**Name: Swikruti Thantharate**

**Roll No.: ugmr20230005**

**Deep Learning**

**1. Introduction**

Deep learning is a subset of machine learning in artificial intelligence (AI) that mimics the workings of the human brain in processing data and creating patterns for decision-making. It involves neural networks with three or more layers. Deep learning is crucial because it enables machines to solve complex problems with higher accuracy, such as image and speech recognition, natural language processing, and autonomous driving.

**Applications:**

1. **Facial recognition and face detection** in photos.
2. **Medical imaging** for better accuracy in identifying tumors or other anomalies in X-ray and MRI images.
3. **Document analysis**.
4. **Autonomous driving**.
5. **Biometric authentication**.
6. **Object detection** by classifying objects based on shapes and patterns found within an image.

**Activation Functions :**

**Activation functions introduce non-linearity into the network, enabling it to learn and model complex data. The activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias to it.**

**Activation functions help the network capture complex patterns and dependencies in the data, enabling it to solve non-linear problems.**

**Activation Functions:**

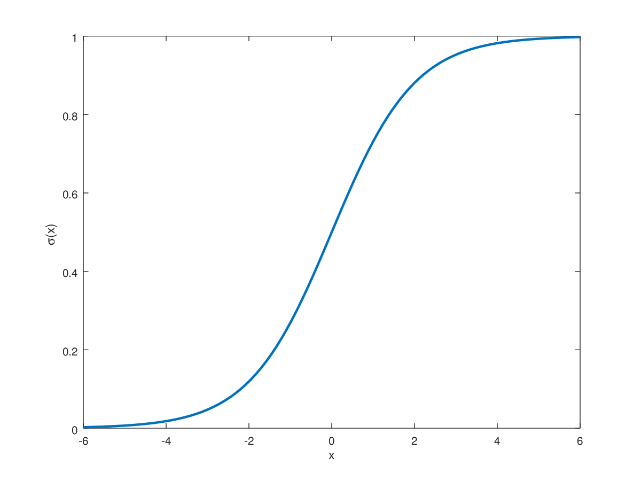
**Sigmoid:**

**• It is a function which is plotted as ‘S’ shaped graph.**

**• Equation : A = 1/(1 + e-x)**

**• Value Range : 0 to 1**

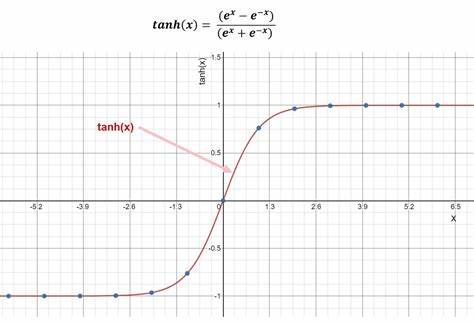
**• Uses : Usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only so, result can be predicted easily to be 1 if value is greater than 0.5 and 0 otherwise.**



**Tanh Function**

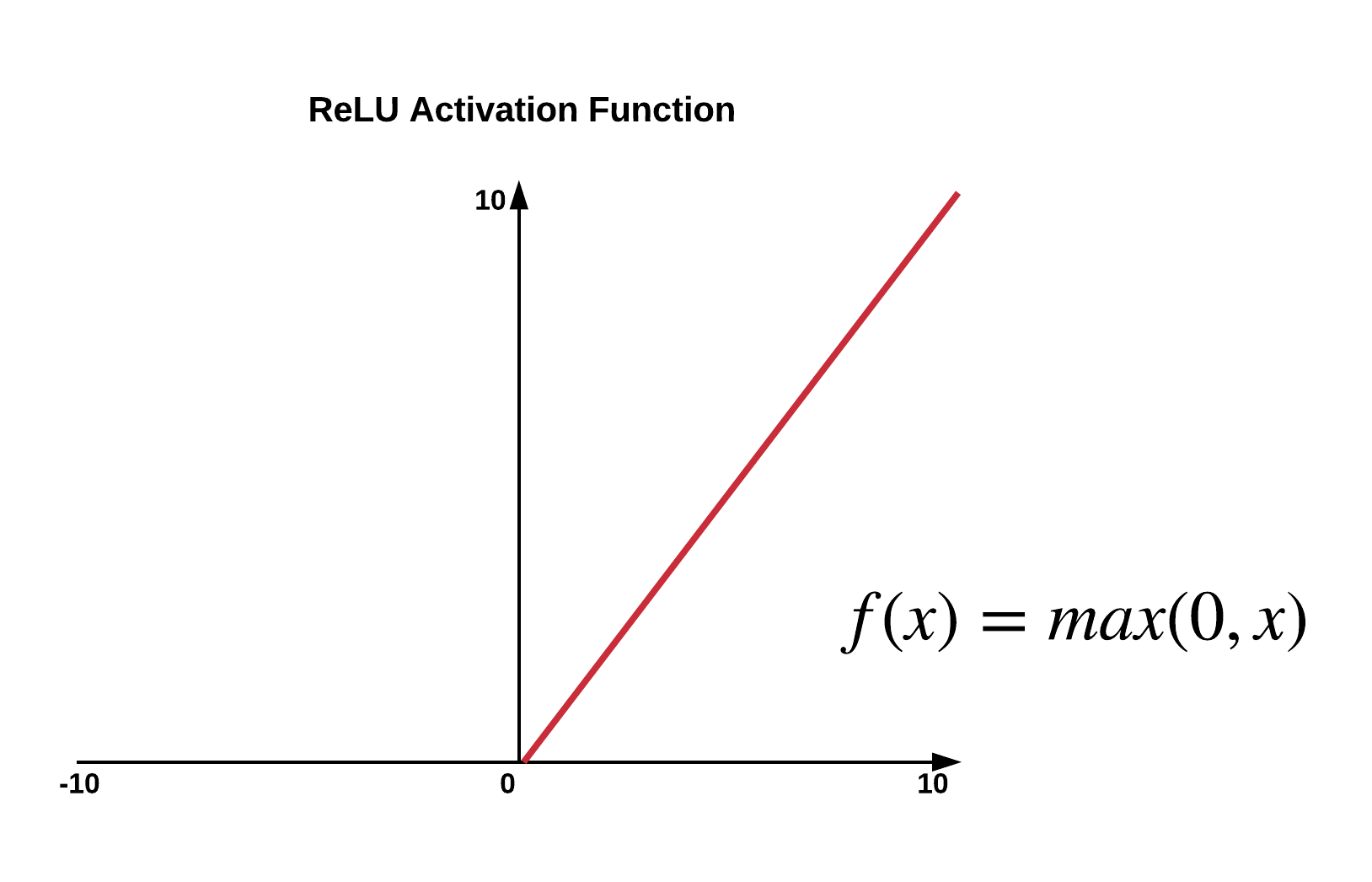
**• Value Range :- -1 to +1**

**• Uses :- Usually used in hidden layers of a neural network as it’s values lies between -1 to 1 hence the mean for the hidden layer comes out be 0 or very close to it, hence helps in centering the data by bringing mean close to 0. This makes learning for the next layer much easier.**



**RELU Function**

* It Stands for Rectified linear unit. It is the most widely used activation function. Chiefly implemented in hidden layers of Neural network.
* Equation :- A(x) = max(0,x). It gives an output x if x is positive and 0 otherwise.
* Value Range :- [0, inf)
* Uses :- ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.



**Training Process**

Training a neural network involves:

* **Forward Propagation:** Passing inputs through the network to generate outputs.
* **Loss Function:** Calculating the error between the network's output and the actual target values.
* **Backpropagation:** Adjusting the network's weights and biases based on the error gradient to minimize the loss.

**Optimization Techniques**

Optimization algorithms are used to adjust the weights and biases to minimize the loss function. Common optimizers include:

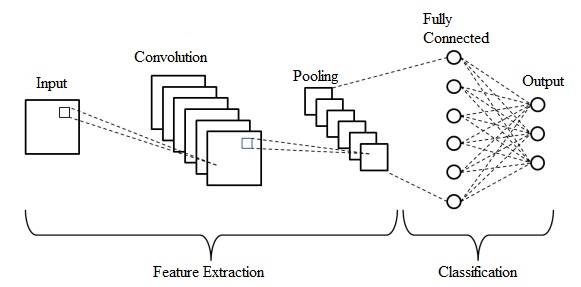
* **Stochastic Gradient Descent (SGD):** Updates weights incrementally using individual training samples.
* **Adam:** Combines the advantages of two other extensions of SGD, AdaGrad and RMSProp.

**Regularization Methods:** Regularization techniques help prevent overfitting, ensuring the model generalizes well to new data:

* **Dropout:** Randomly drops units from the neural network during training to prevent co-adaptation.
* **L2 Regularization:** Adds a penalty to the loss function based on the squared magnitude of weights.
* **Early Stopping:** Stops training when performance on a validation set starts to degrade.

**Convolutional Neural Networks (CNNs):**

CNNs are specialized neural networks designed for processing data with a grid like topology, such as images. Different from fully connected layers in MLPs, in CNN models, one or multiple convolution layers extract features from input by executing convolution operations. Each layer is a set of nonlinear functions of weighted sums at different coordinates of spatially nearby subsets of outputs from the prior layer, which allows for the reuse of the weights. CNNs are widely used for image and video recognition, as well as other tasks involving spatial data, due to their ability to capture local dependencies.



**Key Components of CNNs**

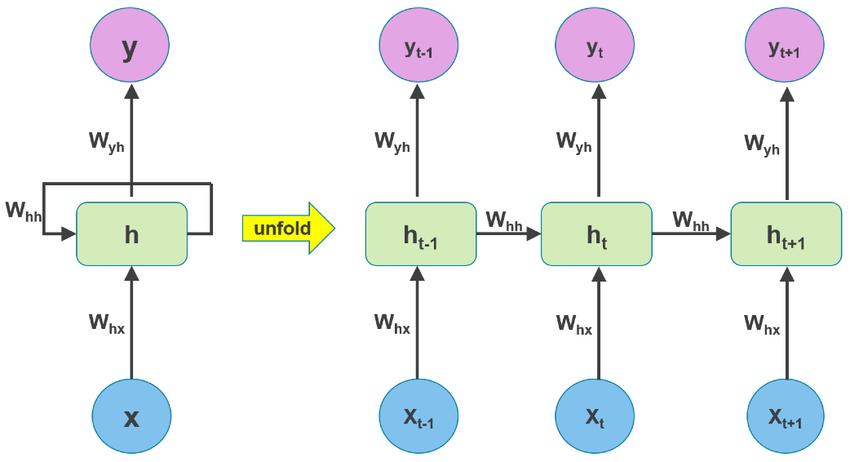
1. Convolutional Layers:
   * Convolution Operation: Applies filters/kernels to input data to extract features like edges and textures.
   * Filters/Kernels: Small matrices that slide over the input to detect specific patterns.
   * Stride: Determines the step size of the filter movement across the input.
   * Padding: Adds zeros to the input's border to maintain spatial dimensions.
2. Activation Functions:
   * ReLU (Rectified Linear Unit): Sets negative values to zero, introducing non-linearity.
3. Pooling Layers:
   * Purpose: Reduces spatial dimensions of feature maps to decrease computation and memory usage.
   * Max Pooling: Takes the maximum value from each patch of the feature map.
   * Average Pooling: Computes the average value of each patch in the feature map.
4. Fully Connected (Dense) Layers:
   * Purpose: Performs high-level reasoning and classification.
   * Structure: Neurons are connected to every neuron in the previous layer.
5. Output Layer:
   * Purpose: Produces final classification or prediction.
   * Activation Function: Typically uses Softmax for multi-class classification tasks.

These components work together in CNNs to extract hierarchical representations of input data, enabling effective feature extraction and classification for tasks like image recognition and analysis.

**Recurrent Neural Network (RNN):**

RNNs are neural networks designed for sequence data, where each node's output is dependent on previous computations. RNNs are suitable for tasks involving sequential data such as time series prediction, natural language processing, and speech recognition.

* Input Layer: Receives the sequence data.
* Hidden Layers: Each step's hidden state is a function of the previous hidden state and the current input, capturing temporal dependencies.
* Output Layer: Produces the final sequence output.



**Key Components of RNNs**

1. **Recurrent Connections:**
   * Each neuron in an RNN is connected to itself through time, enabling it to maintain a state or memory of previous inputs. This cyclic structure allows RNNs to process sequences of inputs.
2. **Long Short-Term Memory (LSTM):**

Standard RNNs can struggle to learn long-term dependencies due to the vanishing gradient problem. LSTM networks introduce a more complex cell structure that includes gates to control the flow of information. LSTM cells maintain a cell state that can carry information over long sequences, controlled by gates that regulate the flow of information.

**3.Bidirectional RNNs**:

 **Idea**: Enhances traditional RNNs by processing sequences in both forward and backward directions.

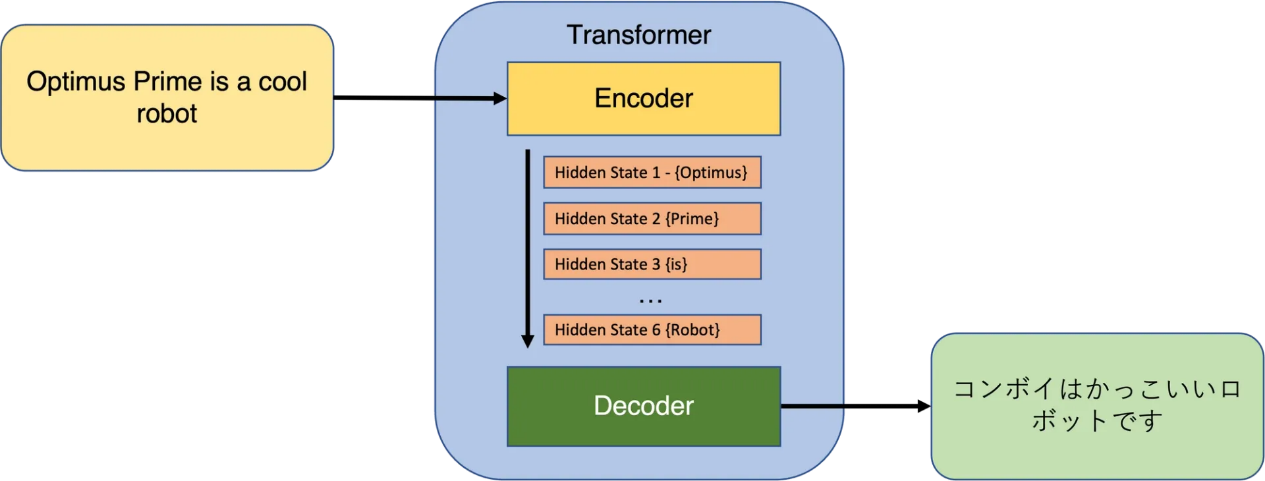
 **Application**: Useful for tasks where context from both past and future inputs is beneficial, such as speech recognition and named entity recognition.

**Transformers:**

Transformers are a model architecture designed for handling sequential data using self-attention mechanisms. They utilize self-attention mechanisms to process input sequences in parallel, addressing the limitations of recurrent neural networks (RNNs) in terms of long-range dependencies and computational efficiency. Transformers consist of an encoder-decoder architecture, where the encoder processes the input sequence and the decoder generates the output

• Encoder: Processes the input sequence, producing a representation of each element.

• Decoder: Takes the encoder's representation and generates the output sequence.



**Key Components:**

1. Self-Attention Mechanism:
   * Idea: Allows the model to focus on different parts of the input sequence by computing attention scores.
   * Formula: The attention score Attention(Q,K,V)\text{Attention}(Q, K, V)Attention(Q,K,V) for a query QQQ and key-value pairs (K,V)(K, V)(K,V) is computed as: Attention(Q,K,V)=softmax(QK⊤dk)V\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^\top}{\sqrt{d\_k}} \right) VAttention(Q,K,V)=softmax(dk​​QK⊤​)V where dkd\_kdk​ is the dimensionality of KKK and QQQ.
   * Function: Helps weigh the importance of each token in the input sequence based on its relevance to the query.
2. Transformer Architecture:
   * Stacked Encoder-Decoder Layers: Transformers are composed of multiple layers of encoders and decoders.
   * Encoder: Processes input sequences (e.g., words in a sentence) and generates contextualized representations using self-attention mechanisms.
   * Decoder: Uses encoder outputs (or a combination of encoder outputs and previous decoder outputs) to generate predictions, such as translated sentences in machine translation tasks.
   * Function: Facilitates capturing long-range dependencies efficiently through self-attention and feedforward neural networks.
3. Feedforward Neural Networks:
   * Idea: Positioned after each attention mechanism in Transformer layers, these networks independently process and transform the outputs.
   * Function: Enhances the model's ability to capture complex relationships and features in the data after attention-based transformations.

These components are integral to Transformer architectures, which have revolutionized natural language processing tasks by enabling efficient processing of sequential data and capturing contextual dependencies effectively.

**PyTorch:**

PyTorch is an open-source machine learning library developed primarily by Facebook's AI Research lab (FAIR). It is widely used for deep learning applications and provides a flexible framework for building and training neural networks.

**Key Features:**

1. **Tensor Computation:**
   * Definition: PyTorch provides efficient multi-dimensional array operations (tensors), akin to NumPy arrays but optimized for GPU acceleration.
   * Application: Used to represent and manipulate data throughout the machine learning pipeline, from input processing to model predictions.
2. **Automatic Differentiation:**
   * Feature: Enabled through PyTorch's autograd package, allowing tensors to automatically compute gradients.
   * Benefit: Facilitates the implementation of gradient-based optimization algorithms such as backpropagation, crucial for training neural networks efficiently.
3. **Dynamic Computation Graphs:**
   * Approach: PyTorch employs a dynamic computational graph, constructed on-the-fly during runtime.
   * Advantage: Offers flexibility and intuitive model construction compared to static graph frameworks, accommodating varying inputs and model structures dynamically.
4. **Modular and Extensible:**
   * Architecture: PyTorch is designed with a modular and extensible architecture.
   * Capabilities: Provides a wide range of built-in modules and utilities for defining neural network components like layers, activations, and loss functions.
   * Flexibility: Supports easy integration and customization of complex neural network architectures.
5. **Support for GPU Acceleration:**
   * Integration: PyTorch seamlessly integrates with CUDA-enabled GPUs for leveraging parallel processing capabilities.
   * Performance: Significantly accelerates computations, particularly beneficial for training deep neural networks on large datasets, enhancing overall performance.

PyTorch's combination of tensor computation, automatic differentiation, dynamic computation graphs, modularity, and GPU acceleration makes it a powerful framework for developing and training state-of-the-art deep learning models efficiently.

**Conclusion:**

Deep learning represents a pivotal advancement in machine learning, particularly through the evolution of artificial neural networks and specialized architectures like CNNs, RNNs, and Transformers. PyTorch, with its powerful tensor computation, automatic differentiation capabilities, and GPU acceleration support, has emerged as a leading framework for developing and deploying deep learning models efficiently.

**Github Repository link :** https://github.com/Swikruti23/Lab-1/blob/f0a596b6b6c2622be9e6db3647ec7b63e3138263/CNN%2C\_RNN\_and\_Transformer.ipynb

**Colab link :** https://colab.research.google.com/github/Swikruti23/Lab-1/blob/main/CNN%2C\_RNN\_and\_Transformer.ipynb