Neural Networks-Part 1

Vidya Samadi, Ph.D. , M.ASCE

Clemson University

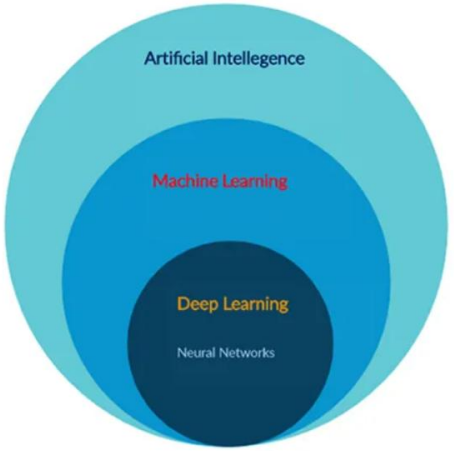
July 30, 2025

Relevant reading

- TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers
- Hands-on Machine Learning with Scikit-Learn and TensorFlow
- Machine Learning for Hackers
- Pattern Recognition and Machine Learning
- Natural Language Processing with Python
- Introduction to Machine Learning with Python
- TensorFlow with RNN

What is Neural Network?

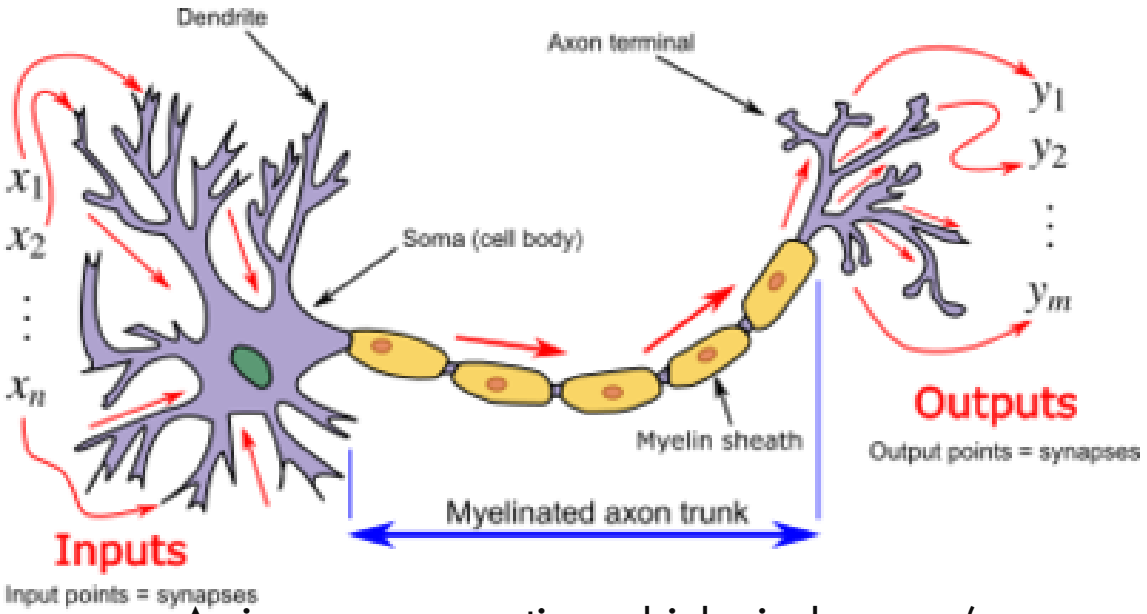
- Neural networks or simply NN form the base of deep learning, which is a subfield of machine learning, where the structure of the human brain inspires the algorithms.
- The way NN work is to take input data, train themselves to recognize patterns found in the data, and then predict the output for a new set of similar data.
- Therefore, a neural network can be thought of as the functional unit of deep learning, which mimics the behavior of the human brain to solve complex data-driven problems.



NN behaves as a biological neuron model!

- The neurons' dendrites refer to as input
- The nucleus process the data and forward the calculated output through the axon
- The width (thickness) of dendrites defines the weight associated with them.

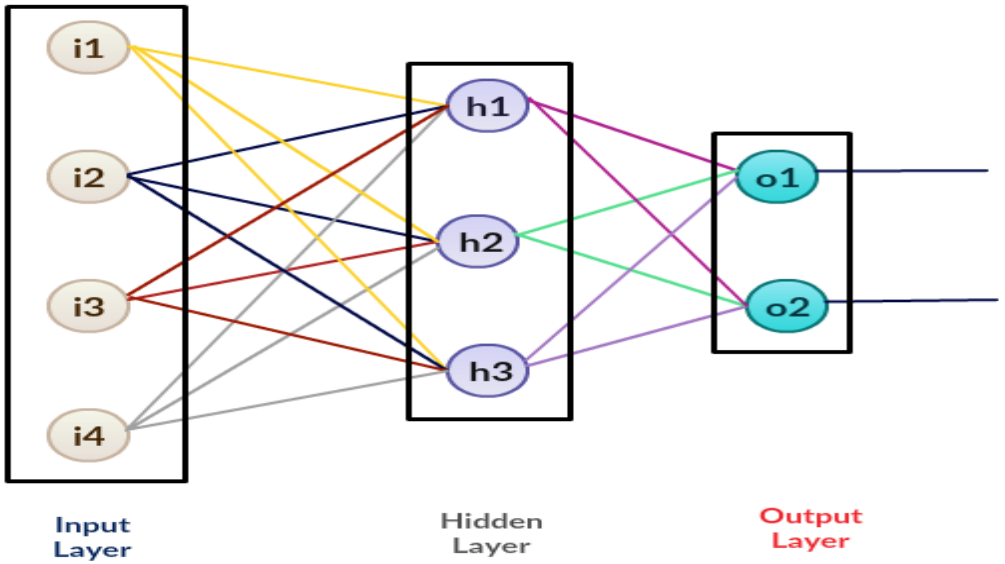
Neuron and myelinated axon, with signal flow from inputs at dendrites to outputs at axon terminals.



An image representing a biological neuron (source: Wikipedia).

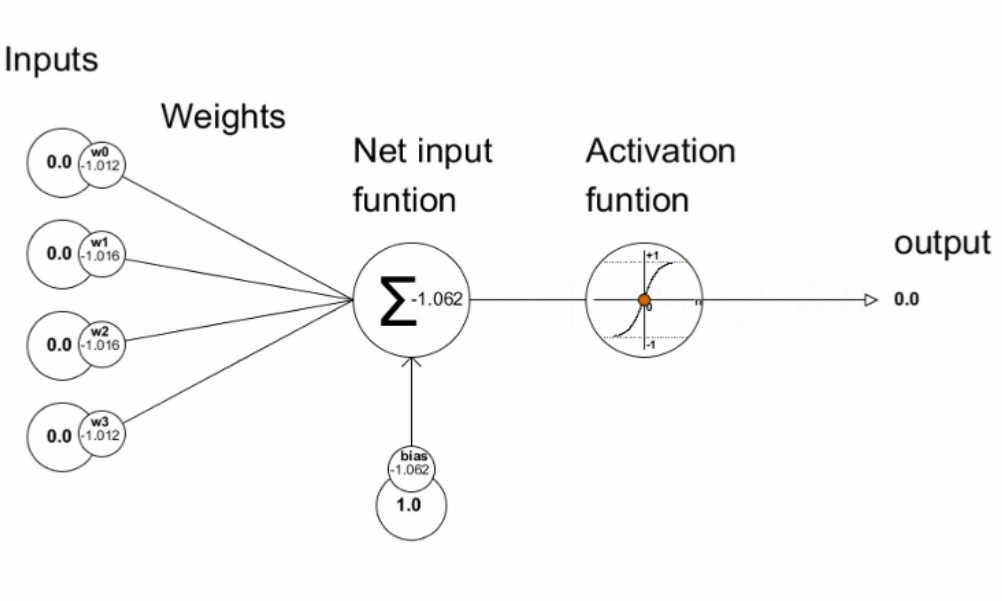
General Structure of NN

- An NN represents interconnected input and output units in which each connection has an associated weight.
- During the learning phase, the network learns by adjusting these weights in order to be able to predict the correct class for input data.



NN Implementation

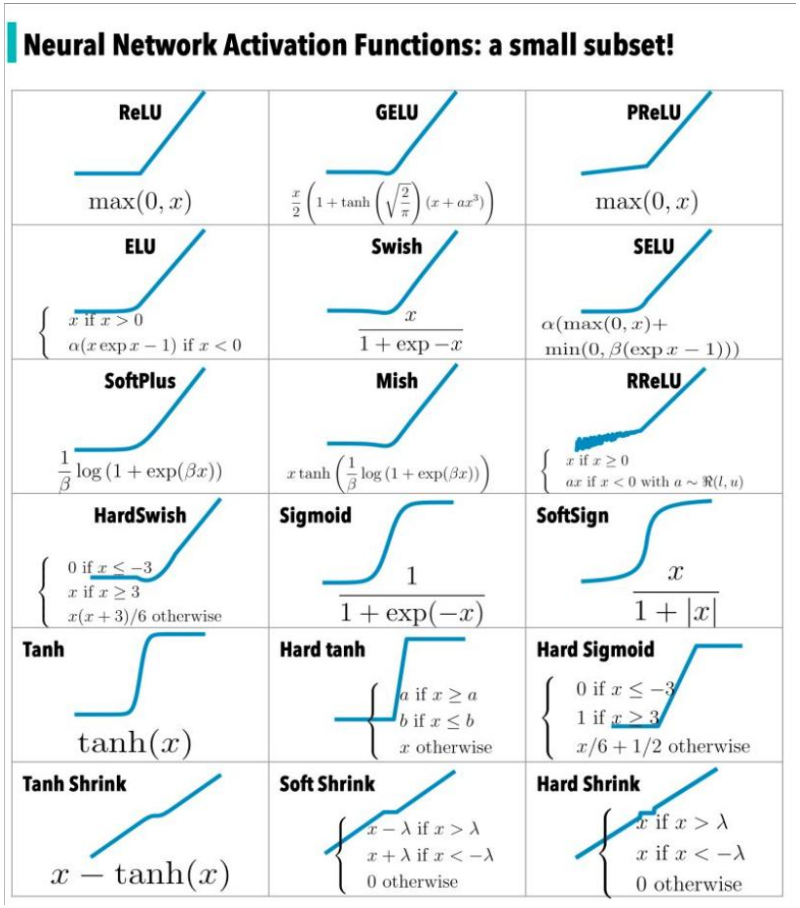
- Take inputs
- Assign random weights to input features
- Run the code for training.
- Find the error in prediction.
- Update the **weight** by gradient descent algorithm.
- Repeat the training phase with updated weights.
- Make predictions.



Types of Activation Functions

NN uses an activation function to simulates the neuron firing or not!

Name	Plot	Function, $f(x)$	Derivative of f, $f'(x)$	Range
Identity		x	1	$(-\infty, \infty)$
Binary step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$\begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$\{0, 1\}$
Logistic, sigmoid, or soft step		$\sigma(x) = \frac{1}{1+e^{-x}}$ [1]	$f(x)(1-f(x))$	$(0, 1)$
tanh		$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$1 - f(x)^2$	$(-1, 1)$
Rectified linear unit (ReLU)[11]		$\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ $= \max\{0, x\} = x\mathbf{1}_{x>0}$	$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$[0, \infty)$



The choice is made by considering the performance of the model or convergence of the loss function.

How to evaluate input and output?

Types of Loss Functions/Cost Functions

Most Common

- Regression Loss Functions
- Mean Squared Error Loss
- Mean Squared Logarithmic Error Loss
- Mean Absolute Error Loss

Binary Classification Loss Functions

- Binary Cross-Entropy
- Hinge Loss
- Squared Hinge Loss

Multi-Class Classification Loss Functions

- Multi-Class Cross-Entropy Loss
- Sparse Multiclass Cross-Entropy Loss
- Kullback Leibler Divergence Loss

Optimizer

- In neural networks, optimizers are crucial to fine-tune a model's parameters throughout the training process, aiming at minimizing a predefined loss function.
- Optimizers facilitate the learning process of neural networks (adjust weights and learning rates) by iteratively refining the weights and biases based on the feedback received from the data.
- Stochastic Gradient Descent (SGD), Adam, and Root Mean Square Propagation (RMSprop), each equipped with distinct update rules, learning rates, and momentum strategies, to enhance overall performance.
- The choice of optimizer depends on the specific application.

Important Neural Network Terms

Before proceeding, there are a few terms that you should be familiar with.

Epoch – The number of times the algorithm runs on the whole training dataset.

Sample – A single row of a dataset.

Batch – It denotes the number of samples to be taken to for updating the model parameters-- In most cases, an optimal batch-size is 64.

Learning rate – It is a parameter that provides the model a scale of how much model weights should be updated.

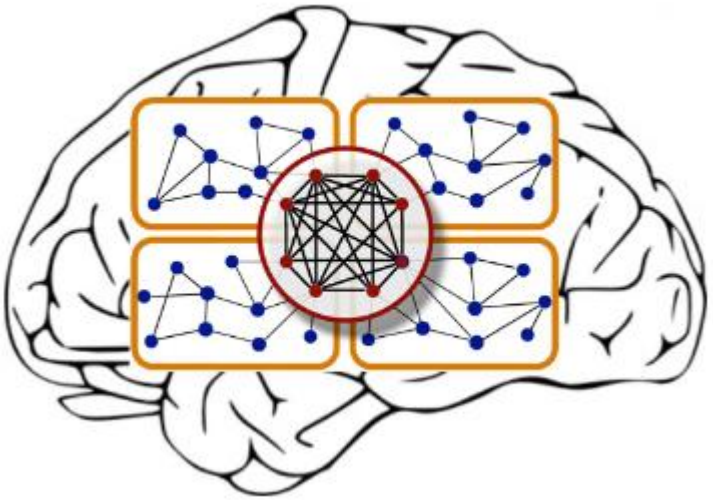
Cost Function/Loss Function – A cost function is used to calculate the cost, which is the difference between the predicted value and the actual value.

Weights/ Bias – The learnable parameters in a model that controls the signal between two neurons.

Topologies in Neural Networks

- Topologies briefly introduce how neurons are connected and how signals flow through the network
- How NN works and functions?
- There are many methods (algorithms) that make these functions/connection!

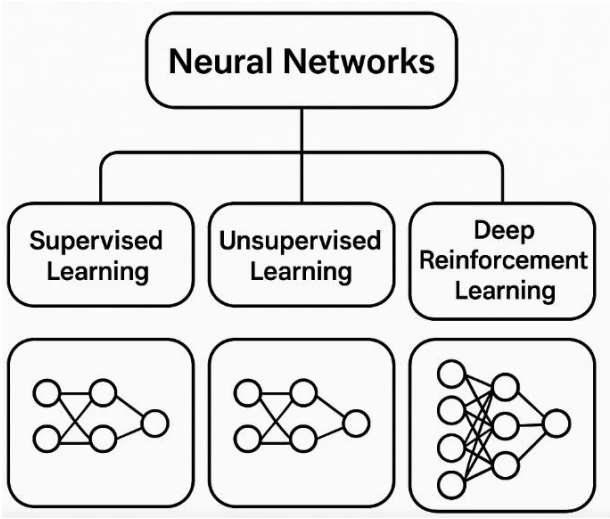
The way neurons are connected determines the structure of NN algorithm



We discuss each algorithm BREIFLY!

Major Types of Neural Networks

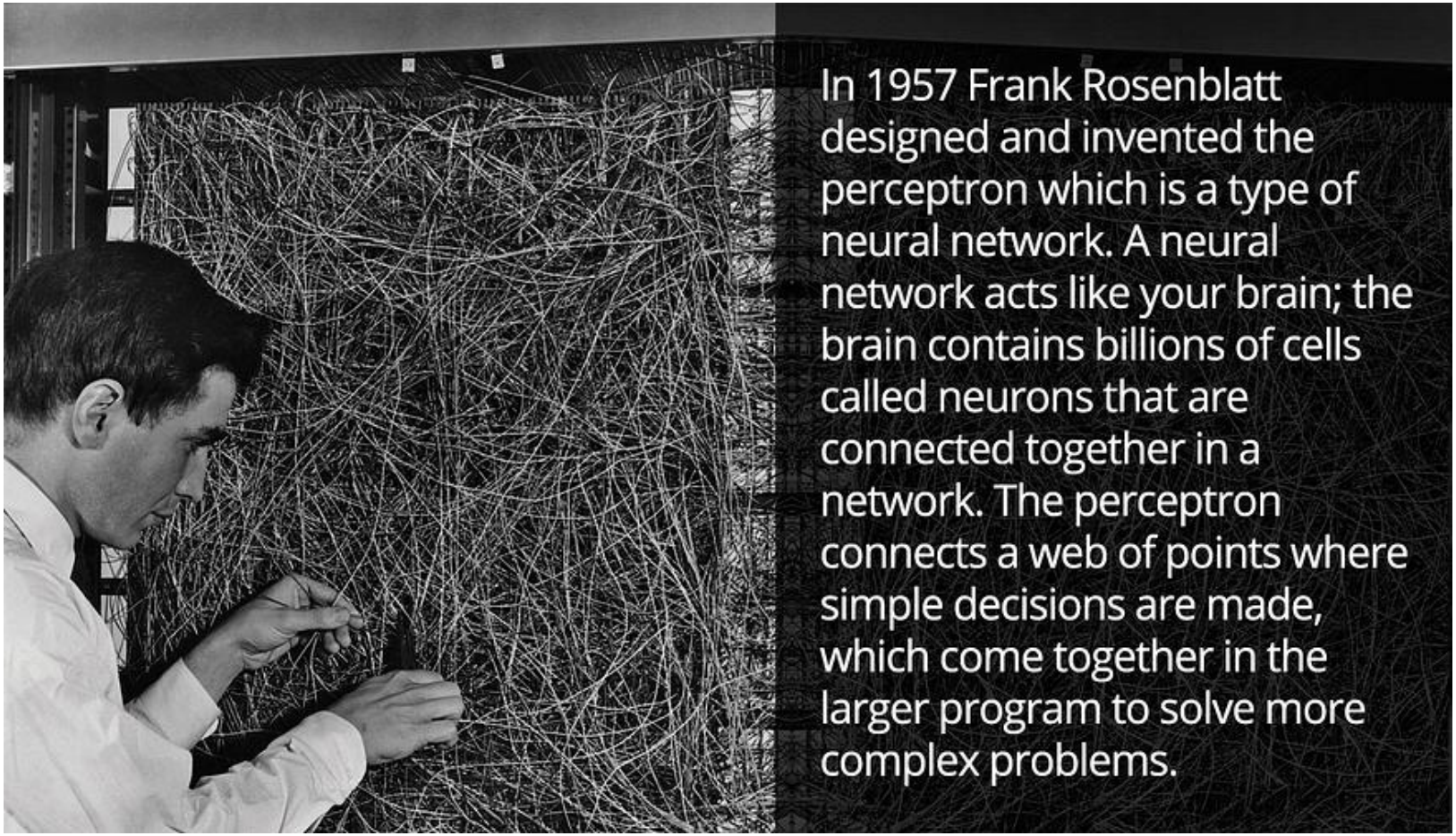
- Observe set of example: training data
- Infer the process that generated that data
- Use inference to make predictions about previously unseen data: test data
- Three major types



- **Supervised Learning:** given a set of feature/label pairs, find a rule that predicts the label associated with previously unseen input.
- **Unsupervised Learning:** given a set of feature vectors (without labels) group them into "clusters" (or create labels for groups).
- **Reinforcement Learning:** uses observations gathered from the interaction with its environment to take actions that would maximize the reward function. The reinforcement learning algorithm (called the agent) continuously learns from its environment using iteration. Example: computers reaching a super-human state and beating humans on computer games.

What is Perceptron?

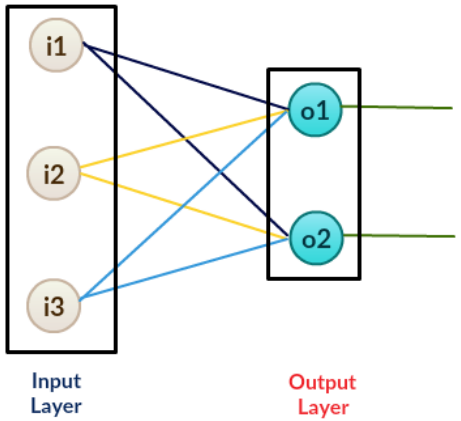
The Perceptron is a foundational algorithm in neural networks.



The perceptron by Frank Rosenblatt (source: Machine Learning Department at Carnegie Mellon University)

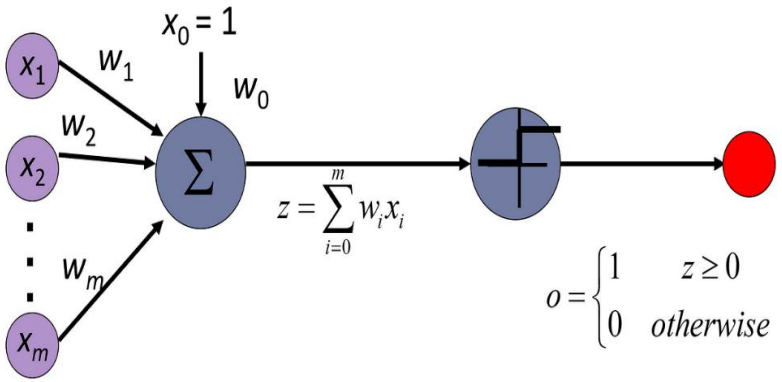
What is Perceptron?-cont.

- Simple model with no hidden layer.
- A perceptron only has an input layer and an output layer.
- It is so called a Linear Binary Classifier (classified output into 0 and 1)



How does it work?

The perceptron has m binary inputs denoted by $x_1, \ldots, x_m$, which represent the incoming signals from its neighboring neurons, and it outputs a single binary value denoted by o that indicates if the perceptron is "firing" or not.



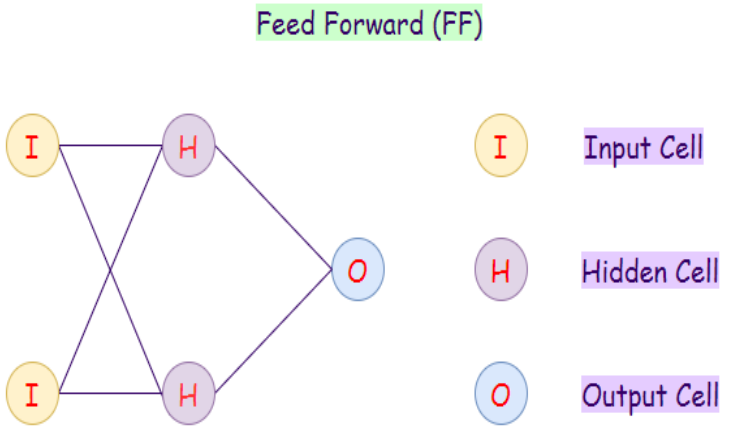
Applications: Classification, encode database (Multilayer Perceptron), & monitor access data.

Feed Forward (FF)

- In FF, all of the perceptron are arranged in layers where the input layer takes in input, and the output layer generates output.
- The nodes do not ever form a cycle.
- Every perceptron in one layer is connected with each node in the next layer. Therefore, all the nodes are fully connected.
- No visible or invisible connection between the nodes in the same layer and no **back-loops!**
- To minimize the error in prediction, the **backpropagation algorithm** will be used to update the weight values.

Applications: Data compression, pattern recognition, & computer vision.

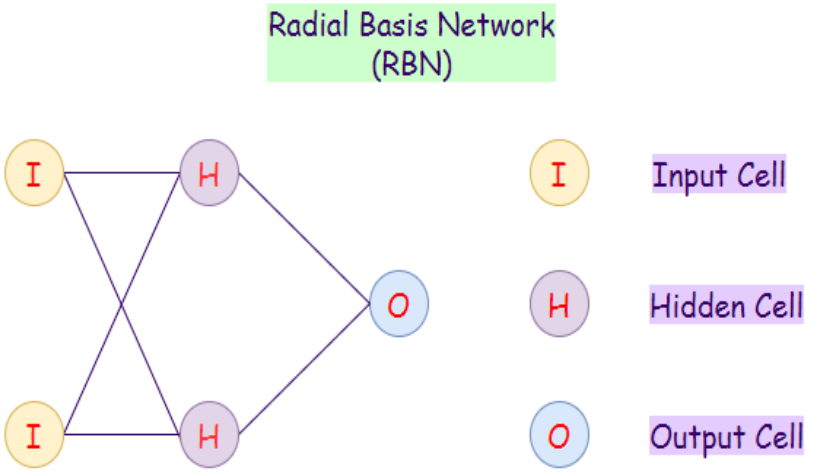
How do we use FF on pre-trained Model?



Radial Basis Network (RBN)

- Used for function approximation problems
- Simplicity in terms of architecture and training algorithms, and their ability to handle high-dimensional data.
- Uses radial basis functions as activation functions to give an output between 0 and 1, to find whether the answer is yes or no. This is perfect for continuous data– RBN ability to approximate any continuous function to arbitrary precision!

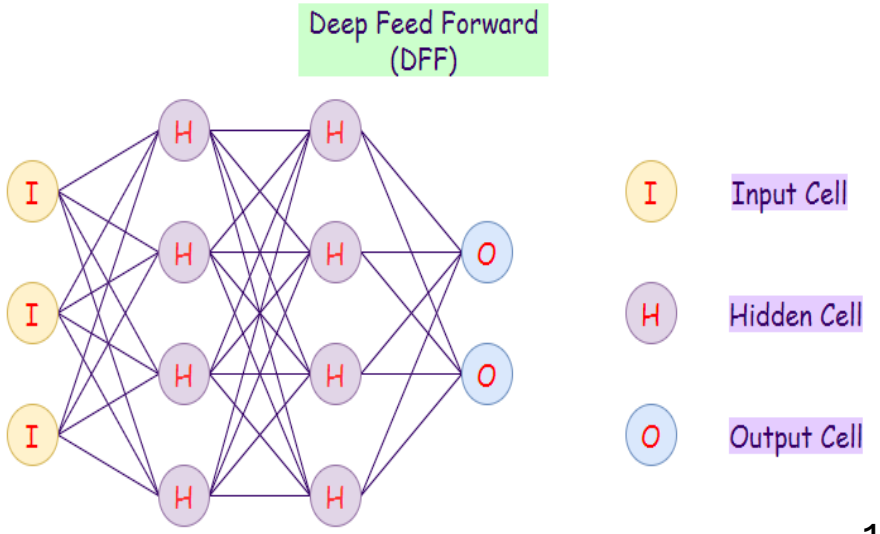
Applications: Function approximation, timeseries prediction, classification, system control.



Deep Feed Forward (DFF)

- A deep feed-forward network is a feed-forward network that uses more than one hidden layer.
- The main problem with using only one hidden layer is the one of overfitting, therefore by adding more hidden layers, we may achieve (not in all cases) reduced overfitting and improved prediction.

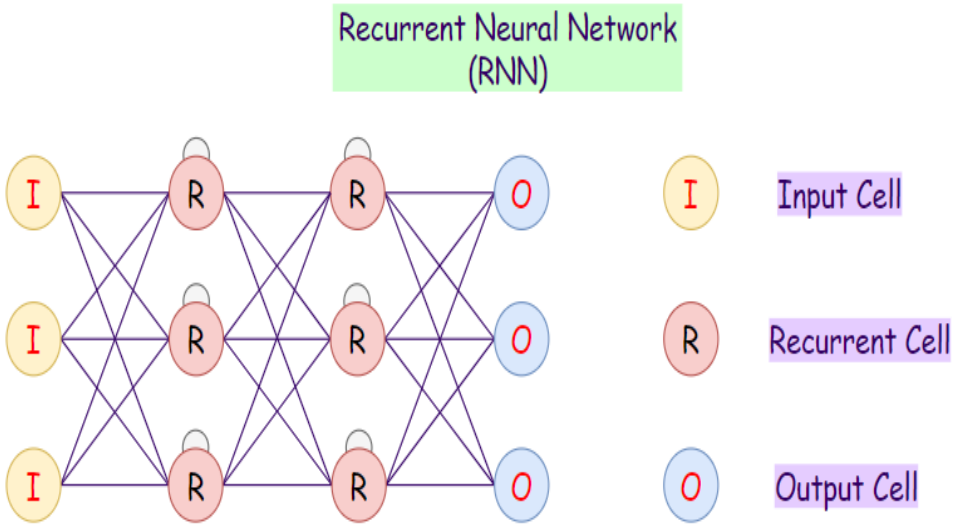
Applications: Data compression, pattern recognition, computer vision, data noise filtering, & time series prediction.



Recurrent Neural Network (RNN)

- RNNs are a variation to feed-forward networks
- RNNs can process inputs and share any lengths and weights across time.
- Use previous information/data in current iterations
- Drawback: slow computational speed, memory loss (vanishing problem)!

Applications: Robot control, time series prediction, & time series anomaly detection.



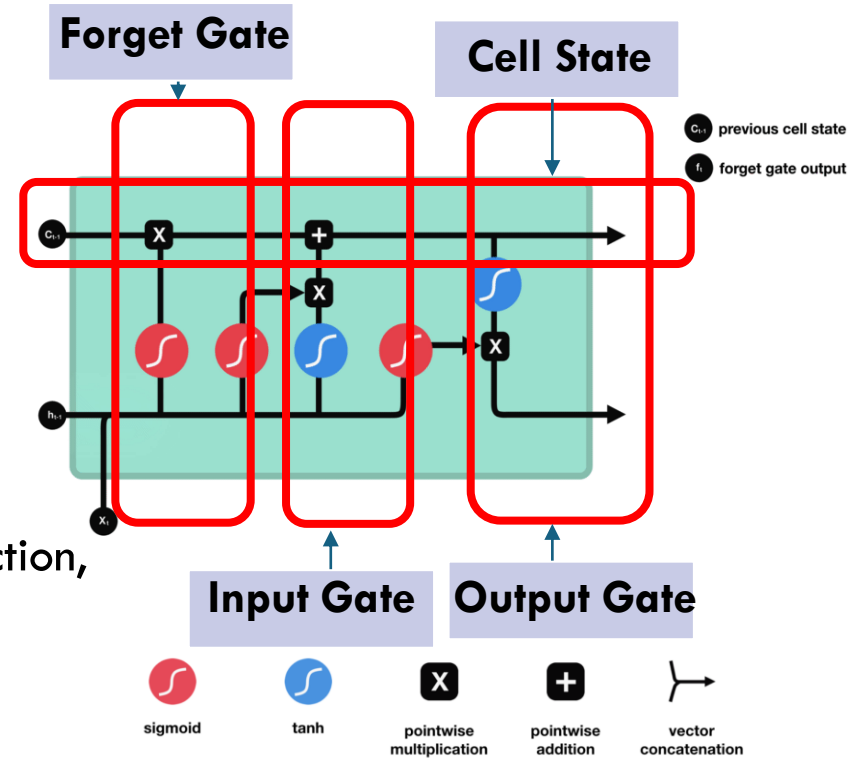
Long Short-Term Memory (LSTM)

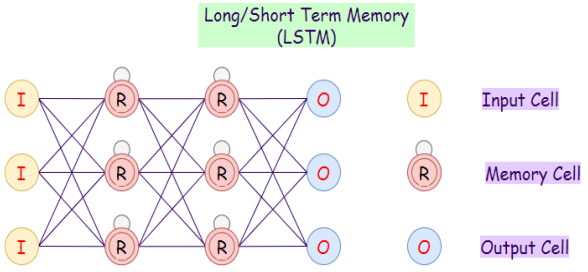
- LSTM networks introduce a memory cell.
- They can process data with memory gaps.
- Suitable for long term simulation that a (long) memory is needed.
- LSTM remembers data from a long time ago, in contrast to simple RNN. **Takes care of vanishing problem.**

Applications: Robot control, time series prediction, & time series anomaly detection.

An input gate, an output gate and a memory (forget) gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

Developed by Jürgen Schmidhuber in 1997
https://www.bioinf.jku.at/publications/older/2604.pdf



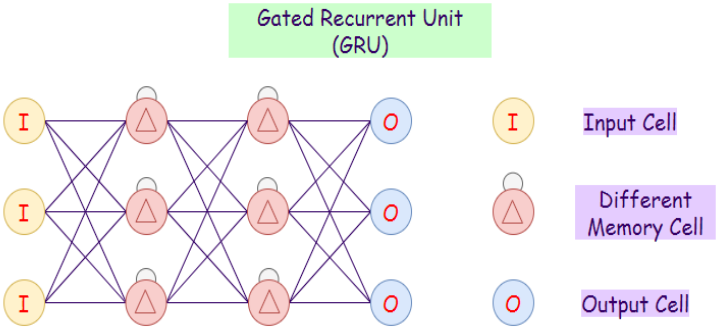


Gated Recurrent Unit (GRU)

- GRUs are a variation of LSTMs because they both have similar designs and mostly produce equally good results.
- GRU has fewer gates and fewer parameters than LSTM, which makes it simpler and faster.
- GRU has a single hidden state which allows it to store and output different information (LSTM has a separate cell state and output), which may limit its capacity.
- GRUs only have three gates, with no Internal Cell State.

Applications: Robot control, time series prediction, & time series anomaly detection.

Update Gate: Determines how much past knowledge to pass to the future.
Reset Gate: Determines how much past knowledge to forget.
Current Memory Gate: Subpart of reset fate.



Developed by Cho et al., 2014
https://arxiv.org/abs/1412.3555

Installing Deep Learning Libraries

Install the necessary libraries. The most common libraries used in deep learning are TensorFlow, Keras, and PyTorch.

```
import numpy as np
import pandas as pd
import datetime as dt
import matplotlib.pyplot as plt
import math
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import SimpleRNN
from keras.layers import Dropout
from keras.layers import GRU, Bidirectional
from keras.optimizers import SGD
from sklearn import metrics
from sklearn.metrics import mean_squared_error
from tensorflow.keras.optimizers import Adam
import tensorflow.keras.backend as K
import tensorflow as tf
from keras.callbacks import ModelCheckpoint
import h5py
```

RNN Structure

```
regressor = Sequential()
regressor.add(SimpleRNN(units = 128,
                        activation = "relu",
                        return_sequences = True,
                        input_shape = (X_train.shape[1],1)))
regressor.add(Dropout(0.2))

regressor.add(SimpleRNN(units = 64,
                        activation = "relu",
                        return_sequences = True))

regressor.add(SimpleRNN(units = 32,
                        activation = "relu",
                        return_sequences = True))

regressor.add(SimpleRNN(units = 4))

regressor.add(Dense(units = 1,activation='sigmoid'))
```

Creating a Sequential model in Keras

Add layer with 128 internal units

the layer will return the full sequence of hidden states (outputs) for each timestep in the input sequence.

The number of neurons in this layer

Adding the output layer

RNN Structure-cont.

```
regressor.compile(optimizer = 'adam',
                  loss = 'mean_squared_error',
                  metrics = ["accuracy"])

regressor.fit(X_train, y_train, epochs = 60, batch_size = 32)
regressor.summary()
```

compiling RNN

Fitting the model-please note it takes a long time to run

Summary

- NN models process information in ways similar to the human brain, they can be applied to many tasks people do.
- They require very large amounts of data to train, so it's not treated as a general-purpose algorithm.
- NNs can be adapted to various tasks, including image classification, regression, anomaly detection, computer vision, classification, etc.
- Any of these NN algorithms can be used as standalone or benchmark models.
- Compared to a single NN algorithm, hybrid neural networks have a promising perspective on the development of modeling intelligence for environmental application.

THANK YOU

The support from the National Science Foundation (award # 2320979) is acknowledged.