

# Deep Learning

# Introduction

## What are Neural Networks?

- ❖ Neural networks are a new method of programming computers.
- ❖ They are exceptionally good at performing pattern recognition and other tasks that are very difficult to program using conventional techniques.
- ❖ Programs that employ neural nets are also capable of learning on their own and adapting to changing conditions.
- ❖ An NN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of NNs as well.

# Is Deep learning Worth the Hype ?

- Wide range of application



- Automated Feature Selection
- Scalable and Robust
- Architecture Innovation and Flexibility
- Understand the underlying distribution of the data

# Building Components

Layers

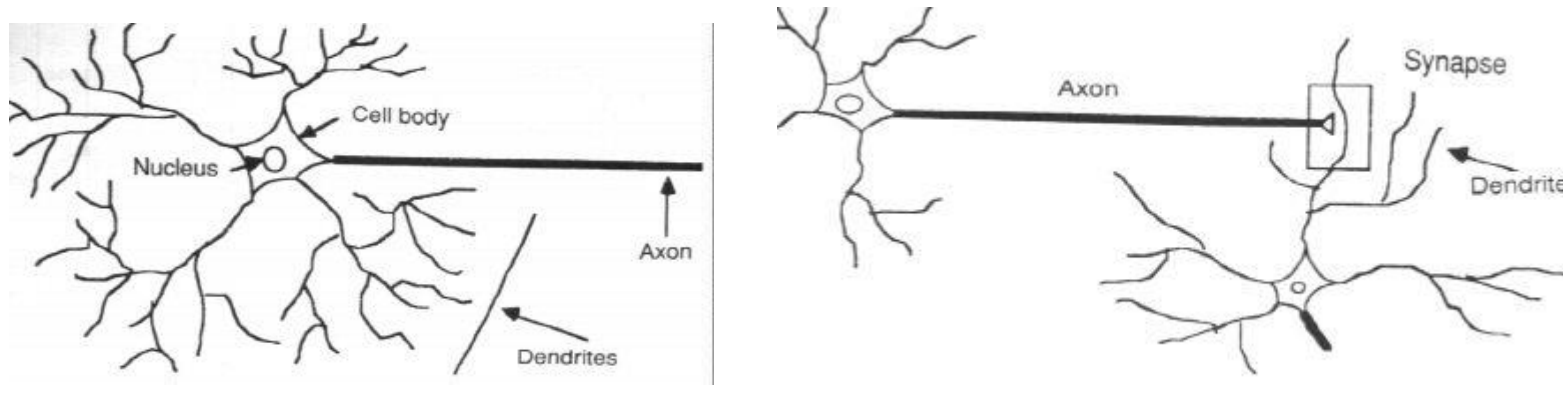
Nodes

Activation  
Functions

Optimizers

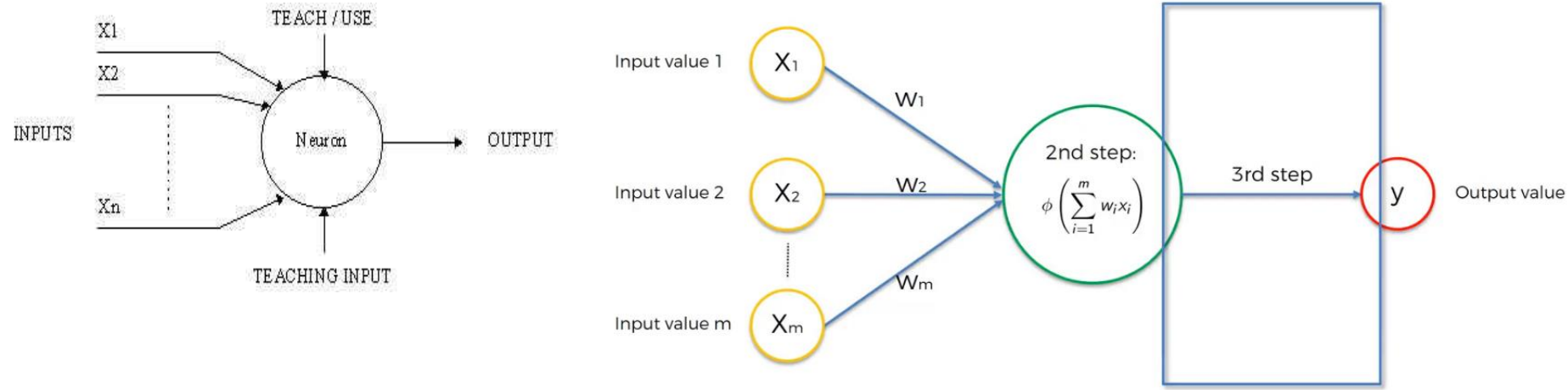
Back  
Propagation

# How the Human Brain learns



- ❖ In the human brain, a typical neuron collects signals from others through a host of fine structures called *dendrites*.
- ❖ The neuron sends out spikes of electrical activity through a long, thin strand known as an *axon*, which splits into thousands of branches.
- ❖ At the end of each branch, a structure called a *synapse* converts the activity from the axon into electrical effects that inhibit or excite activity in the connected neurons.

# A Simple Neuron



An artificial neuron is a device with many inputs and one output.

The neuron has two modes of operation;

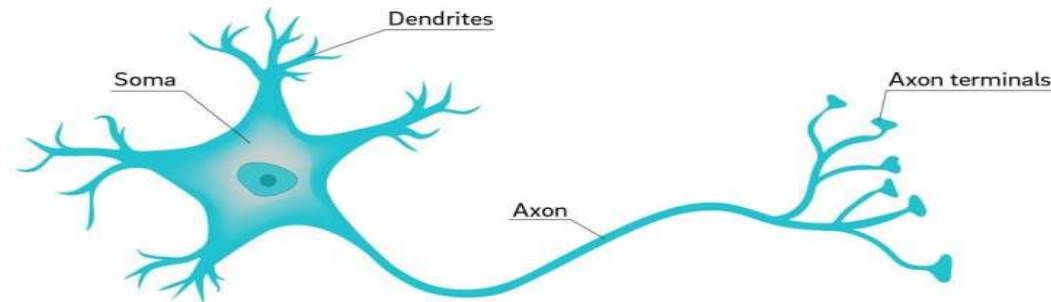
- ✓ the training mode
- ✓ the using mode.

- ❖ In the training mode, the neuron can be trained to fire (or not), for particular input patterns.
- ❖ In the using mode, when a taught input pattern is detected at the input, its associated output becomes the current output. If the input pattern does not belong in the taught list of input patterns, the firing rule is used to determine whether to fire or not.

# What is a neuron?

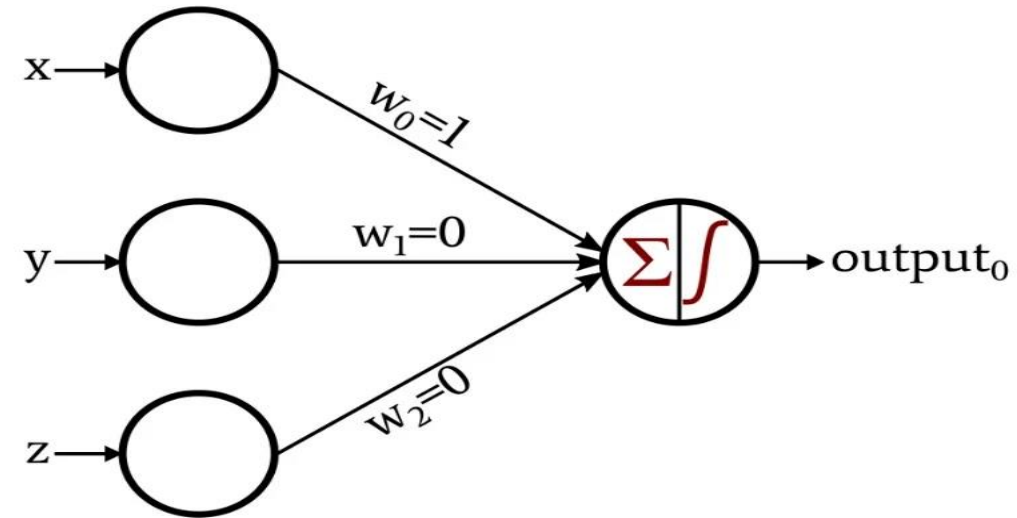
- The **neuron** is the basic working unit of the brain, a specialized cell designed to transmit information to other nerve cells, muscle, or gland cells. **Neurons** are cells within the nervous system that transmit information to other nerve cells, muscle, or gland cells

Neuron



# perceptron

A **perceptron** is a neural network unit (an artificial neuron) that does certain computations to detect features or business intelligence in the input data

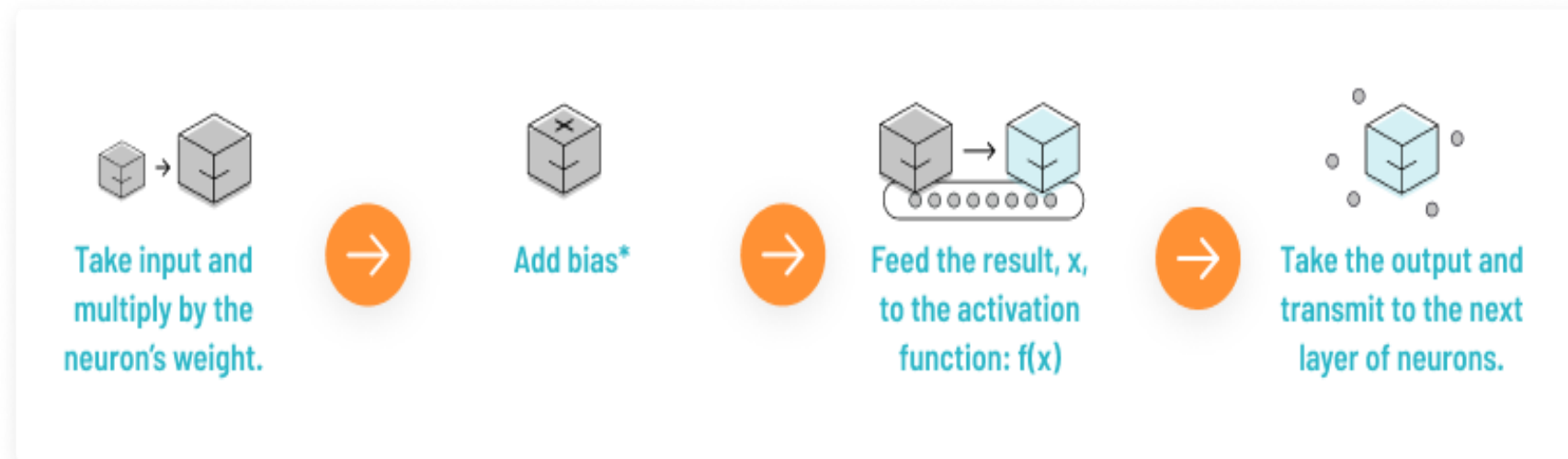




# Activation function

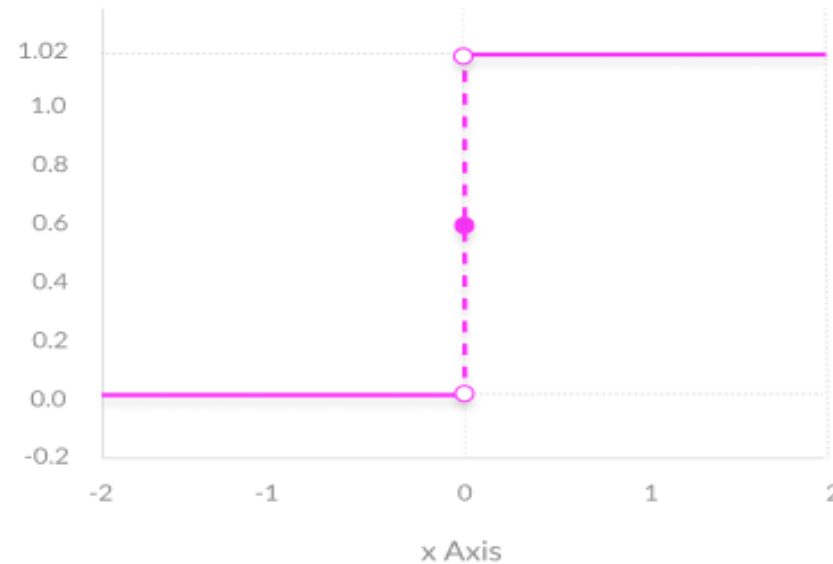
- Activation functions are mathematical equations that determine the output of a neural network. The function is attached to each neuron in the network, and determines whether it should be activated (“fired”) or not, based on whether each neuron’s input is relevant for the model’s prediction.
  - What is the role of Activation function?

# process

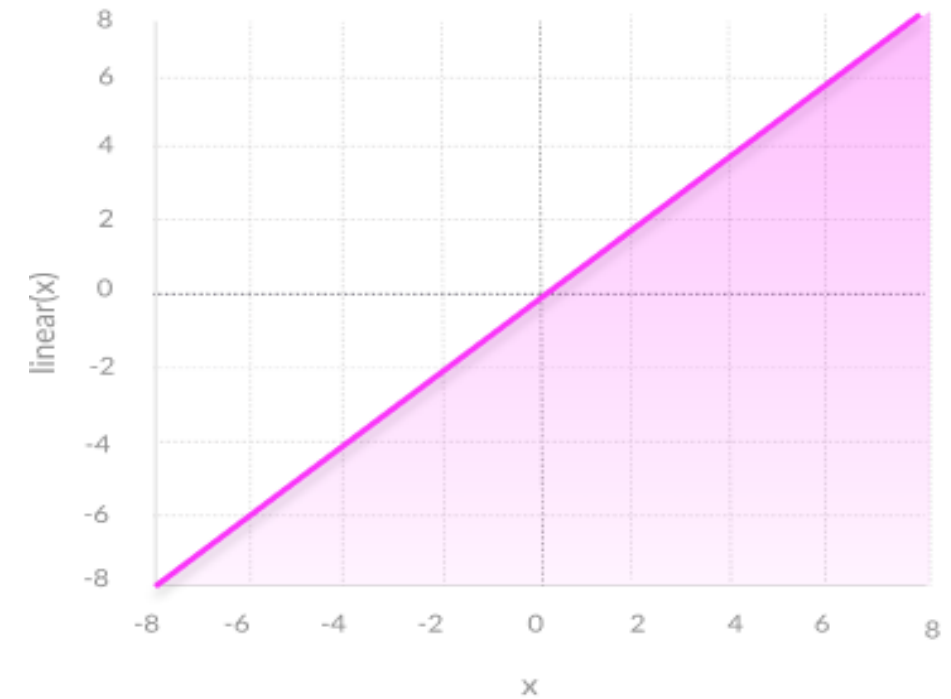


# Types of activation functions

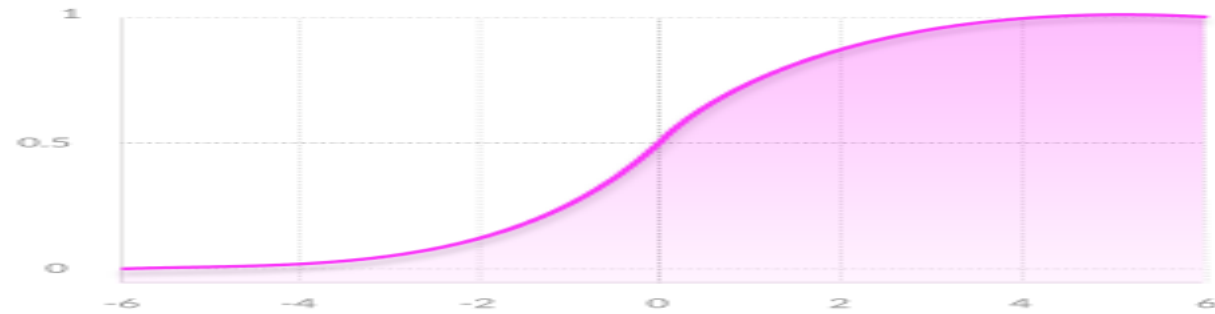
- Binary Step Function



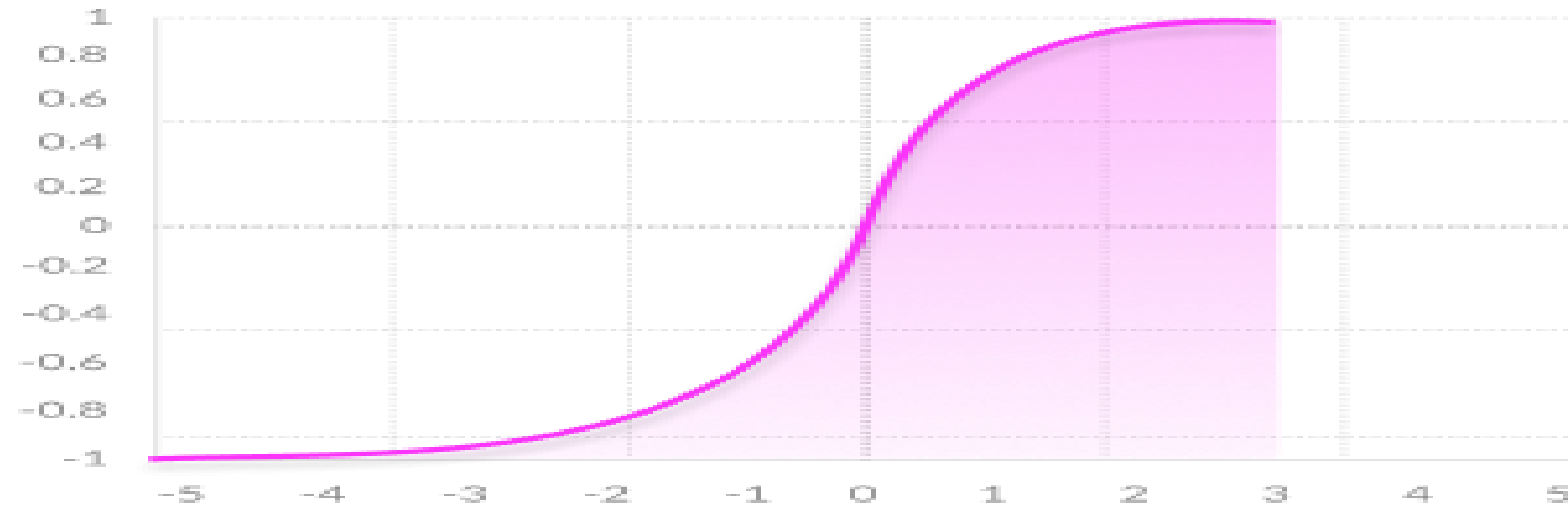
# Linear activation function



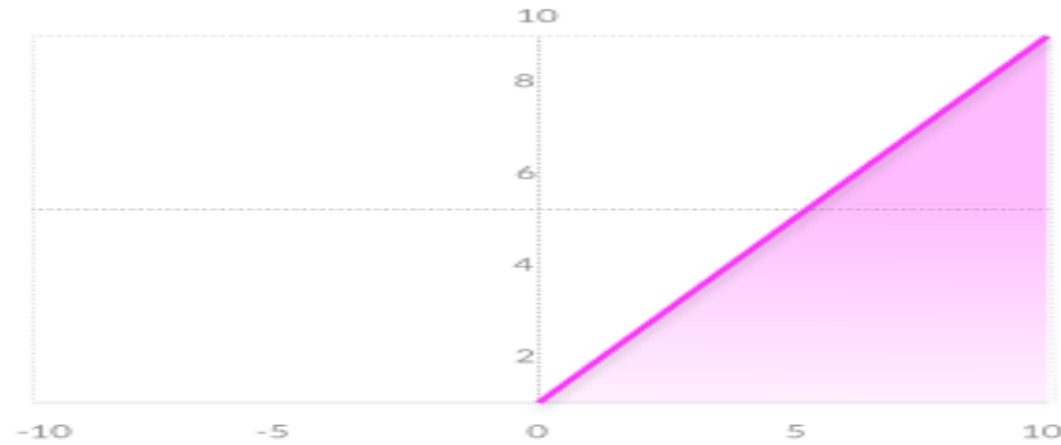
# Sigmoid function



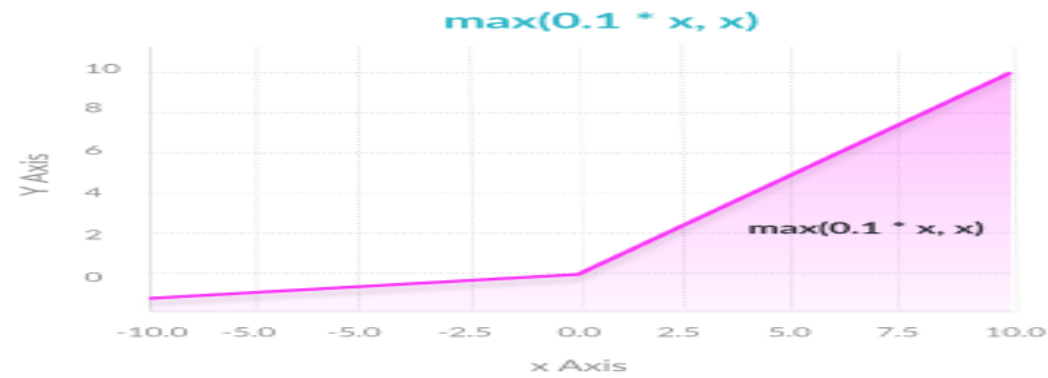
# TanH / Hyperbolic Tangent



# ReLU (Rectified Linear Unit)

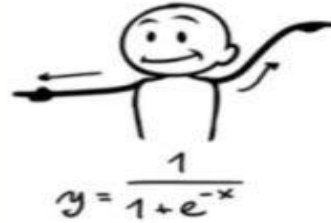


# Leaky ReLU

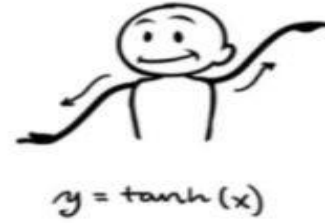




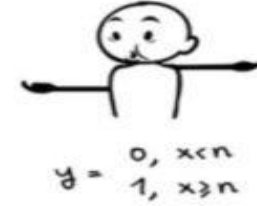
Sigmoid



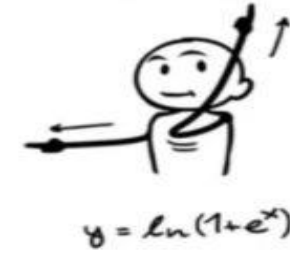
Tanh



Step Function

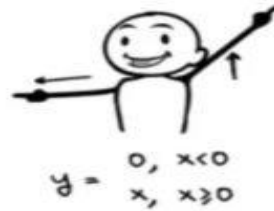


Softplus

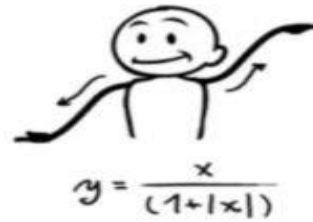


source: sefiks

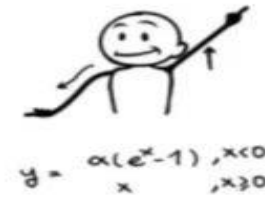
ReLU



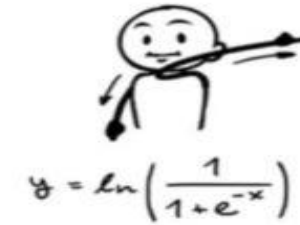
Softsign



ELU



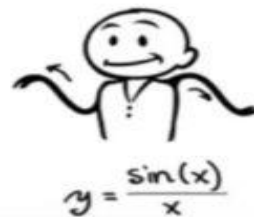
Log of Sigmoid



Swish



Sinc



Leaky ReLU



Mish

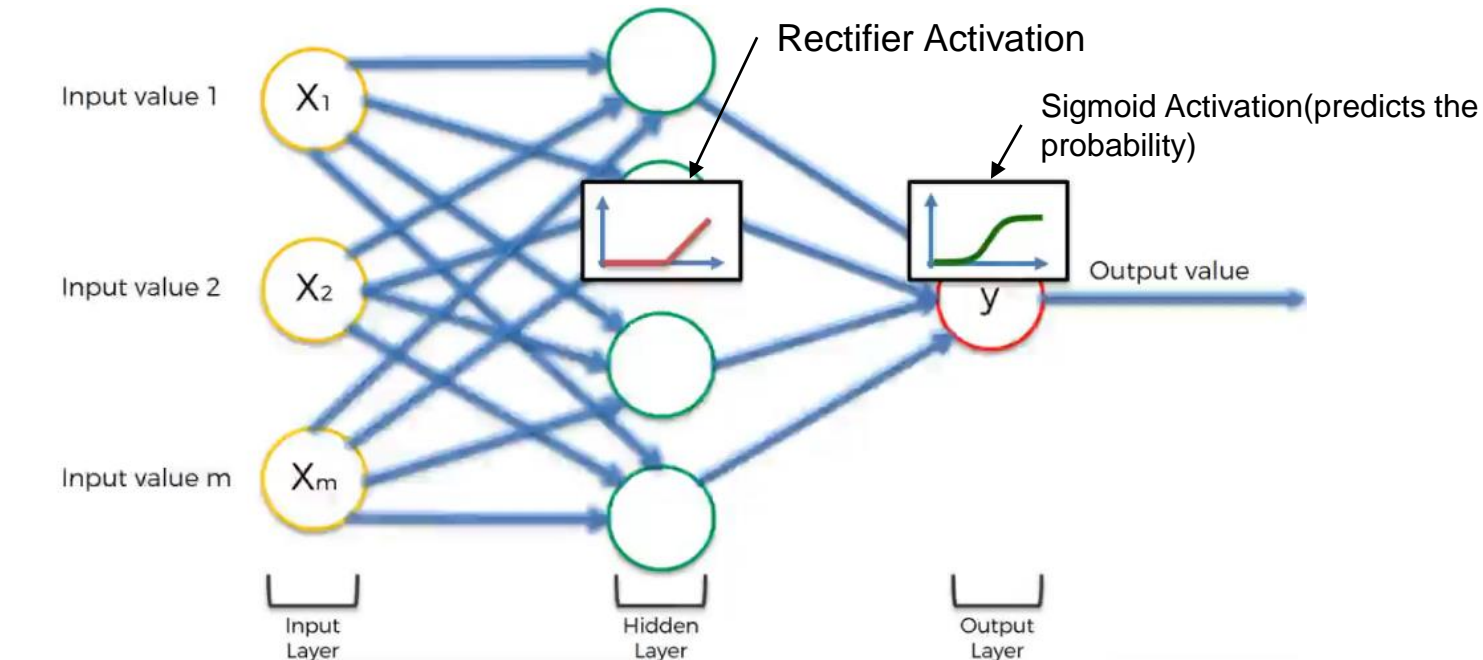


# Activation functions

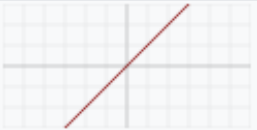


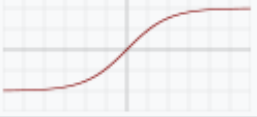

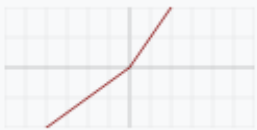
- ❖ Decides whether a neuron should be activated or not.
- ❖ Whether the information that the neuron is receiving is relevant for the given information or should it be ignored.

$$Y = \text{Activation}(\Sigma(\text{weight} * \text{input}) + \text{bias})$$

- ❖ The activation function is the non linear transformation that we do over the input signal. This transformed output is then sent to the next layer of neurons as input.



# Different activation functions:

Name ↕	Plot ↕	Equation
Identity		$f(x) = x$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Logistic (a.k.a. Sigmoid or Soft step)		$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$ <sup>[1]</sup>
TanH		$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$
Rectified linear unit (ReLU) <sup>[10]</sup>		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Leaky rectified linear unit (Leaky ReLU) <sup>[11]</sup>		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$

Name ↕	Equation ↕
Softmax	$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^J e^{x_j}} \quad \text{for } i = 1, \dots, J$
Maxout <sup>[23]</sup>	$f(\vec{x}) = \max_i x_i$

# Choosing the right Activation Function

- ❖ Sigmoid functions and their combinations generally work better in the case of classifiers
- ❖ Sigmoid and tanh functions are sometimes avoided due to the vanishing gradient problem
- ❖ ReLU function is a general activation function and is used in most cases these days
- ❖ If we encounter a case of dead neurons in our networks the leaky ReLU function is the best choice
- ❖ Always keep in mind that ReLU function should only be used in the hidden layers
- ❖ As a rule of thumb, you can begin with using ReLU function and then move over to other activation functions in case ReLU doesn't provide with optimum results

# FFNN

- Input layer
- Hidden layer
- Output layer
- Importance of hidden layer

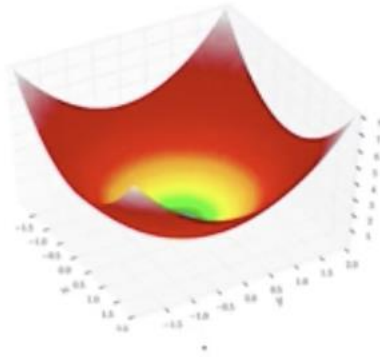
# Training the NN

- Given the  $x(i)$  and  $y(i)$
- Think of what NN design and hyper parameter design might work
- Form a neural network
  - $F(x)$
- Compute weights as estimating the  $y$  for all samples
- Compute the loss
- Tweak  $w$  to reduce the loss
- Repeat the last three steps

# Error and loss function

- In general error/loss for a NN is difference between actual and predicted.
- The goal is to minimize the loss
- Loss function is a function that is used to calculate the error
- You have to choose the loss function based on the problem you have at hand.
- Loss functions are different for classification and regression

# Optimization



## Optimization

In optimization, the main aim is to find weights that reduce loss



# Gradient Descent

- Gradient is calculated by optimizing function
- Gradient is the change in loss with change in weights
- The weights are modified according to the calculated gradient
- Same process is repeated until minima is reached

# Gradient descent variations

- Batch Gradient descent
- Stochastic gradient descent
- Mini-batch gradient descent

# Back Propagation in NN

- Backprop is used while training feed forward NN
- It helps in efficiently calculating the gradient of the loss function with respect to weights
- It helps in minimizing loss by updating weights

# Learning Rate and Momentum

- LR is a hyper parameter to what extent newly acquired weights overrides the existing weights , lies between 0 to 1
- Momentum is used to decide the weight on nodes from previous iterations. It helps in improving training speed and also avoiding local minimas\*

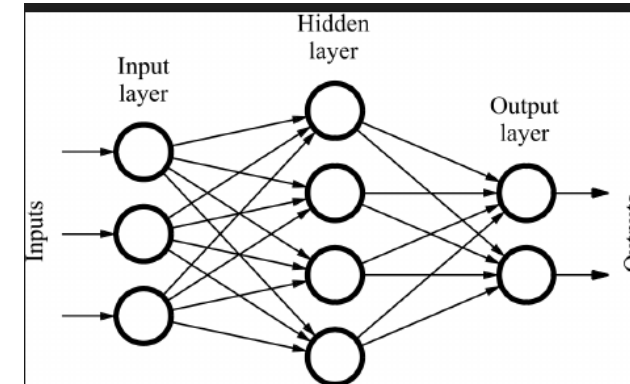
# Early stopping and model saving

- We need to monitor the loss on the validation set while training the NN and if we feel there is no change , then we should stop training.
- Save all the necessary details to reuse the model
- Save the weights, biases and also the model architecture.

# Different types of Neural Networks

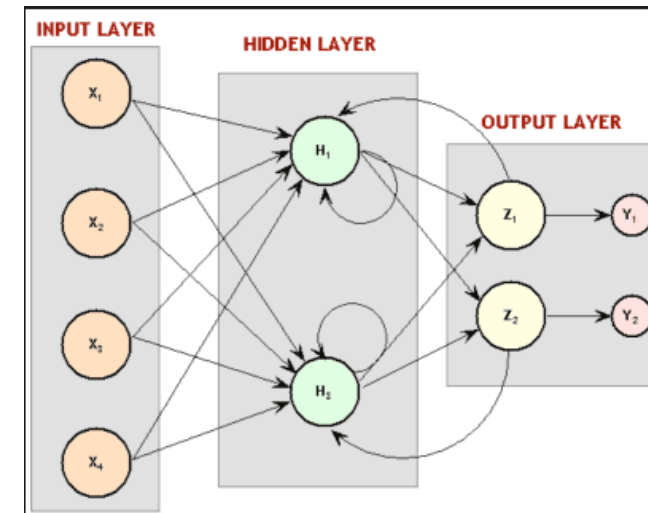
## Feed-forward networks

- ❖ Feed-forward NNs allow signals to travel one way only; from input to output. There is no feedback (loops) i.e. the output of any layer does not affect that same layer.
- ❖ Feed-forward NNs tend to be straight forward networks that associate inputs with outputs. They are extensively used in pattern recognition.
- ❖ This type of organization is also referred to as bottom-up or top-down.



## Feedback networks

- ❖ Feedback networks can have signals traveling in both directions by introducing loops in the network.
- ❖ Feedback networks are dynamic; their 'state' is changing continuously until they reach an equilibrium point.
- ❖ They remain at the equilibrium point until the input changes and a new equilibrium needs to be found.
- ❖ Feedback architectures are also referred to as interactive or recurrent, although the latter term is often used to denote feedback connections in single-layer organizations



# A simple Neural Network

## Network Layers:

**Input Layer** - The activity of the input units represents the raw information that is fed into the network.

**Hidden Layer** - The activity of each hidden unit is determined by the activities of the input units and the weights on the connections between the input and the hidden units.

The weights between the input and hidden units determine when each hidden unit is active, and so by modifying these weights, a hidden unit can choose what it represents.

**Output Layer** - The behavior of the output units depends on the activity of the hidden units and the weights between the hidden and output units.

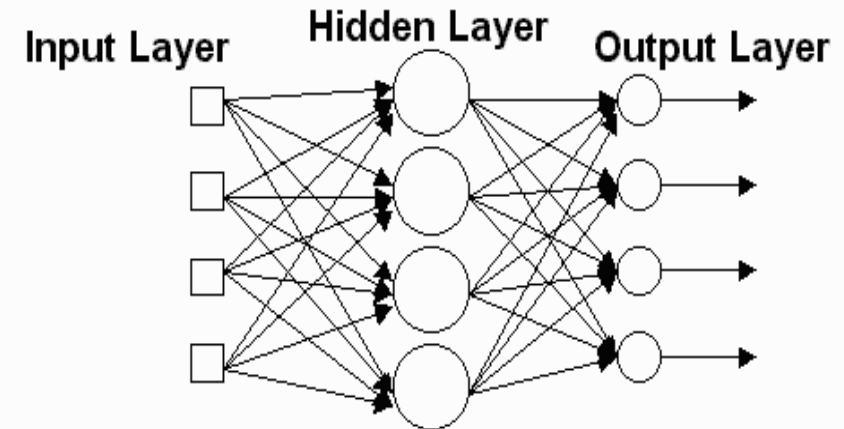
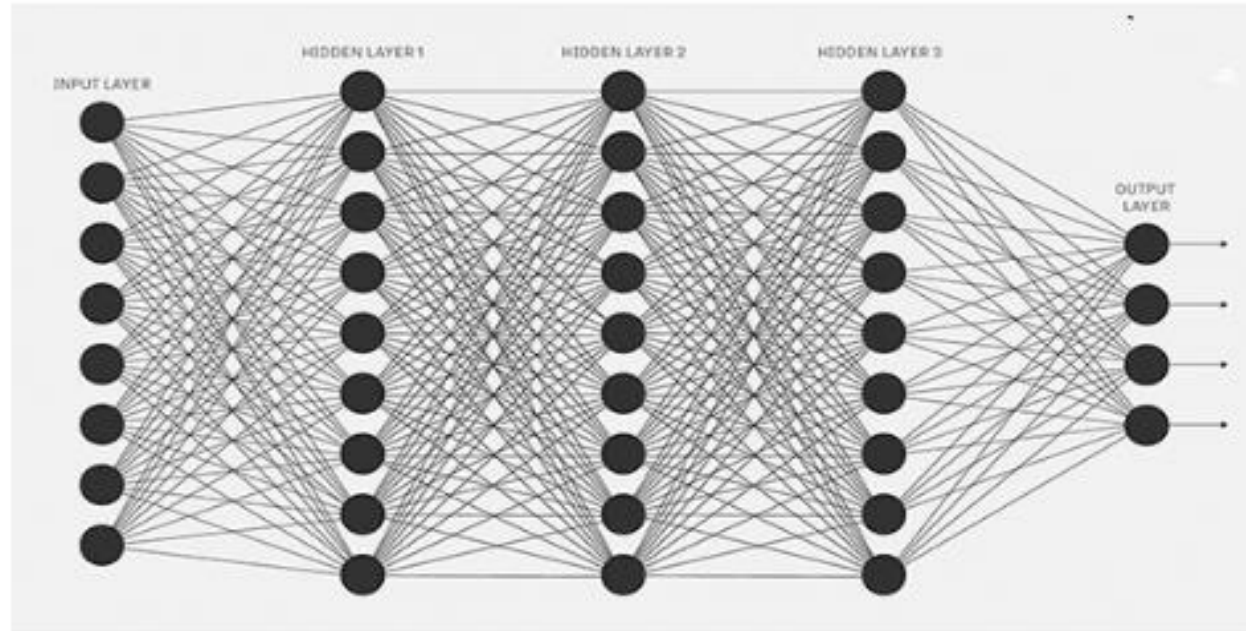


Figure 2 The anatomy of a neural network.

# Neural Network with multiple hidden layers





# Network Structure

The number of layers and of neurons depend on the specific task. In practice this issue is solved by trial and error.

Two types of adaptive algorithms can be used:

- ✓ start from a large network and successively remove some neurons and links until network performance degrades.
- ✓ begin with a small network and introduce new neurons until performance is satisfactory.

# Network Parameters

- ✓ How are the weights initialized?
- ✓ How many hidden layers and how many neurons?
- ✓ How many examples in the training set?

## **Weights:**

In general, initial weights are randomly chosen, with typical values between -1.0 and 1.0 or -0.5 and 0.5.

There are two types of NNs

- Fixed Networks – where the weights are fixed
- Adaptive Networks – where the weights are changed to reduce prediction error.

## **Size of Training Data:**

The number of training examples should be at least five to ten times the number of weights of the network.

# Training of NN

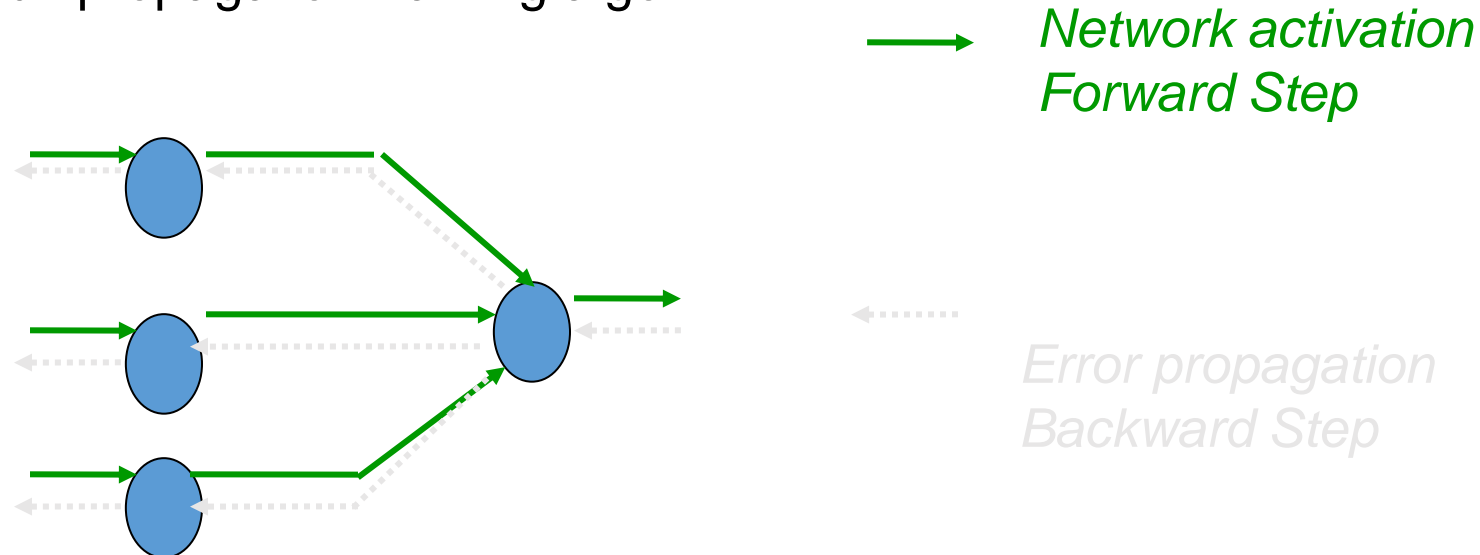
- ❖ The most basic method of training a neural network is trial and error.
- ❖ If the network isn't behaving the way it should, change the weighting of a random link by a random amount. If the accuracy of the network declines, undo the change and make a different one.
- ❖ It takes time, but the trial and error method does produce results.

## Backprop algorithm:

- ❖ The Backprop algorithm searches for weight values that minimize the total error of the network over the set of training examples (training set).
- ❖ Backprop consists of the repeated application of the following two passes:
  - **Forward pass:** in this step the network is activated on one example and the error of (each neuron of) the output layer is computed.
  - **Backward pass:** in this step the network error is used for updating the weights. Starting at the output layer, the error is propagated backwards through the network, layer by layer. This is done by recursively computing the local gradient of each neuron.

# Back Propagation

- ❖ Back-propagation training algorithm

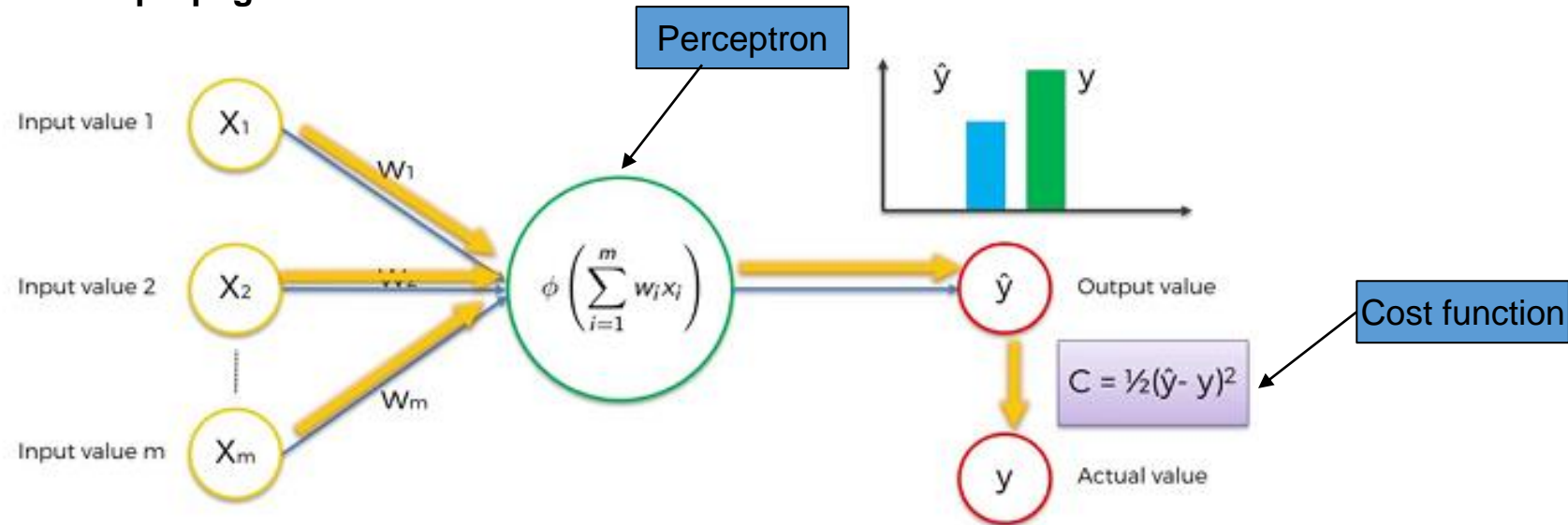


- ❖ Back-propagation adjusts the weights of the NN in order to minimize the network total mean squared error.

# Disadvantage of Neural Network

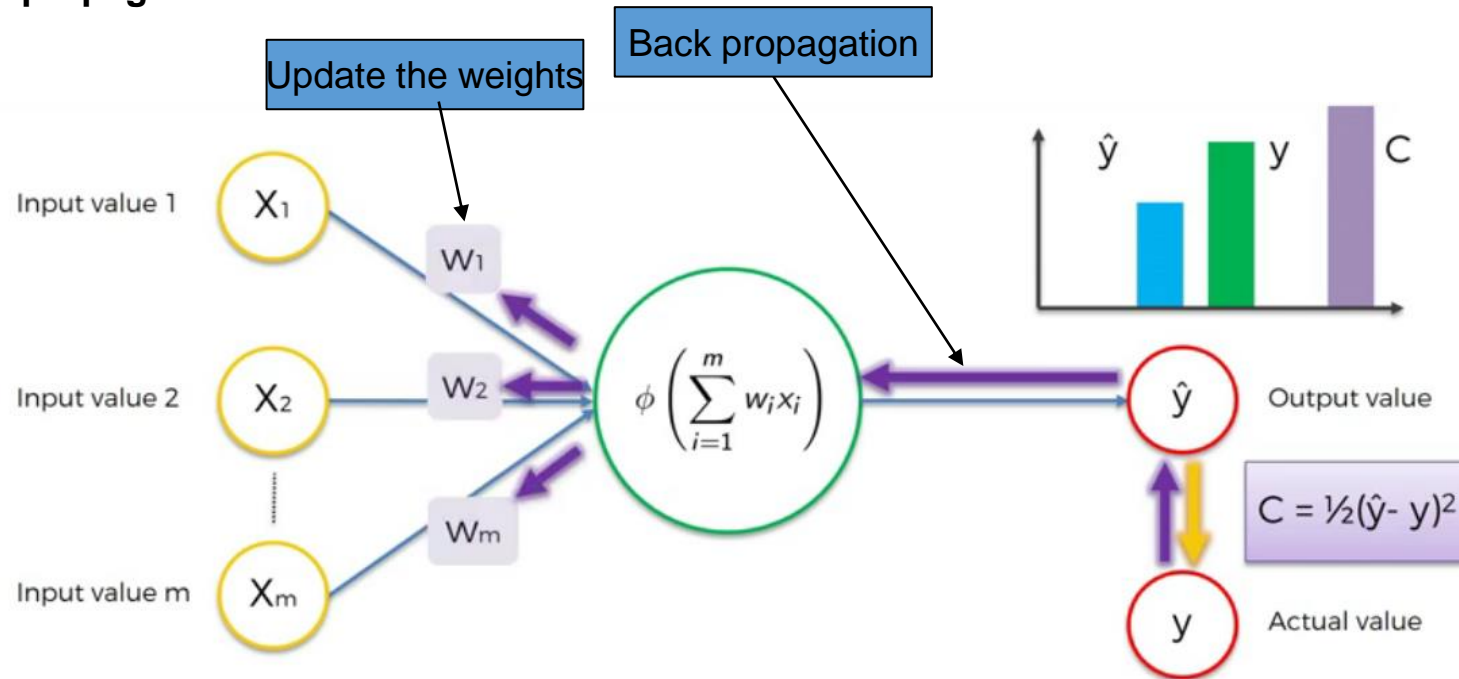
- ❖ The individual relations between the input variables and the output variables are not developed by engineering judgment so that the model tends to be a black box or input/output table without analytical basis.
- ❖ The sample size has to be large.
- ❖ Requires lot of trial and error so training can be time consuming.

## Forward propagation

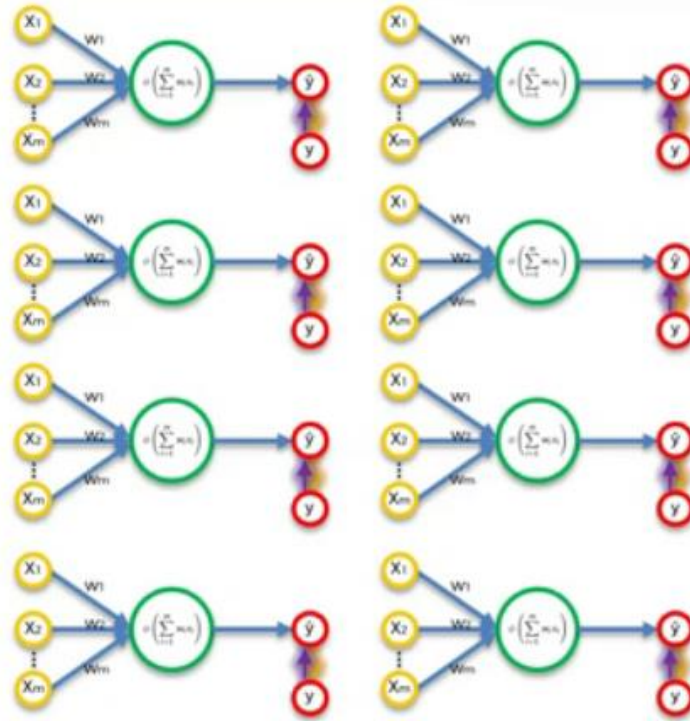


1. Feed features to the input layer
2. Calculate the weights of each neuron
3. Predict the output
4. Compare the actual and predicted output

## Back propagation

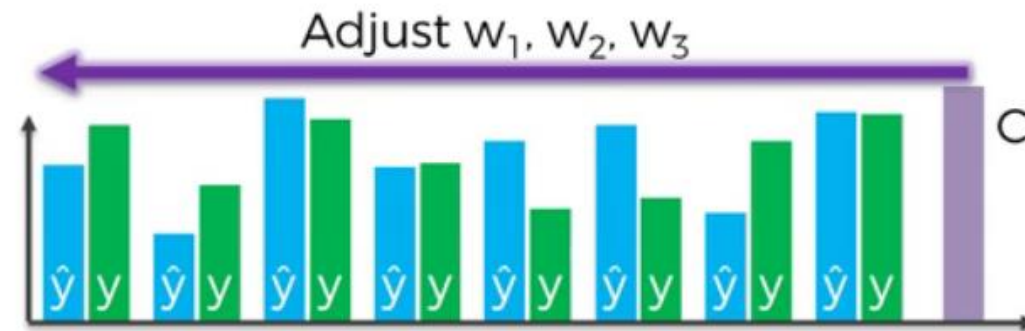


1. Calculate the cost function for predicted output in the forward propagation
2. Back propagate(Gradient descent) and adjust the weights of the layers so that cost function is minimised



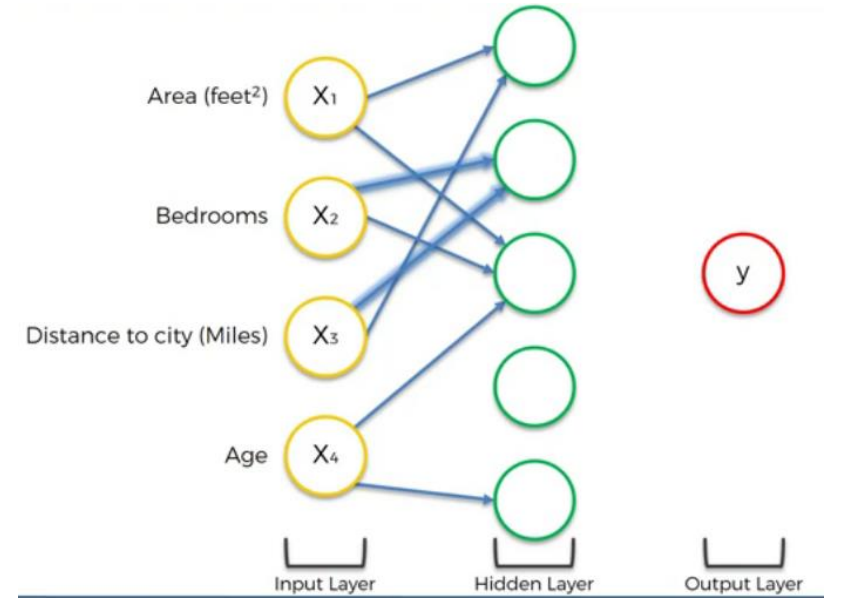
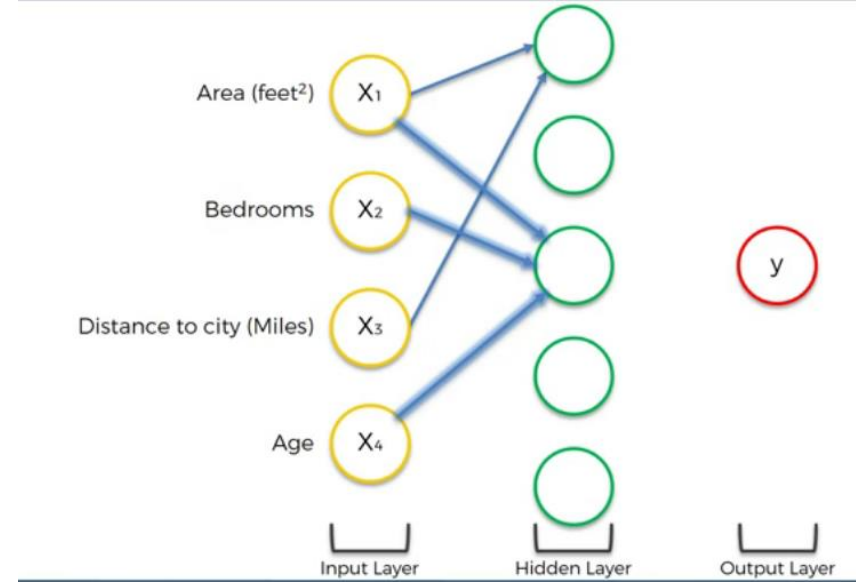
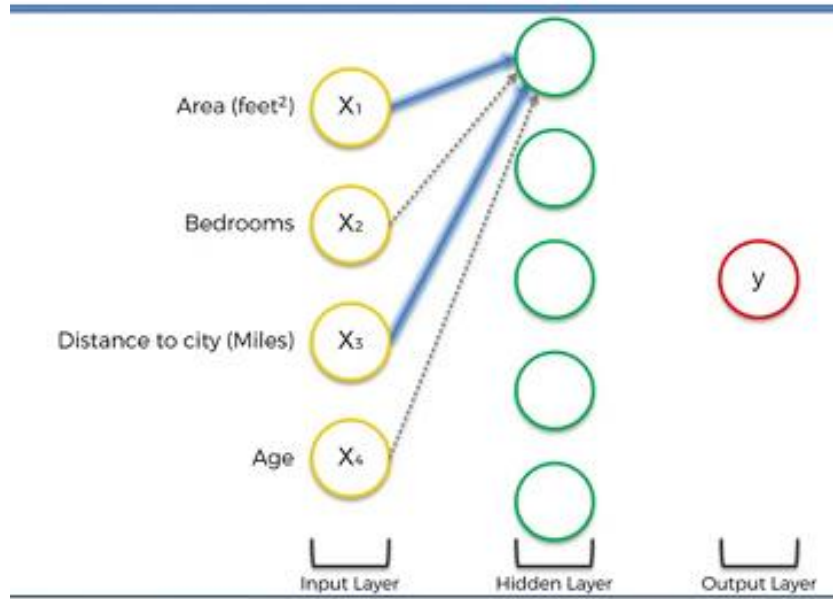
Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

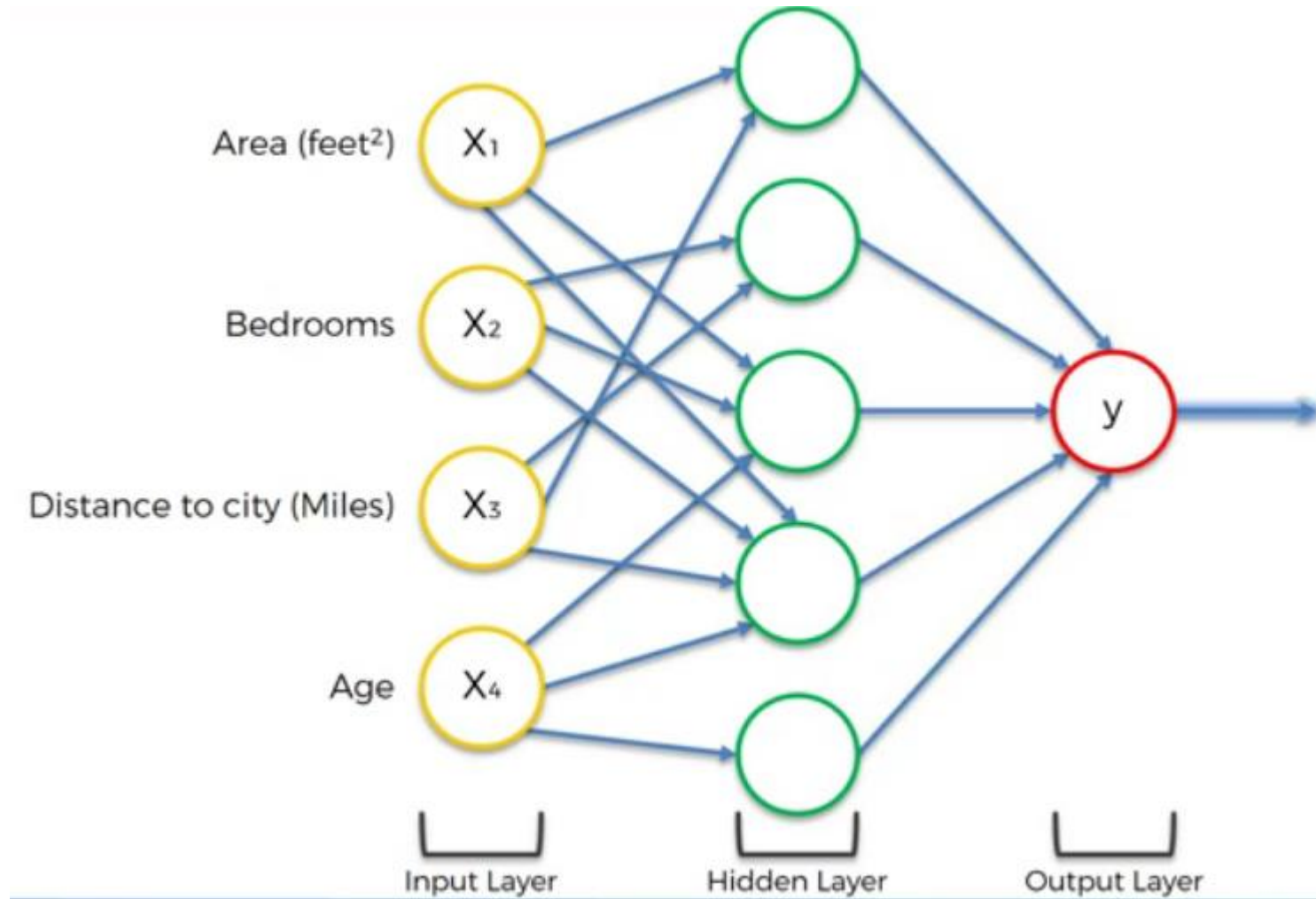
$$C = \sum \frac{1}{2}(\hat{y} - y)^2$$



- ❖ Predict the output for all observations
- ❖ Calculate the cost function
- ❖ Adjust the weight so that cost function is minimised
- ❖ Again predict the output with new adjusted weights and repeat the process until the cost function is minimised

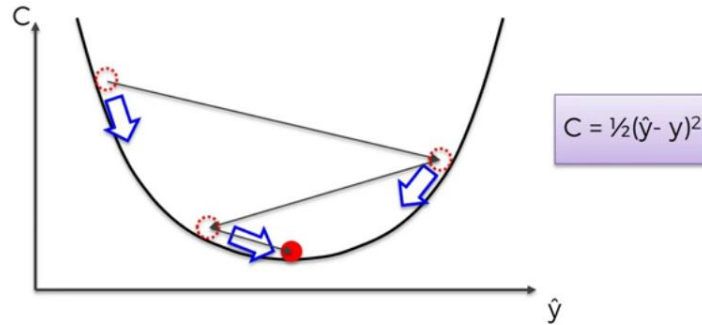




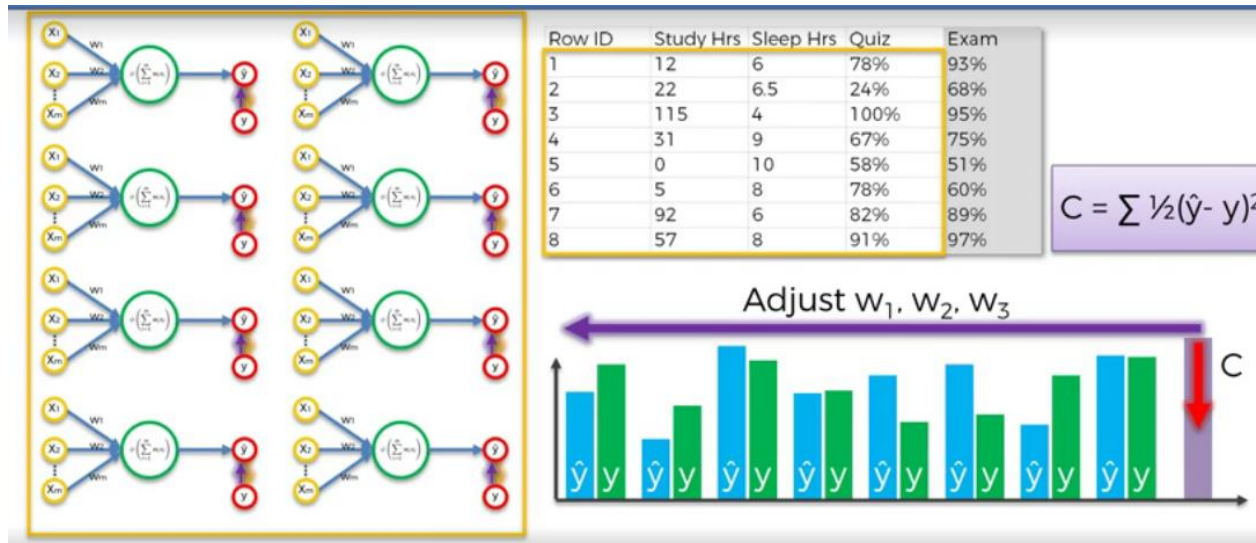
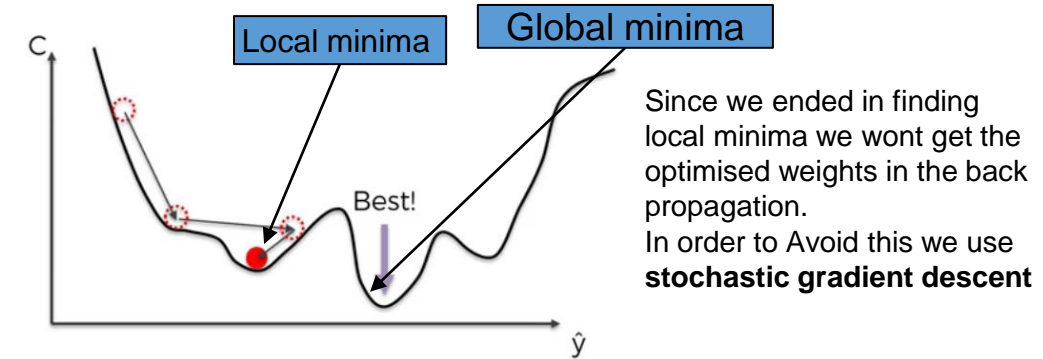


# Gradient Descent

Convex function



Non Convex function



We will calculate the cost function for all observations and update the weights to minimise the cost function .  
This is called **Gradient descent /Batch Gradient descent**

# Stochastic Gradient Descent

Upd w's ←

Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

Batch  
Gradient  
Descent

Upd w's ←

Upd w's ←

Upd w's ←

Upd w's ←

Upd w's ←

Upd w's ←

Upd w's ←

Upd w's ←

Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

Stochastic  
Gradient  
Descent

Stochastic	Batch
Local minima problem is can be resolved	May end up in local minima
Fast	Slow
Less deterministic	More deterministic

We will calculate the cost function for each observations and update the weights to minimise the cost function .

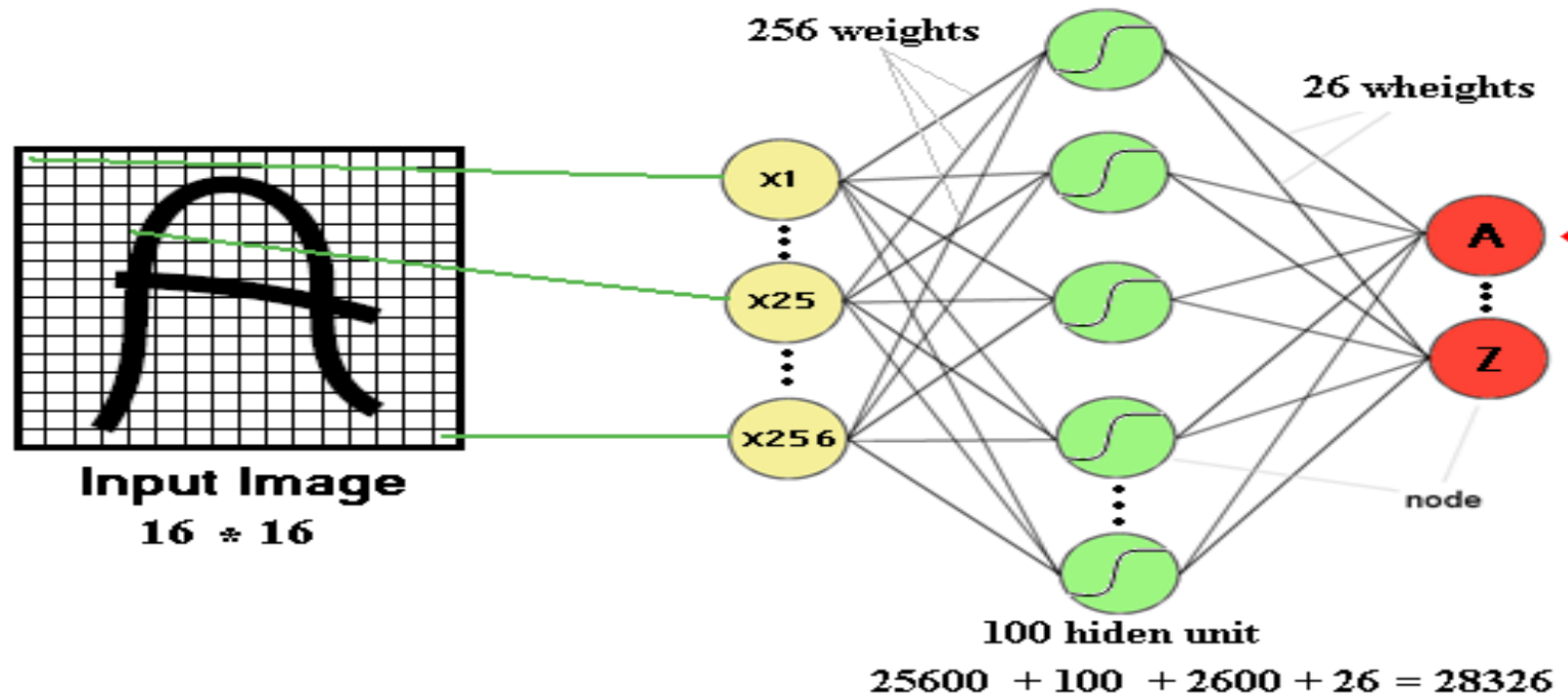
This is called **Stochastic Gradient descent**

# Mini Batch Gradient Descent

	Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
Upd w's ←	1	12	6	78%	93%
	2	22	6.5	24%	68%
	3	115	4	100%	95%
	4	31	9	67%	75%
Upd w's ←	5	0	10	58%	51%
	6	5	8	78%	60%
Upd w's ←	7	92	6	82%	89%
	8	57	8	91%	97%

- ❖ We will first divide the data into mini batches (5 or 8 observations per batch )
- ❖ We will calculate the cost function for each mini observations and update the weights to minimise the cost function .
- ❖ This is called **Mini batch Gradient descent**

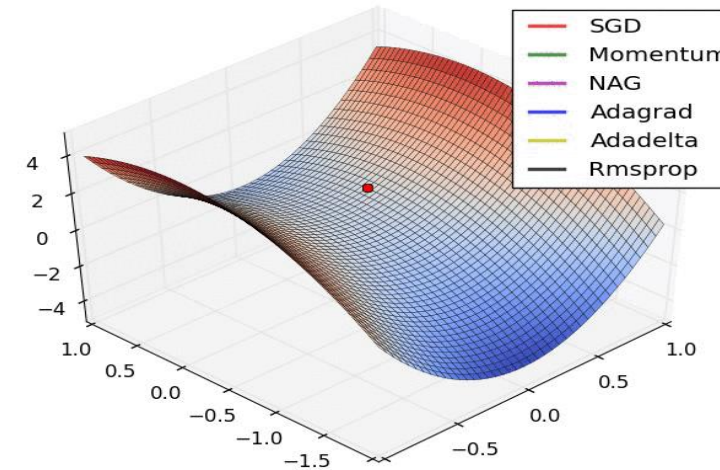
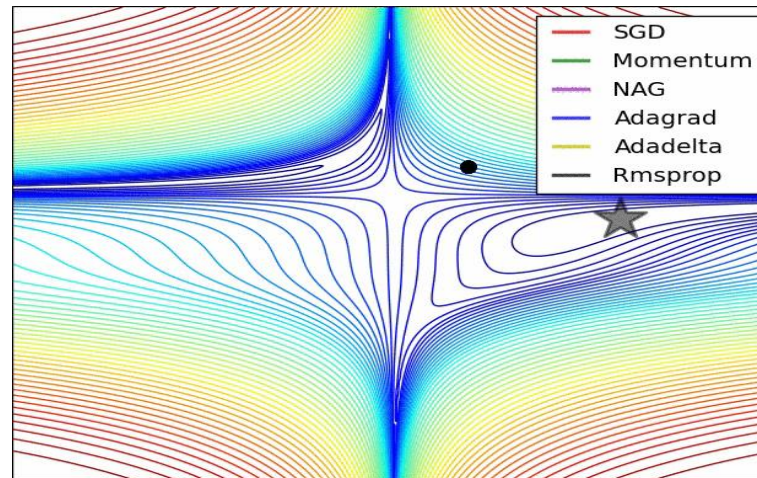
# Number of Weights



© 256 Inputs + 100 Hidden layers[connected to 256 inputs] -----Input side[25600+100(bias weights)]  
26 outputs + 100 Hidden layers[connected to 26 outputs]---- Output side[2600+26(bias weights)]  
Total 28236 weights are calculated



# Learning rate visualization



Animations that may help your intuitions about the learning process dynamics. **Left:** Contours of a loss surface and time evolution of different optimization algorithms. Notice the "overshooting" behavior of momentum-based methods, which make the optimization look like a ball rolling down the hill. **Right:** A visualization of a saddle point in the optimization landscape, where the curvature along different dimension has different signs (one dimension curves up and another down). Notice that SGD has a very hard time breaking symmetry and gets stuck on the top. Conversely, algorithms such as RMSprop will see very low gradients in the saddle direction. Due to the denominator term in the RMSprop update, this will increase the effective learning rate along this direction, helping RMSProp proceed.

# Training ANN

**STEP 1:** Randomly initialise the weights to small numbers close to 0 (but not 0).



**STEP 2:** Input the first observation of your dataset in the input layer, each feature in one input node.



**STEP 3:** Forward-Propagation: from left to right, the neurons are activated in a way that the impact of each neuron's activation is limited by the weights. Propagate the activations until getting the predicted result  $y$ .



**STEP 4:** Compare the predicted result to the actual result. Measure the generated error.



**STEP 5:** Back-Propagation: from right to left, the error is back-propagated. Update the weights according to how much they are responsible for the error. The learning rate decides by how much we update the weights.



**STEP 6:** Repeat Steps 1 to 5 and update the weights after each observation (Reinforcement Learning). Or:  
Repeat Steps 1 to 5 but update the weights only after a batch of observations (Batch Learning).



**STEP 7:** When the whole training set passed through the ANN, that makes an epoch. Redo more epochs.



# Top Deep Learning Frameworks

- **TensorFlow**- Google's open-source platform TensorFlow is perhaps the most popular tool for **Machine Learning** and **Deep Learning**.
- **PyTorch**- PyTorch is an open-source **Deep Learning framework** developed by Facebook.
- Keras - Tool can run on top of TensorFlow, Theano, Microsoft Cognitive Toolkit, and PlaidML

# TensorFlow Overview

# TensorFlow Architecture

Tensorflow architecture works in three parts:

- ✓ Preprocessing the data
- ✓ Build the model
- ✓ Train and estimate the model

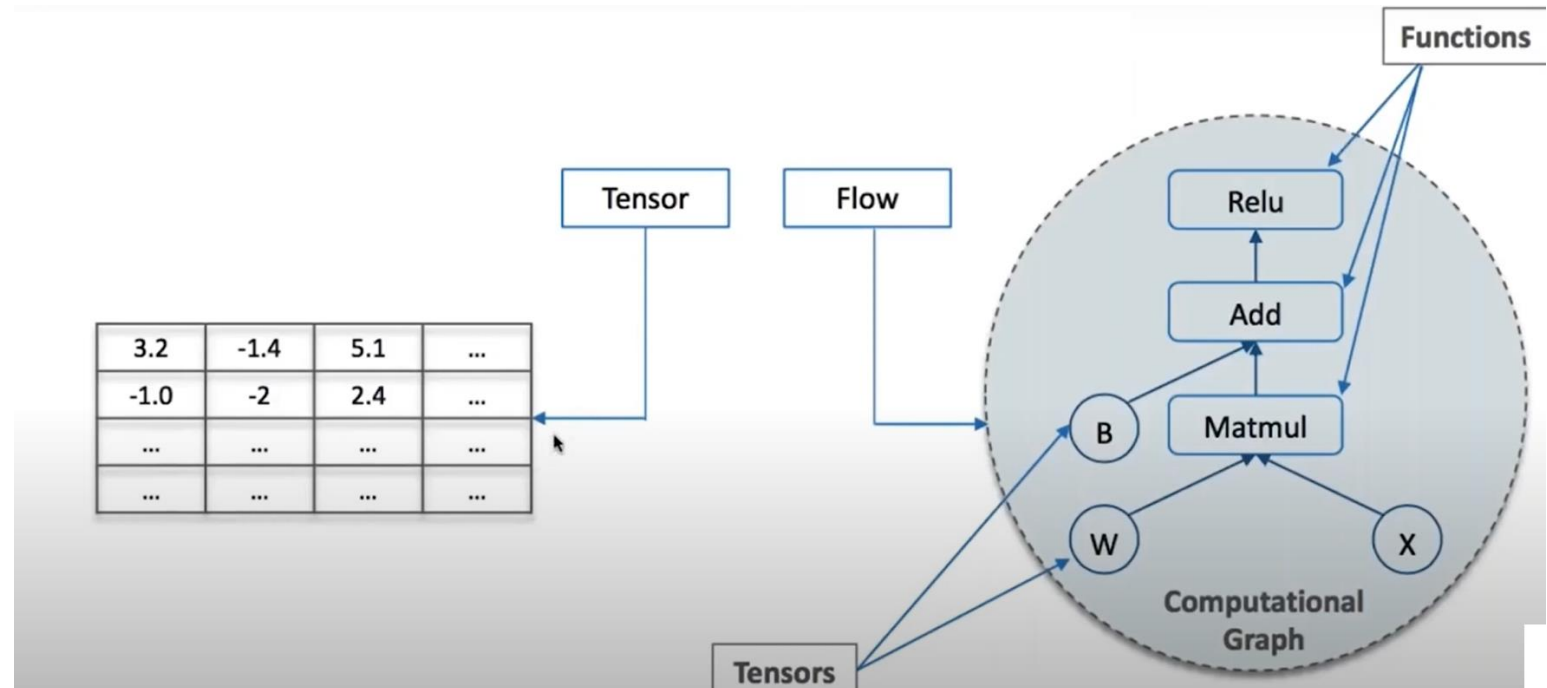
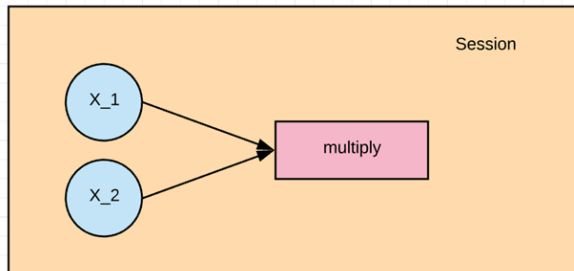
It is called Tensorflow because it takes input as a multi-dimensional array, also known as **tensors**.

Can construct a sort of **flowchart** of operations (called a Graph) that you want to perform on that input

# Components of TensorFlow

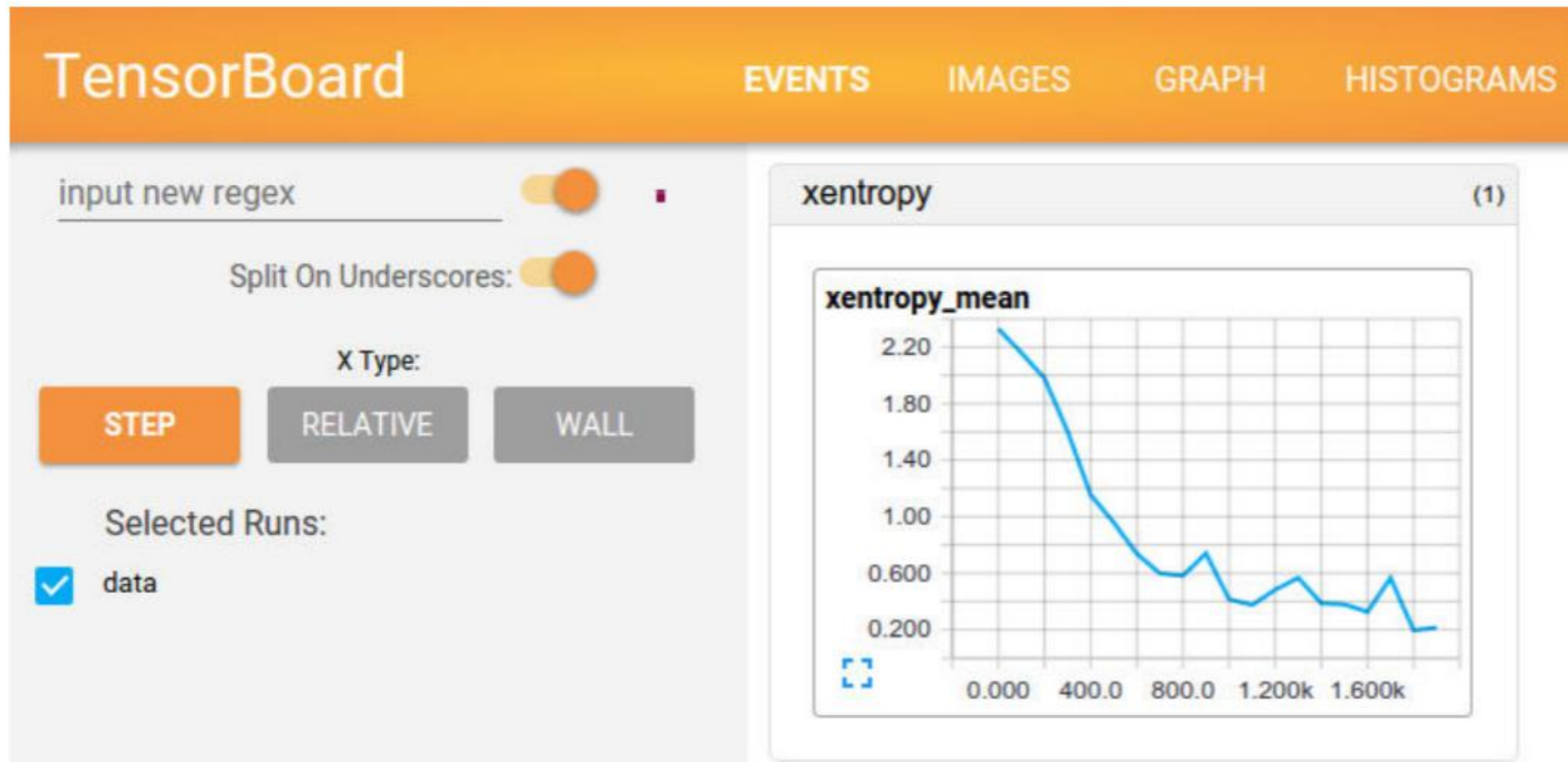
**Tensor** -input data or the result of a computation.

**Graphs**- The graph gathers and describes all the series computations done during the training.



# Tensor board

Tensor board Tensor Board is a suite of web applications for inspecting and understanding your TensorFlow runs and graphs. It can output the graph of the calculation and draw a chart of summary values which you log in the code.





## TensorFlow Overview

# Highlights of the Important changes from TF 1.4

## Eager Execution

TF 2.0 Default execution mode is based on Eager Execution which eliminates manual compilation

## Keras Fully Integrated

Keras API is fully integrated with TF 2.0 , to leverage quick model building and implementations. Includes support for building Data Pipelines , Estimators and Eager Execution

## Consolidation of Modules

Consolidation of APIs , removal redundant ones , made more productive , easy to use and Faster implementation

## Data Pipelines

Data API helps us to build complex input pipelines from various sources. Helps us to handle large volumes of data , formats etc.

# Core Modules and API of TensorFlow 2.0

**Core TensorFlow Modules and APIs for End to End Deep learning Model building Support**

**Keras**

- Easy to use TensorFlow API for quick prototyping of a Deep learning Model.
- With Latest integration and modification can be used for Production as well

**Data**

- Data API helps users to build Data Pipeline for training a Deep learning model

**Accelerators**

- Build a distribution strategy for Model training using TensorFlow Accelerators
- Train on multiple GPUs

**TF Hub**

- Leverage transfer learning using TF Hub , a library for reusable Machine Learning modules.

**TF Functions**

- Construct TensorFlow Graph using Functions Module



# Data Pipelines using TensorFlow 2.0

- Build Data pipeline using `tf.data`
- Design Complex input data pipeline and incorporate transformation Functions as part of it



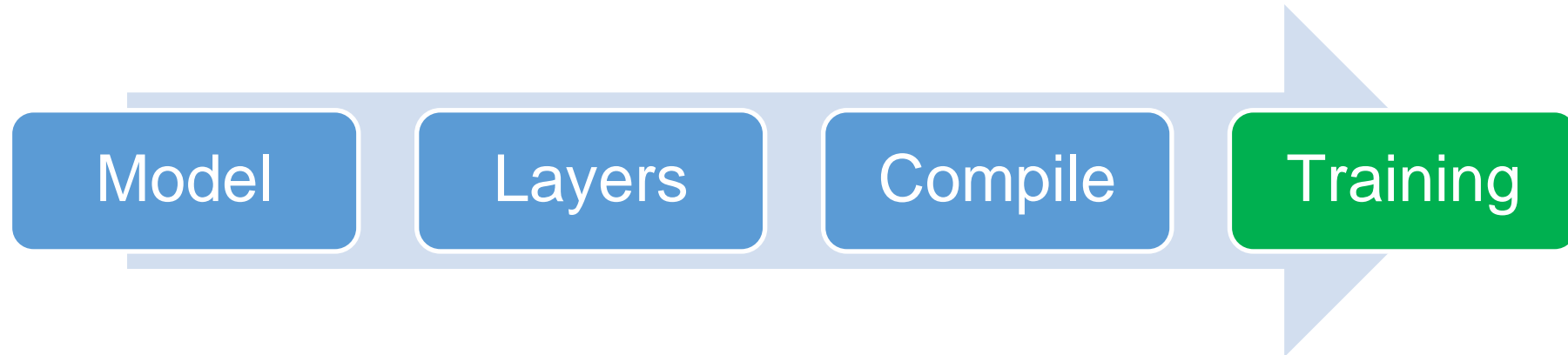
- Data Source

A data source constructs a Dataset from data stored in memory or in one or more files.

- API introduces a `tf.data.Dataset` abstraction that represents a sequence of elements, in which each element consists of one or more components.

- Transformed Data is used to train the model by sending the data in batches

# TensorFlow Keras



- Construct Overall Model structure
- A Framework For the model
- Configure layers based on the Deep learning model type
- Convolutions Layers , Recurrent Layers , Dense Layers etc.
- Assign a Optimizer and Loss functions
- Compile the model
- Train the model based on the number of Epoch Provided

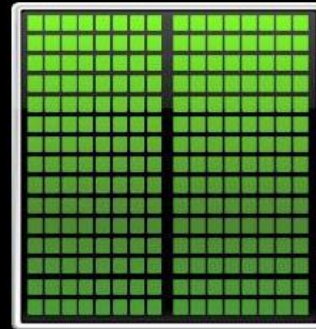
# CPU vs GPU vs TPU

## *The Difference between a CPU and GPU*



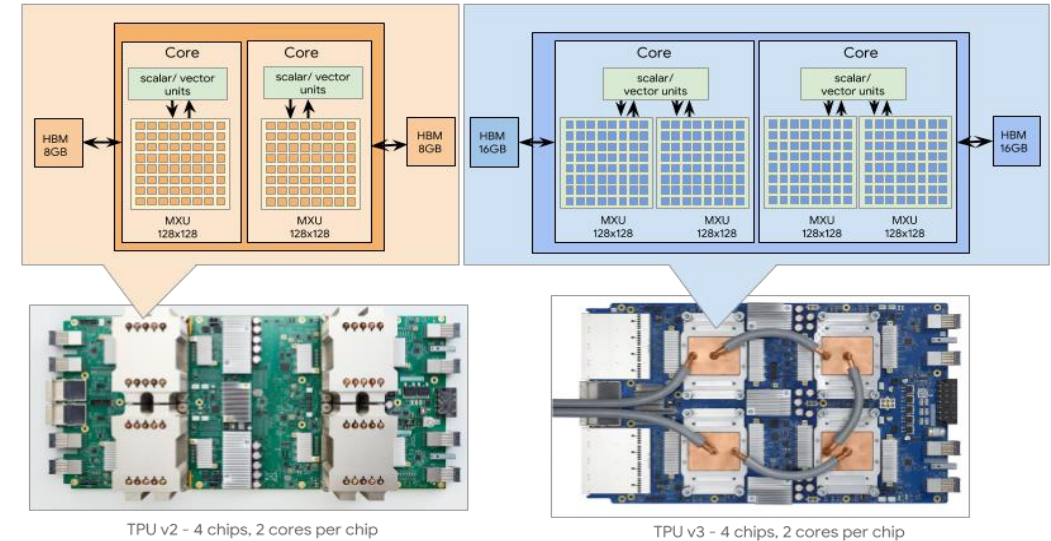
**CPU**

MULTIPLE CORES



**GPU**

THOUSAND OF CORES



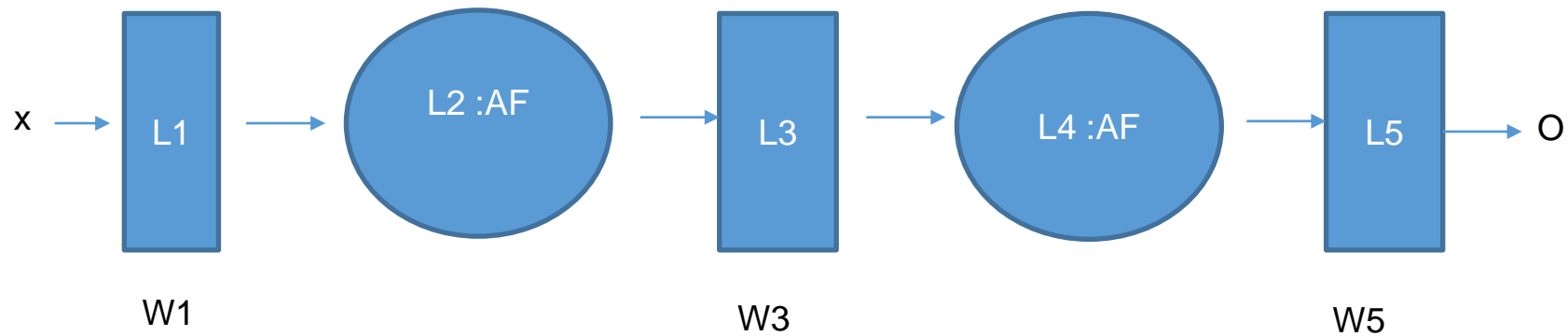
TPU v2 - 4 chips, 2 cores per chip

TPU v3 - 4 chips, 2 cores per chip

# Agenda:

- Feed forward
- Back propagation
- Fully connected layer – forward pass
- Fully connected layer – backward pass
- Activation functions
- Activation functions in practice
- Softmax
- Cross entropy
- Hands on MNIST- with all building blocks explained in detail

# Feed forward

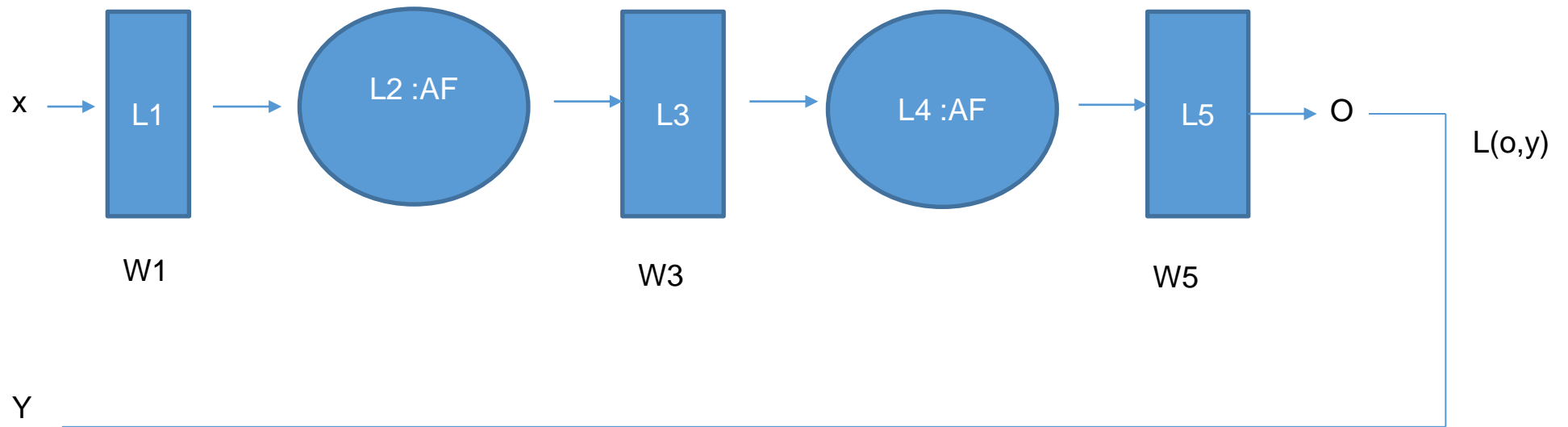


X- input, L1 – Layer 1, Layer 2- Activation function outputs of all layers are  $a_1, a_2, a_3$  and  $a_4$

# Feed forward

- $a_1 = F(x, W_1), x \in \mathbb{R}$
- $a_2 = G(a_1)$
- $a_3 = H(a_2, W_3),$
- $a_4 = J(a_3)$
- $o = K(a_4, W_5) = K\left(J\left(H\left(G\left(F(x, W_1)\right), W_3\right)\right), W_5\right) \in \mathbb{R}$

# Back propagation



# FCN- Forward

- Neurons have connections to all activations of the previous layer
- Number of connections add up very quickly due to all combinations
- Forward pass – one matrix multiplication followed by bias offset and activation function



# FCL – Backward pass

- We have gradients coming from next layer called GradInput
- Size is always equal to the size of output
- We will calculate the gradients which we need to pass to previous layers for continuing the chain rule
- We will also calculate the gradients that we will be using for updating or weights

# Activation functions

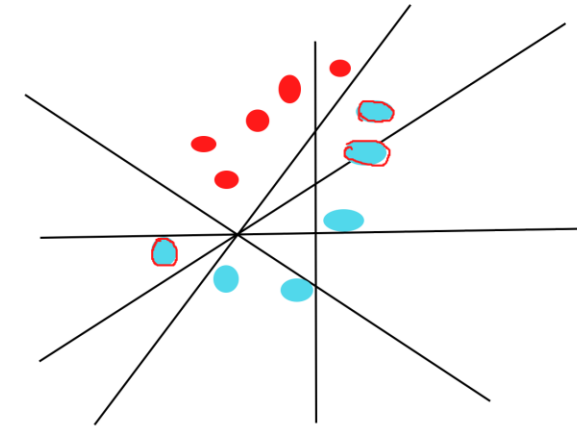
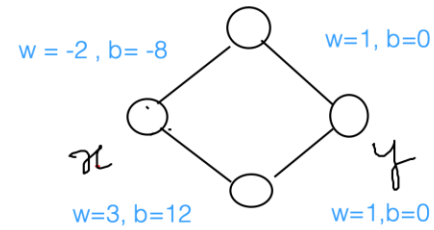
- An activation function takes a single input value and applies a function to it to add non linearity
- Why non-linear? Converts linear into non-linear for having a non-linear decision boundary
- Afs can decide which neuron is switch ON ,it acts as gate

# AF in practise

- Use Relu , monitor the dead neurons
- Never use sigmoid
- Try Leaky Relu
- Tanh
  - Not sure if it performs better!

How relu introduces non-linearity

How relu is  
introducing non  
linearity?



equation for  $(-2 \cdot x) - 8, 3x + 12, x + 4$

# Vanishing gradient

- If we look at the formula for updating the weights of  $W_3$  or  $W_1$  it's a huge multiplications of Gradient with Jacobians
  - If all values are less than 1 we have vanishing gradient problem
  - If all values are greater than one we have exploding gradient problem

# softmax

- Softmax is multinomial logistic classifier , it can handle multiple classes
- Softmax is typically the last layer of NN
- Softmax itself is an activation function , hence it need not be combined with any activation function

# Cross-entropy loss

- CE loss also called log loss quantifies the deviation between predicted output and ground truth
- Perfect prediction would have zero loss
- With Gradient Descent , we will try to reduce the error

# Neural Networks construction



# NN Constructed

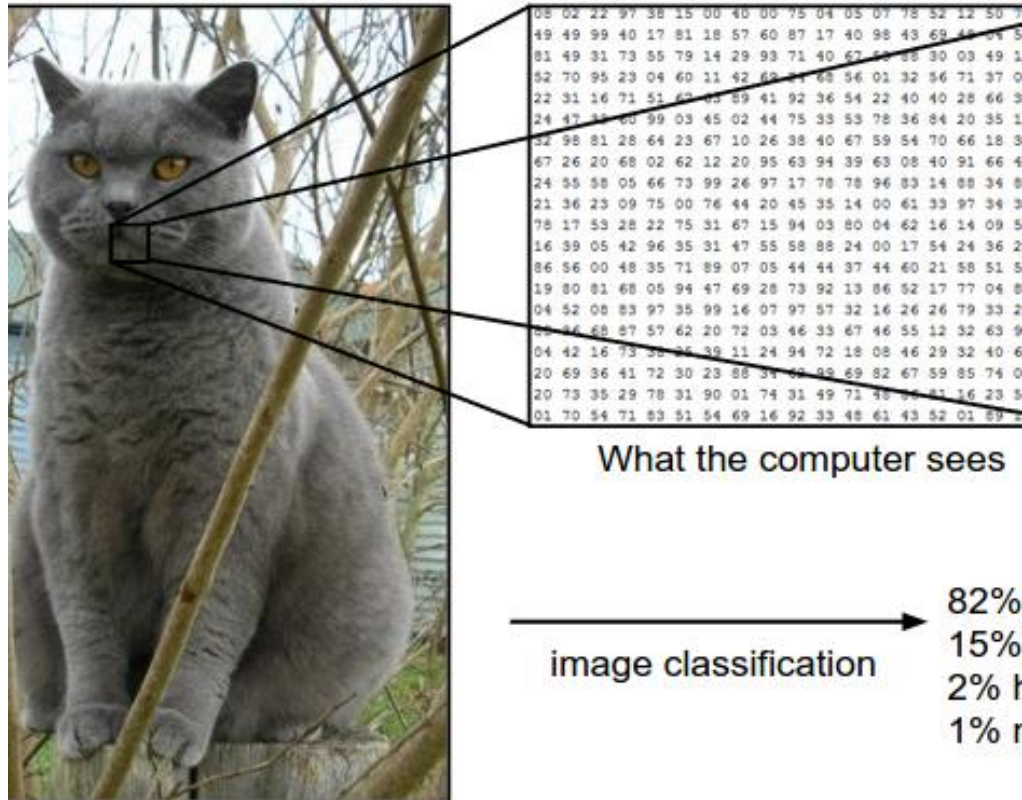
- Data preprocessing
- Data Augmentation
- Weight Initialization
- Regularization
  - Implement all these in already build network

# Data preprocessing

- Data Normalization
  - How to do it?

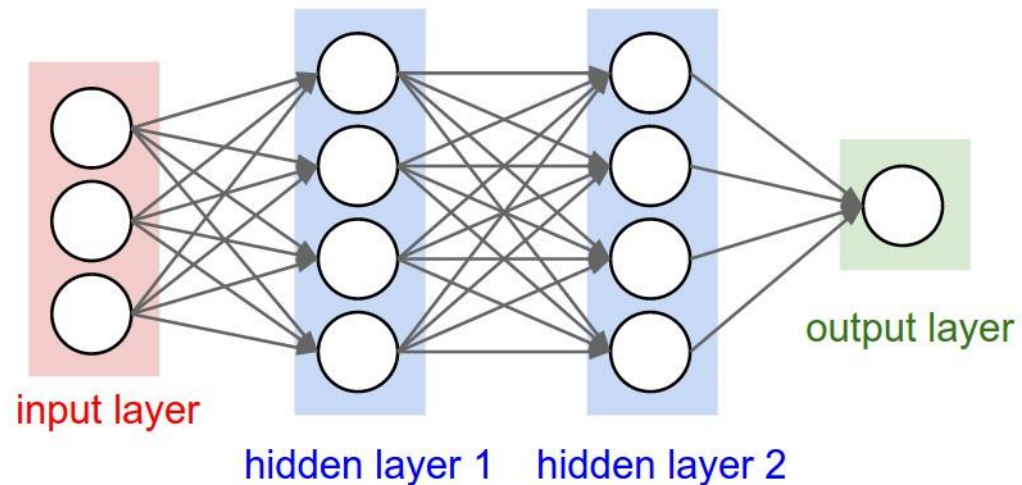
# Data Augmentation

- Rotate
- Grey scale
- Adding random noise
- Horizontal flip
- Color jitter
- Ideal way of doing it , doing all these transformations, since for each epoch do some random transformation for the image



- If we slightly move the image, the pixels are changed and it altogether sees a different image

# Weight initialization



- Lets say we have 3 input, 4 hidden and 1 output neurons.
- $3 * 12$  etc.,
- And  $4 * 4$
- What if I initialize with zero?
  - Not a good idea, because output will always move in same direction

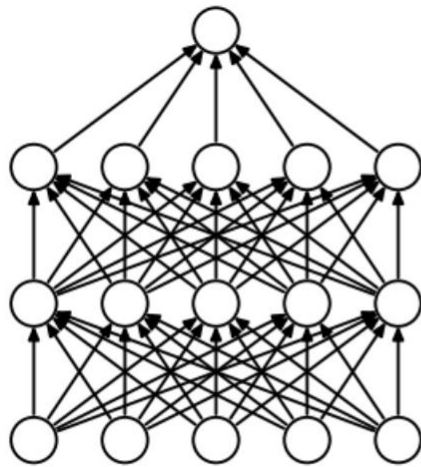
# WI

- Simple random numbers (Normal distribution -0.01 to + 0.01)
  - $W = \text{np.random.randn}(\text{in}, \text{out}) * 0.01$
  - $B = \text{np.random.randn}(\text{out}) * 0.01$
- Xavier Initialization:
  - $W = \text{np.random.randn}(\text{in}, \text{out}) / \text{math.sqrt}(\text{in})$
  - This is implemented in most of the latest ones
- Idea: Take the random numbers and scale it based on the input neurons

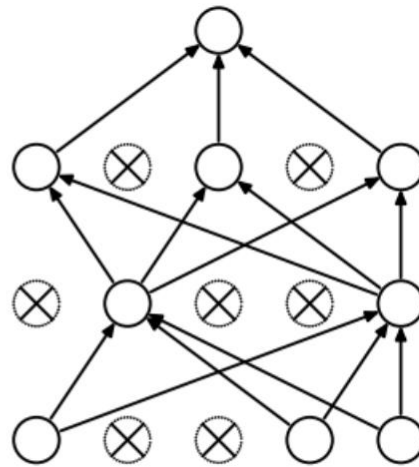
# Regularization ( batch normalization)

- In weight init, we want our activations should be unit gaussians, hence we initialized the weights similarly
- Usually inserted after all fully connected layers, before the non-linearity
- Affine transformation
- BN:
  - Improves the gradient flow
  - Allows higher learning rates
  - Reduces strong dependency on initialization

# Drop out



(a) Standard Neural Net



(b) After applying dropout.

- Dropping out is given as per probability, if  $p=0.5$ , 2 are dropped in one layer and 3 in next layer (total 5 in each layer)
- We will see how to do that in coding



# Dropout

- It forces the network to have redundant representation
  - Has a ear X
  - Has a tail
  - Is fully X
  - Has claws
  - Mischievous looks X
- Predicting the same as CAT

- Dropout is training a large ensemble of models
- Each binary mask is one model ,gets trained on only one batch
- At test time, all neurons are ON
- We must scale the activations for each neuron so that output at testtime == expected output at traintime
  - What does this mean?

- Implement the same on MNIST Data

THANK  
YOU

