning Model

In this phase of building a machine learning model, we have all the necessary ingredients to train our model effectively. This involves utilizing our prepared data to teach the model to recognize patterns and make predictions based on the input features. During the training process, we begin by feeding the preprocessed data into the selected machine-learning algorithm.

Step 5: Evaluating Model Performance

Once you have trained your model, it's time to assess its performance. There are various metrics used to evaluate model performance, categorized based on the type of task: regression/numerical or classification.

1. For regression tasks, common evaluation metrics are:

- **Mean Absolute Error (MAE):** MAE is the average of the absolute differences between predicted and actual values.
- **Mean Squared Error (MSE):** MSE is the average of the squared differences between predicted and actual values.
- **Root Mean Squared Error (**RMSE**):** It is a square root of the MSE, providing a measure of the average magnitude of error.
- **R-squared (R2):** It is the proportion of the variance in the dependent variable that is predictable from the independent variables.

2. For classification tasks, common evaluation metrics are:

- **Accuracy:** Proportion of correctly classified instances out of the total instances.
- **Precision:** Proportion of true positive predictions among all positive predictions.
- **Recall:** Proportion of true positive predictions among all actual positive instances.
- **F1-score:** Harmonic mean of precision and recall, providing a balanced measure of model performance.
- **Confusion Metrics:** It is a matrix that summarizes the performance of a classification model, showing counts of true positives, true negatives, false positives, and false negatives instances.

Step 6: Tuning and Optimizing Your Model

As we have trained our model, our next step is to optimize our model more. Tuning and optimizing helps our model to maximize its performance and generalization ability. This process involves fine-tuning hyperparameters, selecting the best algorithm, and improving features through feature engineering techniques.

Step 7: Deploying the Model and Making Predictions

Deploying the model and making predictions is the final stage in the journey of creating an ML model. Once a model has been trained and optimized, it's to integrate it into a production environment where it can provide real-time predictions on new data.

Challenges Motivating Deep Learning:

1. Handling Complex and High-Dimensional Data

- **Challenge:** Traditional machine learning models often struggle with high-dimensional data, such as images, videos, and large text corpora.
- **Motivation for Deep Learning:** Deep learning models, particularly convolutional neural networks (CNNs) for images and recurrent neural networks (RNNs) for sequences, have been designed to efficiently handle and extract features from complex, high-dimensional data.

2. Feature Engineering

- **Challenge:** Traditional machine learning methods require manual feature engineering, which is time-consuming and often domain-specific.
- **Motivation for Deep Learning:** Deep learning models can automatically learn features from raw data through multiple layers of abstraction, reducing the need for manual feature engineering.

3. Scalability to Large Datasets

- **Challenge:** As the volume of data has increased exponentially, traditional models have struggled to scale effectively.
- **Motivation for Deep Learning:** Deep learning models are designed to scale well with large datasets, benefiting from large-scale data and computational power to improve performance.

4. Improving Model Accuracy

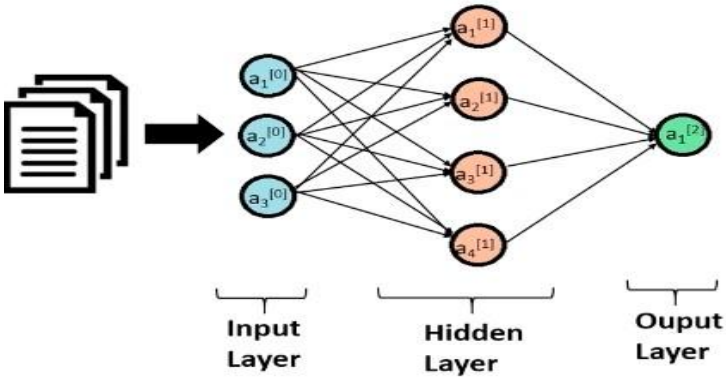
- **Challenge:** Many tasks, such as image recognition, natural language processing, and speech recognition, require very high accuracy.
- **Motivation for Deep Learning:** Deep learning has significantly improved accuracy in these domains, surpassing traditional methods and, in some cases, achieving human-level performance.

5. Non-Linear Relationships

- **Challenge:** Many real-world problems involve complex, non-linear relationships that are difficult for traditional linear models to capture.
- **Motivation for Deep Learning:** Deep neural networks, with their non-linear activation functions, can model complex, non-linear relationships more effectively than traditional models.

Deep Feedforward Networks :

The process of receiving an input to produce some kind of output to make some kind of prediction is known as **Feed Forward**. Feed Forward neural network is the core of many other important neural networks such as convolution neural network.
In the feed-forward neural network, there are not any feedback loops or connections in the network. Here is simply an input layer, a hidden layer, and an output layer.
There can be multiple hidden layers which depend on what kind of data you are dealing with. The number of hidden layers is known as the depth of the neural network. The **deep neural network** can learn from more functions. Input layer first provides the neural network with data and the output layer then make predictions on that data which is based on a series of functions. ReLU Function is the most commonly used activation function in the **deep neural network**.
A deep feedforward network is an artificial neural network where connections between the nodes do not form a cycle. In other words, information moves in one direction—from input nodes, through hidden nodes (if any), and finally to output nodes. The term "deep" refers to the presence of multiple hidden layers between the input and output layers.



$$a_1^{[1]} = \text{activation\_function}(W_{11}^{[1]} * a_1^{[0]} + W_{12}^{[1]} * a_2^{[0]} + W_{13}^{[1]} * a_3^{[1]} + B1)$$

$$a_2^{[1]} = \text{activation\_function}(W_{21}^{[1]} * a_1^{[1]} + W_{22}^{[1]} * a_2^{[1]} + W_{23}^{[1]} * a_3^{[1]} + B1)$$

Let's denote some variables before providing the formulas:
X: Input vector (features) to the neural network.
$W^{(l)}$: Weight matrix for layer l.
$b^{(l)}$: Bias vector for layer l.
$Z^{(l)}$: Weighted sum of inputs for layer l.
$A^{(l)}$: Activation output for layer l.
For a neural network with L layers (including input and output layers), the forward propagation formulas for each layer are as follows:
Input Layer (Layer 0):

$$A^{(0)}=X$$

Hidden Layers (l=1,2,…,L−1):

$$Z^{(l)}=W^{(l)} \cdot A^{(l-1)}+b^{(l)}$$
$$A^{(l)}=\text{Activation}(Z^{(l)})$$

Here, Activation(·) represents the activation function applied element-wise to $Z^{(l)}$.
Output Layer:

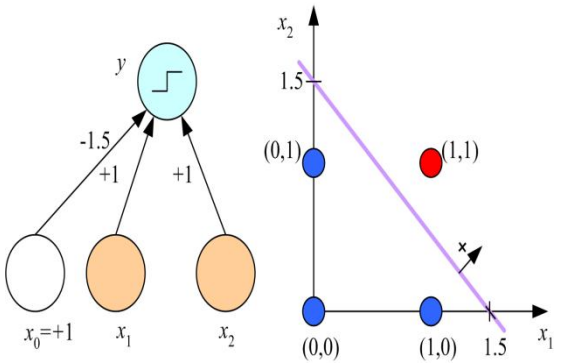
$$Z^{(L)}=W^{(L)} \cdot A^{(L-1)}+b^{(L)}$$
$$Y\hat{}=\text{Activation}(Z^{(L)})$$

Y^ represents the predicted output of the neural network.

Learning XOR:

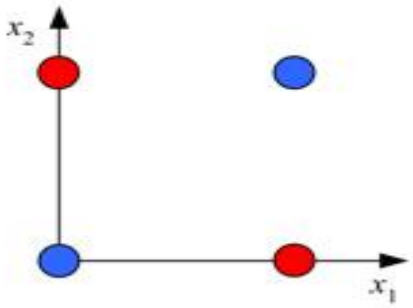
In a Boolean function, the inputs are binary and the output is 1 if the corresponding function value is true and 0 otherwise.
Therefore, it can be seen as a two-class classification problem. As an example, for learning to AND two inputs

x_1	x_2	r
0	0	0
0	1	0
1	0	0
1	1	1



Though Boolean functions like AND and OR are linearly separable and are solvable using the perceptron, certain functions like XOR are not.
XOR problem is not linearly separable. This can also be proved by noting that there are no w0, w1, and w2 values that satisfy the following set of inequalities-

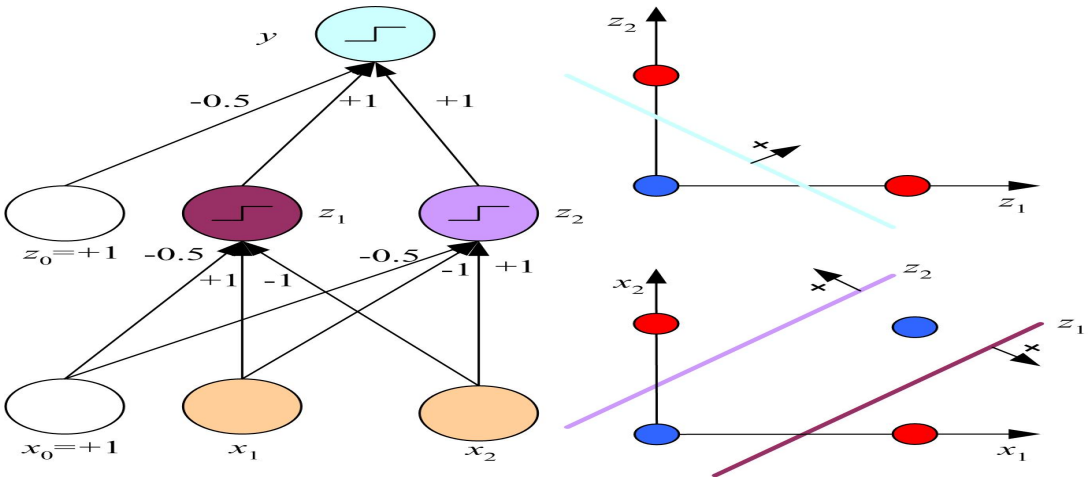
x_1	x_2	r
0	0	0
0	1	1
1	0	1
1	1	0



A perceptron that has a single layer of weights can only approximate linear functions of the input and cannot solve problems like the XOR, where the discriminant to be estimated is nonlinear.
The multi-layer perceptron that solves the XOR problem. The hidden units and the output have the threshold activation function with threshold at 0.
For example,

x_1 XOR x_2 = (x_1 AND ~ x_2) OR (~ x_1 AND x_2)

The multi-layer perceptron that solves the XOR problem. The hidden units and the output have the threshold activation function with threshold at 0.



Gradient-Based Learning:

Gradient-based learning refers to a family of algorithms that are used in machine learning and optimization, particularly in training neural networks.

Gradient-Based Learning:

- **Loss Function**: This is a function that measures how well the model is performing. For a neural network, this could be something like Mean Squared Error (MSE) for regression or Cross-Entropy for classification tasks. The goal is to minimize this function.
- **Gradient**: The gradient is a vector of partial derivatives of the loss function with respect to the model parameters. It points in the direction of the steepest increase in the loss function.
- **Gradient Descent**: This is an iterative optimization algorithm used to minimize the loss function. The parameters are updated by moving in the opposite direction of the gradient (since we want to minimize the loss, not maximize it).
    - **Learning Rate**: A hyperparameter that controls the size of the steps taken during each update. Too high a learning rate might cause the algorithm to overshoot the minimum, while too low a learning rate can make the learning process slow.
    - **Variants of Gradient Descent**:
        1. **Batch Gradient Descent**: Computes the gradient using the entire dataset. This can be very slow for large datasets.
        2. **Stochastic Gradient Descent (SGD)**: Computes the gradient using a single sample. This introduces noise into the optimization process but can make the learning faster.
        3. **Mini-Batch Gradient Descent**: A compromise between batch and stochastic gradient descent, where the gradient is computed using a small batch of samples.
- **Backpropagation**: In the context of neural networks, backpropagation is an algorithm used to compute the gradient of the loss function with respect to each weight in the network. It uses the chain rule of calculus to efficiently compute these gradients by propagating the error backward through the network.

- **Convergence**: The process of gradient descent continuing until the loss function reaches a minimum. However, this could be a local minimum (for non-convex functions) rather than a global minimum.
- **Regularization**: To prevent overfitting, regularization techniques like L1 or L2 regularization can be added to the loss function. These techniques penalize large weights and thus encourage the model to find simpler solutions.

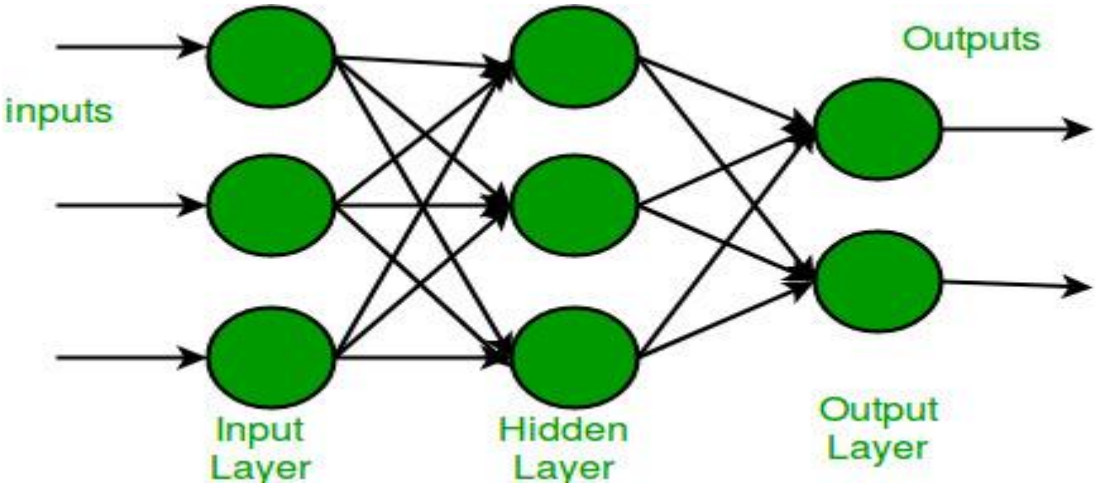
Hidden Units :

Hidden units in deep feedforward neural networks are neurons located in the hidden layers between the input and output layers. They process inputs by applying a weighted sum followed by a non-linear activation function like **ReLU, Sigmoid, or Tanh**. The number of hidden units and layers determines the network's capacity to learn complex patterns. During training, their weights are adjusted through backpropagation to minimize the loss function. Proper tuning of hidden units is crucial to avoid underfitting or overfitting, enabling the network to model data effectively.

Architecture Design:

A perceptron that has a single layer of weights can only approximate linear functions of the input and cannot solve problems like **the XOR**, where the **discriminant** to be estimated is non-linear

Multi-layer perception is also known as MLP. It is fully connected dense layers, which transform any input dimension to the desired dimension. A multi-layer perception is a neural network that has multiple layers. To create a neural network we combine neurons together so that the outputs of some neurons are inputs of other neurons.



Here are the key components and concepts associated with a Multilayer Perceptron:

Input Layer: The first layer in the MLP, consisting of nodes that represent the input features of the model. Each node corresponds to a feature in the input data.

Hidden Layer: Layers between the input and output layers are referred to as hidden

layers. Each node in a hidden layer takes inputs from all nodes in the previous layer, performs a weighted sum, applies an activation function, and passes the result to the next layer.

Output Layer: The final layer that produces the model's output. The number of nodes in the output layer depends on the type of task (e.g., binary classification, multi-class classification, regression).

Weight and Bias: Each connection between nodes has an associated weight, which the network learns during training. Additionally, each node has a bias term. These weights and biases are adjusted during the training process to minimize the error between predicted and actual outputs.

Activation functions: Activation functions introduce non-linearity into the network, enabling it to learn complex relationships in the data. Common activation functions include the sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU).

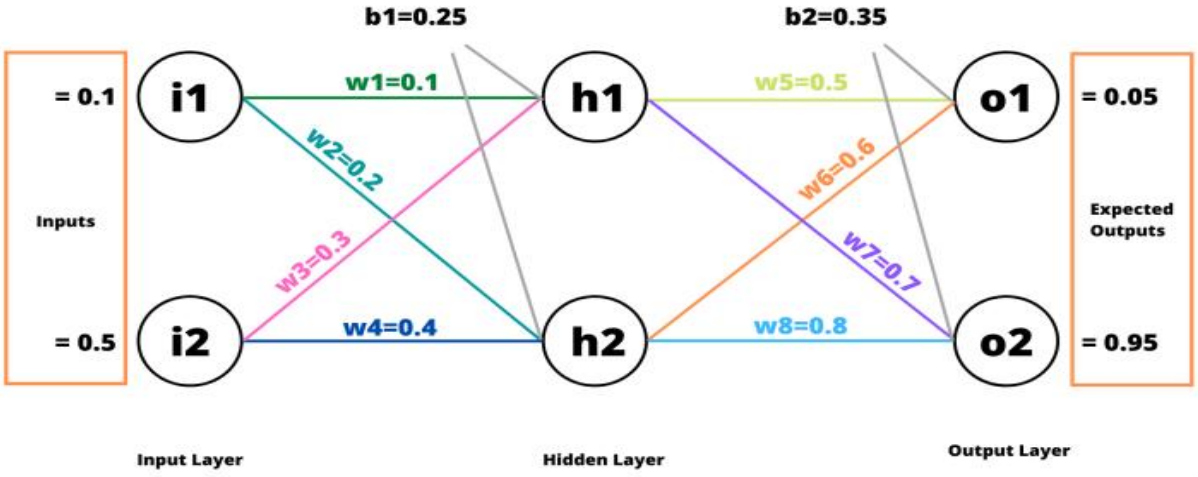
Back-Propagation :

Backpropagation is a process involved in training a neural network. It involves taking the error rate of a forward propagation and feeding this loss backward through the neural network layers to fine-tune the weights.

The aim of Backpropagation (backward pass) is to distribute the total error back to the network so as to update the weights in order to minimize the cost function (loss). The weights are updated in such as way that when the next forward pass utilizes the updated weights, the total error will be reduced by a certain margin (until the minima is reached).

To update the weight, we calculate the error correspond to each weight with the help of a total error. The error on weight w is calculated by differentiating total error with respect to w.

$$\text{Error}_w = \frac{\partial E_{total}}{\partial w}$$



If we look closely at the example neural network, we can see that **Error_w (Total**

Error) is affected by **output (Predicted Output),** output is affected by **sum (Weighted Sum)**, and sum is affected by **w (Weight in neurons).** It's time to recall the Chain Rule.

For weights in the output layer For w5

$$\frac{\partial E_{total}}{\partial w5} = \frac{\partial E_{total}}{\partial output_{o1}} * \frac{\partial output_{o1}}{\partial sum_{o1}} * \frac{\partial sum_{o1}}{\partial w5}$$

Like wise compute for all weights.

For weights in the hidden layer For w1

$$\frac{\partial E_1}{\partial w1} = \frac{\partial E_1}{\partial output_{o1}} * \frac{\partial output_{o1}}{\partial sum_{o1}} * \frac{\partial sum_{o1}}{\partial output_{h1}} * \frac{\partial output_{h1}}{\partial sum_{h1}} * \frac{\partial sum_{h1}}{\partial w1}$$

Once we've computed all the new weights, we need to update all the old weights with these new weights. Once the weights are updated, one Backpropagation cycle is finished. Now the forward pass is done and the total new error is computed. And based on this newly computed total error the weights are again updated. This goes on until the loss value converges to minima.

Backpropagation Algorithm:

Step 1: Inputs X, arrive through the preconnected path.

Step 2: The input is modeled using true weights W. Weights are usually chosen randomly.

Step 3: Calculate the output of each neuron from the input layer to the hidden layer to the output layer.

Step 4: Calculate the error in the outputs

Backpropagation Error= Actual Output – Desired Output

Step 5: From the output layer, go back to the hidden layer to adjust the weights to reduce the error.

Step 6: Repeat the process until the desired output is achieved.

Other Differentiation Algorithms:

- **Computational Complexity**

The computational complexity of differentiation algorithms refers to the resources (time and space) required to compute derivatives.

- **Symbolic Differentiation:**

1. **Complexity:** Often exponential in the size of the expression.

2. **Description:** Manipulates algebraic expressions to find the exact derivative. Can be slow and produce very large expressions (expression swell).

- **Numerical Differentiation:**

1. **Complexity:** Typically linear with respect to the number of function evaluations.
2. **escription:** Approximates derivatives using finite differences. It's simple but can be inaccurate due to rounding errors and requires multiple function evaluations.

- **Automatic Differentiation (AD):**

1. **Complexity:** Linear with respect to the number of operations in the function.
2. **Description:** AD evaluates both the function and its derivatives simultaneously by applying the chain rule. It provides exact derivatives efficiently, with a time complexity close to that of evaluating the function itself.

- **Algebraic substitution**

Algebraic substitution is a mathematical technique used to simplify or solve equations by replacing a variable or expression with another equivalent expression. This method is often used to make complex equations more manageable, to find roots, or to transform equations into a more usable form.

For example, if you have an equation like y=x^2+2x+1, you can substitute u=x+1 to simplify it to y=u^2, which is easier to work with.