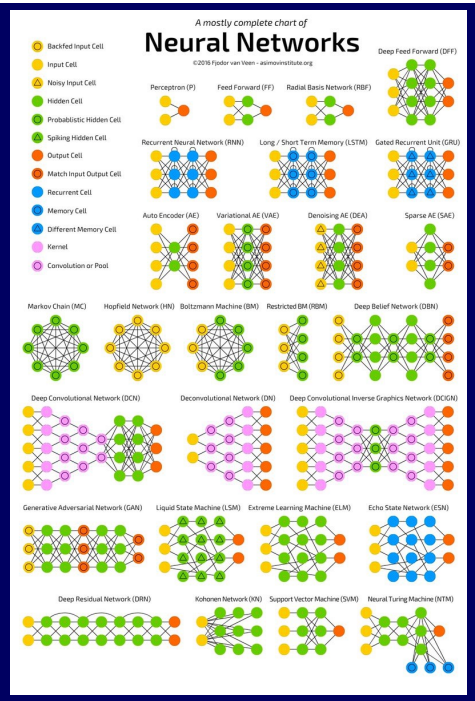
ID1214 AI
Artificial Neural Networks and Deep Learning



The Neural Network Zoo
https://www.asimovinstitute.org/neural-network-zoo/

Anne Håkansson

Professor in Computer Science with specializiation in Intelligent software systems, KTH Royal Institute of Technology, Sweden

Professor in Computer Science with focus on AI, UiT The Arctic University of Norway, Tromsø, Norway

Contents Part: 1 (2)

Part 1:

- **What is an Artificial Neural Network?**
  - **Applications**
  - **Terminology**
  - **Training process of Neural Networks**
- **Internally in Neural Networks**
  - **Weights**
  - **Bias**
  - **Activation level / Threshold**
  - **Activation functions**
  - **Measure and Stop the Epoch**

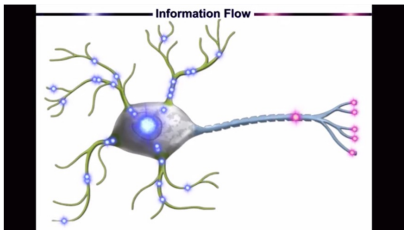
Part 2:

Feed-Forward neural networks

Normalize data

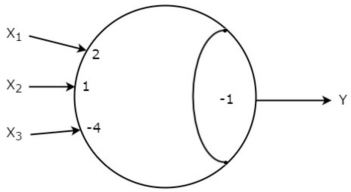
- Primary algorithms; supervised Feed-Forward with Back-propagation
  - Unsupervised / Reinforcement
  - Deep Neural Networks & Deep Learning
- Problems
- Readings

Artificial Neural Network (ANN)

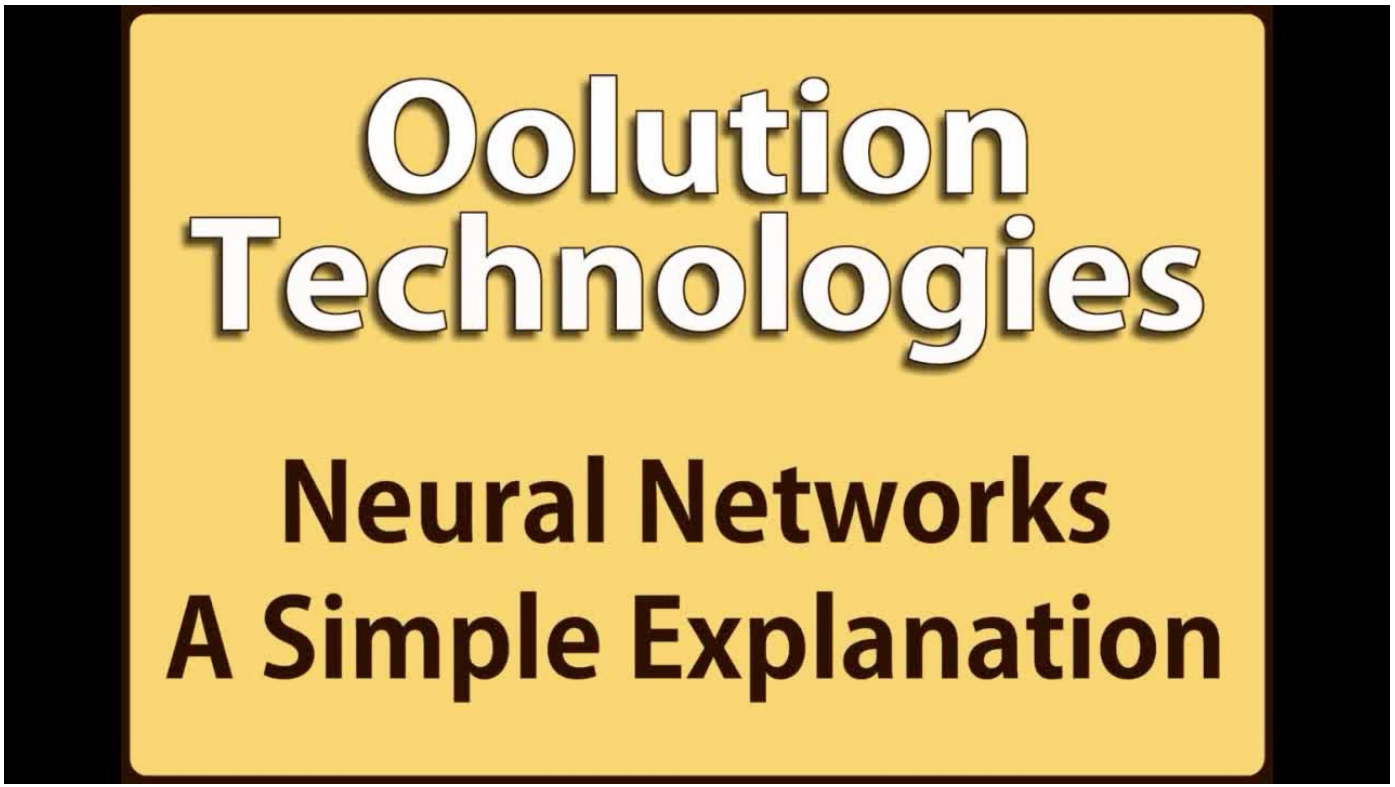


- Inspired by real (biological) neural parts - Based on biological synapses and neurons

- *McCulloch & Pitts - 1943* - created a computational model for neural networks called threshold logic (e.g., Digital Circuits)
- *1950 - a neural network, S N A R C,* (Marvin Minsky and Dean Edmonds) a connected network of approximately 40 (Hebb) synapses (mini-brain with memory).
- Single-layer Perceptron ***McCulloch-Pitts neuron***
- Just a “Fancy math equation”
- Vocabulary in NN - very chaotic



Biological Simulation
Technology Architecture



https://www.youtube.com/watch?v=gcK_5x2KsLA

ANN Good for, e.g.,:

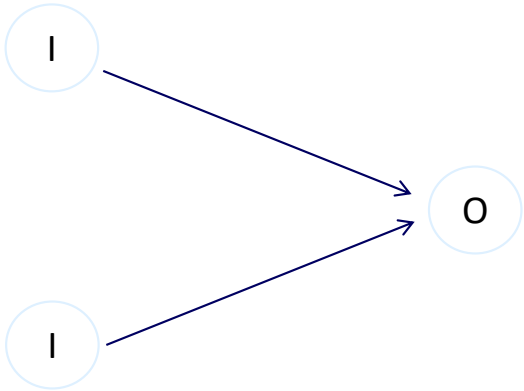
- To *infer a function from observations*.
- *Useful for* tasks where *design by hand is impractical* (complexity of the data*)*.
- *Function approximation* – prediction via regression analysis *including time series prediction*, fitness approximation and modeling
- *Classification*, including pattern and sequence recognition, novelty detection and sequential decision making
- *Clustering with Data processing*, including filtering, blind source separation and compression

When can you use this… ANN / DL

- Predict which team will win a game (e.g., soccer)
- Predict stock market - whether a company's stock price will go up or down on any given day
- Medicine,
- Electronic Nose, Security, complex applications
- Control, including Computer numerical control
- Robotics, including directing manipulators
- Black and white images (hand-writing) - ANN
- Colour images – DL

Machine learning – only one single layer (1950s)

AND - OR



” **Single-layer perceptron**”

- *One single layer - the perceptron = an algorithm for supervised learning of binary classifiers (Rosenblatt, 1958).*

- *decides whether or not an input) represented by a vector of numbers) belongs to some specific class.*
- *Circuit boards – AND, OR - gates*

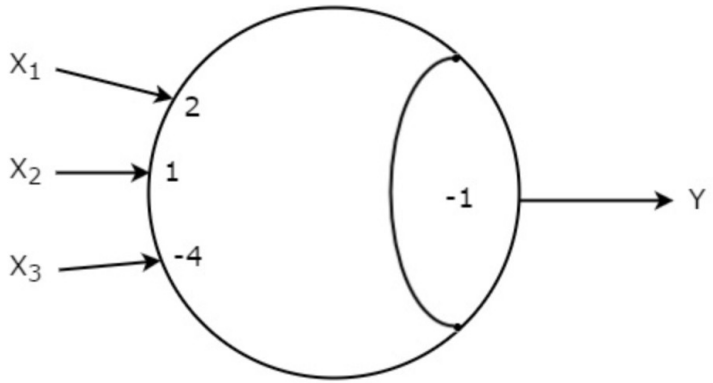
For more logic gates, see:

https://learnabout-electronics.org/Digital/dig21.php

Perceptron – multi-layer perceptron

A Perceptron Is:

- algorithm
- processing unit
- linear classifier
- neural network



- a single-layered network - algorithm: binary (two-class) classification machine learning algorithm

- a single neuron: used for two-class classification problems and provides the foundation for later developing much larger networks.

Perceptron Algorithm

- *Inspired* by neuron - the information processing of a single neural cell
- N*euron:* accepts input signals (via its dendrites), which pass the electrical signal down to the cell body.

- Similarly, the *Perceptron:* 1) receives input signals from examples of training data 2) add weight and 3) combine these in a linear equation (this equation is called the activation).

-> the Perceptron becomes a classification algorithm for two classes (0 and 1)

-> linear equation can be used to separate the two classes)

The Perceptron algorithm is the *simplest type of artificial neural network*.

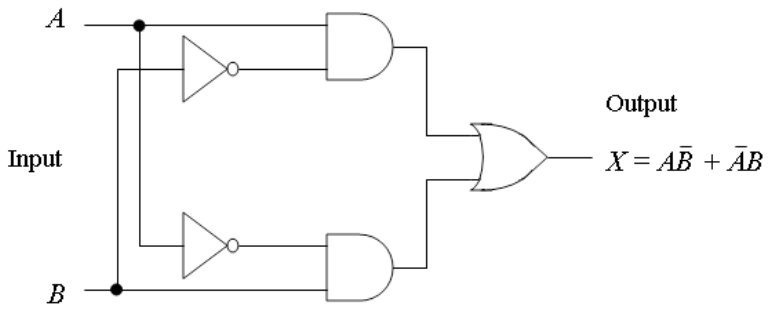
It is closely related to linear regression and logistic regression -> make predictions in a similar way (e.g., a weighted sum of inputs).

Single-layer perceptrons - Not good for XOR

If the training set D is not linear - > it will never get to the state with all the input vectors classified correctly.

https://machinelearningmastery.com/implement-perceptron-algorithm-scratch-python/

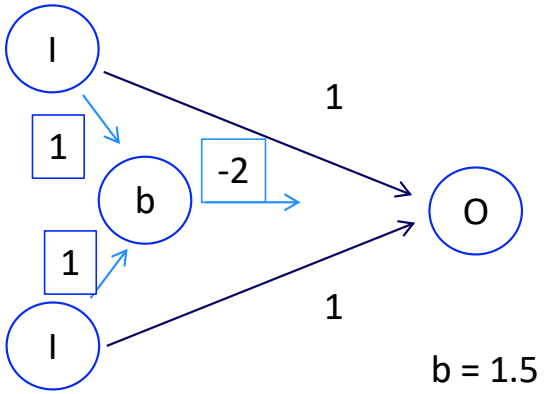
Why Multiple-layers?



XOR – Need *Multiple components* to achieve the XOR logic

XOR gate

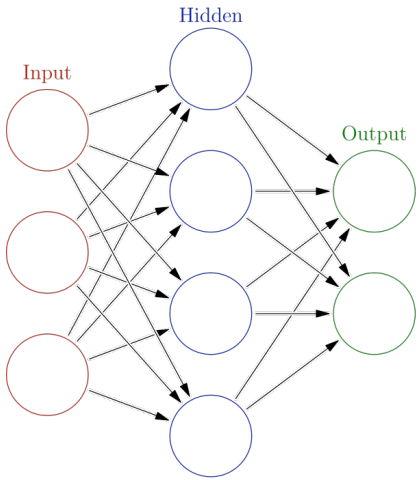
Input 1	Input 2	Output
	1	1
1		1
1	1	



Handle complex problems

” **Multiple-layer perceptron**”

Artificial Neural Network (ANN)



Circular node represents an artificial neuron
Arrow represents a connection from the output of one neuron to the input of another.

- An artificial neural network is an interconnected group of nodes (perceptrons)
- Each node - represents a neuron
- Layers: Arrange Neuron, *one input layer and one output level – AND – one to several hidden layers (*typically)
- The neurons from one layer are connected to neurons in other layers
- Connections between neurons are weighted
  -> By *modify these weights*, ANN can perform complex classification tasks.

Size image = 28x28 pixels ==> 784 pixels ==> input (data)
MNIST data set over handwritten letters
0 = black, 255 = white (grey scale 1-254)
For MNIST, there are 10 digits, so the output layer has 10 neurons.

Input
(from *up left*)
0,
0,
0,
0,
0,
0,
.
.
98,
126,
189,
135,
87,
0,
.
.
91,
143,
0
0
0
.

Output network
0.2,
0.1,
0.1,
0.1,
0,9
0.1,
0.2,
0.5,
0.3,
0.6

Desired label
0,
0,
0,
0,
1,
0,
0,
0,
0,
0

Classification - MNIST two files:
images — file with all pictures
Labels — file with all labels

Output == number of letters
0-9

Size image = 28x28 pixels ==> 784 pixels ==> input
0 = black, 255 = white (grey 1-254)

Input
(*up left*)
0,
0,
0,
0,
0,
0,
.
.
87,
125,
156,
180,
185,
184
159,
122
95,
.
.
0
.

Output	**Desired label**
0.09,	0,
0.1,	0,
0.2,	0,
0,9	**1,**
0.2,	0,
0.4,	0,
0.2,	0,
0.3,	0,
0.5,	0,
0.	0

Output == number of letters
0-9

Size image = 28x28 pixels ==> 784 pixels ==> input
0 = black, 255 = white (grey 1-254)

Input
(*up left*)
0,
0,
0,
0,
0,
0,
.
.
87,
125,
156,
180,
185,
184
159,
122
95,
.
.
0
.

DEFINE THE NETWORK WITH INPUT NODES OUTPUT NODES AND A NUMBER OF HIDDEN LAYERS WITH NUMBER OF NODES

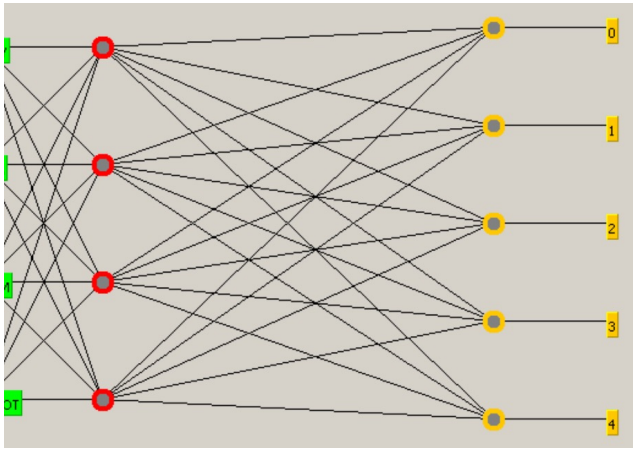
Output
0,
0,
0,
1,
0,
0,
0,
0,
0,
0

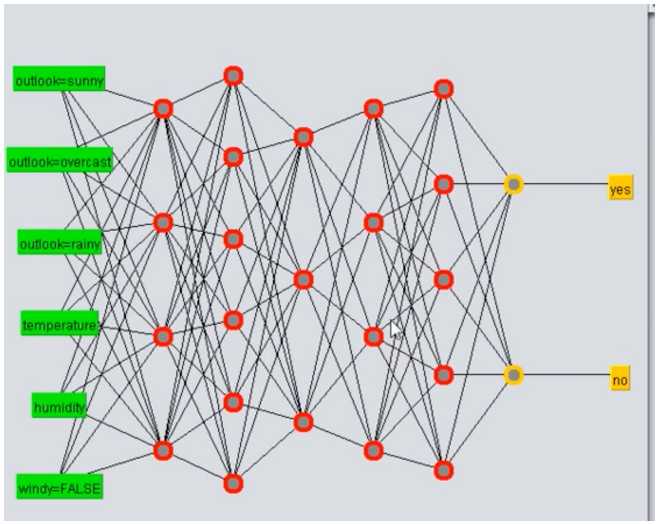
Output == number of letters 0-9

MNIST dataset – 60,000 training images and 10,000 testing images

WEKA

DEFINE THE NETWORK WITH INPUT NODES
OUTPUT NODES AND A NUMBER OF HIDDEN LAYERS WITH NUMBER OF NODES





Number of neurons in hidden layer?

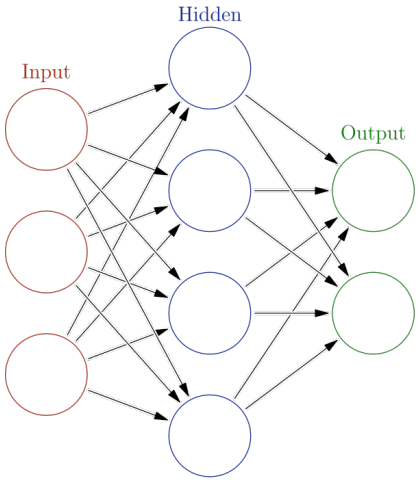
Question:

1. How do we know how many neurons we should have in the hidden layer?

2. How many hidden layers?

The answer: – we don't know!

Need to elaborate… to get the best output



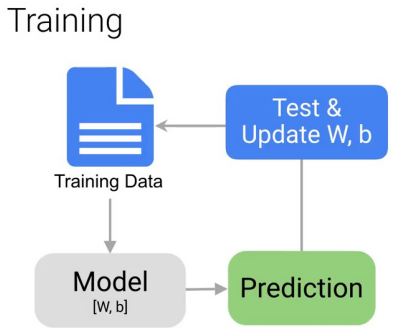
Terminology

Neural Nets Lecture >>
https://www.youtube.com/watch?v=uXt8qF2Zzfo

- X – Input
- Y – Output
- Hidden layer(s) – a layer of nodes (not connected to the *environment* but good for dividing complex tasks)
- W – Weights, a constant between [–b, b], *commonly* -1 to 1
- T – *Threshold* decides if a neuron will become active
- Bias – used for setting the activation level (Threshold), together with the sum of input perceptrons
- *How well we are doing* = Compare desire value with actual value

Computed			Desired(Label)				Correct?
0.1	0.9	0.0	0	1	0		*yes*
0.3	0.4	0.3	0	0	1		*no*

ANN – training process



A training step

- A training step – ANN

1. Get training data

input features (X) och **target labels (Y)**.

2. Model – (W - Weight, b - bias) $\text{output} = f(W \cdot X + b)$

3. Prediction – for input(X) $y_{\text{pred}} = f(W \cdot X + b)$

4. Test and Update W, b (b – only if needed)

$$\text{loss} = L(y_{\text{pred}}, y_{\text{true}})$$

Implement Neural networks, LAB

Choose dataset

IF choose digits data set – use MNIST sample

Step 1: Import all the required libraries

Step 2: Load the MNIST data-set

Step 3: Create the Neural Network by training

- Layers are the most important building blocks
- Number of nodes (perceptrons)
- Select activation function

```
>>> train_images.shape
(60000, 28, 28)
>>> len(train_labels)
60000
>>> train_labels
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

And here's the test data:

```
>>> test_images.shape
(10000, 28, 28)
>>> len(test_labels)
10000
>>> test_labels
array([7, 2, 1, ..., 4, 5, 6], dtype=uint8)
```

Basic types of activation functions:

Sigmoid
Hyperbolic Tangent
Heaviside (Unit) Step
ReLU - Rectified Linear Units

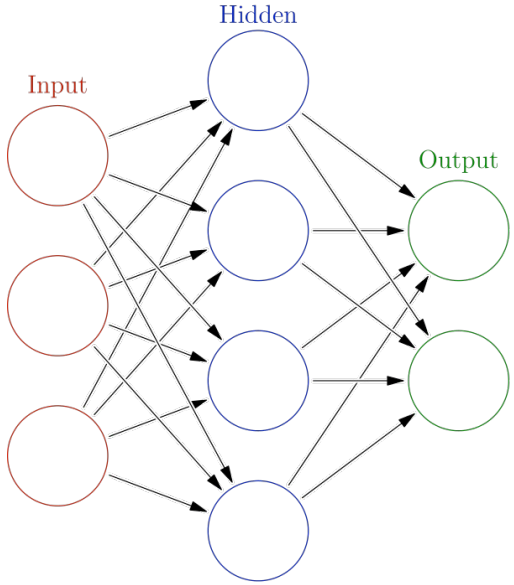
https://towardsai.net/p/machine-learning/how-to-build-and-train-your-first-neural-network-9a07d020c4bbs

Model - Training process of Neural Networks

1) The network receives inputs (classified or clustered)

2) Apply the algorithm - Some neurons are fired

=>Training the model



Single-hidden layer :

Fired neurons pass signals to *output*

Multi-layer network:

Fired neurons pass signals to *next layer's* neurons

– Analogy: strengthens or weakens the electric impulses

⇒ a complex pattern arranged throughout the network

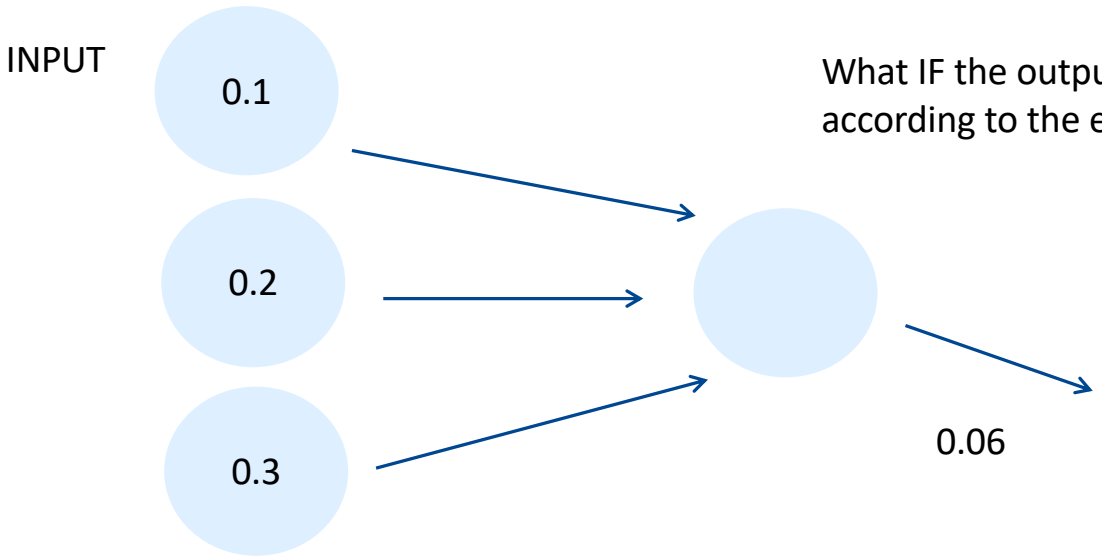
- Final result – some output neurons are fired

Computed			Desired(Label)			Correct?
0.1	0.9	0.0	0	1	0	*yes*
0.3	0.4	0.3	0	0	1	*no*

TRAIN ???

How do the different inputs relate to each other?

What IF the output is WRONG according to the expectation?



Internally in Neural Network

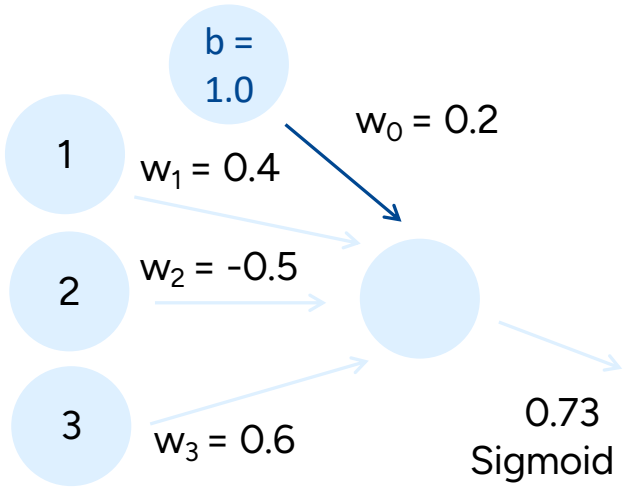
Perceptron (neuron) (a basic circle in a diagram)

Supervised training – known output

- Neurons “signal feed into other neurons”

w = is a constant, randomly produced

y = mx + b



- Activation level:

1) Weights

2) Input Calculation = (1*0.4) + (2*-0.5) + (3*0.6) = **1.2**

***b = bias*:** constant to support the activation level (threshold),

very important to make it work = fire a neuron or not (keep in range of hyperspace) = 1.0 * 0.2 = 0.2

3) Total sum of the inputs + bias: 1.2 + 0.2 (bias) = 1.22

4) Activation

3) Weights

E.g.,: W > 0 enhances, excites the firing w < 0 inhibits the firing

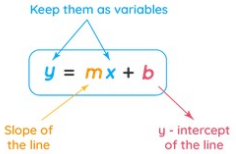
Weights are *randomly* initialized and distributed in the network

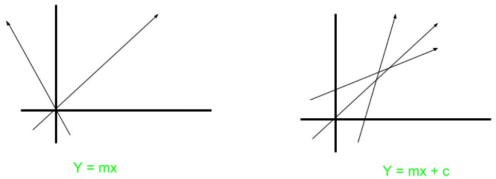
Training – apply training data + *adjust the weights* until the output is correspond to the label
=> the training data is run through the network - multiple times

Epoch - Every time the *data set* - run through - *each pass of entire data set*
Carefully change weights for every epoch, *train slowly,* - adjust weights slowly

When to stop? Look at the total error – How far out am I?
The error should decrease to *the level that is acceptable*
The most common way of measure: *Mean Squared error*

3) Bias

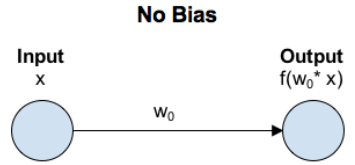




Bias make adjustments *within* neurons.

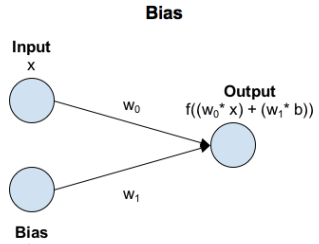
Bias is a constant which helps the model in a way that it can fit best for the given data.

Change bias - increasing the value of triggering activation function.



Bias is learnable – has own weights

b - Bias almost always use weight value 1 (absolutely not 0)



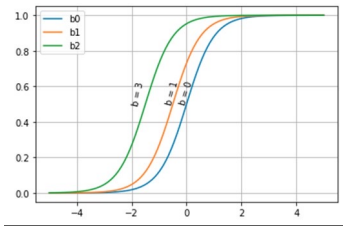
Bias "before perceptron " » supports fire - mx + b

Y = mx + b (b = y intercept)

Bias " in perceptron " » sets the Threshold $^{-(x+b)}$

= > 1 / 1 + $e^{-(x+b)}$

- $(x^n * w^n + b * w)$



Purpose of the bias term is to shift the position of the curve left or right to delay or accelerate the activation of a node.

Activation level - uses Linear polynomial

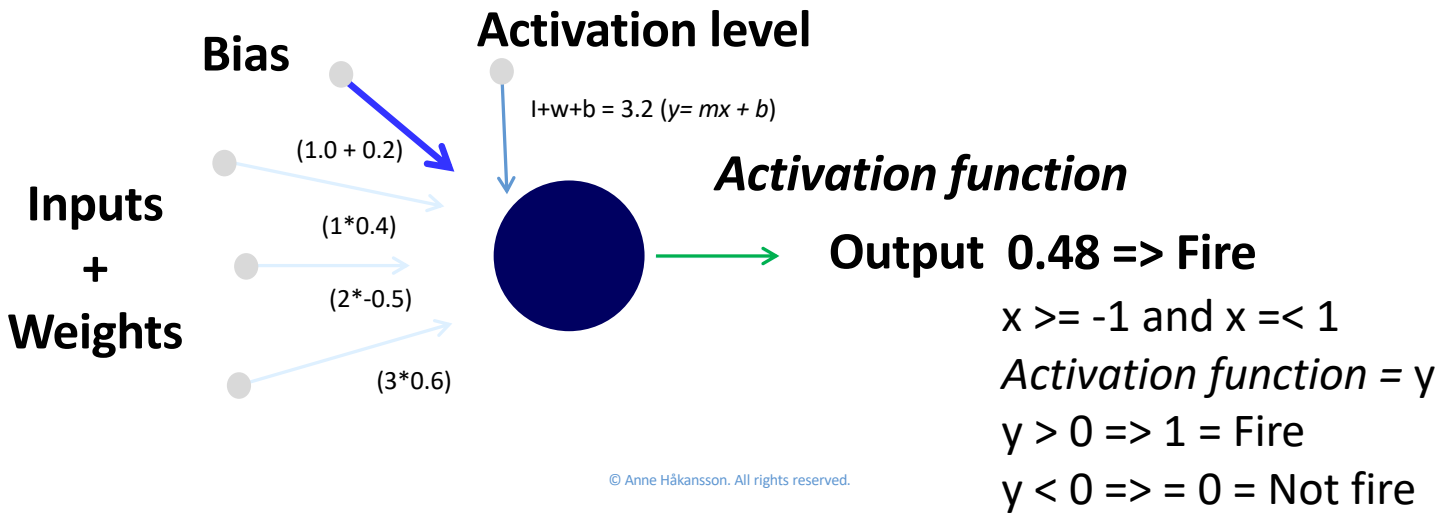
A neuron receives a number of *inputs + weights*

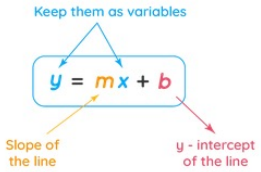
BIAS supports setting “the breakpoint” - threshold

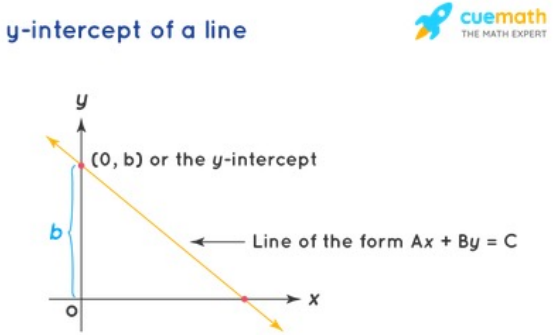
Calculate value=> **activation level** of the neuron (Linear polynomial)

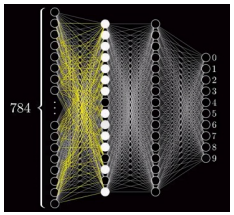
Activation function is applied to the inputs (e.g., Sigmoid, ReLU)

active= true/ not active false









b = 1.0

$w_0 = 0.02$

0

$w_1 = 1.0$

0.2

$w_2 = 0.65$

0.2

$w_3 = -0.4$

F = mx + b

$$sigmoid\ function = \frac{1}{1 + e^{-(w*x+b)}}$$

4) Activation functions

Activation function – calculation (inside each neuron) using a function

in example Activation is 3.2 = 0.73 (see Sigmoid)

Most common Activation function (for threshold):

Logistic Sigmoid

– give output between [0,1] = (y - always positive, (x,y) = (0, 0.5))

Sigmoid $y = 1 / 1 + e^{-x}$

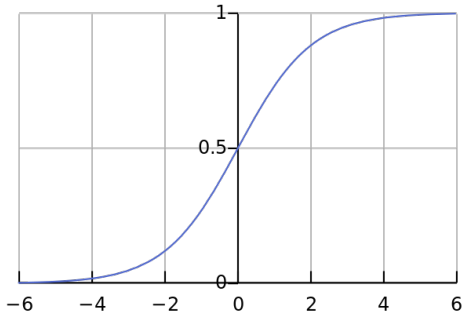
$e \approx 2{,}718281828$

Easy to analyze

Easy to calculate

“Soft” transition between 1 and 0

Intercept at 0,5



Other Activation functions

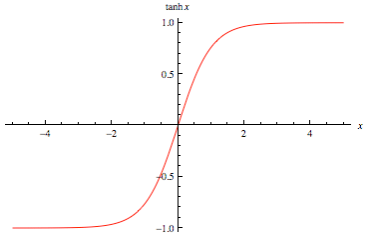
Hyperbolic Tangent

Output *between* [-1, 1]

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

Intercept at 0,0 (Origin)

(-x, -y), (x, y)

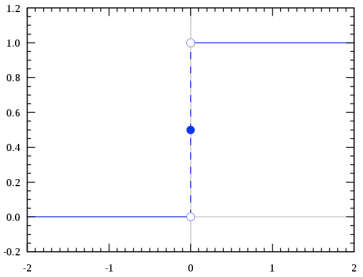


Heaviside (Unit) Step (original function)

Output *either* 0 or 1

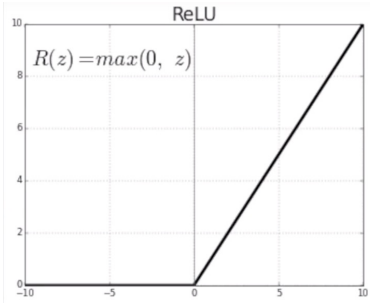
If (x <0) then y = 0

else if (x >= 0) then y = 1 (Boolean)



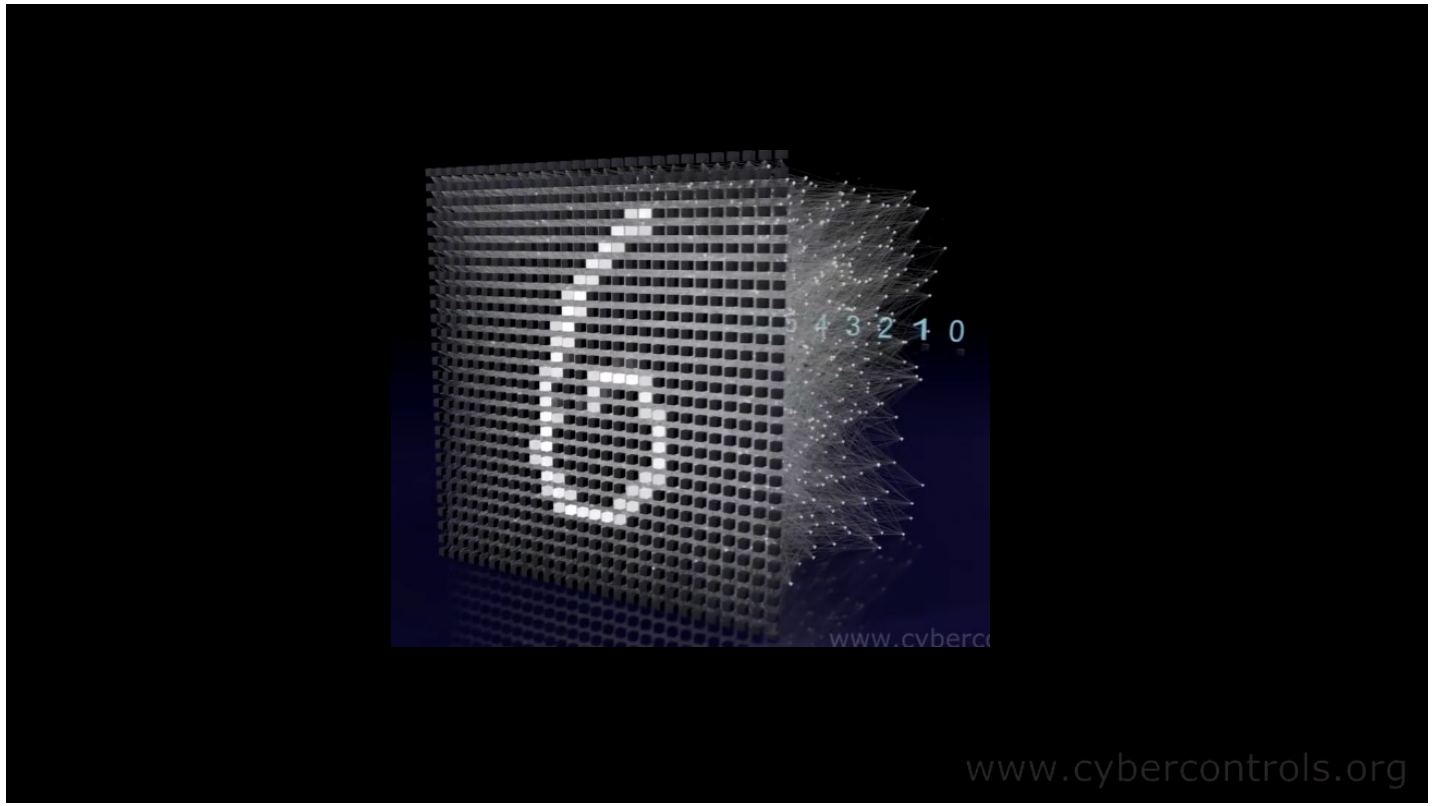
ReLU - Rectified Linear Unit

$f(x) = x^{+} = \max(0, x)$



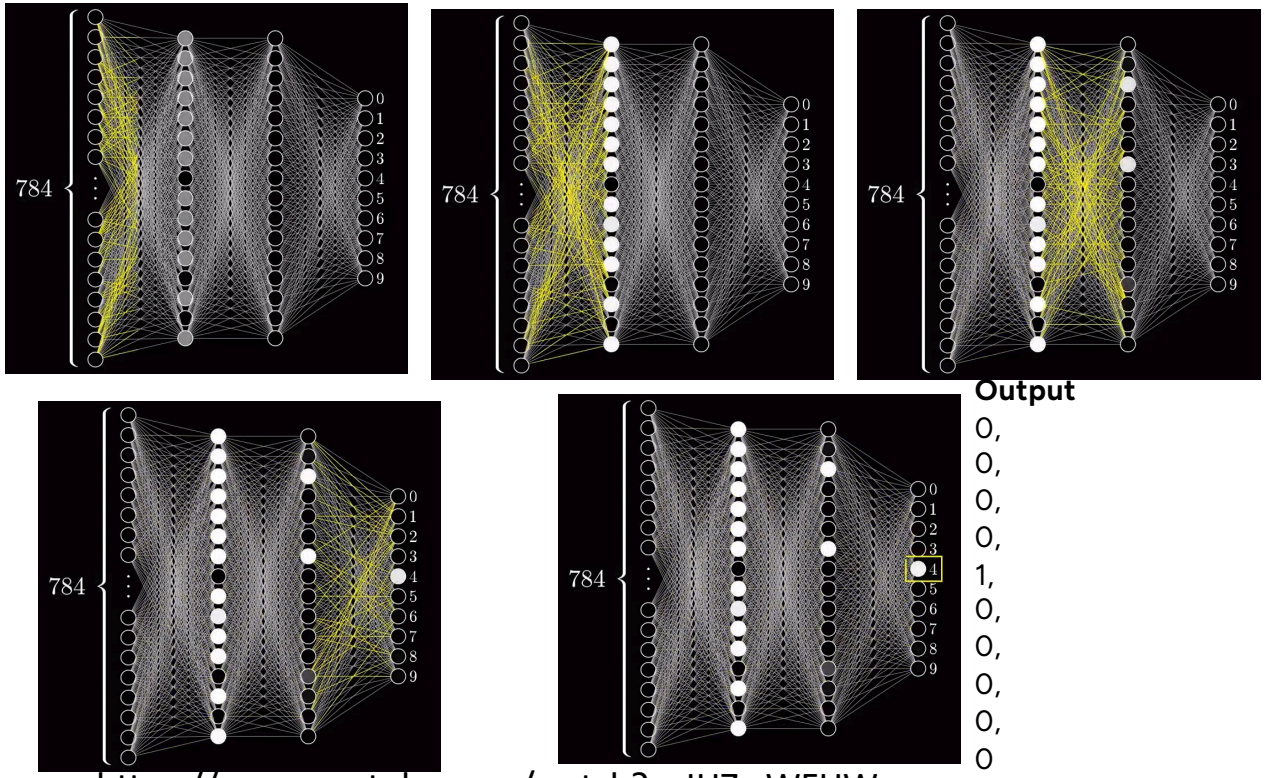
ANN and DL

https://www.youtube.com/watch?v=3JQ3hYko51Y



Training process showing fired / non-fired – input: example for digits (in this case: number 4).
Pixel format - byte image, number is stored as integer - range of values 0 to 255. ... (zero = black, 255 = white)

Input
0,
0,
0,
0,
0,
0,
.
.
198,
156,
0,
0,
.
.
.



Output
0,
0,
0,
0,
1,
0,
0,
0,
0,
0

https://www.youtube.com/watch?v=IHZwWFHWa-w

How many layers / neurons

Generally - 1–3 (up to 5) hidden layers will serve well for most problems.

Starting with 1–3 layers and 1–100 neurons and slowly adding more layers and neurons until start overfitting.

In general, using the same number of neurons for all hidden layers will suffice.

-> MNIST See Perceptron movie – 10 000 neurons – 3 hidden layers (about 4000 + 3000 + 3000 – good?)

How many layers / neurons – Improve performance

Improve performance :

(some datasets)

>> having a large first layer and following it up with “smaller layers”- fewer neurons.

The first layer - learn a lot of lower-level features -> feed into a few higher order features in the subsequent layers.

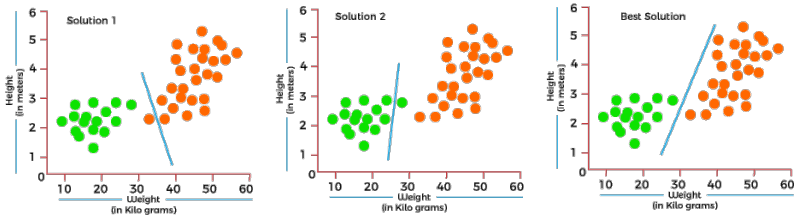
Commonly:

- get more of a “performance boost” from adding more layers than adding more neurons in each layer.

- *track loss and accuracy* within weights and biases dashboard to *see* which hidden layers + hidden neurons combo leads to the best loss.
- with choosing a smaller number of layers/neurons – consequence: if this number is too small, your network will not be able to learn the underlying patterns in your data and thus be useless.

An approach to counteract this is to start with a huge number of hidden layers + hidden neurons and then use dropout and early stopping to let the neural network size itself down for you

Cost functions



- ***Measure model performances for a given dataset.***
- Calculates the difference between the expected value and predicted value and represents it as a single real number. - use for training

1. Regression Cost Function –
   - Error = Actual Output-Predicted output
   - Mean of all errors during training (one of the simplest ways possible)

2. Binary Classification cost Functions - predictions for 0 (cat) or 1 (dog)

- **y = 1** → y^→1\hat{y} \to 1y^→1
- **y = 0** → y^→0\hat{y} \to 0y^→0

3. Multi-class Classification Cost Function - multi-class classification with the target values ranging from 0 to 1, 3, ….,n classes.

label y represented as one-hot vector, $y = [0, 0, 1, 0]$

Mean Squared error for training
– *Can* be used to stop the Epoch (the training) by comparing epoch results)

Handle the errors between the computed and desired values

- *Mean Squared error* - Adjust the weights of the neural network to *minimize the mean squared error* on training set

In the *simplest* terms, mean squared error is defined as:

*-> Simply calculate the **mean** of the **squares** of the **errors** = = difference between the actual value and expected value. Because of the square, it avoids any possibility of negative error.*

1) *Calculate the error value*
   – make everything to positive values using X^2 (e.g., -0,5 -> 2,50)
2) *Sum all values*: Sum (actual value – prediction value)^2
3) *Compare* this value with the output = error distance

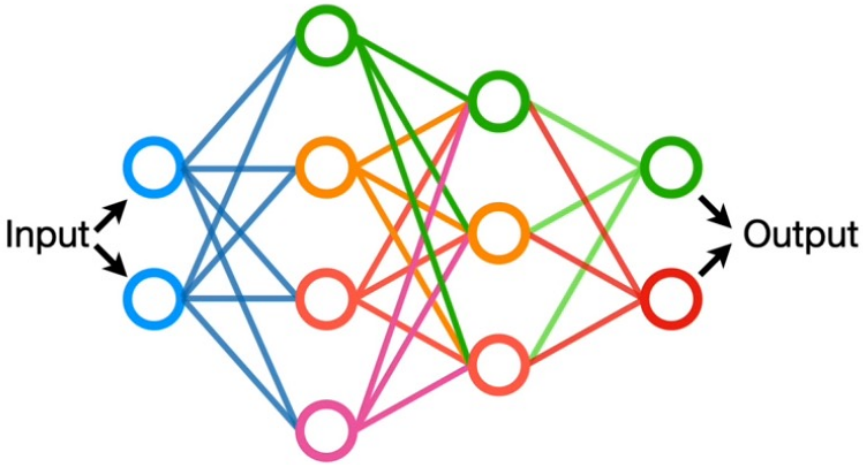
Actual value	Expected value	Error	Squared Error
0.1	0	0.1	0.01
0.9	1	-0.1	0.01
0.0	0	0	0
0.3	0	0.3	0.09
0.4	0	0,4	0.16
0.3	1	-0.7	0.49
SUM:			*0,76*

$(0.1-0)^2+(0.9-1)^2+(0.0-0)^2+$
$(0.3-0)^2+(0.4-0)^2+(0.3-1)^2$= SUM (Squared error)

MSE= SUM Squared error (0,76)/observations (6) = 0,126667

Neural Networks Pt. 1: Inside the Black Box

However, people call them a **black box** because it can be hard to understand what they are doing.



- **https://www.youtube.com/watch?v=CqOfi41LfDw**

Questions

What are the definitions of artificial neural networks?

What can ANN be used for?

When is ANN good to use?

What is a single-layer perceptron?

What is a multiple-layer perceptron?

What is an ANN? What does the ANN consist of?

How does the neural network training process work?

What are weights?

What is bias?

What is an activation level and what is activation function?

Which are the four most common activation functions? How do these work?

What is cost function, mean squared error, and when is it used?