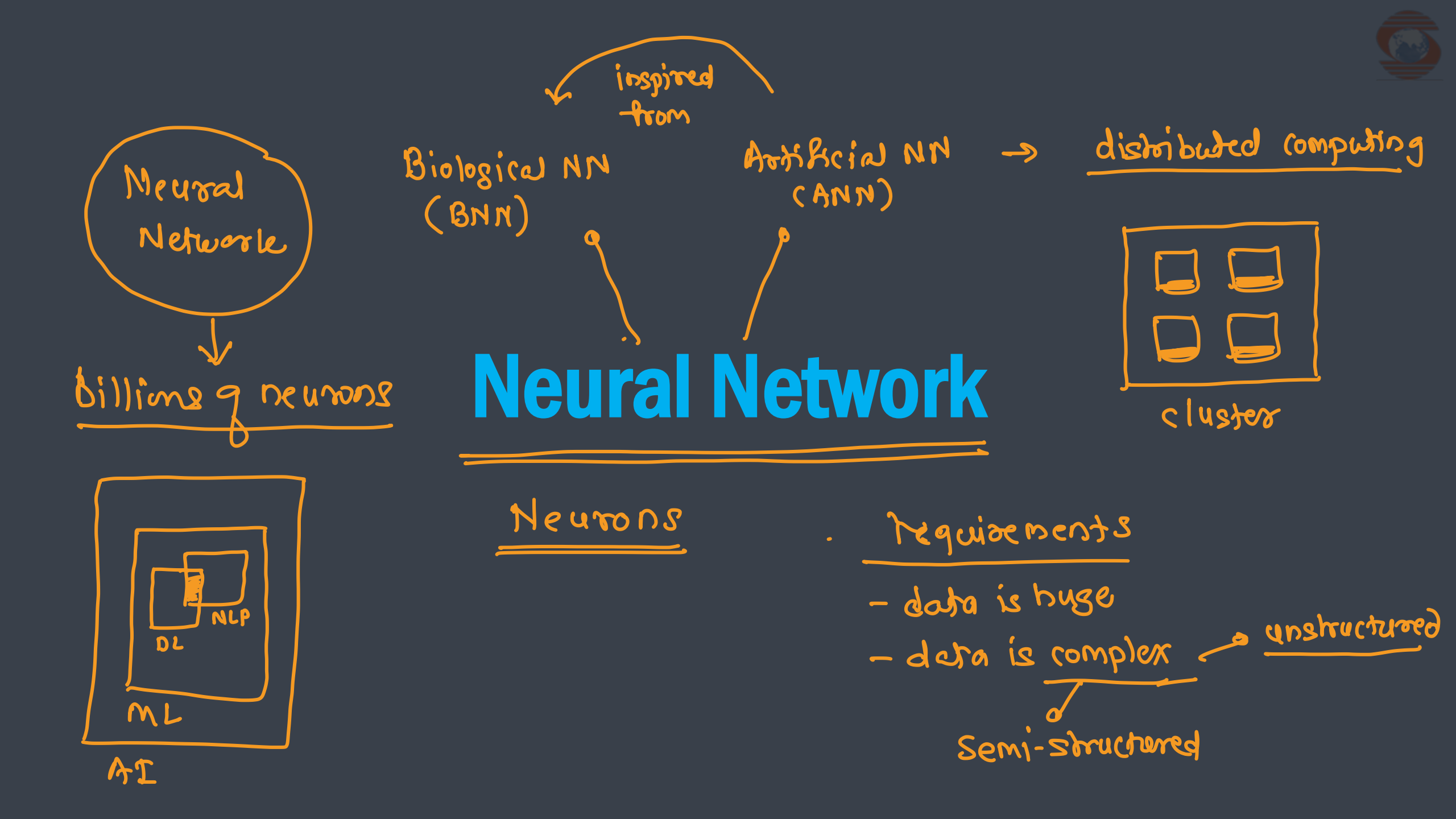

Neural Network
inspired from
Biological NN (BNN)
Artificial NN (ANN)
distributed computing
Neural Networks
billions of neurons
cluster
AI
ML
DL
NLP
Neurons
Requirements
- data is huge
- data is complex
unstructured
Semi-structured

Introduction

- A neural network is a method in artificial intelligence (AI) that teaches computers to process data in a way that is inspired by the human brain → BNN
- It is a type of machine learning (ML) process, called deep learning, that uses interconnected nodes or neurons in a layered structure that resembles the human brain
- It creates an adaptive system that computers use to learn from their mistakes and improve continuously
- Thus, artificial neural networks attempt to solve complicated problems, like summarizing documents or recognizing faces, with greater accuracy
- Neural networks can help computers make intelligent decisions with limited human assistance
- This is because they can learn and model the relationships between input and output data that are nonlinear and complex
  - → unstructured

Where are they used ?

- Medical diagnosis by medical image classification → CNN
- Targeted marketing by social network filtering and behavioral data analysis → RNN | LSTM | transfo
- Financial predictions by processing historical data of financial instruments → ANN
- Electrical load and energy demand forecasting
- Process and quality control
- Chemical compound identification
- Computer Vision → CNN
  - Visual recognition in self-driving cars so they can recognize road signs and other road users
  - Content moderation to automatically remove unsafe or inappropriate content from image and video archives
  - Facial recognition to identify faces and recognize attributes like open eyes, glasses, and facial hair
  - Image labeling to identify brand logos, clothing, safety gear, and other image details
- Speech recognition
  - Assist call center agents and automatically classify calls
  - Convert clinical conversations into documentation in real time
  - Accurately subtitle videos and meeting recordings for wider content reach

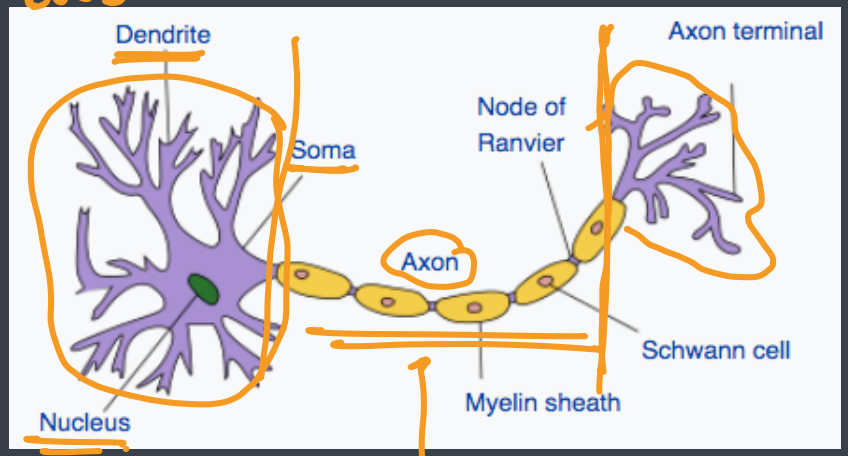
complex < huge unstructured

Where are they used ?

- **Natural language processing**
  - Automated virtual agents and chatbots
  - Automatic organization and classification of written data → tagging
  - Business intelligence analysis of long-form documents like emails and forms → summarization
  - Indexing of key phrases that indicate sentiment, like positive and negative comments on social media → sentiment analysis
  - Document summarization and article generation for a given topic → generative AI
- **Recommendation engines**
  - Neural networks can track user activity to develop personalized recommendations
  - They can also analyze all user behavior and discover new products or services that interest a specific user.

Structure of Neural Network

- The human brain is the inspiration behind neural network architecture
- Human brain cells, called neurons, form a complex, highly interconnected network and send electrical signals to each other to help humans process information
- Similarly, an artificial neural network is made of artificial neurons that work together to solve a problem
- Artificial neurons are software modules, called nodes, and artificial neural networks are software programs or algorithms that, at their core, use computing systems to solve mathematical calculations



body

Neuron

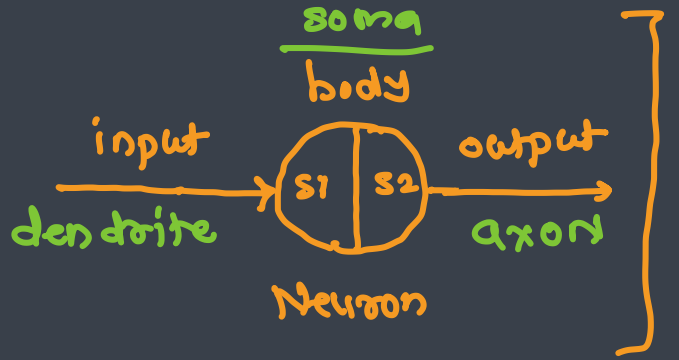
connection

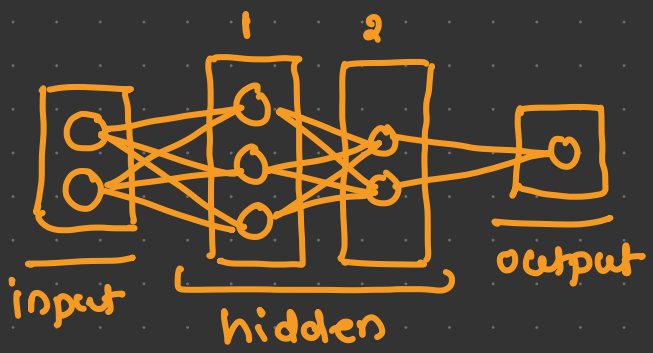
body → logic

connection → input, output

↳ body of neuron
- ↳ step I : summation function
- ↳ step II : activation function

perceptron

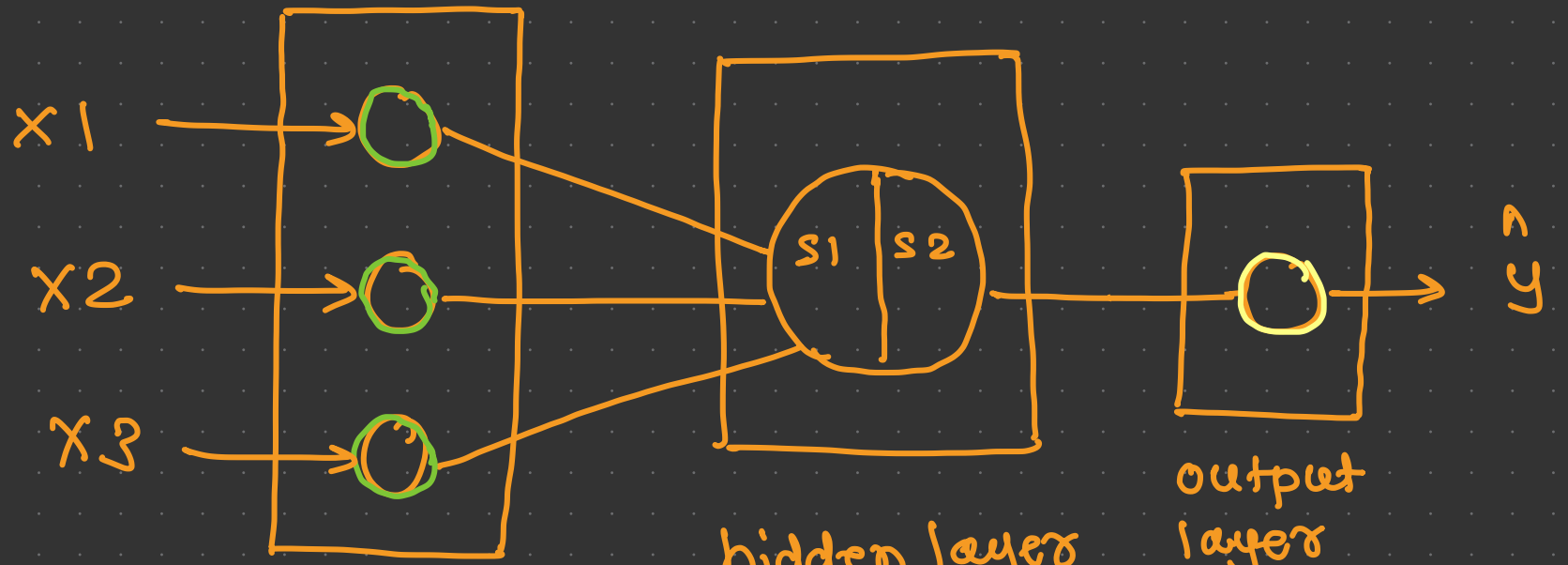




deep neural network

x1	x2	x3	y

simple neural network



logic → calculation

Architecture of Simple Neural Network

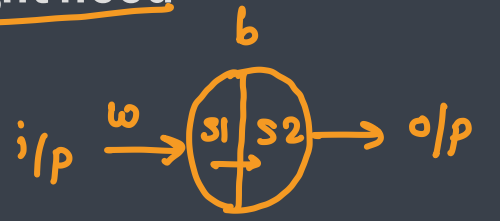
- A basic neural network has interconnected artificial neurons in three layers
- **Input Layer** — #neurons present = # independent variables → mandatory, only one layer
  - Information from the outside world enters the artificial neural network from the input layer
  - Input nodes process the data, analyze or categorize it, and pass it on to the next layer
- **Hidden Layer(s)** → optional → can be any no depending upon data, resources, & complexity
  - Hidden layers take their input from the input layer or other hidden layers
  - Artificial neural networks can have a large number of hidden layers
  - Each hidden layer analyzes the output from the previous layer, processes it further, and passes it on to the next layer
- **Output Layer** — mandatory → only one is required
  - The output layer gives the final result of all the data processing by the artificial neural network
  - It can have single or multiple nodes
  - For instance, if we have a binary (yes/no) classification problem, the output layer will have one output node, which will give the result as 1 or 0. However, if we have a ~~multi-class~~ multi-label classification problem, the output layer might consist of more than one output node.

Architecture of Deep Neural Network

- Deep neural networks, or deep learning networks, have several hidden layers with millions of artificial neurons linked together
- A number, called weight, represents the connections between one node and another
- The weight is a positive number if one node excites another, or negative if one node suppresses the other
- Nodes with higher weight values have more influence on the other nodes
- Theoretically, deep neural networks can map any input type to any output type
- However, they also need much more training as compared to other machine learning methods
- They need millions of examples of training data rather than perhaps the hundreds or thousands that a simpler network might need

weight = coefficient (β estimator)

$y = 2x_1 + x_2$



b = bias

$w_1 .. w_n$ ⇒ weight } neuron

$x_1 .. x_n$ ⇒ independent variables

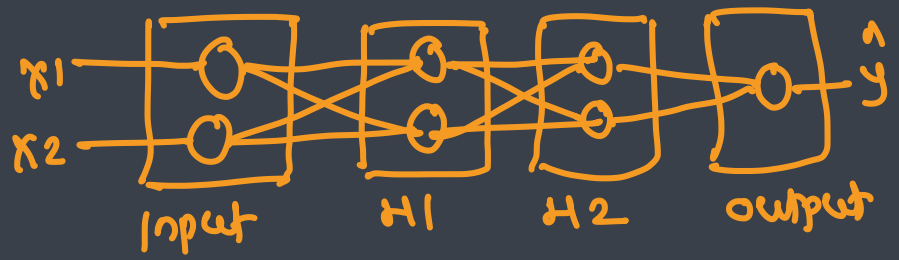
w = +ve or -ve

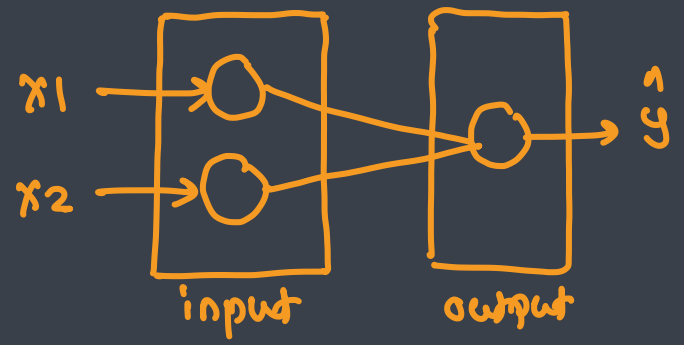
$y = 2x_1 - x_2$

S1 → summation function : $z = w_1x_1 + w_2x_2 + .. + w_nx_n + b = \sum w_ix_i + b$

Perceptron

- A single-layer feedforward neural network was introduced in the late 1950s by Frank Rosenblatt
- It was the starting phase of Deep Learning and Artificial neural networks
- Perceptron is one of the simplest Artificial neural network architectures
- It consists of a single layer of input nodes that are fully connected to a layer of output nodes
- Types of Perceptron
  - Single-Layer Perceptron
    - This type of perceptron is limited to learning linearly separable patterns
    - They are effective for tasks where the data can be divided into distinct categories through a straight line
  - Multilayer Perceptron
    - Multilayer perceptrons possess enhanced processing capabilities as they consist of two or more layers, adept at handling more complex patterns and relationships within the data





Components of Perceptrons

- **Input Features:** The perceptron takes multiple input features, each input feature represents a characteristic or attribute of the input data → input layer
- **Weights:** Each input feature is associated with a weight, determining the significance of each input feature in influencing the perceptron's output. During training, these weights are adjusted to learn the optimal values. weight : initially selected random value
- **Summation Function:** The perceptron calculates the weighted sum of its inputs using the summation function. The summation function combines the inputs with their respective weights to produce a weighted sum. $(z) = \sum w_i x_i + b$
- **Activation Function:** The weighted sum is then passed through an activation function. Perceptron uses Heaviside step function functions. which take the summed values as input and compare with the threshold and provide the output as 0 or 1.
- **Output:** The final output of the perceptron, is determined by the activation function's result. For example, in binary classification problems, the output might represent a predicted class (0 or 1).

Components of Perceptrons

- **Bias:** A bias term is often included in the perceptron model. The bias allows the model to make adjustments that are independent of the input. It is an additional parameter that is learned during training.

- **Learning Algorithm (Weight Update Rule):** During training, the perceptron learns by adjusting its weights and bias based on a learning algorithm. A common approach is the perceptron learning algorithm, which updates weights based on the difference between the predicted output and the true output.

↳ speed of learning → hyperparameter

weights & bias = randomly selected as low as possible & later will get adjusted as per learning from the iterations

Forward Propagation → calculation of $\hat{y}$

- **Input Layer:** Each feature in the input layer is represented by a node on the network, which receives input data
- **Weights and Connections:** The weight of each neuronal connection indicates how strong the connection is. Throughout training, these weights are changed based on learning
- **Hidden Layers:** Each hidden layer neuron processes inputs by multiplying them by weights, adding them up, and then passing them through an activation function. By doing this, non-linearity is introduced, enabling the network to recognize intricate patterns.
- **Output:** The final result is produced by repeating the process until the output layer is reached

$\hat{y}$

input layer → hidden layer(s) → output layer

forward propagation

$w_1\ w_2\ b_1$

$\hat{y} = (\sum w_i x_i + b_1) \Rightarrow$

$(\sum w_i x_i + b)$

$w_3\ b_2$

$y = \beta_1 x_1 + \beta_2 x_2 + \beta_0$

Activation Functions

Activation Function

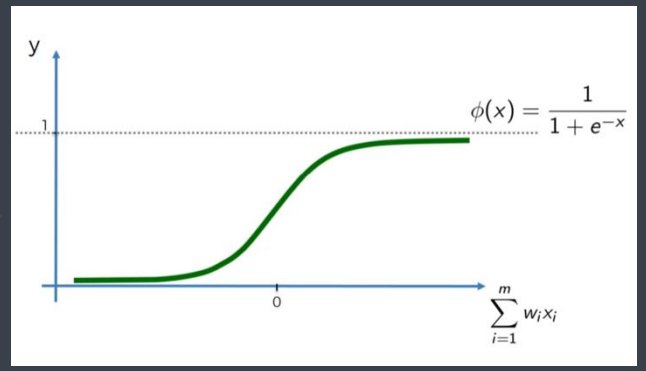
- An activation function in an artificial neural network (ANN) is a mathematical function applied to the output of each neuron (or node) to introduce non-linearity into the model
- This non-linearity allows the network to learn complex patterns and relationships in the data, enabling it to perform tasks such as classification and regression
- Merits
  - Introduce Non-Linearity
    - Without activation functions, a neural network would essentially behave like a linear model, regardless of how many layers it has. Activation functions allow the model to learn non-linear mappings
  - Control Output
    - They determine the output of neurons, which can be used in further computations or as final predictions
  - Enable Learning
    - Activation functions help in the optimization process by providing gradients during backpropagation, which are essential for adjusting weights

$y = ax = \max(0, x)$

Sigmoid Function

- The sigmoid activation function is a mathematical function that maps any real-valued number into a value between 0 and 1. It's commonly used in binary classification tasks.

- $\text{sigmoid}(z) = \frac{1}{1+e^{z}}$



z = value generated by summation function

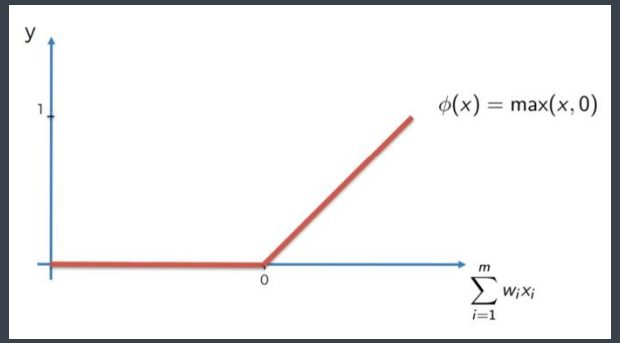
= $\sum w_i x_i + b$

- Properties
  - Output Range: The output of the sigmoid function is always between 0 and 1, which makes it particularly useful for models that predict probabilities.
  - S-shaped Curve: The graph of the sigmoid function has an "S" shape, which makes it smooth and differentiable
  - Non-linear: The sigmoid function introduces non-linearity, allowing the neural network to learn complex patterns
- Use Cases
  - Binary Classification: Often used in the output layer of binary classifiers.
  - Logistic Regression: Commonly applied in logistic regression models.

Relu Function

- The ReLU (Rectified Linear Unit) activation function is one of the most widely used activation function. It introduces non-linearity into the model while maintaining computational efficiency.

- ReLU (x) = max(0, x)



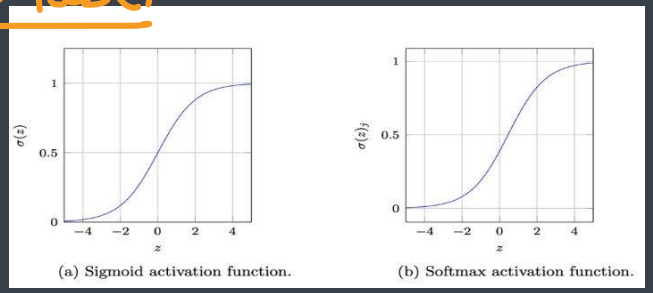
- Properties
  - Non-linearity: ReLU allows the model to learn complex patterns by introducing non-linearity
  - Output Range: The output ranges from 0 to infinity. It outputs zero for any negative input, which can help in preventing the vanishing gradient problem.
  - Sparsity: Because it outputs zero for half of the input space, ReLU can lead to sparse representations, which can improve model efficiency.
  - Computational Efficiency: ReLU is computationally efficient, as it requires only a simple thresholding at zero
- Use Cases
  - ReLU is commonly used in hidden layers of deep neural networks due to its advantages in training and efficiency, making it a standard choice in modern architectures.

Softmax Function

- The Softmax function takes a vector of real numbers and converts it into a probability distribution
- Each value is transformed into a number between 0 and 1, and the sum of all output values equals 1
- Particularly it is used in multi-class classification

multi-label

- $softmax(z) = \left(\frac{e^z}{\sum e^z}\right)$



(a) Sigmoid activation function. (b) Softmax activation function.

(2)

0	-0.01
1	-0.02
2	-0.80
3	-0.03
4	⋮
9	-0.04
	1.0

- Properties
  - Output Range: The output values are in the range (0, 1), which makes them interpretable as probabilities.
  - Sum to One: The sum of all output probabilities is 1, satisfying the property of a probability distribution.
  - Sensitivity to Differences: Softmax emphasizes the largest logits, making it more likely for the model to predict the class with the highest score
- Use Cases

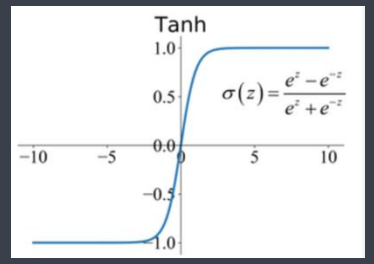
label

labels

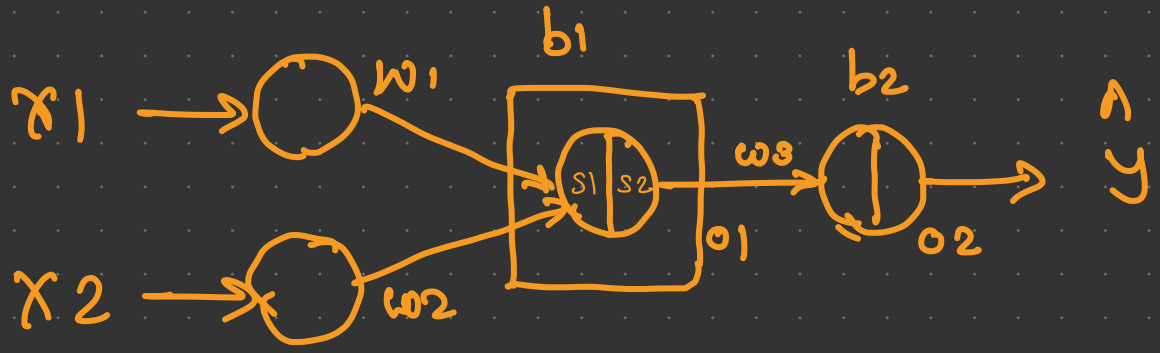
  - Multi-Class Classification: It is commonly used in the output layer of neural networks when there are multiple classes
  - Reinforcement Learning: Often used in policy networks to select actions based on their probabilities

Tanh Function

- The tanh (hyperbolic tangent) activation function is another widely used activation function. It maps input values to a range between -1 and 1, making it useful for models that need to output zero-centered values.
- $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$



- Properties
  - Output Range: The output values range from -1 to 1, which means it is zero-centered. This helps in faster convergence during training, as the outputs are balanced around zero.
  - S-shaped Curve: The tanh function has an "S" shaped curve similar to the sigmoid function, but it is steeper, which can lead to better gradients for training.
  - Non-linearity: Like other activation functions, tanh introduces non-linearity, enabling the network to learn complex patterns.
- Use Cases
  - Hidden Layers: Tanh is often used in the hidden layers of neural networks, especially in recurrent neural networks (RNNs) and other architectures where zero-centered outputs are beneficial.
  - Situations Needing Non-linear Transformations: It's suitable for problems requiring a non-linear transformation with outputs centered around zero.



x1	x2	y
10	20	4
20	30	5

$\eta = 0.2$ — learning rate] hyperparam

Iteration I. $w_1 = 0.01$, $w_2 = 0.02$, $b_1 = 0.2$

step I → summation function = $z = \sum w_i x_i + b_1$

$z = 0.01 \times 10 + 0.02 \times 20 + 0.2$

$z = 0.7$

step II → activation function = $\sigma(z) = \max(0, 0.7) = 0.7$

Relu

$\boxed{o_1 = 0.7}$ → output generated by hidden layer

Iteration I

output layer

$w_3 = 0.03$, $b_2 = 0.3$

Step I = summation function = $z = \sum w_i x_i + b$

$z = 0.03 \times 0.7 + 0.3$

$\boxed{z = 0.321}$

Step II = activation function = ReLu $= \max(0, z) =$

$= \max(0, 0.321)$

$\boxed{o_2 = 0.321}$ → generated at output neuron

$\boxed{\hat{y} = o_2 = 0.321}$

$\hat{y} = 0.321$, $y = 4$, loss function : $(y - \hat{y})$, MAE / MSE / RMSE

hyperparam

optimizer → updates/optimizes weights

SGD

Error $= y - \hat{y} = 4 - 0.321 = 3.6$

$$w_{new} = w_{old} - \underbrace{\Delta L(w_{old})}_{\text{gradient of old weight}} = \left[w_{old} - \frac{dL}{dw_{old}} \right]$$

input layer → hidden layer → output layer

forward propagation → calculate $\hat{y}$

input layer ← hidden layer ← output layer

backpropagation → learn and adjust weights

epoch = forward propagation + backpropagation

↳ hyper parameter

Backpropagation

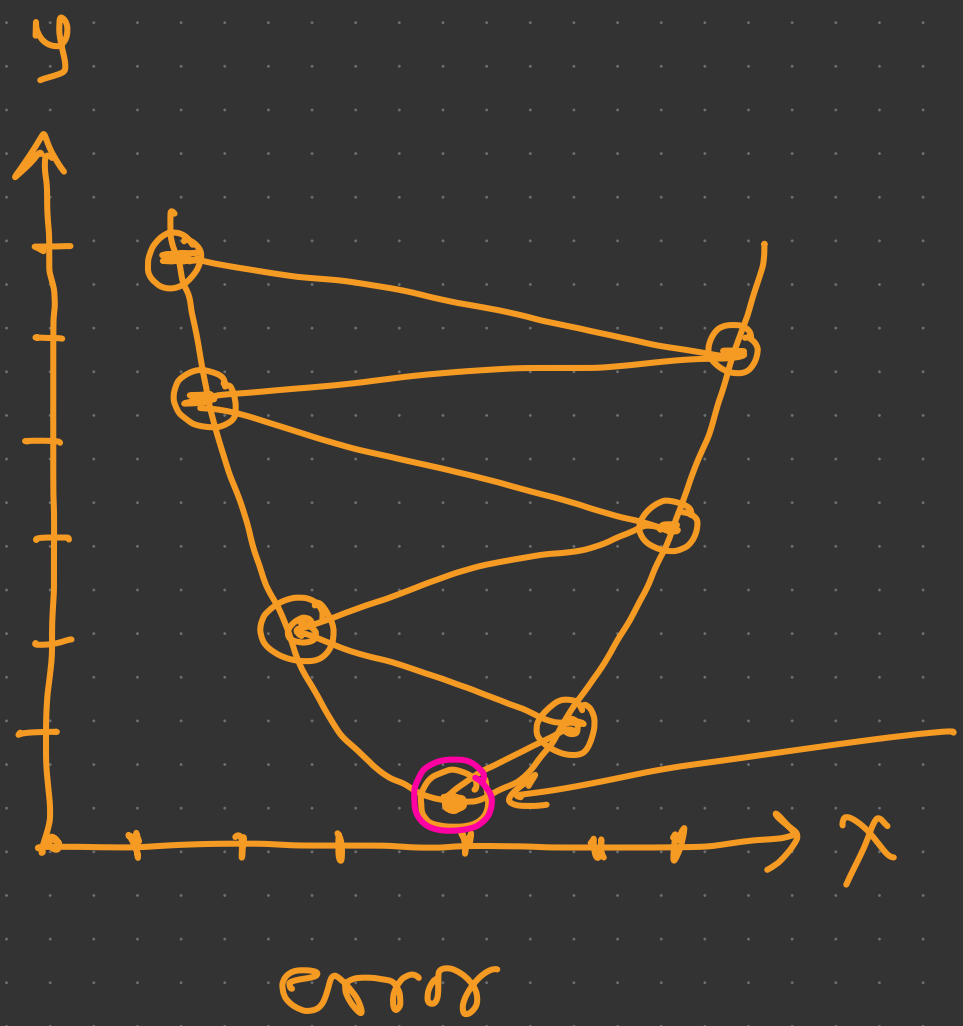
- **Loss Calculation:** The network's output is evaluated against the real goal values, and a loss function is used to compute the difference. For a regression problem, the Mean Squared Error (MSE) is commonly used as the cost function.
- **Loss Function:** MSE, MAE etc.
- **Gradient Descent:** Gradient descent is then used by the network to reduce the loss. To lower the inaccuracy, weights are changed based on the derivative of the loss with respect to each weight.
- **Adjusting weights:** The weights are adjusted at each connection by applying this iterative process, or backpropagation, backward across the network.
- **Training:** During training with different data samples, the entire process of forward propagation, loss calculation, and backpropagation is done iteratively, enabling the network to adapt and learn patterns from the data.
- **Actvation Functions:** Model non-linearity is introduced by activation functions like the rectified linear unit (ReLU) or sigmoid. Their decision on whether to "fire" a neuron is based on the whole weighted input.

Optimizers

Gradient Descent

- Gradient descent is a widely used optimization algorithm in machine learning and deep learning for minimizing the loss function of a model. The primary goal of gradient descent is to find the optimal parameters (weights and biases) that minimize the difference between the predicted outputs and the actual outputs. → minimize error
- How Gradient Descent Works → weights
  - Initialization: Start with random values for the model parameters
  - Compute the Loss: Calculate the loss (or cost) using a loss function, which measures how well the model's predictions match the actual data.
  - Calculate the Gradient: Compute the gradient of the loss function with respect to each parameter. The gradient is a vector that points in the direction of the steepest increase of the loss function.
  - Update Parameters: Adjust the parameters in the opposite direction of the gradient to reduce the loss
    - $\theta = \theta - \alpha \Delta L(\theta)$ α = learning Rate = hyperparameter
  - Iterate: Repeat the process (compute loss, calculate gradient, update parameters) until the loss converges to a minimum or until a predefined number of iterations is reached → epoch

gradient



$$w_{new} = w_{old} - \frac{\delta L}{\delta w_{old}}$$

Global minima

globally (all epochs)

minimum error

Variants of Gradient Descent

- **Batch Gradient Descent**
  - Uses the entire training dataset to compute the gradient at each step
  - This can be computationally expensive for large datasets
- **Stochastic Gradient Descent (SGD)**
  - Updates the parameters using only one training example at a time
  - This makes it faster and allows for online learning but introduces more noise in the updates
- **Mini-Batch Gradient Descent**
  - Combines the benefits of both batch and stochastic gradient descent by using a small random subset (mini-batch) of the training data to compute the gradient
  - This approach balances speed and accuracy

size of mini-batch is a hyperparameter

optimizer = "sgd", mini_batch = [20]

Advanced Optimizers

- Momentum
  - Helps accelerate gradients vectors in the right directions, thus leading to faster converging
- Nesterov Accelerated Gradient (NAG)
  - A variant of momentum that looks ahead at the future position of the parameters, providing a more accurate update
- Adagrad
  - Adapts the learning rate for each parameter based on the historical gradients, allowing for more frequent updates for infrequent features
- RMSprop
  - Similar to Adagrad, but it uses a moving average of squared gradients to normalize the gradients
- Adam
  - Combines the advantages of both momentum and RMSprop
  - It computes adaptive learning rates for each parameter and maintains a moving average of the gradients

Adam Optimizer

- The Adam optimizer (short for Adaptive Moment Estimation) is a popular optimization algorithm used in training machine learning models, particularly deep learning models
- It combines the benefits of two other extensions of stochastic gradient descent: AdaGrad and RMSprop
- Key Features of Adam
  - Adaptive Learning Rates
    - Adam computes adaptive learning rates for each parameter from estimates of first and second moments of the gradients
    - This allows it to adjust the learning rate based on how frequently a parameter is updated
  - Momentum
    - Adam incorporates momentum by using an exponentially weighted average of past gradients (first moment) and the squared gradients (second moment)
    - This helps smooth out updates and accelerates convergence

Adam Optimizer

- Advantages of Adam
  - **Efficient:** It requires minimal memory & is computationally efficient, making it suitable for large datasets and models
  - **Adaptive:** The adaptive learning rate allows it to perform well on problems with sparse gradients
  - **Robust:** It often converges faster than other optimizers and works well in practice across a variety of tasks
- **Disadvantages**
  - **Hyperparameter Sensitivity:** **The choice of hyperparameters, particularly the learning rate, can significantly impact performance.**
  - **Not Always Optimal:** **In some cases, it may not converge to the optimal solution as well as other optimizers, particularly in certain deep learning scenarios.**
- **Use Cases**
  - Adam is widely used in training deep learning models across various applications, including computer vision, natural language processing, and reinforcement learning, due to its robustness and efficiency

Machine Learning Vs Deep Learning

- **Machine Learning**
  - Traditional machine learning methods require human input for the machine learning software to work sufficiently well
  - A data scientist manually determines the set of relevant features that the software must analyze
  - This limits the software's ability, which makes it tedious to create and manage

- **Deep Learning**
  - On the other hand, in deep learning, the data scientist gives only raw data to the software
  - The deep learning network derives the features by itself and learns more independently → automatically
  - It can analyze unstructured datasets like text documents, identify which data attributes to prioritize, and solve more complex problems

Types of Neural Network

huge data ← ANN → Artificial NN, fully connected net

1. Convolutional neural networks (CNN) → image processing

- It can input images, identify the objects in a picture, and differentiate them from one another
- Their real-world applications include pattern recognition, image recognition, and object detection
- A CNN's structure consists of three main layers
  - First is the convolutional layer, where most of the computation occurs
  - Second is the pooling layer, where the number of parameters in the input is reduced
  - Lastly, the fully connected layer classifies the features extracted from the previous layers

CNN → ANN (converter) converts images to numeric data

2. Recurrent neural networks(RNN) → text processing → sequential data

- It can translate language, speech recognition, natural language processing, and image captioning
- Examples of products using RNNs include smart home technologies and voice command features on mobile phones (context)
- Feedback loops in the structure of RNNs allow information to be stored similarly to how your memory works

Radial basis functions networks (RBF)

- Radial basis function (RBF) networks differ from other neural networks because the input layer performs no computations
- Instead, it passes the data directly to the hidden layer. As a result, RBFs have a faster learning speed.
- Applications of RBF networks include time series prediction and function approximation

Types of Neural Network

My name is Amit. I live in Pune.

3. Long short-term memory networks (LSTM)
   - These networks are unique and can sort data into short-term and long-term memory cells depending on whether or not the data needs to be looped back into the network as data points or entire sequences
   - LSTM can also be used in handwriting recognition and video-to-text conversion

- Multilayer perceptrons (MLP)
  - These are a neural network capable of learning the relationship between linear and non-linear data
  - Through backpropagation, MLPs can reduce error rates
  - Applications that benefit from MLPs include face recognition and computer vision

4. Generative adversarial networks (GAN)

generate text / image / video → Gen AI

   - They can generate new data sets that share the same statistics as the training set and often pass as actual data
   - An example of this you've likely seen is art created with AI
   - GANs can replicate popular art forms based on patterns in the training set, creating pieces often indistinguishable from human artwork

Types of Neural Network

- Deep belief networks (DBN)
  - They are unique because they stack individual networks that can use each other's hidden network layers as the input for the next layer
  - This allows for the neural networks to be trained faster. They are used to generate images and motion-capture data

- Self-organising maps (SOM)
  - Self-organising maps (SOMs), or Kohonen maps, can transform extensive complex data sets into understandable two-dimensional maps where geometric relationships can be visualized
  - This can happen because SOMs use competitive learning algorithms in which neurons must compete to be represented in the output
  - This is decided by which neurons best represent the input
  - Practical applications of SOMs include displaying voting trends for analysis and organizing complex data collected by astronomers so it can be interpreted

Advantages of Neural Network

- **Ability to Learn Complex Patterns**
  - Neural networks can model intricate relationships in data, making them effective for tasks like image recognition, natural language processing, and speech recognition.
- **Non-Linearity**
  - With activation functions, neural networks can capture non-linear relationships, allowing them to learn more complex functions than linear models.
- **Feature Extraction**
  - Deep neural networks, especially convolutional neural networks (CNNs), can automatically extract relevant features from raw data, reducing the need for manual feature engineering.
- **Scalability**
  - Neural networks can scale well with large amounts of data. They tend to perform better as the dataset size increases, leveraging more data to improve model performance.
- **Generalization**
  - When properly trained, neural networks can generalize well to unseen data, making them suitable for real-world applications.

Advantages of Neural Network

- **Versatility** → type of data, type of problem
  - Neural networks can be adapted to a wide range of tasks, from classification and regression to generative modeling and reinforcement learning. They can be used in various domains, including finance, healthcare, and robotics.
- **Parallel Processing** → distributed computing
  - Neural networks can take advantage of parallel processing, particularly when implemented on GPUs, significantly speeding up training times.
- **Robustness to Noise**
  - They can be resilient to noisy inputs, which is beneficial in many real-world applications where data is imperfect.
- **Transfer Learning** *** chatbot → [text processing] → custom data → embeddings → (database) → chroma
  - Pre-trained neural networks can be fine-tuned for specific tasks, enabling efficient training on smaller datasets while leveraging knowledge from larger, related datasets.
- **Continuous Learning** **
  - Neural networks can be designed to learn continuously from new data, allowing them to adapt to changing environments over time.

Disadvantages of Neural Network

- **Complexity**
  - Neural networks can be highly complex, making them difficult to design, tune, and interpret. Choosing the right architecture and hyperparameters often requires significant expertise and experimentation.
- **Data Requirements**
  - They typically require large amounts of labeled data to train effectively. Insufficient data can lead to overfitting, where the model learns noise rather than meaningful patterns.
- **Computational Cost**
  - Training neural networks can be resource-intensive, requiring significant computational power and memory, especially for deep architectures. This can lead to long training times.
- **Overfitting** high train & low test accuracy
  - Without proper regularization techniques, neural networks can overfit to the training data, especially when the model is too complex relative to the amount of training data. → Dropout, Early stop, Regularization
- **Lack of Interpretability**
  - Neural networks are often considered "black boxes." Understanding how they make decisions can be challenging, which is a critical concern in applications like healthcare and finance where interpretability is essential.

Disadvantages of Neural Network

Gradient → features / weights

- **Gradient Vanishing and Exploding**
  - In deep networks, gradients can become very small (vanishing) or very large (exploding), making training difficult. This issue can hinder learning and convergence, especially in recurrent neural networks (RNNs).
- **Hyperparameter Tuning**
  - Neural networks have many hyperparameters (e.g., learning rate, batch size, number of layers) that can significantly affect performance. Finding the right combination often requires extensive experimentation.
- **Dependence on Initialization**
  - The initial weights of a neural network can affect the training process. Poor initialization can lead to suboptimal solutions or slow convergence.
- **Sensitivity to Noise**
  - While they can be robust to some noise, neural networks can also be sensitive to outliers or adversarial examples, which can lead to misclassification.
- **Long Training Times**
  - Training deep networks can take a long time, particularly without sufficient computational resources. This can be a barrier to rapid prototyping and experimentation.

① tensorflow
- → google
- → ecosystem
- → building products
- → cross platform support
  - → desktop
  - → web
  - → mobile apps
- → less configurable
- → less coding

② pytorch
- → meta
- → used in research tasks
- → fully configurable
- → more coding

TensorFlow

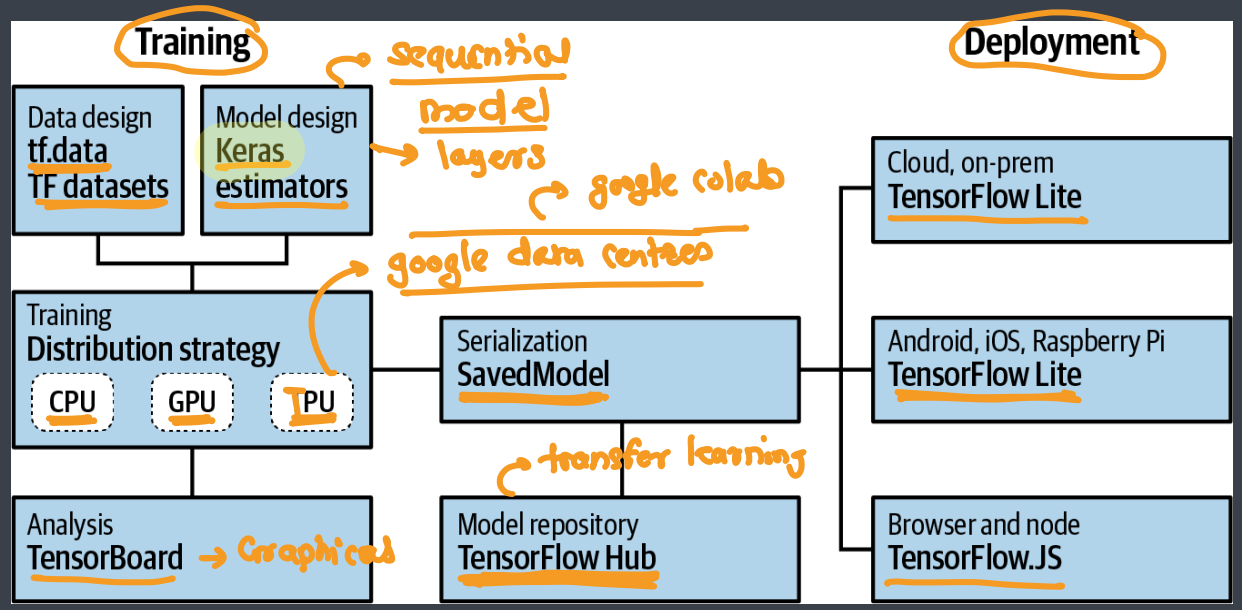
Google

TensorFlow

- TensorFlow is an open source platform for creating and using machine learning models
- It implements many of the common algorithms and patterns needed for machine learning, saving you from needing to learn all the underlying math and logic and enabling you to just to focus on your scenario
- It's aimed at everyone from hobbyists, to professional developers, to researchers pushing the boundaries of artificial intelligence
- Importantly, it also supports deployment of models to the web, cloud, mobile, and embedded systems

tensor ⇒ array of values

⇒ similar to vectors in R



Hello World

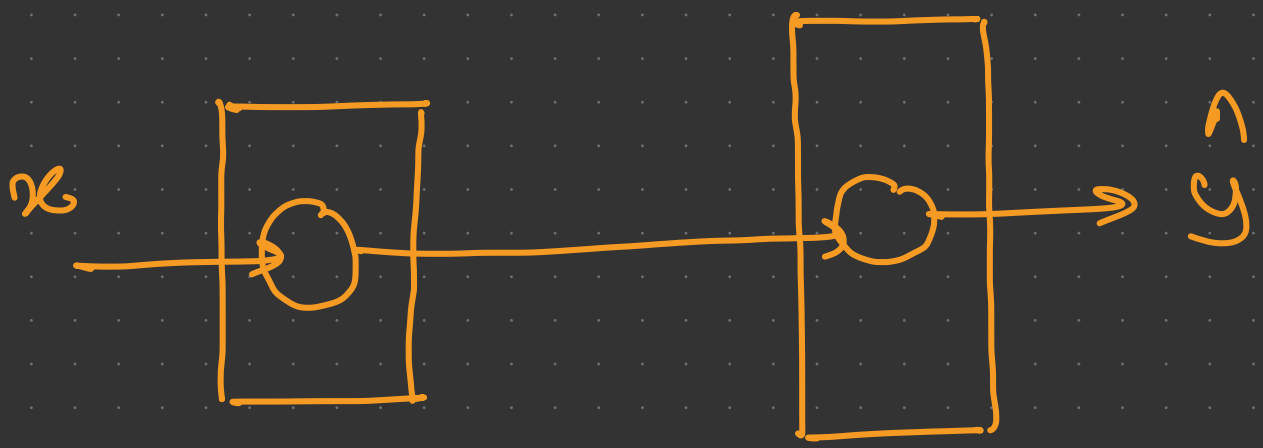
```
import tensorflow as tf
import numpy as np
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([Dense(units=1, input_shape=[1])])
model.compile(optimizer='sgd', loss='mean_squared_error')

xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
ys = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype=float)

model.fit(xs, ys, epochs=500)

print(model.predict([10.0]))
```

model

layer

layer

applicable only for input layer



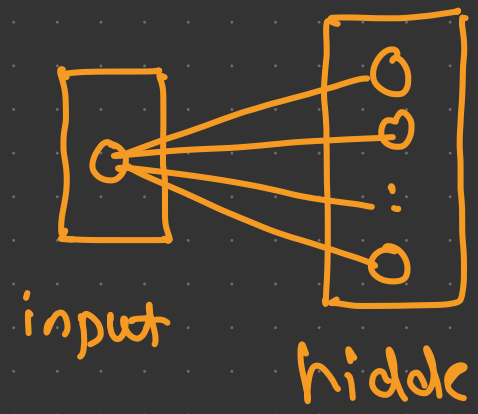
input layer

output layer

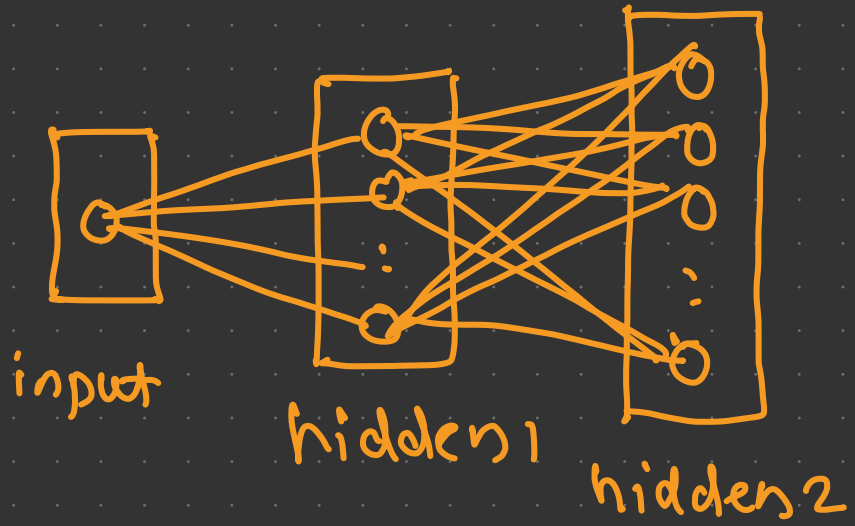
Dense(units = n, input_shape = [s])

↑ # independent variables

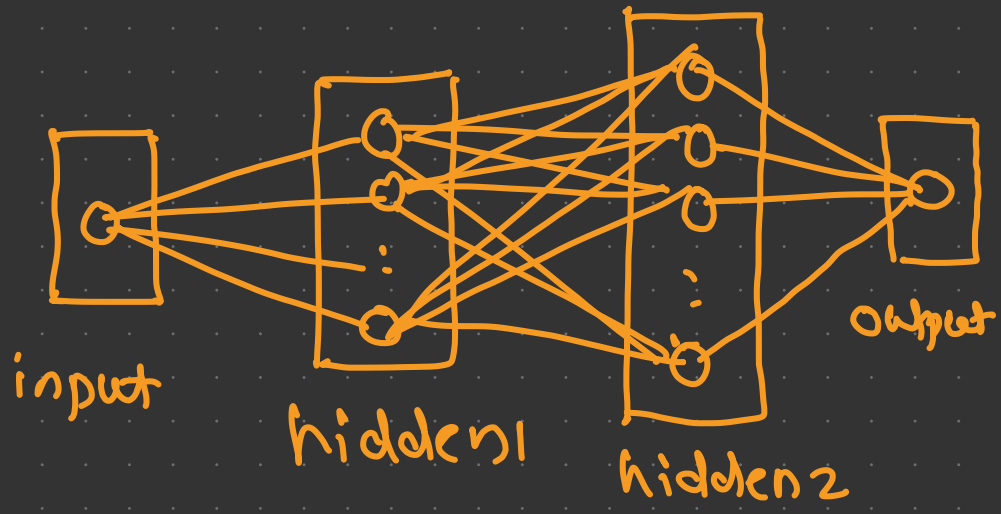
model.add(Dense(unit=10, input_shape=[1]))



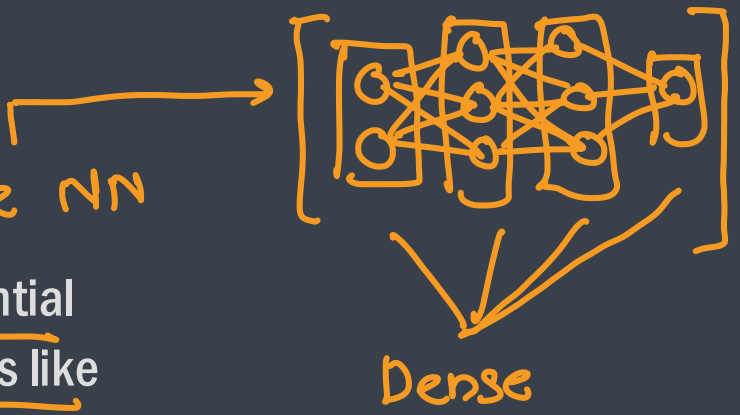
model.add(Dense(unit=20))



model.add(Densen(units=1))



Hello World Explained



- Sequential
  - When using TensorFlow, you define your layers using Sequential
  - Inside the Sequential, you then specify what each layer looks like
- Layer
  - A layer is represented as Dense in TensorFlow
  - "Dense" means a set of fully (or densely) connected neurons where where every neuron is connected to every neuron in the next layer
- Compiling model
  - The computer starts guessing to create the model (formula) and comparing with the observed value
  - The optimizer used in this stage helps the machine to optimize the guess
    - SGD: stochastic gradient descent, a complex mathematical function that, when given the values, the previous guess, and the results of calculating the errors (or loss) on that guess, can then generate another one
    - ADAM: the optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments

Tensorboard Requirements

- Install tensorboard
  - pip install tensorflow tensorboard
- Install tensorboard extension
  - pip install jupyter-tensorboard
- Start tensorboard
  - %load_ext tensorboard
  - %tensorboard --logdir logs/fit

Hello World with Tensorboard

```python
import tensorflow as tf
from tensorflow import keras
import datetime

# Define a callback to save logs
log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)

# Sample model (replace with your model)
model = keras.Sequential([
    keras.layers.Dense(64, activation='relu', input_shape=(784,)),
    keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Dummy data (replace with your training data)
import numpy as np
x_train = np.random.random((1000, 784))
y_train = np.random.randint(10, size=(1000,))

# Train the model with TensorBoard callback
model.fit(x_train, y_train, epochs=5, callbacks=[tensorboard_callback])
```