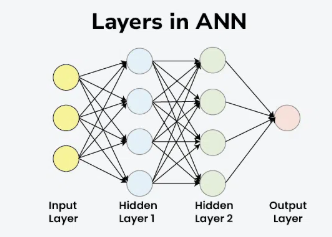
**DIFFERENT TYPES OF HIDDEN LAYER IN DEEP LEARNING**

**ASSIGNMENT:** 01 **NAME:** MANIKANDAN .S  **REG\_NUMBER :**122012173015

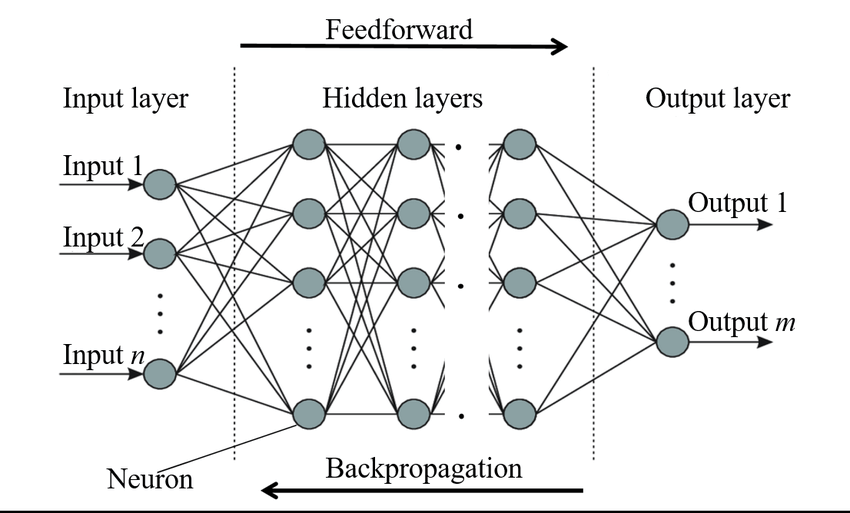
**COURSE\_NAME:** APPLIED DEEP LEARNING **COURSE\_CODE:** XAI602C

**INTRODUCTION**

* Deep learning is a type of artificial intelligence (AI) that allows computers to learn from data and develop sophisticated algorithms. It uses multiple layers of mathematical computations – known as ‘hidden layers’ – to recognize patterns in large datasets and make decisions or predictions based on these patterns.
* In an [ANN](https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/), data flows from the input layer, through one or more hidden layers, to the output layer. Each layer consists of neurons that receive input, process it, and pass the output to the next layer. The layers work together to extract features, transform data, and make predictions.
* An ANN typically consists of three primary types of layers:
  + Input Layer
  + Hidden Layers
  + Output Layer
* Each layer is composed of nodes (neurons) that are interconnected. The layers work together to process data through a series of transformations.
* The number of hidden layers needed for any given deep learning model largely depends on the complexity of the problem it must solve. Typically, more complex problems may involve a deeper network with more hidden layers, while simpler problems may only require few if any at all. Most deep learning approaches generally involve between one and 15 hidden layers; though deep networks with over 150 exist depending on the application requirements

**TYPES OF HIDDEN LAYERS IN ARTIFICIAL NEURAL NETWORKS**

**1. Dense (Fully Connected) Layer**

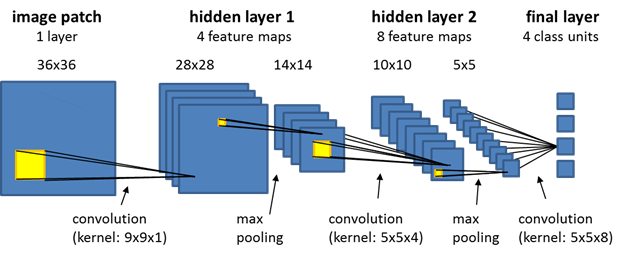
A dense layer is the most common type of hidden layer in an ANN. Every neuron in a dense layer is connected to every neuron in the previous and subsequent layers. This layer performs a weighted sum of inputs and applies an activation function to introduce non-linearity. The [activation function](https://www.geeksforgeeks.org/activation-functions-neural-networks/) (like [ReLU](https://www.geeksforgeeks.org/why-is-relu-used-as-an-activation-function/), [Sigmoid](https://www.geeksforgeeks.org/derivative-of-the-sigmoid-function/), or Tanh) helps the network learn complex patterns.

**Key Points:**

* **Role**: Learns representations from input data.
* **Function**: Performs weighted sum and activation.
* **Example**: Common in fully connected neural networks.

**2. Convolutional Layer**

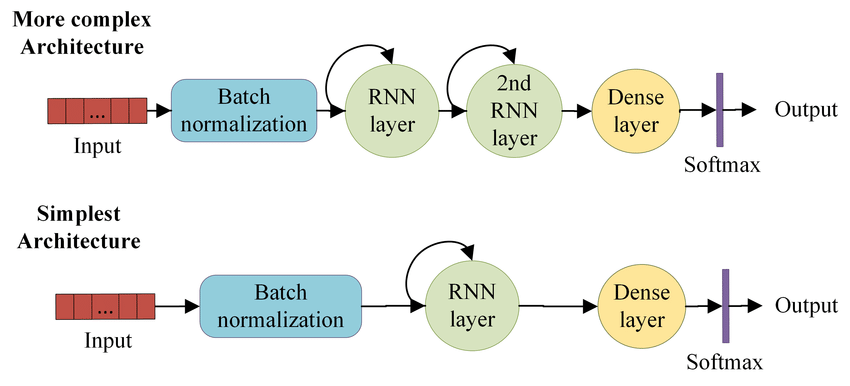
Convolutional layers are primarily used in [Convolutional Neural Networks (CNNs)](https://www.geeksforgeeks.org/convolutional-neural-network-cnn-in-machine-learning/) for image processing tasks. They apply convolution operations to the input, capturing spatial hierarchies in the data. Convolutional layers use filters to scan across the input and generate feature maps. This helps in detecting edges, textures, and other visual features.



**Key Points:**

* **Role**: Extracts spatial features from images.
* **Function**: Applies convolution using filters.
* **Example**: Detects edges and textures in images.

**3. Recurrent Layer**

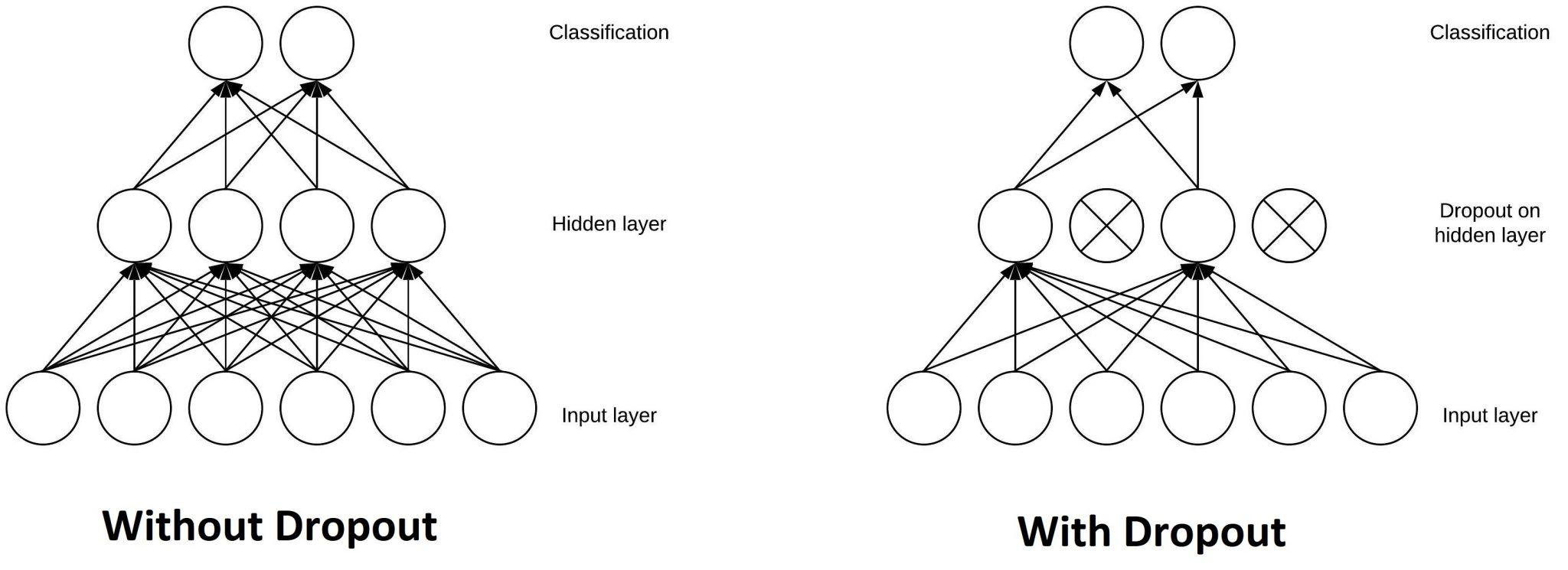
Recurrent layers, such as [Long Short-Term Memory (LSTM)](https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/) and [Gated Recurrent Unit (GRU),](https://www.geeksforgeeks.org/gated-recurrent-unit-networks/) are used in [Recurrent Neural Networks](https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/) (RNNs) for sequence data like time series or natural language. They have connections that loop back, allowing information to persist across time steps. This makes them suitable for tasks where context and temporal dependencies are important.

**Key Points:**

* **Role**: Processes sequential data with temporal dependencies.
* **Function**: Maintains state across time steps.
* **Example**: Language modeling, time series prediction.

**4. Dropout Layer**

[Dropout layers](https://www.geeksforgeeks.org/dropout-in-neural-networks/) are a [regularization](https://www.geeksforgeeks.org/regularization-in-machine-learning/) technique used to prevent overfitting. They randomly drop a fraction of the neurons during training, which forces the network to learn more robust features and reduces dependency on specific neurons. During training, each neuron is retained with a probability ppp.

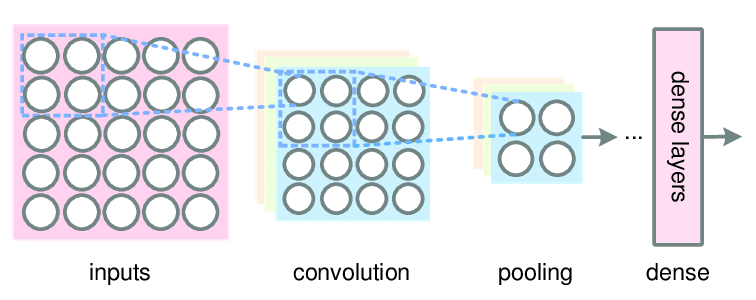


**Key Points:**

* **Role**: Prevents overfitting.
* **Function**: Randomly drops neurons during training.
* **Example**: Common in[deep learning](https://www.geeksforgeeks.org/deep-learning-tutorial/) models to improve generalization.

**5. Pooling Layer**

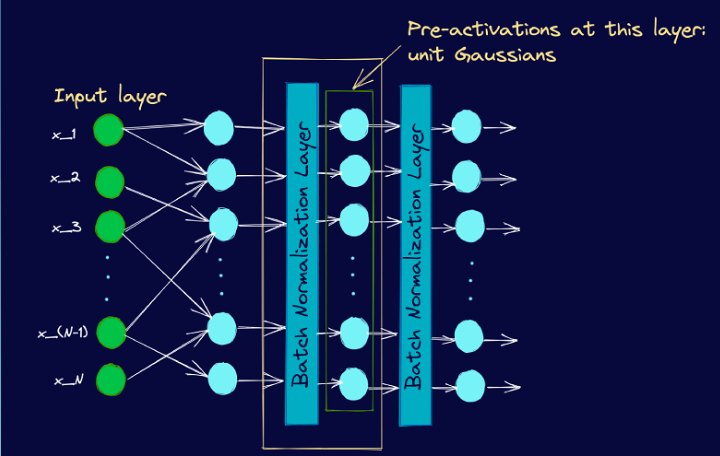
A [**Pooling Layer**](https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/) is used to reduce the spatial dimensions of the data, thereby decreasing the computational load and controlling overfitting. Common types of pooling include Max Pooling and Average Pooling.



**Use Cases:** Dimensionality reduction in CNNs

**6. Batch Normalization Layer**

A [**Batch Normalization Layer**](https://www.geeksforgeeks.org/what-is-batch-normalization-in-cnn/) normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. This helps in accelerating the training process and improving the performance of the network.



**Use Cases:** Stabilizing and speeding up training

**REFERENCE**

[Topic DL01: Activation functions and its Types in Artifical Neural network | by abhigoku10 | Medium](https://abhigoku10.medium.com/activation-functions-and-its-types-in-artifical-neural-network-14511f3080a8)

[Fully connected (dense) artificial neural network. | Download Scientific Diagram](https://www.researchgate.net/figure/Fully-connected-dense-artificial-neural-network_fig1_358145060)

**DIFFERENT TYPES OF ACTIVATION FUNCTION IN DEEP LEARNING**

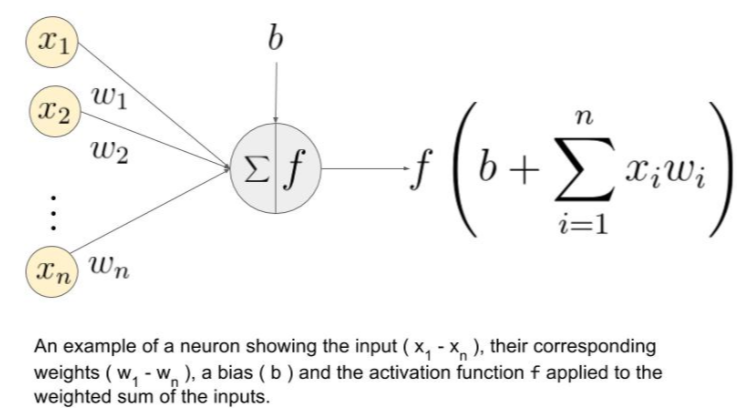
**ASSIGNMENT:** 02 **NAME:** MANIKANDAN .S **REG\_NUMBER :**122012173015

**COURSE\_NAME:** APPLIED DEEP LEARNING **COURSE\_CODE:**XAI602C

**INTRODUCTION**

* Deep learning algorithms are multi-level representation learning techniques that allow simple non-linear modules to transform representations from the raw input into the higher levels of abstract representations, with many of these trans- formations producing learned complex functions.
* The DL research was inspired by the limitations of the conventional learning algorithms especially being limited to processing data in raw form, and the human learning techniques by changing the weights of the simulated neural connections based on experiences, obtained from past data.
* The use of representation learning, which is the technique that allows machines to discover relationships from raw data, needed to perform certain tasks like classification and detection. Deep learning, a sub-field of machine learning (ML), is more recently being referred to as representation learning in some literature. The typical
* artificial neural networks (ANN) are biologically inspired computer pro- grammes, designed by the inspiration of the workings of the human brain. These ANNs are called networks because they are composed of different functions, which gathers knowledge by detecting the relationships and patterns in data using past experiences known as training examples in most literature.

The learned patterns in data are modified by an appropriate AF and presented as the output of the neuron as



**ACTIVATION FUNCTION IN NEURAL NETWORK?**

* As observed for the above figure when we do not have the activation function the weights and bias would simply do a linear transformation.
* A linear equation is simple to solve but is limited in its capacity to solve complex problems and have less power to learn complex functional mappings from data. A neural network without an activation function is just a linear regression model.
* The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks. We would want our neural networks to work on complicated datas like *videos, audio, speech etc*. Linear transformations would never be able to perform such tasks

**WHAT CONDITION THE ACTIVATION FUNCTION SHOULD SATISFY?**

* Activation functions make the back-propagation possible since the gradients are supplied along with the error to update the weights and biases. Without the differentiable non linear function, this would not be possible.
* So the functions should be differentiable and monotonic.
* **Derivative or Differential:** Change in y-axis w.r.t. change in x-axis.It is also known as slope.
  + - **Monotonic function:** A function which is either entirely non-increasing or non-decreasing.

**1.LINEAR OR IDENTITY ACTIVATION FUNCTION**

* As you can see the function is a line or linear.Therefore, the output of the functions will not be confined between any range*.*



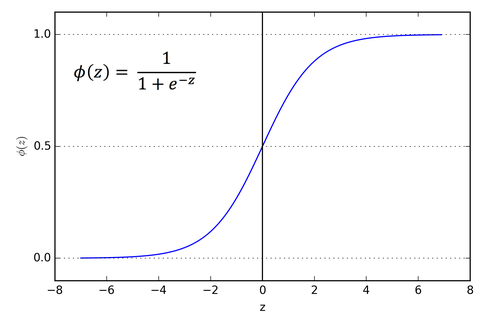
**Equation :**f(x) = x

**Range :** (-infinity to infinity)

**2.NON-LINEAR ACTIVATION FUNCTION**

* The Nonlinear Activation Functions are the most used activation functions.It makes it easy for the model to generalize or adapt with variety of data and to differentiate between the output.
* The Nonlinear Activation Functions are mainly divided on the basis of their **range or curves**-

**1. SIGMOID OR LOGISTIC ACTIVATION FUNCTION**

The Sigmoid Function curve looks like a S-shape.

**Equation :** f(x) = 1 / 1 + exp(-x)

**Range :** (0 to 1)

**Pros:**

1.The function is **differentiable**.That means, we can find the slope of the sigmoid curve at any two points

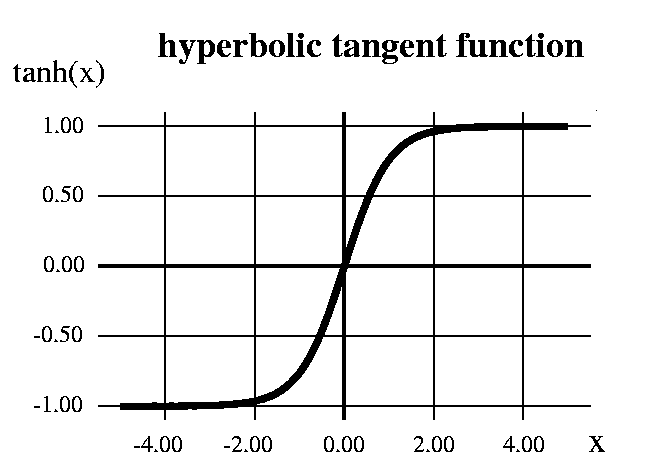
2.The function is **monotonic**but function’s derivative is not

**Cons:**

1.It gives rise to a problem of “**vanishing gradients**”, since the Y values tend to respond very less to changes in X

2.Secondly , its output isn’t zero centered. It makes the gradient updates go too far in different directions. **0 < output < 1, and it makes optimization harder.**

**2. TANH OR HYPERBOLIC TANGENT ACTIVATION FUNCTION:**



**Equation :** **f(x) = 1 — exp(-2x) / 1 + exp(-2x) or 2 \*sigmoid(2x)-1**

**Range :** (-1 to 1)

**Pros:**

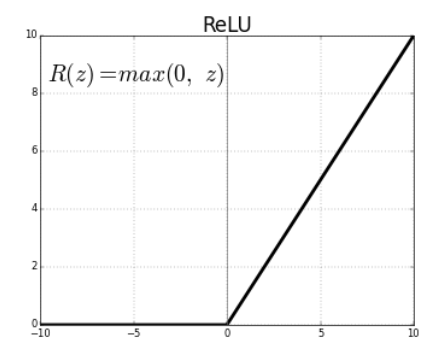
1. The function and its **derivative**both are**monotonic**
2. Output is zero centered
3. Optimization is *easier*

**Cons:**

1. It also suffers vanishing gradient problem
2. It saturate and kill gradients.

**3. RELU (RECTIFIED LINEAR UNIT) ACTIVATION FUNCTION**

The ReLU is the most used activation function in the world right now



**Equation :** **f(x) = max(0,x)**

**Range :** (0 to infinity)

**Pros:**

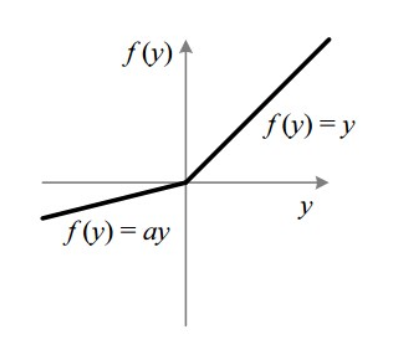
1. The function and its **derivative**both are**monotonic**.
2. Due to its functionailty it does not activate all the neuron at the same time
3. It is efficient and easy for computation.

**Cons:**

1. The outputs are not zero centered similar to the sigmoid activation function
2. When the gradient hits zero for the negative values, it does not converge towards the minima which will result in a dead neuron while back propagation.

**4. LEAKY RELU**

To solve the ReLU problem we have leaky ReLU



**Equation :** f(x) = ax for x<0 and x for x>0

**Range :** (0.01 to infinity)

**Pros:**

1. The function and its **derivative**both are**monotonic**
2. It allows negative value during back propagation
3. It is efficient and easy for computation.

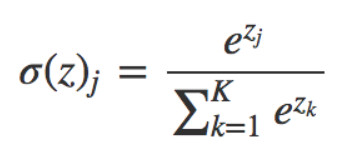
**Cons:**

1. Results are not always consistent
2. During the front propagation if the learning rate is set very high it will overshoot killing the neuron

* The idea of leaky ReLU can be extended even further. Instead of multiplying x with a constant term we can multiply it with a hyperparameter which seems to work better the leaky ReLU. This extension to leaky ReLU is known as **Parametric ReLU**.

**5. SOFTMAX**

* The softmax function is also a type of sigmoid function but it is very useful to handle classification problems having multiple classes .



* The softmax function is shown above, where z is a vector of the inputs to the output layer (if you have 10 output units, then there are 10 elements in z). And again, j indexes the output units, so j = 1, 2, …, K.
* The softmax function is ideally used in the output layer of the classifier where we are actually trying to attain the probabilities to define the class of each input.

**REFERENCE**

[Topic DL01: Activation functions and its Types in Artifical Neural network | by abhigoku10 | Medium](https://abhigoku10.medium.com/activation-functions-and-its-types-in-artifical-neural-network-14511f3080a8)

**DIFFERENT TYPES OF OPTIMIZER IN PYTHON FOR NEURAL NETWORKS**

**ASSIGNMENT:** 03 **NAME:** MANIKANDAN .S  **REG\_NUMBER :**122012173015

**COURSE\_NAME:** APPLIED DEEP LEARNING **COURSE\_CODE:** XAI602C

**INTRODUCTION**

Deep learning, a subset of machine learning, tackles intricate tasks like speech recognition and text classification. Comprising components such as activation functions, input, output, hidden layers, and loss functions, deep learning models aim to generalize data and make predictions on unseen data

To optimize these models, various algorithms, known as optimizers, are employed. Optimizers adjust model parameters iteratively during training to minimize a loss function, enabling neural networks to learn from data. This guide delves into different optimizers used in deep learning, discussing their advantages, drawbacks, and factors influencing the selection of one optimizer over another for specific applications.

Common optimizers include Stochastic Gradient Descent (SGD), Adam, and RMSprop, each employing specific update rules, learning rates, and momentum for refining model parameters. Optimizers play a pivotal role in enhancing accuracy and speeding up the training process, shaping the overall performance of deep learning models

**NEED FOR OPTIMIZERS IN DEEP LEARNING**

Choosing an appropriate optimizer for a deep learning model is important as it can greatly impact its performance. Optimization algorithms have different strengths and weaknesses and are better suited for certain problems and architectures.

**FEW POINTERS TO KEEP IN MIND WHEN CHOOSING AN OPTIMIZER:**

* Understand the problem and model architecture, as this will help you determine which optimizer is most suitable
* Experiment with different optimizers in deep learning to see which one works best for your problem
* Adjust the hyperparameters of the optimizer, such as the learning rate, to see if it improves performance
* Remember that the optimizer's choice is not the only factor affecting model performance.
* Other important factors include the choice of architecture, the quality of the data, and the amount of data available.

**TYPES OF OPTIMIZERS**

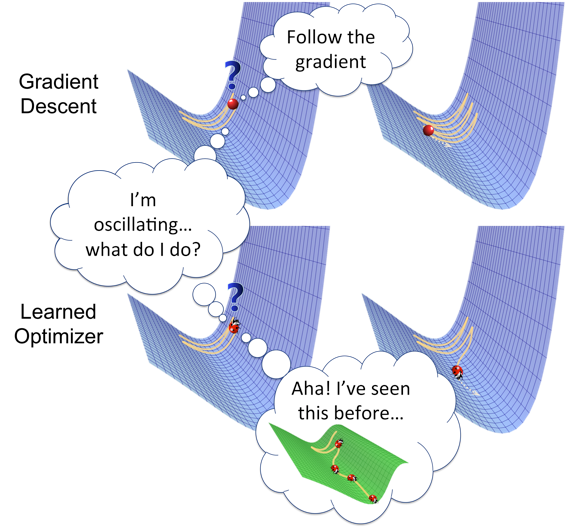
Many types of optimizers are available for training machine learning models, each with its **own strengths and weaknesses**. Some optimizers are better suited for certain types of models or data, while others are more general-purpose.

This section will briefly overview the most commonly used optimizers, starting with the simpler ones and progressing to the more complex ones.

**1.GRADIENT DESCENT**

Gradient descent is a simple optimization algorithm that updates the model's parameters to minimize the loss function. We can write the basic form of the algorithm as follows:

θ=θ−α⋅∇θL(θ)*θ*=*θ*−*α*⋅∇*θ*​*L*(*θ*)

where θ*θ* is the model parameter, L(θ)*L*(*θ*) is the loss function, and α*α* is the learning rate.

**Pros:**

* Simple to implement.
* Can work well with a well-tuned learning rate.

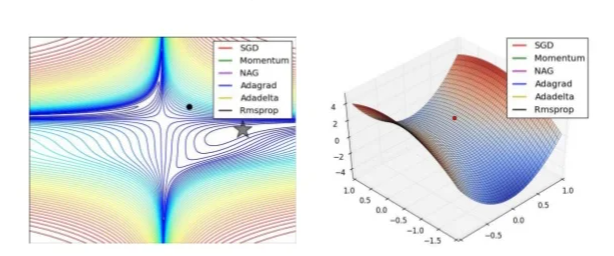
**Cons:**

* It can converge slowly, especially for complex models or large datasets.
* Sensitive to the choice of learning rate.

**2.STOCHASTIC GRADIENT DESCENT**

Stochastic gradient descent (SGD) is a variant of gradient descent that involves updating the parameters based on a small, randomly-selected subset of the data (i.e., a "mini-batch") rather than the full dataset. We can write the basic form of the algorithm as follows:

θ=θ−α⋅∇θL(θ;x(i);y(i))*θ*=*θ*−*α*⋅∇*θ*​*L*(*θ*;*x*(*i*);*y*(*i*))

where (x(i),y(i))(*x*(*i*),*y*(*i*)) is a mini-batch of data. 

**Pros:**

* It can be faster than standard gradient descent, especially for large datasets.
* Can escape local minima more easily.

**Cons:**

* It can be noisy, leading to less stability.
* It may require more hyperparameter tuning to get good performance.

**3.STOCHASTIC GRADIENT DESCENT WITH MOMENTUM**

SGD with momentum is a variant of SGD that adds a "momentum" term to the update rule, which helps the optimizer to continue moving in the same direction even if the local gradient is small. The momentum term is typically set to a value between 0 and 1.

**Pros:**

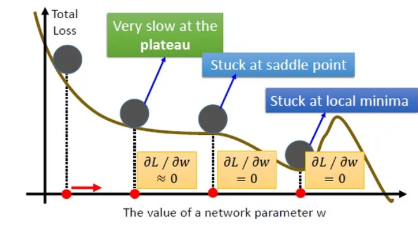
* It can help the optimizer to move more efficiently through "flat" regions of the loss function.
* It can help to reduce oscillations and improve convergence.

**Cons:**

* Can overshoot good solutions and settle for suboptimal ones if the momentum is too high.
* Requires tuning of the momentum hyperparameter.

**Mini-Batch Gradient Descent**

Mini-batch gradient descent is similar to SGD, but instead of using a single sample to compute the gradient, it uses a **small, fixed-size "mini-batch" of samples**. The update rule is the same as for SGD, except that the gradient is averaged over the mini-batch. This can reduce noise in the updates and improve convergence.



**Pros:**

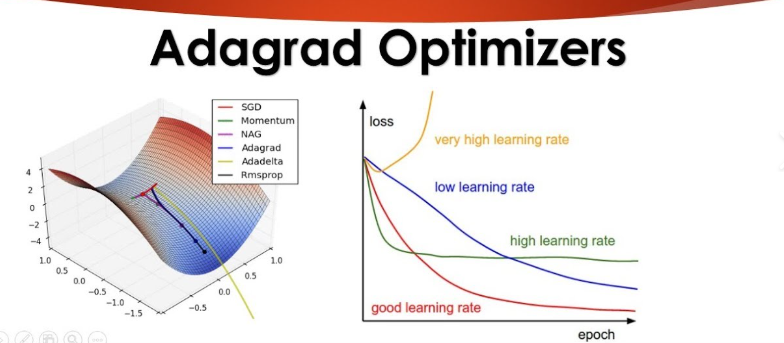
* It can be faster than standard gradient descent, especially for large datasets.
* Can escape local minima more easily.
* Can reduce noise in updates, leading to more stable convergence.

**Cons:**

* Can be sensitive to the choice of mini-batch size.

**4.ADAGRAD**

Adagrad is an optimization algorithm that **uses an adaptive learning rate per parameter**. The learning rate is updated based on the historical gradient information so that parameters that receive many updates have a lower learning rate, and parameters that receive fewer updates have a larger learning rate. The update rule can be written as follows:

**Pros:**

* It can work well with sparse data.
* Automatically adjusts learning rates based on parameter updates.

**Cons:**

* Can converge too slowly for some problems.
* Can stop learning altogether if the learning rates become too small.

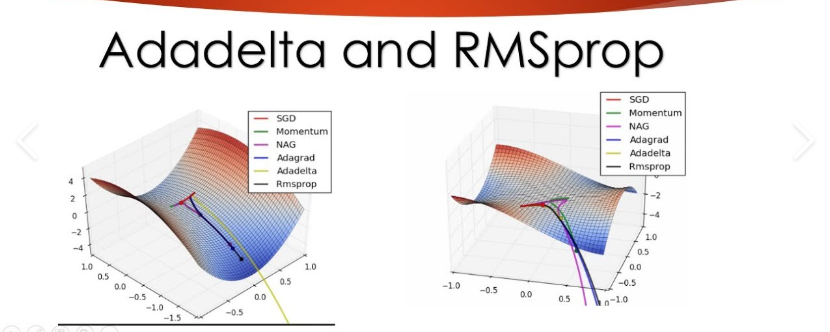
**5.RMSPROP**

RMSProp is an optimization algorithm similar to Adagrad, but it **uses an exponentially decaying average** of the squares of the gradients rather than the sum. This helps to reduce the monotonic learning rate decay of Adagrad and improve convergence.

**Pros:**

* It can work well with sparse data.
* Automatically adjusts learning rates based on parameter updates.
* Can converge faster than Adagrad.

**Cons:**

* It can still converge too slowly for some problems.
* Requires tuning of the decay rate hyperparameter.
* 

**6.ADADELTA**

AdaDelta is an optimization algorithm similar to RMSProp but does not require a hyperparameter learning rate. Instead, it uses an **exponentially decaying average** of the gradients and the squares of the gradients to determine the updated scale.

**Pros:**

* Can work well with sparse data.
* Automatically adjusts learning rates based on parameter updates.

**Cons:**

* Can converge too slowly for some problems.
* Can stop learning altogether if the learning rates become too small.

**7.ADAM**

Adam (short for "adaptive moment estimation") is an optimization algorithm that combines the ideas of SGD with momentum and RMSProp. It **uses an exponentially decaying average of the gradients** and the squares of the gradients to determine the updated scale, similar to RMSProp. It also uses a momentum term to help the optimizer move more efficiently through the loss function.

Where m*m* and v*v* are the momentum and velocity vectors, respectively, and β1*β*1​ and β2*β*2​ are decay rates for the momentum and velocity.



**Pros:**

* Can converge faster than other optimization algorithms.
* Can work well with noisy data.

**Cons:**

* It may require more tuning of hyperparameters than other algorithms.
* May perform better on some types of problems.

**How Do Optimizers Work in Deep Learning?**

Optimizers in deep learning adjust the model's parameters to minimize the loss function. The loss function measures how well the model can make predictions on a given dataset, and the goal of training a model is to find the set of model parameters that yields the lowest possible loss.

The optimizer uses an optimization algorithm to search for the parameters that minimize the loss function. The optimization algorithm uses the gradients of the loss function to the model parameters to determine the direction in which we should adjust the parameters.

The **gradients** are computed using backpropagation, which involves applying the chain rule to compute the gradients of the loss function to each of the model parameters.

The **optimization algorithm** then adjusts the model parameters to minimize the loss function. This process is repeated until the loss function reaches a minimum or the optimizer reaches the maximum number of allowed iterations.

**REFERENCE**

[Topic DL01: Activation functions and its Types in Artifical Neural network | by abhigoku10 | Medium](https://abhigoku10.medium.com/activation-functions-and-its-types-in-artifical-neural-network-14511f3080a8)

[Complete Guide to Adam Optimization | by Layan Alabdullatef | Medium](https://medium.com/@LayanSA/complete-guide-to-adam-optimization-1e5f29532c3d)

**DIFFERENT TYPES OF ENCODER IN PYTHON FOR NEURAL NETWORKS**

**ASSIGNMENT:** 04 **NAME:** MANIKANDAN .S  **REG\_NUMBER :**122012173015

**COURSE\_NAME:** APPLIED DEEP LEARNING **COURSE\_CODE:** XAI602C

**INTRODUCTION**

Encoding is a technique of converting categorical variables into numerical values so that it could be easily fitted to a machine learning model.

Before getting into the details, let’s understand about the different types of categorical variables.

Nominal categorical variable:

Nominal categorical variables are those for which we do not have to worry about the arrangement of the categories.

Example,

i. suppose we have a gender column with categories as Male and Female.  
ii. We can also have a state column in which we have different states like NY, FL, NV, TX  
So here we don’t have to worry about the arrangement of the categories.

Ordinal Categorical variable :

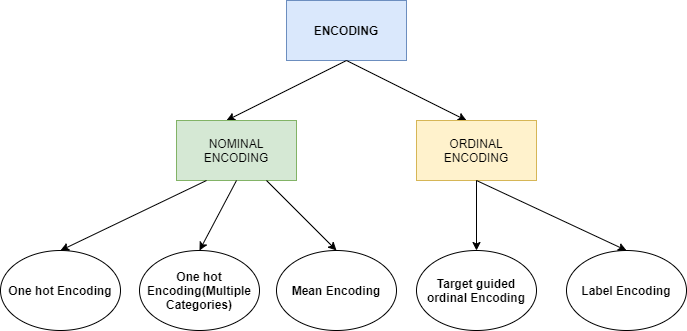
Ordinal categories are those in which we have to worry about the rank. These categories can be rearranged based on ranks.

Example,

i. Suppose in a dataset there is an education column which we will use to predict the salary of the person. The education column has categories like ‘bachelors’,’masters’,’PHD’. Based on the above categorieswe can rearrange this and assign ranks to each category. Based on the education level ‘PHD’ will get the highest rank (PHD-1, masters-2, bachelors-3).

Now that we have discussed about the type of categorical variables, let’s see the different types of encoding:

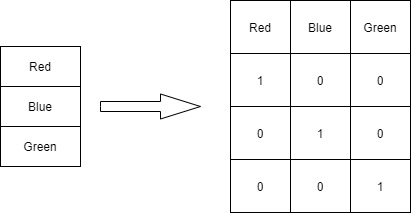
1. Nominal Encoding
2. Ordinal Encoding

****

1. One Hot Encoding

This method is applied to nominal categorical variables.

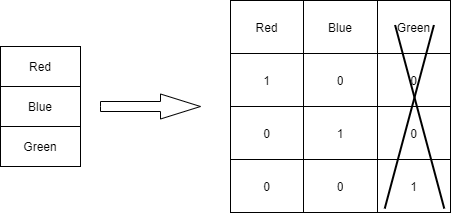
Example, suppose we have a column containing 3 categorical variables, then in one hot encoding 3 columns will be created each for a categorical variable.



One Hot Encoding

Dummy Variable Trap

We can skip the last column ‘Green’ as 0,0 signifies green. This means, suppose we have ‘n’ columns, then the one hot encoding should create ‘n-1’ columns.



Dummy Variable Trap

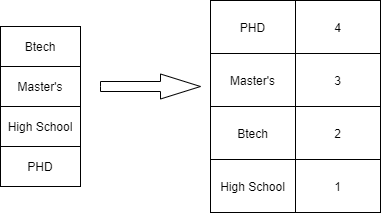
Disadvantage

Suppose we have a column which has 100 categorical variables. Now if we try to convert the categorical variables into dummy variable then we will get 99 columns. This will increase the dimension of the overall dataset which will lead to curse of dimensionality.

So basically, if there is a lot of categorical variables in a column then we should not apply this technique.

2. Label Encoding

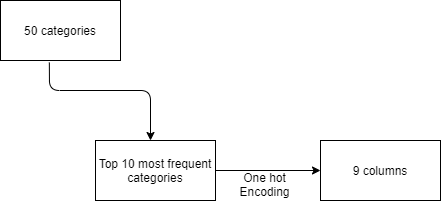
This technique will be used only for Ordinal categories. Ranks are provided based on the importance of the category. Below table illustrates that PHD is considered as the highest degree, so the highest label is given to it and so on.

****

Label Encoding

3. One hot Encoding (Multiple Categories) — Nominal Categories

In this method, we will consider only those categories which has the most number of repetitions and we will consider the top 10 repeating categories and apply one hot-encoding to only those categories.

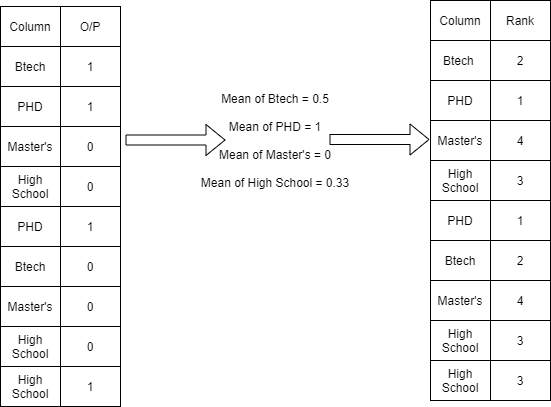
****

One hot encoding-multiple categories

The above technique was for Nominal variables. What shall we do if such kind of scenario arises for Ordinal variable. Let’s see how to handle such scenario.

4. Target guided ordinal categories

In this method, we calculate the mean of each categorical variable based on the output and then rank them. Below table illustrates this.

****

Target Encoding

We can apply this technique but cant do this with nominal as we dont know the order in case of nominal variables unlike in the case of Ordinal where we know the order of variables.

To overcome this limitation for Nominal variables we use another technique called Mean Encoding

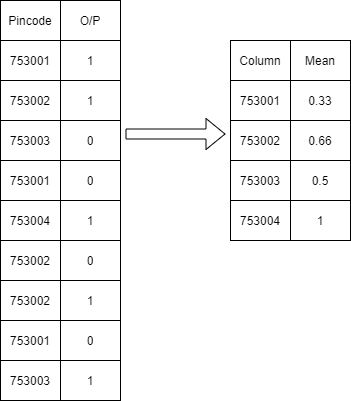
5. Mean Encoding

In this method, we will convert the categories into their mean values based on the output.

This type of approach will be applicable where we have a lot of categorical variables for a particular column.

Example, suppose we have a column as pincode which contains all the pincodes of a city. It will contain many pincodes with multiple occurances. To encode we can use this technique which will convert all the pincodes into their mean values based on the output column.

Below table will illustrate the approach:

****

**Mean Encoding**