Rapport Deep Learning

GeeksforGeeks.org

Réalisé par :

Maryame Laouina

Table de matière

chapter 1 : Introduction to Deep Learning.....5

Introduction.....5

Difference Between Artificial Intelligence vs Machine Learning vs Deep Learning.....8

chapter 2 : Basic Neural Network.....12

1. Difference between ANN and BNN.....12

2. Single Layer Perceptron in TensorFlow.....13

3. Multi-Layer Perceptron Learning in Tensorflow.....14

4. Deep Neural net with forward and back propagation from scratch – Python.....15

5. Understanding Multi-Layer Feedforward Networks.....17

6. ML – List of Deep Learning Layers.....17

chapter 3 : Activation Functions.....20

1. introduction.....20

2. Types Of Activation Function in ANN.....23

Chapter 4 : Artificial Neural Network.....24

1. Cost function in neural networks.....24

2. How does Gradient Descent work.....25

3. Vanishing or Exploding Gradients Problems.....26

Vanishing Gradients Problem:.....27

Exploding Gradients Problem:.....27

Solutions:.....27

4. Choose the optimal number of epochs.....28

5. Batch Normalization in Deep Learning.....30

6. Difference between Sequential and functional API..... 31
7. Classification & Regression..... 33
8. Fine-Tuning & Hyperparameters..... 34
Hyperparameters:..... 35
Introduction to Convolution Neural Network..... 35
1- Definition :..... 35
2- CNN architecture..... 36
3- How Convolutional Layers works..... 36
4- Layers used to build ConvNets..... 37
5- What are some common activation functions used in CNNs?..... 38
6- What are some common regularization techniques used in CNNs?..... 38
7- Advantages of Convolutional Neural Networks (CNNs):..... 39
8- Disadvantages of Convolutional Neural Networks (CNNs):..... 39
Digital Image Processing Basics..... 39
1- Definition :..... 39
2- The basic steps involved in digital image processing are:..... 39
3- What is an image?..... 40
3-1 Types of an image..... 40
3-2 Image as a Matrix..... 40
4- Advantages of Digital Image Processing:..... 40
5- Disadvantages of Digital Image Processing:..... 41
6- Difference between Image Processing and Computer Vision:..... 41
CNN | Introduction to Pooling Layer..... 42
1- What is the Pooling Layer ?..... 42
2- Why use Pooling Layers?..... 42
3- Types of Pooling Layers:..... 42
4- Advantages of Pooling Layer:..... 43
5- Disadvantages of Pooling Layer:..... 43
Convolutional Neural Network (CNN) Architectures..... 43
1- LeNet-5..... 44
2- AlexNNet..... 44
3- GoogleNet (Inception vl)..... 45

4- ResNet (Residual Network).....45
5- DenseNet.....46
Object Detection vs Object Recognition vs Image Segmentation..... 47
1- Object Recognition:.....47
1-2 Object Recognition Using Machine Learning.....47
1-3 Object Recognition Using Deep Learning.....47
2- Image Classification :..... 48
3- Image Segmentation:.....48
4- Applications:.....49
YOLO v2 – Object Detection..... 49
1- Definition.....50
2- Architecture Changes vs YOLOv1:.....50
2-1 Batch Normalization:.....50
2-2 High Resolution Classifier:.....50
2-3 Use Anchor Boxes For Bounding Boxes:.....50
2-4 Dimensionality clusters:.....51
2-5 Direct Location Problem:.....52
2-6 Fine Grained Features :.....52
3- Architecture:.....52
4- Training:.....53
5- Results and Conclusion:.....53
Basis of NLP..... 54
1- What is NLP?.....54
1-2 Advantages of NLP.....54
1-3 Disadvantages of NLP.....54
1-4 Components of NLP.....54
1-5 Applications of NLP.....54
1-6 Phases of Natural Language Processing.....55
2- Introduction to NLTK: Tokenization, Stemming, Lemmatization, POS Tagging.....55
2-1 Tokenization.....55
2-2 Stemming and Lemmatization.....56
2-3 Part of Speech Tagging.....56

3- Word Embeddings in NLP.....56
3-1 Goal of Word Embeddings.....56
3-2 Implementations of Word Embeddings:.....56
3-3 Word2Vec:.....56
3-4 GloVe:.....57
3-5 Benefits of using Word Embeddings:.....58
3-6 Drawbacks of Word Embeddings:.....58
Introduction to Recurrent Neural Network.....58
1- Définition.....58
2- Types Of RNN.....58
❖ One to One.....59
❖ One To Many.....59
❖ Many to One.....59
❖ Many to Many.....60
3- Recurrent Neural Network Architecture.....60
4- How does RNN work?.....61
4-1 Backpropagation Through Time (BPTT).....61
5- Issues of Standard RNNs.....62
6- Advantages and Disadvantages of Recurrent Neural Network.....62
❖ Advantages.....62
7- Applications of Recurrent Neural Network.....62
8- Difference between RNN and Simple Neural Network.....63
Short term Memory.....64
1- LSTM.....64
2- Shallow Attention Fusion & Deep Attention Fusion.....65
Deep Learning | Introduction to Long Short Term Memory.....65
1- What is LSTM?.....66
1-1 Bidirectional LSTM.....66
1-2 Architecture and Working of LSTM.....66
1-3 LTSM vs RNN.....67
2- LSTM – Derivation of Backpropagation through time.....68
Gated Recurrent Unit Networks.....69

1- Principle of GRU..... 69
2- Outlines :..... 70
3- How do Gated Recurrent Units solve the problem of vanishing gradients?..... 71
Generative learning..... 71
1- Autoencoders -Machine Learning..... 71
1-1 What are Autoencoders?..... 71
1-2 Architecture of Autoencoder in Deep Learning..... 71
1-3 Types of Autoencoders..... 72
1-4 How Autoencoders works ?..... 73
2- Variational AutoEncoders..... 74
2-1 What is a Variational Autoencoder?..... 74
2-2 Architecture of Variational Autoencoder..... 74
3- Contractive Autoencoder (CAE)..... 75
3-1 The Loss function:..... 75
3-2 Relationship with Sparse Autoencoder..... 75
3-3 Relationship with Denoising Autoencoder..... 76
Generative Adversarial Networks..... 76
1. Basics of Generative Adversarial Networks (GANs)..... 76
Introduction to GANs..... 76
Components of GANs..... 76
Iterative Learning Process..... 77
Applications of GANs..... 77
Understanding GANs in Practice..... 77
Generative Adversarial Network (GAN)..... 77
Key Components of GANs..... 77
Working of GANs..... 78
Applications of GANs..... 78
Advantages and Disadvantages of GANs..... 78
Use Cases of Generative Adversarial Networks..... 79
Newly Discovered Use Cases of GANs..... 79
Cycle Generative Adversarial Network (CycleGAN)..... 80
Key Features of CycleGAN..... 80

Generator and Discriminator Architecture..... 80
StyleGAN – Style Generative Adversarial Networks..... 81
Architecture:..... 81
Reinforcement Learning..... 82
Introduction :..... 83
Main points in Reinforcement learning..... 83
Working on Reinforcement Learning:..... 83
Thompson Sampling:..... 85
Examples of Real-World Applications:..... 86
Unraveling Markov Decision Processes:..... 86
The Building Blocks of an MDP:..... 86
The Bellman Equation:..... 87
Meta-Learning: Learning to Learn..... 89
Q-Learning in python..... 90
Introduction :..... 90
Explanation:..... 90
Deep Q-Learning..... 91
Introduction :..... 91
Here's how DQN works in a nutshell:..... 92
DQN tackles a key challenge:..... 92

chapter 1 : Introduction to Deep Learning

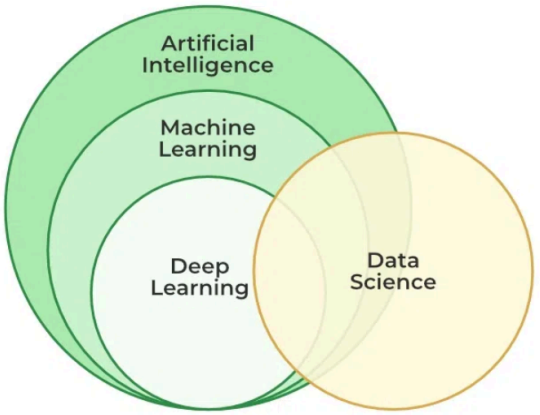
Introduction

In summary, Deep Learning, a subset of Machine Learning, relies on artificial neural networks, specifically deep neural networks (DNNs), inspired by the human brain. It excels in learning intricate patterns without explicit programming, capitalizing on advances in processing power and the abundance of large datasets. With its key feature being the use of deep neural networks with multiple interconnected layers, Deep Learning achieves success in diverse fields like image recognition and natural language processing. Its ability to autonomously learn from data, coupled with the

accessibility of cloud computing and specialized hardware, indicates a promising future as data availability and computing resources continue to expand.

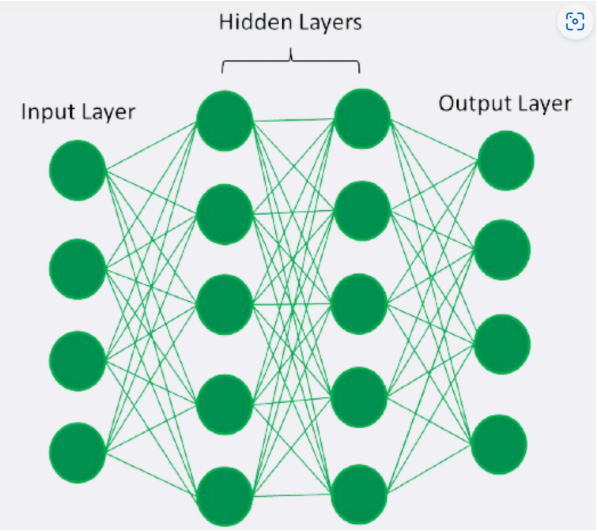
What is Deep Learning?

Deep learning, a subset of machine learning, employs artificial neural networks to process complex data through interconnected nodes. Successful in applications like computer vision, it spans supervised, unsupervised, and reinforcement learning. Convolutional Neural Networks and Recurrent Neural Networks excel in tasks such as image classification. Unsupervised methods, like autoencoders, identify patterns, while reinforcement learning, using algorithms like DQNs and DDPG, maximizes rewards in decision-making. Deep learning's adaptability ensures ongoing transformative impact in diverse domains.



Artificial neural networks

Artificial neural networks, inspired by human neuron function, consist of interconnected layers, including input, hidden, and output layers. Neurons in the hidden layer compute weighted inputs from the previous layer, adjusting weights during training for optimal performance. These artificial neurons form the network, varying in complexity based on dataset patterns. In a fully connected setup, information flows from input through hidden layers to produce the final output. Weights between units control their influence, enabling the network to learn and generate valuable outputs. This process of information transformation and learning characterizes the functionality of artificial neural networks.



Fully Connected Artificial Neural Network

Difference between Machine Learning and Deep Learning

Machine Learning (ML) and Deep Learning (DL) are subsets of Artificial Intelligence, sharing similarities but differing in various aspects:

Machine Learning	Deep Learning
- Applies statistical algorithms to uncover hidden patterns and relationships in datasets.	- Utilizes artificial neural network architecture for learning patterns in datasets.
- Effective with smaller datasets.	- Requires larger volumes of data compared to machine learning
- Suited for low-complexity tasks.	- Particularly adept at handling complex tasks like image processing and natural language processing.
- Requires less time for model training.	- Takes more time for model training.

- Involves manual extraction of relevant features for model creation.	- Automatically extracts relevant features in an end-to-end learning process.
- Less complex and offers easier result interpretation.	- More complex, operating like a black box, making result interpretations challenging.
- Can operate on CPUs or with less computing power.	- Requires high-performance computers with GPUs for optimal performance.

Types of neural networks

Deep Learning models possess the capability to autonomously learn features from data, making them particularly effective in tasks such as image recognition, speech recognition, and natural language processing. Key architectures in deep learning include Feedforward Neural Networks (FNNs), which follow a linear information flow and find use in diverse applications like image classification. Convolutional Neural Networks (CNNs) specialize in image and video recognition, automatically extracting features for tasks such as image classification and object detection. Recurrent Neural Networks (RNNs) excel in processing sequential data, maintaining an internal state that captures previous inputs, rendering them suitable for tasks like speech recognition, natural language processing, and language translation.

Applications of Deep Learning

Deep learning finds diverse applications in computer vision, natural language processing (NLP), and reinforcement learning.

Computer Vision: Deep learning excels in object detection, image recognition, and segmentation, enabling tasks like self-driving cars and medical imaging.

Natural Language Processing (NLP): Deep learning facilitates automatic text generation, language translation, sentiment analysis, and speech recognition, enhancing communication and understanding of human language.

Reinforcement Learning: Deep learning in reinforcement learning powers game-playing achievements, robotics tasks, and control systems for optimizing complex environments like power grids and traffic management.

Challenges in Deep Learning

While deep learning has made substantial progress in various domains, it faces several challenges that need attention:

1.Data Availability:
- Deep learning relies on extensive datasets for effective training, posing a challenge in collecting sufficient and diverse data for optimal performance.

2. Computational Resources:
- Training deep learning models is computationally demanding, necessitating specialized hardware like GPUs and TPUs, raising concerns about resource accessibility.

3. Time-Consuming:
- Handling sequential data can be time-consuming, with the duration extending to days or months depending on computational resources, affecting efficiency.

4. Interpretability:
-Deep learning models, operating like black boxes, present challenges in result interpretation due to their inherent complexity, making it difficult to understand their decision-making processes.

5. Overfitting:
- Repeated training of models can lead to overfitting, where the model becomes overly specialized to the training data, resulting in poor performance on new, unseen data.

advantages and disadvantages of deep learning

Deep Learning boasts several advantages, including high accuracy, automated feature engineering, scalability, flexibility, and continual improvement with more data. However, it comes with drawbacks such as high computational requirements, the need for substantial labeled data, challenges in model interpretability, the risk of overfitting, and a black-box nature. These disadvantages should be carefully weighed against the benefits when deciding whether to employ Deep Learning for a specific task.

Difference Between Artificial Intelligence vs Machine Learning vs Deep Learning

Artificial Intelligence (AI) integrates human intelligence into machines through algorithms. It involves training machines to mimic human brain functions, emphasizing learning, reasoning, and self-correction. Machine Learning (ML), a subset of AI, enables systems to learn autonomously from experiences and improve without explicit programming, focusing on accessing and utilizing data. Deep Learning (DL), a sub-part of ML, utilizes Neural Networks to emulate human brain behavior, identifying patterns and classifying information on larger datasets, with a self-administered prediction mechanism. AI, ML, and DL collectively strive for machines to exhibit human-like cognitive capabilities, with each level representing a nuanced approach to achieving this goal.

Below is a table of differences between Artificial Intelligence, Machine Learning and Deep Learning:

Artificial Intelligence	**Machine Learning**	**Deep Learning**
AI is the broad field focused on imbuing machines with human-like intelligence through algorithms. It encompasses subfields such as robotics, natural language processing, and computer vision. AI systems can be rule-based, knowledge-based, or data-driven.	ML is a subset of AI, concentrating on algorithms that learn from data without explicit programming. ML algorithms fall into categories like supervised learning (using labeled data), unsupervised learning (working with unlabeled data), and reinforcement learning (learning through trial and error)	DL is a subset of ML that employs Neural Networks to simulate human brain functionality. It excels in tasks like image and speech recognition by utilizing deep neural networks with multiple layers. DL is particularly effective when dealing with large datasets.

Example:

the examples highlight applications of Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning (DL) across various industries:

Artificial Intelligence (AI):

AI encompasses tasks requiring human intelligence, and some examples include:

- Speech recognition for applications like self-driving cars and security systems.
- Personalized recommendations in e-commerce and streaming services.
- Predictive maintenance in analyzing sensor data to foresee equipment failures.
- Medical diagnosis through AI-powered systems analyzing patient data.
- Autonomous vehicles using AI algorithms for navigation and decision-making.
- Virtual Personal Assistants (VPAs) like Siri or Alexa utilizing natural language processing.
- Fraud detection in financial transactions.

Machine Learning (ML):

ML is a subset of AI, focusing on algorithms learning from data. Examples include:

- Image recognition in self-driving cars, security systems, and medical imaging.
- Speech recognition for transcription and word identification.
- Natural language processing (NLP) for chatbots and virtual assistants.
- Recommendation systems analyzing user data for personalized suggestions.
- Sentiment analysis in social media monitoring.
- Predictive maintenance systems for anticipating equipment failures.
- Spam filters in emails and credit risk assessment by financial institutions.

Deep Learning (DL):

DL, a type of ML, uses artificial neural networks with multiple layers. Examples include:

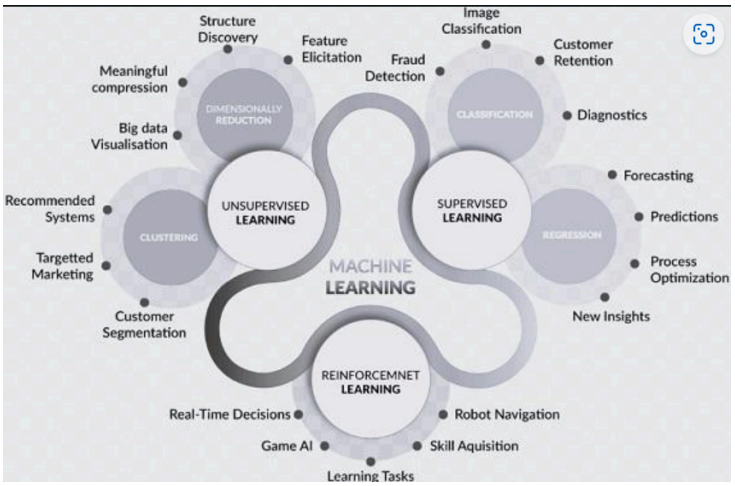
- Image and video recognition for classification and analysis.
- Generative models creating new content based on existing data.
- Autonomous vehicles analyzing sensor data for decision-making.
- Image classification recognizing objects and scenes.
- Speech recognition transcribing spoken words into text.
- Natural language processing for sentiment analysis and language translation.
- Recommender systems making personalized recommendations.
- Fraud detection in financial transactions.
- Game-playing AI competing at a superhuman level.
- Time series forecasting for future values in data like stock prices and weather patterns.

What Does an AI Engineer Do?

What Does a AI Engineer Do?

An AI Engineer is a professional responsible for designing, developing, and implementing artificial intelligence (AI) systems. Key responsibilities include the design and development of AI algorithms, data analysis to identify patterns, model training and evaluation for accuracy improvement, deployment and maintenance of AI models in production environments, collaboration with stakeholders to meet requirements, staying updated with AI advancements for research and innovation, and effective communication of results. Strong backgrounds in computer science, mathematics, statistics, and experience in developing AI algorithms, along with familiarity with programming languages like Python and R, are essential for AI Engineers.

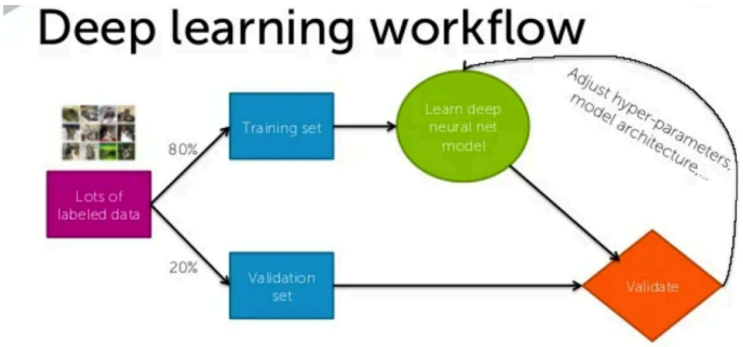
What Does a Machine Learning Engineer Do?



What Does a Machine Learning Engineer Do?

A Machine Learning Engineer is a professional responsible for designing, developing, and implementing machine learning (ML) systems. Key responsibilities include designing and developing ML algorithms for specific problems, analyzing and interpreting data to identify patterns, training and evaluating ML models for accuracy improvement, deploying and maintaining ML models in production environments, collaborating with stakeholders to meet requirements, staying updated with the latest advancements in ML for research and innovation, and effectively communicating the results of their work to stakeholders. Machine Learning Engineers must possess a strong background in computer science, mathematics, and statistics, along with experience in developing ML algorithms and solutions. Familiarity with programming languages such as Python and R, as well as experience with ML frameworks and tools, is crucial for success in this role.

What Does a Deep Learning Engineer Do?



What Does a Deep Learning Engineer Do?

A Deep Learning Engineer is a professional responsible for designing, developing, and implementing deep learning (DL) systems. Key responsibilities include designing and developing DL algorithms, analyzing and interpreting large datasets to identify patterns, training and evaluating DL models for accuracy improvement, deploying and maintaining DL models in production environments, collaborating with stakeholders to meet requirements, staying updated with the latest advancements in

DL for research and innovation, and effectively communicating the results of their work to stakeholders. Deep Learning Engineers work closely with data scientists, software engineers, and business leaders to ensure that DL solutions meet the needs of stakeholders.

chapter 2 : Basic Neural Network

1. Difference between ANN and BNN

1. Artificial Neural Network (ANN):

- Based on a Feed-Forward strategy, passing information through nodes until reaching the output node.
- Advantages: Can learn regardless of data type, suitable for financial time series forecasting.
- Disadvantages: Simple architecture makes it hard to explain behavior, dependency on hardware.

2. Biological Neural Network (BNN):

- Structure includes synapse, dendrites, cell body, and axon, with processing carried out by neurons.
- Advantages: Synapses are input processing elements, capable of processing highly complex parallel inputs.
- Disadvantages: Lack of controlling mechanism, slower processing speed due to complexity.

3. Differences between ANN and BNN:

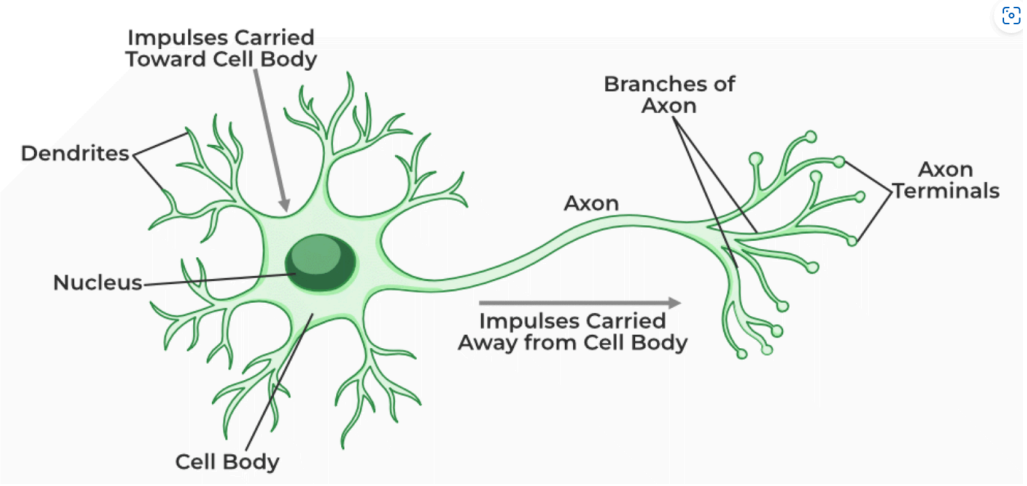
- Neurons: BNN neurons are more complex and diverse, having multiple dendrites and axons, while ANN neurons are simplified with usually only a single output.
- Synapses: BNN synapses are more flexible, with connections modified by learning and experience, while ANN connections are fixed with weights determining strength.
- Neural Pathways: BNN pathways are highly complex and diverse, modifiable by experience, while ANN pathways are simpler and predetermined by network architecture.

The table provides a comparison between Artificial Neural Networks (ANN) and Biological Neural Networks (BNN) across various parameters:

Parameters	**ANN**	**BNN**

Structure	Input, weight, output, hidden layer.	Dendrites, synapse, axon, cell body.
Learning	Requires very precise structures and formatted data.	Can tolerate ambiguity.
Processor	Complex, high-speed processor, one or a few.	Simple, low-speed processor, large number.
Memory	Separate from the processor, non-content addressable	Integrated into the processor, content-addressable.
Computing	Centralized, sequential, stored programs.	Distributed, parallel, self-learning.
Reliability	Very vulnerable.	Robust.
Expertise	Numerical and symbolic manipulations.	Perceptual problems.
Operating Environment	Well-defined, well-constrained.	Poorly defined, un-constrained.
Fault Tolerance	The potential for fault tolerance.	Performance degraded even on partial damage.

2. Single Layer Perceptron in TensorFlow

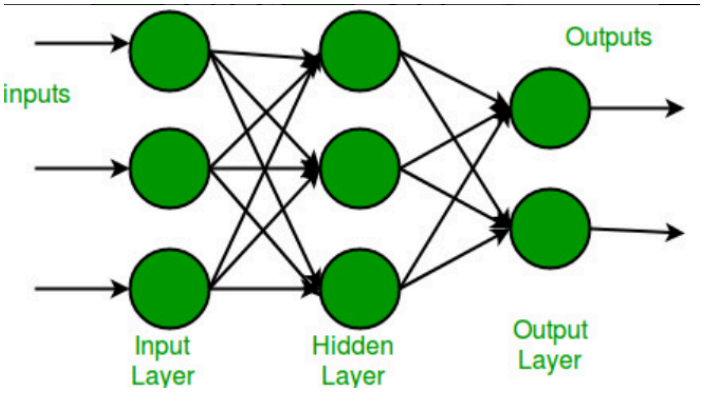


Structure of a biological neuron

Biological neuron has three basic functionality

-Receive signal from outside.
-Process the signal and enhance whether we need to send information or not.
-Communicate the signal to the target cell which can be another neuron or gland.

In the same way, neural networks also work.

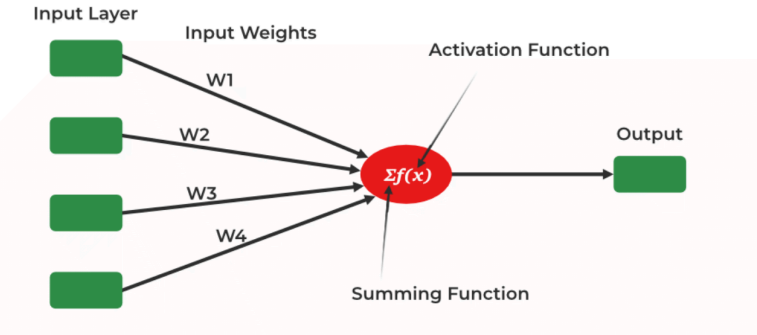


Neural Network in Machine Learning

What is Single Layer Perceptron?

It is one of the oldest and first introduced neural networks. It was proposed by Frank Rosenblatt in 1958. Perceptron is also known as an artificial neural network. Perceptron is mainly used to compute the logical gate like AND, OR, and NOR which has binary input and binary output.

The main functionality of the perceptron is:
Takes input from the input layer
Weight them up and sum it up.
Pass the sum to the nonlinear function to produce the output.



Single-layer neural network

3. Multi-Layer Perceptron Learning in Tensorflow

A Multi-Layer Perceptron (MLP), also known as a multi-layered neural network, is a type of neural network with fully connected dense layers that can transform input dimensions to the desired output dimensions. It consists of an input layer, one or more hidden layers, and an output layer. Each layer contains nodes (neurons), and connections between nodes are formed to process information.

In an MLP, the input layer has one neuron for each input, the output layer has one neuron for each output, and hidden layers can have any number of nodes. The connections between nodes are illustrated in a schematic diagram. The nodes in each layer process information and pass it to the next layer. A sigmoid activation function is commonly used in every node, converting real values to a range between 0 and 1.

The implementation of an MLP involves coding in Python using the TensorFlow library. The network can be trained using various algorithms to optimize the weights and biases, allowing it to learn and make predictions based on input data.

4. Deep Neural net with forward and back propagation from scratch – Python

The implementation of a deep neural network from scratch, consisting of a hidden layer with four units and one output layer. The process involves several key steps:

1. Visualizing the input data: Understand and visualize the data that the neural network will be trained on.

2. Deciding the shapes of Weight and bias matrix: Define the shapes of the weight and bias matrices, crucial components of the neural network.

3. Initializing matrices and functions: Initialize the matrices and choose activation functions for the hidden and output layers.

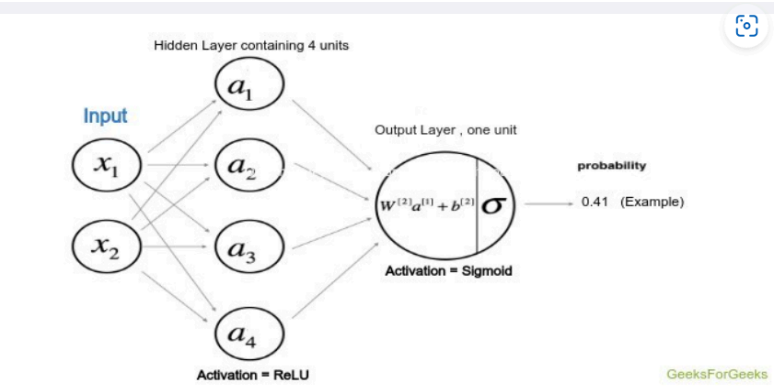
4. Implementing forward propagation: Develop the forward propagation method, which involves passing input data through the network to generate predictions.

5. Implementing cost calculation: Define a method to calculate the cost or error between the predicted output and the actual output.

6. Backpropagation and optimizing: Implement the backpropagation algorithm to optimize the weights and biases, improving the model's performance.

7. Prediction and visualizing the output: Use the trained model for making predictions and visualize the output.

The architecture of the model is depicted with a hidden layer utilizing the Hyperbolic Tangent as the activation function, while the output layer, designed for a classification problem, employs the sigmoid function. The step-by-step implementation allows for a comprehensive understanding of building a deep neural network from the ground up.



Model Architecture

Weights and bias:

The weights and the bias that is going to be used for both the layers have to be declared initially and also among them the weights will be declared randomly in order to avoid the same output of all units, while the bias will be initialized to zero. The calculation will be done from the scratch itself and according to the rules given below where W1, W2 and b1, b2 are the weights and bias of first and second layer respectively. Here A stands for the activation of a particular layer.

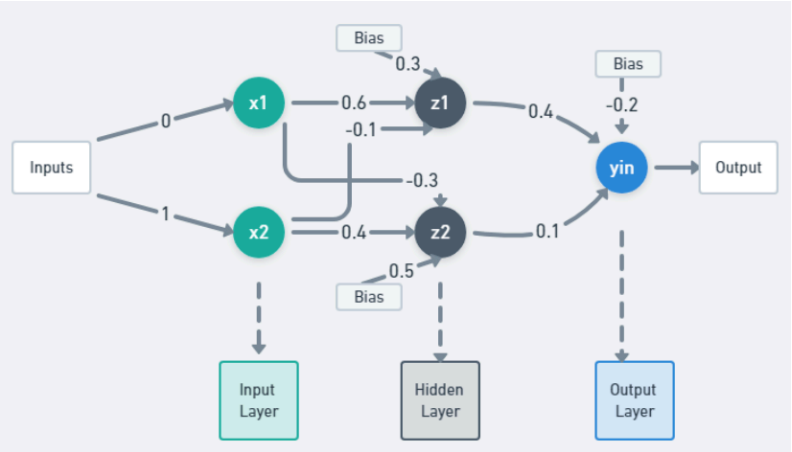
$$
\begin{aligned}
z^{[1]} &= W^{[1]}x + b^{[1]} \\
a^{[1](i)} &= \tanh\left(z^{[1]}\right) \\
z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\
\hat{y} &= a^{[2]} = \sigma\left(z^{[2]}\right) \\
y_{\text{prediction}} &= \begin{cases} 1 & \text{if } a^{[2]} > 0.5 \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

Cost Function:

The cost function of the above model will pertain to the cost function used with logistic regression. Hence, in this tutorial we will be using the cost function:

$$L = Y * \log\left(Y\_pred\right) + (1 - Y) * \log\left(1 - Y\_pred\right)$$

5. Understanding Multi-Layer Feed Forward Networks



Backpropagation Network (BPN)

The provided information explains the workings of a simple multi-layer feed-forward neural network, specifically a backpropagation network. The network consists of an input layer with two neurons (x1 and x2), a hidden layer with two neurons (z1 and z2), and an output layer with one neuron (yin).

The weights and bias vectors for each neuron are initialized randomly. The conditions for backpropagation networks include using a differentiable activation function, inputting a bias of 1, and following certain constraints.

The process involves three steps:

1. Computing the Output (y): The output is calculated by finding yin and applying the activation function. The values of z1 and z2 in the hidden layer are calculated using the sigmoid function. The final output is then determined using these values.

2. Backpropagation of Errors: The error between the output and hidden layer (δk) and between the hidden and input layer (δj) is calculated. The errors are then used to update the weights.

3. Updating Weights: The weights for the output layer (wjk) and hidden layer (vij) are updated using the calculated errors and the learning rate.

6. ML – List of Deep Learning Layers

To specify the architecture of a neural network with all layers connected sequentially, create an array of layers directly. To specify the architecture of a network where layers can have multiple inputs or outputs, use a LayerGraph object. Use the following functions to create different layer types.

Input Layers:

1. imageInputLayer:
   - This layer is designed for inputting images into a neural network.
   - It includes functionality for applying data normalization to the input images.
   - Image normalization is a common preprocessing step that helps in training neural networks by scaling pixel values to a standard range.

2. sequenceInputLayer:

- Specifically tailored for handling sequential data input in a neural network.
- It is suitable for tasks involving sequences, time series, or other ordered data.
- Provides a way to input and handle sequential information efficiently.

Learnable Layers:

1. convolution2dLayer:
   - Applies sliding filters to the input.
   - Convolves the input by moving filters along the input vertically and horizontally, computing the dot product of weights and input, and adding a bias term.

2. transposedConv2dLayer:
   - Upsamples feature maps.
   - Performs transposed convolution, which helps in upsampling feature maps.

3. fullyConnectedLayer:
   - Multiplies the input by a weight matrix.
   - Adds a bias vector to the result.

4. lstmLayer:
   - A recurrent neural network (RNN) layer designed for time series and sequence data.
   - Performs additive interactions, facilitating improved gradient flow over long sequences during training.
   - Well-suited for learning long-term dependencies in sequential data.

Activation Layers:

1. reluLayer:
   - Performs a threshold operation on each element of the input.
   - Sets any value less than zero to zero.

2. leakyReluLayer:
   - Performs a threshold operation.
   - Multiplies any input value less than zero by a fixed scalar, allowing a small, non-zero gradient for negative inputs.

3. clippedReluLayer:
   - Performs a threshold operation.
   - Sets any input value less than zero to zero.

- Caps any value above a specified ceiling to that ceiling value.

Normalization and Dropout Layers:

1. batchNormalizationLayer:
   - Normalizes each input channel across a mini-batch.
   - Involves subtracting the mini-batch mean and dividing by the mini-batch standard deviation.
   - Shifts the normalized input by a learnable offset and scales it by a learnable scale factor.
   - Ideal for use between convolutional layers and nonlinearities (e.g., ReLU layers) to enhance training speed and reduce sensitivity to network initialization.

2. crossChannelNormalizationLayer:
   - Performs channel-wise normalization.
   - Applied across channels, normalizing the response of different neurons to stimuli.
   - Useful for enhancing the generalization capabilities of the model.

3. dropoutLayer:
   - Randomly sets input elements to zero with a specified probability during training.
   - Introduces a form of regularization by preventing co-adaptation of neurons.
   - Helps improve the model's robustness and generalization by reducing overfitting.

Pooling layers

1. averagePooling2dLayer:
   - Performs down-sampling by dividing the input into rectangular pooling regions.
   - Computes the average values of each region.
   - Helps reduce the spatial dimensions of the input while retaining essential information.

2. maxPooling2dLayer:
   - Performs down-sampling by dividing the input into rectangular pooling regions.
   - Computes the maximum value of each region.
   - Reduces spatial dimensions and retains the most prominent features from each region.

3. maxUnpooling2dLayer:
   - Unpools the output of a max-pooling layer.
   - Useful for retrieving the original spatial dimensions after down-sampling.
   - Performs the reverse operation of max pooling, allowing for upsampling based on the previously stored max values.

combination layers :

1. additionLayer:
   - Performs element-wise addition of multiple inputs.
   - The layer takes inputs named 'in1', 'in2', ..., 'inN', where N is the number of inputs.
   - The number of inputs must be specified when creating the layer.
   - All inputs to the addition layer must have the same dimensions.

2. depthConcatenationLayer:
   - Takes multiple inputs that share the same height and width.
   - Concatenates the inputs along the third dimension (depth).
   - Useful for combining information from multiple sources or branches in a neural network.
   - Requires inputs with consistent height and width, but they can have different depth dimensions.

output layers :

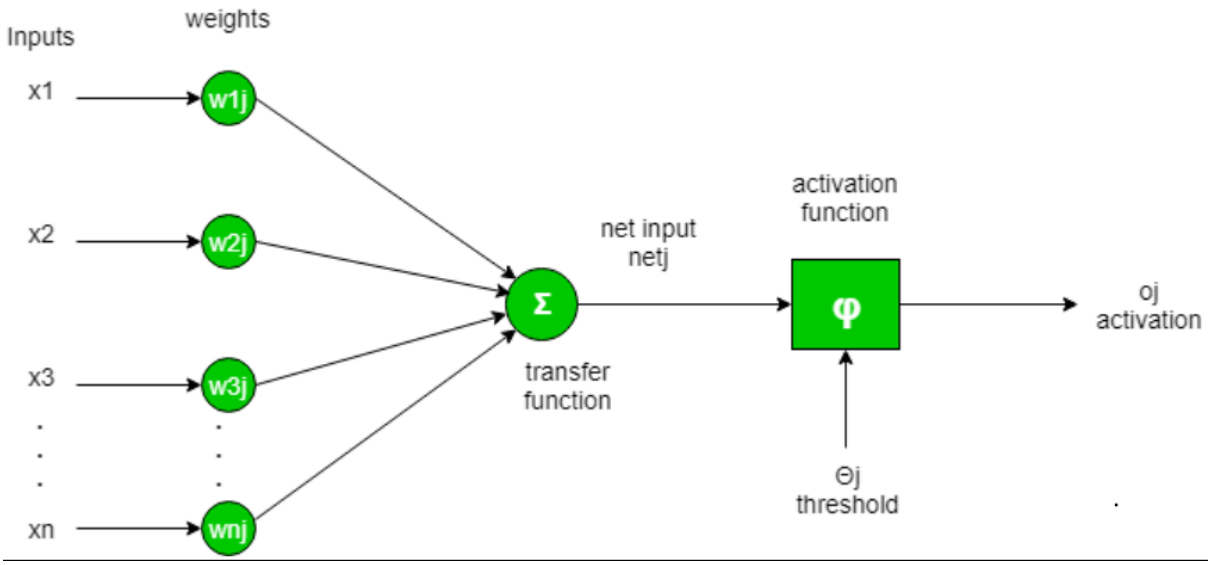
1. softmaxLayer:
   - Applies a softmax function to the input.
   - Often used for multiclass classification problems.
   - Converts the raw output scores into probabilities, making them suitable for classification.

2. classificationLayer:
   - Holds the name of the loss function used for training the network in multiclass classification.
   - Specifies the loss function used during the training process to optimize the model for classification tasks.

3. regressionLayer:
   - Holds the name of the loss function used for training the network in regression tasks.
   - Also contains information about response names.
   - Designed for training networks to predict continuous numerical values.

chapter 3 : Activation Functions

1. introduction



$$\text{net input} = \sum_{(\text{weight} * \text{input})+\text{bias}}$$

Now the value of net input can be any anything from -inf to +inf. The neuron doesn't really know how to bound to value and thus is not able to decide the firing pattern. Thus the activation function is an important part of an artificial neural network. They basically decide whether a neuron should be activated or not. Thus it bounds the value of the net input.

The activation function is a non-linear transformation that we do over the input before sending it to the next layer of neurons or finalizing it as output

Types of Activation Functions

Several different types of activation functions are used in Deep Learning. Some of them are explained below:

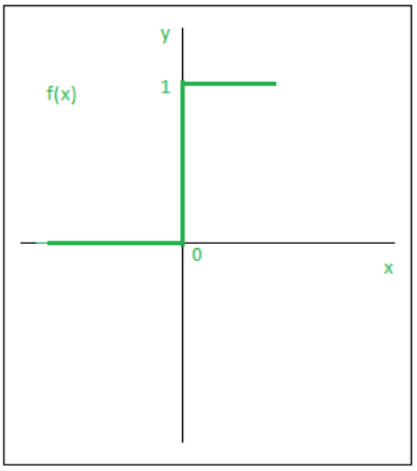
Step Function:

Step Function is one of the simplest kind of activation functions. In this, we consider a threshold value and if the value of net input say y is greater than the threshold then the neuron is activated.

Mathematically,

$$f(x) = 1, \text{if x} >= 0 \qquad f(x) = 0, \text{if x} < 0$$

Given below is the graphical representation of step function.

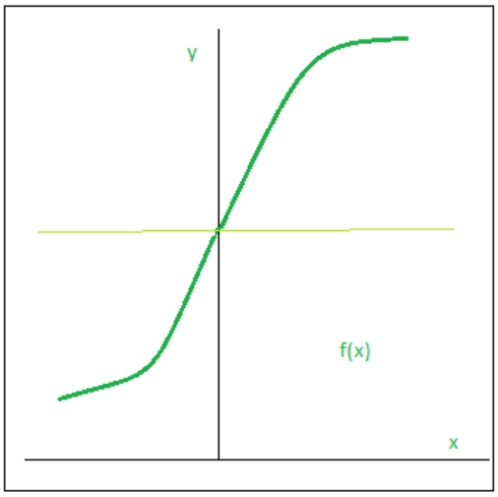


Sigmoid Function:

Sigmoid function is a widely used activation function. It is defined as:

$$\frac{1}{(1+e^{-x})}$$

Graphically,



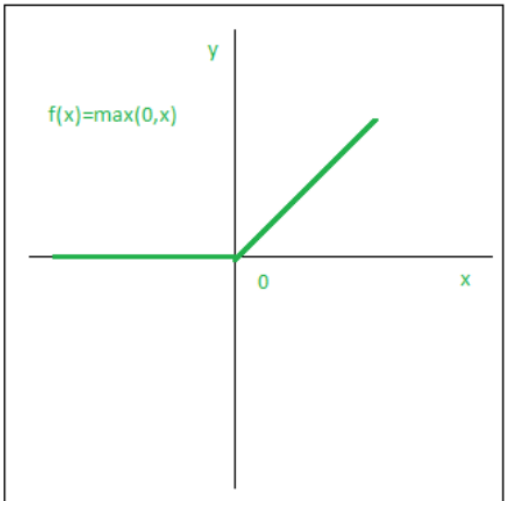
This is a smooth function and is continuously differentiable. The biggest advantage that it has over step and linear function is that it is non-linear. This is an incredibly cool feature

of the sigmoid function. This essentially means that when I have multiple neurons having sigmoid function as their activation function – the output is non linear as well. The function ranges from 0-1 having an S shape.

ReLU:

The ReLU function is the Rectified linear unit. It is the most widely used activation function. It is defined as:

$$f(x) = \max(0, x)$$

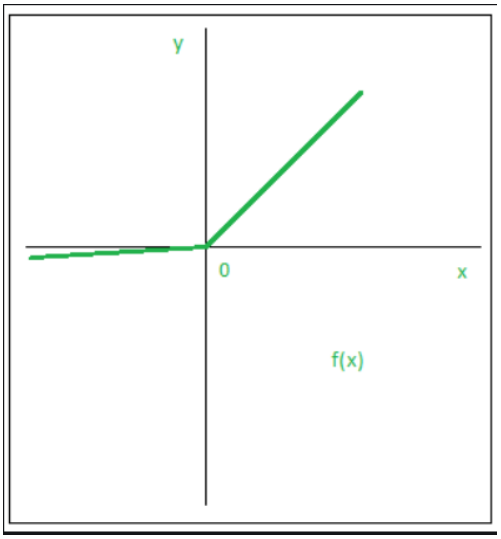


The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time. What does this mean ? If you look at the ReLU function if the input is negative it will convert it to zero and the neuron does not get activated.

Leaky ReLU:

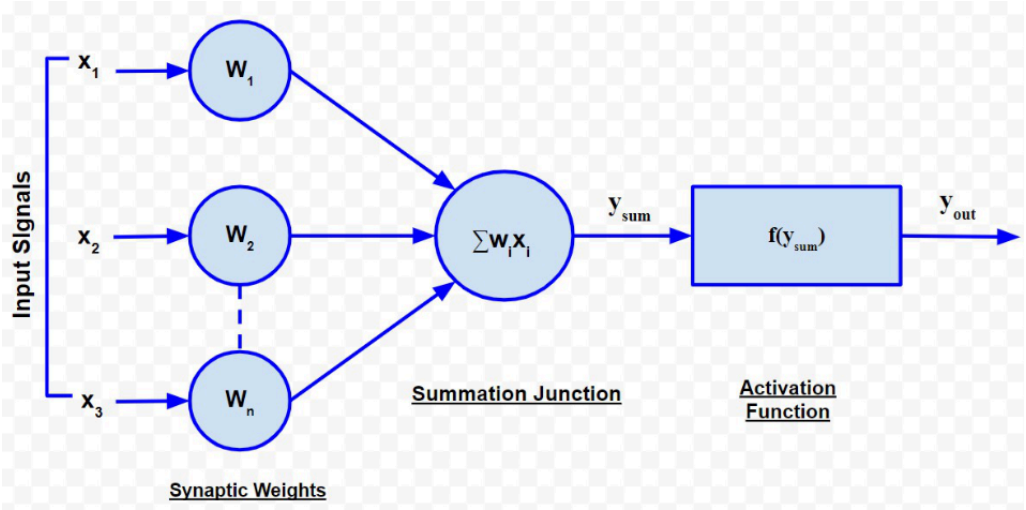
Leaky ReLU function is nothing but an improved version of the ReLU function.Instead of defining the Relu function as 0 for x less than 0, we define it as a small linear component of x. It can be defined as:

$$f(x) = ax, x < 0 \quad f(x) = x, \text{otherwise}$$



2. Types Of Activation Function in ANN

The biological neural network has been modeled in the form of Artificial Neural Networks with artificial neurons simulating the function of a biological neuron. The artificial neuron is depicted in the below picture:



Structure of an Artificial Neuron

Each neuron consists of three major components:

A set of 'i' synapses having weight wi. A signal xi forms the input to the i-th synapse having weight wi. The value of any weight may be positive or negative. A positive weight has an extraordinary effect, while a negative weight has an inhibitory effect on the output of the summation junction.
A summation junction for the input signals is weighted by the respective synaptic weight. Because it is a linear combiner or adder of the weighted input signals

A threshold activation function (or simply the activation function, also known as squashing function) results in an output signal only when an input signal exceeding a specific threshold value comes as an input. It is similar in behaviour to the biological neuron which transmits the signal only when the total input signal meets the firing threshold.
Types of Activation Function :

There are different types of activation functions. The most commonly used activation function are listed below:
A. Identity Function: Identity function is used as an activation function for the input layer. It is a linear function having the form

As obvious, the output remains the same as the input.

B. Threshold/step Function: It is a commonly used activation function. As depicted in the diagram, it gives 1 as output of the input
is either 0 or positive. If the input is negative, it gives 0 as output. Expressing it mathematically,

The threshold function is almost like the step function, with the only difference being a fact that \theta is used as a threshold value instead of . Expressing mathematically,

y

This means that f(x) is zero when x is less than zero and f(x) is equal to x when x is above or equal to zero. This function is differentiable, except at a single point x = 0. In that sense, the derivative of a ReLU is actually a sub-derivative.

D. Sigmoid Function: It is by far the most commonly used activation function in neural networks. The need for sigmoid function stems from the fact that many learning algorithms require the activation function to be differentiable and hence continuous. There are two types of sigmoid function:

1. Binary Sigmoid Function

Chapter 4 : Artificial Neural Network

Artificial Neural Networks (ANNs) are a class of machine learning models inspired by the structure and function of biological neural networks in the human brain. ANNs are composed of interconnected artificial neurons, which are organized into layers. These networks are used for various tasks, including classification, regression, and pattern recognition, and have achieved remarkable success in a wide range of applications, such as image recognition, natural language processing, and reinforcement learning.

1. Cost function in neural networks

Definition and Purpose: The cost function, also known as a loss function, is a mathematical formula used to measure the difference between the predicted output of the neural network and the actual output. It quantifies the error of the model, providing a metric to guide the training process.

Types of Cost Functions: Different types of neural networks and problems require different cost functions. Common examples include:

a. Mean Squared Error (MSE) for regression problems.
b. Cross-Entropy Loss for classification problems.
c. Hinge Loss, often used in Support Vector Machines (SVMs) and also applicable in neural networks for classification tasks.

Role in Backpropagation: The cost function is crucial in the backpropagation algorithm. The gradient of the cost function with respect to the network's weights is computed to update the weights. This process is iteratively repeated, reducing the cost function value, which ideally leads to a more accurate model.

Choosing a Cost Function: The choice of a cost function depends on several factors, such as the type of machine learning problem (regression, classification, etc.), the distribution of the data, and the desired properties of the model (like robustness to outliers).

Impact on Model Performance: The choice and behavior of the cost function significantly impact the model's learning and performance. For instance, a poorly chosen cost function might lead to slow or suboptimal learning, or not properly penalizing certain types of errors.

2. How does Gradient Descent work

Gradient Descent is a fundamental optimization algorithm used in machine learning and deep learning to minimize a function, particularly a cost function in neural networks. Here's a step-by-step explanation of how it works:

Objective: The primary goal of gradient descent is to find the minimum of a function. In the context of neural networks, this function is usually the cost (or loss) function, which measures the difference between the predicted output and the actual output.

Starting Point: Gradient descent starts with a random initialization of the parameters (weights and biases in a neural network). These initial values are the starting point for the optimization process.

Calculating the Gradient: The gradient of the cost function is computed with respect to each parameter. The gradient is a vector that points in the direction of the steepest increase of the function. By calculating the gradient, the algorithm understands in which direction the function increases or decreases most rapidly.

Moving in the Opposite Direction of the Gradient: To minimize the function, gradient descent takes a step in the opposite direction of the gradient. If the gradient is the direction of the steepest ascent, moving against it leads towards the steepest descent.

Learning Rate: The size of the steps taken in the opposite direction of the gradient is determined by the learning rate. A higher learning rate makes larger steps, which could speed up the process but risk overshooting the minimum. A smaller learning rate makes finer adjustments but can slow down the convergence.

Iterative Process: Gradient descent is an iterative process. In each iteration, the algorithm recalculates the gradient based on the updated parameters and then moves the parameters in the opposite direction of this new gradient.

Convergence: The process repeats until the algorithm reaches a point where the cost function converges to a minimum value, or until a predefined number of iterations are completed. Convergence is often determined by the change in the value of the cost function between iterations being below a small threshold.

Types of Gradient Descent:

- Batch Gradient Descent: Computes the gradient using the entire dataset. This is computationally expensive but leads to stable convergence.
- Stochastic Gradient Descent (SGD): Computes the gradient using a single sample at each iteration. This is faster and can escape local minima but is more erratic.
- Mini-batch Gradient Descent: Strikes a balance between batch and stochastic methods by computing the gradient on a small subset of the data.

Challenges:

- Choosing the right learning rate is critical.
- The algorithm can get stuck in local minima instead of finding the global minimum.
- The shape of the cost function (e.g., having many plateaus) can make convergence slow.

Gradient descent is widely used because of its simplicity and effectiveness in optimizing a wide range of functions, particularly in the field of machine learning.

3. Vanishing or Exploding Gradients Problems

Vanishing and exploding gradients are significant challenges in training deep neural networks, particularly when using gradient-based optimization methods like gradient descent. These problems often occur in networks with many layers (deep networks) and can significantly hinder the learning process.

Vanishing Gradients Problem:

Description: In the vanishing gradients problem, the gradients of the network's loss function become increasingly small as they are propagated back through each layer during training. This occurs especially when using activation functions like the sigmoid or tanh.
Consequences: As a result, the weights in the earlier layers of the network are updated very slowly or not at all, making it difficult to train deep networks effectively. This leads to prolonged training times and often results in suboptimal performance.
Causes: It often occurs due to the multiplication of gradients through the network layers, where the gradients are less than 1.0. This multiplication can exponentially decrease the gradient's magnitude as it backpropagates through the layers.

Exploding Gradients Problem:

Description: The exploding gradients problem is the opposite of the vanishing gradients issue. Here, the gradients of the network's loss function become excessively large. This can happen when the derivatives of the activation functions used in the network are greater than 1.
Consequences: Large gradients can lead to huge updates to the network weights, causing the model to diverge and making it unstable. In extreme cases, it can lead to numerical overflow and result in NaN values.
Causes: Like the vanishing gradients, this problem is also related to the depth of the network and the multiplication of gradients. However, in this case, the gradients are larger than 1.0, causing the exponential increase in gradient values as they are propagated back.

Solutions:

Activation Functions: Using activation functions like ReLU (Rectified Linear Unit) and its variants (like Leaky ReLU, Parametric ReLU, etc.) can mitigate these problems, as they do not saturate in the same way as sigmoid or tanh.
Weight Initialization: Proper initialization of weights can help in preventing vanishing and exploding gradients. Techniques like Xavier/Glorot or He initialization are designed to keep the gradients in a reasonable range.
Batch Normalization: Applying batch normalization normalizes the input of each layer, which can help maintain the gradients within a stable range throughout the training.
Gradient Clipping: In the case of exploding gradients, gradient clipping involves scaling down gradients when they exceed a certain threshold, which prevents excessively large updates to the network's parameters.
Skip Connections: Architectures like ResNet introduce skip connections, which help in mitigating the vanishing gradient problem by allowing gradients to flow through alternate paths.
Use of LSTM/GRU in RNNs: In recurrent neural networks, Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) architectures are designed to avoid the vanishing gradient problem, making them effective for learning long-range dependencies.

Addressing the vanishing and exploding gradients issues is crucial for training deep neural networks effectively and achieving good performance, especially in complex tasks that require learning long-range dependencies or using deep architectures.

4. Choose the optimal number of epochs

Choosing the optimal number of epochs in deep learning is a crucial step to strike the right balance between underfitting and overfitting your model. Overfitting occurs when your model learns to fit the training data too well and performs poorly on unseen data, while underfitting occurs when your model hasn't learned enough from the training data and performs poorly on both training and test data. Here are several techniques to help you determine the optimal number of epochs:

Validation Curve:

a. Split your dataset into training and validation sets.
b. Train your model for a range of epochs (e.g., 10, 20, 30, ...), and monitor the performance (e.g., loss or accuracy) on the validation set at each epoch.
c. Plot the validation performance as a function of the number of epochs.
d. Look for the point where the validation performance starts to level off or degrade; this is a good indication of the optimal number of epochs.

Early Stopping:

e. Implement early stopping by monitoring the validation performance during training.
f. Define a patience parameter that represents the number of epochs to wait before stopping training when the validation performance doesn't improve.
g. Stop training when the validation performance hasn't improved for a specified number of epochs.
h. The epoch at which training stops can be considered the optimal number of epochs.

Learning Rate Scheduling:

i. Use learning rate schedules like learning rate annealing or learning rate decay.
j. Start with a relatively high learning rate and decrease it during training.
k. The number of epochs at which you decrease the learning rate can affect the optimal number of epochs.

Cross-Validation:

l. If you have a small dataset, consider using k-fold cross-validation.

m. Split your data into k subsets, train and validate your model on different subsets in each fold.
n. Compute the average performance across all folds for different numbers of epochs and choose the one with the best performance.

Monitor Training and Validation Curves:

o. Keep a close eye on both the training and validation loss curves.
p. If the training loss continues to decrease while the validation loss starts increasing, your model may be overfitting, and you should stop training.

Experiment and Iterate:

q. Don't be afraid to experiment with different numbers of epochs.
r. Start with a reasonable range and fine-tune the number of epochs based on your observations.

Use Pretrained Models:

s. If applicable, consider using pretrained models as a starting point.
t. Fine-tuning a pretrained model may require fewer epochs than training from scratch.

Regularization Techniques:

u. Apply regularization techniques like dropout or L1/L2 regularization to help reduce overfitting, potentially allowing you to train for more epochs.

5. Batch Normalization in Deep Learning

Batch Normalization (BatchNorm or BN) is a technique commonly used in deep learning to improve the training and performance of deep neural networks. It was introduced by Sergey Ioffe and Christian Szegedy in their 2015 paper titled "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift."

Batch Normalization is primarily applied to the layers within a neural network, typically before the activation function. It normalizes the input to a layer across the mini-batch during training. Here's how Batch Normalization works:

Mini-Batch Statistics: During each forward pass in training, BatchNorm calculates the mean and standard deviation of the input values for each feature across the mini-batch.

Normalization: It normalizes the input values of each feature by subtracting the mean and dividing by the standard deviation. This centers and scales the input data, making it have a mean of zero and a standard deviation of one.

Scaling and Shifting: After normalization, the data is scaled using learnable parameters (gamma) and shifted using another set of learnable parameters (beta). This step allows the model to learn the optimal scale and shift for each feature, which helps maintain representational capacity.

Benefits of Batch Normalization:

Accelerated Training: BatchNorm reduces internal covariate shift, which helps stabilize and speed up the training process. It allows for higher learning rates and can lead to faster convergence.

Regularization: BatchNorm acts as a form of regularization by adding noise to the training process, which can help prevent overfitting.

Improved Gradient Flow: By normalizing the inputs, BatchNorm can mitigate issues like vanishing and exploding gradients, making it easier to train deep networks.

Reduced Sensitivity to Initialization: Models with BatchNorm are less sensitive to the choice of initial weights, making it easier to initialize and train deep networks effectively.

Batch Normalization is typically applied to convolutional layers and fully connected layers in deep neural networks, although it may not be suitable for all types of neural architectures. It is often used in conjunction with other regularization techniques like dropout and weight decay to further enhance the model's performance and generalization.

6. Difference between Sequential and functional API

In Keras, a popular deep learning library, there are two primary ways to build neural network models: the Sequential API and the Functional API. These two approaches have some key differences and are suited to different use cases.

Sequential API:

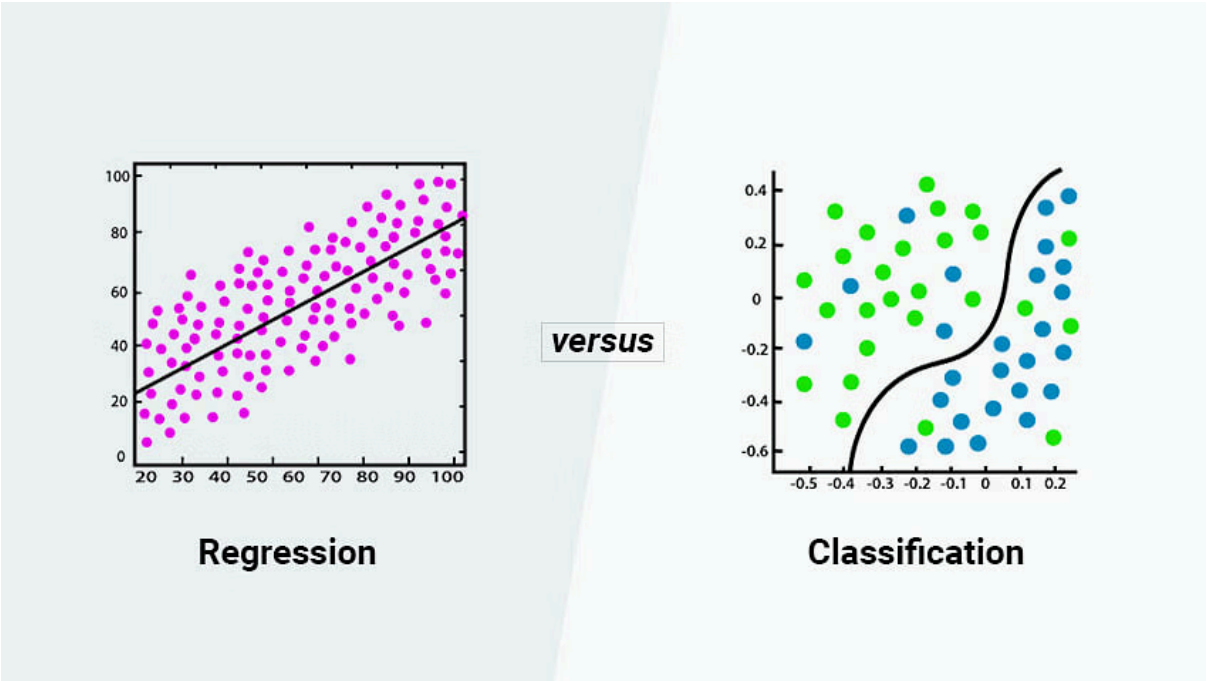
- Simplicity: The Sequential API is straightforward and easy to use, making it ideal for building simple, linear stack models where layers are added sequentially one after another.
- Limited Architectural Flexibility: Sequential models are limited to linear, feedforward architectures. You can't create models with multiple inputs, multiple outputs, or complex branching or sharing of layers.
- Code Simplicity: If your model can be represented as a single sequence of layers from input to output, the Sequential API results in concise and easy-to-read code.

Functional API:

- Flexibility: The Functional API offers much greater flexibility and allows you to create complex, non-linear architectures with multiple inputs, multiple outputs, shared layers, and even models with branches or skips connections.
- Multiple Inputs and Outputs: You can build models that take multiple inputs and produce multiple outputs, which is essential for tasks like siamese networks, multi-input/multi-output models, and more.
- Shared Layers: The Functional API lets you define and reuse layers, which is particularly useful when you want to share weights between layers in different parts of your model.
- Custom Models: With the Functional API, you can create custom models by specifying the connections between layers explicitly, making it suitable for a wide range of deep learning architectures.

The choice between the Sequential API and the Functional API depends on the complexity and architecture of your neural network. If you're building a simple, sequential model, the Sequential API is straightforward and efficient. However, if your model has multiple inputs, outputs, shared layers, complex branching, or requires custom connections, the Functional API provides the necessary flexibility and control.

7. Classification & Regression



Classification and regression are two fundamental types of supervised machine learning tasks that involve predicting a target variable based on input features. They serve different purposes and are used in various applications.

Classification:

- Classification is a supervised learning task where the goal is to assign a predefined category or label to an input data point.
- It is used when the target variable is categorical or consists of discrete classes or labels. Examples include spam detection (classifying emails as spam or not), image classification (identifying objects in images), and sentiment analysis (categorizing text as positive, negative, or neutral).
- Common algorithms for classification include decision trees, random forests, support vector machines, k-nearest neighbors, and deep neural networks.

Regression:

- Regression is a supervised learning task where the goal is to predict a continuous numerical value as the output.
- It is used when the target variable is quantitative and not discrete. Examples include predicting house prices based on features like square footage, predicting stock prices, and estimating the age of a person based on certain characteristics.

- Common algorithms for regression include linear regression, polynomial regression, support vector regression, and various forms of neural networks.

Classification aims to categorize data into predefined classes or labels, while regression focuses on predicting numerical values. Both classification and regression are essential tools in machine learning and are applied in a wide range of real-world problems, from image recognition and natural language processing to financial forecasting and healthcare diagnostics. The choice between classification and regression depends on the nature of the target variable and the problem at hand.

8. Fine-Tuning & Hyperparameters

Fine-tuning and hyperparameters are crucial concepts in machine learning, particularly in training models to achieve better performance. Let's delve into each of these concepts:

Fine-Tuning:

Fine-tuning refers to the process of adjusting an already pre-trained machine learning model to perform better on a specific task or dataset. It is particularly common in transfer learning, where you take a pre-trained model (often on a large dataset) and adapt it to a new task.

The idea behind fine-tuning is that pre-trained models have already learned useful features from the original data, and you can leverage these learned features to improve performance on a related task with a smaller dataset.

Fine-tuning typically involves:

a. Removing the last few layers of the pre-trained model, which were designed for the original task.

b. Adding new layers tailored to the specific task.

c. Re-training the model on the new task-specific dataset while keeping the earlier layers frozen (not updated).

d. Optionally adjusting the learning rate and other hyperparameters.

Fine-tuning is commonly used in areas like computer vision (e.g., using pre-trained convolutional neural networks for object detection), natural language processing (e.g., fine-tuning pre-trained language models for sentiment analysis), and more.

Hyperparameters:

- Hyperparameters are parameters that are not learned from the training data but are set prior to training and are essential for controlling the learning process of a machine learning model.
- Examples of hyperparameters include the learning rate, the number of hidden layers and units in a neural network, the depth of a decision tree, the regularization strength, and batch size, among others.
- Choosing the right hyperparameters is crucial because they can significantly impact the model's performance. Suboptimal hyperparameters can lead to underfitting or overfitting.
- Hyperparameter tuning, also known as hyperparameter optimization or grid search, is the process of systematically searching for the best combination of hyperparameters to achieve optimal model performance. This can be done through techniques like random search, grid search, or Bayesian optimization.
- Tools and libraries like scikit-learn, TensorFlow, Keras, and PyTorch provide ways to tune hyperparameters and search for the best settings automatically.

Fine-tuning is the process of adapting a pre-trained model for a specific task, while hyperparameters are configuration settings that control how a machine learning model learns during training. Properly fine-tuning models and tuning hyperparameters are essential steps in achieving high-performance machine learning models on various tasks and datasets.

Introduction to Convolution Neural Network

1- Definition :

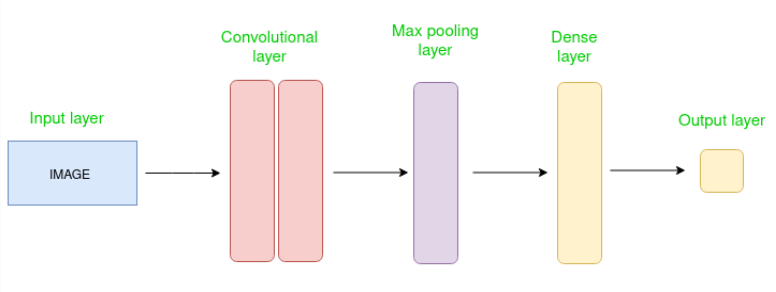
A **Convolutional Neural Network (CNN)** is a type of Deep Learning neural network architecture commonly used in Computer Vision. Computer vision is a field of Artificial Intelligence that enables a computer to understand and interpret the image or visual data.

A typical Neural Network comprises three key layers:

- **Input Layers**: Receive the initial data input. Neuron count aligns with total data features (e.g., pixels in an image).
- **Hidden Layers**: Intermediate layers where data is processed. They may vary in number and neurons, performing computations using weighted connections and applying activation functions for nonlinearity.
- **Output Layer**: Produces final results by using functions like sigmoid or softmax to convert outputs into probability scores for respective classes.

Data enters the model, generating output through feedforward. Error is computed using functions like cross-entropy or square loss to assess performance. Backpropagation, involving derivative calculations, minimizes the loss during training.

2- CNN architecture



The Convolutional layer applies filters to the input image to extract features, the Pooling layer downsamples the image to reduce computation, and the fully connected layer makes the final prediction. The network learns the optimal filters through backpropagation and gradient descent.

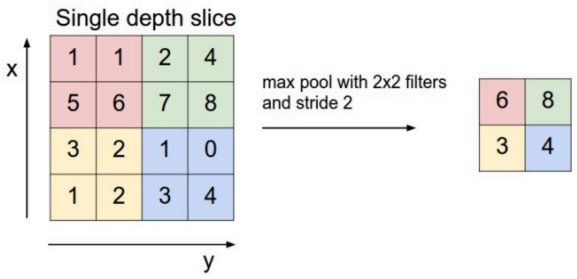
3- How Convolutional Layers works

- Convolutional Neural Networks CNN employ shared parameters.
- An image is represented as a cuboid with length, width (image dimensions), and height (channels like red, green, blue).
- A small filter/kernel processes image patches to produce K outputs vertically.
- The filter slides across the image, generating a new image with varied dimensions and depths.
- This process is convolution, altering the image's channels and dimensions.
- Convolution involves smaller weights due to the filter's patch size.

- Mathematically, CNN uses learnable filters of small widths, heights, and the same depth as the input volume .
- During the forward pass, filters slide across the input volume with strides (2, 3, or 4 for high-dimensional images), computing dot products with kernel weights and input patches.
- Sliding filters yield 2-D outputs stacked together, forming an output volume with a depth equal to the number of filters. The network learns these filters.

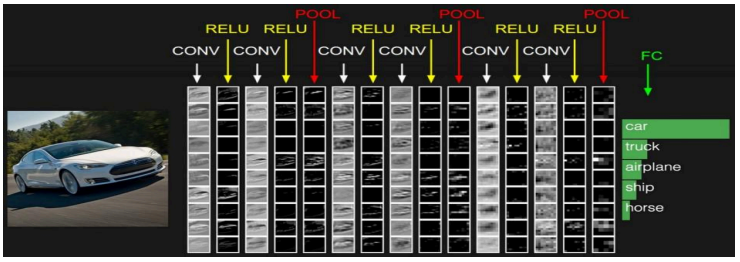
4- Layers used to build ConvNets

- Convolutional Neural Networks, also called covnets, consist of layers transforming volumes via differentiable functions.
- **Input Layers**: Receive image input, typically 32x32x3 in CNN, representing raw image data.
- **Convolutional Layers**: Extract features using learnable filters/kernels (e.g., 2x2, 3x3, 5x5 matrices). These filters slide over input images, computing dot products with image patches, generating feature maps. Using 12 filters results in a 32x32x12 output volume.
- **Activation Layer**: Applies activation functions (e.g., ReLU, Tanh, Leaky ReLU) element-wise, introducing nonlinearity. Output remains the same size: 32x32x12.
- **Pooling Layer**: Periodically reduces volume size, aiding computation speed, reducing memory usage, and preventing overfitting. Common types include max pooling and average pooling. A 2x2 max pooling layer with stride 2 yields a resultant volume of 16x16x12.



- **Flattening:** The resulting feature maps are flattened into a one-dimensional vector after the convolution and pooling layers so they can be passed into a completely linked layer for categorization or regression.
- **Fully Connected Layers:** It takes the input from the previous layer and computes the final classification or regression task.

Example :



- **Output Layer**: The output from the fully connected layers is then fed into a logistic function for classification tasks like sigmoid or softmax which converts the output of each class into the probability score of each class.

5- What are some common activation functions used in CNNs?

- **Rectified Linear Unit (ReLU)**: ReLU is a non-saturating activation function that is computationally efficient and easy to train.
- **Leaky Rectified Linear Unit (Leaky ReLU)**: Leaky ReLU is a variant of ReLU that allows a small amount of negative gradient to flow through the network. This can help to prevent the network from dying during training.
- **Parametric Rectified Linear Unit (PReLU)**: PReLU is a generalization of Leaky ReLU that allows the slope of the negative gradient to be learned.

6- What are some common regularization techniques used in CNNs?

- **Dropout**: Randomly deactivates neurons during training, encouraging the network to learn robust features independent of any single neuron.
- **L1 Regularization**: Penalizes the absolute value of weights, potentially reducing weights and enhancing network efficiency.
- **L2 Regularization**: Penalizes the square of weights, also aiding in weight reduction and improving network efficiency.

7- Advantages of Convolutional Neural Networks (CNNs):

- Good at detecting patterns and features in images, videos, and audio signals.
- Robust to translation, rotation, and scaling invariance.
- End-to-end training, no need for manual feature extraction.
- Can handle large amounts of data and achieve high accuracy.

8- Disadvantages of Convolutional Neural Networks (CNNs):

- Computationally expensive to train and require a lot of memory.
- Can be prone to overfitting if not enough data or proper regularization is used.
- Requires large amounts of labeled data.
- Interpretability is limited, it's hard to understand what the network has learned.

Digital Image Processing Basics

1- Definition :

Digital Image Processing involves using a digital computer to process digital images using algorithms. It employs computer algorithms to enhance images and extract valuable information. The aim is to improve image quality, derive meaningful insights

2- The basic steps involved in digital image processing are:

- **Image acquisition**: Capturing or importing images into a computer via a digital camera or scanner.
- **Image enhancement**: Improving visual quality by adjusting contrast, reducing noise, and eliminating artifacts.
- **Image restoration**: Eliminating degradation like blurring, noise, or distortion from images.
- **Image segmentation**: Dividing images into segments corresponding to specific objects or features.
- **Image representation & description**: Representing images for computer analysis and describing features concisely.
- **Image analysis**: Using algorithms to extract information, recognize objects, detect patterns, and quantify features.
- **Image synthesis & compression**: Creating new images or compressing existing ones for reduced storage and transmission.
- **Applications**: Digital image processing finds widespread use in fields like medical imaging, remote sensing, computer vision, and multimedia.

3- What is an image?

An image is a 2D function represented by F(x,y), where x and y denote spatial coordinates, and F(x,y) signifies the image's intensity at that point. In digital form, an image is a structured two-dimensional

array, composed of finite elements known as pixels, each holding a specific value at a distinct location.

3-1 Types of an image

- BINARY IMAGE
- BLACK AND WHITE IMAGE
- 8 bit COLOR FORMAT
- 16 bit COLOR FORMAT

A 16 bit format is actually divided into three further formats which are Red, Green and Blue. That famous RGB format.

3-2 Image as a Matrix

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & f(0,2) & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & f(1,2) & \dots & f(1,N-1) \\ \vdots & \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & f(M-1,2) & \dots & f(M-1,N-1) \end{bmatrix}$$

4- Advantages of Digital Image Processing:

- Enhanced image quality: Algorithms in digital image processing enhance visual quality, enhancing clarity, sharpness, and information content.
- Automated image tasks: Digital image processing automates tasks like object recognition, pattern detection, and measurements.
- Enhanced efficiency: Algorithms process images swiftly, enabling rapid analysis of extensive data sets.

5- Disadvantages of Digital Image Processing:

- Computational intensity: Some algorithms in digital image processing demand substantial computational resources.
- Interpretation challenges: Complex algorithms may produce results challenging for human interpretation.
- Input-output dependency: Output quality heavily relies on input image quality, impacting the final result.

- Algorithmic limitations: Difficulties exist in recognizing objects in cluttered or poorly lit scenes, or with significant deformations or occlusions.

6- Difference between Image Processing and Computer Vision:

Image Processing	**Computer Vision**
Image processing is mainly focused on processing the raw input images to enhance them or preparing them to do other tasks	Computer vision is focused on extracting information from the input images or videos to have a proper understanding of them to predict the visual input like the human brain.
Image processing uses methods like Anisotropic diffusion, Hidden Markov models, Independent component analysis, Different Filtering etc.	Image processing is one of the methods that is used for computer vision along with other Machine learning techniques, CNN etc.
Image Processing is a subset of Computer Vision.	Computer Vision is a superset of Image Processing.

CNN | Introduction to Pooling Layer

1- What is the Pooling Layer ?

The pooling operation involves sliding a two-dimensional filter over each channel of the feature map and summarizing the features lying within the region covered by the filter.

2- Why use Pooling Layers?

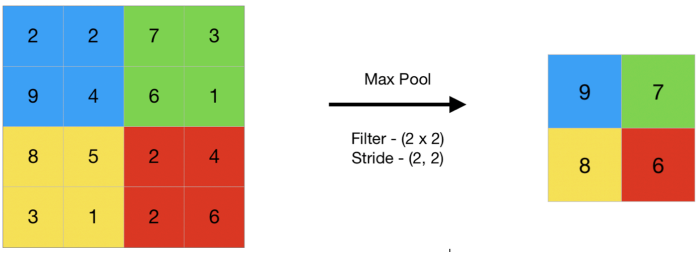
- Pooling layers decrease feature map dimensions, reducing parameters and computational load in the network.

- They summarize features within regions of the convolution layer's feature map. This allows subsequent operations to work on summarized features rather than precise feature positions .

3- Types of Pooling Layers:

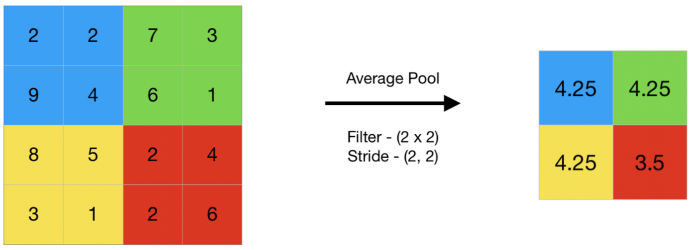
- **Max Pooling**

  Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.



- **Average Pooling**

  Average pooling computes the average of the elements present in the region of the feature map covered by the filter. Thus, while max pooling gives the most prominent feature in a particular patch of the feature map, average pooling gives the average of features present in a patch.



- **Global Pooling**

  Global pooling reduces each channel in the feature map to a single value. Thus, an **nh x nw x nc** feature map is reduced to **1 x 1 x nc** feature map. This is equivalent to using a filter of dimensions **nh x nw** i.e. the dimensions of the feature map.

  Further, it can be either global max pooling or global average pooling.

4- Advantages of Pooling Layer:

- Dimensionality reduction: Pooling layers effectively reduce spatial feature map dimensions, lowering computational load and mitigating overfitting by reducing model parameters.
- Translation invariance: These layers aid in achieving translation invariance within feature maps, ensuring that object position in an image doesn't impact classification results. Consistent features are detected irrespective of object position.
- Feature selection: Max pooling highlights salient features, while average pooling retains more information, assisting in the selection of critical input features.

5- Disadvantages of Pooling Layer:

- Information loss: Pooling layers discard input feature map information, potentially crucial for final classification or regression tasks.
- Over-smoothing: They might excessively smooth feature maps, leading to the loss of fine-grained details critical for classification or regression.
- Hyperparameter tuning: Introducing hyperparameters like pooling region size and stride, demanding optimization for optimal model performance. This process can be time-consuming and requires expertise in model development.

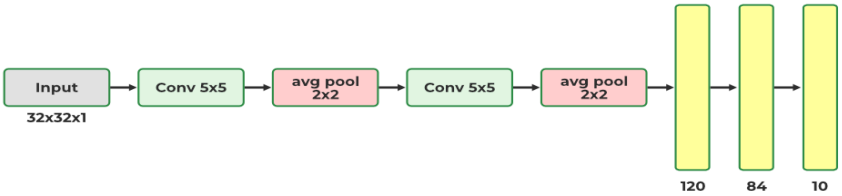
Convolutional Neural Network (CNN) Architectures

Over many years, **CNN architectures have evolved**. Many variants of the fundamental CNN Architecture have been developed, leading to amazing advances in the growing deep-learning field.

Let’s discuss, How CNN architecture developed and grow over time.

1- LeNet-5

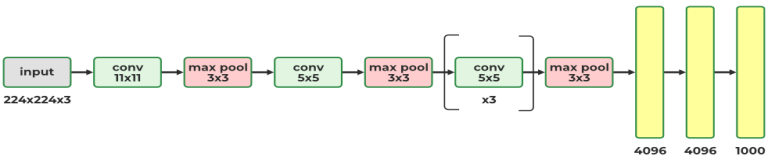
- The First LeNet-5 architecture is the most widely known CNN architecture. It was introduced in 1998 and is widely used for handwritten method digit recognition.
- LeNet-5 has 2 convolutional and 3 full layers.
- This LeNet-5 architecture has 60,000 parameters.



- The LeNet-5 has the ability to process higher one-resolution images that require larger and more CNN convolutional layers.
- The leNet-5 technique is measured by the availability of all computing resources

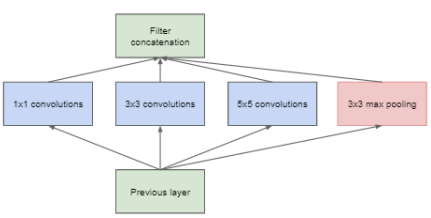
2- AlexNNet

- AlexNet, introduced by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, dominated the 2012 ImageNet ILSVRC challenge with a remarkable 17% top-5 error rate, outperforming competitors.
- It featured a larger and deeper architecture than LeNet-5, stacking convolutional layers directly and employing Rectified Linear Units (ReLUs) as activation functions.
- Notably, AlexNet was the first CNN architecture to harness GPU acceleration, significantly enhancing its performance capabilities.

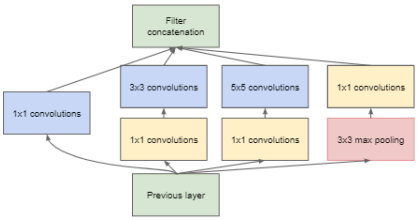


3- GoogleNet (Inception vl)

- The GoogleNet architecture was created by Christian Szegedy from Google Research and achieved a breakthrough result by lowering the top-5 error rate to below 7% in the ILSVRC 2014 challenge.
- GoogleNet has fewer parameters than AlexNet, with a ratio of 10:1 (roughly 6 million instead of 60 million)
- The architecture of the inception module looks as shown in Fig.



(a) Inception module, naïve version

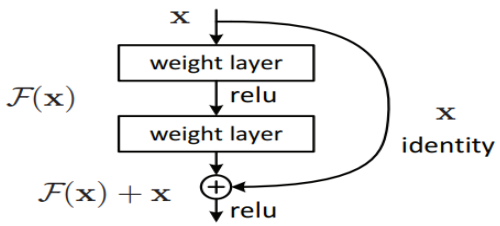


(b) Inception module with dimension reductions

4- ResNet (Residual Network)

- Residual Network (ResNet) won the ILSVRC 2015 challenge with a 3.6% top-5 error rate using a deep CNN of 152 layers, credited to its skip connections.
- These skip connections add the input signal to a higher-layer output, aiding in training very deep networks.
- In training, the network aims to replicate a target function h(x). By using skip connections, ResNet models f(x) = h(x) – x, facilitating residual learning for better function approximation.

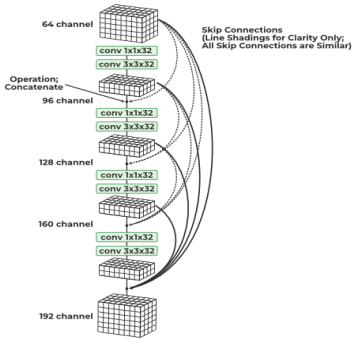
```
F(x) = H(x) - x which gives H(x) := F(x) + x.
```



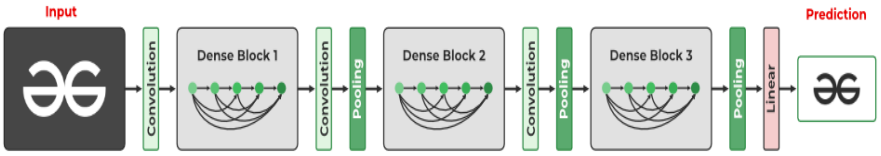
- Regular neural network weights start near zero, resulting in close-to-zero outputs initially.
- Skip connections enable the network to emulate the identity function, benefiting training when the target function aligns with the identity function.
- Multiple skip connections allow progress even when some layers haven't learned yet.
- Deep residual networks are a series of residual units, each functioning as a small neural network with a skip connection.

5- DenseNet

- DenseNet introduced a densely connected convolutional network, where each layer's output connects to the subsequent layers' inputs.
- This design aimed to combat accuracy decline caused by vanishing and exploding gradients in deep neural networks.
- By creating direct connections between layers, DenseNet addresses the issue of data loss over long distances within the network.
- These direct connections help retain information, ensuring better information flow compared to networks with distant connections.



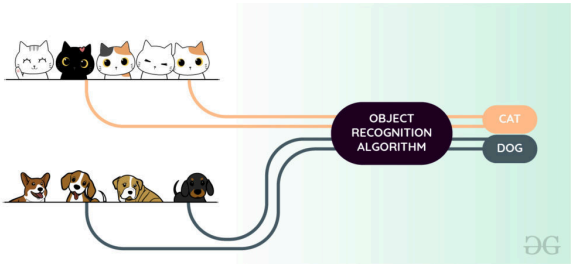
- Convolutions within a dense block use ReLU activation and batch normalization, with stride 1 to enable channel-wise concatenation.
- Pooling layers are inserted between dense blocks to further reduce dimensionality.
- Despite concatenating outputs, DenseNet remains parameter-efficient. It uses 1×1 convolutions after concatenation to reduce channels, followed by economical 3×3 convolutions.
- Each DenseNet step, employing 1×1 and 3×3 convolutions, incrementally adds K channels to the data, maintaining a constant growth rate.
- DenseNet performs well with growth rates (K) ranging from 12 to 40, showing linear channel growth within dense blocks.
- The DenseNet architecture, combining dense blocks and pooling layers, forms a Tu DenseNet network. DenseNet21 has 121 layers but is adjustable and extendable to over 200 layers.



Object Detection vs Object Recognition vs Image Segmentation

1- Object Recognition:

Object recognition is the technique of identifying the object present in images and videos. The goal of this field is to teach machines to understand (recognize) the content of an image just like humans do.



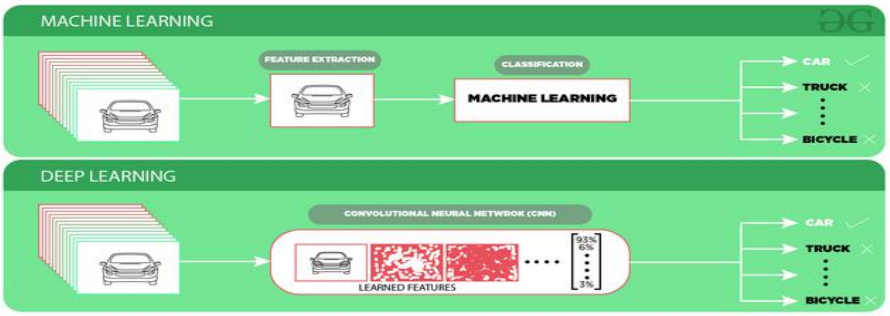
1-2 Object Recognition Using Machine Learning

Before the advent of deep learning, object recognition primarily relied on classical methods:

- HOG (Histogram of Oriented Gradients) and SVM (Support Vector Machine): A prevalent method for object detection, utilizing histogram descriptors from positive and negative image samples to train an SVM model.
- Bag of Features Model: Similar to bag of words for text, it represents images as unordered collections of features like SIFT or MSER.
- Viola-Jones Algorithm: Widely applied for real-time face detection, involving Haar-like feature extraction and boosting classifiers. This algorithm uses a cascade of boosted classifiers to detect faces and has a detection time of 2 fps, suitable for real-time face recognition systems.

1-3 Object Recognition Using Deep Learning

CNN for Object Recognition: CNNs are a popular choice for object recognition tasks like image classification. They take an image as input and provide probabilities for different classes. Higher probabilities indicate the presence of specific objects, while lower probabilities suggest the absence of those objects.



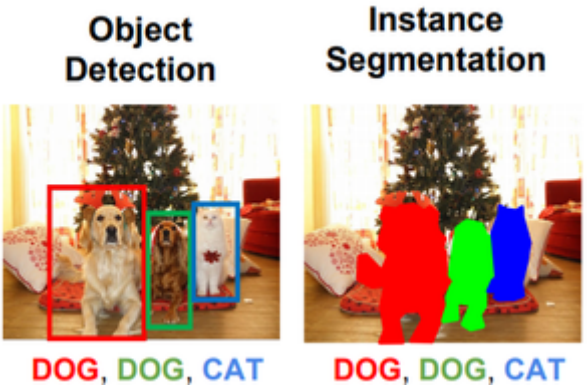
2- Image Classification :

In Image classification, it takes an image as an input and outputs the classification label of that image with some metric (probability, loss, accuracy, etc). For Example: An image of a cat can be classified as a class label “cat” or an image of Dog can be classified as a class label “dog” with some probability.

- Object Localization: Locates objects within images, providing their positions via bounding boxes in terms of position, height, and width.
- Object Detection: Combines image classification and localization by producing bounding boxes with corresponding class labels. Handles multi-class classification, localization, and multiple occurrences of objects within images.

3- Image Segmentation:

Image Segmentation is an advanced method that uses pixel-wise masks to precisely identify object boundaries in an image. This detailed approach is crucial in fields like medical imaging and satellite analysis. Techniques like Mask R-CNN, introduced by K He et al. in 2017, have significantly improved image segmentation methods.



There are primarily two types of segmentation:

- Instance Segmentation: Multiple instances of the same class are separate segments i.e. objects of the same class are treated as different. Therefore, all the objects are coloured with different colors even if they belong to the same class.
- Semantic Segmentation: All objects of the same class form a single classification ,therefore , all objects of the same class are coloured by the same color.

4- Applications:

- Driverless Cars
- Medical Image Processing
- Surveillance and Security

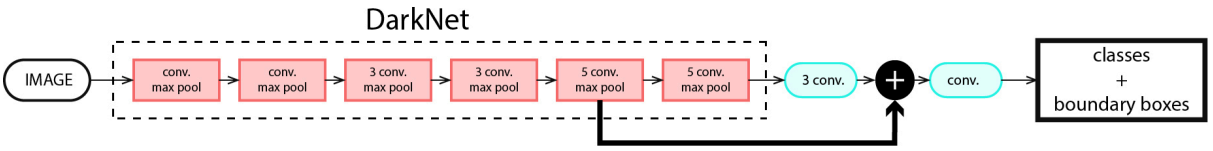
YOLO v2 – Object Detection

1- Definition

In terms of speed, YOLO stands out as it can process frames at a high rate of up to 150 FPS for small networks. However, concerning accuracy, YOLO doesn't achieve state-of-the-art results. It demonstrates a Mean Average Precision (mAP) of 63% when trained on PASCAL VOC2007 and PASCAL VOC 2012. In contrast, the Fast R-CNN, which was leading at that time, achieved an mAP of 71%.

2- Architecture Changes vs YOLOv1:

The previous YOLO architecture faced several issues compared to state-of-the-art methods like Fast R-CNN. It encountered localization errors and had a low recall rate.



2-1 Batch Normalization:

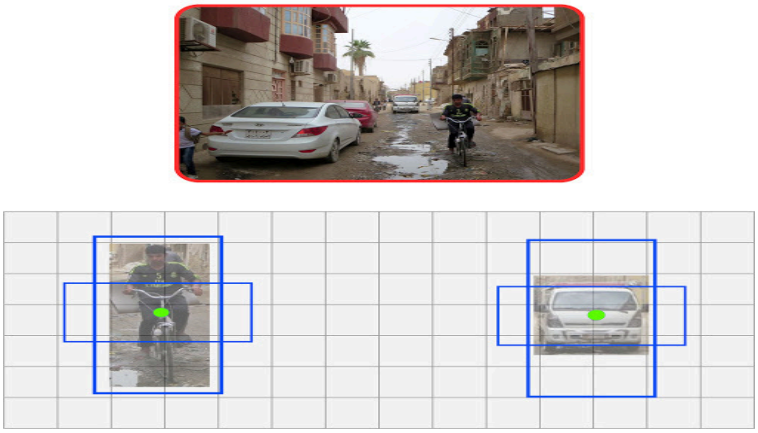
By adding batch normalization to the architecture we can increase the convergence of the model that leads us for faster training. This also eliminates the need for applying other types of normalization such as Dropout without overfitting.

2-2 High Resolution Classifier:

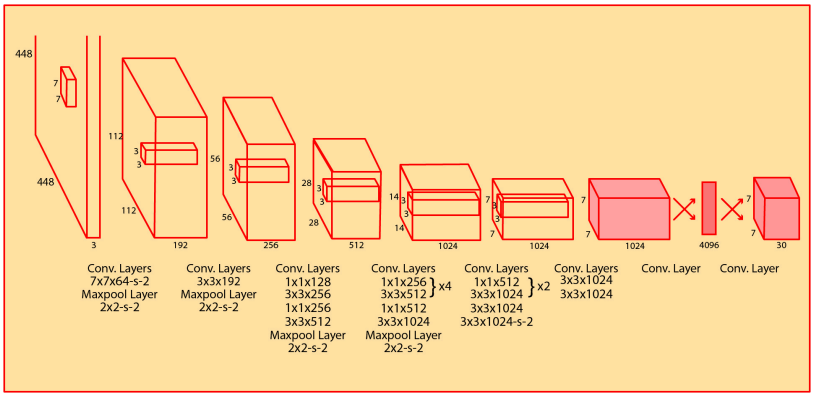
The YOLOv2 version trains on higher resolution *(448 * 448)* for *10* epochs on ImageNet data. This gives the network time to adjust the filters for higher resolution. By training on 448*448 images size the mAP increased by *4%*.

2-3 Use Anchor Boxes For Bounding Boxes:

In this version of yolov2, we remove the fully connected layer and instead add the anchor boxes to predict the bounding boxes. We made the following changes in the architecture:

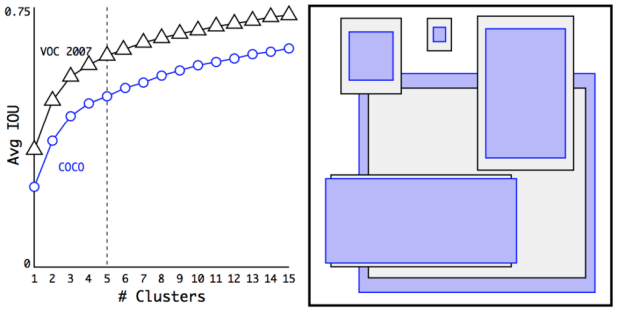


We remove the fully connected layer responsible for predicting bounding boxes and replace it with anchor boxes prediction.



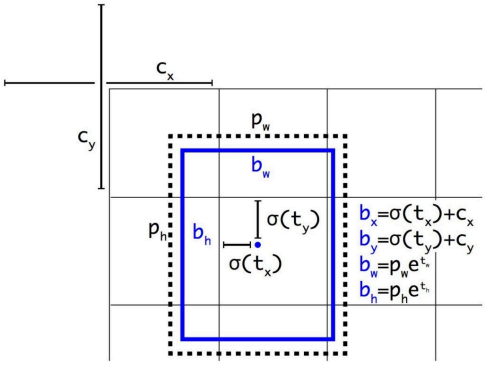
2-4 Dimensionality clusters:

- Determine the optimal number of anchors (priors) denoted as K for maximum accuracy in bounding box identification.
- Utilize K-means clustering, emphasizing IOU maximization over Euclidean distance minimization.
- YOLO v2 opts for K=5, demonstrating balanced algorithm performance.
- Increasing K beyond 5 shows negligible impact on accuracy.
- IOU-based clustering with K=5 achieves a 61% mean Average Precision (mAP).



2-5 Direct Location Problem:

The previous version of YOLO does not have a constraint on location prediction which makes it unstable on early iteration. YOLOv2 predicts 5 parameters (tx, ty, tw, th, to (objectness score)) and applies the sigma function to constraint its value falls between 0 and 1.



2-6 Fine Grained Features :

YOLOv2 which generates *13 * 13* is sufficient for detecting large objects. However, if we want to detect finer objects we can modify the architecture such that the output of previous layer *26 * 26 * 512* to *13 * 13 * 2048* and concatenates with the original *13 * 13 * 1024* output layer making our output layer of size.

3- Architecture:

- YOLO v2 employs training on various architectures like VGG-16, GoogleNet, and its own called Darknet-19.
- Darknet-19 is preferred due to its lower processing requirement (5.58 FLOPS) compared to VGG-16 (30.69 FLOPS) and customized GoogleNet (8.52 FLOPS) for 224x224 image size.
- The Darknet-19 structure entails replacing the final convolution layer with three 3x3 convolution layers, each with 1024 filters, and a subsequent 1x1 convolution layer for detection output.
- For VOC, YOLO v2 predicts 5 boxes, each with 5 coordinates (tx, ty, tw, th, to) and 20 classes per box, amounting to a total of 125 filters.

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

4- Training:

YOLOv2 undergoes two types of training:

- For classification, it trains on ImageNet-1000 for 160 epochs, using Darknet-19 architecture. It starts with a learning rate of 0.1, weight decay of 0.0005, and momentum of 0.9. Standard data augmentation techniques are employed during this training.
- For detection, modifications in the Darknet-19 architecture are implemented. Training for 160 epochs occurs on COCO and VOC datasets, starting with a learning rate of 10^-3, weight decay of 0.0005, and momentum of 0.9.

5- Results and Conclusion:

Detection Frameworks	Train	mAP	FPS
Fast R-CNN [5]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[15]	2007+2012	73.2	7
Faster R-CNN ResNet[6]	2007+2012	76.4	5
YOLO [14]	2007+2012	63.4	45
SSD300 [11]	2007+2012	74.3	46
SSD500 [11]	2007+2012	76.8	19
YOLOv2 288×288	2007+2012	69.0	91
YOLOv2 352×352	2007+2012	73.7	81
YOLOv2 416×416	2007+2012	76.8	67
YOLOv2 480×480	2007+2012	77.8	59
YOLOv2 544×544	2007+2012	**78.6**	40

This model has also been the basis of the YOLO9000 model which is able to detect more than 9000 classes in real-time.

Basis of NLP

1- What is NLP?

NLP stands for Natural Language Processing. It is the branch of Artificial Intelligence that gives the ability to machine understand and process human languages.

- **Heuristics-Based NLP**: This is the initial approach of NLP. It is based on defined rules. Which comes from domain knowledge and expertise.
- **Statistical Machine learning-based NLP:** It is based on statistical rules and machine learning algorithms. In this approach, algorithms are applied to the data and learned from the data, and applied to various tasks.
- **Neural Network-based NLP:** This is the latest approach that comes with the evaluation of neural network-based learning, known as Deep learning.

1-2 Advantages of NLP

- NLP helps us to analyze data from both structured and unstructured sources.
- NLP is very fast and time efficient.
- NLP offers users to ask questions about any subject and give a direct response within milliseconds.

1-3 Disadvantages of NLP

- For the training of the NLP model, A lot of data and computation are required.
- Many issues arise for NLP when dealing with informal expressions, idioms, and cultural jargon.
- NLP results are sometimes not to be accurate, and accuracy is directly proportional to the accuracy of data.

1-4 Components of NLP

There are two components of Natural Language Processing:

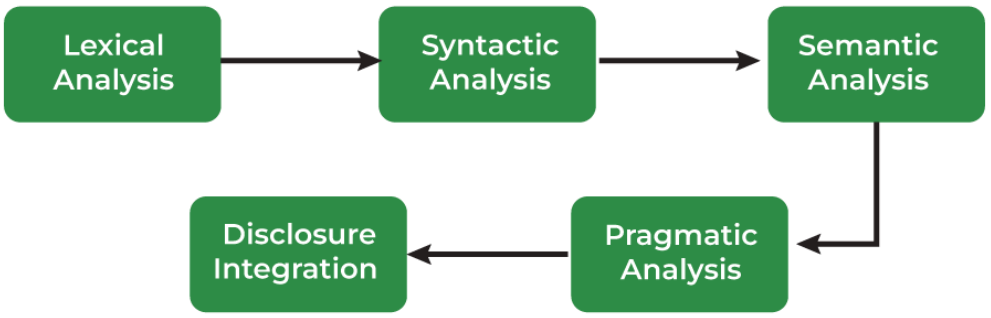
- Natural Language Understanding
- Natural Language Generation

1-5 Applications of NLP

- Text and speech processing like-Voice assistants – Alexa, Siri, etc.
- Text classification like Grammarly, Microsoft Word, and Google Docs

- Chatbot and Question Answering like:- website bots
- Language Translation like:- Google Translate
- Text summarization

1-6 Phases of Natural Language Processing



2- Introduction to NLTK: Tokenization, Stemming, Lemmatization, POS Tagging

The Natural Language Toolkit (NLTK) is a comprehensive Python library for diverse NLP tasks. It covers text pre-processing, text vectorization, and more. Here, we'll delve into NLTK basics, exploring Tokenization, Stemming, Lemmatization, and POS Tagging in this article.

- **Installation:**

```
pip install nltk
```

2-1 Tokenization

Tokenization refers to breaking down the text into smaller units. It entails splitting paragraphs into sentences and sentences into words. It is one of the initial steps of any NLP pipeline

Work Tokenization

It involves breaking down the text into words.

```
 "I study Machine Learning on GeeksforGeeks." will be word-
tokenized as
  ['I', 'study', 'Machine', 'Learning', 'on', 'GeeksforGeeks',
'.'].
```

2-2 Stemming and Lemmatization

Stemming is the process of reducing words to their root or base form, stripping suffixes or prefixes to obtain the word's core meaning. It simplifies words to their common base, facilitating analysis in natural language processing.

Lemmatization involves grouping together the inflected forms of the same word. This way, we can reach out to the base form of any word which will be meaningful in nature. The base from here is called the Lemma.

- **Lemmatizers are slower and computationally more expensive than stemmers.**

2-3 Part of Speech Tagging

Part of Speech (POS) tagging refers to assigning each word of a sentence to its part of speech. It is significant as it helps give a better syntactic overview of a sentence.

3- Word Embeddings in NLP

Word embeddings are numerical representations of words or documents in a lower-dimensional space. They facilitate capturing semantic relationships between words by placing similar words closer together in the embedding space

3-1 Goal of Word Embeddings

- To reduce dimensionality
- To use a word to predict the words around it
- Interword semantics must be captured

3-2 Implementations of Word Embeddings:

Word embeddings are methods for extracting text features to represent words while preserving both syntactic and semantic information. Unlike traditional approaches like Bag-of-Words, CountVectorizer, and TFIDF that rely on word counts and do not retain syntactic or semantic meanings, word embeddings provide a compact representation of words in a reduced-dimensional space. This enables more efficient model training and maintains semantic relationships between words.

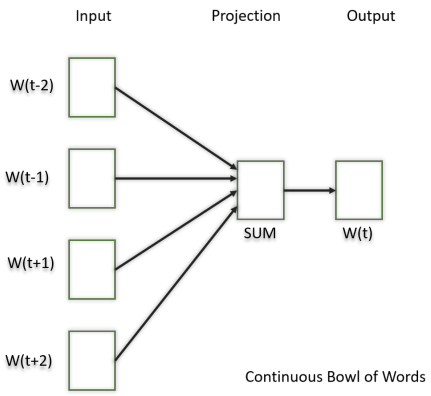
3-3 Word2Vec:

In Word2Vec every word is assigned a vector. We start with either a random vector or one-hot vector.

- **One-Hot vector:** A representation where only one bit in a vector is 1.If there are 500 words in the corpus then the vector length will be 500

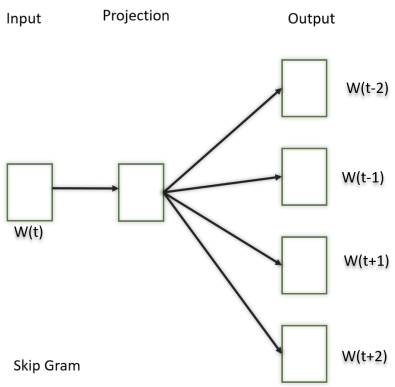
Continuous Bag of Words(CBOW)

In this model what we do is we try to fit the neighboring words in the window to the central word.



Skip Gram

we try to make the central word closer to the neighboring words. It is the complete opposite of the CBOW model. It is shown that this method produces more meaningful embeddings.



3-4 GloVe:

This method creates word embeddings by analyzing a text corpus, generating a co-occurrence matrix that assigns values based on word proximity within the text.

People generally use pre-trained models for word embeddings. Few of them are:

- SpaCy
- fastText
- Flair etc.

3-5 Benefits of using Word Embeddings:

- It is much faster to train than hand build models like WordNet(which uses *graph embeddings*)
- Almost all modern NLP applications start with an embedding layer
- It Stores an approximation of meaning

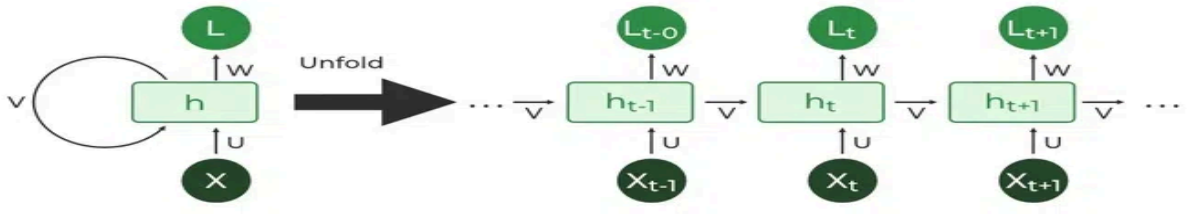
3-6 Drawbacks of Word Embeddings:

- It can be memory intensive
- It is corpus dependent. Any underlying bias will have an effect on your model
- It cannot distinguish between homophones. Eg: brake/break, cell/sell, weather/whether etc.

Introduction to Recurrent Neural Network

1- Definition

A Recurrent Neural Network (RNN) is a type of neural network that uses sequential information by feeding the output of the previous step as input to the current step. It maintains a hidden state or memory, enabling it to remember past inputs and reducing parameter complexity by sharing parameters across inputs and hidden layers.



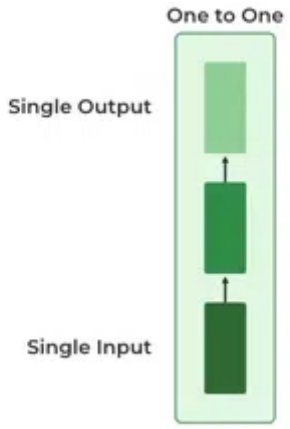
2- Types Of RNN

There are four types of RNNs based on the number of inputs and outputs in the network.

1. One to One
2. One to Many
3. Many to One
4. Many to Many

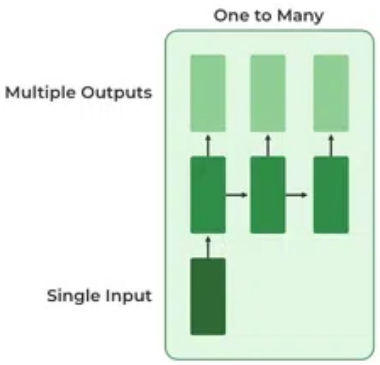
❖ One to One

In this Neural network, there is only one input and one output.



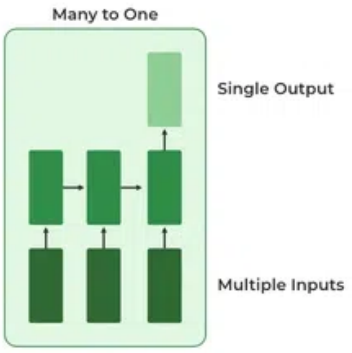
❖ One To Many

In this type of RNN, there is one input and many outputs associated with it.



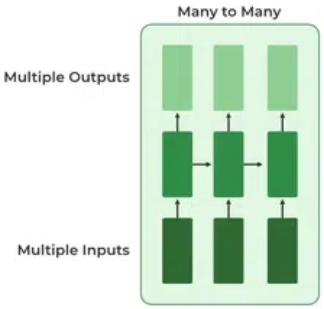
❖ Many to One

In this type of network, Many inputs are fed to the network at several states of the network generating only one output.



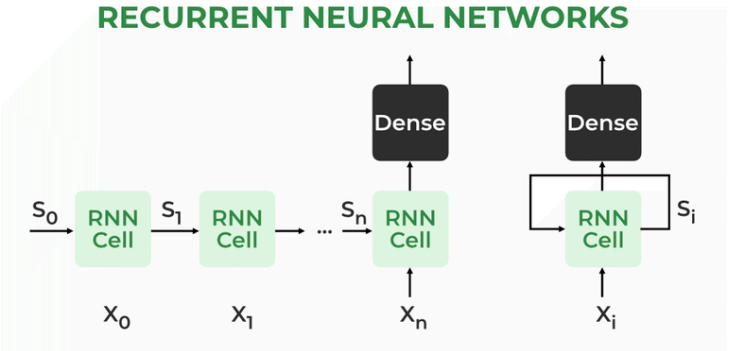
❖ Many to Many

In this type of neural network, there are multiple inputs and multiple outputs corresponding to a problem.



3- Recurrent Neural Network Architecture

In RNNs, the architecture resembles that of other deep neural networks in terms of input and output structure. The distinction lies in how information moves from input to output. Unlike typical deep neural networks where different weight matrices exist for each Dense network, in RNNs, the weight remains consistent across the network.



4- How does RNN work?

The Recurrent Neural Network consists of multiple fixed activation function units, one for each time step. Each unit has an internal state which is called the hidden state of the unit. This hidden state signifies the past knowledge that the network currently holds at a given time step

The formula for calculating the current state:

$$h_t = f(h_{t-1}, x_t)$$

where,

- h_t -> current state
- h_{t-1} -> previous state
- x_t -> input state

Formula for applying Activation function(tanh)

$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

where,

- w_{hh} -> weight at recurrent neuron
- w_{xh} -> weight at input neuron

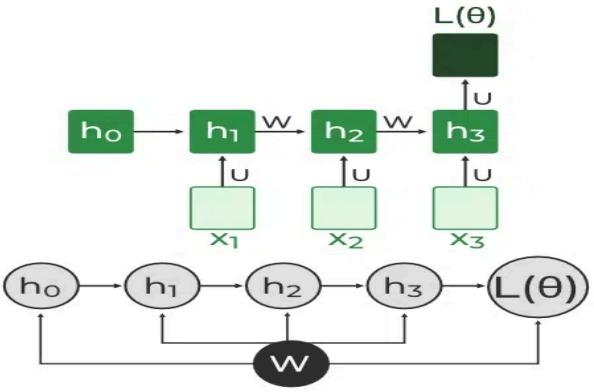
The formula for calculating output:

$$y_t = W_{hy}h_t$$

- Y_t -> output
- W_{hy} -> weight at output layer

4-1 Backpropagation Through Time (BPTT)

In RNN the neural network is in an ordered fashion and since in the ordered network each variable is computed one at a time in a specified order like first h1 then h2 then h3 so on. Hence we will apply backpropagation throughout all these hidden time states sequentially.



5- Issues of Standard RNNs

In RNNs, the vanishing gradient problem causes diminishing gradients during training, making it challenging. Conversely, the exploding gradient issue arises when gradients grow exponentially, leading to excessively large updates in the network's weights.

6- Advantages and Disadvantages of Recurrent Neural Network

- **Advantages**
  1. An RNN remembers each and every piece of information through time. It is useful in time series prediction only because of the feature to remember previous inputs as well. This is called Long Short Term Memory.
  2. Recurrent neural networks are even used with convolutional layers to extend the effective pixel neighborhood.
- **Disadvantages**
  1. Gradient vanishing and exploding problems.
  2. Training an RNN is a very difficult task.
  3. It cannot process very long sequences if using tanh or relu as an activation function.

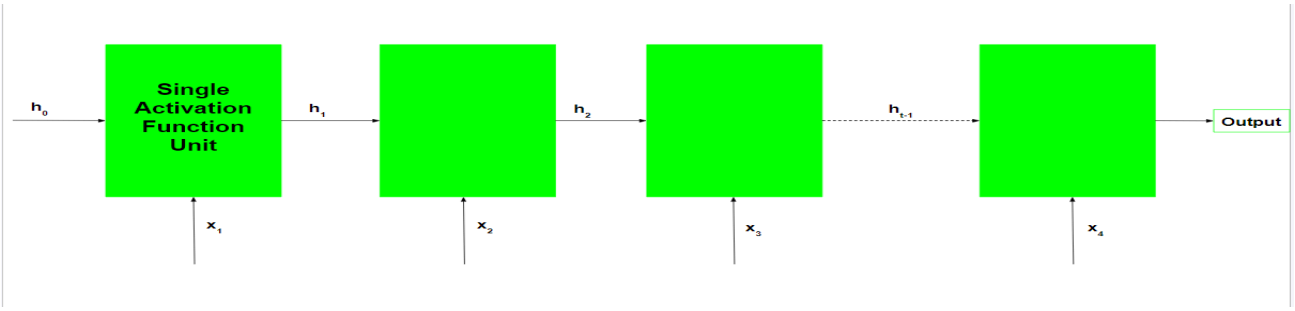
7- Applications of Recurrent Neural Network

- Language Modelling and Generating Text
- Speech Recognition
- Machine Translation

8- Difference between RNN and Simple Neural Network

Recurrent Neural Network	Deep Neural Network
Weights are same across all the layers number of a Recurrent Neural Network	Weights are different for each layer of the network
Recurrent Neural Networks are used when the data is sequential and the number of inputs is not predefined.	A Simple Deep Neural network does not have any special method for sequential data also here the the number of inputs is fixed
The Numbers of parameter in the RNN are higher than in simple DNN	The Numbers of Parameter are lower than RNN
Exploding and vanishing gradients is the the major drawback of RNN	These problems also occur in DNN but these are not the major problem with DNN

The basic work-flow of a Recurrent Neural Network is as follows



Note that h0 is the initial hidden state of the network. Typically, it is a vector of zeros, but it can have other values also .

- **Working of each Recurrent Unit:**

Receive the previous hidden state vector and the current input vector.

- Treat each element in these vectors as a separate orthogonal dimension.
- Multiply each element in the hidden state vector by the hidden state weights.
- Similarly, multiply each element in the current input vector by the current input weights.
- Generate the weighted hidden state vector and the weighted current input vector.
- Add these two parameterized vectors together.
- Apply the hyperbolic tangent (tanh) function element-wise to the result.
- Output the new hidden state vector, representing updated network memory in the recurrent unit.

During the training of the recurrent network, the network also generates an output at each time step. This output is used to train the network using gradient descent.

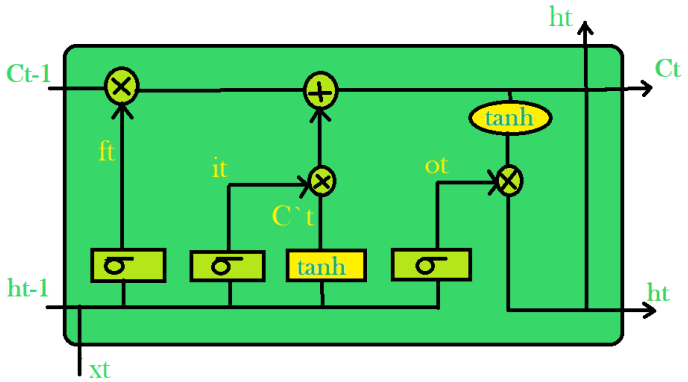
Short term Memory

Researchers and neurologists widely agree on two key mechanisms affecting brain functions, altering neuron operations through synaptic changes. Short-term memory (STM), or working memory, allows temporary retention and manipulation of around 7 ± 2 items for 20-30 seconds, involving various sensory data. Investigating STM involves understanding how neural information is temporarily stored in sequences, managing uncertain storage needs, and transferring data to long-term memory (LTM).

In this article, we will be discussing LSTM which is a popular use case of STM. We will also discuss the different variants in LSTM.

1- LSTM

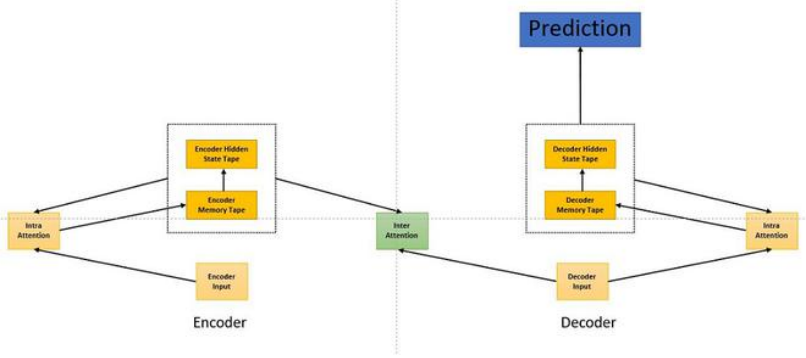
LSTM was proposed by Sepp Hochreiter and Jurgen Schmidhuber in the 1990s. The main idea behind this is to correct the failures of RNN in certain situations. It fails to store information for a longer period of time. At times, a reference to certain information stored quite a long time ago is required to predict the current output. But RNNs are absolutely incapable of handling such "long-term dependencies". Below is the architecture of LSTM:



The LSTM cell receives three inputs: the previous cell state Ct-1, the hidden state ht-1, and the input state xt. It utilizes these inputs in three stages. First, it uses a sigmoid layer with xt and ht-1 to generate the forget gate ft. Next, it concatenates xt and ht-1, passing them separately through sigmoid and tanh layers. Their pointwise multiplication forms the input gate. Finally, the output gate employs the sigmoid of the input from the input gate, which is then pointwise multiplied with the tanh of the cell state

2- Shallow Attention Fusion & Deep Attention Fusion

Shallow attention fusion treats the LSTM-N (similar to LSTM except the memory cell is replaced with memory network) as a separate module that can be readily used in an encoder-decoder architecture. The decoder and encoder are both molded by an intra-attention module. Inter attention is triggered when the decoder reads a target token similar to it.



Deep Learning | Introduction to Long Short Term Memory

1- What is LSTM?

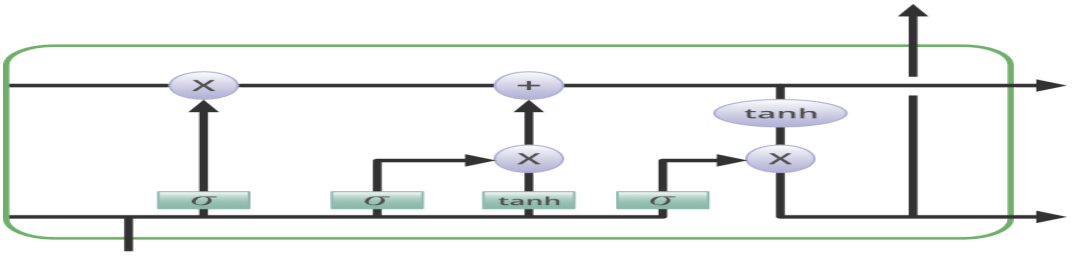
LSTM is an advanced type of recurrent neural network designed to overcome the challenge of learning long-term dependencies in sequential data. Unlike traditional RNNs, LSTMs introduce a memory cell that retains information over extended periods. They excel in tasks involving long-term dependencies, such as language translation, speech recognition, and time series forecasting. LSTMs feature three gates—the input, forget, and output gates—that control the flow of information within the memory cell.

1-1 Bidirectional LSTM

Bidirectional LSTM (Bi LSTM/ BLSTM) is a recurrent neural network (RNN) that is able to process sequential data in both forward and backward directions. This allows Bi LSTM to learn longer-range dependencies in sequential data than traditional LSTMs, which can only process sequential data in one direction.

1-2 Architecture and Working of LSTM

LSTM architecture has a chain structure that contains four neural networks and different memory blocks called cells.



Information is retained by the cells and the memory manipulations are done by the gates. There are three gates

- **Forget gate :**

The forget gate in LSTM discards outdated information from the cell state. It takes inputs xt (current input) and ht-1 (previous cell output), processes them through weights and a bias, and passes them through a sigmoid function. An output of 0 means forgetting the information, while an output of 1

retains it for future use. The equation is: $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$, where W_f is the weight matrix, $[h_{t-1}, x_t]$ is the concatenated input, b_f is the bias, and σ is the sigmoid function.

❖ Input gate

The input gate in LSTM controls the addition of relevant information to the cell state. It uses a sigmoid function to filter and retain information from inputs ht-1 and xt, akin to the forget gate. Then, it generates a vector using the tanh function, encompassing potential values from ht-1 and xt. This vector is multiplied with the regulated values to acquire valuable information. The equations are: $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$ and $\hat{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$. The resulting process combines the previously disregarded state ($f_t \odot C_{t-1}$) with the updated candidate values ($i_t \odot \hat{C}_t$) using element-wise multiplication.

❖ Output gate

The task of extracting useful information from the current cell state to be presented as output is done by the output gate. First, a vector is generated by applying tanh function on the cell. Then, the information is regulated using the sigmoid function and filter by the values to be remembered using inputs *ht-1* and *xt*. At last, the values of the vector and the regulated values are multiplied to be sent as an output and input to the next cell. The equation for the output gate is:

1-3 LTSM vs RNN

Feature	**LSTM (Long Short-term Memory)**	**RNN (Recurrent Neural Network)**

Memory	Has a special memory unit that allows it to learn long-term dependencies in sequential data	Does not have a memory unit
Directionality	Can be trained to process sequential data in both forward and backward directions	Can only be trained to process sequential data in one direction
Training	More difficult to train than RNN due to the complexity of the gates and memory unit	Easier to train than LSTM
Applications	Machine translation, speech recognition, text summarization, natural language processing, time series forecasting	Natural language processing, machine translation, speech recognition, image processing, video processing

2- LSTM – Derivation of Backpropagation through time

As the name suggests, backpropagation through time is similar to backpropagation in DNN(deep neural network) but due to the dependency of time in RNN and LSTM, we will have to apply the chain rule with time dependency.

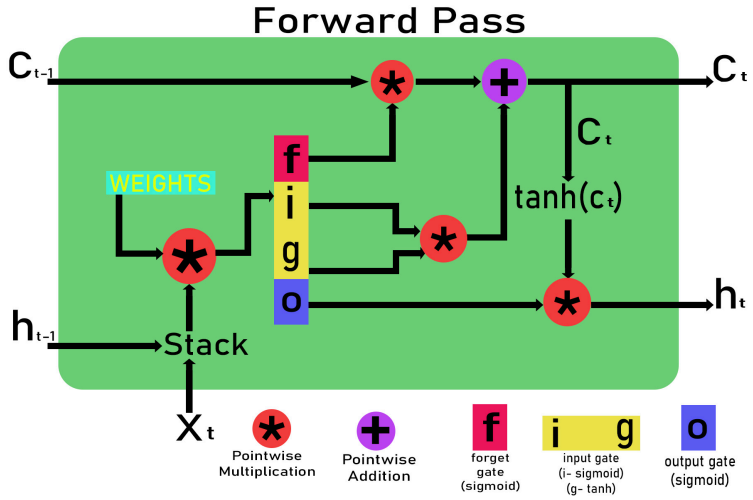
Let the input at time t in the LSTM cell be xt, the cell state from time t-1 and t be ct-1 and ct and the output for time t-1 and t be ht-1 and ht . The initial value of ct and ht at t = 0 will be zero.

Step 1 : Initialization of the weights .

Step 2 : Passing through different gates

Step 3 : Calculating the output ht and current cell state ct.

Step 4 : Calculating the gradient through backpropagation through time at time stamp t using the chain rule.



Finally the gradients associated with the weights are,

- $dE/dw_{xo} = dE/d_o * (d_o/dw_{xo}) = E\_delta * tanh(c_t) * sigmoid(z_o) * (1-sigmoid(z_o)) * x_t$
- $dE/dw_{ho} = dE/do * (do/dw_{ho}) = E\_delta * tanh(c_t) * sigmoid(z_o) * (1-sigmoid(z_o)) * h_{t-1}$
- $dE/db_o = dE/do * (do/db_o) = E\_delta * tanh(c_t) * sigmoid(z_o) * (1-sigmoid(z_o)$
- $dE/dw_{xf} = dE/df * (df/dw_{xf}) = E\_delta * o * (1-tanh^2(c_t)) * c_{t-1} * sigmoid(z_f) * (1-sigmoid(z_f)) * x_t$
- $dE/dw_{hf} = dE/df * (df/dw_{hf}) = E\_delta * o * (1-tanh^2(c_t)) * c_{t-1} * sigmoid(z_f) * (1-sigmoid(z_f)) * h_{t-1}$
- $dE/db_o = dE/df * (df/db_o) = E\_delta * o * (1-tanh^2(c_t)) * c_{t-1} * sigmoid(z_f) * (1-sigmoid(z_f)$
- $dE/dw_{xi} = dE/di * (di/dw_{xi}) = E\_delta * o * (1-tanh^2(c_t)) * g * sigmoid(z_i) * (1-sigmoid(z_i)) * x_t$
- $dE/dw_{hi} = dE/di * (di/dw_{hi}) = E\_delta * o * (1-tanh^2(c_t)) * g * sigmoid(z_i) * (1-sigmoid(z_i)) * h_{t-1}$
- $dE/db_i = dE/di * (di/db_i) = E\_delta * o * (1-tanh^2(c_t)) * g * sigmoid(z_i) * (1-sigmoid(z_i)$
- $dE/dw_{xg} = dE/dg * (dg/dw_{xg}) = E\_delta * o * (1-tanh^2(c_t)) * i * (1-tanh^2(z_g)) * x_t$
- $dE/dw_{hg} = dE/dg * (dg/dw_{hg}) = E\_delta * o * (1-tanh^2(c_t)) * i * (1-tanh^2(z_g)) * h_{t-1}$
- $dE/db_g = dE/dg * (dg/db_g) = E\_delta * o * (1-tanh^2(c_t)) * i * (1-tanh^2(z_g))$

Using all gradients, we can easily update the weights associated with input gate, output gate, and forget gate .

Gated Recurrent Unit Networks

1- Principle of GRU

Gated Recurrent Unit (GRU) is a type of recurrent neural network (RNN) that was introduced by Cho et al. in 2014 as a simpler alternative to Long Short-Term Memory (LSTM) networks. Like LSTM, GRU can process sequential data such as text, speech, and time-series data.

GRU (Gated Recurrent Unit) uses gating mechanisms to control information flow in the network. It consists of two gates—the reset gate and the update gate. The reset gate manages the previous hidden state, while the update gate regulates the new input's impact on the hidden state, determining the GRU's output.

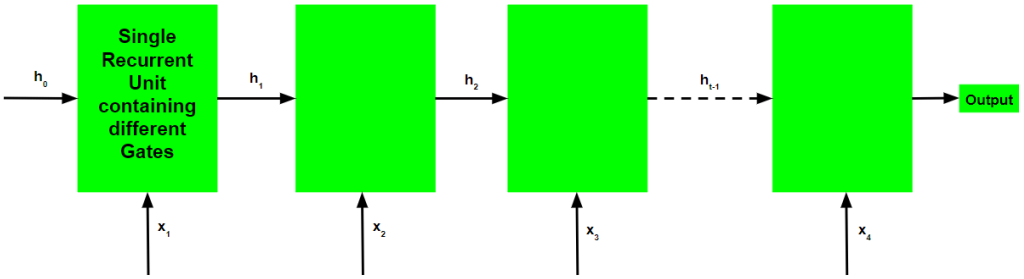
The equations used to calculate the reset gate, update gate, and hidden state of a GRU are as follows:

Reset gate: ***r_t = sigmoid(W_r * [h_{t-1}, x_t])***
Update gate: ***z_t = sigmoid(W_z * [h_{t-1}, x_t])***
Candidate hidden state: ***h_t' = tanh(W_h * [r_t * h_{t-1}, x_t])***
Hidden state: ***h_t = (1 − z_t) * h_{t-1} + z_t * h_t'***
where W_r, W_z, and W_h are learnable weight matrices, x_t is the input at time step t, h_{t-1} is the previous hidden state, and h_t is the current hidden state.

2- Outlines :

- To address the challenges of vanishing-exploding gradients in Recurrent Neural Networks (RNNs), variations like LSTM and GRU were developed.
- GRU, although less recognized compared to LSTM, is equally effective and consists of three gates without an Internal Cell State.
- Unlike LSTM, GRU integrates stored information from LSTM's Internal Cell State into its hidden state and passes it to the next GRU unit.
- The gates in GRU include the Update Gate (z) determining future influence, the Reset Gate (r) deciding forgotten information, and the Current Memory Gate (\overline{h}_{t}) introducing non-linearity and zero-mean to the input.
- GRU's workflow resembles that of a basic RNN but differs in its internal mechanism, utilizing gates to modulate the current input and previous hidden state within each recurrent unit.

The basic work-flow of a Gated Recurrent Unit Network is similar to that of a basic Recurrent Neural Network when illustrated, the main difference between the two is in the internal working within each recurrent unit as Gated Recurrent Unit networks consist of gates which modulate the current input and the previous hidden state.



The Back-Propagation Through Time Algorithm for a Gated Recurrent Unit Network is similar to that of a Long Short Term Memory Network and differs only in the differential chain formation.

3- How do Gated Recurrent Units solve the problem of vanishing gradients?

Thus the Back-Propagation Through Time algorithm adjusts the respective weights in such a manner that the value of the chain of derivatives is as close to 1 as possible.

Generative learning

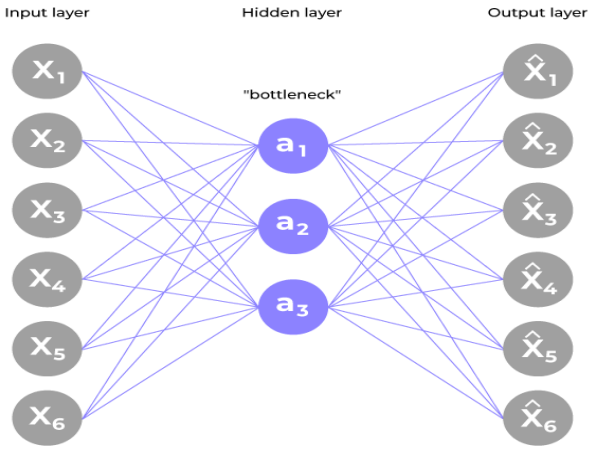
1- Autoencoders -Machine Learning

1-1 What are Autoencoders?

Autoencoders are neural network algorithms that learn efficient data representations without labels. They consist of an encoder and decoder: the encoder reduces input data into a compact representation, while the decoder reconstructs the original input. This unsupervised learning method helps define essential data features by encoding and decoding information to capture meaningful patterns.

1-2 Architecture of Autoencoder in Deep Learning

The general architecture of an autoencoder includes an encoder, decoder, and bottleneck layer.



Encoder:

- Input layer handles raw input data.
- Hidden layers progressively reduce input dimensionality, forming the encoder.
- Bottleneck layer significantly reduces dimensionality, representing compressed encoding.

Decoder:

- Bottleneck layer expands the encoded representation to original input dimensionality.
- Hidden layers gradually increase dimensionality to reconstruct input.
- Output layer generates reconstructed output resembling the input data.

Training:

- Loss function (e.g., MSE for continuous data) measures input-reconstructed output difference.
- Autoencoder minimizes reconstruction loss during training, capturing key input features.

Post-Training:

- Only the encoder segment remains for encoding similar data.

Network Constraints:

- Small Hidden Layers: Forces focus on representative data features.
- Regularization: Encourages diverse network training.
- Denoising: Trains network to eliminate noise from input data.
- Activation Function Tuning: Reduces hidden layer sizes by adjusting activation functions.

1-3 Types of Autoencoders

There are diverse types of autoencoders and analyze the advantages and disadvantages associated with different variation:

❖ Denoising Autoencoder

Denoising autoencoder works on a partially corrupted input and trains to recover the original undistorted image. As mentioned above, this method is an effective way to constrain the network from simply copying the input and thus learn the underlying structure and important features of the data.

❖ Sparse Autoencoder

This type of autoencoder typically contains more hidden units than the input but only a few are allowed to be active at once. This property is called the sparsity of the network. The sparsity of the network can be controlled by either manually zeroing the required hidden units, tuning the activation functions or by adding a loss term to the cost function.

❖ Variational Autoencoder

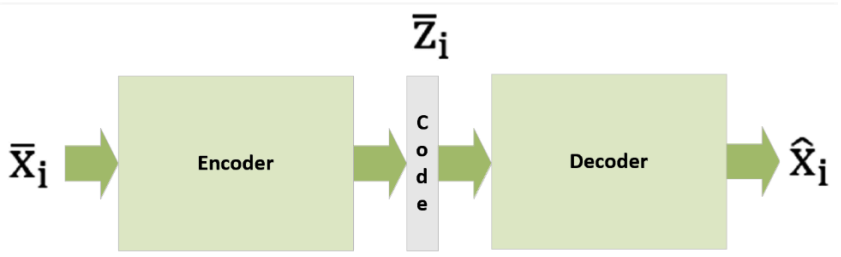
Variational autoencoder makes strong assumptions about the distribution of latent variables and uses the Stochastic Gradient Variational Bayes estimator in the training process.

❖ Convolutional Autoencoder

Convolutional autoencoders are a type of autoencoder that use convolutional neural networks (CNNs) as their building blocks. The encoder consists of multiple layers that take a image or a grid as input and pass it through different convolution layers thus forming a compressed representation of the input

1-4 How Autoencoders works ?

Here is a simple illustration of a generic autoencoder:



For a p-dimensional vector code, a parameterized function, e(•) is the definition of the encoder:

$$\bar{z}_i = e(\bar{x}_i, \bar{\theta}_e) \;\; where \;\; \bar{x}_i \in \mathbb{R}^n \; and \; \bar{z}_i \in \mathbb{R}^p$$

In an analogous way, the decoder is another parameterized function, d(•):

$$\hat{x}_i = d(\bar{z}_i, \bar{\theta}_d) \;\; where \;\; \hat{x}_i \in \mathbb{R}^n \; and \; \bar{z}_i \in \mathbb{R}^p$$

Thus, when given an input sample, xi, a full autoencoder, an amalgamated function, will provide the best alternative as output:

$$\hat{x}_i = d(e(\bar{x}_i, \bar{\theta}_e), \bar{\theta}_d) = g(\bar{x}_i, \bar{\theta})$$

An autoencoder is trained using the back-propagation algorithm frequently based on the mean square error cost function, the reason being an autoencoder is generally applied through neural networks.

$$C(X, \hat{X}, \bar{\theta}) = \frac{1}{m} \sum_i (\bar{x}_i - g(\bar{x}_i, \bar{\theta}))$$

2- Variational AutoEncoders

autoencoders aim to compress and reconstruct data using an encoder-decoder structure. Variational Autoencoders (VAEs) stand out by introducing probabilistic encoding, expanding their applications. VAEs consist of an encoder for efficient data encoding and a decoder to regenerate dataset-like images. Reconstruction loss drives these architectures, updating the neural network through the loss function.

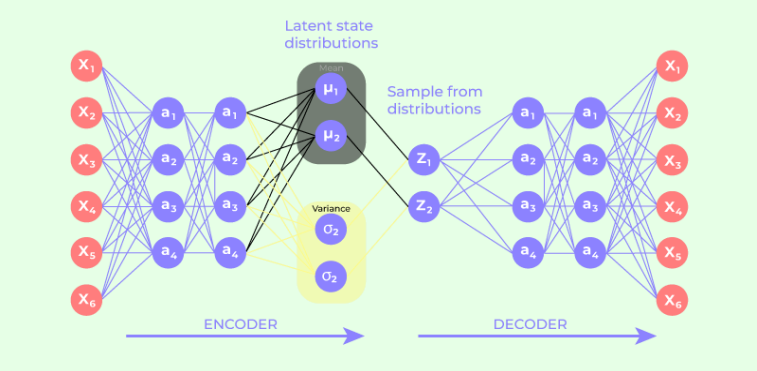
2-1 What is a Variational Autoencoder?

Variational Autoencoder (VAE), proposed in 2013 by Kingma and Welling, employs a probabilistic method in latent space representation. Unlike traditional autoencoders, VAEs output probability distributions for latent attributes. They find use in data compression and synthetic data creation by introducing statistical descriptions for dataset samples in latent space. Specifically, VAEs have encoders generating probability distributions at the bottleneck layer rather than single output values.

2-2 Architecture of Variational Autoencoder

- Variational Autoencoders (VAEs) use an encoder-decoder architecture, differentiating them from traditional autoencoders.

- The encoder transforms input data into a probabilistic distribution within the latent space, offering multiple representations.
- Decoder reconstructs data from sampled points in this distribution during training.
- Model adjustments aim to minimize reconstruction loss and regularize the latent space to a specified distribution, balancing accuracy and generalization.
- This iterative process yields a meaningful latent space representation encapsulating data features.
- The optimized latent code allows for precise reconstruction and novel sample generation due to the probabilistic nature of the latent space.



3- Contractive Autoencoder (CAE)

Contractive Autoencoder was proposed by the researchers at the University of Toronto in 2011 in the paper Contractive auto-encoders: Explicit invariance during feature extraction. The idea behind that is to make the autoencoders robust of small changes in the training dataset.

To deal with the above challenge that is posed in basic autoencoders, the authors proposed to add another penalty term to the loss function of autoencoders. We will discuss this loss function in detail.

3-1 The Loss function:

Contractive autoencoder adds an extra term in the loss function of autoencoder, it is given as:

$$\|J_h(X)\|_F^2 = \sum_{ij} \left(\frac{\partial h_j(X)}{\partial X_i} \right)^2$$

i.e the above penalty term is the Frobenius Norm of the encoder, the frobenius norm is just a generalization of Euclidean norm.

3-2 Relationship with Sparse Autoencoder

Sparse autoencoders aim for most representation components to be close to 0, achieved when lying in the left-saturated sigmoid region. This yields small values and derivatives, creating a highly contractive mapping, although this isn't the main objective of sparse autoencoders.

3-3 Relationship with Denoising Autoencoder

Denoising autoencoders (DAEs) emphasize robust reconstruction, indirectly enhancing representation robustness. In contrast, Contractive Autoencoders (CAEs) prioritize the robustness of representation by focusing on the first derivative of the Jacobian matrix.

Generative Adversarial Networks

1. Basics of Generative Adversarial Networks (GANs)

Introduction to GANs

Generative Adversarial Networks (GANs) are a powerful approach in generative modeling using deep learning methods, such as Convolutional Neural Networks (CNNs). They are used in unsupervised learning to discover and learn patterns in input data, enabling the generation of new examples that resemble the original dataset.

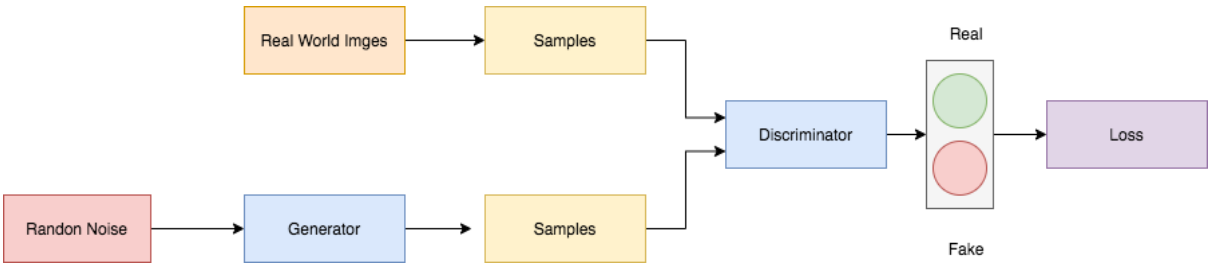
Components of GANs

- Generator: This component is trained to generate new data. For instance, in computer vision, it creates new images from existing real-world images.
- Discriminator: This component compares the generated images to real-world examples and classifies them as real or fake.

Working of GANs (Example: Generating Dog Images)

Training the Discriminator: The process starts by feeding the discriminator with both randomly generated images (from the generator) and real dog images. The discriminator then assesses these images and assigns probabilities indicating their authenticity.

Training the Generator: Based on the feedback from the discriminator, the generator adjusts its process to create more realistic images. This is achieved through backpropagation, where the loss from the discriminator is used to update the generator's weights.



Iterative Learning Process

- ❖ The generator and discriminator are trained in an iterative process.
- ❖ As training progresses, the generator becomes more adept at creating images that closely resemble real-world images.

Applications of GANs

Image Generation: Creating new images that are visually similar to a given dataset.

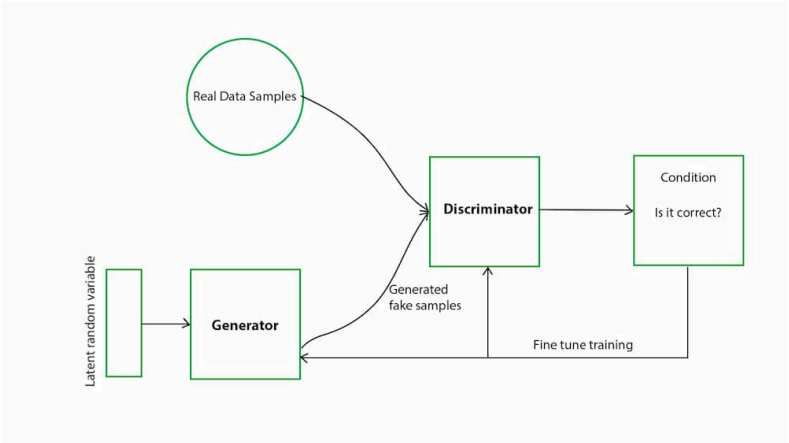
- Super Resolution: Enhancing the resolution of images.
- Image Modification: Altering images for various effects, such as changing the style.
- Photo-Realistic Images: Generating images that are indistinguishable from real photographies.
- Face Aging: Modifying facial images to predict future aging effects.

Understanding GANs in Practice

- GANs represent a complex yet fascinating area of deep learning.
- Their unique adversarial training process sets them apart from other neural network models.
- The potential applications of GANs are vast and continually expanding.

Generative Adversarial Network (GAN)

Generative Adversarial Networks (GANs) represent an advanced approach in deep learning, especially in the field of generative modeling. They are often built upon complex architectures like convolutional neural networks. The primary goal of GANs is to autonomously identify patterns in input data to generate new examples closely resembling the original dataset.



Key Components of GANs

- Generator: This component of a GAN generates novel data instances. It takes random noise as input and transforms it into data samples, like images or text, resembling those in the training set.
- Discriminator: The discriminator's role is to differentiate between actual data and data produced by the generator. It acts as a binary classifier, assigning probabilities to its inputs regarding their authenticity.

$$J_D = -\frac{1}{m}\Sigma_{i=1}^{m} log\ D(x_i) - \frac{1}{m}\Sigma_{i=1}^{m} log(1 - D(G(z_i))$$

Working of GANs

GANs operate through adversarial training, involving both the generator and discriminator in a competitive game. The generator aims to produce data that the discriminator cannot easily differentiate from real data. Through this process, the generator improves its ability to create realistic samples.

$$min_G\ max_D(G, D) = [\mathbb{E}_{xp_{data}}[log\ D(x)] + \mathbb{E}_{zp_z(z)}[log(1 - D(g(z)))]$$

Types of GAN Models

- Vanilla GAN: The most basic form, with both generator and discriminator as simple multi-layer perceptrons.
- Conditional GAN (CGAN): Introduces conditional parameters into the model, allowing control over the generated data.
- Deep Convolutional GAN (DCGAN): Uses convolutional networks instead of

multi-layer perceptrons, enhancing the quality and stability of the generated data.

- Laplacian Pyramid GAN (LAPGAN): Employs multiple levels of Generator and Discriminator networks at different levels of a Laplacian Pyramid, creating high-quality images.
- Super Resolution GAN (SRGAN): Focuses on generating higher-resolution images from low-resolution inputs.

Applications of GANs

GANs find extensive applications in various fields, including:

- Image Synthesis and Generation: Creating new, lifelike images that mimic the training data.
- Image-to-Image Translation: Converting images from one domain to another while retaining their core characteristics.
- Text-to-Image Synthesis: Producing images that correspond to text descriptions.
- Data Augmentation: Enhancing the robustness of machine learning models by generating synthetic data samples.
- Data Generation for Training: Improving the resolution and quality of low-resolution images.

Advantages and Disadvantages of GANs

Advantages:

- ❖ Synthetic data generation.
- ❖ High-quality, photorealistic results.
- ❖ Applicability in unsupervised learning tasks.
- ❖ Versatility in a wide range of applications.

Disadvantages:

- ❖ Training instability and complexity.
- ❖ High computational cost and potential for slow training.
- ❖ Risk of overfitting and bias.
- ❖ Challenges in ensuring interpretability and accountability.

Use Cases of Generative Adversarial Networks

Generative Adversarial Networks (**GANs**) have emerged as a versatile and powerful class of neural networks, finding applications in a wide array of fields. Here's an overview of their diverse use cases:

Image Synthesis: GANs are capable of generating new, realistic images from a given data distribution, such as creating images of faces, landscapes, or animals.
Text-to-Image Synthesis: This involves generating images from text descriptions. For example, GANs can create visual scenes, objects, or attributes described in words.
Image-to-Image Translation: GANs can translate images from one domain to another. This includes tasks like turning grayscale images to color, altering seasons in a landscape, or converting sketches into photorealistic images.
Anomaly Detection: They are used to identify outliers or anomalies in data, such as detecting fraudulent activities in financial transactions, network intrusions, or identifying unusual patterns in medical imaging.
Data Augmentation: GANs can increase the size and diversity of datasets, aiding in training deep learning models in areas like computer vision, speech recognition, or natural language processing.
Video Synthesis: They are used in generating new, realistic video sequences from specific data distributions, which could include human actions, animal behaviors, or animated sequences.
Music Synthesis: GANs have the capability to generate new, original music based on certain data distributions, covering various musical genres, styles, or instrumentations.
3D Model Synthesis: They can also be employed in generating new, realistic 3D models from a specified data distribution, applicable in objects, scenes, or shapes creation.

Newly Discovered Use Cases of GANs

Security: GANs play a significant role in cybersecurity by helping to counter adversarial attacks that aim to fool deep learning architectures.
Data Generation: In scenarios where data is scarce, like health diagnostics, GANs are used to generate high-quality data.
Privacy-Preserving: GANs are instrumental in data encryption, particularly in defense and military applications, ensuring data confidentiality.
Data Manipulation: GANs enable pseudo-style transfer in images or modifying specific parts of a text without altering the entire content.**Advantages and Disadvantages of GAN Use Cases**

Advantages:

- High-quality, photorealistic image generation.

- Ability to generate images from text descriptions, enhancing creative applications.
- Image-to-image translation for various practical uses.
- Effective anomaly detection in various domains.
- Enhancing deep learning model performance through data augmentation.
- Generating realistic video sequences and music.

Disadvantages:

- Difficulty in training and high computational resources requirement.
- Potential for overfitting and lack of diversity in the generated data.
- Reflecting biases present in training data, leading to unfair outcomes.
- Challenges in interpretability and accountability.
- Quality control issues in the generated synthetic data.

Generative Adversarial Networks have revolutionized the field of artificial intelligence, demonstrating remarkable versatility across a spectrum of applications. From enhancing creativity in art and design to contributing significantly to cybersecurity and medical diagnostics, GANs have opened new horizons in technology and innovation.

Cycle Generative Adversarial Network (CycleGAN)

Cycle Generative Adversarial Network (CycleGAN) is a type of GAN that has been specifically designed for image translation tasks. It addresses the challenge of image-to-image translation in the absence of paired examples, making it possible to translate an image from one domain to another without the need for a corresponding image in the target domain.

Key Features of CycleGAN

- **Image Reconstruction:** CycleGAN approaches image translation as an image reconstruction problem, using generators to convert an input image to a reconstructed image and then reverse the process.
- **Unpaired Image Translation:** Unlike other GANs, CycleGAN can perform image translation on unpaired images, where there is no direct correspondence between input and output images.
- **Architecture**: CycleGAN consists of two parts: the generator and the discriminator. The generator produces samples from a desired distribution, and the discriminator determines whether a sample is real or generated by the generator.
- **Two Mapping Functions:** It contains two mapping functions (G and F) that act as

generators, along with their corresponding discriminators (Dx and Dy). These mappings allow for the translation of images from one domain to another and vice versa.

- **Loss Functions:** In addition to adversarial loss, CycleGAN uses forward and backward cycle consistency loss to ensure that the translated images can be reliably converted back to their original form.

Generator and Discriminator Architecture

- **Generator:** Consists of three sections - encoder, transformer, and decoder. The encoder compresses the image, the transformer modifies it, and the decoder reconstructs the image back to its original size.
- **Discriminator:** Uses a PatchGAN discriminator, which classifies whether different parts (or patches) of the image are real or fake.

Applications of CycleGAN

- Collection Style Transfer: Training the model on landscape photographs to mimic the style of various artists.
- Object Transformation: Transforming objects from one class to another (e.g., zebras to horses).
- Season Transfer: Converting images from one season to another (e.g., winter to summer).
- Photo Generation from Paintings: Transforming photos from paintings and vice versa.
- Photo Enhancement: Improving image quality from smartphone cameras to DSLR-like quality.

Evaluation Metrics

- AMT Perceptual Studies: Assessing the realism of generated images.
- FCN Scores: Using semantic segmentation metrics to evaluate the quality of generated images.

Drawbacks and Limitations

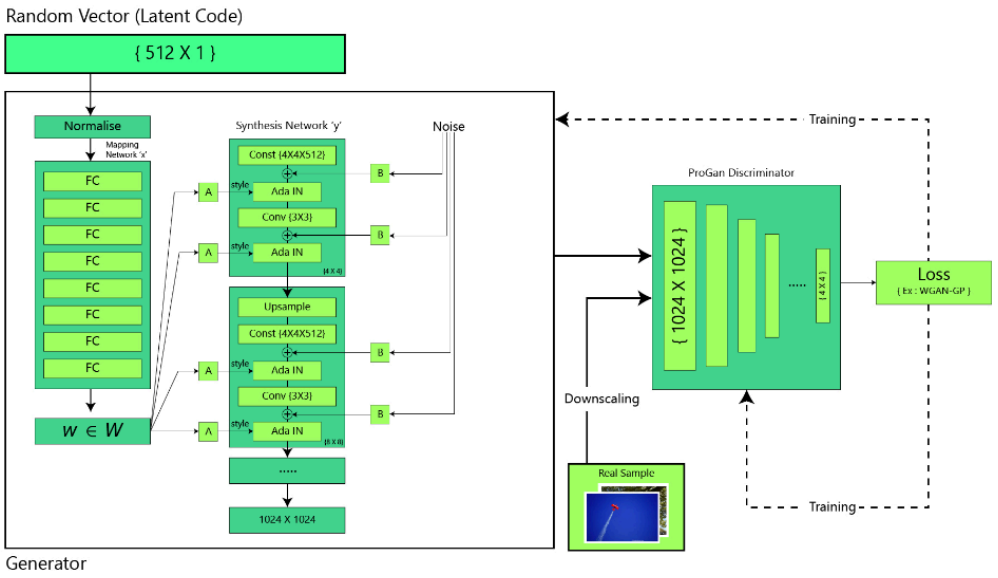
While CycleGAN excels at color and texture transformations, it struggles with geometric transformations due to limitations in the generator architecture. This can lead to failure cases where CycleGAN cannot adequately perform the desired image transformations.CycleGAN represents a significant advancement in GAN technology, particularly in its ability to perform complex image translation tasks without the need for paired training data. This opens up numerous possibilities for creative and practical applications across various domains.

StyleGAN – Style Generative Adversarial Networks

Style GAN proposes a lot of changes in the generator part which allows it to generate photo-realistic high-quality images as well as modify some parts of the generator part.

Architecture:

Style GAN uses the baseline progressive GAN architecture and proposed some changes in the generator part of it. However, the discriminator architecture is quite similar to baseline progressive GAN. Let's look at these architectural changes one by one.



Style GAN architecture

- **Baseline Progressive Growing GANs:** Style GAN uses baseline progressive GAN architecture which means the size of generated image increases gradually from a very low resolution *(4×4)* to high resolution *(1024 x1024)*. This is done by adding a new block to both the models to support the larger resolution after fitting the model on smaller resolution to make it more stable..
- **Bi-linear Sampling:** The authors of the paper uses bi-linear sampling instead of nearest neighbor up/down sampling (which was used in previous Baseline Progressive GAN architectures) in both generator and discriminator. They implement this bi-linear sampling by low pass filtering the activation with a separable 2nd order binomial filter after each of the upsampling layer and before each of the downsampling layer.
- **Mapping Network and Style Network**: The goal of the mapping network is to generate the input latent vector into the intermediate vector whose different elements control different visual features. Instead of directly providing a latent vector to the input layer the mapping is used. In this paper, the latent vector *(z)* of size 512 is mapped to another vector of 512 *(w)*. The mapping function is implemented using 8-layer MLP (8- fully

connected layers). The output of mapping network (w) then passed through a learned affine transformation (A) before passing into the synthesis network which AdaIN module. This model converts the encoded mapping into the generated image.

Reinforcement Learning

Introduction :

Reinforcement learning (RL) is becoming increasingly popular in AI, even though supervised learning (SL) is still the most common approach. RL has recently had some impressive successes, such as mastering old Atari games, winning at Go, and even beating professional players at 1v1 Dota.

SL is used more often because it's more familiar and relies on labeled data. With SL, you provide the model with input and the desired output, and it learns to map the two together. However, this means that the model can never be better than the data it's trained on.

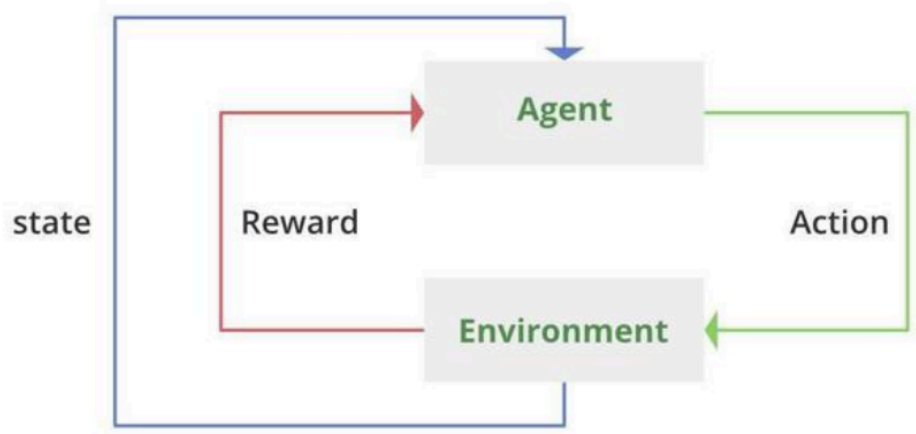
RL, on the other hand, learns through trial and error in dynamic environments. This means that it can potentially surpass human performance. However, RL can be more difficult to implement and requires more computational resources.

Overall, RL is a promising approach for complex AI tasks, but SL is still the most widely used method.

Main points in Reinforcement learning

- Input: The input should be an initial state from which the model will start
- Output: There are many possible outputs as there are a variety of solutions to a particular problem
- Training: The training is based upon the input, The model will return a state and the user will decide to reward or punish the model based on its output.
- The model keeps continues to learn.
- The best solution is decided based on the maximum reward.

Working on Reinforcement Learning:



Reinforcement learning (RL) stands as a potent method in artificial intelligence, enabling agents to grasp knowledge through experimentation. Unlike supervised learning, reliant on labeled data and precise directives, RL agents acquire insights by engaging with their environment, gaining rewards for accomplishing objectives and facing penalties for errors. This characteristic renders RL particularly effective for challenges that challenge conventional methods, like controlling robots or mastering complex games.

The Basic Setup:

Imagine an agent playing a simple game like Pong. At each timestep, the agent receives an image of the game screen (the input frame) and must choose an action (e.g., move the paddle up or down). The agent's goal is to score points and avoid losing.

Policy Gradients: Learning from Rewards

An often-used technique to train an RL agent involves employing policy gradients. This method commences with a basic neural network, which takes in the input frame and forecasts an action. The agent executes this action and notes the subsequent reward. This data informs adjustments to the neural network's weights, enhancing its tendency to select actions that yield favorable rewards in forthcoming scenarios.

The Credit Assignment Dilemma:

However, RL faces a unique challenge called the credit assignment dilemma. When the agent receives a reward (or penalty) at the end of an episode, it can be difficult to determine which

specific actions led to that outcome. This is especially true for long and complex tasks where many decisions contribute to the final result.

Sparse Rewards: A Bottleneck for Learning

Another challenge is sparse rewards. In many real-world tasks, rewards are infrequent and only occur when the agent achieves a specific goal. This can make it difficult for the agent to learn effectively, as it may take a long time to receive enough feedback to improve its behavior.

Montezuma's Revenge: A Test of Complexity

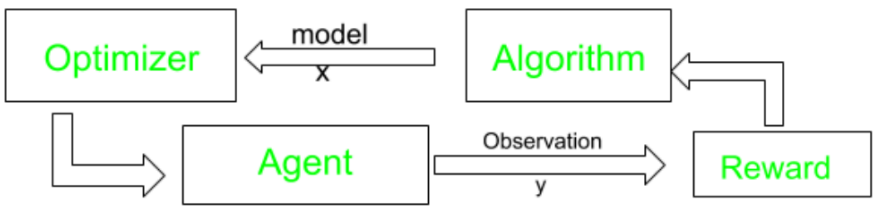
Consider the classic video game Montezuma's Revenge. In this game, the agent must navigate a series of ladders, jump over obstacles, collect a key, and reach a door to progress to the next level. Through random exploration alone, the agent is unlikely to ever receive a reward, as the sequence of actions required is too long and intricate. This highlights the limitations of RL in complex tasks with sparse rewards.

Reward Shaping: A Helping Hand

One way to overcome the challenge of sparse rewards is through reward shaping. This involves providing the agent with additional rewards along the way, even if it hasn't reached the final goal yet. These intermediate rewards can help guide the agent towards the desired behavior and make learning more efficient.

Thompson Sampling:

Imagine a robot tasked with learning how to pick up and deposit cans into a container. Each attempt provides valuable feedback, either a rewarding "success" or a penalty of "failure."



This trial-and-error approach lies at the heart of Thompson Sampling, an algorithm designed to tackle the exploration-exploitation dilemma in situations with uncertain outcomes.

The dilemma boils down to this: should we exploit what we already know works well, or explore new possibilities to potentially discover even better options? Thompson Sampling strikes a balance, initially focusing on exploration – trying different actions multiple times to gather information. As it learns from successes and failures, it gradually shifts towards exploitation, favoring actions that have consistently yielded positive results. This adaptive approach mimics the ideal trade-off in real-world scenarios, maximizing learning while minimizing unnecessary exploration.

Multi-Armed Bandit

Think of Thompson Sampling as a master strategist at a multi-armed bandit game, a slot machine with many levers (arms) offering unknown rewards. Each lever pull represents an action, and the jackpot signifies the desired reward. Without knowing the individual payoffs, the strategist must maximize their winnings by choosing the most lucrative levers. This analogy perfectly fits real-world situations like online advertising, where Thompson Sampling can help platforms select ads that are most likely to be clicked by users. Similarly, doctors can leverage this algorithm to personalize treatment plans for patients based on their individual responses to different therapies.

Algorithm:

- At each time step t= 1, … , T, pull one arm out of N arms.
- For each arm i, reward is generated from a fixed but unknown distribution support [0, 1], mean μ_i
- Our goal is to maximize the reward.
- Optimal arm: $i^* = \arg\max_i \mu_i$
- **Regret:** Loss suffered by not pulling the optimal arm.
- Optimal arm is the arm with expected reward: $\mu^* = \max_j \mu_j$
- Expected regret for pulling another arm i: $\Delta_i = \mu^* - \mu_i$
- Let arm I was pulled Ki(T) times in time T:
  Expected regret in any time t,

$$Regret(T) = \sum_{i \neq i^*} \Delta_i E[k_i(T)]$$

Examples of Real-World Applications:

- Personalizing Medical Treatments: Doctors can potentially use it to select the most effective experimental treatments for patients based on their individual characteristics and responses.
- Robot Learning: Robots can leverage Thompson Sampling to improve their performance in tasks like picking and placing objects, as described in the example provided.
- Optimizing Online Ads: Thompson Sampling can be used to determine which ads to display to web users to maximize clicks and conversions.

Unraveling Markov Decision Processes:

Imagine an agent trapped in a grid world, its goal to reach a shimmering diamond while steering clear of fiery danger. Each move brings not just a change in scenery, but also a reward or penalty shaping its journey. This world embodies the principles of a Markov Decision Process (MDP), a powerful framework for modeling decisions under uncertainty.

The Building Blocks of an MDP:

States: These represent the agent's possible locations, like the specific grid squares in our example. Think of them as snapshots of the world the agent perceives.

Actions: The levers at the agent's disposal, like moving up, down, left, or right. These are the tools it uses to navigate the world and shape its destiny.

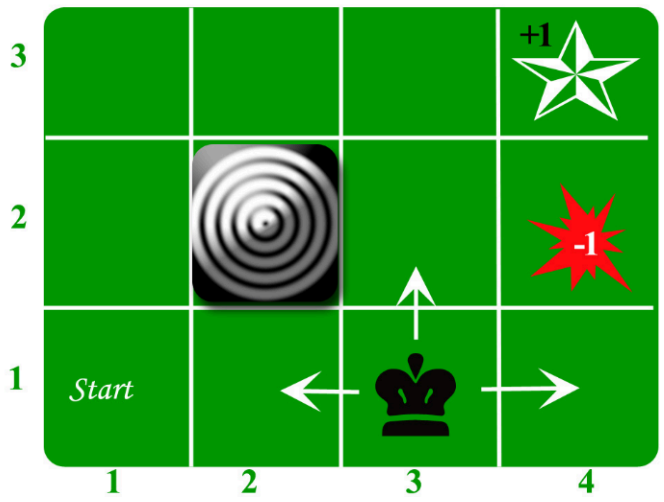
Rewards: The prizes and punishments peppering the journey. Stepping into the diamond nets a hefty reward, while venturing near the fire incurs a penalty. Rewards guide the agent towards its goal.

Model (Transition Function): This invisible puppeteer dictates how actions change the world. Taking a step "up" might not always land you where you expect, thanks to a bit of unpredictable noise. The model captures these probabilities, weaving a web of possibilities.

Policy: The agent's secret compass, mapping states to actions. It whispers which direction to take at each crossroads, aiming to maximize the total reward over the entire journey.

Delving into the Grid World:

In our 3x4 grid, the agent starts at square one, eager to reach the diamond nestled at the corner. Walls block some paths, while the fiery square holds a hefty negative reward. Each move carries a small reward, encouraging progress, while the final goal promises a grander bonus. The challenge lies in navigating the grid with its 80% success rate for intended actions and 20% chance of unexpected turns, all while maximizing the sum of rewards.



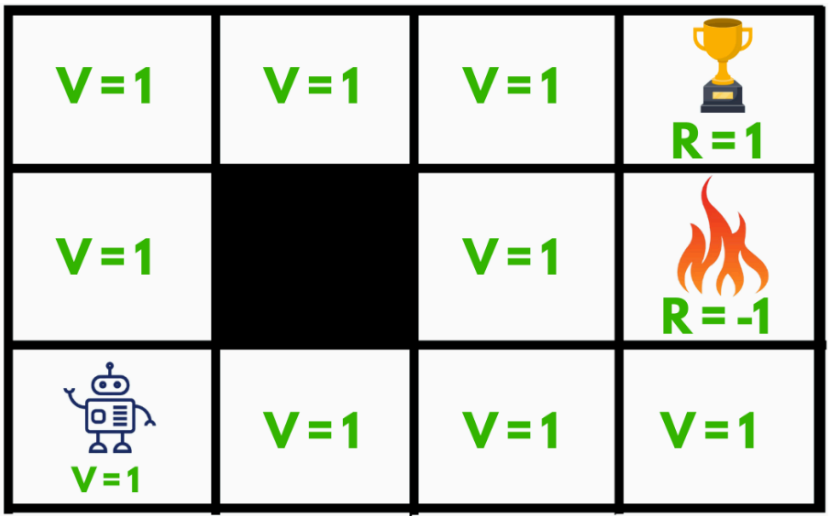
The Power of MDPs:

This grid world is just a glimpse into the vast potential of MDPs. They offer a structured language for describing decision-making problems under uncertainty, making them applicable to diverse fields like robot planning, resource allocation, and even medical diagnosis. By understanding the interplay between states, actions, rewards, and the ever-present model, we can design algorithms that help agents navigate complex worlds and make optimal choices, even when the future remains shrouded in mist.

The Bellman Equation:

the Bellman Equation guides agents towards optimal decisions by factoring in not just immediate rewards, but also the expected value of future rewards. In simpler terms, it tells the agent that the long-term payoff of an action is the sum of its immediate reward and the anticipated rewards from the subsequent actions it'll take.

Maze Escape:

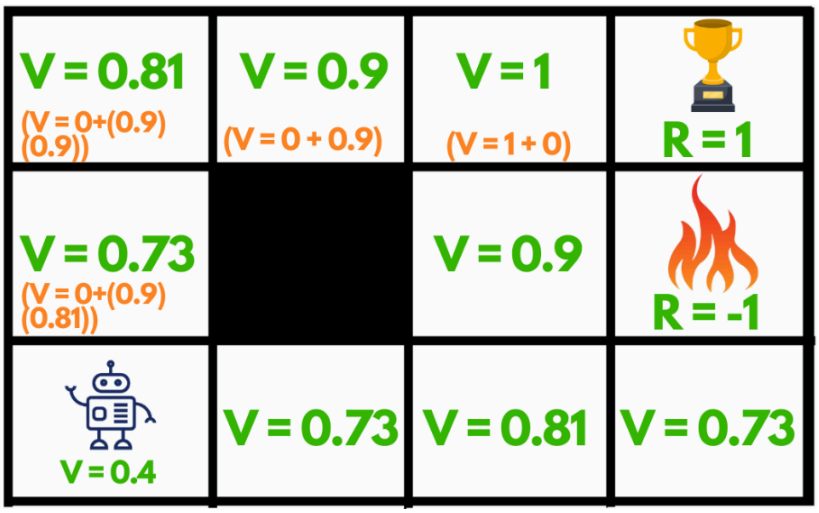


Consider our agent maneuvering through a perilous maze, aiming for the prized trophy state (reward +1) while steering clear of a fiery pitfall (reward -1). Without the Bellman Equation, the agent could chance upon the trophy, backtrack its movements, and assign an equal value (V = 1) to every state along the route. However, this approach falters when the starting position alters. The agent, lacking insight into the best path, becomes disoriented amidst a landscape of indistinguishable values.

Enter the Bellman Equation:

$V(s) = \max_a(R(s,a) + \gamma V(s'))$

This equation equips our agent with the power of forethought. It dissects each state (s), considering all possible actions (a) and their respective rewards (R(s,a)). It then takes into account the expected value ($\gamma V(s')$) of the next state (s') reached after taking action (a). The key term here is "max_a," which instructs the agent to choose the action that leads to the highest future reward, including immediate gains and anticipated benefits down the line.



Decoding the Variables:

V(s): Denotes the value of the current state (s).

R(s,a): Represents the reward received for taking action (a) in state (s).

γ: The discount factor (between 0 and 1) determines how much weight the agent gives to future rewards compared to immediate ones. A higher γ prioritizes long-term gains.

s': Represents the next state reached after taking action (a) in state (s).

Meta-Learning: Learning to Learn

Imagine an AI that can adapt to new challenges with barely any data, becoming smarter with each task it tackles. That's the promise of meta-learning, a revolutionary approach that teaches machines to "learn to learn."

Unlike traditional machine learning models trained on one specific task, meta-learning models are exposed to diverse tasks. This broadens their knowledge base, enabling them to quickly adjust to new situations with just a few examples. Think of it like a seasoned mentor guiding a novice, transferring crucial skills to tackle unforeseen problems.

This "learning to learn" process unfolds in two key phases:

- ➢ **Meta-training:** The mentor model, the "meta-learner," analyzes different tasks, extracting generalizable patterns and insights. These form the foundation for training a "base learner" model, capable of adapting to new tasks.
- ➢ **Meta-testing:** The base learner leverages the knowledge acquired during meta-training. With minimal data from a new task, it fine-tunes its parameters, learning to perform effectively in the new context.

This swift adaptability offers several advantages:

- Faster learning: New tasks are mastered with fewer steps and less data.
- Better generalization: Learned knowledge transfers well to new situations, even vastly different ones.
- Scalability: Automated model selection and fine-tuning pave the way for broader AI applications.
- Reduced data needs: Efficient learning even with limited data makes it ideal for resource-constrained settings.
- Improved performance: AI models become sharper and more accurate by adapting to diverse environments.

The applications of meta-learning are diverse and exciting:

- Few-shot learning: Quickly acquiring new skills with just a handful of examples, ideal for medical diagnosis or robotics.

  Reinforcement learning: Continuously adapting in dynamic environments, mastering new strategies on the fly.

- Natural language processing: Understanding complex language patterns and nuances across different contexts.
- Image classification: Recognizing objects under diverse visual conditions with ease.

Model selection and hyperparameter optimization: Choosing the best model and settings for specific tasks automatically.

Q-Learning in python

Introduction :

Q-Learning is a fundamental technique in Reinforcement Learning that uses Q-values to estimate the future reward of taking a specific action in a given state. These Q-values are constantly updated using the TD-Update rule, balancing immediate rewards with the potential future gain of different actions.

Episodes and Rewards:

An episode represents the agent's journey from a starting state to a terminal state. During this journey, the agent takes actions, transitions to new states, and receives rewards for its choices. These rewards act as feedback, guiding the agent towards actions that lead to desired outcomes.

TD-Update Rule:

This crucial formula updates Q-values based on several factors:

Formula:

$Q(S, A) \leftarrow Q(S, A) + \alpha * [R + \gamma * \max Q(S', A') - Q(S, A)]$

Explanation:

Q(S, A): The current Q-value for taking action A in state S.

α: The learning rate ($0 \leq \alpha \leq 1$), determining the step size for updates.

R: The immediate reward received after taking action A in state S.

γ: The discount factor ($0 \leq \gamma \leq 1$), valuing immediate rewards more than future ones.

max Q(S', A'): The highest Q-value for any action A' in the next state S'.

Components:

Current Reward (R): Influences the update directly, rewarding beneficial actions and discouraging detrimental ones.

Estimated Future Reward (max Q(S', A')): Incorporates anticipation of potential gains, guiding the agent towards long-term success.

Discount Factor (γ): Prioritizes immediate rewards over distant ones, shaping the agent's time preference.

Learning Rate (α): Controls the update magnitude, ensuring gradual learning and avoiding overcorrections.

Action Selection: Q-Learning often uses the epsilon-greedy policy to balance exploration and exploitation. With probability epsilon, the agent chooses a random action to explore the environment. With probability (1-epsilon), it selects the action with the highest Q-value, exploiting its current knowledge for optimal rewards.

Limitations and Alternatives: While powerful, Q-Learning can be slow and resource-intensive, especially in complex environments. Combining it with Deep Learning, known as Deep Q-Learning, tackles larger problems using neural networks to represent Q-values and extract valuable features from the environment.

Implementation

=> Look the notebook (**ML / RL Algorithm : Python Implementation using Q-learning**)

Deep Q-Learning

Introduction :

The quest for optimal behavior in complex environments drives many advancements in Artificial Intelligence. Q-Learning stands as a fundamental technique, building a matrix of "Q-values" that guide an agent towards maximizing rewards over time. This approach works well for small situations, but it quickly becomes impractical as the number of states and actions explode.

Enter Deep Q-Learning (DQN), a powerful upgrade that leverages the magic of deep neural networks. Instead of a rigid table, DQN uses a neural network to approximate the Q-values, allowing it to tackle vast and intricate environments. Think of it as a sophisticated map, constantly learning and adapting to guide the agent through the maze of possibilities.

Here's how DQN works in a nutshell:

Key Equation: The target Q-value calculation remains essential:

target = $R(s,a,s') + \gamma * \max_{a'} Q(s', a')$

Components:

- **target:**The value the agent aims to achieve for the Q-value of taking action in state s.
- **R(s,a,s'):** The immediate reward received for taking action in state s and transitioning to state s'.

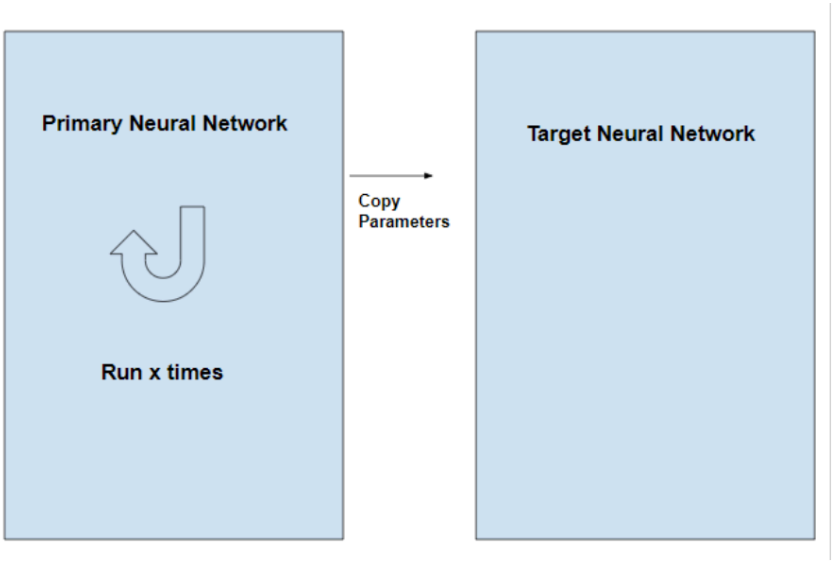
- **γ:** The discount factor ($0 \leq \gamma \leq 1$), prioritizing immediate rewards over future ones. Higher γ values place more importance on long-term rewards.
- **max_a' Q(s', a'):** The maximum Q-value for any action a' in the next state s'. It represents the best possible future outcome from the current state.

How it works:

- **Feed the Network:** The initial state of the environment is fed into the neural network.
- **Action Options:** The network spits out Q-values for all possible actions the agent can take in that state.
- **Future Potential:** These Q-values represent the expected cumulative reward of taking each action, considering both immediate gains and future possibilities.
- **Learning from Experience:** As the agent interacts with the environment, taking actions and observing outcomes, the network continuously updates its Q-values, learning which actions truly lead to the most rewarding paths.

DQN tackles a key challenge:

the variable target problem. Q-values themselves are constantly changing, making it difficult to determine the ideal target for the network's learning process. DQN solves this by employing two neural networks:



- [ ] **Primary Network:** This network actively interacts with the environment, adjusting its parameters based on new experiences.

- [ ] **Target Network:** This network has the same architecture as the primary one, but its parameters are frozen. It acts as a stable anchor point, providing consistent target values for the primary network to learn from.

Both networks work in tandem, constantly updating and refining the agent's understanding of the environment, enabling it to make optimal decisions in increasingly complex situations.

END