DEEP LEARNING PRIMER

Module 1

Learning Objectives

You will be able to:

- Understand differences between Machine Learning and Deep Learning
- Understand Deep Learning concepts

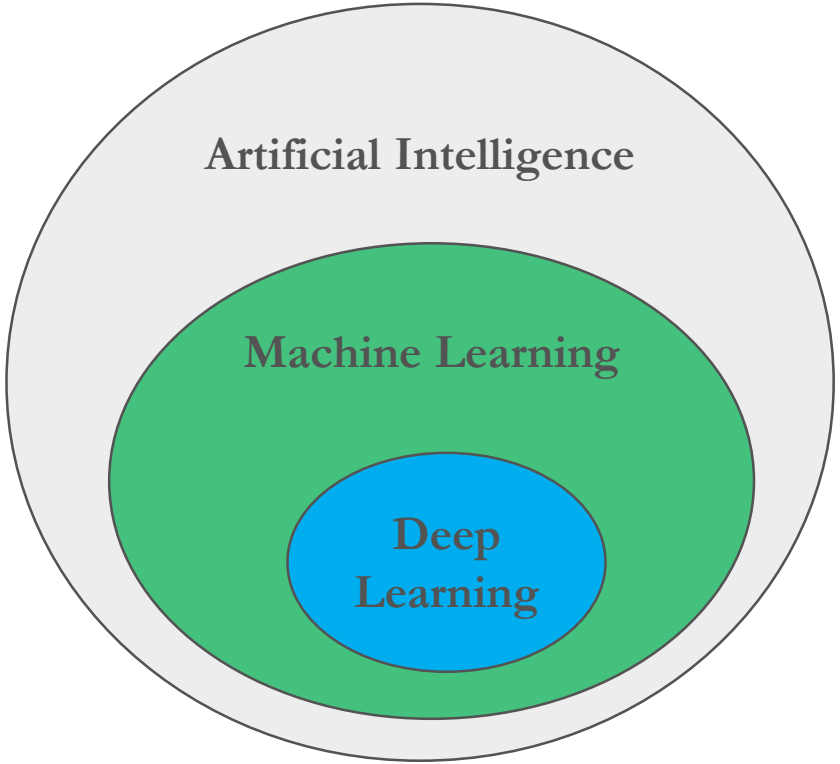
MACHINE LEARNING VS. DEEP LEARNING

AI / Machine Learning / Deep Learning

Artificial Intelligence (AI):
Broader concept of machines being able to carry out 'smart' tasks

Machine Learning:
Current application of AI that machines learn from data using mathematical, statistical models

Deep Learning: (Hot!)
Using Neural Networks to solve some hard problems



Advances in AI

Many of the latest advances in AI are due to Deep Learning

- Self-driving cars
- Image recognition
- Language translation

About Deep Learning

Deep Learning uses Neural Networking techniques

Neural Networks fell out of favor in the ‘90s as statistics-based methods yielded better results

Now making a comeback due to:

- Big Data: we have so much data to train our algorithms
- Big Data ecosystems: Hadoop*, Spark*, Cloud
- Big Compute: cloud computing
- Advances in hardware: CPU, GPU and TPU

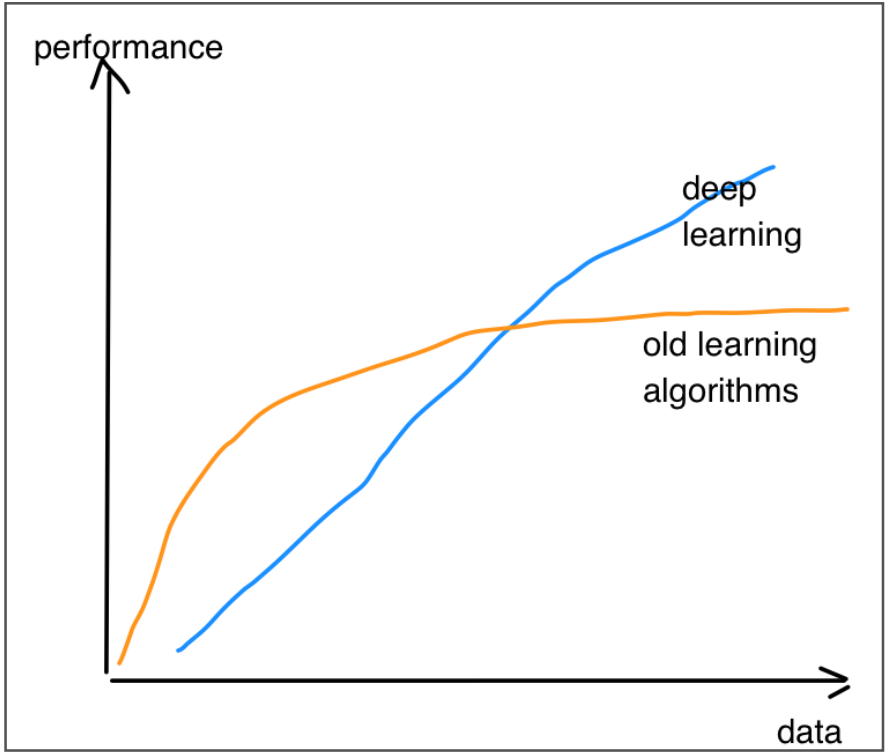
ML vs. DL

	Machine Learning	Deep Learning
Data size (see next slide for graph)	Performs reasonably well on small / medium data	Need large amount of data for reasonable performance
Scaling	Doesn't scale with large amount of data	Scales well with large amount of data
Compute power	Doesn't need a lot of compute (works well on single machines)	Needs a lot of compute power (usually runs on clusters)
CPU/GPU	Mostly CPU bound	Can utilize GPU for certain computes (massive matrix operations)
Feature engineering	Features need to be specified manually by experts	DL can learn high-level features from data automatically
Execution time	Training usually takes seconds, minutes, hours	Training takes lot longer (days)
Interpretability	Easy to interpret	Hard to understand the final result

Scaling With Data

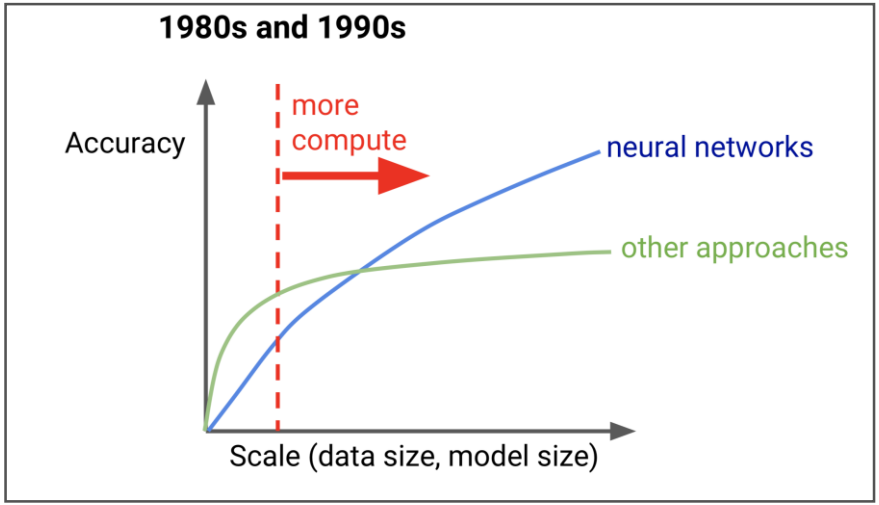
ML algorithms tend to not scale well for massive amount of datasets

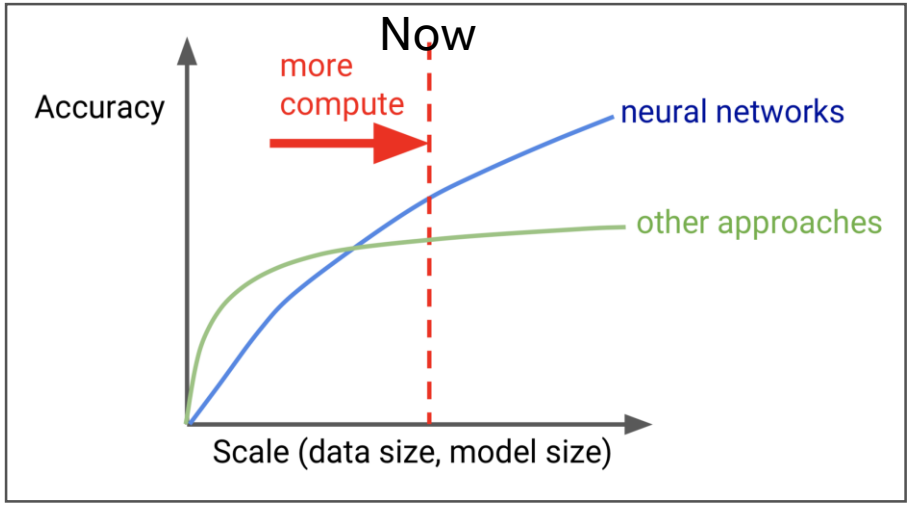
See next slides for Jeff Dean's (Google*) take on this



Scaling With Data

Now that we have massive amount of data, DL algorithms are outperforming ML algorithms

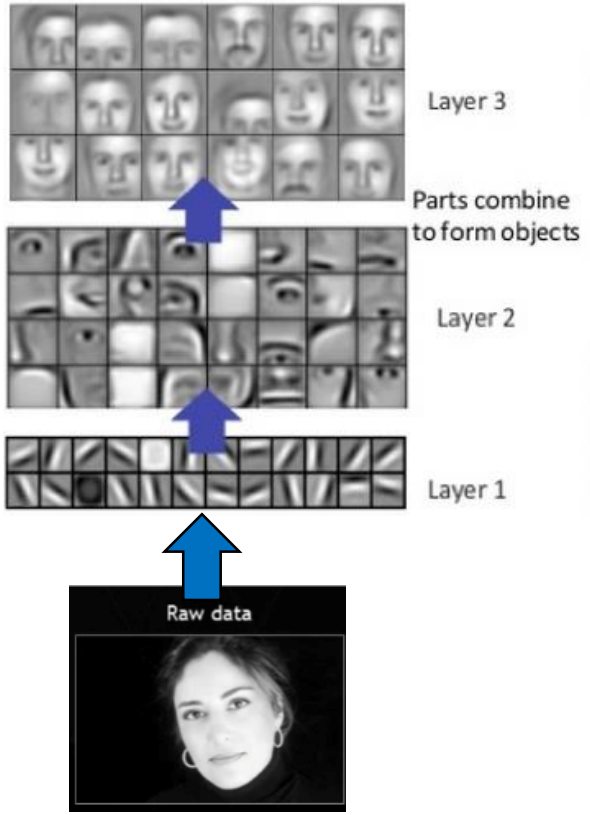




A Deep Learning Example : Image Recognition

Here each layer is doing some computation and sending the results to the next layer

Subsequent layers can recognize complex shapes like eyes, noses, etc.

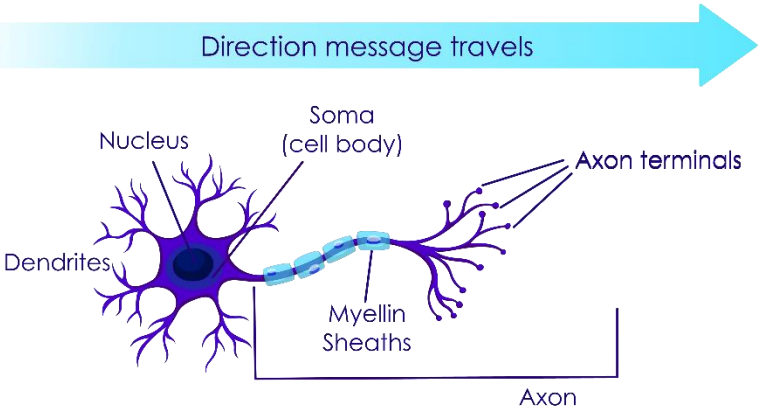


PERCEPTRON

A Brief History: Modeling After Neurons in Brain

The human brain is a bunch of interconnected neurons

The neuron is like a “gate” – produces an output



History, 1943: McCulloch Pitts Neural Model

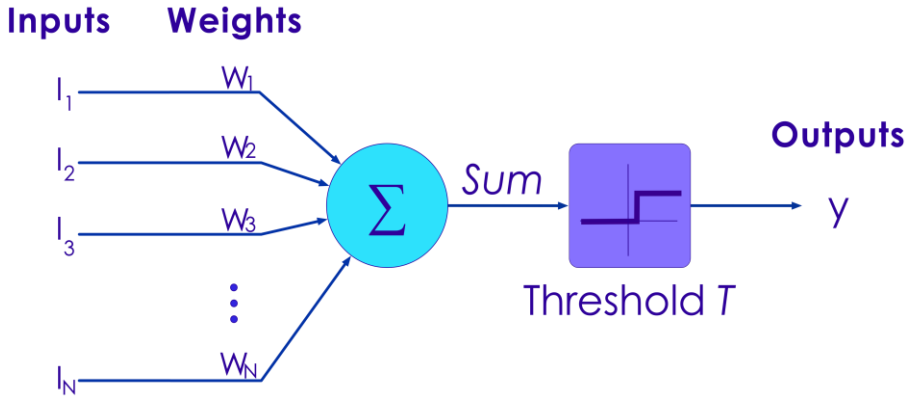
McCulloch and Pitts defined a simple model of a neuron

It consisted of $_N$ inputs In and $_N$ Weights

Go to a transfer (sum) function, apply a threshold to an output

Limitations:

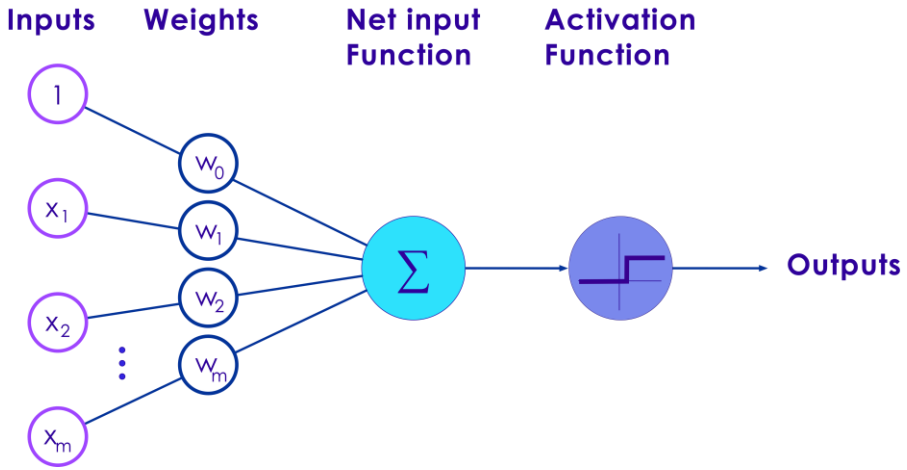
- Binary output
- No way to automatically train weights W_N



History, 1957: Frank Rosenblatt

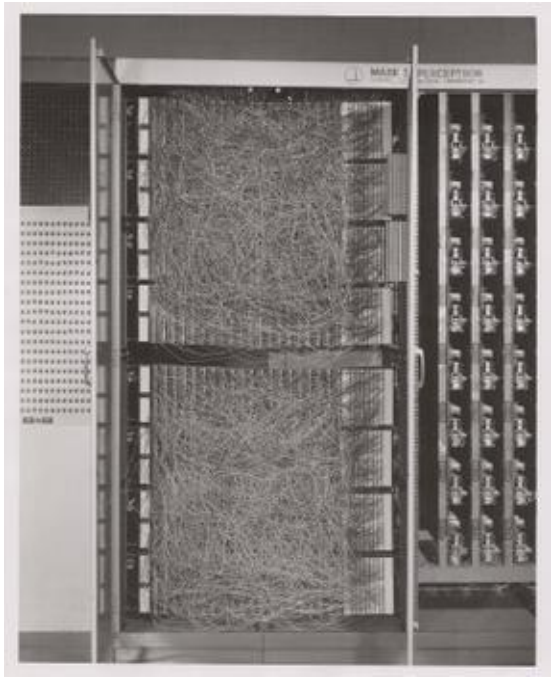
Frank Rosenblatt invented the Perceptron:

- Simplest type of feedforward neural network



Perceptron (Single Layer Perceptron)

The perceptron is a linear model used for binary classification with a simple input–output relationship



Mark 1 Perceptron

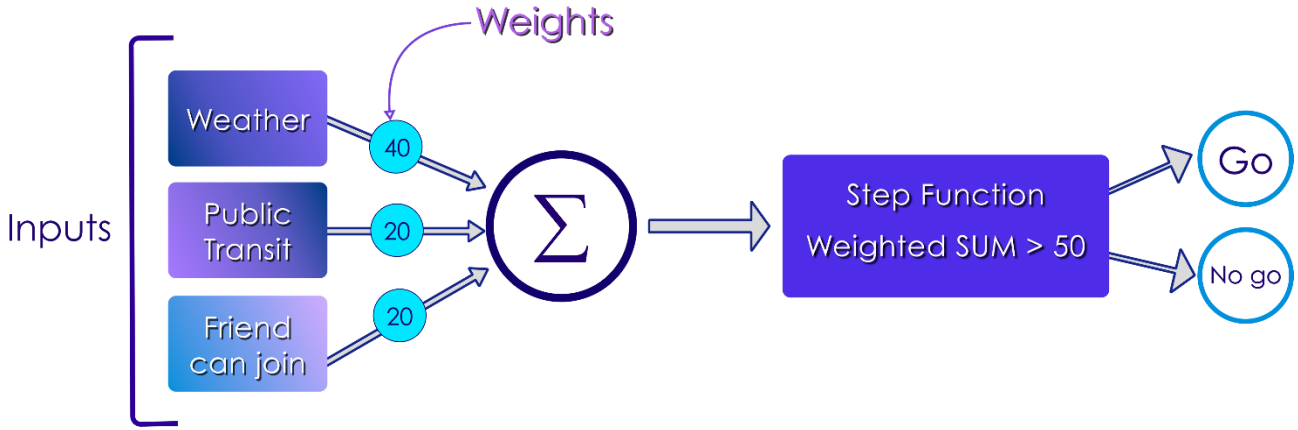
A Very Simple Perceptron

Here this perceptron is deciding if I should go to a concert.

It considers various inputs (weather, if a friend will join, etc.)

And different weights

If the final score is > 50, then the answer is YES



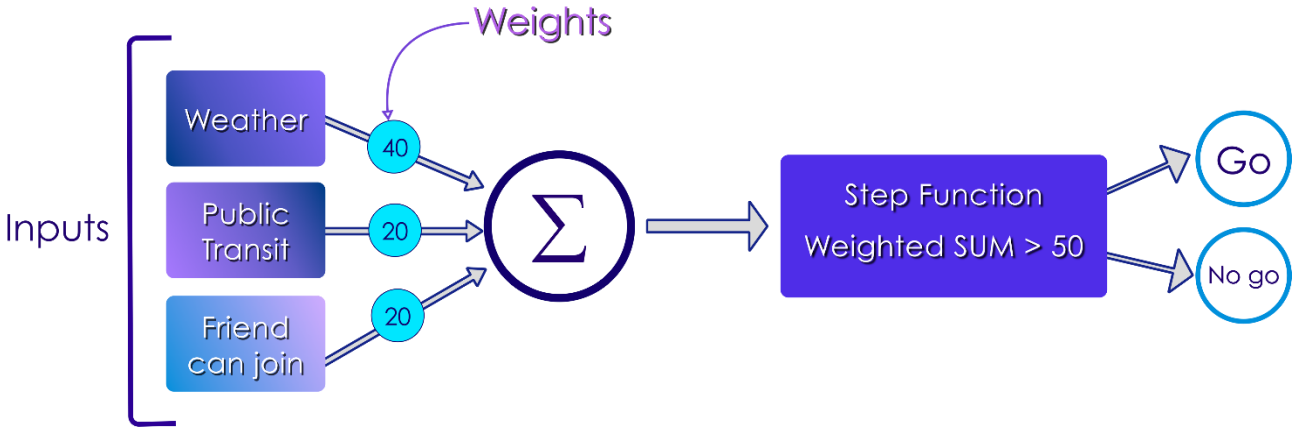
A Very Simple Perceptron

What is the outcome if

- Weather is good
- And a friend can join?

What is the outcome if

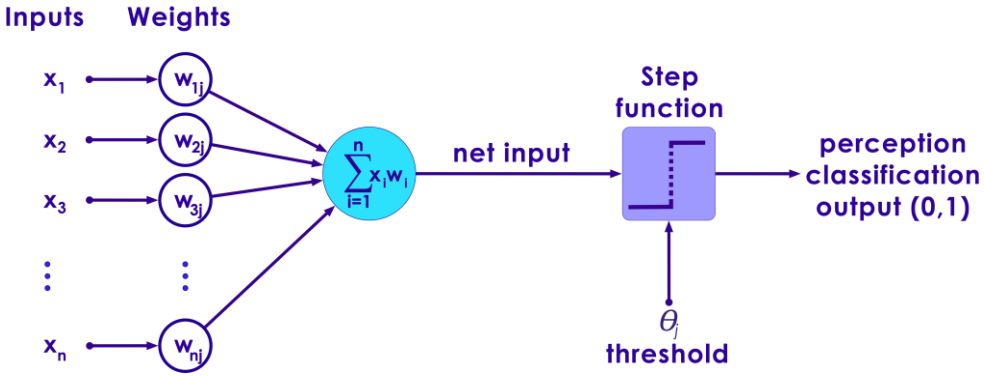
- You can go by public transit
- And a friend can join



Perceptron (Generalized)

Perceptron will have multiple inputs and an output

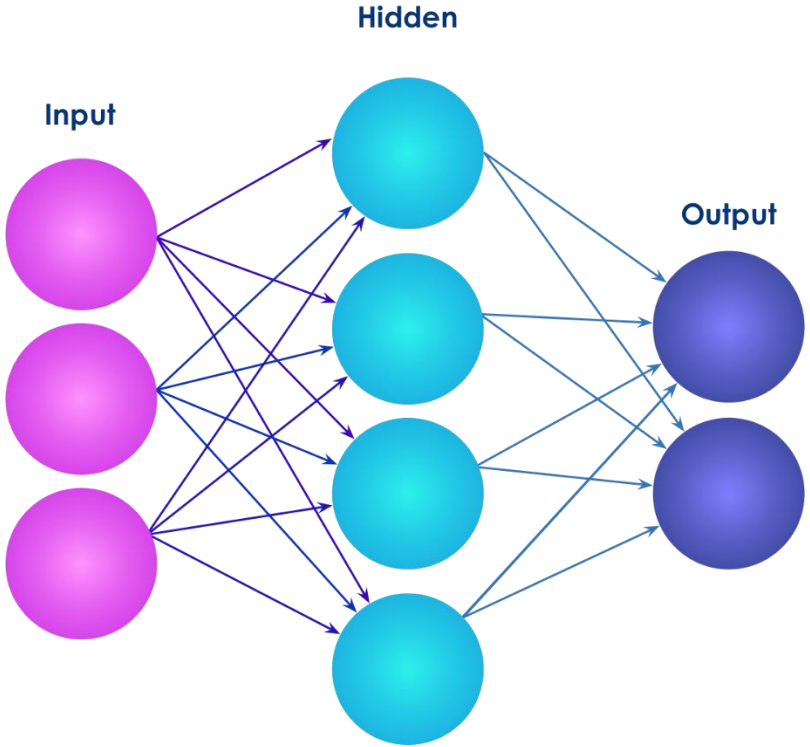
Terminology	Explanation
X_i	Input
W_i	Weight for the input
x.w	Dot product of inputs and weights $\sum x_i \oplus w_i$
n	Number of inputs
b	Bias term (does not depend on input values, shifts decision boundary from origin)



Constructing Neural Networks

We add multiple layers

Each layer can have many neurons

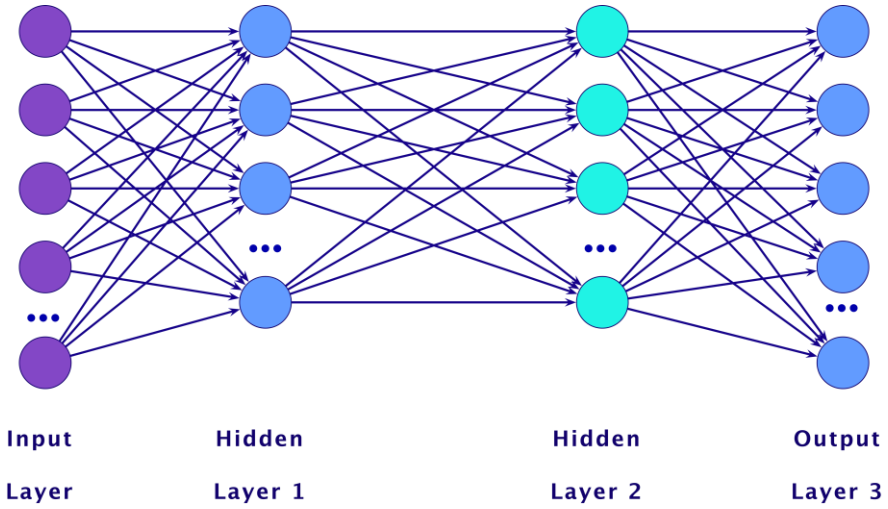


'Deep' Neural Networks

If the network has more than ONE hidden layer, it is called DEEP network (or more than 3 layers total including input and output)

More layers gives the network the ability to adapt to complex data

However, more layers take more time to train



Examples of 'Deep' Neural Networks

Image Recognition

- ResNet (from 2015) with 152 layers
  A CNN (Convolutional Neural Network) from Microsoft that won ILSRVC 2015 competition
- AlexNet (8 layers)
  Won ImageNet Challenge (millions of images in 20,000 categories) in 2012

Machine Translation

- **Google Translate*** uses Deep Neural Networks (DNN) to translate between languages (English to French, etc.) with very high accuracy

Examples of Deep Neural Networks

Reinforcement Learning

Bots learning to play games automatically (at superhuman levels!)

Demo1 : Deep Mind playing Atari* Breakout

Demo2 : Deep Mind AlphaGo*

Demo3 : Open AI bots playing Dota 2* and beating pro human teams!

Sizing Layers

Each Layer can have many neurons

How do we decide the number of neurons per layer?

Let's start with an example:

- Assume our input has 3 dimensions / Features (A,B,C)
- And we are classifying them into 2 classes X & Y

Sizing Layers

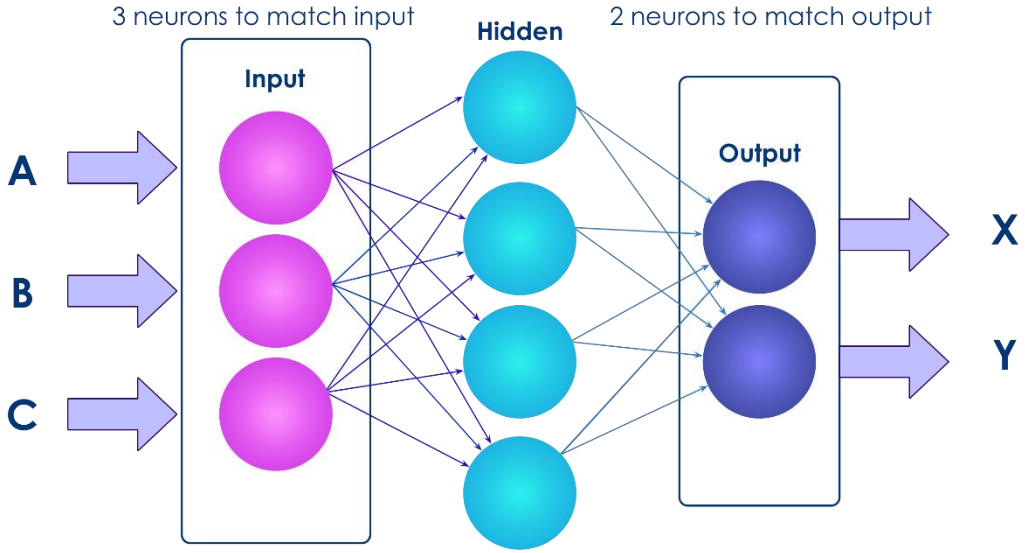
Input and Output are straight forward

Input layer is sized to match number of features

- in this case 3, for inputs A,B,C

Output layer is sized to match number of output classes

- 2 for outputs X,Y



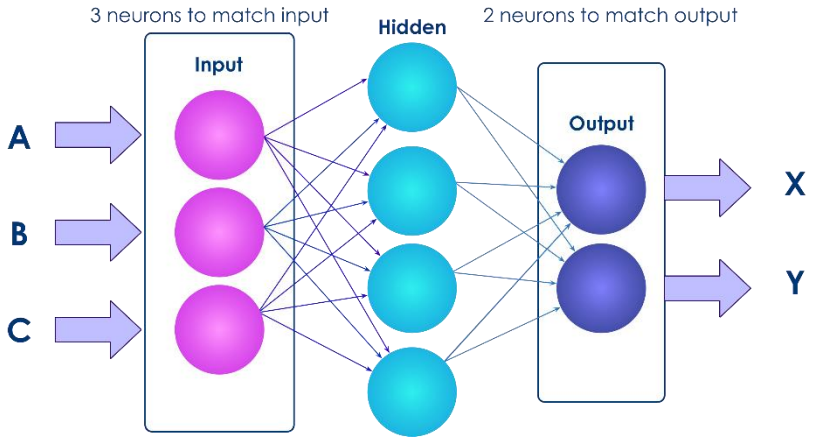
Sizing Hidden Layers

How about Hidden Layer sizing? How many neurons in Hidden Layer?

This is a 'hyper parameter' for the network

We can try a few choices and evaluate the results

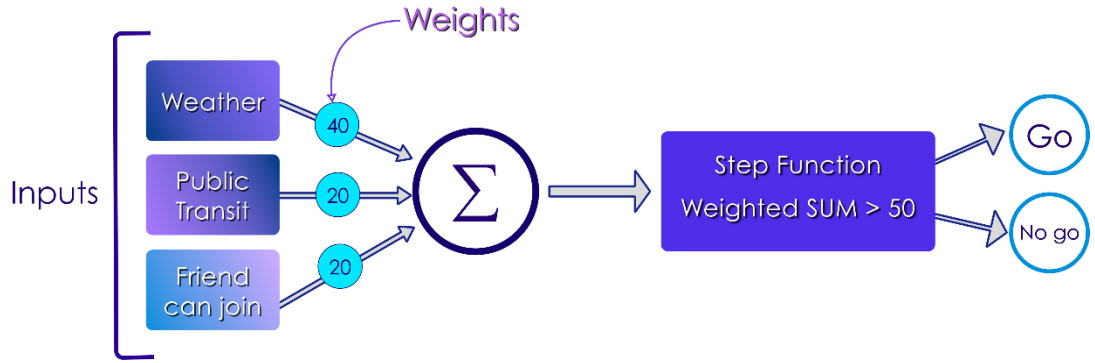
Note: Large number of neurons can 'overfit' the data



ACTIVATION FUNCTIONS

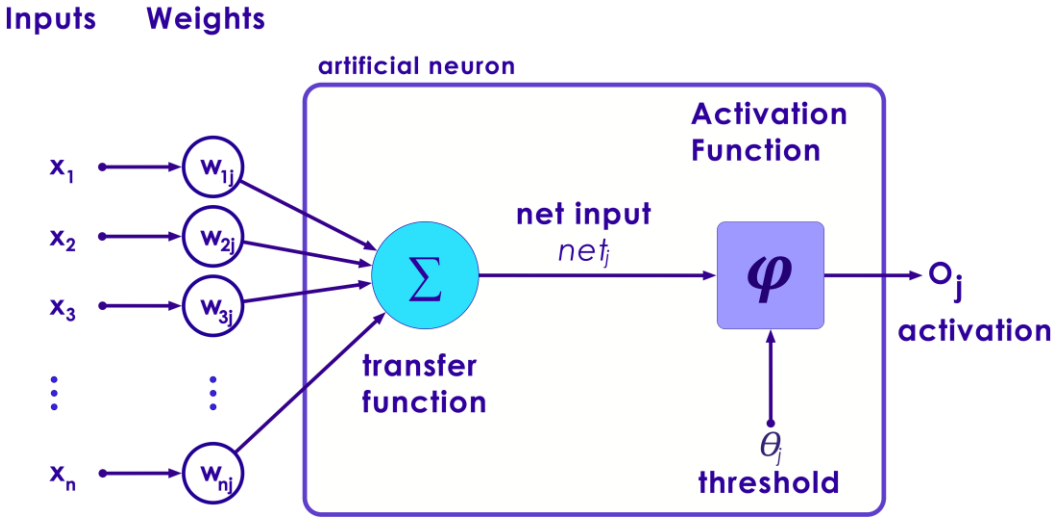
Activation Function

- In this example, the activation function is a pretty simple 'comparison function'
  - If the (dot product) sum of weight and input < 50 , output is NO
  - If the sum is >= 50, output is YES
  - So we are converting a simple number input (score) into a binary YES/NO answer



Activation Function

- Activation functions propagate the output of one layer's nodes forward to the next layer
- Allows us to create different output values
- Allows us to model non-linear functions



Activation Function

Input sum = $\sum$ (Wi . Xi) + bias term

Output = activation_function (input sum)

= g (input_sum)

One liner

Output = g ($\sum$ (Wi . Xi) + b)

Types of Activation Functions

The following Activation Functions are in use:

- Linear
- SIGMOID*
- Tanh
- Rectified Linear Unit (ReLU) (and its variants)

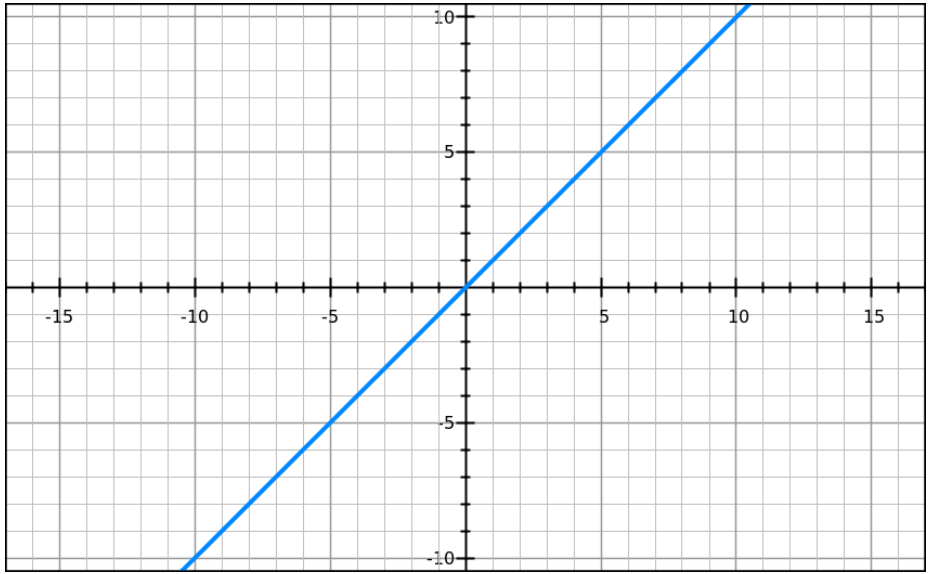
Activation Function: Linear

Very simple

Used for Linear Regression

Output = weight* Input + Intercept

Y = aX + b



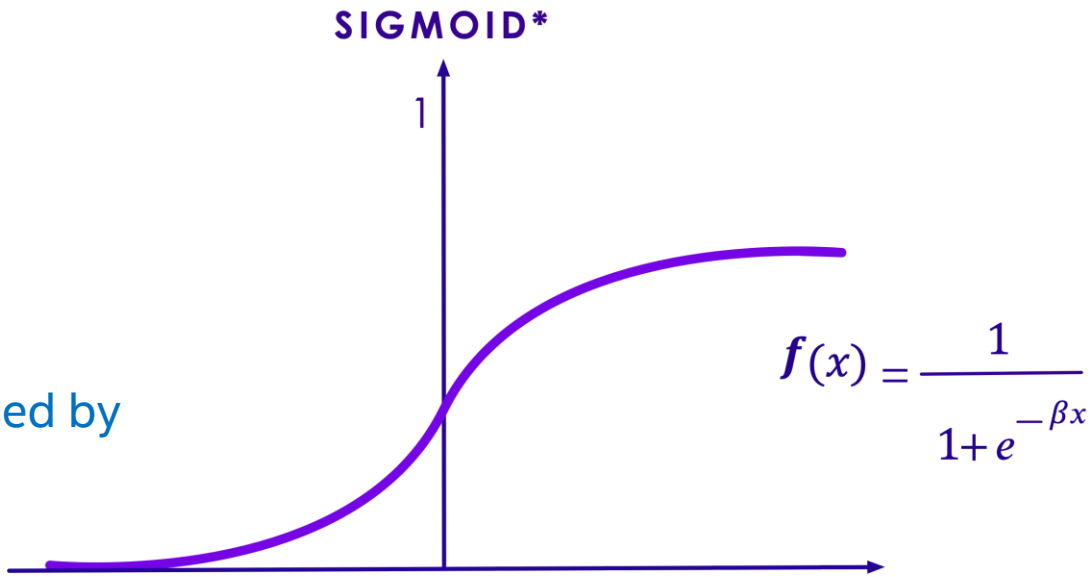
Activation Function : SIGMOID*

Comes from Logistic Regression

It reduces the input values to an output of 0 to 1 (probability)

SIGMOID* was the oldest / first Activation Function used

However, now it has been eclipsed by other Activation Functions that produce better results

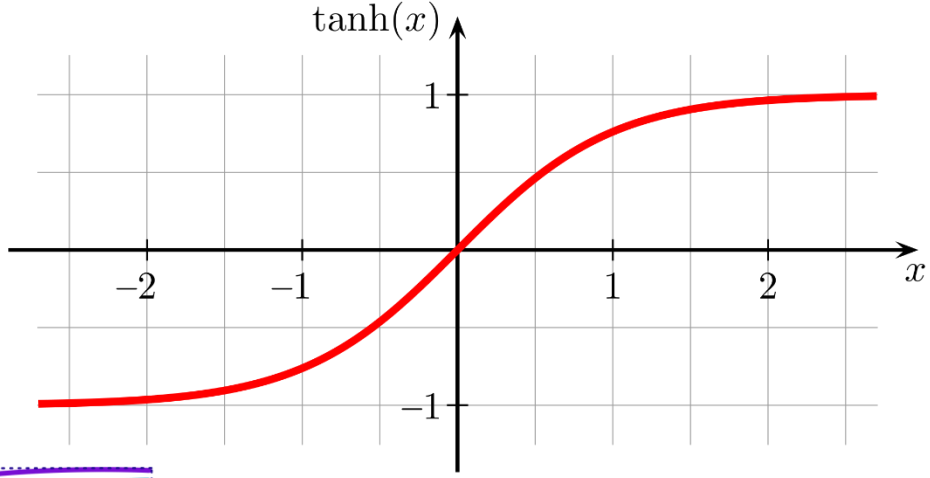


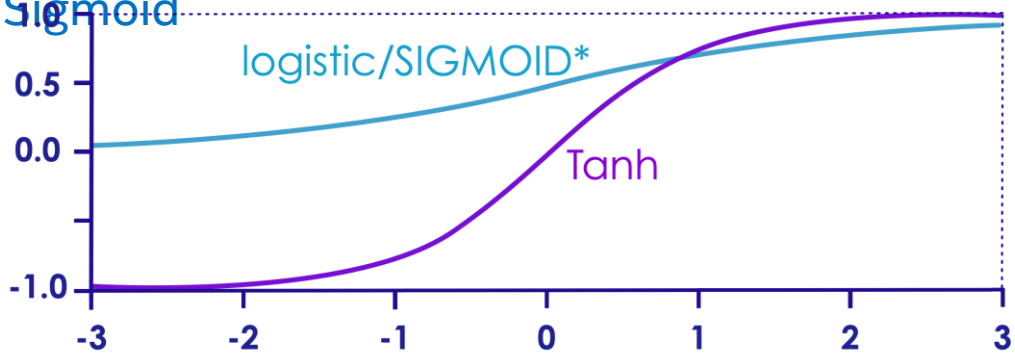
Activation Function : Tanh

Tanh function ranges from -1 to +1

(SIGMOID* function is between 0 and +1)

So Tanh function can better deal with negative numbers compared to Sigmoid





Activation Function : Rectified Linear

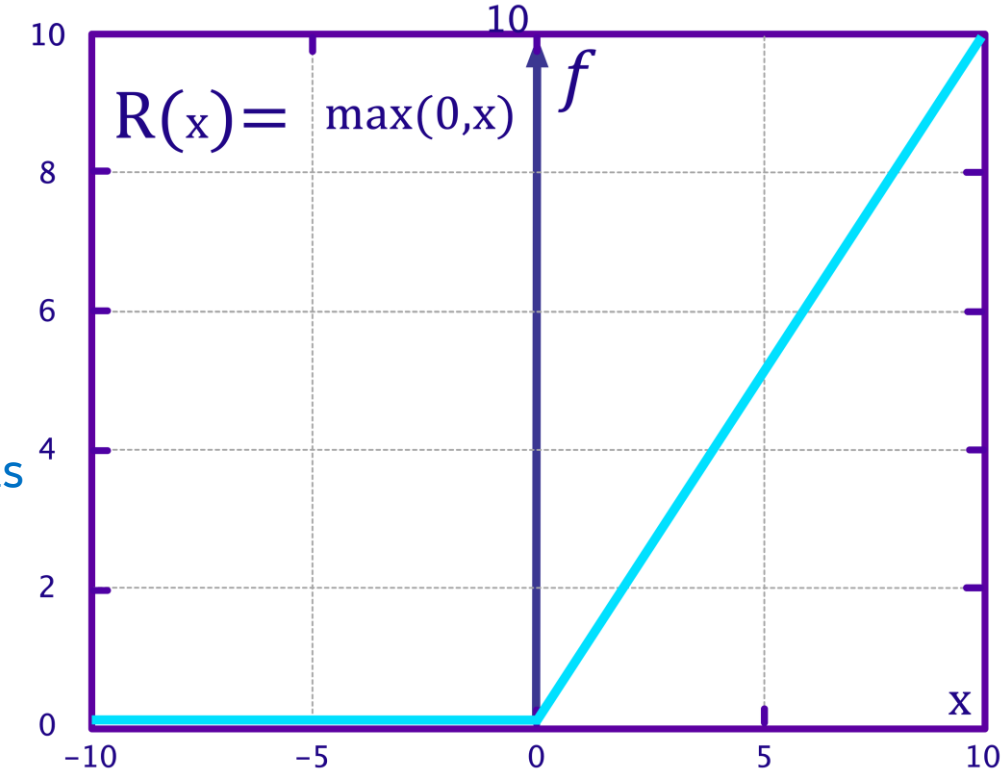
ReLU is a very simple and effective Activation Function

If input is above zero, it passes it on

If input goes below zero, it is clipped at zero

ReLU is the current state of the art, as it has been proven to work in many different datasets

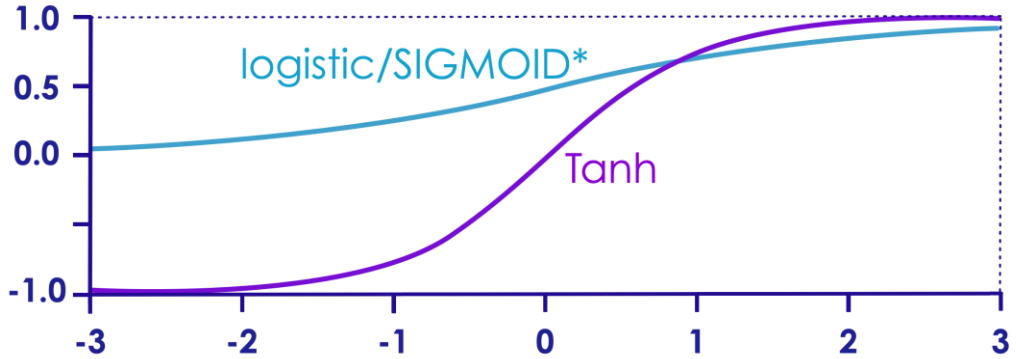
ReLu is also very fast to compute!



SIGMOID* / Tanh

SIGMOID* and Tanh both suffer from the **Vanishing Gradient Problem:**

- The derivative of a Sigmoid is less than .25
- As we propagate that through many layers, that gradient becomes smaller and smaller and eventually 'vanishes' (too small)

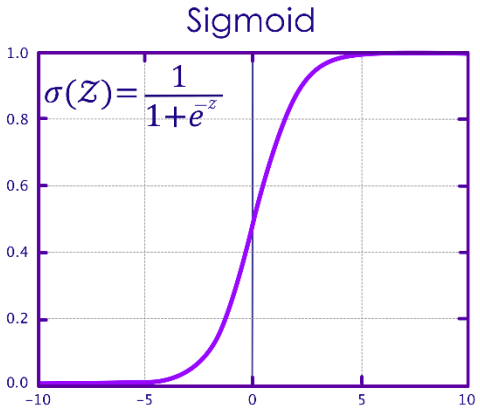


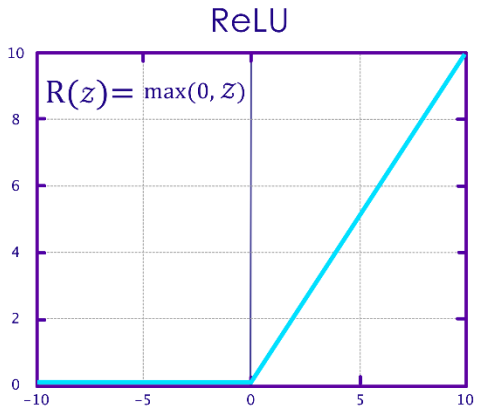
ReLU to Rescue!

ReLU doesn't suffer from

- Vanishing Gradient Problem
- Exploding Gradient Problem

Here, simpler is actually better

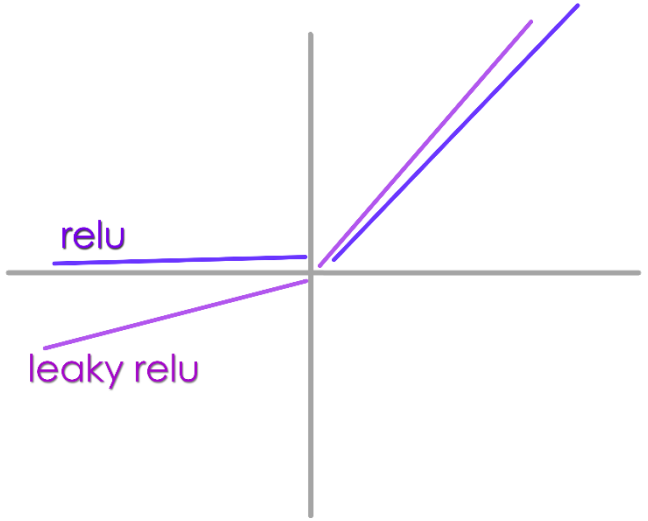




Activation Function : Leaky ReLU

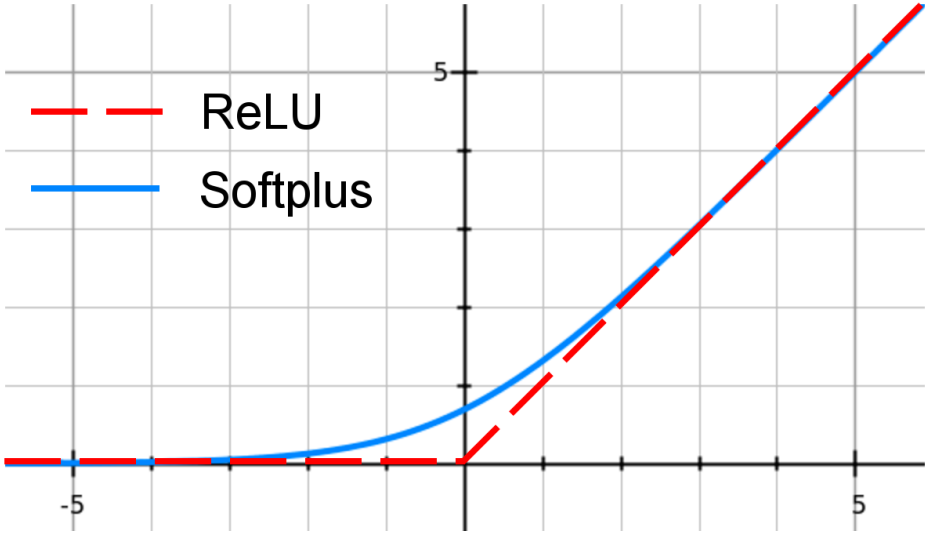
- Leaky ReLU fixes an inherent problem with ReLU
  - When input dips below zero, ReLU produces ZERO
  - This is 'dying ReLU' problem
- When input is below zero, Leaky ReLU will produce a 'small signal'

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$$



Activation Function: SoftPlus

SoftPlus is 'smooth version of ReLU'

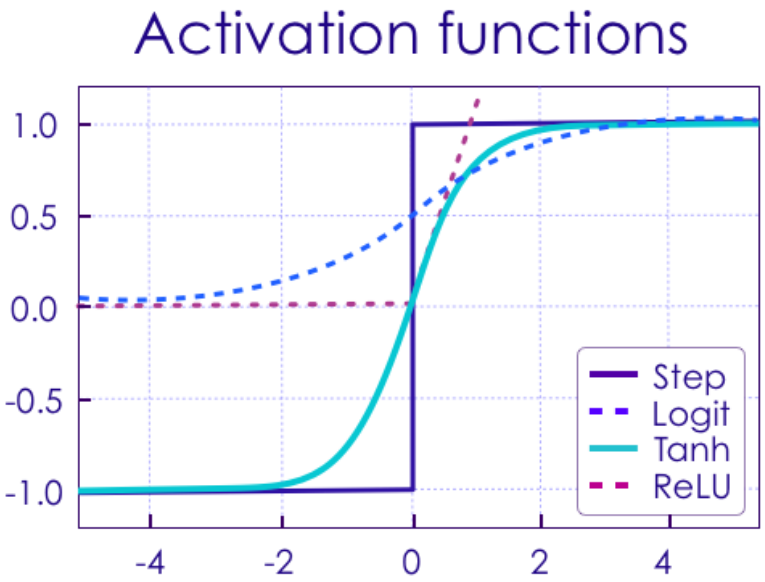


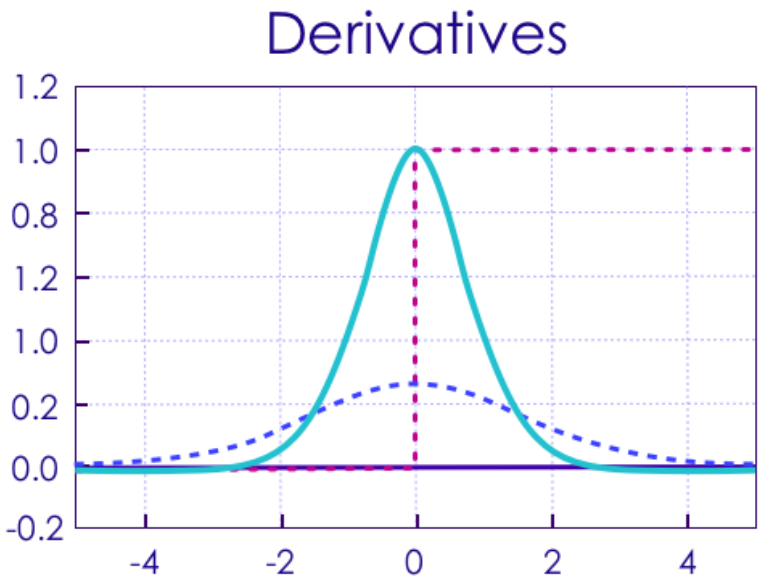
Activation Function: SoftMax*

- SoftMax* is a generalized Logistic Regression
  - Logistic Regression can do binary outcome (0 / 1)
  - SoftMax* gives out probabilities for multiple classes

digit (outcome)	0	1	2	3	4	5	6	7	8	9
Probability	0.9	0	0	0	0	0	0	0	0	0.1

Activation Functions - Comparison





LOSS FUNCTIONS

Loss Functions

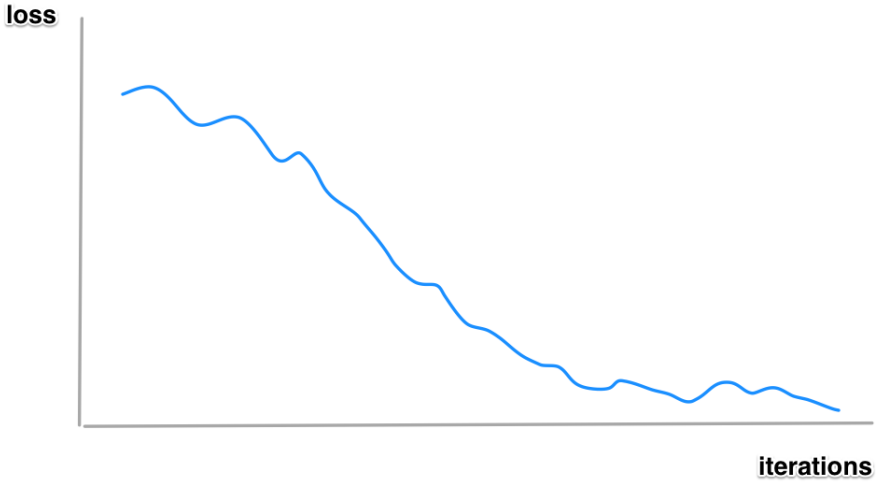
Loss functions quantify how close a given Neural Network is to the truth

How much the predictions deviate from expected outputs

- Convert this difference from Vectors to a 'single number'

Goal is to minimize Loss Function

- Zero would mean the model is 'perfect'
- It is hard to achieve in practice, but we can get pretty close to zero



Loss Functions For Regression: Mean Squared Error (MSE) Loss Function

Very much like 'Ordinary Least Squares' in Linear Regression

Pros

- Very simple to understand
- Fast to compute

Cons

- Can be prone to outliers

$$L(W, b) = \frac{1}{N} \sum_{i=1}^{N} \left(\widehat{Y_i} - Y_i \right)^2$$

Loss Functions For Regression: Mean Absolute Error Loss (MAE)

Averages 'absolute error' across all data points

$$L(W, b) = \frac{1}{2N} \sum_{i=1}^{N} \sum_{j=1}^{M} \left| \widehat{y_{ij}} - y_{ij} \right|$$

Loss Functions for Regressions Takeaway

Both 'Mean Squared Error' (MSE) and 'Mean Absolute Error' (MAE) are used widely

And they perform pretty well on most scenarios

If the inputs span a large range (X1 = 1 to 10, X2 = 1000 to 1000000), then consider normalizing them first

Other options to consider:

- Mean squared log error loss (MSLE)
- Mean absolute percentage error loss (MAPE)

Loss Function for Classification

- Classifiers segment data into 'buckets' (fraud / not-fraud, spam/not-spam)
- Most of the time, the classifier gives a probability (80% spam, 20% not spam)
- Loss Functions:
  - Hinge Loss
  - Logistics Loss

Loss Functions for Classification: Hinge Loss

- Hinge Loss is used heavily when the network does hard binary classification (0 or 1)
- Can be extended for doing 'multiclass classification'

$$\frac{1}{N}\sum_{i=1}^{N} max\left(0, 1 - y_{ij} \times \hat{y}_{ij}\right)$$

Loss Function for Classification : Logistic Loss

- Logistics Loss is preferred when we get probabilities instead of hard classifications
- For example, model says 70% spam, 30% not spam, instead of giving a 0 or 1
- Last Layer uses 'SoftMax*' activation function

digit (outcome)	0	1	2	3	4	5	6	7	8	9
Probability	0.75	0	0	0	0	0	0	0	0.15	0.10

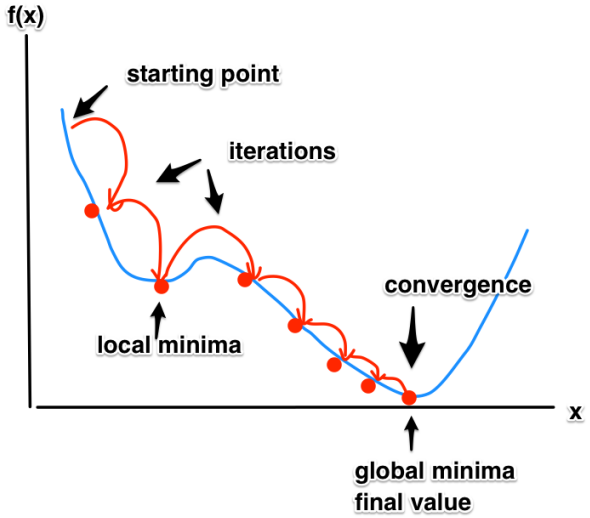
HYPER-PARAMETERS

Hyperparameters

- Models and networks have parameters that we adjust during optimization
  - Goal is to minimize error and maximize performance
- Hyperparameters
  - Learning rate
  - Regularization

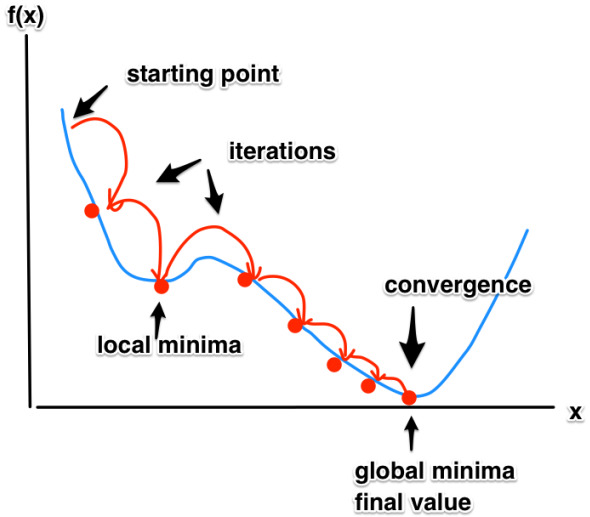
Hyperparameter : Learning Rate (alpha, α)

- Learning rate controls the amount by which we adjust parameters during optimization
- It is a coefficient that scales the size of steps
- During back propagation we multiply error gradient by learning rate



Hyperparameter : Learning Rate (alpha, α)

- Larger learning rate (close to 1.0) means:
  - Bigger steps and less number of iterations
  - But can overshoot and miss the minimum
- Smaller learning rate (0.0001) means:
  - Smaller leaps, and lots of iterations
  - Can find the minimum, but can take a long time
- What is the 'perfect' learning rate?
  - Experiment (it varies for each dataset and computation)



Hyperparameters : Regularization (lambda, ƛ)

- Regularization helps avoid 'over fitting'
- It does this by controlling coefficients so they don't get too large
- L2 (Ridge) Regularization
  - Shrinks coefficients to avoid over fitting
- L1 (Lasso) Regularization
  - Shrinks coefficients like L2, but can also make them zero
  - Effectively removing the variable from consideration (Variable Selection)

Summary

We learned about:

- Deep Learning vs. Machine Learning
- How Neural Nets have evolved
- Activation functions
- Loss functions
- Hyperparameter tuning

APPENDIX

Recommended Resources

- Intel® AI Academy : https://software.intel.com/ai-academy

- https://software.intel.com/en-us/ai-academy/basics

- **"Getting Started with Deep Learning"**
  by Josh Patterson, Adam Gibson
  Link1