Introduction to Neural Networks

What are artificial neural networks (ANN's)?

- Neural networks or ANN's are modeled based on the human brain for recognizing patterns in data as done by humans using sensory information. Before data can be analyzed by the computer it has to be labelled.

- Neural network are a group of neurons organized in layers. Each neuron is a mathematical operation that takes it's input, multiplies it by it's weights and then passes the sum through the activation function to the other neurons. Neural network learns how to classify an input through adjusting it's weights based on previous examples.

What are artificial neural networks (ANN's)?

- There are various artificial intelligence (AI) algorithms such as linear regression, logistic regression, k-nearest neighbors, support vector machines (SVM) and deep learning/ deep neural networks (DNN).

- DNN are most popularly used as they have spurred advances in text and speech recognition, computer vision and OCR, as well as empowering reinforced learning and robotic movement, along with other applications.

Applications of ANN's

Supervised Learning: It depends upon labeled datasets; where humans annotate the dataset in order for a neural network to establish a relationship with the data.

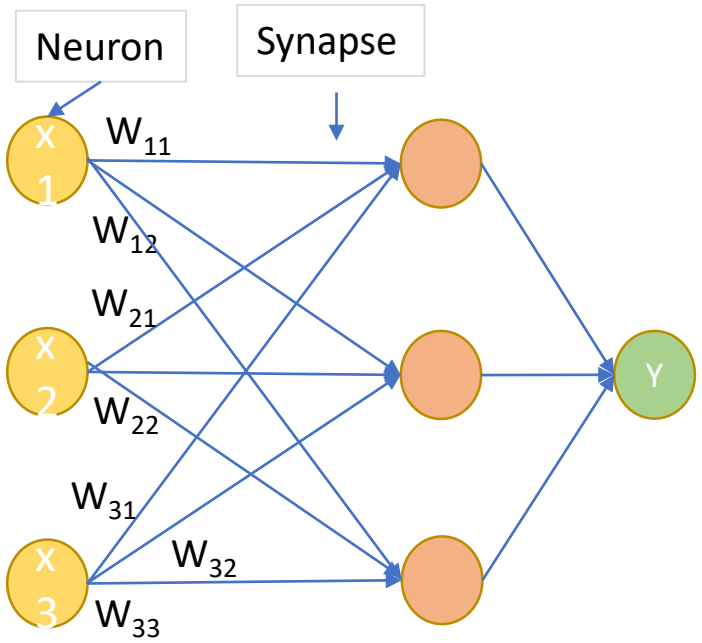
- Face recognition, emotion detection
- Object detection (cars, stop signs, pedestrians, lane lines etc.)
- Voice recognition, speech to text, identifying songs
- Classify email as spam or fraudulent
- Sentiment classification based on text

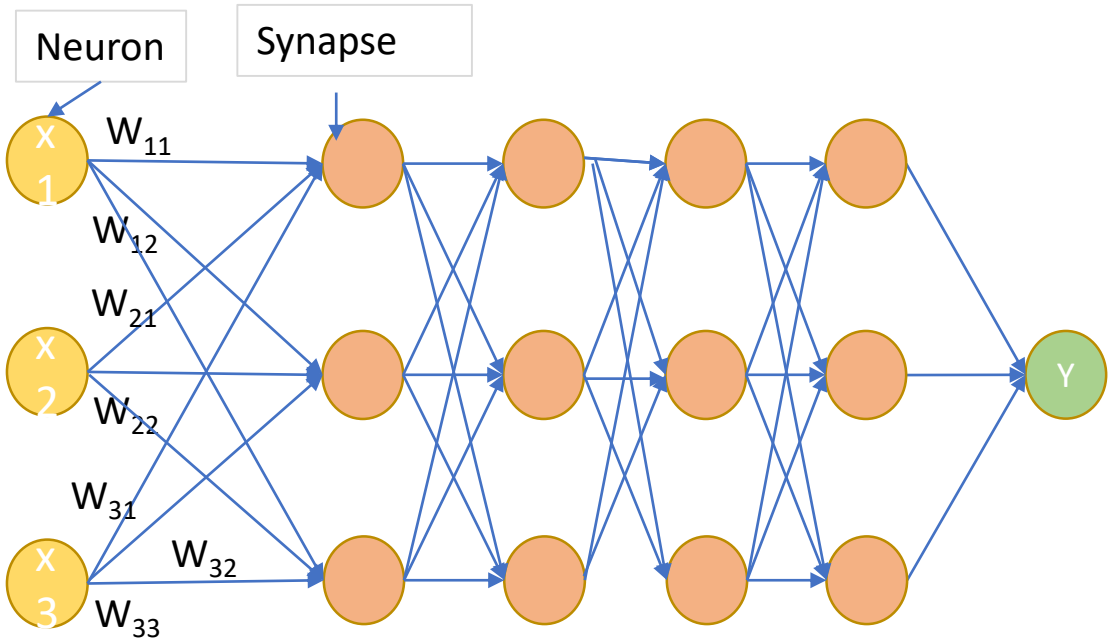
Applications of ANN's

Unsupervised Learning: When labeled data is not present, which is the case for most of the data.

- Anomaly detection: Identifying unusual patterns to prevent fraud
- Clustering automatically by splitting the data into groups base on their similarities
- Data associations by identifying features/ item that reoccur

How does a neural network look like?





Input Layer (Layer 0)

Hidden Layer (Layer 1)

Output Layer

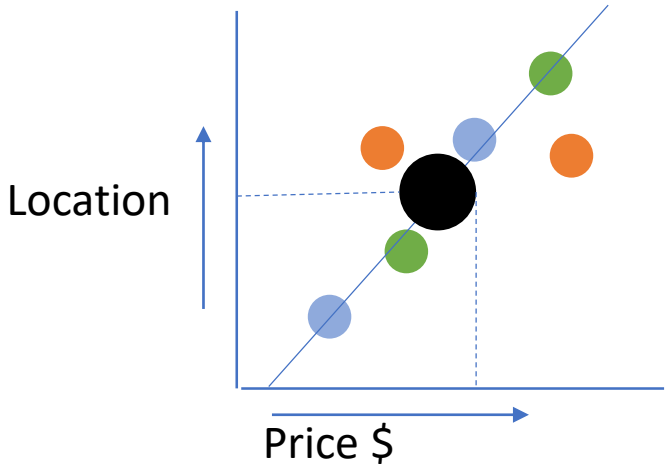
Input Layer (Layer 0)

Multiple Hidden Layers (Layers 1-4)

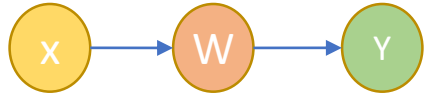
Output Layer

Neural network-> Single Feature

Housing Problem

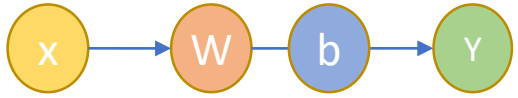


$$Y = W * X + b$$



Input
Location

Predicted
Price $



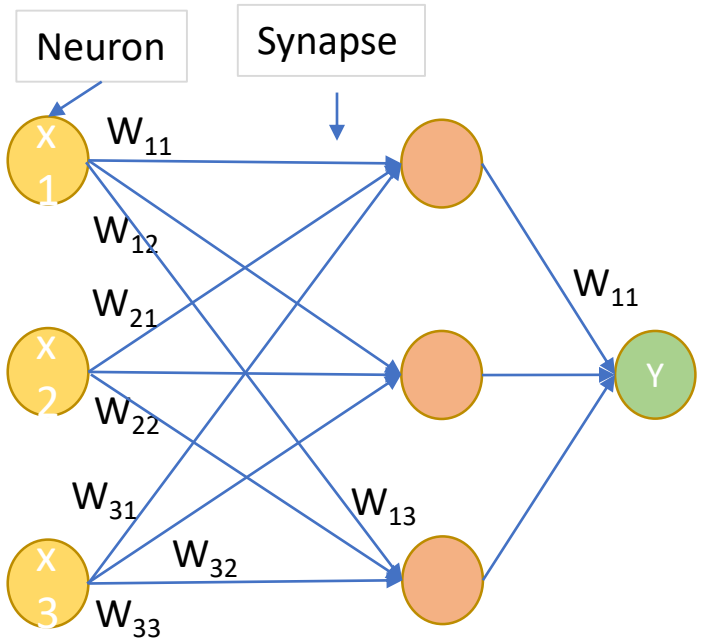
Input
Location

Predicted
Price $

How does a neural network look like?

Features
Location
Area (Sq. Footage)
School District (Ranking)

$$\begin{pmatrix} a1 \\ a2 \\ a3 \end{pmatrix} = \sigma\left(\begin{pmatrix} .8 & .5 & .6 \\ .1 & .2 & .3 \\ .4 & .5 & .7 \end{pmatrix} \begin{pmatrix} .7 \\ .4 \\ .5 \end{pmatrix} + \begin{pmatrix} 5 \\ 4 \\ 3 \end{pmatrix} \right)$$



Input Features (Layer 0)

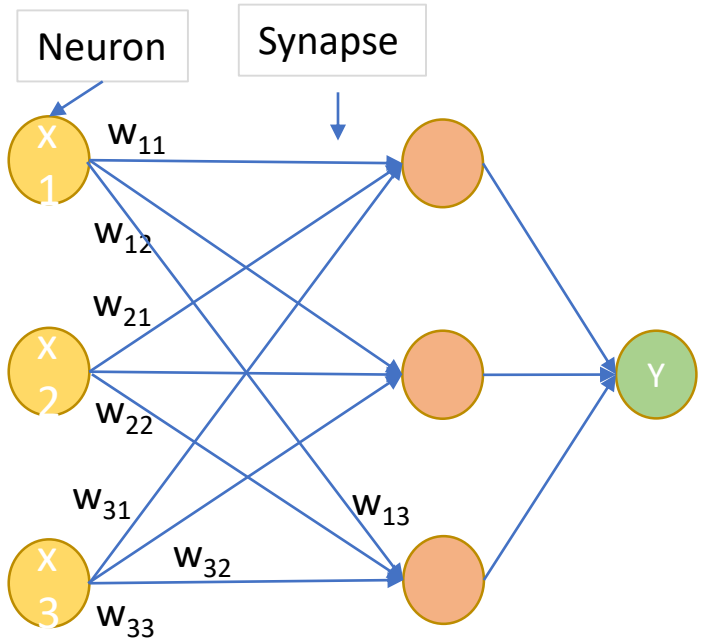
Hidden Layer (Layer 1)

Output ->Price

How does a neural network look like?

Features
Location
Area (Sq. Footage)
School District (Ranking)

$$\begin{bmatrix} a1 \\ a2 \\ a3 \end{bmatrix} = \sigma \left(\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} + \begin{bmatrix} b1 \\ b2 \\ b3 \end{bmatrix} \right)$$



$Z^{[1]}=W^{[1]}X+B^{[1]}$

$A^{[1]} = \sigma(Z^{[1]})$

$A^{[1]} = \sigma(W^{[1]}X+B^{[1]})$

Layer1

$Z^{[2]}=W^{[2]}A^{[1]}+B^{[2]}$

$A^{[2]} = \sigma(Z^{[2]})$

Layer2

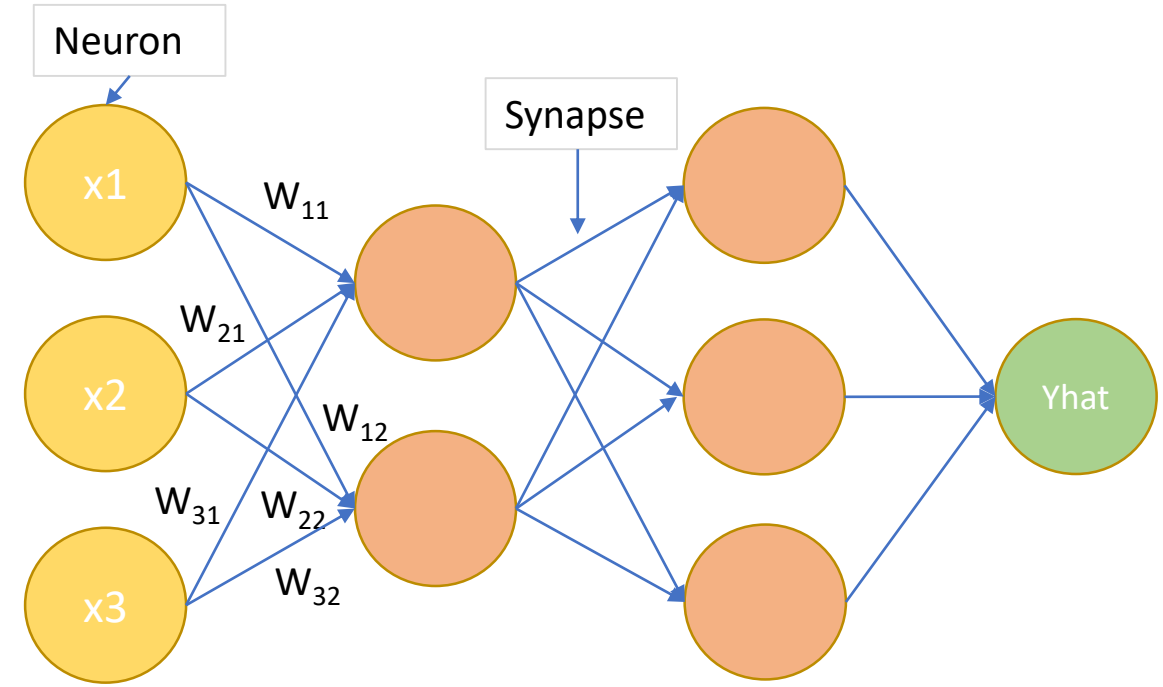
Input Features (Layer 0)

Hidden Layer (Layer 1)

Output ->Price

Note: Generally final activation used is a softmax to squeeze the output in to a probability distribution between 0 and 1.

Forward Propagation: General Equation



Layer1:

$Z^{[1]}=W^{[1]}X+B^{[1]}$

$A^{[1]}= \sigma(Z^{[1]})$

Layer2:

$Z^{[2]}=W^{[2]}A^{[1]}+B^{[2]}$

$A^{[2]}= \sigma(Z^{[2]})$

Output Layer:

$Z^{[3]}=W^{[3]}A^{[2]}+B^{[3]}$

$A^{[3]}= \sigma(Z^{[3]})$

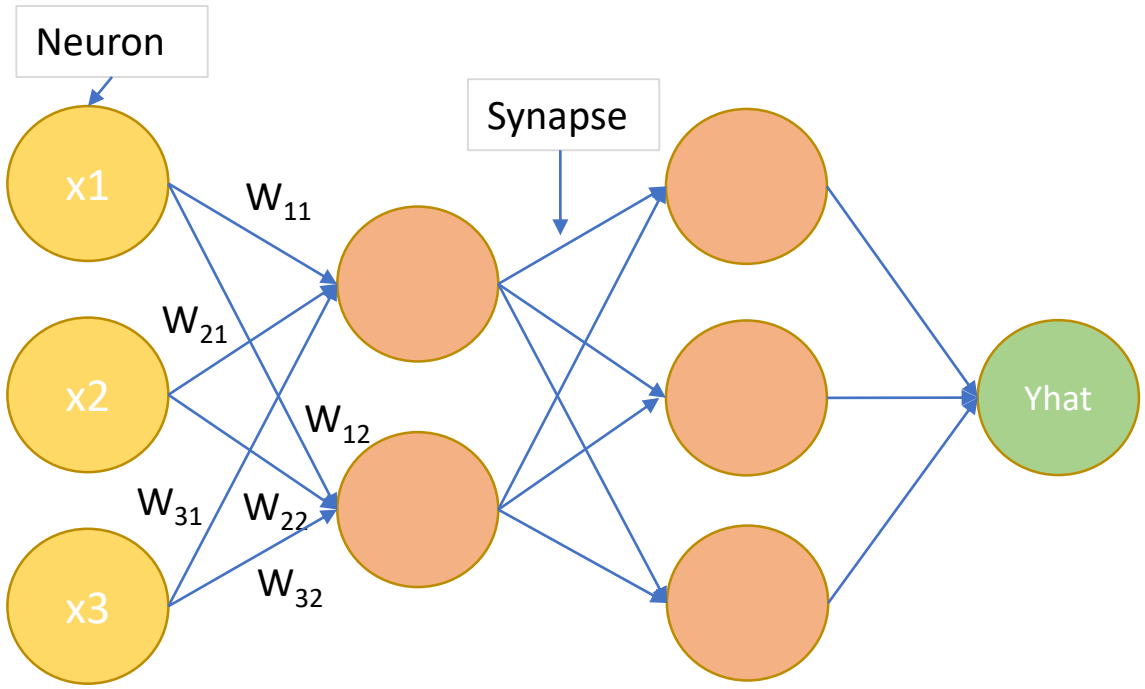
General Equation:

$Z^{[i]}=W^{[i]}A^{[i-1]}+B^{[i]}$

$A^{[i]}=F^{[i]}(Z^{[i]})$

$W^{[i]}=(n^{[i]},n^{[i-1]})$; $B^{[i]}=(n^{[i]},1)$

Forward Propagation



Input Layer (Layer 0)

Hidden Layer (Layer 1)

Hidden Layer (Layer 2)

Outer Layer (Layer 3)

Layer1:

$Z^{[1]}=W^{[1]}X+B^{[1]}$; [2,1]=[2,3].[3,1]+[2,1]

$A^{[1]}= \sigma(Z^{[1]})$

Layer2:

$Z^{[2]}=W^{[2]}A^{[1]}+B^{[2]}$; [3,1]=[3,2].[2,1]+[3,1]

$A^{[2]}= \sigma(Z^{[2]})$

Output Layer:

$Z^{[3]}=W^{[3]}A^{[2]}+b^{[3]}$; [1,1]=[1,3].[3,1]+[1,1]

$A^{[3]}= \sigma(Z^{[3]})$

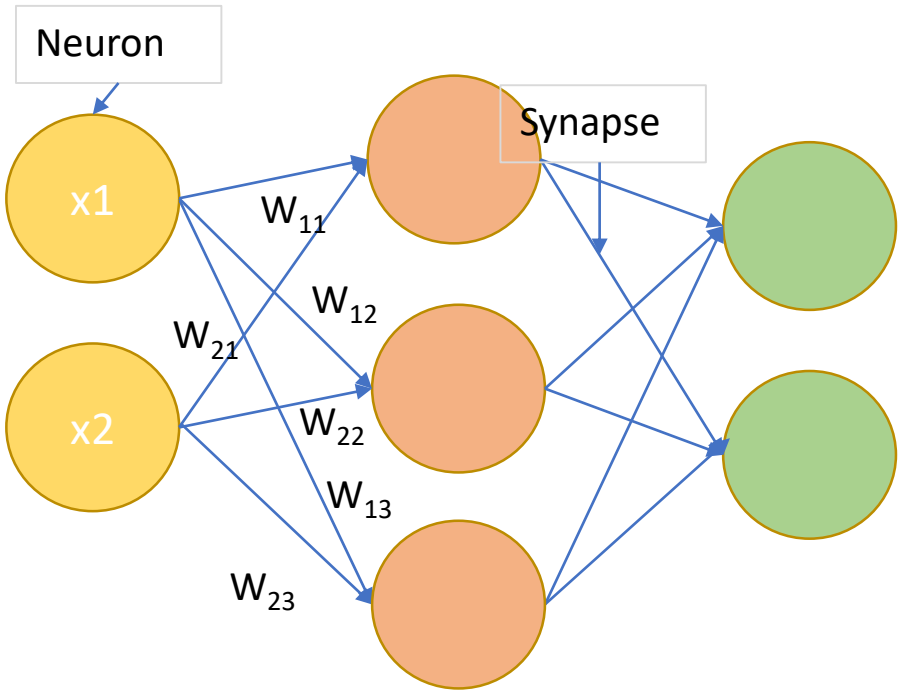
General Equation:

$Z^{[i]}=W^{[i]}A^{[i-1]}+B^{[i]}$;

$W^{[i]}=(n^{[i]},n^{[i-1]})$; $B^{[i]}=(n^{[i]},1)$

$A^{[i]}=F^{[i]}(Z^{[i]})$

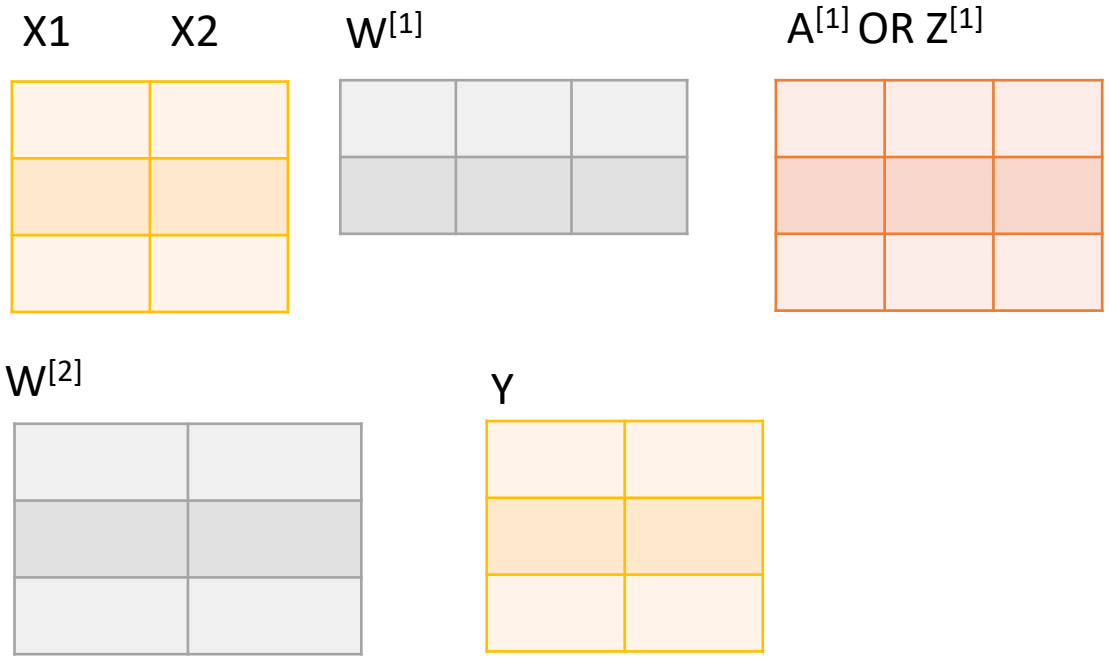
Forward Propagation – Matrix Dimensions



Input Layer (Layer 0)

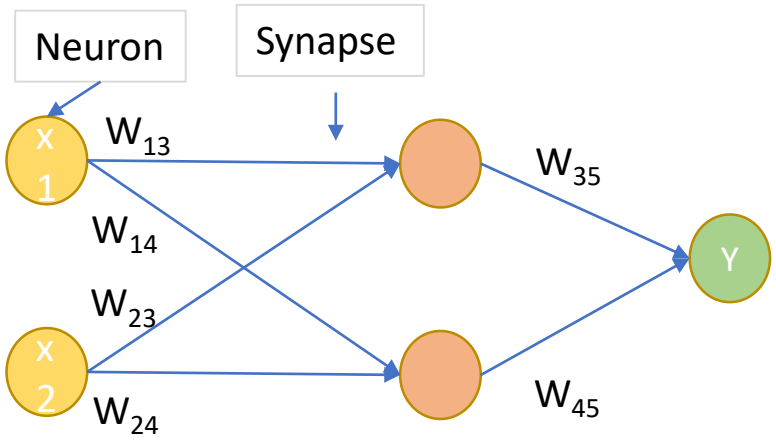
Hidden Layer (Layer 1)

Outer Layer (Layer 2)



Input	[Features, Input Neurons]
Weights	[Neurons in previous layer, Neurons in present layer]
Bias	[1, Neurons in present layer]
Output Weights	[Hidden layer neurons, Neurons in output layer]

Practice Problem: Solve the neural network



Input, x1	**1**
Input, x2	0
w13	2
w14	1
w23	-3
w24	4
w35	2
w45	-1

$f(x) = 1$ if x>=0
else $f(x) = 0$

Note: Real world problems are often more complicated as there are more unknowns (no weight information) than known values(only inputs and outputs are known).
This is where the term training comes in to play to identify the appropriate weights to achieve the desired output.

Discussion so far-> Neural network layers

Input Layer : Provides information from the outside world to the network, no computation is performed at this layer, nodes here just pass on the information(features) to the hidden layer.

Hidden Layer : Nodes of this abstraction layer are not exposed to the outer world. Hidden layer performs computation on the features entered through the input layer and transfer the result to the output layer.

Output Layer : This layer brings up the information learned by the network to the outer world.

Discussion so far-> Activation Function

They are used to determine the output of neural network like yes or no by squishing the output in a range of values. Range can lie in between 0 to 1 or -1 to 1 etc. depending upon the function selected.

Activation Functions can be basically divided into 2 types:

- Linear Activation Function
- Non-linear Activation Functions

So what are neurons? -> Linear Activation Function

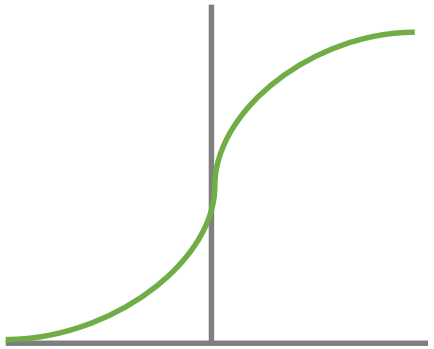
Linear Activation Function: Equation : Linear function has the equation similar to as of a straight line i.e. y = ax

No matter how many layers we have, if all are linear in nature, the final activation function of last layer is nothing but just a linear constant function of the input of first layer. Therefore, it is not used for training ANN's or DNN's.

Example: For linear regression of calculation of price of a house, linear activation is applied at output layer but non-linear activation function is used for the hidden layers.

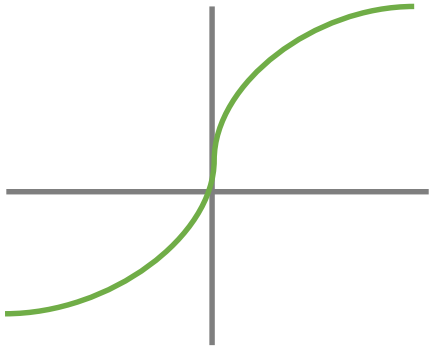
Non-Linear Activation Functions

Sigmoid



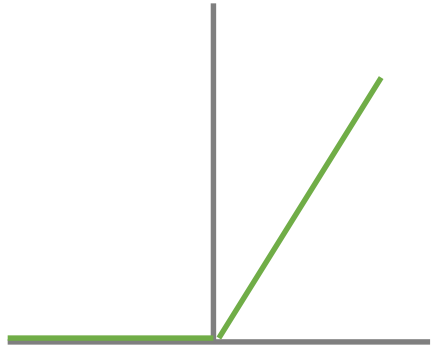
$$f(x) = \frac{1}{1+e^{-(x)}}$$

Tanh



$$\tanh(x) = \frac{e^{(x)} - e^{-(x)}}{e^{(x)} + e^{-(x)}}$$

ReLU



$$relu(x) = \max(0, x)$$

- A neural network without an activation function is just a linear regression model.
- The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.

Sigmoid-> Non-Linear Activation Function

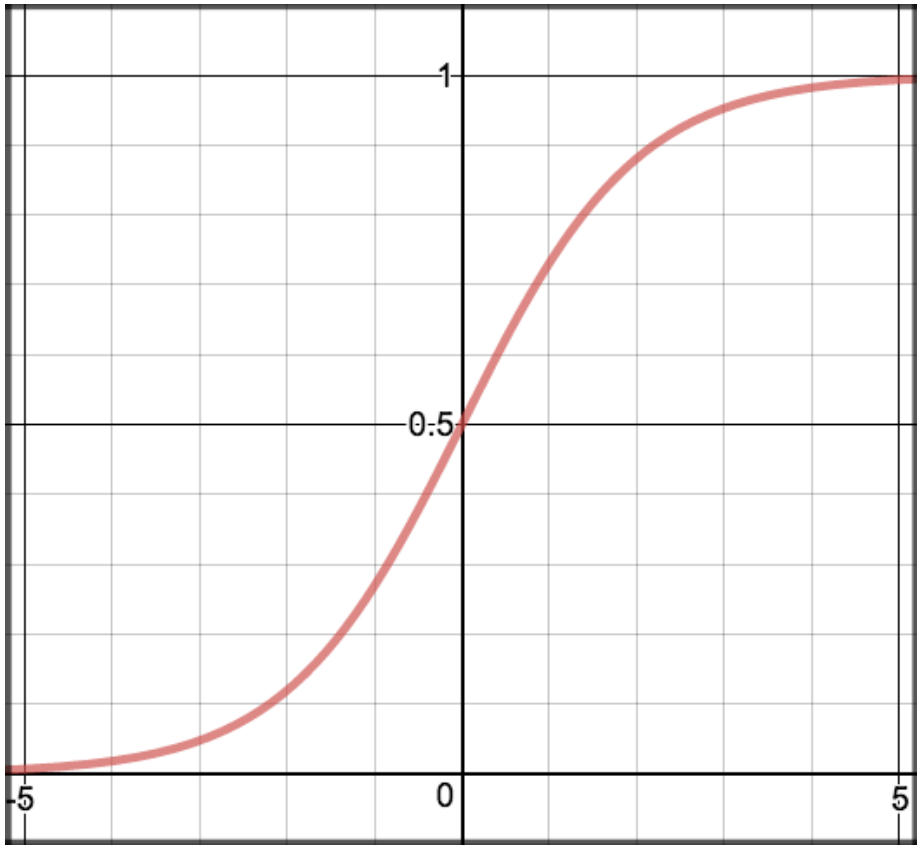
Small changes in x would also bring about large changes in the value of y.

Value Range : 0 to 1

Uses : Usually used in output layer of a binary classification, where result is either 0 or 1, result can be predicted easily to be 1 if value is greater than 0.5 and 0 otherwise.

Disadvantages: Extremely slow learning near the ends; y changes very slowly with changes in x.

It can lead to vanishing/ exploding gradient.

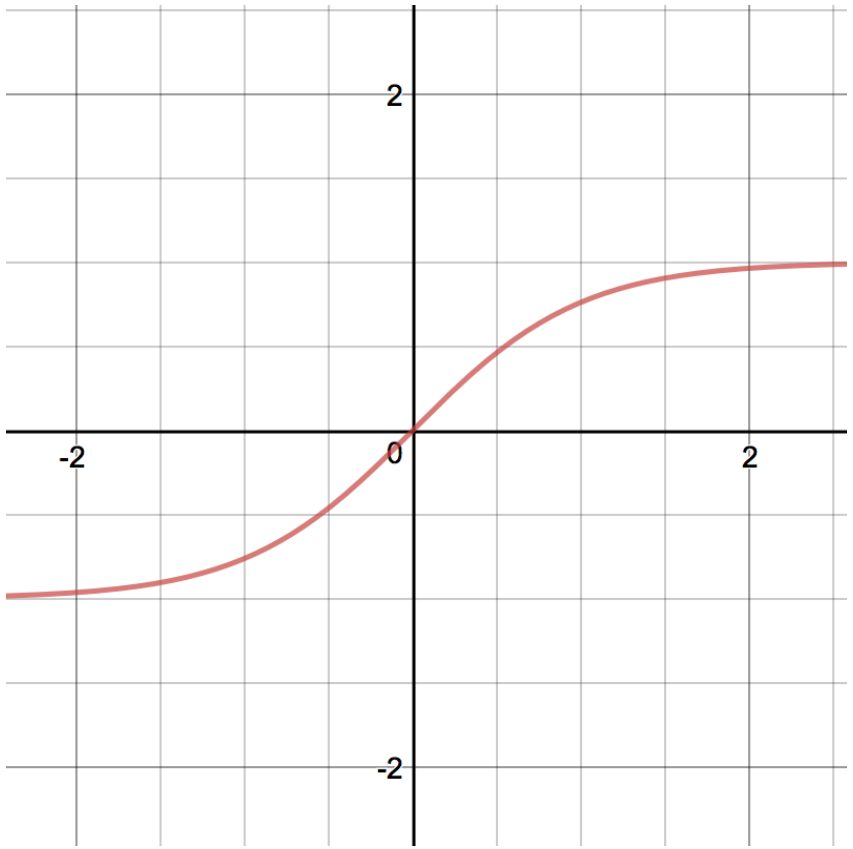


So what are neurons? -> Non-Linear Activation Function

Tanh function: Mathematically shifted version of the sigmoid function and can be derived from each other.

Value Range :- -1 to +1

Uses : Usually used in hidden layers of a neural network as it’s values lies between -1 to 1 hence the mean for the hidden layer comes out be close to 0 which helps in centering the data . This makes learning for the next layer much easier.

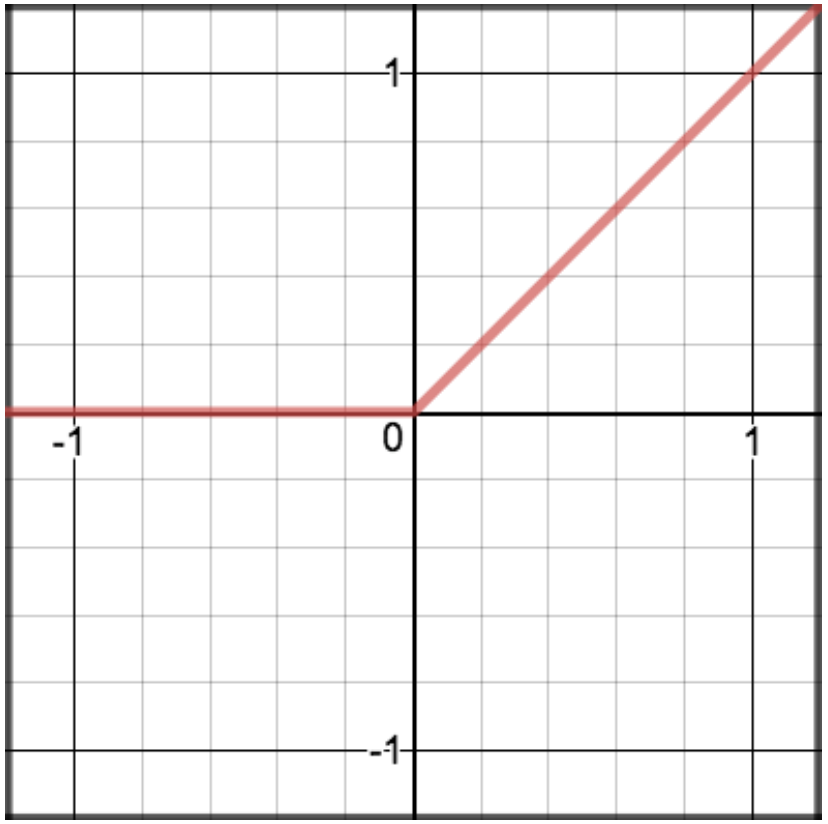


So what are neurons? -> Non-Linear Activation Function

RELU :- Rectified linear unit function is the most widely used activation function. Value Range :- [0, inf)

Uses :- ReLU is cheaper than tanh and sigmoid since it involves simpler mathematical operations as it activates only a few neurons.

Disadvantages: For activations in the region (x<0), gradient will be 0 for which the weights will not get adjusted during descent which leads to neurons not responding to variations in error/ input.

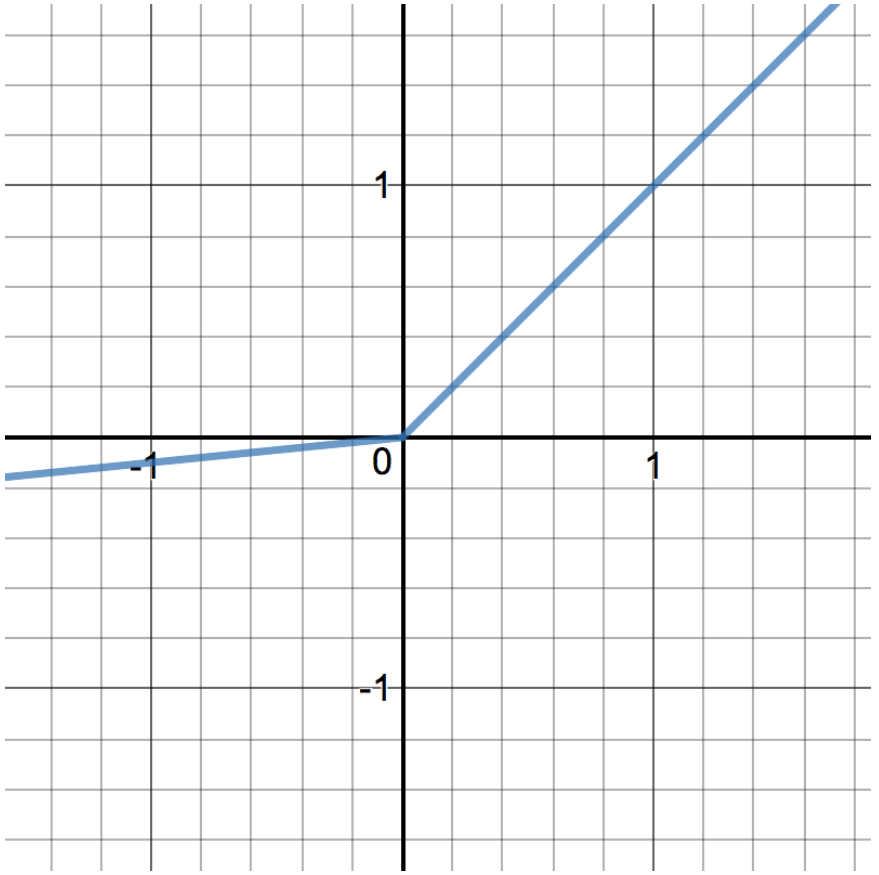


So what are neurons? -> Non-Linear Activation Function

Leaky RELU : Improved version of the ReLU function since in ReLU, the gradient is 0 for x<0, which made the neurons die for activations in that region. Leaky ReLU is defined as a function where value is a small linear component of x for x less than 0.

Value Range :- [az, inf), a=small constant

Uses :- Fixes the dying ReLU problem by having a small negative slope (of 0.01, or so).



Non-Linear Activation Functions->Softmax

- Softmax Function : It is also a type of sigmoid function but useful for classification problems.
- Uses : To handle multiple classes, the softmax function would squeeze the outputs for each class between 0 and 1 by dividing by the sum of the outputs.
- Ouput:- Ideally used in the output layer of the classifier where we are actually trying to attain the probabilities to define the class of each input.

Activation Function Selection

- Sigmoid functions and their combinations generally work better in the case of classifiers
- Sigmoids 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
- Softmax is used for classifying different classes by defining them in terms of probabilities.

So what are neurons? -> Activation Function Selection

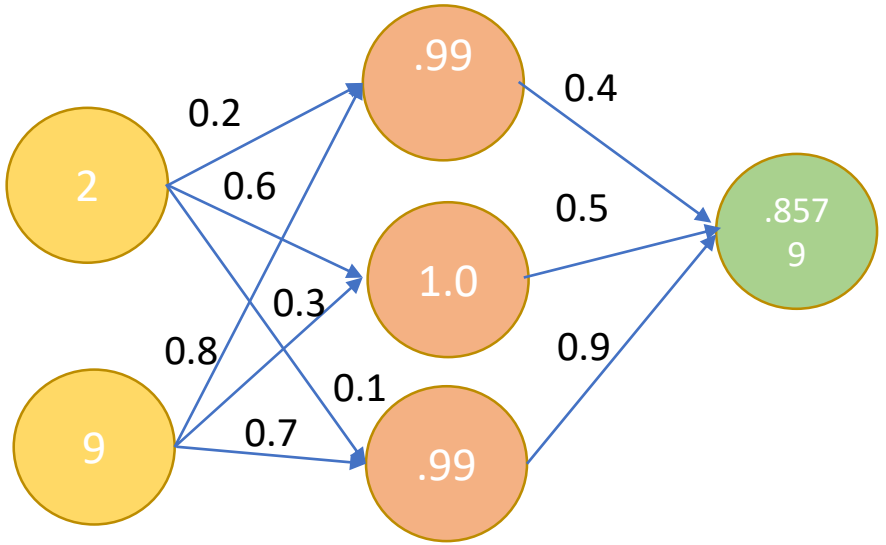
- If we encounter a case of dead neurons in our networks the leaky ReLU function is the best choice
- Always keep in mind that ReLU function should only be used in the hidden layers
- As a rule of thumb, you can begin with using ReLU function and then move over to other activation functions in case ReLU doesn’t provide with optimum results

Vanishing/ Exploding Gradient Problem

- DNN's when run through backpropagation a common problem with using sigmoid activation function is the vanishing gradient
- Training times scale up tremendously and accuracy decreases
- Cost is calculated by difference between the NN's predicted output and the actual output
- Cost is minimized by adjusting weights and biases by training
- Training requires gradient which is the change in cost with respect to a change in weight or a bias
- Gradients are smaller in the initial layers where simple patterns can be learned y the NN as compared to a later layer
- During back propagation, each gradient is a derivative of previous gradients and hence they become flatter and flatter, and eventually they get vanished.

Forward Propagation: Example

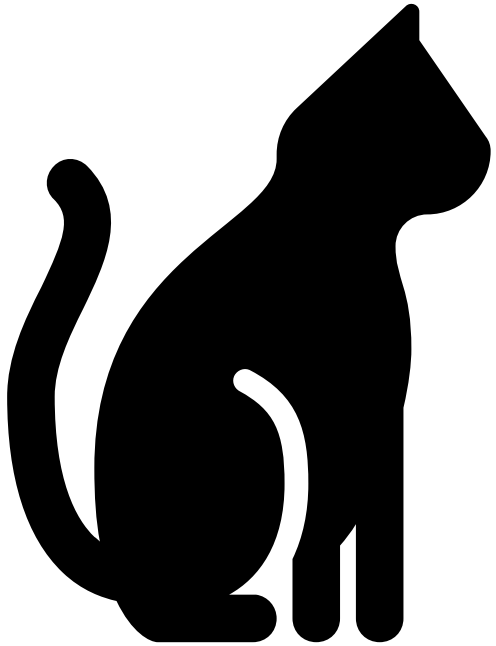
Time walked (hours), Amount Eaten (grams)	Cat's Fat Gain (grams)
2,9	92
1,5	86
3,6	89
5,10	?



First row from the data and it's predicted output

Summary/ Overview of Forward Propagation

- Takes inputs as a matrix (2D array of numbers)
- Dot product of the input by weights
- Applies an activation function
- Returns an output
- Error is calculated by taking the difference from the desired output from the data and the predicted output. This creates our gradient descent, which we can use to alter the weights
- The weights are then altered slightly according to the error.
- To train, this process is repeated 1,000+ times to achieve better accuracy



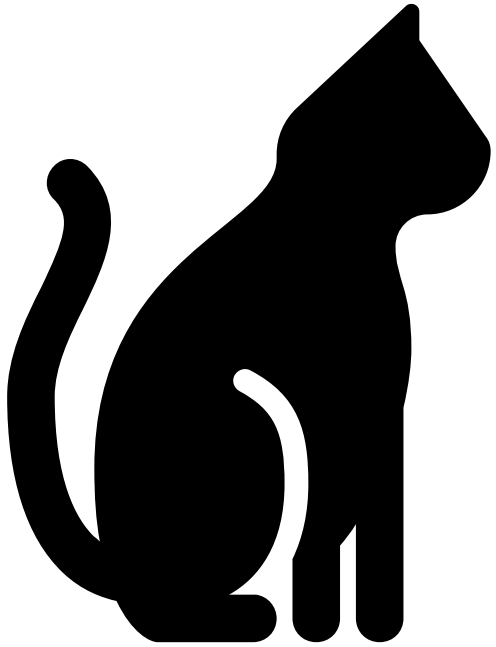
So what are neurons?

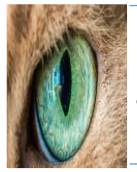
Brain: What is this shape on the left?

- **2 Eyes**
- **4 Legs**
- **1 Tail**
- **1 Unicorn Horn**

?

So what are neurons?





2 Eyes 80%

1 Tail 75%

2 Ears 66%

4 Legs 88%

1 Horn 70%

So what are neurons?

2 Eyes 80%

85%

1 Tail 75%

60%

2 Ears 66%

70%

4 Legs 88%

90%

1 Horn 70%

95%

Weights

- Assign probability of a feature in a cat based on the observation
- Are weights increasing or decreasing our probability of a feature being closer to a cat?

$Z = 0.80 \times (-0.85) + 0.75 \times 0.62 + 0.66 \times 0.70 + 0.88 \times 0.90 + 0.7 \times (-0.95)$

Remember, probability value lies between 0 and 1.

So what are neurons?


2 Eyes 80%

x 85%


1 Tail 75%

x 60%


2 Ears 66%

x 70%


4 Legs 88%

x 90%


1 Horn 70%

x 95%

Σ

Sigmoid Activation

Activation

- This is important to normalize the output
- Selecting sigmoid activation

$Z = \text{sigmoid}(0.80 \times (-0.85) + 0.75 \times 0.62 + 0.66 \times 0.70 + 0.88 \times 0.90 + 0.7 \times (-0.95))$

So what are neurons?->Summary

Summary:

- Neuron takes the inputs from the outside world and multiplies them by their weights based on feature selection, which is summed up and activation function is applied
- Neuron’s adjust the weights based on ton of data or examples of inputs and outputs.
- Let’s say we show the neuron a 10,000 examples of drawings of cats and not cats and for each of those examples we show what features are present and how strongly we are sure they are here.

So what are neurons?->Summary

Summary:

Based on 1000's of images the neuron decides if features are:

- Important and positive ->Feature is present in most images (for example, every drawing of cat had a tail in it so the weight must be positive and large)
- Not important -> Feature is present in a few images (for example only a few drawings had 2 eyes, so the weight must be small)
- Important and negative -> Negative feature present in most images (for example every drawing containing a horn has been in fact a drawing of a unicorn not a cat so the weight must be large and negative).

So what are neurons?->Summary->Blackbox Concept

- We showed that neuron decides if the image contains a cat based on specific features.
- In real life we generally don’t know the features selected to predict the final output. In our example we stated that this is not a cat based on the features like a horn.
- But in real life the neural network can choose features like pixels in certain position and the process them through many, many classifications that we don’t know nothing about and based on that predict the final output.
- This is called the black box approach because we only see the input and output not all the computations done in the middle. Because of that the layers that are between the input and output layers are called the hidden layers.

Loss function

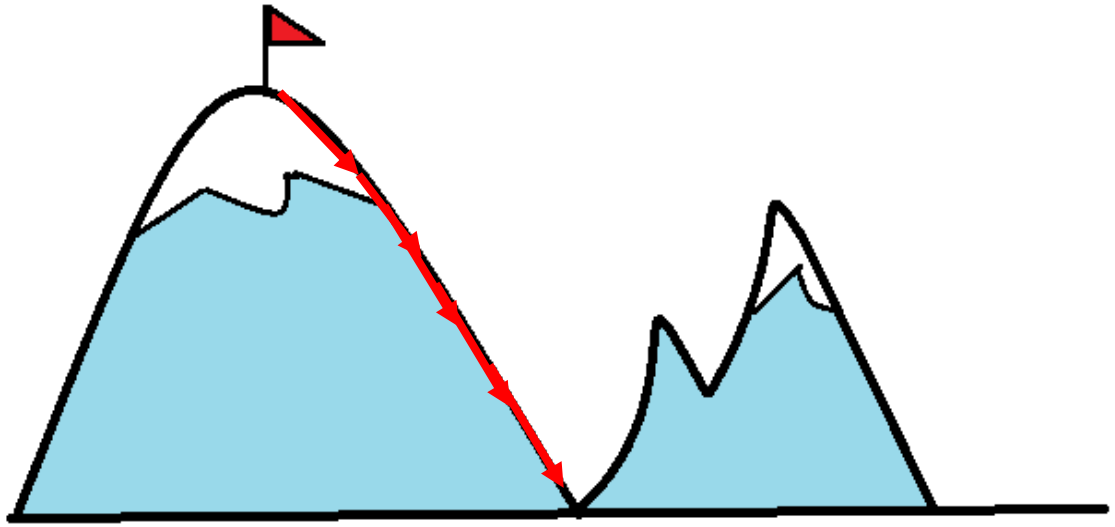
- Loss function-> Difference between the predicted and the actual output
- Mean Squared Error (MSE): Average squared difference between the estimated values and what is estimated. MSE= $\frac{1}{n}\sum_{i=1}^{n}$(Target-Actual)2
- Gradient Descent: Rate of change of our loss with respect to our weights or derivative of the loss function to understand how the weights affect the input

Gradient Descent

- Optimization algorithm used to minimize cost function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient.
- The whole goal is to minimize error in predictions and in turn increasing the accuracy of the model.
- Weights and biases are tweaked in a model which are the model parameters to achieve the best possible accuracy.

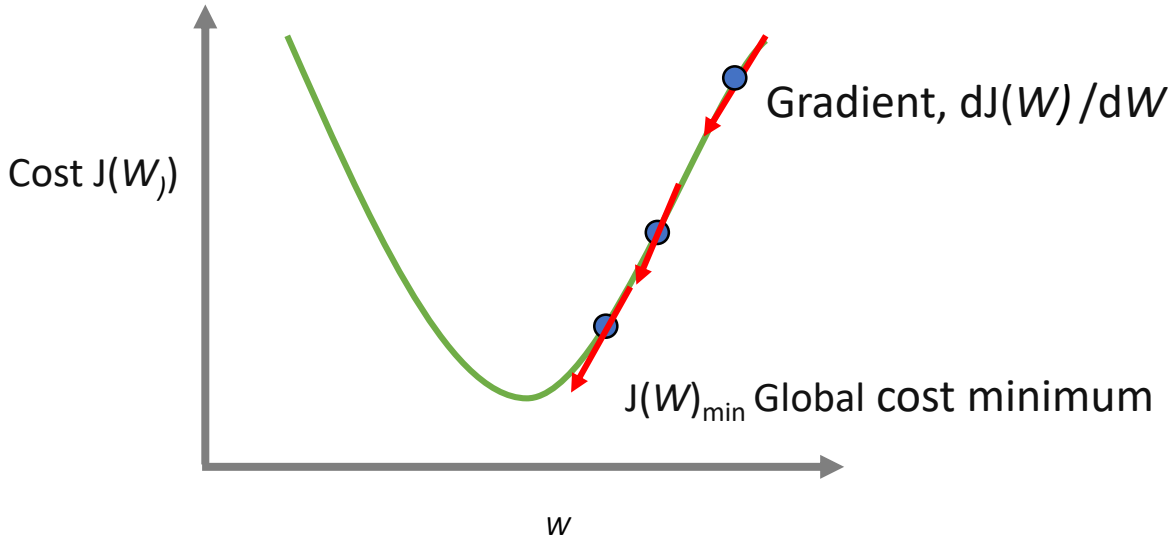
Gradient Descent: How does it work?

- Imagine you are on top of a mountain.
- Goal is to reach the bottom field, but there is a problem: you are blindfolded. What do you do then?
- Well, you will have to take small steps around and move towards the direction of the higher incline.
- You do this iteratively, moving one step at a time until finally reach the bottom of the mountain.



Gradient Descent: How does it work?

- Solving mathematically, the slope is a derivative value of inclination at a specific point
- When derivative is:
  - positive, slope goes down on right
  - negative, slope goes down on left
  - equal to 0, it means it is stagnant



$$w_j := w_j - \alpha . \frac{\partial J(w_0, w_1)}{\partial w_j}$$

$$\text{SSE} = J(w_0, w_1) = \tfrac{1}{2}\text{Sum (Predicted Value - Actual Value)}^2 = \tfrac{1}{2m} \sum_{i=1}^{m} (a_\Theta(x_i) - y_i)^2$$

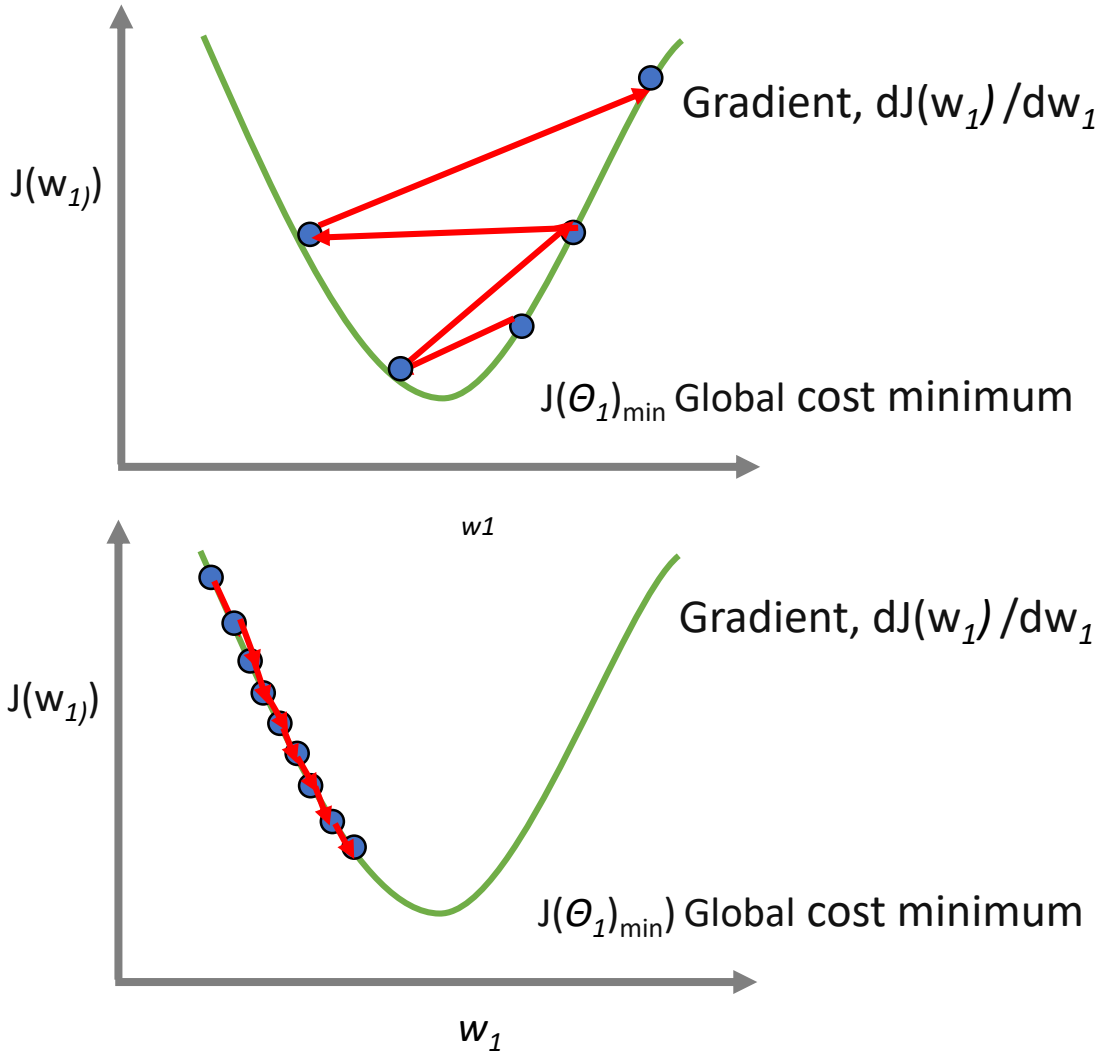
$$h_\Theta(x_i) = w_0 + w_1 x_i$$

Gradient Descent: Learning Rate Selection

- $w_j := w_j - \alpha . \frac{\partial J(w_0, w_1)}{\partial w_j}$

Learning Rate:

- If α is too large, gradient descent will bounce around and can miss the minimum.
- If α is too small, gradient descent will be slow and may never reach the minimum
- As w_1 becomes smaller with multiple iterations derivative becomes smaller and is 0 at the minimum



Summary: Gradient Descent

- The procedure starts off with initial values for the coefficient or coefficients for the function. These could be 0.0 or a small random value.
- The cost of the coefficients is evaluated by plugging them into the function and calculating the cost;

  cost = f(coeff)
- The derivative is the slope of the function at a given point. We need to know the slope so that we know the direction (sign) to move the coefficient values in order to get a lower cost on the next iteration.

  delta = ∂(cost)
- Now that we know from the derivative which direction is downhill, we can now update the coefficient values. A learning rate parameter (alpha) must be specified to define how much coefficients can change on each update.

  coefficient = coefficient – (alpha * delta)
- This process is repeated until the cost of the coefficients is close to 0.

Backward Propagation: Gradient Calculation and equation

- Forward propagation is when data is passed through the network’s parameters to make a predicted output.
- Prediction is then measured from the ground truth with the loss function where the goal is to minimize the error using optimization method like gradient descent.
- Backward propagation is where based on the predicted output from the network algorithm backpropagates error information to alter the parameters to reduce the error which is done with the help of gradient descent. In a neural network using an error function, it calculates the gradient of the error function with respect to the neural network's weights.
- Gradient calculation is performed from the last layer and moved towards the first layer i.e. backwards.

Diabetic or not, example: Colab Notebook

Person	Smoking	Obesity	Exercise	Diabetic
Person 1	0	1	0	1
Person 2	0	0	1	0
Person 3	1	0	0	0
Person 4	1	1	0	1
Person 5	1	1	1	1

1 is true and 0 is false; Assumption is that there is a strong co-relationship between diabetes and obesity.

Intuition: Supervised Learning

- Goal is to classify email as spam or not-spam.
- Can rules for a certain keywords be created as spam? Sounds a bit complicated!
- Instead we can use a training set consisting of the emails themselves along with their labels, in this case spam or not-spam.
- Given this data, you can analyze the text in the email and utilize the spam/not-spam labels to teach a machine learning classifier what words occur in a spam email and which do not – without complicated hand-coding.
- This is an example of supervised learning.

Reference: PyImage Book

Intuition: Supervised Learning

- With the training data, model is created through a training process where predictions are made on the input data and corrected when the predictions are wrong.
- This training process continues until the model achieves some desired stopping criterion, such as a low error rate or a maximum number of training iterations.
- Common supervised learning algorithms as discussed previously, include Logistic Regression, Support Vector Machines SVMs), Random Forests and Artificial Neural Networks.

Reference: PyImage Book

Basic Classification Exercise

Steps:

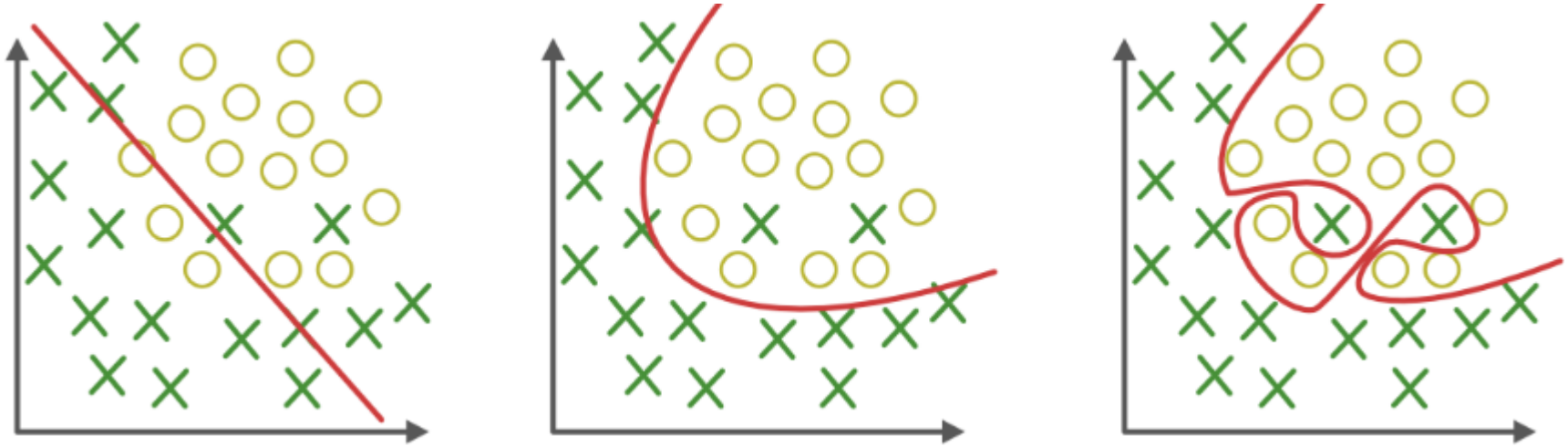
1. Gather or create a dataset: Categories should be balanced else overfitting can be expected with higher frequency images
2. Split dataset in to train and test: Training set has to be independent of the test set. Examples for common train/ test sizes: 90/10, 85/15, 80/20, 75/25
3. Data Pre-processing. Example resizing, orientation change etc.
4. Feature extraction – applicable for hand coded ML only (example, HOG)
5. Model or network training: Learning of the training takes place here and model learns from its mistakes
6. Analysis of the trained network: Here, trained model is checked for predictions against our test set which is considered as the ground truth

Reference: https://www.tensorflow.org/tutorials/keras/basic_classification

https://mc.ai/logistic-regression-with-a-neural-network-mindset-cats-vs-dogs/

Regularization

- Regression model at times overfits the data. Occurs due to learning a complex model.
- Goal of regularization is to reduce this complexity without reducing the order of the polynomial function.



Reference Image: https://www.geeksforgeeks.org/regularization-in-machine-learning/

Common ANN Applications

Classification is done by human annotation of datasets. This is known as supervised learning.

Common Classification Applications:

- Detect faces, identify people in images, recognize facial expressions (angry, joyful)
- Identify objects in images (stop signs, pedestrians, lane markers…)
- Detect voices, identify speakers, transcribe speech to text, recognize sentiment in voices; Classify text as spam (in emails); recognize sentiment in text (customer feedback)

Reference: https://skymind.ai/wiki/neural-network

Common ANN Applications

Clustering or grouping is the detection of similarities. Learning without labels is called unsupervised learning. Unlabeled data is the majority of data in the world.

Search: Comparing documents, images or sounds to surface similar items.; Anomaly detection: The flipside of detecting similarities is detecting anomalies, or unusual behavior. In many cases, unusual behavior correlates highly with things you want to detect and prevent, such as fraud.

Predictive Analytics: With classification, deep learning is able to establish correlations between, say, pixels in an image and the name of a person. You might call this a static prediction. By the same token, exposed to enough of the right data, deep learning is able to establish correlations between present events and future events. It can run regression between the past and the future. The future event is like the label in a sense. Deep learning doesn't necessarily care about time, or the fact that something hasn't happened yet. Given a time series, deep learning may read a string of number and predict the number most likely to occur next.

Hardware breakdowns (data centers, manufacturing, transport);Health breakdowns (strokes, heart attacks based on vital stats and data from wearables)

Reference: https://skymind.ai/wiki/neural-network

Popular DL Frameworks

Google

PYTORCH

TensorFlow

Caffe