PYTHON PROGRAMMING AND MACHINE LEARNING

NEURAL NETWORK

Yunghans Irawan (yirawan@nus.edu.sg)

Objectives

- Understand the basic concept of artificial neural network

- Able to implement a simple artificial neural network for classification task

Agenda

- Perceptron
- Multi-layer Perceptron
- Backward Propagation

Advantage of ANN

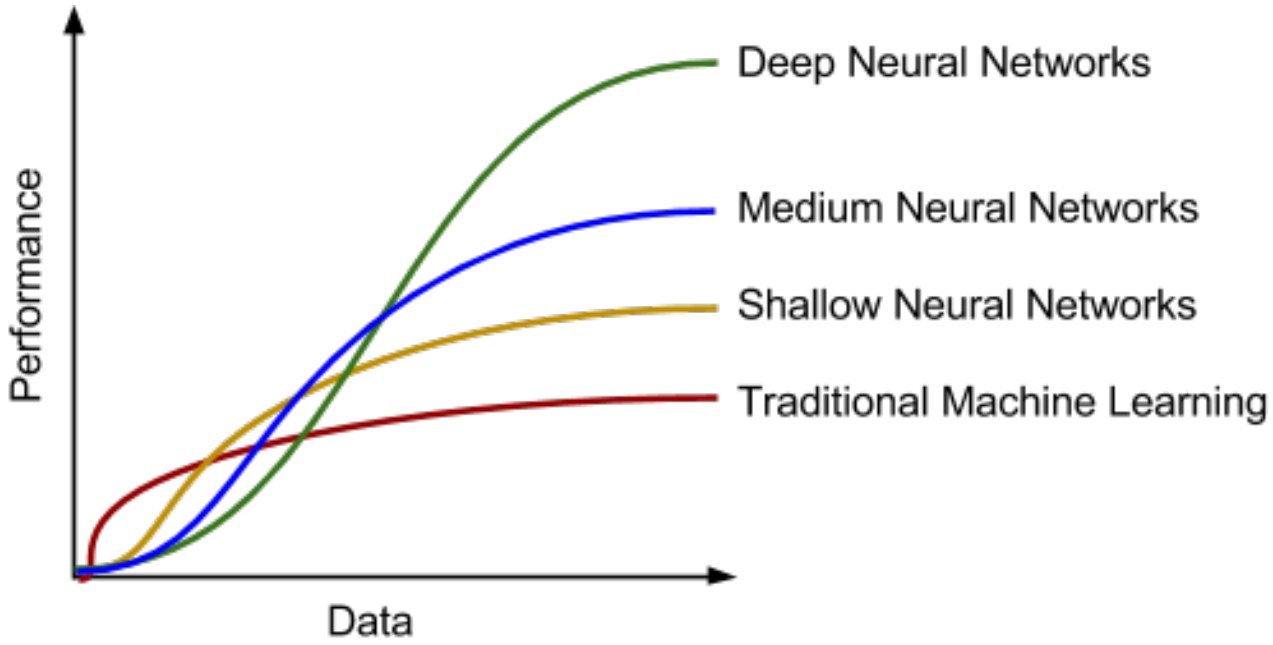
- Flexible function approximator
- As long as you add more data, more layers, mix and match different types of layers
- Good for linear and non-linear problems
- More popular now with hardware and memory speedups (makes fast training possible)

Neural Networks Applications

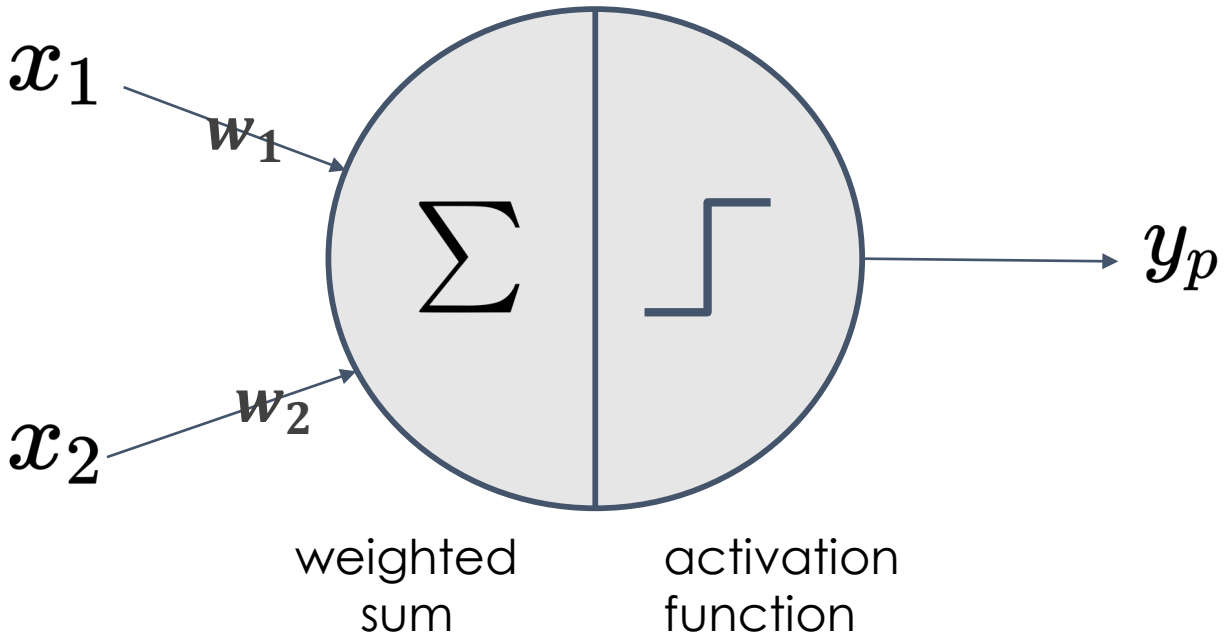
What are neural networks used for?

- **Classification**: Assigning each object to a known specific class
- **Clustering**: Grouping together objects similar to each other
- **Pattern Association**: Presenting of an input sample triggers the generation of specific output pattern
- **Function approximation**: Constructing a function generating almost the same outputs from input data as the modeled process
- **Optimization**: Optimizing function values subject to constraints
- **Forecasting**: Predicting future events on the basis of past history
- **Control**: Determining values for input variables to achieve desired values for output variables

Performance (accuracy) vs data

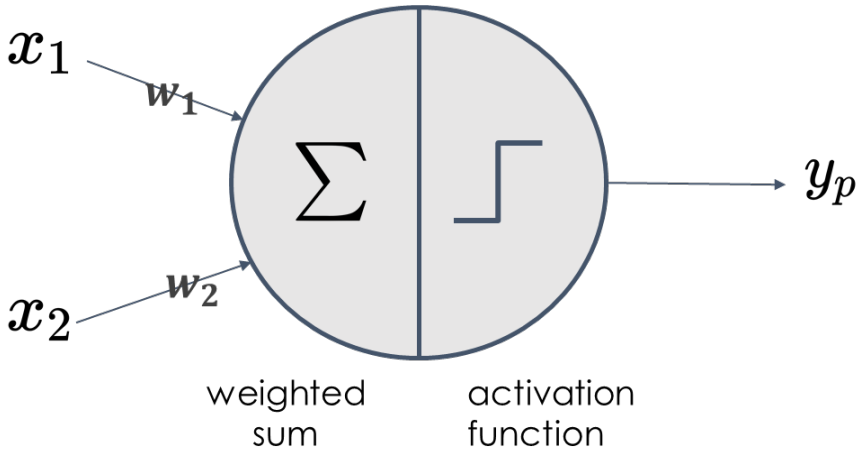


Perceptron: basic "cell" of a neural network



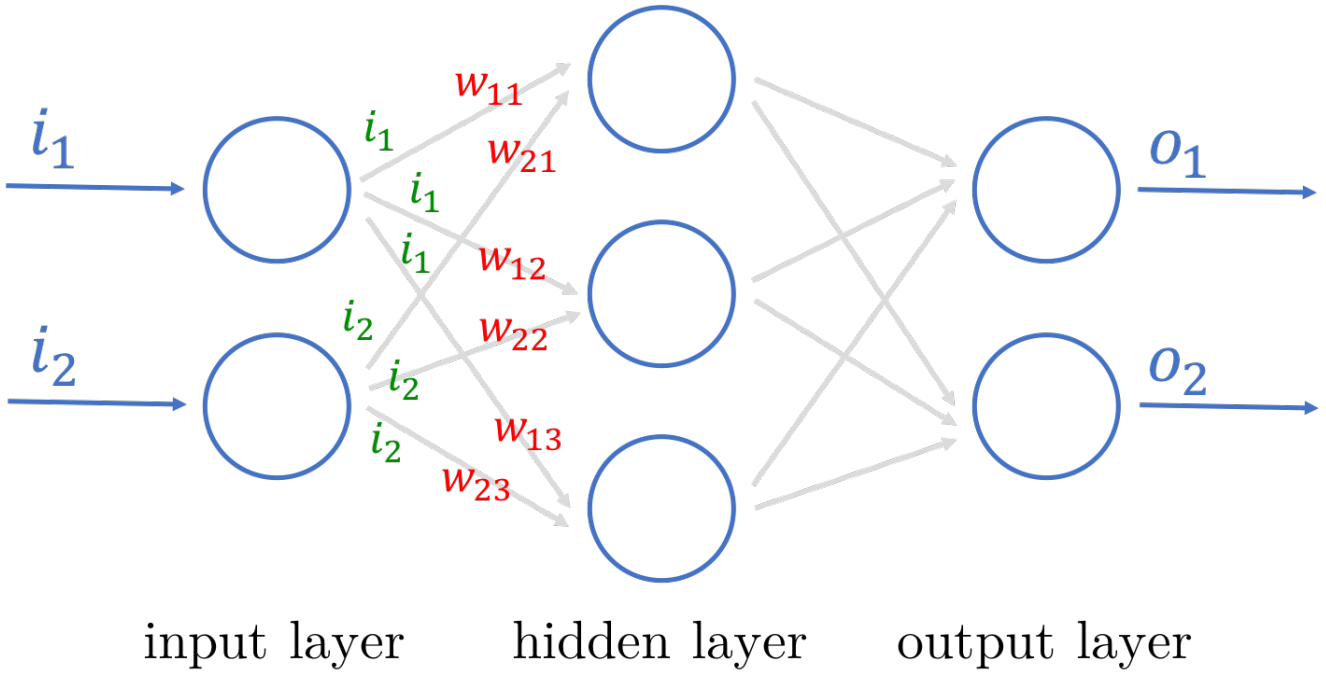
Stages of a Perceptron

1. Take a weighted sum of the inputs
2. Apply a non-linear transformation

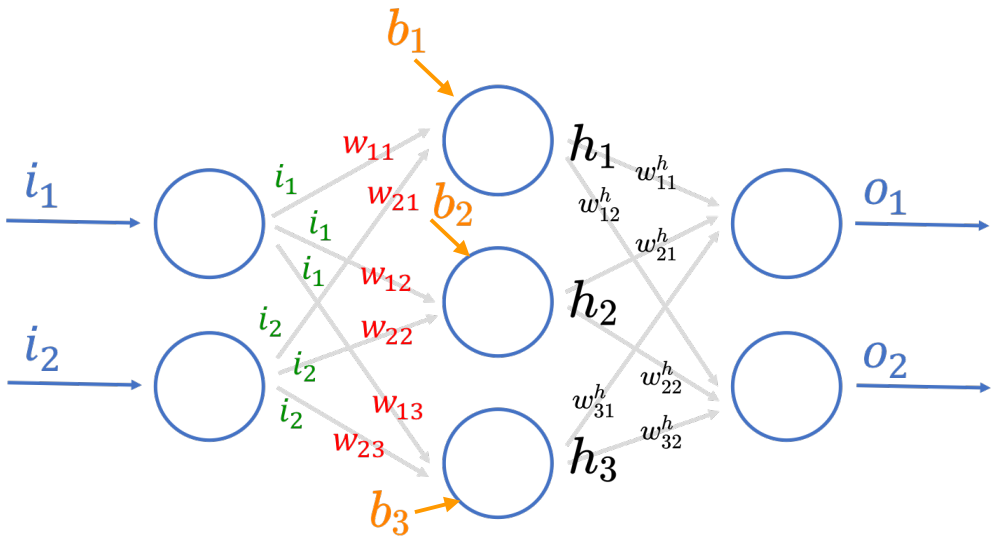


Fun fact:
Binary Logistic Regression = single neuron with sigmoid as non-linearity

Multi-layer Perceptron (layers = 2)



MLP with Bias (example)



$$I = \begin{bmatrix} i_1 \\ i_2 \end{bmatrix}$$

$$W_{ih} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{22} & w_{22} & w_{23} \end{bmatrix}$$

$$B = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$h = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = sigmoid(W_{ih}^T I + B)$$

$$h_1 = sigmoid(w_{11} i_1 + w_{21} i_2 + b_1)$$

$$h_2 = sigmoid(w_{12} i_1 + w_{22} i_2 + b_2)$$

$$h_3 = sigmoid(w_{13} i_1 + w_{23} i_2 + b_3)$$

Feedforward vs. Back Propagation

Feedforward, or "forward pass" is simply an operation that traverses the neural network from the input to the output layers

- This performs the prediction

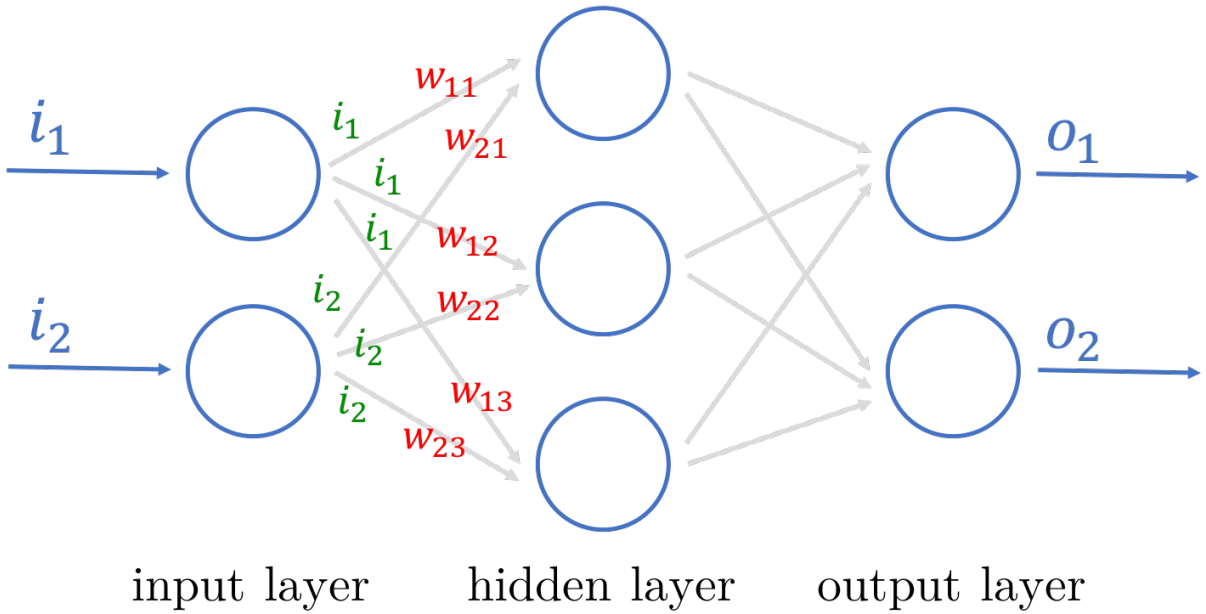
Conversely, **back propagation** or "backprop" is an operation that traverses backwards (from output layers to the input layers)

- This performs the weight updates

Reason: neural networks are multi-layer and we perform the dot products layer by layer. The output of the previous layer is the input to the next layer.

Multi-layer Perceptron

Forward pass – perform the prediction



Calculate the average error using loss function

Back propagation – update the weight layer by layer

Types of Non-linearities

- Also known as "Activation Functions" - think of a brain cell firing on some threshold (i.e. "activating")
- Common ones:
  - Sigmoid
  - Tanh
  - ReLU
- Limited use non-linearities:
  - Softmax (only used for output of multi-class classification)

Choosing Non-linearities

- Choosing the right Activation Function
- Heuristics to know which activation function should be used in which situation. Good or bad – there is no rule of thumb.

Choosing Non-linearities

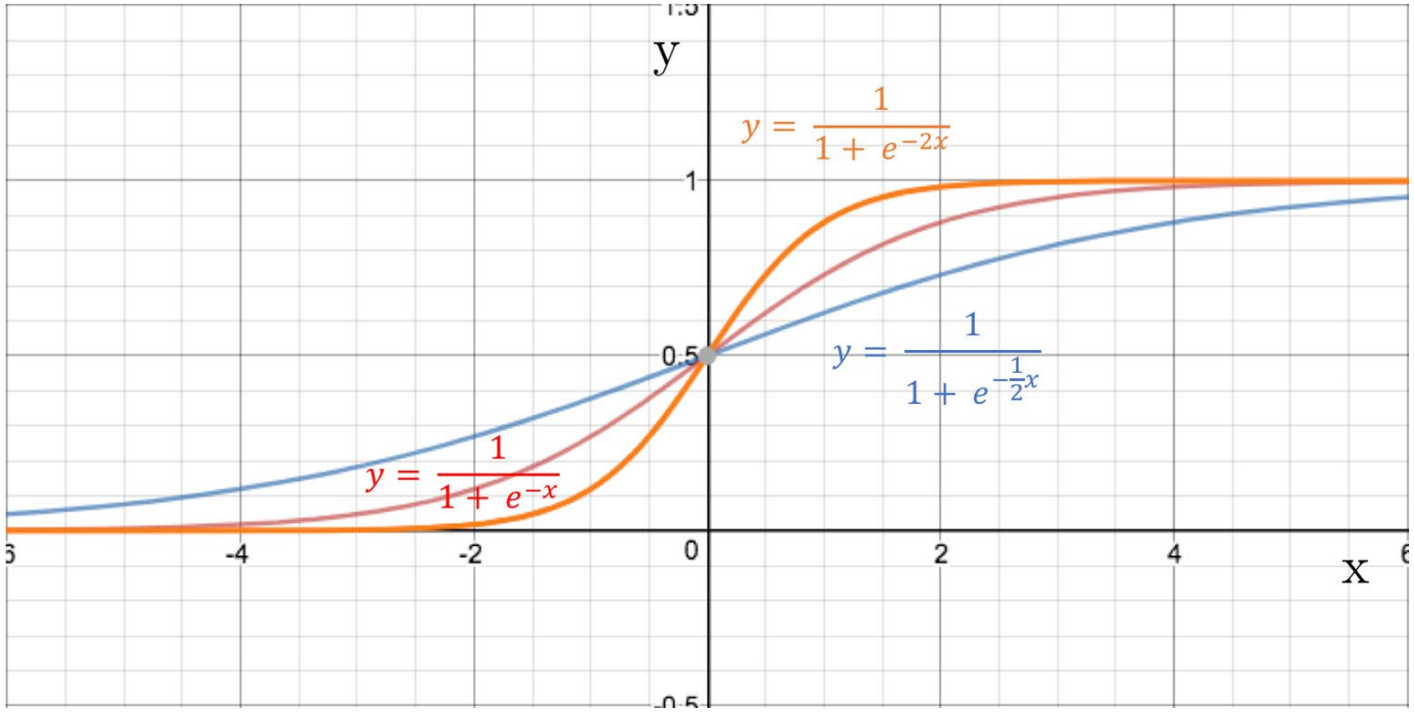
- However depending upon the properties of the problem we might be able to make a better choice for easy and quicker convergence of the network.
  - Sigmoid functions and their combinations generally work better in the case of classifiers
  - Sigmoids and tanh functions are sometimes avoided due to the vanishing gradient problem
  - ReLU function is a general activation function and is used in most cases these days
  - 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

Why try to change Bias / Weights?

- Think of a perceptron as a brain cell
  - A brain cell takes in inputs, and tries to provide the best answer
- The Weights and Bias are analogous to Linear Regression
  - Weights change gradient
  - Bias changes the intercept
- We are "fitting" a curve in each perceptron (cell)
- A neural network is a combination of layers and layers of these cells (like an Ensemble)

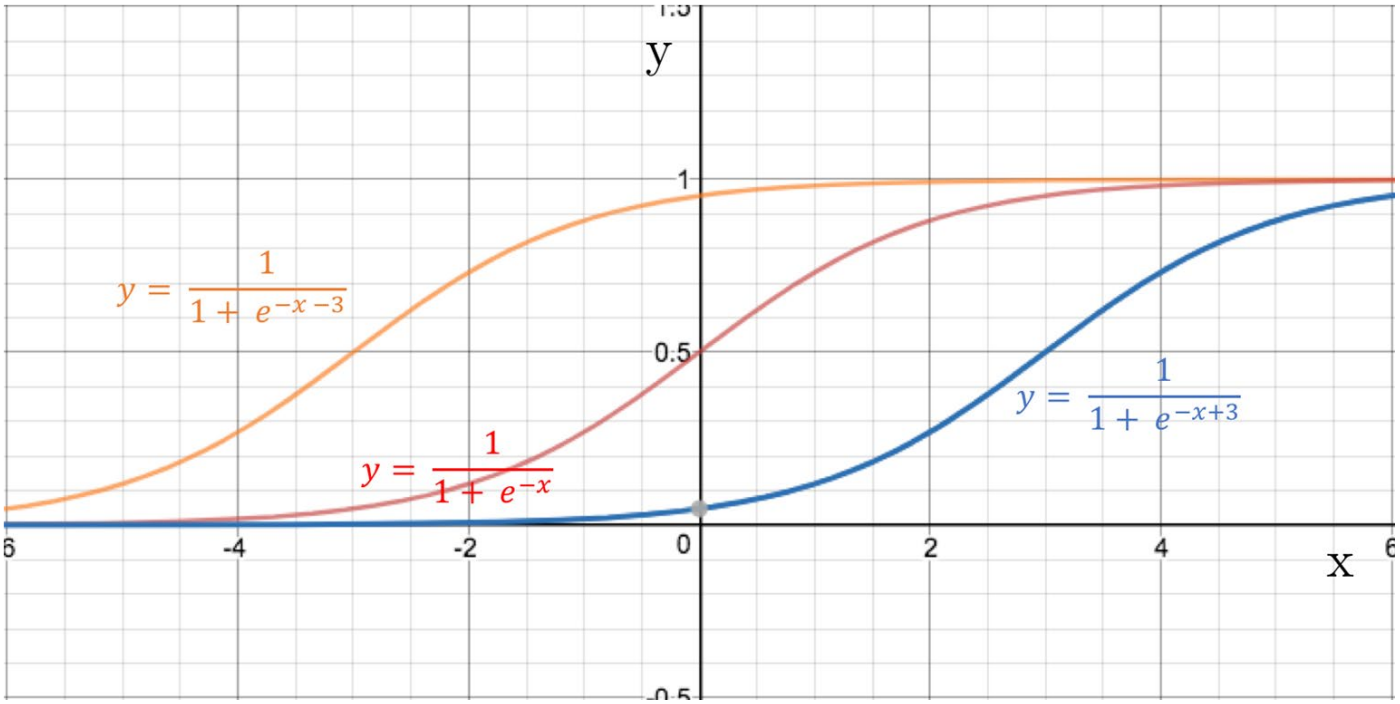
Effect of Weights

Weights increase or flatten the **gradient**

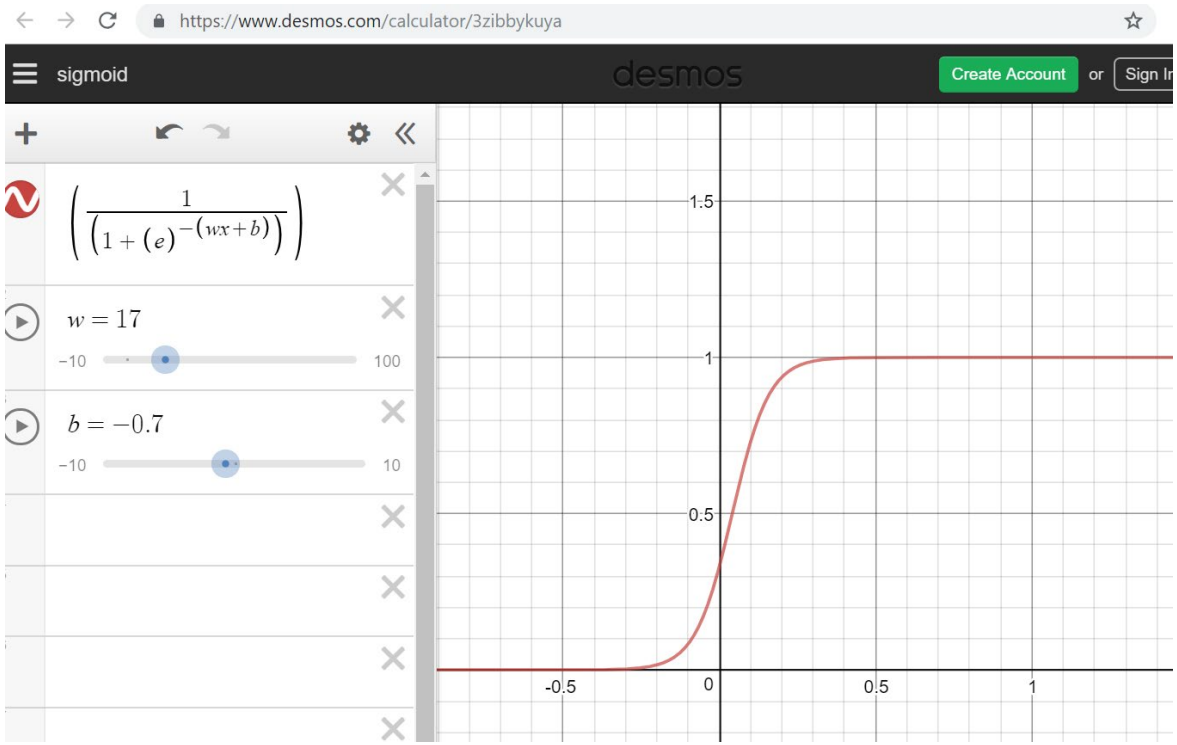


Effect of Bias

Bias shifts the **threshold** left and right



Sigmoid playground



https://www.desmos.com/calculator/3zibbykuya

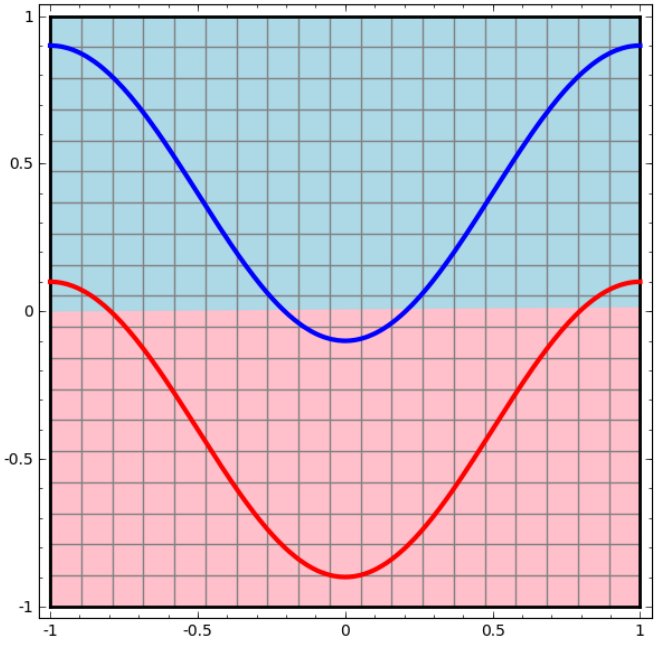
For those who is want to read some deeper reading

Effect of Hidden Layer

Neural Network with only an input layer and an output layer.

Such a network simply tries to separate the two classes of data by dividing them with a line

Source: Chris Olah

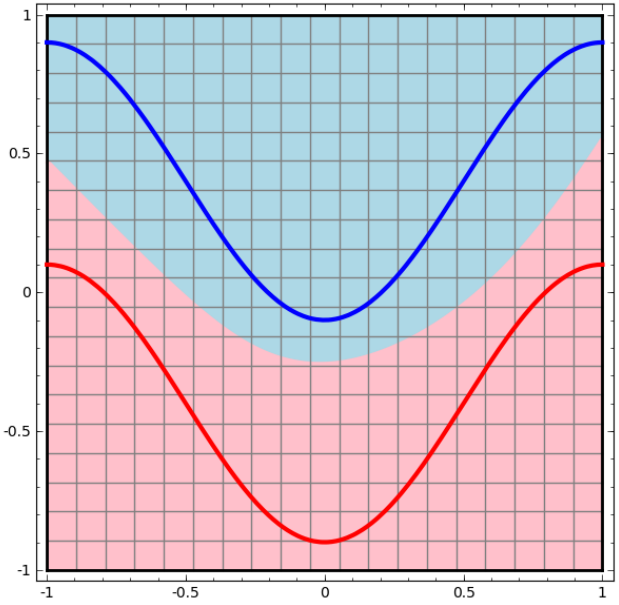


For those who is want to read some deeper reading

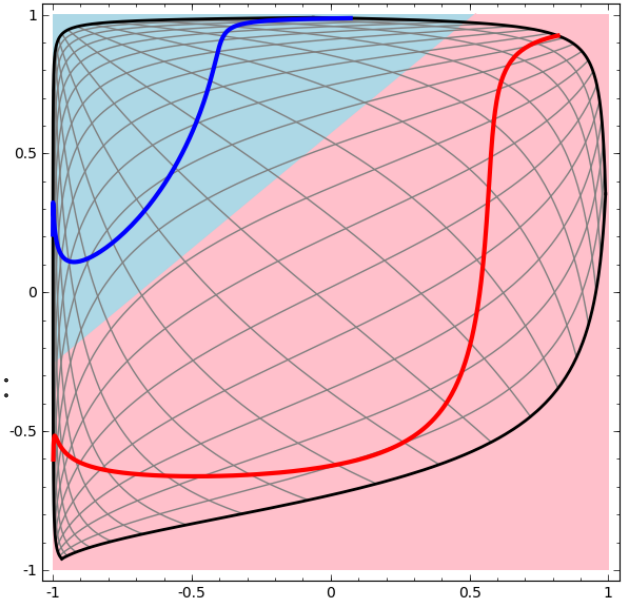
Effect of Hidden Layer

With a hidden layer, the neural network separates the data with a more complicated curve than a line.

Source: Chris Olah



The hidden layer learns a representation so that the data is linearly separable:



Sample, Epoch, Batch

- **Sample**: one row of a dataset
- **Batch**: a set of N samples. The samples in a batch are processed independently, in parallel. If training, a batch results in only one update to the model.
- **Epoch**: "one pass over the entire dataset", used to separate training into distinct phases. Useful for logging and periodic evaluation using validation set

Training a Neural Network

1. Randomly initialise weights
2. For each Epoch
    a. For each Batch
        i. Perform forward pass on each sample, and compute the average error
        ii. Perform backward propagation on batch to compute gradients of Cost function w.r.t. weights
        iii. Update weights using Gradient Descent

Enhancements:

- Early stopping - stop the training when validation loss starts to increase

How many layers, neurons?

- More layers: more generalising capacity
- https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw

One hidden layer is sufficient for the large majority of problems.

So what about size of the hidden layer(s)--how many neurons? There are some empirically-derived rules-of-thumb, of these, the most commonly relied on is '*the optimal size of the hidden layer is usually between the size of the input and size of the output layers*'. Jeff Heaton, author of Introduction to Neural Networks in Java offers a few more.

In sum, for most problems, one could probably get decent performance (even without a second optimization step) by setting the hidden layer configuration using just two rules: (i) number of hidden layers equals one; and (ii) the number of neurons in that layer is the mean of the neurons in the input and output layers.

Hands on References

- Deep Learning interactive playground ← try me!
- Keras
- Keras hello world (Iris dataset and scikit-learn) ← read me!
- Keras tutorials - for Keras practice
- Advanced Keras
- Pre-trained networks in Keras
- Concatenating inputs in Keras

Theory References

- https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/
- Usenet comp.ai.neural-nets deep learning FAQ - covers topics such as how many hidden layers to use, etc.
- Deep Learning book by Ian Goodfellow, et al
- Neural Network design