SWINBURNE UNIVERSITY OF TECHNOLOGY

COS30018 – Week 6: Artificial Neural Networks & Deep Learning

Nature-inspired computing

- Nature has always served as a source of inspiration for engineers and scientists
- The best problem solver known in nature is:
  - the (human) brain that created “the wheel, New York, wars and so on” (after Douglas Adams’ Hitch-Hikers Guide)
  - the evolution mechanism that created the human brain (after Darwin’s Origin of Species)
- Answer 1 → neurocomputing
  - Today
- Answer 2 → evolutionary computing
  - Weeks 7 + 8

Presentation Outline

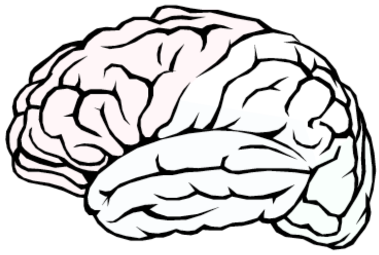
- Artificial Neural Networks
    - Perceptron
    - Perceptron Training Algorithm
    - Back Propagation
- Deep Learning

Supervised Learning using Artificial Neural Networks

Neural Networks

- Human brain consists of around 86 billion interconnected "neurons"
- Each neuron is connected to about 1000+ neighbouring neurons
- Each neuron responds to stimuli and passes output to other neurons

- In ML, artificial neural networks are loosely modelled on the human brain



SWINBURNE UNIVERSITY OF TECHNOLOGY

Biological Networks of Neurons

- A neuron receives inputs (electrochemical signals) from its neighbours
- If enough inputs are received at the same time, it gets *activated*
- Once activated, a neuron fires, giving an output that may (in turn) activate connected neurons
- The behaviour of a biological network of neurons is dependent on connection types and activation strength (*synaptic function*)

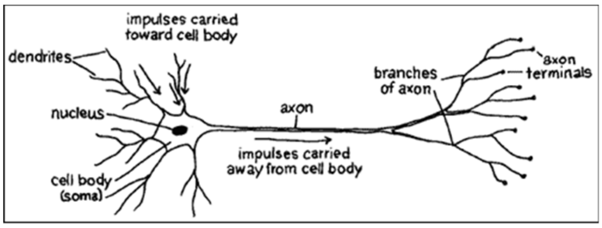


Image Source: https://science.education.nih.gov/supplements/webversions/BrainAddiction/guide/lesson2-1.html

SWINBURNE UNIVERSITY OF TECHNOLOGY

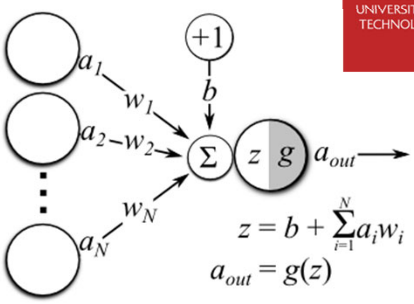
Artificial Neural Networks

- Loosely modelled on the brain
- Consist of many processing units (neurons) connected together.
- As a collection, they have sophisticated behaviour
  - "Connectionist model" – emergent process of interconnected networks of simple units,
  - "Parallel distributed processing"
- Many different kinds of neural networks exist
- Mainly used for learning & reasoning
- Can be treated as a *black-box* for inputs and outputs

Artificial Neuron

- A neuron has a bunch of weighted inputs (take inputs a_1, a_2... a_n and multiply by the weight w_1, w_2,... w_n)
- A bias is then added to the total (as a control parameter)
- If final result is greater than a threshold, then the **activation function** is used to compute the output
  - Strength of output depends upon the activation function chosen
- The output is then fed to other neurons

- The weights and bias are determined through **training** (i.e., the ANN **learns** a **model**)

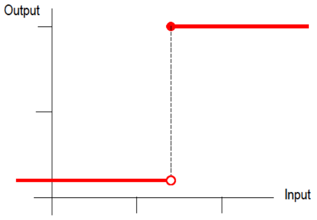


https://theclevermachine.wordpress.com/2014/09/11/a-gentle-introduction-to-artificial-neural-networks/

Activation Functions

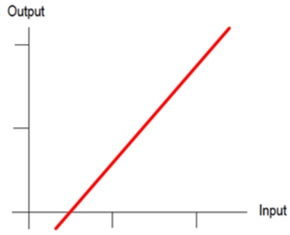
- step, linear, sigmoid, tanH, Gaussian, ReLU

$$y = \begin{cases} A_1, if\ w.x + b > 0 \\ A_0, \quad otherwise \end{cases}$$



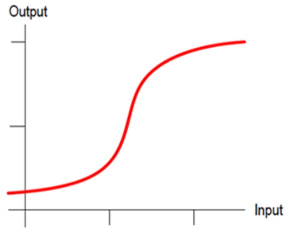
Step Threshold

$$y = \sum_{i=1}^{N} x_i w_i$$



Linear

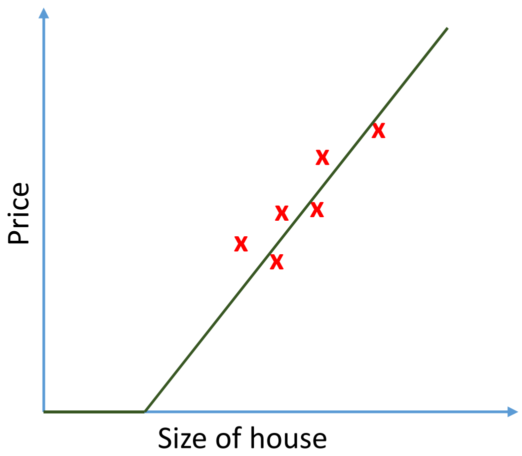
$$y = \frac{1}{1 + e^{-x}}$$

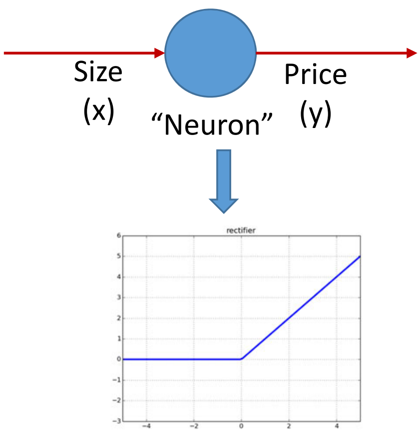


Sigmoid

Learn more about activation functions - https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

Price Prediction –Single Neuron





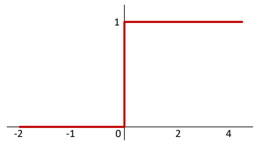
Rectified **L**inear **U**nit

Perceptron

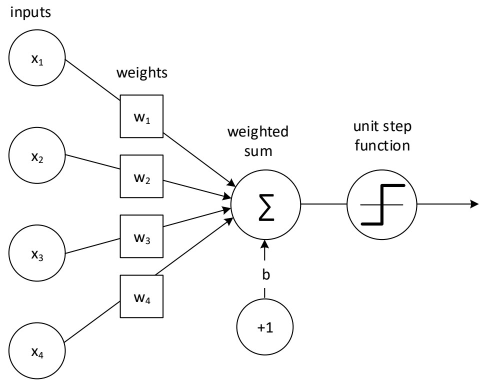
- Simplest neural network possible
- Single neuron that performs binary classification
- Uses a *step activation function*
  - Also referred to as *Heaviside Step Function*

$$f(x) = \begin{cases} 1, & \text{if } w.x + b > t \\ 0, & \text{otherwise} \end{cases}$$

where $w.x$ is the dot product $\sum_{i=1}^{N} x_i w_i$ and t is the threshold



Step Activation Function

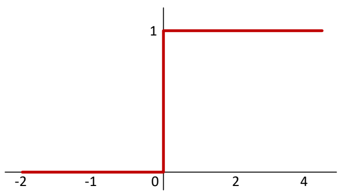


Training a Perceptron

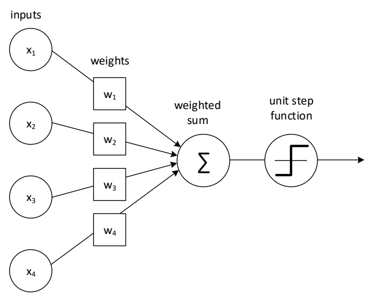
- Initialize model with random weights (usually between -0.5 and 0.5)
- Iteratively adjust weights on the basis of training examples that pair inputs with targets
- Modify weights at each step according to the perceptron training rule

  $w_i \leftarrow w_i + \alpha(t\text{-}o) * x_i$

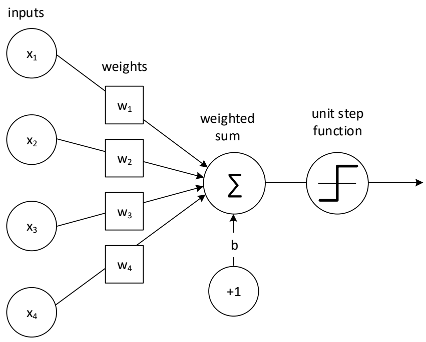
  where t = *target output*, o = *observed output*, α = *learning rate, and* x_i = *input*



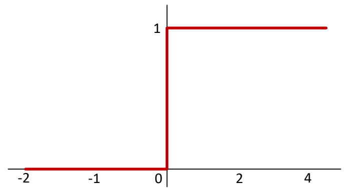
Step Activation Function



Training a Perceptron



- Randomly initialise all weights
- **Loop** through training set
  - If *actual output* is 1 and *target output* is 0 (false positive), decrement ***active weights***
  - If *actual output* is 0 and *target output* is 1 (false negative), increment ***active weights***
- **Until** Network gives the correct outputs (or some time/step limit is exceeded)
- The perceptron training rule will converge if the training set is linearly separable



Step Activation Function

Prediction using Perceptron

Old Training Data

#	Student	HD (2010)	Male	Works Hard	Drinks	HD (2011)
1	Bob	yes	yes	no	yes	No
2	Howard	yes	yes	yes	no	Yes
3	Natasha	yes	no	yes	yes	Yes
4	Bush	no	yes	no	yes	No
5	Blair	no	yes	yes	yes	No
6	Mary	no	no	yes	no	Yes

New Test Data

1	Tony	yes	yes	yes	no	?

- Can we predict the grade of a student based on historical data?
  - Learn using the example data where input and expected output is known
  - Adjust weights of active connections if the expected output does not match input

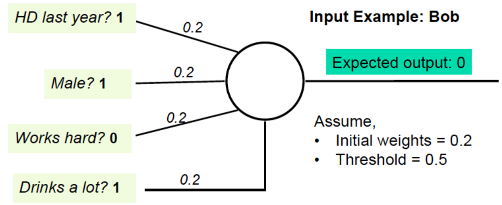
Training the Perceptron (1) - Bob

Old Training Data

#	Student	HD (2010)	Male	Works Hard	Drinks	HD (2011)
1	Bob	yes	yes	no	yes	No
2	Howard	yes	yes	yes	no	Yes
3	Natasha	yes	no	yes	yes	Yes
4	Bush	no	yes	no	yes	No
5	Blair	no	yes	yes	yes	No
6	Mary	no	no	yes	no	Yes

New Test Data

#	Student	HD (2010)	Male	Works Hard	Drinks	HD (2011)
1	Tony	yes	yes	yes	no	?



- Initial weights = 0.2, Threshold = 0.5
- Start with the first record in the training set
- Weighted sum = (0.2 * 1 + 0.2 * 1 + 0.2 * 0 + 0.2 * 1) = 0.6
- Check weighted sum against threshold
- Since 0.6 > threshold (0.5), Output = 1 (Incorrect)
- Hence, weights need to be adjusted

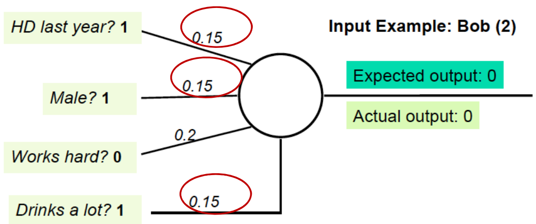
Training the Perceptron (2) - Bob

Old Training Data

#	Student	HD (2010)	Male	Works Hard	Drinks	HD (2011)
1	Bob	yes	yes	no	yes	No
2	Howard	yes	yes	yes	no	Yes
3	Natasha	yes	no	yes	yes	Yes
4	Bush	no	yes	no	yes	No
5	Blair	no	yes	yes	yes	No
6	Mary	no	no	yes	no	Yes

New Test Data

#	Student	HD (2010)	Male	Works Hard	Drinks	HD (2011)
1	Tony	yes	yes	yes	no	?



- Update the active connection weights
  - **Decrement weight by 0.05** (domain dependent)
- Weighted sum = 0.15 + 0.15 + 0 + 0.15 = 0.45
- Since 0.45 < threshold (0.5), Output = 0 (Correct)
- Hence, weights do not need any further adjustments
- Perceptron is able to correctly predict the output for the first record

Training the Perceptron (3) - Howard

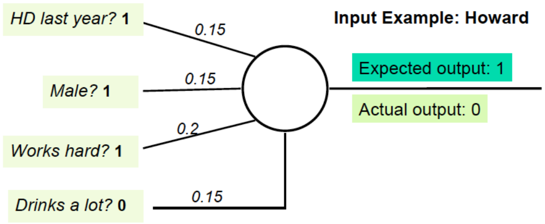
- Train the perceptron using new training data
- Weighted sum (0.15 + 0.15 + 0.2) = 0.5
- 0.5 < threshold (0.5), Output = 0 (Incorrect)
- **Active weights need adjustment**

Old Training Data

#	Student	HD (2010)	Male	Works Hard	Drinks	HD (2011)
1	Bob	yes	yes	no	yes	No
2	Howard	yes	yes	yes	no	Yes
3	Natasha	yes	no	yes	yes	Yes
4	Bush	no	yes	no	yes	No
5	Blair	no	yes	yes	yes	No
6	Mary	no	no	yes	no	Yes

New Test Data

1	Tony	yes	yes	yes	no	?



Training the Perceptron (4) – Howard

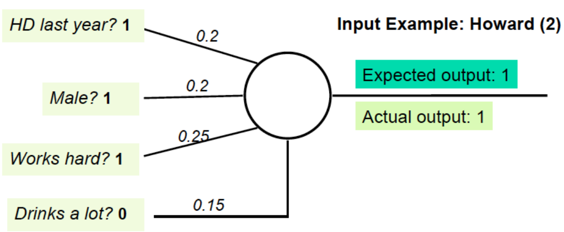
- **Increasing weights of active connections by 0.05**
- Weighted sum (0.2 + 0.2 + 0.25) = 0.65
- 0.65 > threshold (0.5), Output = 1 (Correct)
- Weights do not need any further adjustments

Old Training Data

#	Student	HD (2010)	Male	Works Hard	Drinks	HD (2011)
1	Bob	yes	yes	no	yes	No
2	Howard	yes	yes	yes	no	Yes
3	Natasha	yes	no	yes	yes	Yes
4	Bush	no	yes	no	yes	No
5	Blair	no	yes	yes	yes	No
6	Mary	no	no	yes	no	Yes

New Test Data

1	Tony	yes	yes	yes	no	?



Training the Perceptron (5) - Natasha

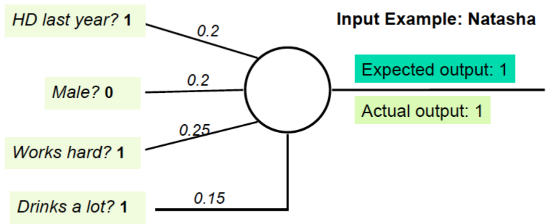
- Weighted sum (0.2 + 0 + 0.25 + 0.25) = 0.6
- 0.6 > threshold (0.5), Output = 1 (Correct)
- **Weights are left unchanged**

Old Training Data

#	Student	HD (2010)	Male	Works Hard	Drinks	HD (2011)
1	Bob	yes	yes	no	yes	No
2	Howard	yes	yes	yes	no	Yes
3	Natasha	yes	no	yes	yes	Yes
4	Bush	no	yes	no	yes	No
5	Blair	no	yes	yes	yes	No
6	Mary	no	no	yes	no	Yes

New Test Data

1	Tony	yes	yes	yes	no	?



Training the Perceptron (6) - Bush

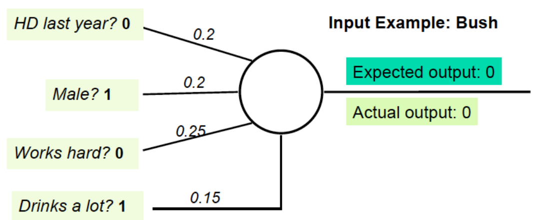
- Weighted sum (0 + 0.2+ 0 + 0.15) = 0.35
- 0.35 < threshold (0.5), Output = 0 (Correct)
- **Weights are left unchanged**

Old Training Data

#	Student	HD (2010)	Male	Works Hard	Drinks	HD (2011)
1	Bob	yes	yes	no	yes	No
2	Howard	yes	yes	yes	no	Yes
3	Natasha	yes	no	yes	yes	Yes
4	Bush	no	yes	no	yes	No
5	Blair	no	yes	yes	yes	No
6	Mary	no	no	yes	no	Yes

New Test Data

1	Tony	yes	yes	yes	no	?



Training the Perceptron (7) - Blair

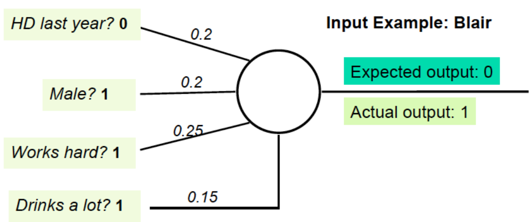
- Weighted sum (0 + 0.2+ 0.25 + 0.15) = 0.6
- 0.6 > threshold (0.5), Output = 1 (Incorrect)
- **Weights need to be changed**

Old Training Data

#	Student	HD (2010)	Male	Works Hard	Drinks	HD (2011)
1	Bob	yes	yes	no	yes	No
2	Howard	yes	yes	yes	no	Yes
3	Natasha	yes	no	yes	yes	Yes
4	Bush	no	yes	no	yes	No
5	Blair	no	yes	yes	yes	No
6	Mary	no	no	yes	no	Yes

New Test Data

1	Tony	yes	yes	yes	no	?



Training the Perceptron (8) - Blair

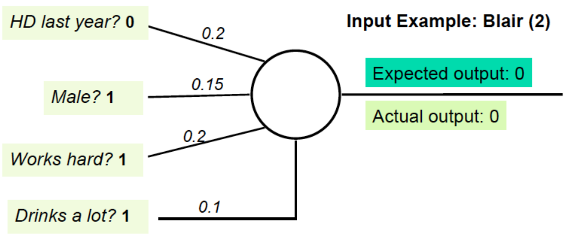
- New weights – 0.2, 0.15, 0.2, 0.1
- Weighted sum (0 + 0.15+ 0.2 + 0.1) = 0.45
- 0.45 < threshold (0.5), Output = 0 (Correct)
- **Weights need not be changed**

Old Training Data

#	Student	HD (2010)	Male	Works Hard	Drinks	HD (2011)
1	Bob	yes	yes	no	yes	No
2	Howard	yes	yes	yes	no	Yes
3	Natasha	yes	no	yes	yes	Yes
4	Bush	no	yes	no	yes	No
5	Blair	no	yes	yes	yes	No
6	Mary	no	no	yes	no	Yes

New Test Data

1	Tony	yes	yes	yes	no	?



Training the Perceptron (9) - Mary

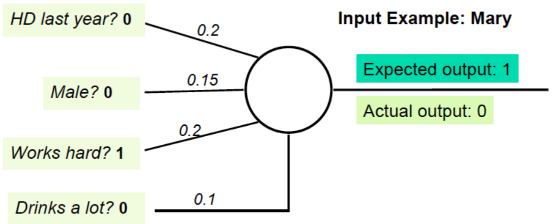
- Weighted sum (0 + 0 + 0.2 + 0) = 0.2
- 0.2 < threshold (0.5), Output = 0 (Incorrect)
- **Weights need to be changed (for '*works hard*')**

Old Training Data

#	Student	HD (2010)	Male	Works Hard	Drinks	HD (2011)
1	Bob	yes	yes	no	yes	No
2	Howard	yes	yes	yes	no	Yes
3	Natasha	yes	no	yes	yes	Yes
4	Bush	no	yes	no	yes	No
5	Blair	no	yes	yes	yes	No
6	Mary	no	no	yes	no	Yes

New Test Data

#	Student	HD (2010)	Male	Works Hard	Drinks	HD (2011)
1	Tony	yes	yes	yes	no	?



Training the Perceptron (10) - Mary

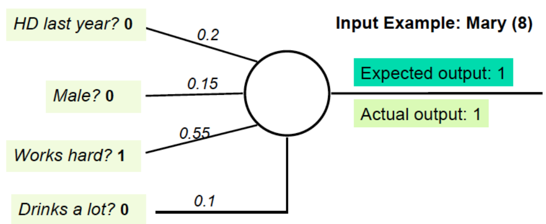
- After multiple adjustments, the weight for "Works hard" is set to 0.55
- 0.55 < threshold (0.5), Output = 1 (Correct)
- **Weights need not be changed any more**

Old Training Data

#	Student	HD (2010)	Male	Works Hard	Drinks	HD (2011)
1	Bob	yes	yes	no	yes	No
2	Howard	yes	yes	yes	no	Yes
3	Natasha	yes	no	yes	yes	Yes
4	Bush	no	yes	no	yes	No
5	Blair	no	yes	yes	yes	No
6	Mary	no	no	yes	no	Yes

New Test Data

#	Student	HD (2010)	Male	Works Hard	Drinks	HD (2011)
1	Tony	yes	yes	yes	no	?



Training the Perceptron (11) - Bob

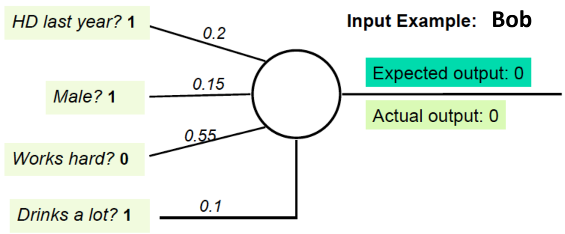
- The process is repeated starting with Bob again
  Weighted sum (0.2 + 0.15 + 0.1) = 0.45
- 0.45 < threshold (0.5), Output = 0 (Correct)
- **Process is repeated until sufficient accuracy has been achieved**

Old Training Data

#	Student	HD (2010)	Male	Works Hard	Drinks	HD (2011)
1	Bob	yes	yes	no	yes	No
2	Howard	yes	yes	yes	no	Yes
3	Natasha	yes	no	yes	yes	Yes
4	Bush	no	yes	no	yes	No
5	Blair	no	yes	yes	yes	No
6	Mary	no	no	yes	no	Yes

New Test Data

#	Student	HD (2010)	Male	Works Hard	Drinks	HD (2011)
1	Tony	yes	yes	yes	no	?



Training the Perceptron (12) - Howard

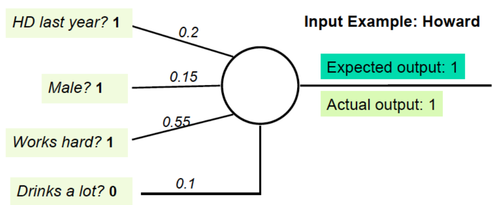
- We test the perceptron with Howard
  Weighted sum (0.2 + 0.15 + 0.55) = 0.9
- 0.9 < threshold (0.5), Output = 1 (Correct)

Old Training Data

#	Student	HD (2010)	Male	Works Hard	Drinks	HD (2011)
1	Bob	yes	yes	no	yes	No
2	Howard	yes	yes	yes	no	Yes
3	Natasha	yes	no	yes	yes	Yes
4	Bush	no	yes	no	yes	No
5	Blair	no	yes	yes	yes	No
6	Mary	no	no	yes	no	Yes

New Test Data

#	Student	HD (2010)	Male	Works Hard	Drinks	HD (2011)
1	Tony	yes	yes	yes	no	?



- This setting works fine for Natasha, Bush and Mary as well

Training the Perceptron (13) - Blair

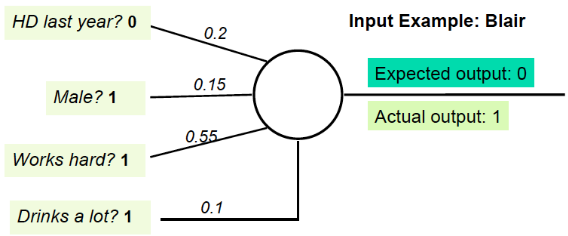
- We test the perceptron with Blair
- Weighted sum (0 + 0.15 + 0.55 + 0.1) = 0.8
- 0.9 < threshold (0.5), Output = 1 (Incorrect)

Old Training Data

#	Student	HD (2010)	Male	Works Hard	Drinks	HD (2011)
1	Bob	yes	yes	no	yes	No
2	Howard	yes	yes	yes	no	Yes
3	Natasha	yes	no	yes	yes	Yes
4	Bush	no	yes	no	yes	No
5	Blair	no	yes	yes	yes	No
6	Mary	no	no	yes	no	Yes

New Test Data

#	Student	HD (2010)	Male	Works Hard	Drinks	HD (2011)
1	Tony	yes	yes	yes	no	?



- Weights will be adjusted a lot to bring down the factor on 'Works hard'
- Eventually, the perceptron will settle down (Do we get 100% accuracy?)

Trained Perceptron - Prediction

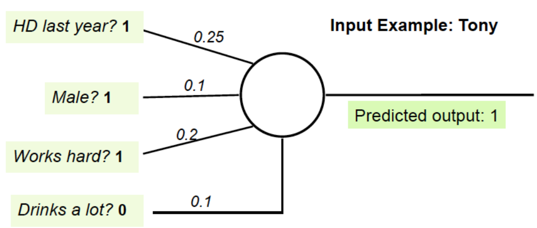
- The trained perceptron can now predict if Tony will do well

Old Training Data

#	Student	HD (2010)	Male	Works Hard	Drinks	HD (2011)
1	Bob	yes	yes	no	yes	No
2	Howard	yes	yes	yes	no	Yes
3	Natasha	yes	no	yes	yes	Yes
4	Bush	no	yes	no	yes	No
5	Blair	no	yes	yes	yes	No
6	Mary	no	no	yes	no	Yes

New Test Data

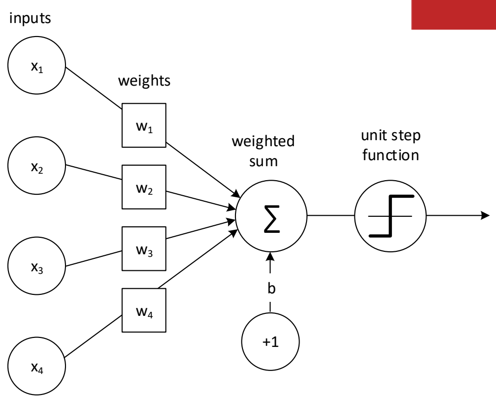
#	Student	HD (2010)	Male	Works Hard	Drinks	HD (2011)
1	Tony	yes	yes	yes	no	?



- Weighted sum (0.25 + 0.1 + 0.2 + 0.1) = 0.65
- 0.65 > threshold (0.5), Output = 1
- The perceptron predicts that Tony will get a HD

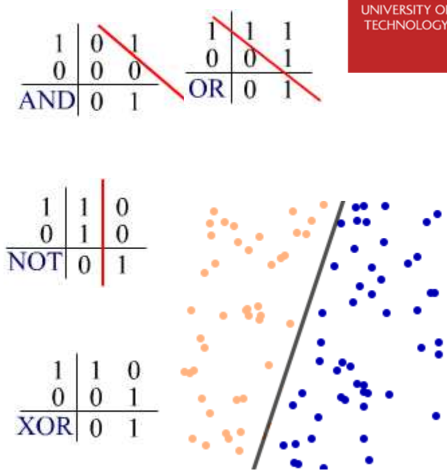
Perceptron - Summary

- Perceptron → single neuron that classifies a set of inputs into one of two categories
- Uses a step activation function
- Learning involves choosing values for the weights $w_1, \ldots, w_n$ so that the actual output *o* matches the target output *t*
- The learning procedure will converge if the data is linearly separable and a sufficiently small learning rate is used

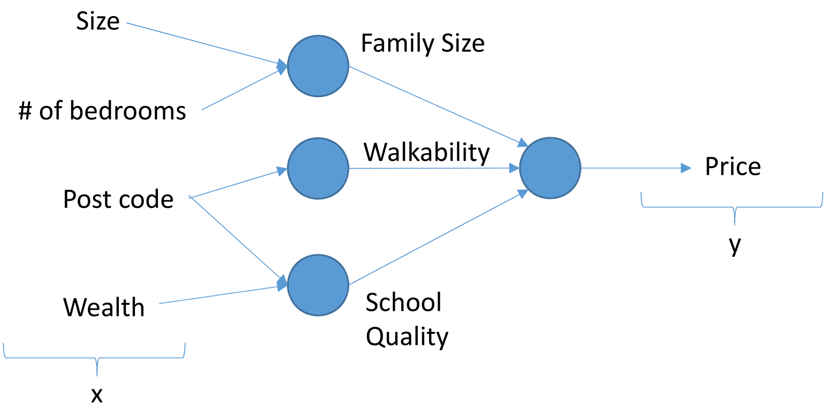


Perceptrons & Linearly Separable Regions

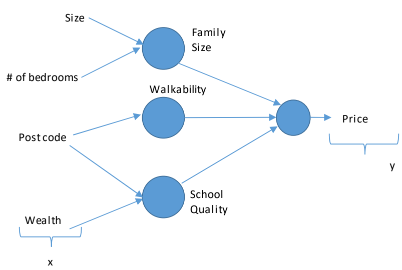
- Perceptrons can only classify linearly separable functions
  - The two axes are the inputs which can take values of 0 and 1
  - Numbers on the graph are the expected output for a particular input
  - AND, OR and NOT are linearly separable
- XOR is not linearly separable
- However, XOR function can be represented using a multilayer network

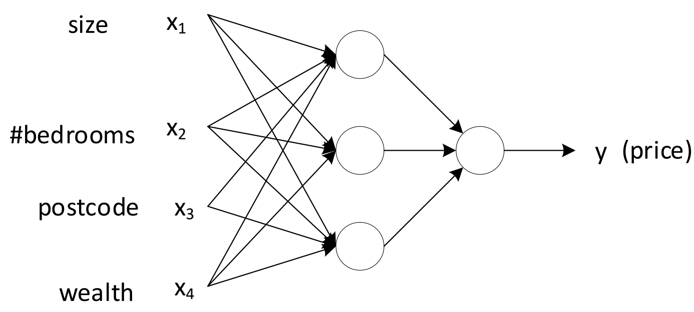


Price Prediction – Bigger Neural Network



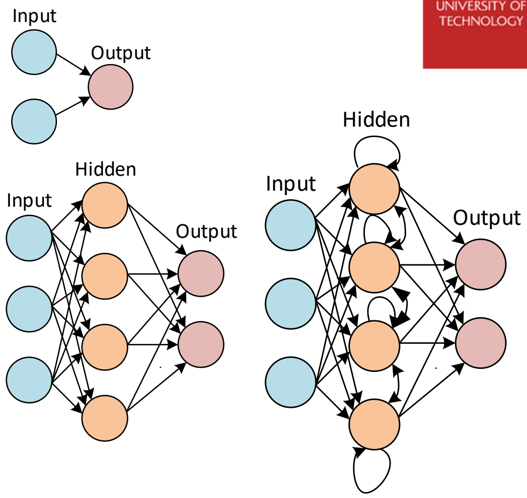
Price Prediction – Bigger Neural Network





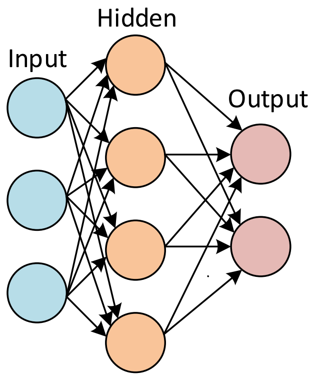
Complex Neural Network - Structures

- There are 2 main categories of neural network structures
- Acyclic/Feed-Forward Networks – no loops, no internal state
  - Single-layer perceptron
  - Multi-layer perceptron
- Cyclic/Recurrent Networks – feeds its outputs back into its own inputs, directed cycles, supports short-term memory



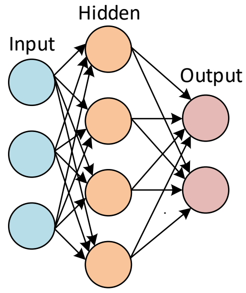
Multiple-Layer Neural Networks

- A complex neural network involves a set of neurons (e.g. perceptrons) interconnected in layers to solve more complex problems
  - Input layer neurons connect to hidden layer neurons
  - Hidden layer neurons connect to output layer neurons
  - Output layer can have multiple outputs
- Multilayer neural networks can classify a range of functions including non-linearly separable ones

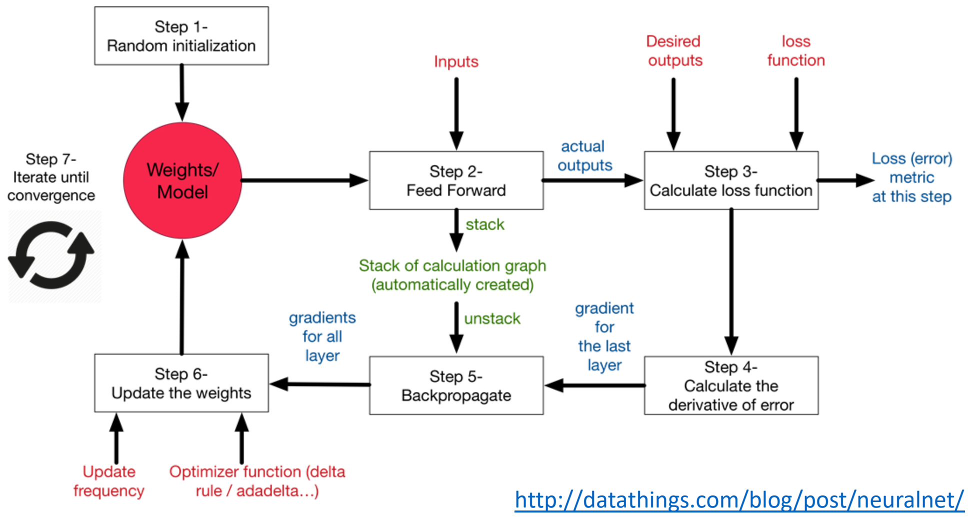


Learning in Multiple-Layer Neural Networks

- Multilayer NN learn the same way as perceptrons
- **Forward pass** – compute the values from input to output
- **Backward pass** – propagate errors backwards to adjust the connection weights
- Error occurs when <u>expected output != actual output</u>
- **Cost/Loss function** – a measure of “how good or bad” a NN did w.r.t its given training sample and the expected output



Learning in Multiple-Layer Neural Networks



http://datathings.com/blog/post/neuralnet/

Backpropagation

- An approach for training a ANN
  - used to optimize weights in a multi-layer NN so that it can learn to correctly map arbitrary inputs to outputs
  - Objective of training the network is to *minimize* value of the *cost function* by adjusting the weights

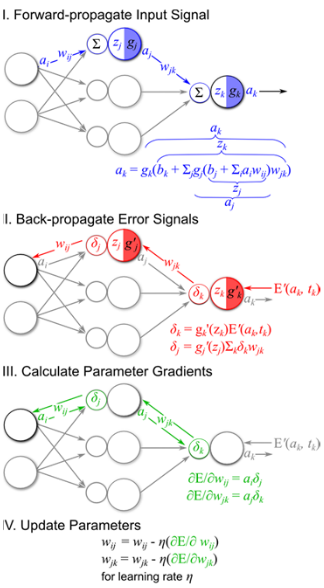
$$W := W - \alpha \frac{\partial J}{\partial W}$$

- α – learning rate, W - weight, J – cost function
- $\frac{\partial J}{\partial W}$ - Partial derivative of J w.r.t W

NOTE: partial derivative of a function of several variables is its derivate with respect to one variable keeping every other variable constant

Back Propagation Algorithm

- **Feedforward** – input signals are forward propagated though the network towards the outputs
  - The output is a composite function of the weights, inputs and activation function(s)
- **Verification** - Actual output value is compared with expected output.
  - ERROR = Desired output – Actual output
- **Backpropagation** – Network errors are back propagated backwards towards to the inputs

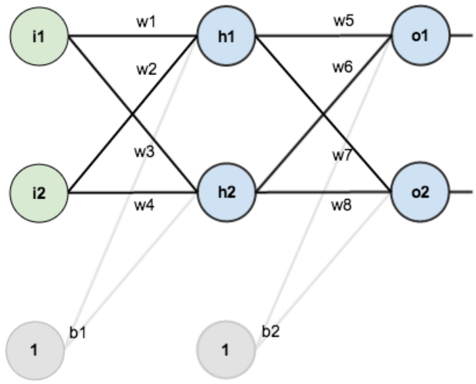


https://theclevermachine.wordpress.com/2014/09/11/a-gentle-introduction-to-artificial-neural-networks/

Back Propagation – Step by Step Example

Neural Network Structure

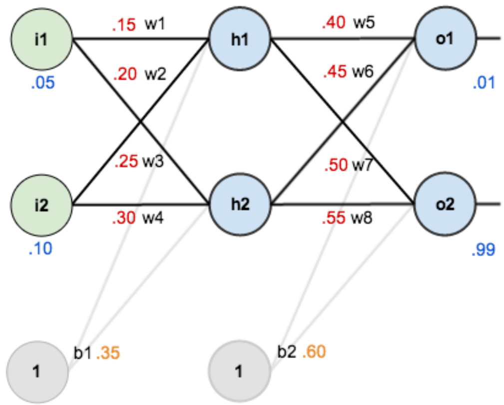
- Neural network with two inputs, two hidden neurons, and two output neurons
- Hidden and output neurons have a bias



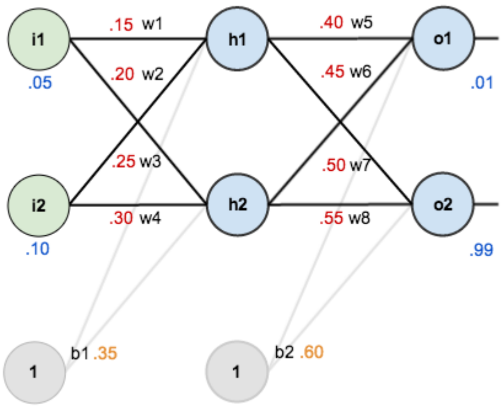
https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Back Propagation – Step by Step Example

Initial Weights, Biases and Training Inputs/Outputs



Back Propagation – Step by Step Example



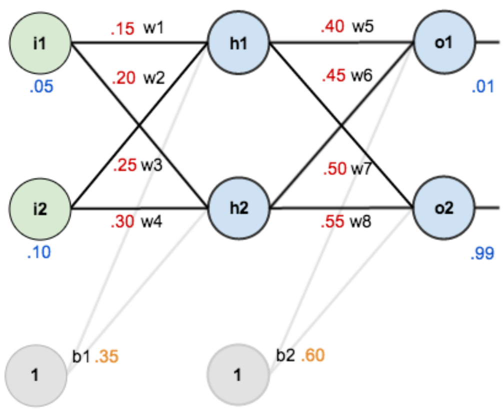
Goal of Back Propagation

- Optimize weights so that neural network can learn to correctly map arbitrary inputs to outputs
- Given the inputs 0.05 and 0.10, the neural network should output 0.01 and 0.99

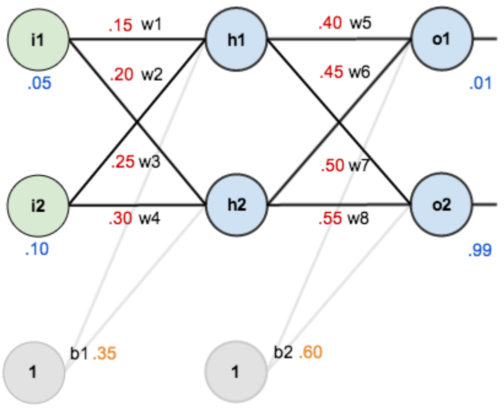
Back Propagation – Step by Step Example

Forward Pass

- Feed the inputs forward through the network
- Compute net input to each hidden neuron, apply activation function to it, repeat process with output layer neurons



Back Propagation – Step by Step Example



Forward Pass – 1:

Calculate total net input net_{h1} for $h1$

$net_{h1}=w_1*i_1 + w_2*i_2 + b_1$

$net_{h1}= 0.15*0.05+0.2*0.1+0.35$

$net_{h1}= 0.3775$

Apply activation function to calculate output out_{h1} of $h1$

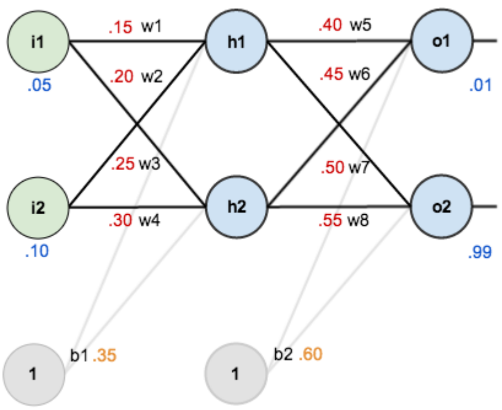
$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} \text{ (logistic function)}$$

$$= \frac{1}{1+e^{-0.3775}} = 0.59329992$$

Do the same for h_2

$out_{h2} = 0.596884378$

Back Propagation – Step by Step Example



Forward Pass - 2

Calculate net input net_{o1} for $o1$

$net_{o1}=w_5*out_{h1} + w_6*out_{h2} + b_2$

$= 0.40* 0.59329992 +0.45* 0.596884378 + 0.60$

$= 1.105905967$

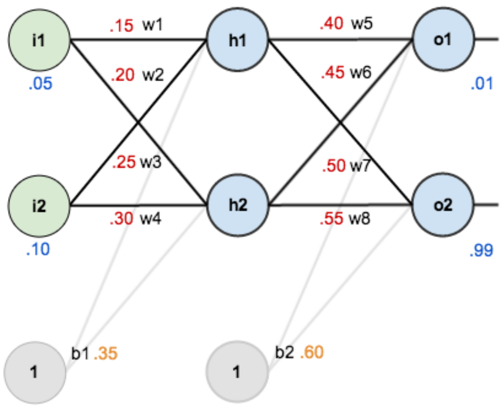
Apply activation function to calculate output out_{o1} of $o1$

$out_{o1} = 0.75136507$

Do the same for o_2

$out_{o2} = 0.772928465$

Back Propagation – Step by Step Example



Calculate Error

Calculate the total error of each output neuron using the *Squared Error Function*

$$E = \sum \tfrac{1}{2}(target - output)^2$$

Error E_{o1} for output neuron *o1*

$$E_{o1} = \tfrac{1}{2}(target_{o1} - out_{o1})^2$$

$$E_{o1} = \tfrac{1}{2}(0.01 - 0.7514)^2$$

$$E_{o1} = 0.274811083$$

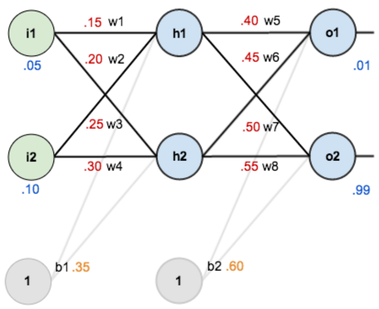
Error E_{o2} for output neuron *o2*

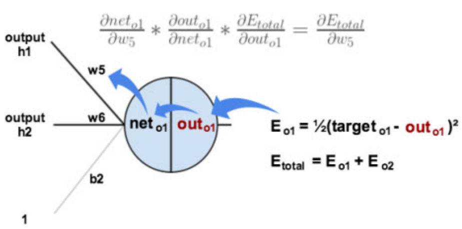
$$E_{o2} = 0.023560026$$

Total Error is the sum of the errors

$$E_{total} = 0.274811083 + 0.023560026$$
$$= 0.298371109$$

Back Propagation – Step by Step Example





Backward Pass

We want to determine how much a change in w_5 contributes to the total error E_{total}

$$\frac{\partial E_{total}}{\partial w_5} = \boxed{\frac{\partial E_{total}}{\partial out_{o1}}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

How much does the error change w.r.t out_{01}?

$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

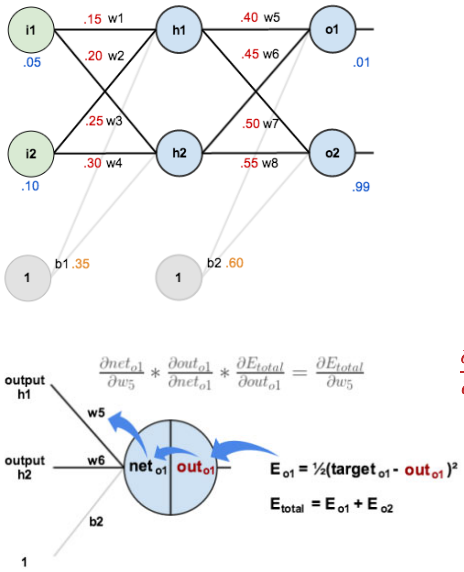
$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1})$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(0.01 - 0.75136507) = 0.74136507$$

Back Propagation – Step by Step Example

SWINBURNE UNIVERSITY OF TECHNOLOGY



Backward Pass

We want to determine how much a change in w_5 contributes to the total error E_{total}

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \boxed{\frac{\partial out_{o1}}{\partial net_{o1}}} * \frac{\partial net_{o1}}{\partial w_5}$$

How much does the output out_{o1} change w.r.t net_{o1}?

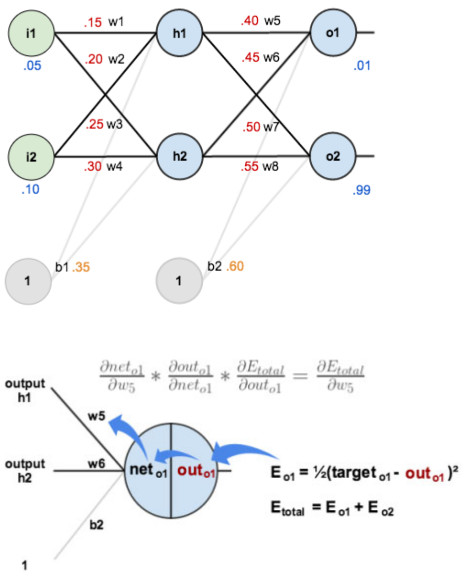
$$out_{o1} = \frac{1}{1 + e^{-net_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.775136507\ (1 - 0.75136507)$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = 0.186815602$$

Back Propagation – Step by Step Example

SWINBURNE UNIVERSITY OF TECHNOLOGY



Backward Pass

We want to determine how much a change in w_5 contributes to the total error E_{total}

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \boxed{\frac{\partial net_{o1}}{\partial w_5}}$$

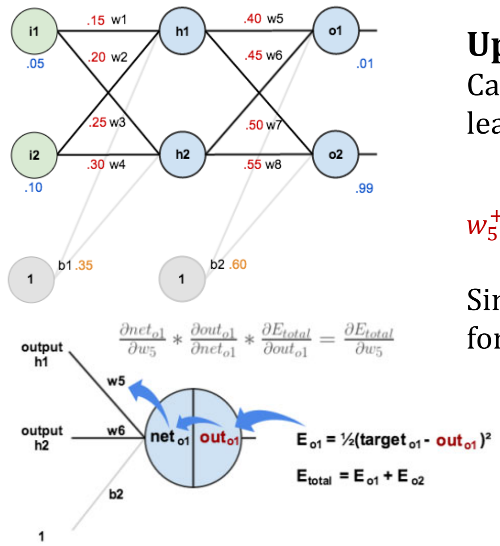
How much does the net input net_{o1} change w.r.t w_5?

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial w_5} = out_{h1} + 0 + 0 = 0.593269992$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.59327$$
$$= 0.082167041$$

Back Propagation – Step by Step Example



Update weight

Calculating new weight (using an optional learning rate α of 0.5)

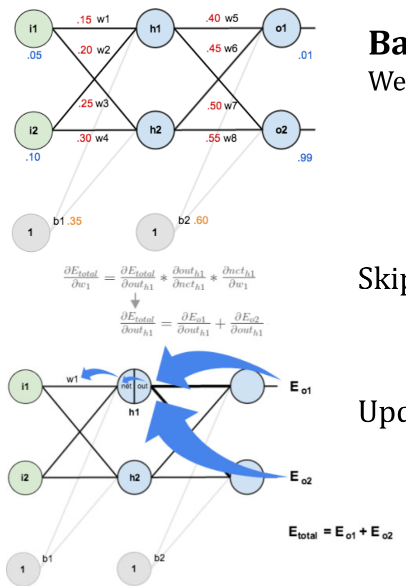
$$w_5^+ = w_5 - \alpha * \frac{\partial E_{total}}{\partial w_5}$$

$$w_5^+ = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

Similarly, we can calculate new weights for w_6, w_7 and w_8

$$w_6^+ = 0.408666186$$
$$w_7^+ = 0.511301270$$
$$w_8^+ = 0.561370121$$

Back Propagation – Step by Step Example



Backward Pass - 2

We calculate the new value for the weight w_1, w_2, w_3, w_4

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

Skipping a few steps....

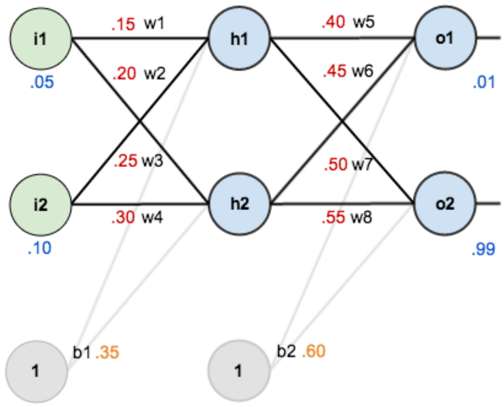
$$\frac{\partial E_{total}}{\partial w_1} = 0.036350306 * 0.241300709 * 0.05 = 0.000438568$$

Updating the weights.... $w_1^+ = w_1 - \alpha * \frac{\partial Etotal}{\partial w_1}$

$$w_1^+ = 0.15 - 0.5 * 0.000438568 = 0.149780716$$

$$w_2^+ = 0.19956143, \qquad w_3^+ = 0.24975114$$
$$w_4^+ = 0.29950229$$

Back Propagation – Step by Step Example



- After updating all the weights, the total error of the network is reduced to 0.298371109
- The original error was 0.291027924
- If this process is repeated 10000 times, the error is reduced to 0.0000351085
- At this point, the inputs of 0.05 and 0.1 produce outputs of 0.015912196 (vs 0.01 target) and 0.984065734 (vs 0.99 target)

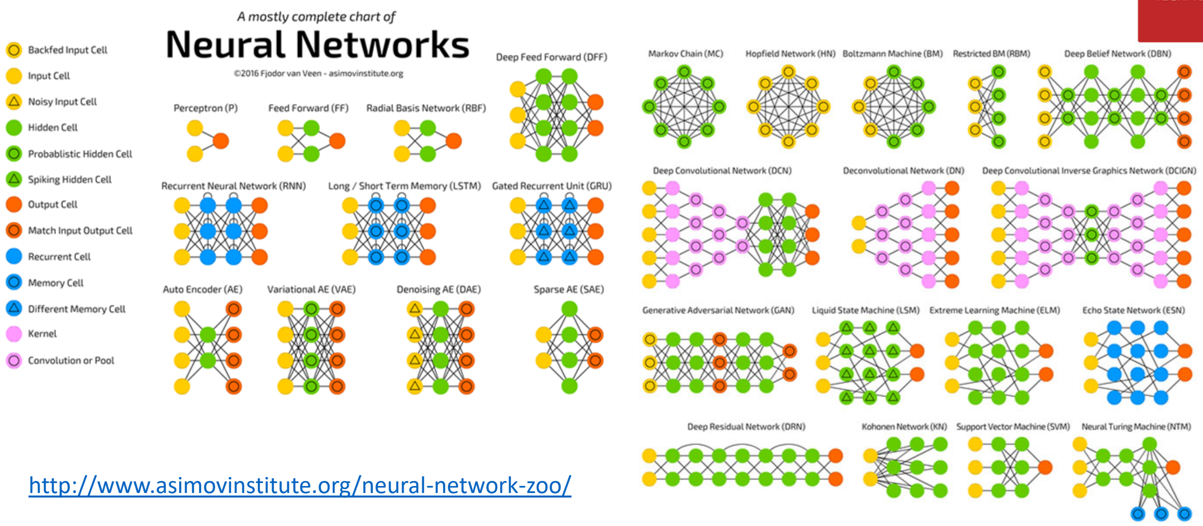
Developing a Neural Network

- Needs a fair degree of mathematical competence
- Not simple (also very time consuming)
- Considerations include:
    - Identification of features in the problem domain worth mapping into inputs/outputs (Problem Representation)
    - Determining the number of hidden layers and nodes
    - Setting up initial parameters – weights, thresholds, increments
    - What learning method to use?
- No rules exist, based on experience (and solid understanding of problem domain)

Weakness of Neural Networks

- Neural networks are *opaque* – it is very hard to check that the weights after training are sensible
- A neural network can never explain how it arrived at a particular conclusion
- Loss of human control, lack of trust since AI cannot explain all decisions and actions, or take responsibility – **Black Box AI**
- Some work is being done on combating the opaqueness problem
- Partnership on AI (https://www.partnershiponai.org) is a step towards safe and trustworthy AI technologies – supported by tech giants including Google, IBM and Microsoft

Types of Neural Networks



http://www.asimovinstitute.org/neural-network-zoo/

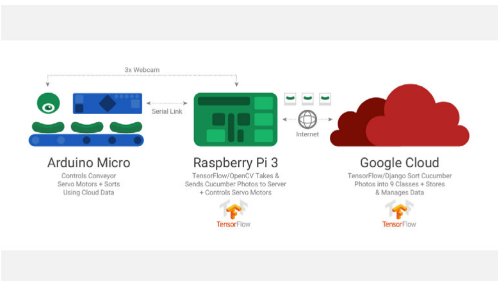
Machine Learning as a Service

SWINBURNE UNIVERSITY OF TECHNOLOGY

- Machine learning is no longer for experts only
- Software developers can use cutting-edge, commercially usable machine learning frameworks
    - Scikit-learn
    - Google Tensorflow
    - PyTorch
    - Microsoft Cognitive Toolkit
    - Theano
    - Caffe
    - Deeplearning4j
- Major cloud providers including Amazon, Google and Microsoft offer machine learning as a service

Cucumber Farming & TensorFlow

SWINBURNE UNIVERSITY OF TECHNOLOGY



- Cucumber classification into 9 different classes using Tensorflow based on size, thickness, color, texture, small scratches, whether they are crooked, whether they have prickles,

https://cloud.google.com/blog/big-data/2016/08/how-a-japanese-cucumber-farmer-is-using-deep-learning-and-tensorflow