NEURAL NETWORKS

Dr. Brian Mc Ginley

NEURAL NETWORKS

- Neural networks arise from attempts to model human/animal brains
- Many models and many claims of biological plausibility
- We will focus on multi-layer perceptrons
- Mathematical properties rather than plausibility
  - An Artificial Neural Network (ANN)

NEURAL NETWORKS

- There are problems that are difficult for humans but easy for computers
  - E.g. calculating large arithmetic problems
- And there are problems easy for humans but difficult for computers
  - E.g. recognising a picture of a person from the side
- Neural Networks attempt to solve problems that would normally be easy for humans but hard for computers

THE BRAIN

- Many machine learning methods inspired by biology, e.g., the (human) brain
- Our brain has 10^{11} neurons, each of which communicates (is connected) to 10^{4} other neurons

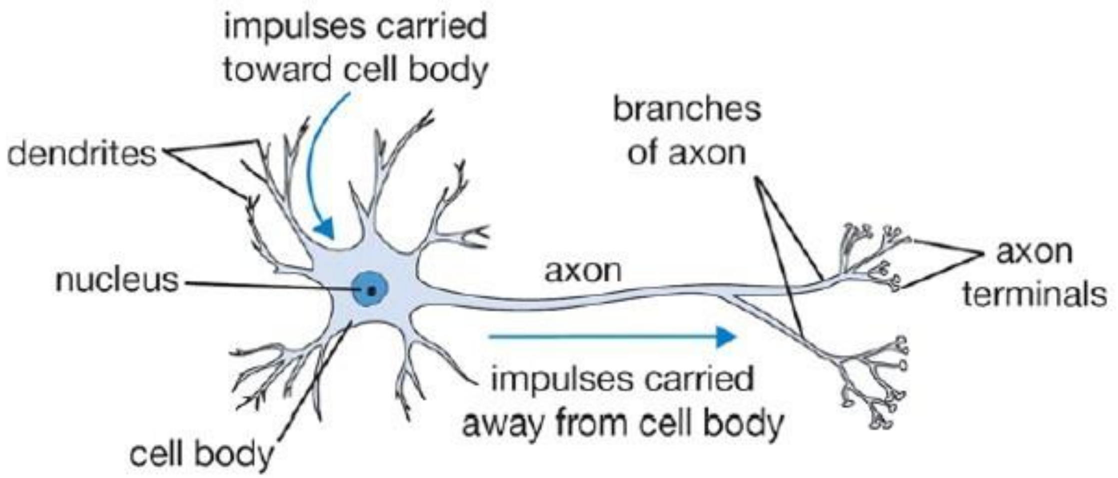


Figure : The basic computational unit of the brain: Neuron

WHAT ARE NEURAL NETWORKS FOR?

- The machine learning algorithms that we have looked at so far are extremely useful and should be the first option you look at for any problem.
- However, when the number of features goes up and the problems become much more complex and non-linear then the previous algorithms can struggle to cope.
- In these cases, a Neural Network can be useful.
- While it may seem that deep neural networks are the only game in town, this is a mistake, and we should always use a simpler algorithm if it will adequately solve the problem.
- Simpler algorithms use less resources and usually have greater causal reasoning.

EXAMPLE

- The MNIST Data set is made up of 60k training samples and 10k test samples of handwritten letters. Each sample is a greyscale image of 28 × 28 pixels. That’s 784 features and the problem is highly non-linear. Imagine trying to find a hyper-plane in 784-dimensional space, that separates each of these out.

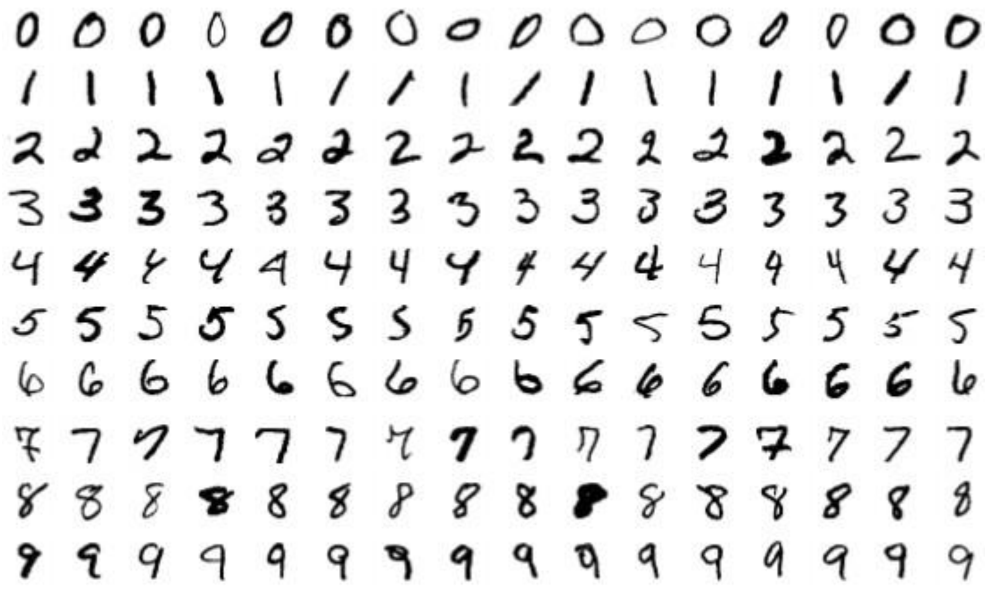


Figure: Samples of MNIST Data Set
Image By Josef Steppan - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=64810040

EXAMPLE

- If we try to model the non- linearity by a polynomial, then the number of features begins to increase very rapidly. So, in cases like this it makes sense to try a neural network.

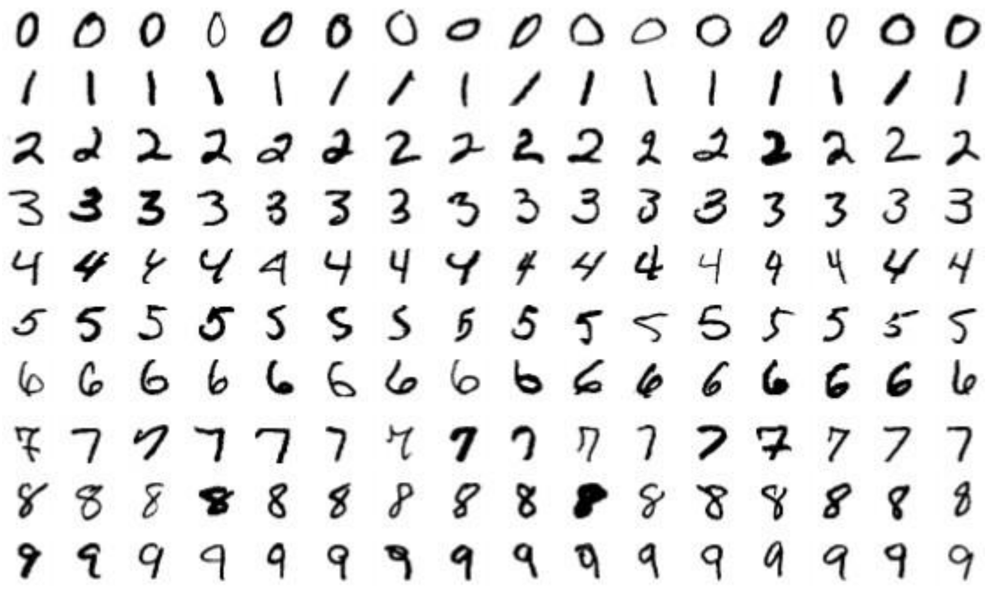


Figure: Samples of MNIST Data Set

Image By Josef Steppan - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=64810040

MATHEMATICAL MODEL OF A NEURON

- Neural networks define functions of the inputs (hidden features), computed by neurons
- Artificial neurons are called units

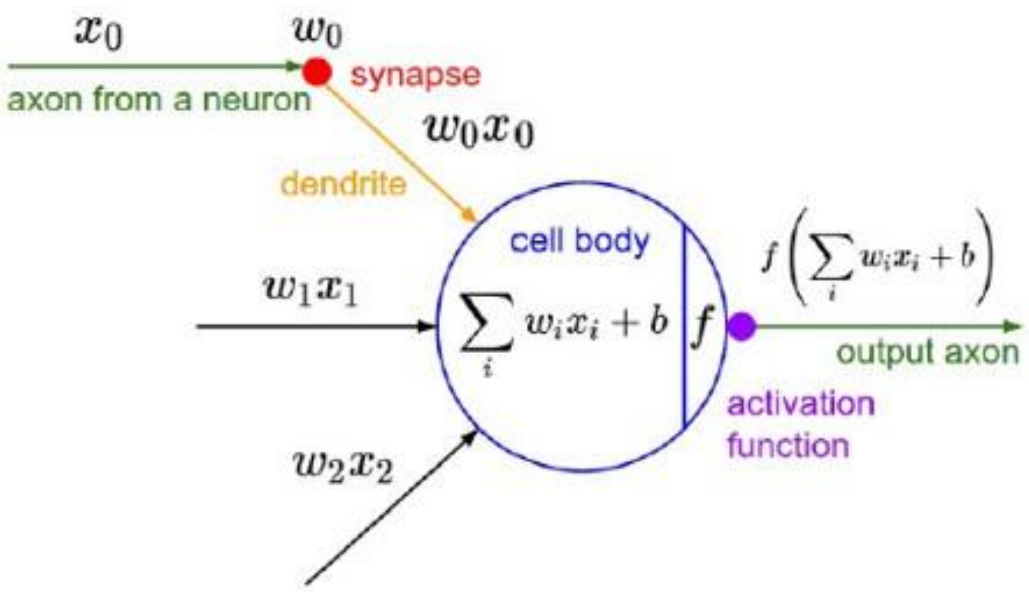
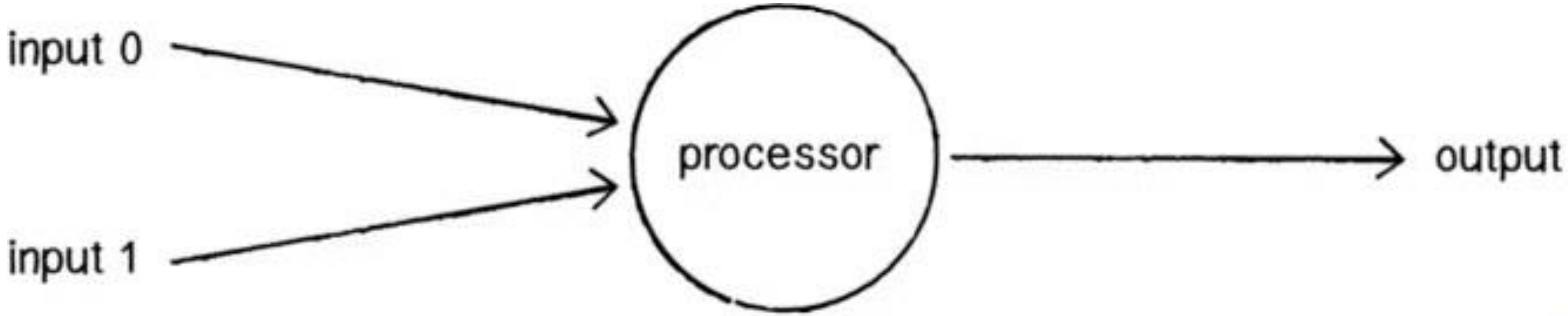


Figure : A mathematical model of the neuron in a neural network

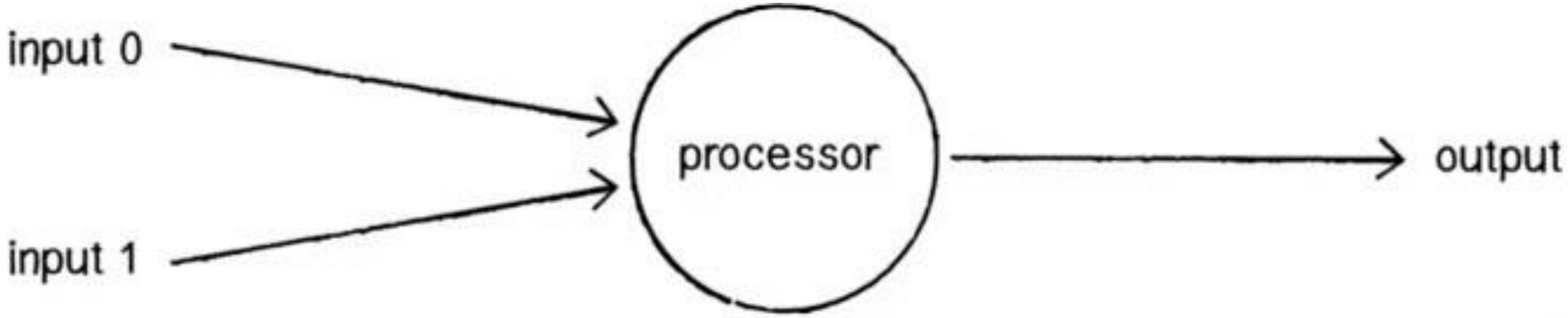
PERCEPTRONS

- We start with the Perceptron (McCulloch–Pitts neuron - 1943)
  - Simulated by Rosenblatt in 1957
- A perceptron consists of one or more inputs, a processor and a single output
- A perceptron follows the ”feed-forward” model, meaning inputs are sent into the neuron, are processed and result in an output



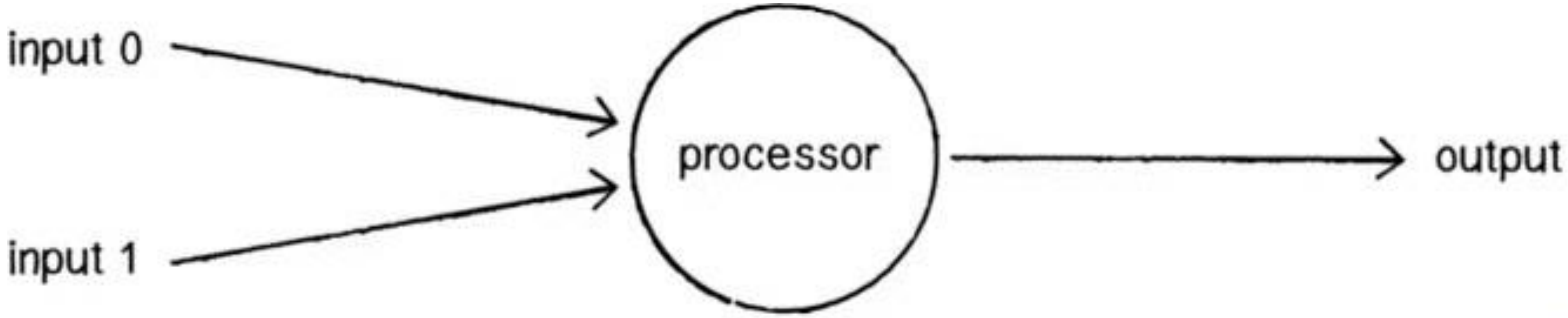
PERCEPTRONS

- Receive Inputs
- Weight Inputs
- Sum Inputs
- Generate Output



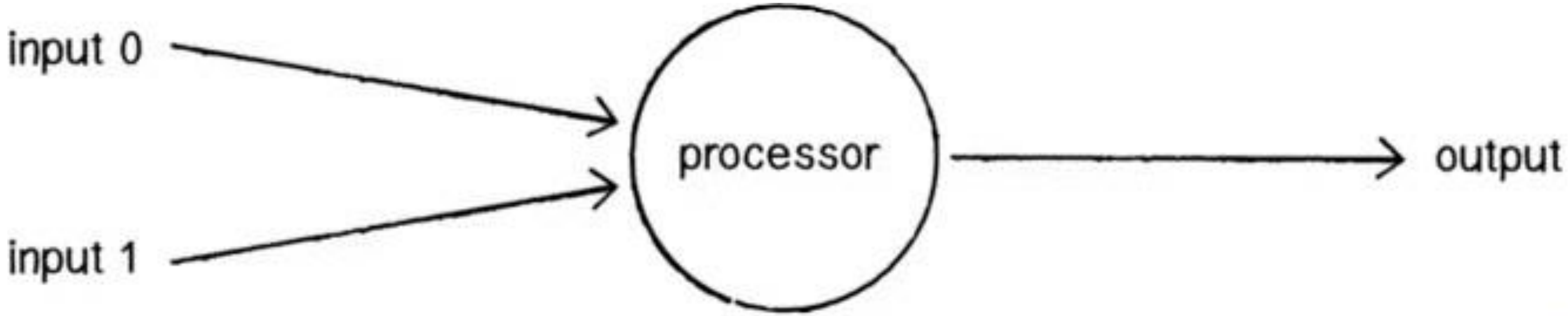
EXAMPLE

- Say we have two inputs x = 12 and y = 4
- Each input that is sent into the neuron must first be weighted
  - Multiplied by some value often between -1 and 1
- Typically, we start with random weights Say w_1 is 0.5 and w_2 is -1
- So Input1*w_1 : 12*0.5 = 6, Input2*w_2: 4*-1 = -4



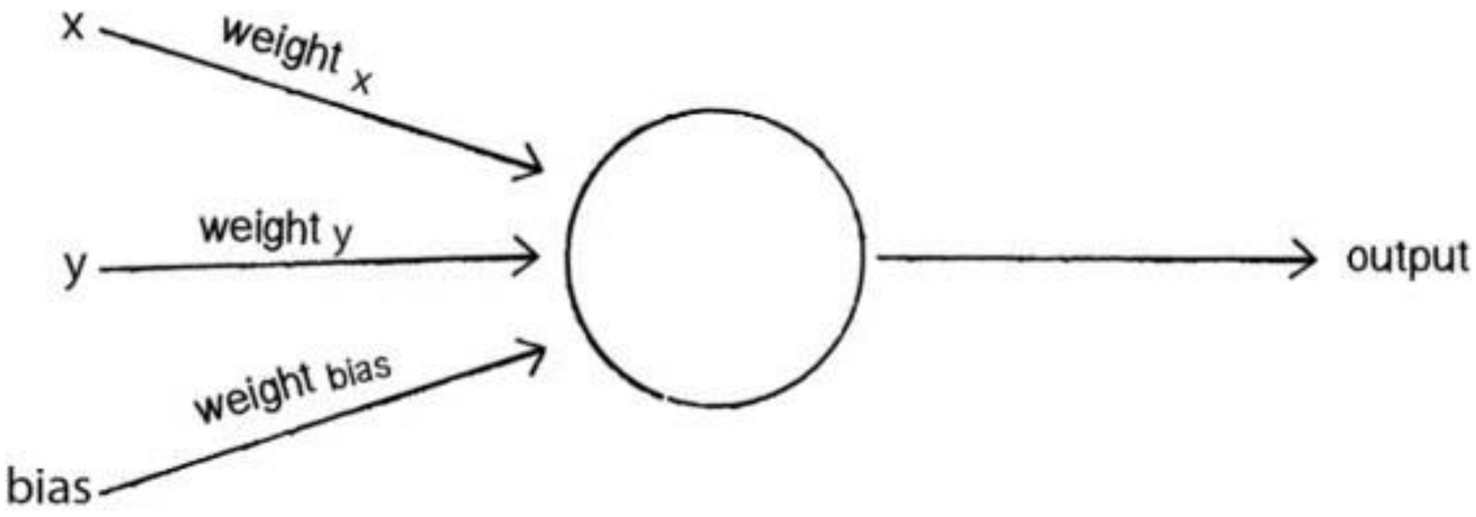
EXAMPLE

- The output of a perceptron is generated by passing that sum through an activation function.
- In the case of a simple binary output, the activation functions is what tells the perceptron to "fire" or not
- Many activation functions to choose from
  - Trigonometric, step, logistic/sigmoid, rbf, relu



BIAS

- One more thing to consider is Bias
- Imagine both inputs were equal to zero, then any sum no matter the weights would also be zero – this is bad
- To avoid this problem, we add a third input known as bias (for now with a value of 1, remember b in previous models…)



PERCEPTRON

- To actually train a single perceptron we do the following, we initialise it with random weights - remember:
  1. Provide the perceptron with inputs for which there is a known answer.
  2. Ask the perceptron to "guess" the answer
  3. Compute the error
  4. Adjust all the weights according to the error
  5. Goto 1 and repeat
     - Repeat until we reach an error we are satisfied with (set beforehand)
- This is how a single perceptron would work.

HISTORY

- We start with the Perceptron (McCulloch–Pitts neuron - 1943)
  - Simulated by Rosenblatt in 1957
- 1969: Minsky and Papert showed that a single layer of perceptrons was incapable of solving the XOR problem.
  - Even though Minsky and Papert knew that a multi-layer perceptron network could do the job, the paper caused a huge decline in interest and funding in ANNs for >10 years
- 1970 - Backpropagation (for training multiple layers) was discovered by Finnish Masters student Seppo Linnainmaa.
  - Algorithm not discovered again until 1980s by Rumelhart (1982) and Hinton (1986) – kickstarting new research in the field

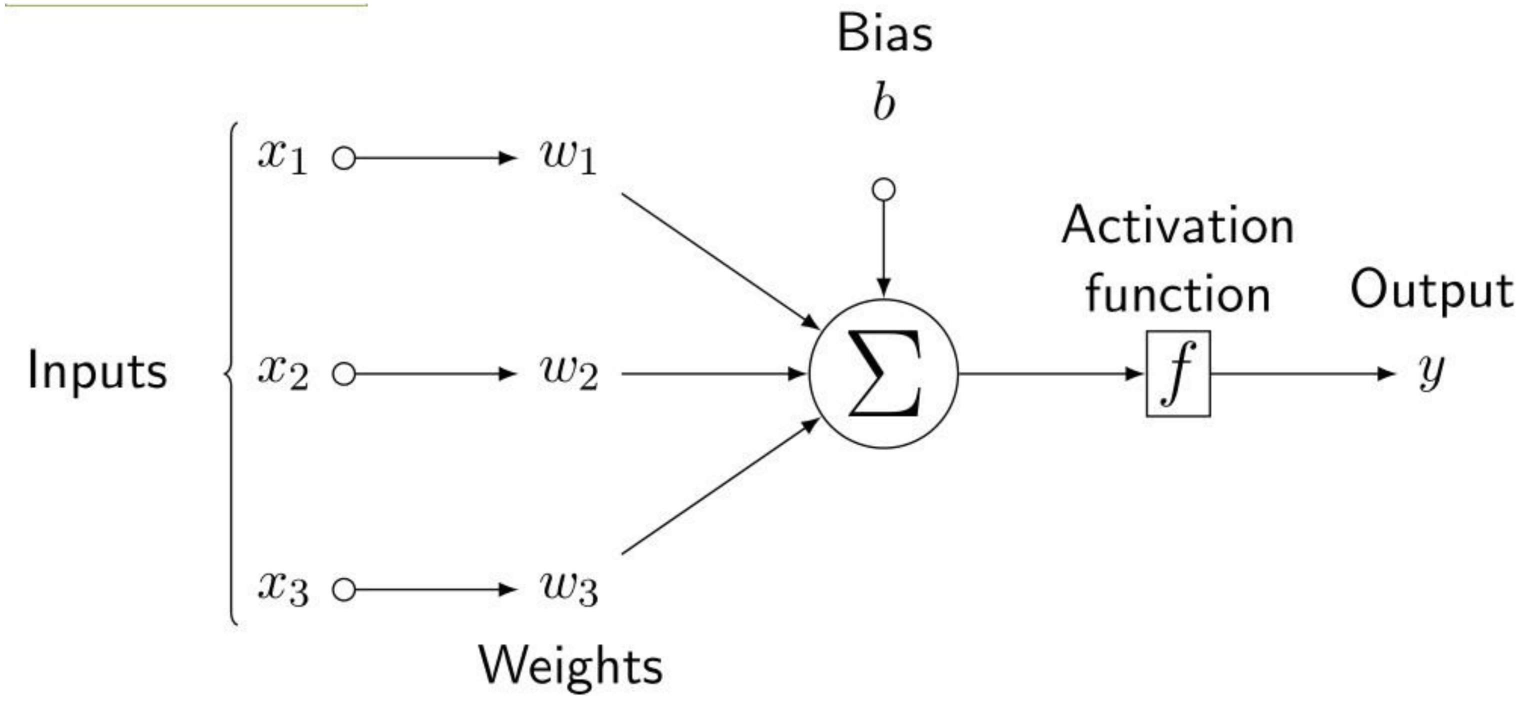
MULTI-LAYER PERCEPTRON

- To create a neural network you link many perceptrons together in layers multi-layer perceptrons (feed-forward)

```
from sklearn.neural_network import MLPClassifier import  tensorflow as tf
```

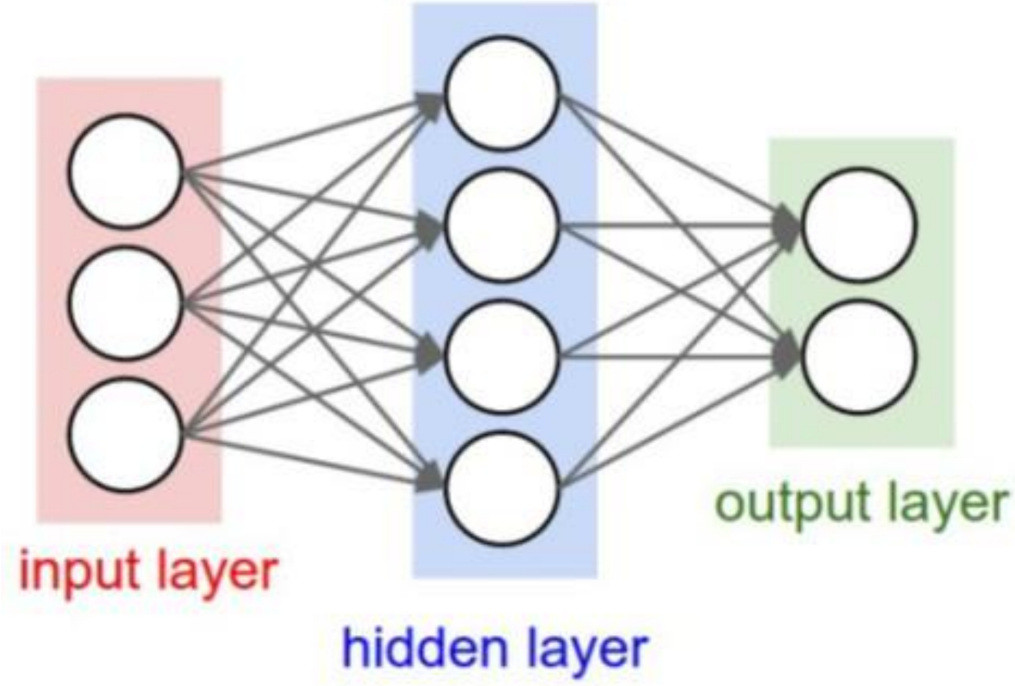
PERCEPTRON - NICER PICTURE?

- The Perceptron, AKA Artificial neuron, is the building block of an artificial neural network.



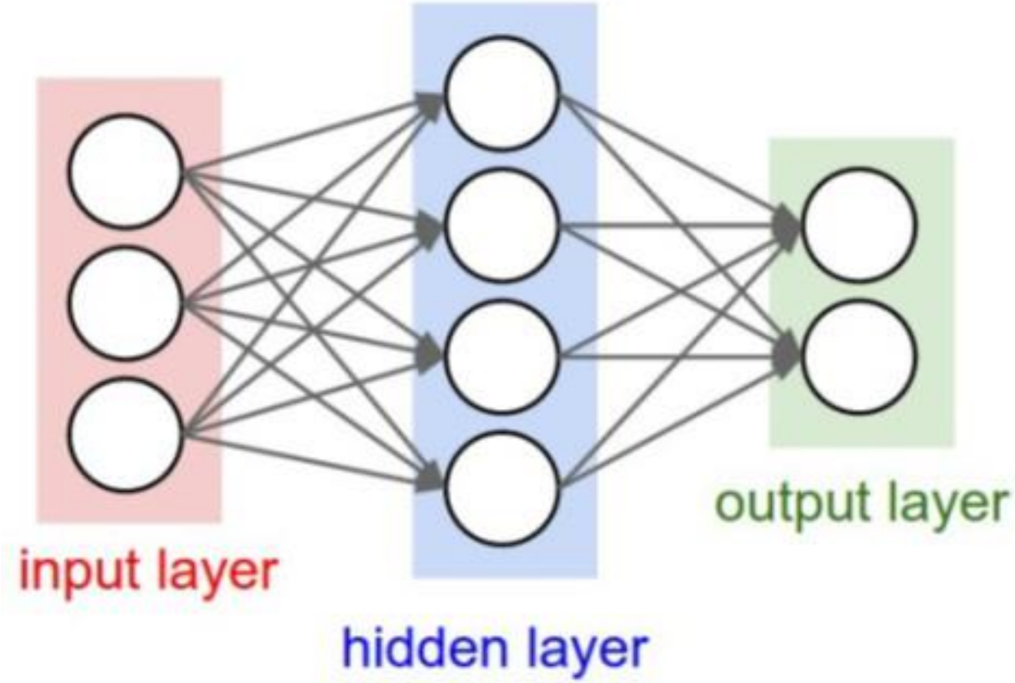
NEURAL NETWORKS

- You'll have:
  - an input layer
  - an output layer
  - And some layers in between, known as hidden layers
- They are called hidden layers because you don't directly "see" anything but the input or output



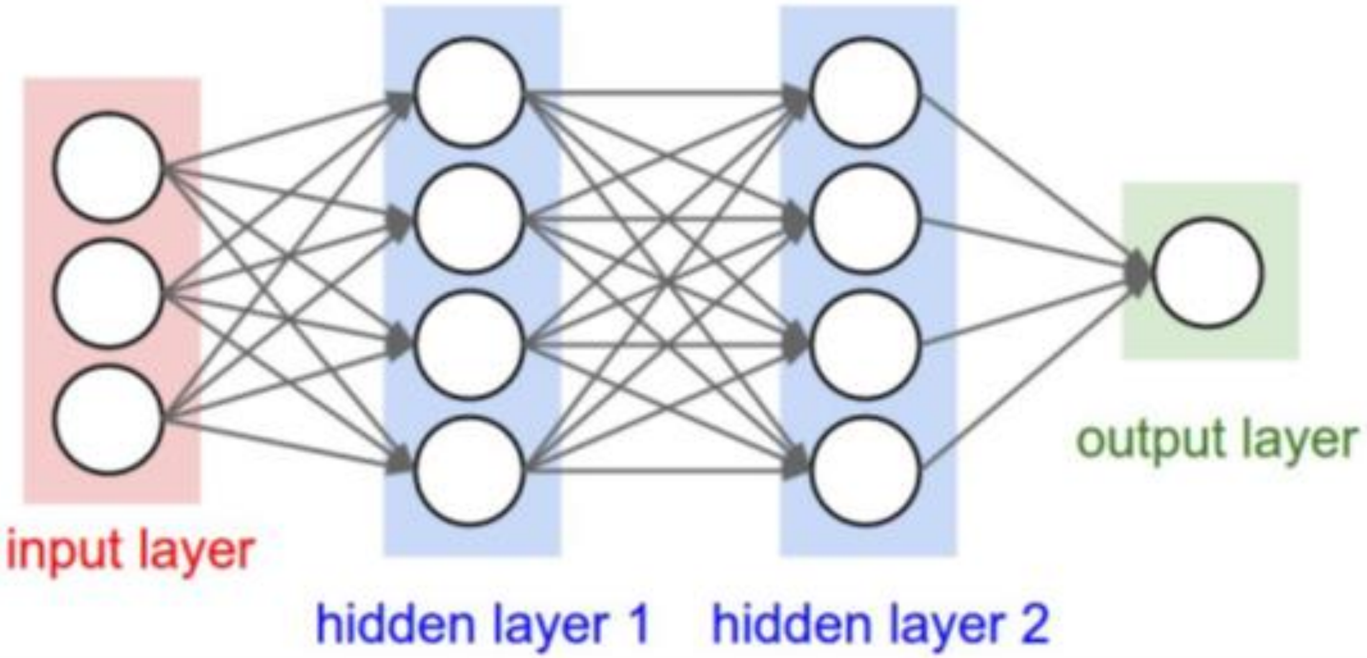
NEURAL NETWORK ARCHITECTURE (MULTI-LAYER PERCEPTRON)

- Naming conventions: We have a N- layer neural network
  - N−1 layers of hidden units
  - One output layer
  - Input does not count as a layer
- This example is a 2-layer neural network with 3 input units (features), 4 hidden units (in the one hidden layer) and 2 output units



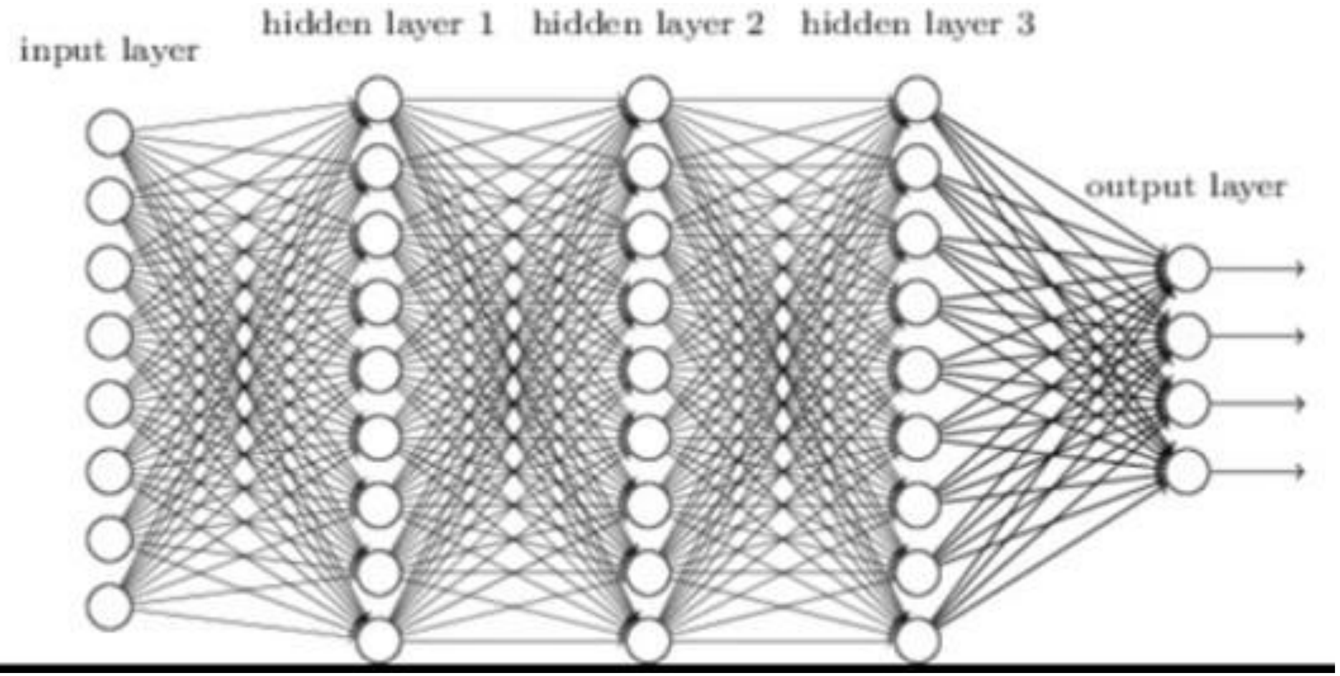
NEURAL NETWORK ARCHITECTURE (MULTI-LAYER PERCEPTRON)

- Going "deeper": a 3-layer neural network with two layers of hidden units



DEEP LEARNING

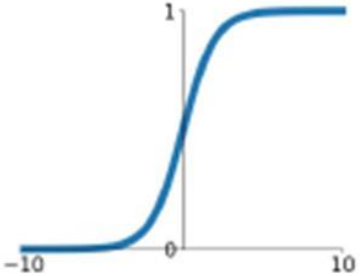
- Have you heard of the term "Deep Learning"?
- This is a current buzzword.
- It is really just a Neural Network with many hidden layers, causing it to be "deep"
- Microsoft's vision recognition uses 152 layers



Activation functions

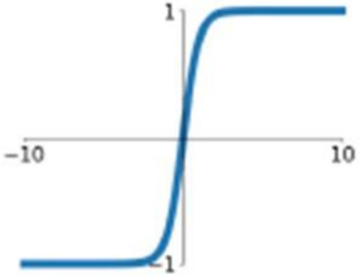
Sigmoid

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



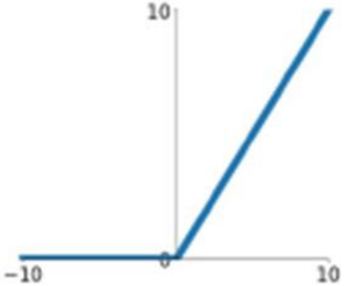
tanh

$\tanh(x)$



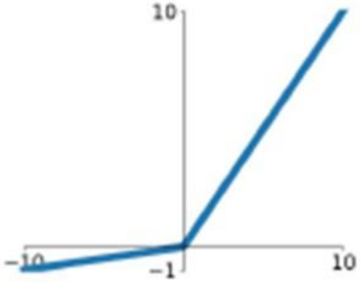
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

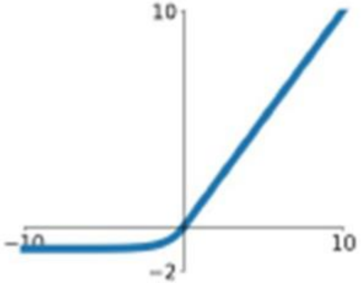


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

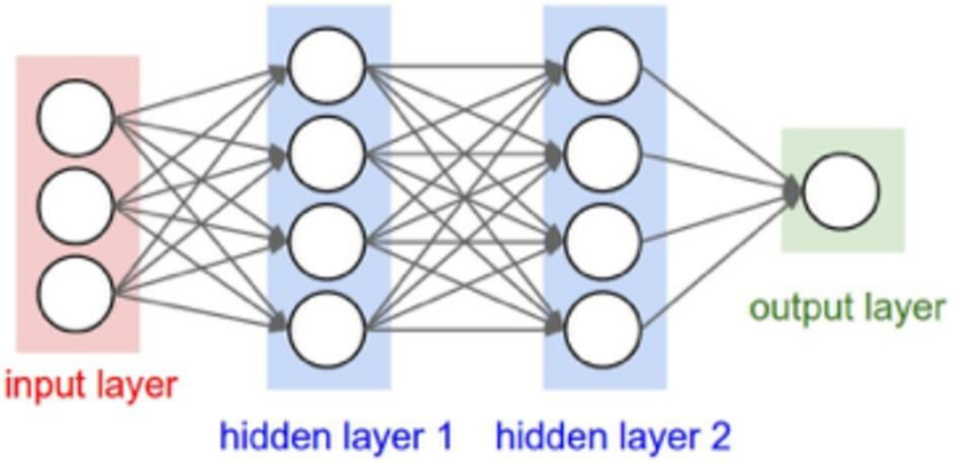
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



PYTHON CODE

- Can be implemented efficiently using matrix operations

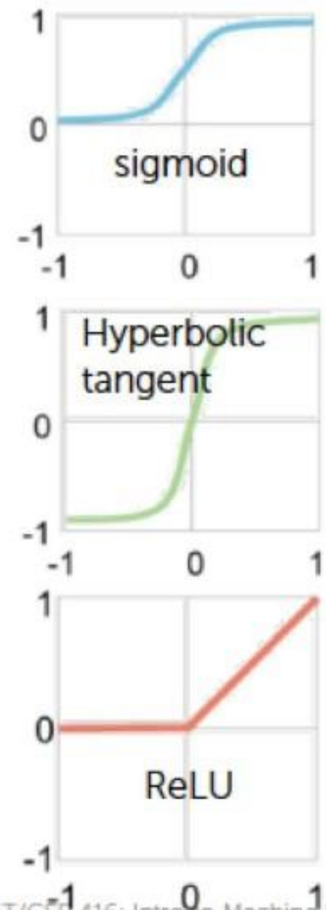
Example Feed-forward computation of a Neural Network




```python
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

WHICH ACTIVATION FUNCTION?

- Sigmoid
  - Historically popular, but (mostly) fallen out of favor
    - Neuron's activation saturates (weights get very large → gradients get small)
    - Not zero-centered → other issues in the gradient steps
  - When put on the output layer, called "softmax" because interpreted as class probability (soft assignment)
- Hyperbolic tangent $g(x) = \tanh(x)$
  - Saturates like sigmoid unit, but zero-centered
- Rectified linear unit (ReLU) $g(x) = x^+ = \max(0,x)$
  - Most popular choice these days
  - Fragile during training and neurons can "die off"... be careful about learning rates
  - "Noisy" or "leaky" variants
- Softplus $g(x) = \log(1+\exp(x))$
  - Smooth approximation to rectifier activation



RECTIFIED LINEAR UNIT (RELU)

- The Rectified Linear Unit is now the most popular activation function.
- The reason for this, is that it is the most-simple but still seems to work well in practice.
- Alex Krizhevsky showed, in the paper that won the ImageNet challenge in 2012, that using ReLU instead of tanh allowed the model to converge six times faster.
- Also, some suggestion from neuroscience that this is most similar to what biological neurons do.

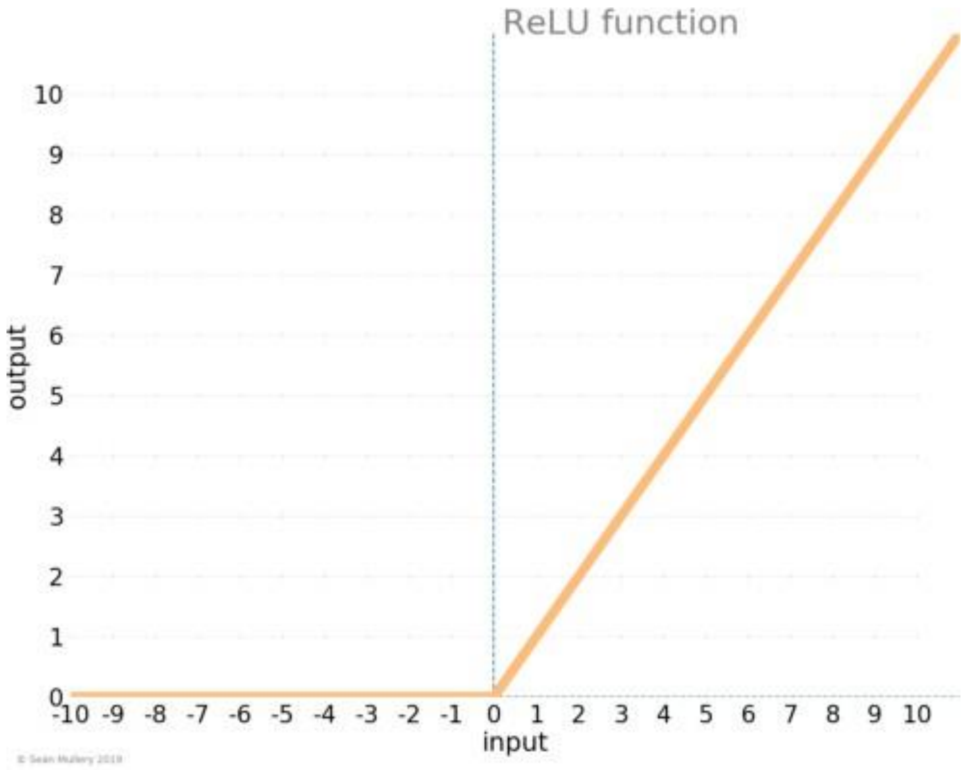


Figure: Rectified Linear Unit

IN PRACTICE

- Use ReLU.
- Be careful with your learning rates
- Neurons can "die" using ReLU (I will go into more detail later)
- Try out Leaky ReLU (should not "die")
- Try out tanh but don't expect much
- Some will say don't use sigmoid anymore

INTERACTIVE DEMO

- https://playground.tensorflow.org/
- Use XOR dataset
- Explore impact of
  - Number of hidden units
  - Activation function

NEURAL NETS

- How to train Neural Nets?
  - Set up a loss function
  - Apply Gradient Descent
- Procedure
  - Forward - Computes Loss
  - Backward - Calculate Gradient
  - Update - Uses Gradient to increment weights