# Deep Learning

# What is Deep Learning

Deep Learning is collection of statistical techniques of machine learning for learning feature hierarchies that are actually based on artificial neural networks.

# Deep learning vs Machine learning

|  |  |  |
| --- | --- | --- |
|  | Deep Learning | Machine Learning |
| **Data** | Needs a big data | Performs well with a small to a medium dataset |
| **Hardware requirements** | Requires machines with GPU | Works with low-end machines |
| **Engineering peculiarities** | Needs to understand the basic functionality of the data | Understands the features and how they represent the data |
| **Training time** | Long | Short |
| **Processing time** | A few hours or weeks | A few seconds or hours |
| **Number of Algorithms** | Few | Many |
| **Data interpretation** | Difficult | Some ML algorithms are easy to interpret, whereas some are hardly possible |

# What is Neuron and Neural Networks, Types of Deep learning Networks

## Neuron

Neurons are nerve cells that send messages all over your body to allow you to do everything from breathing to talking, eating, walking, and thinking.

## Neural Networks

A neural network is a machine learning model inspired by the human brain, using interconnected nodes (neurons) to process data and learn patterns.

## Types of Deep Learning Networks

* **Perceptron**
* **Feed Forward Networks**
* **Multi-Layer Perception (ANN)**
* **Radial Based Networks**
* **Convolutional Neural Networks** (use for image data)
* **Recurrent Neural Networks** (use for text data)
* **Long Short-Term Memory Networks**

# Single Layer Perceptron

A single-layer perceptron is a fundamental building block in deep learning, representing the simplest form of a neural network. It consists of a single layer of neurons that receive inputs, perform a weighted sum, and apply an activation function to produce an output. This output is typically a binary classification (0 or 1) or a continuous value.

# Perceptron Work

dataset = pd.read\_csv("preceptron\_customer\_purchase\_dataset.csv")

dataset.head(3)

plt.figure(figsize=(4,3))

sns.scatterplot(x="Age", y="AdClicks", data=dataset, hue="Purchase")

plt.show()

x = dataset.iloc[:,:-1]

y = dataset["Purchase"]

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=42)

from sklearn.linear\_model import Perceptron

pr = Perceptron() # alpha=0.01

pr.fit(x\_train, y\_train)

pr.score(x\_train, y\_train)\*100, pr.score(x\_test, y\_test)\*100

from mlxtend.plotting import plot\_decision\_regions

plt.figure(figsize=(4,3))

plot\_decision\_regions(x.to\_numpy(),y.to\_numpy(),clf=pr)

plt.show()

# Multi-Layer Perceptron (ANN)

A Multilayer Perceptron (MLP) is a type of neural network that uses multiple layers of interconnected nodes (neurons) to learn complex patterns and relationships in data. It's a fundamental architecture in deep learning, known for its ability to handle non-linear problems and learn from training data.

* **Input Layer:** Receives the initial data or features as input.
* **Hidden Layers:** Perform computations on the input data, transforming it and extracting features. A network with one or more hidden layers is considered a deep neural network.
* **Output Layer:** Generates the final prediction or result based on the processed information from the hidden layers.

# Forward Propagation and Back Propagation

## Forward Propagation

Forward propagation is the process of passing input data through a neural network, layer by layer, to generate an output.

## Back Propagation

Back Propagation is one of the important concepts of a neural network. Our task is to classify our data best. For this, we have to update the weights of parameter and bias.

# Activation Functions for Neutral Networks

# Activation Function

An activation function decides whether a neuron should be activated or not. This means that it will decide whether the neuron’s input to the network is important or not in the process of prediction using simpler mathematical operations.

**The activation function is categorized into two main parts:**

* Binary Step Function
* Linear Activation Function
* Non-Linear Activation Function

## Binary Step Function

Binary step function depends on a threshold value that decides whether a neuron should be activated or not.

## Linear Activation Function

The output of functions is not restricted in between any range. Its range is specified from -infinity to infinity.

## Non-Linear Neural Networks Activation Functions

* Sigmoid / Logistic Activation Function
* Tanh Function (Hyperbolic Tangent)
* ReLU Function
* Softmax Function

## How to Choose the Right Activation Function?

* A few rules for choosing activation function for output layer based on the type of prediction problem.
* **Regression –** Linear Activation Function
* **Binary Classification –** Sigmoid / Logistic Activation Function
* **Multiclass Classification –** Softmax Function
* **Multilabel Classification –** Sigmoid
* The Activation Function used in hidden layers is typically chosen based on the type of neural network architecture:
* **Convolutional Neural Network (CNN):** ReLU Activation Function
* **Recurrent Neural Network (RNN):** Tanh and / or Sigmoid Activation Function

# Loss Functions

The loss function is a method of evaluating how well your algorithm is modeling your dataset. It is a mathematical function of the parameters of the machine learning algorithms.

## Regression:

* MSE (Mean Squared Error) use it in all regression problems but not use it when outlier present in your dataset
* MAE (Mean Absolute Error) use it when many outliers present in your dataset
* Hubber Loss use it when 30% to 40% outliers present in your dataset

## Classification:

* Binary Cross-Entropy use when dataset output is binary format like 0, 1
* Categorical Cross-Entropy use when data is categorical format like cat, dog, cow
* Sparse Categorical Cross-Entropy use when data is multiple classes like 10 to 20 class
* **Categorical cross-entropy and sparse categorical cross-entropy:**
* If target column has one hot encode to classes like 0 0 1, 0 1 0, 1 0 0 then use categorical cross-entropy.
* If target column has numerical encoding to classes like 1, 2, 3, 4….N then use sparse categorical cross entropy.

## Auto-Encoder:

* KL Divergence

## GAN:

* Discriminator loss
* Minmax GAN loss

## Object detection:

* Focal loss

## Word embedding:

* Triplet loss

# Optimizer in neural network

Optimizers are algorithms or methods used to change the attributes of your neural network such as weights and learning rate in order to reduce the losses.

## Types of Optimizer

* Gradient Descent
* Stochastic Gradient Descent
* Stochastic Gradient Descent with momentum
* Mini-Batch Gradient Descent
* Adagrad
* RMSProp
* AdaDelta
* Adam (mostly use this)

# Practical Multi-layer Perceptron - ANN

dataset = pd.read\_csv("mlp\_practice\_dataset.csv")

dataset.head(3)

input\_data = dataset.iloc[:,:-1]

output\_data = dataset.iloc[:,-1]

input\_data.shape

from sklearn.preprocessing import StandardScaler important in neural network

ss = StandardScaler()

input\_data = pd.DataFrame(ss.fit\_transform(input\_data), columns=input\_data.columns)

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(input\_data, output\_data, test\_size=0.2, random\_state=42)

import tensorflow

from keras.layers import Dense

from keras.models import Sequential

ann = Sequential()

ann.add(Dense(4,input\_dim = 6, activation="relu"))

ann.add(Dense(2, activation="relu"))

ann.add(Dense(1, activation="sigmoid"))

ann.compile(optimizer='adam', loss="binary\_crossentropy", metrics=["accuracy"])

ann.fit(x\_train, y\_train, batch\_size=50, epochs = 10)

prd1 = ann.predict(x\_train)

prd\_data1 = []

for i in prd1:

if i > 0.5:

prd\_data1.append(1)

else:

prd\_data1.append(0)

prd = ann.predict(x\_test)

prd\_data = []

for i in prd:

if i > 0.5:

prd\_data.append(1)

else:

prd\_data.append(0)

from sklearn.metrics import accuracy\_score

accuracy\_score(y\_test, prd\_data)

accuracy\_score(y\_train, prd\_data1)

prd1 = ann.predict(np.array([[-1.625039, 0.910547, 0.485495, -1.134185, 1.240032, -1.178237]])) for checking it

prd\_data1 = []

for i in prd1:

if i[0] > 0.5:

prd\_data1.append(1)

else:

prd\_data1.append(0)

prd\_data1

# Improve the Performance of a Neural Network

* **Hyper Parameter**

No. of hidden layers, no. of nodes, activation function, loss function, optimizer, batch size, no of epochs, learning rate

* **Vanishing / Exploding Gradient**

Optimizer, initialize weight, batch normalization

* **Data**

Transfer learning for data creation

* **Slow Training**
* **Overfitting**

L1 & L2 regularization, Early stopping, batch normalization

# Identify Overfitting in deep learning (Early stopping, regularization)

## Identify Overfitting

Overfitting is a common explanation for the poor performance of a predictive model.

Overfitting refers to an unwanted behavior of a machine learning algorithm used for predictive modeling.

## Overfitting techniques

* Cross Validation - Train with more data - Remove Features
* Early Stopping - Regularization - Ensembling - Hyper parameter

## Practical – Early Stopping

from keras.callbacks import EarlyStopping

ann.fit(x\_train, y\_train, batch\_size=50, epochs = 10, **validation\_data=(x\_test, y\_test), callbacks = EarlyStopping()**)

train\_accuracy = ann.history.history["accuracy"]

test\_accuracy = ann.history.history["val\_accuracy"]

plt.plot([i for i in range(1,11)],train\_accuracy)

plt.plot([i for i in range(1,11)],test\_accuracy)

plt.show()

## Practical – Regularization

from keras.regularizers import L1, L2, L1L2

ann.add(Dense(4,input\_dim = 6, activation="relu**", kernel\_regularizer=L2(l2=0.01)**))

ann.add(Dense(2, activation="relu", **kernel\_regularizer=L2(l2=0.01)**))

ann.add(Dense(1, activation="sigmoid"))

# Batch Normalization

Batch normalization is a supervised learning method for normalizing the interlayer outputs of a neural network. As a result, the next layer receives a “reset” of the output distribution from the preceding layer, allowing it to analyze the data more effectively.

## Practical – Batch Normalization

**from keras.layers import BatchNormalization**

ann.add(Dense(4,input\_dim = 6, activation="relu"))

**ann.add(BatchNormalization())**

ann.add(Dense(2, activation="relu"))

**ann.add(BatchNormalization())**

ann.add(Dense(1, activation="sigmoid"))

# Dropout Layer

All the forward and backwards connections with a dropped node are temporarily removed, thus creating a new network architecture out of the parent network. The nodes are dropped by a dropout probability of p.

## Practical – Dropout Layer

**from keras.layers import Dropout**

ann.add(Dense(4,input\_dim = 6, activation="relu"))

**ann.add(Dropout(0.5))**

ann.add(Dense(2, activation="relu"))

**ann.add(Dropout(0.5))**

ann.add(Dense(1, activation="sigmoid"))

# Vanishing Gradient Problem

The Vanishing gradient problem is encountered when training artificial neural networks with gradient-based learning methods and backpropagation. In such methods, during each iteration of training each of neural network’s weights receives an update proportional to the partial derivative of the error function with respect to the current weight.

## When Face Vanishing Gradient Problem

When you use large amount of hidden layer like Deep Neural Network

When you use Sigmoid or tanh activation function in hidden layer.

## Solution

Proper Weight Initialization

Using Non-Structuring Activation Functions

Batch Normalization

Gradient Clipping

## How to identify it

When loss is constant during model training.

# Hyper Parameter Tuning in Deep Learning

ann.add(Dense(4,input\_dim = 6, activation="relu")) use max no of hidden layer

ann.add(Dense(3, activation="relu")) no of nodes in parameter shapes

ann.add(Dense(2, activation="relu"))

ann.add(Dense(1, activation="sigmoid"))

batch\_size=100

epochs = 50

optimizer='adam'