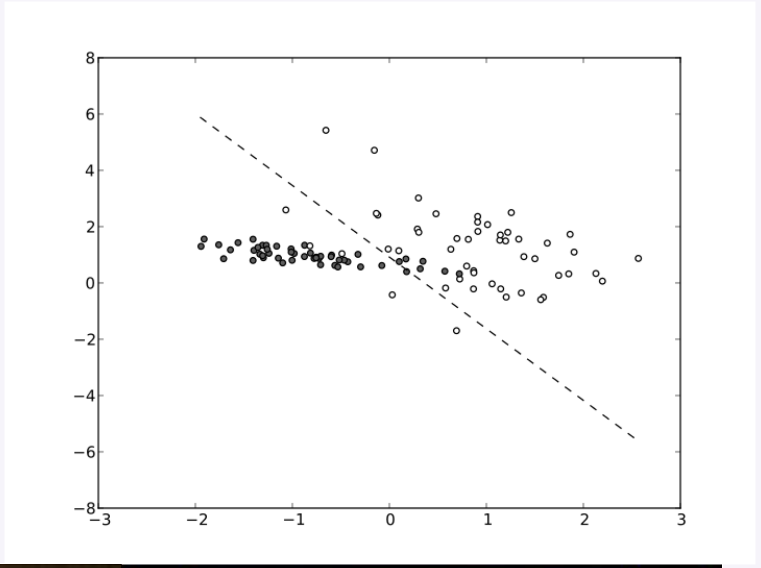
Neural Networks

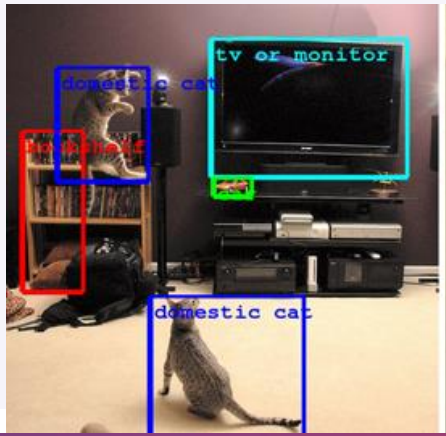
By Laurence Squires

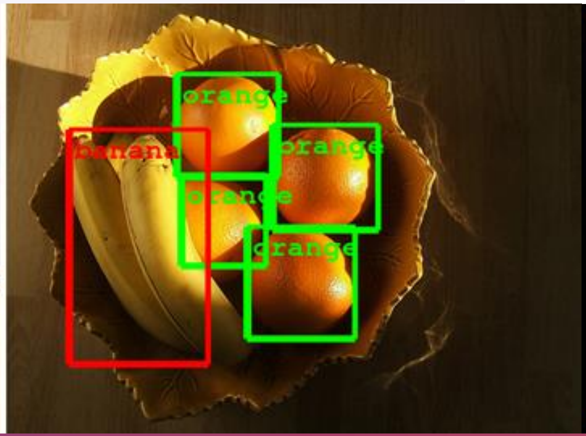
Machine learning

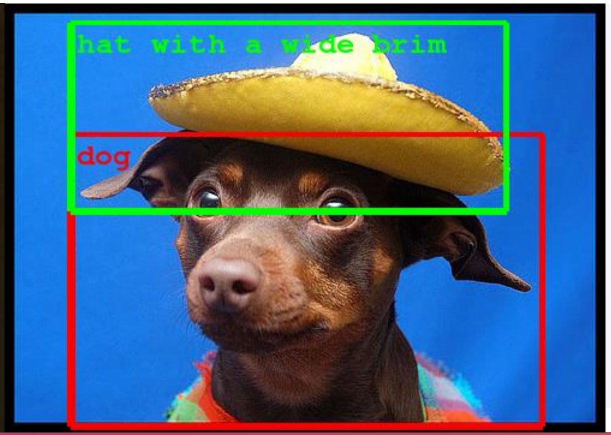
What is it?

- Type of A.I. (possibly the ultimate A.I. ?!?!?!)
- Algorithms that 'learn' how to classify data
- The algorithms slowly change their own variables until they find a solution with the lowest error
- Examples:
    - Google translate
    - Watson
    - NLP (Natural Language Processing)
    - GoogLeNet ImageNet



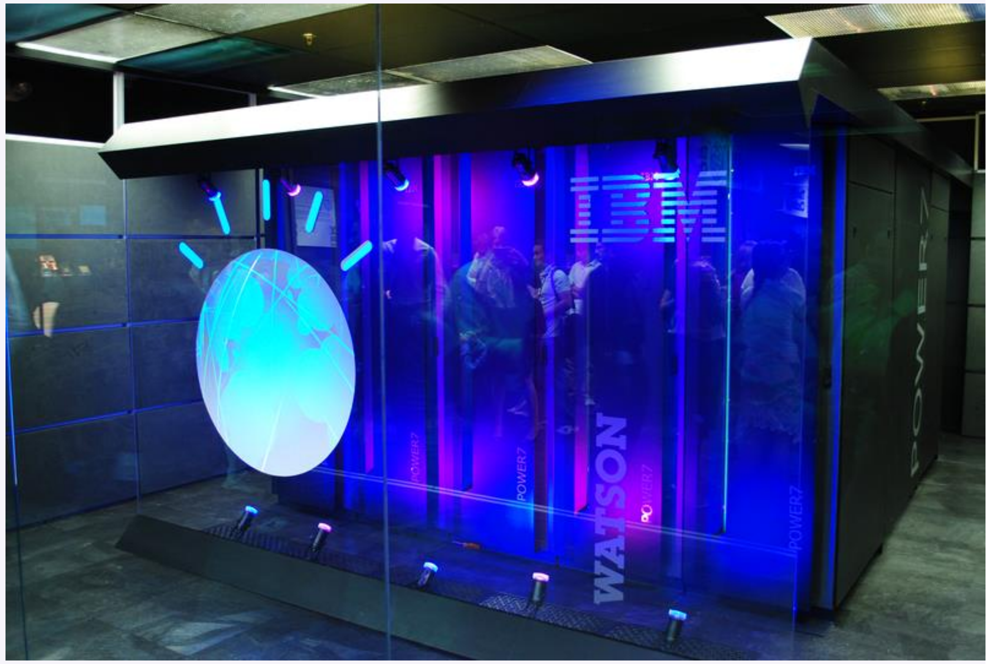






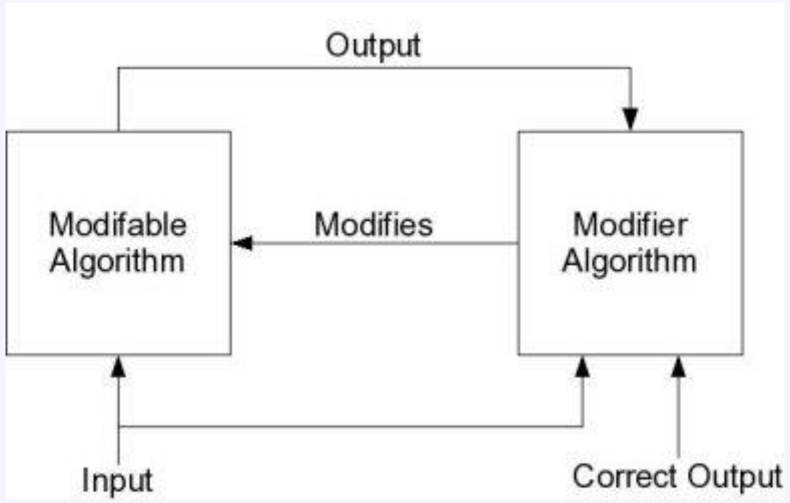
Watson

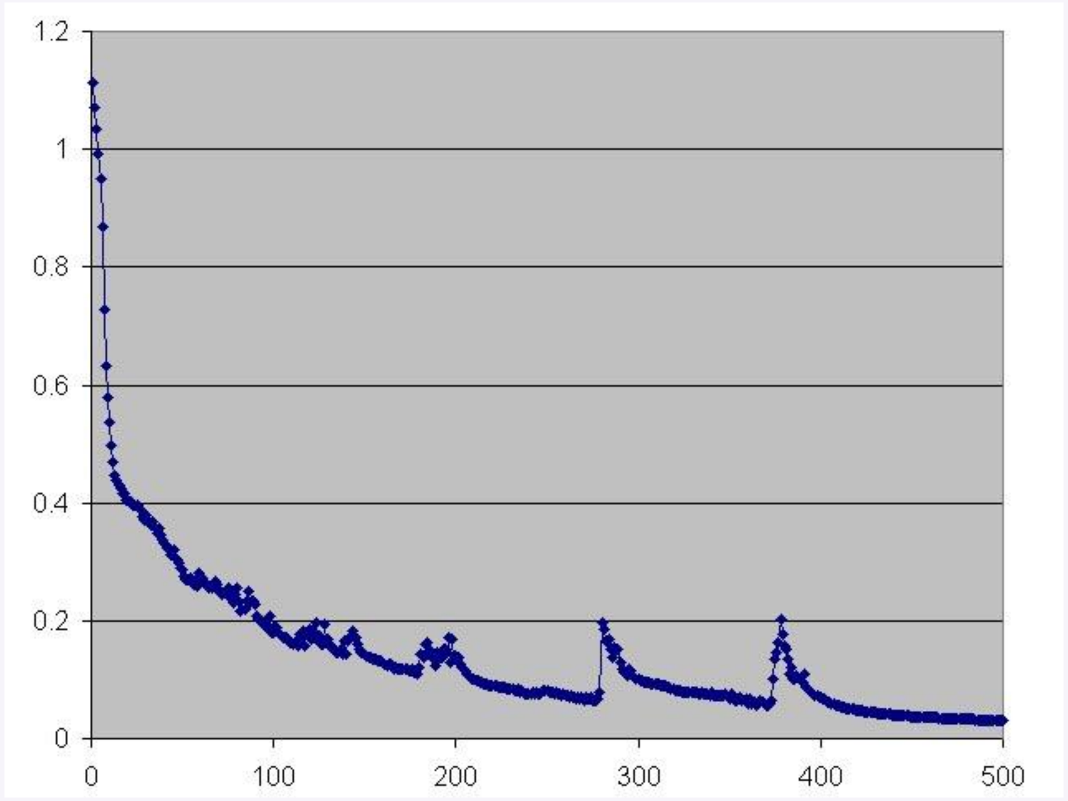
- A supercomputer designed by IBM
- Beat the two world champions at jeopardy by using the information of the whole of Wikipedia and other knowledge sources, taking up only 4 terabytes of storage.
- Now working on utilization management decisions in lung cancer treatment.
- The information banks are constructed and queried using machine learning algorithms.



Normal learning algorithm routine

- Construct an algorithm with random variables
- Have a large set of inputs and their expected outputs (training data)
- Allow the algorithm to learn the pattern/link between within the training data
- Test the algorithm on unseen new data to test its error

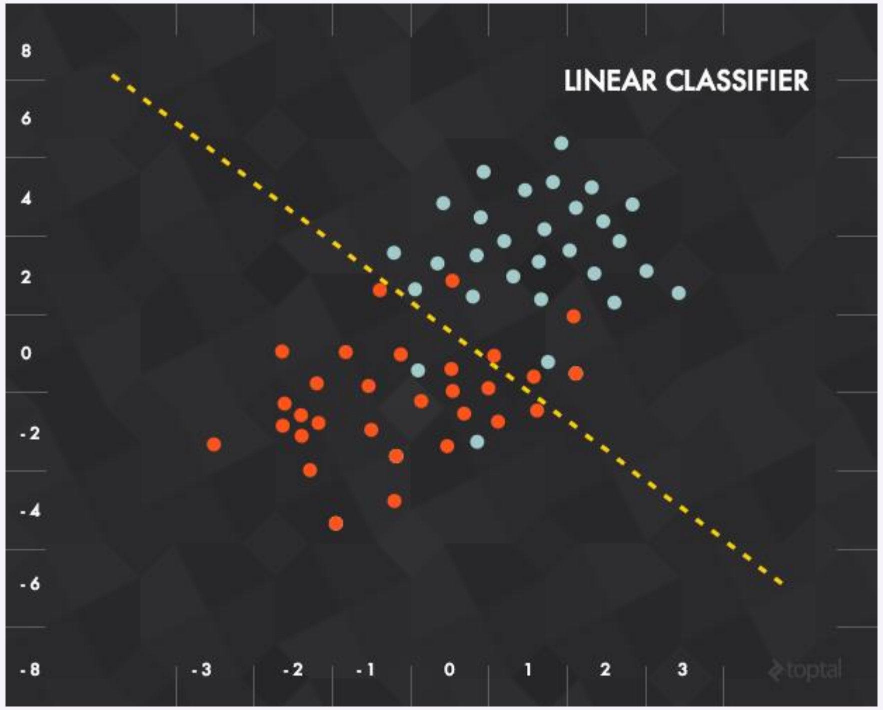




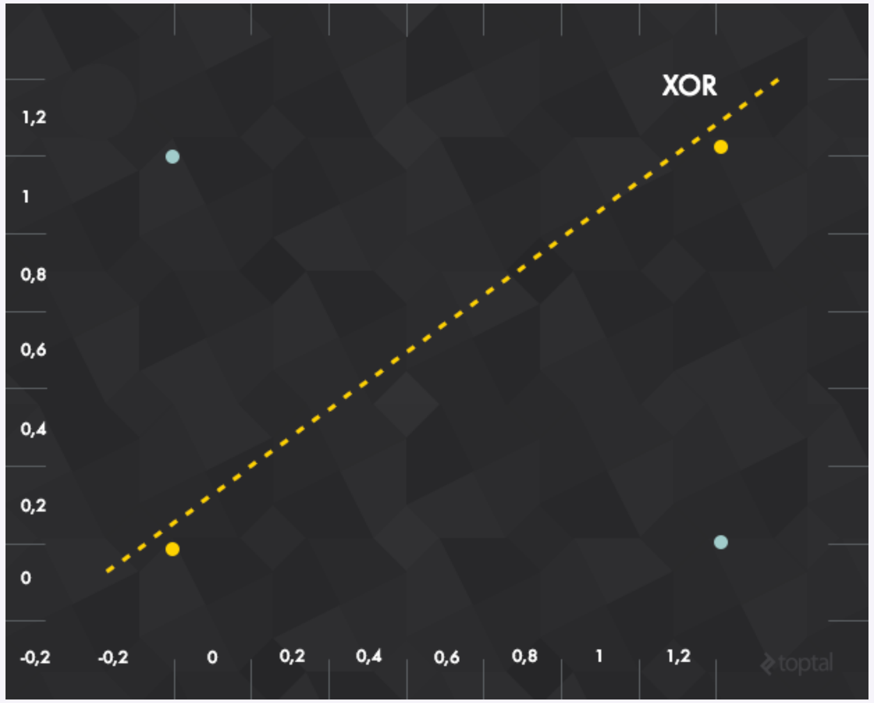
Why networks are useful

Linear classifiers can't solve everything

Best solution with linear



Failing at XOR

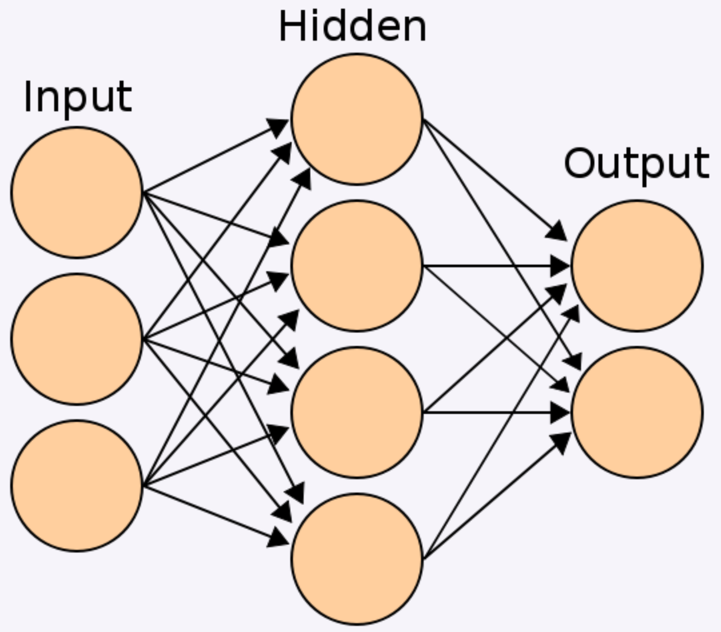


Neural network

Basis

A neural network is designed to mimic the interaction of neurones in the brain.

- An input, output and one or more hidden layers
- Each layer contains nodes
- Each node (after the input nodes) calculates their value based on the inputs from the previous layer
- Each connection between a node has a weight used for the calculation
- Each node has its own bias used for the calculation



Neural network

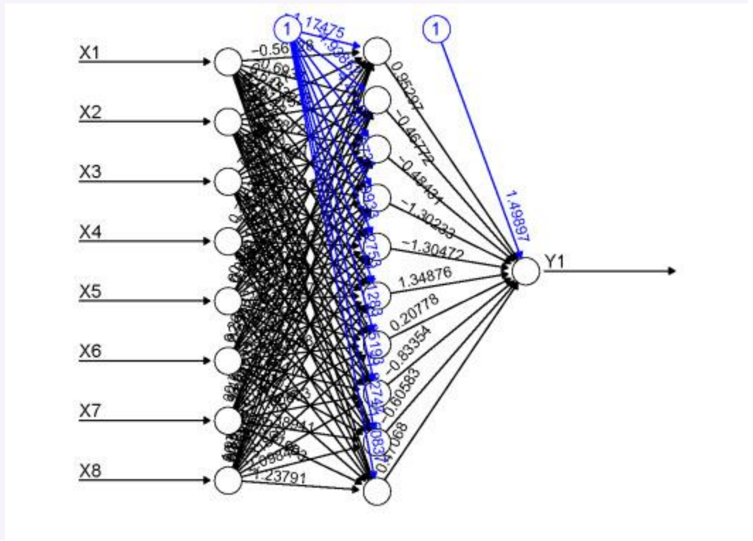
Design based on human neuron interactions

- Input(s) are ‘clamped’ to the input node(s)
- The hidden nodes calculate their value based on the input nodes, the weights of the connections and the bias. (basically maths happens)
- The same things occur for each of the hidden layers (if there are multiple layers) until we reach the end.

- After each iteration of propagation (moving the values forward), the error is calculated.
- Then the error back propagates (moves back through) through the network and adjusts each weight

and bias to minimise the overall error

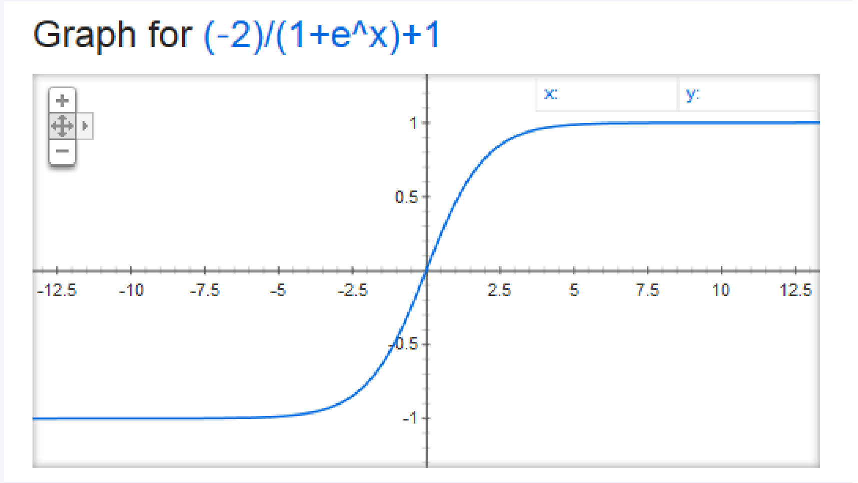
These steps are repeated for all the training data as many times as needed until the network has ‘learnt’ the best setup

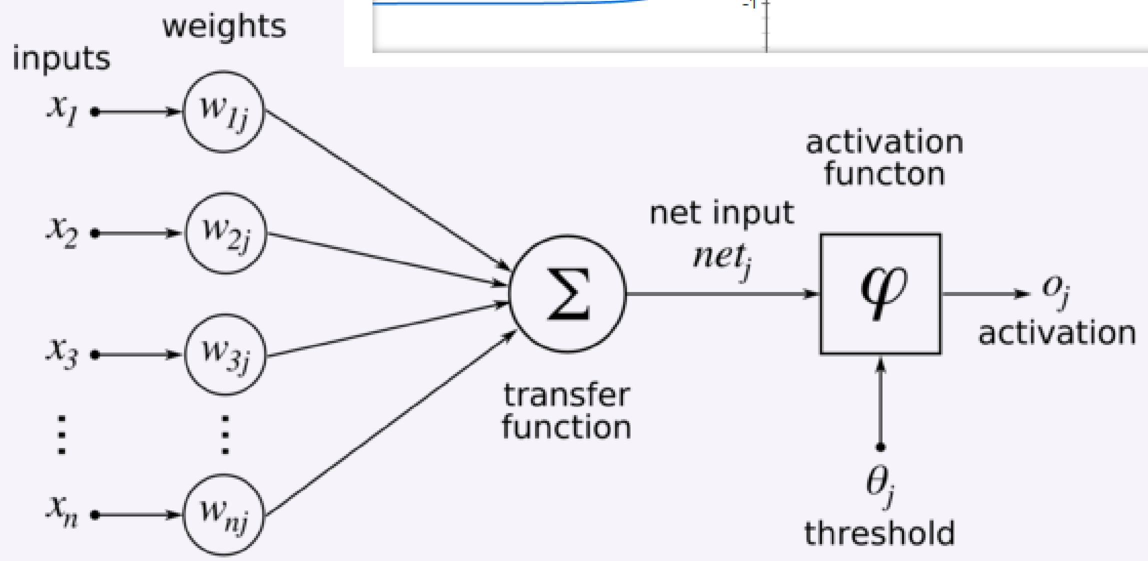


Maths stuff

Hidden nodes

- Each hidden node takes the sum of all its inputs times the corresponding connection weights
- Net input = $\sum(input_n * weight_n)$
- The net gets added to a bias (in this diagram called threshold)
- This value gets passed through the activation function.
- Normally this is a sigmoid function usually tanh
- $output = \tanh(net\ input + bias)$



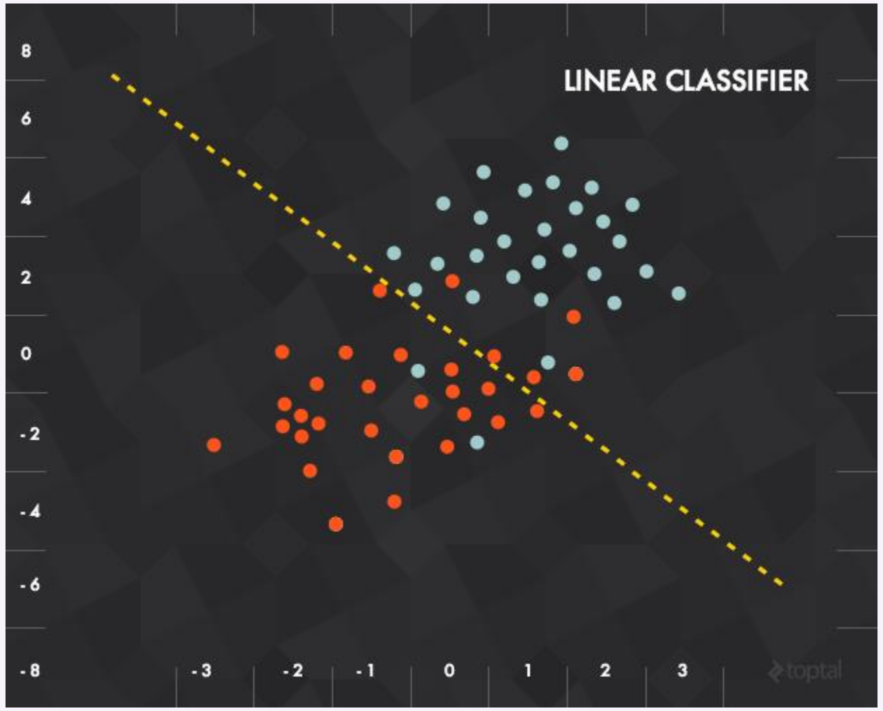


- This happens for each node so requires a lot of computing power, especially if there are many nodes and layers

The output step

Output nodes

- Usually the output nodes have a linear classifier for their activation function
- They still operate like a hidden node
- It depends on the network but this normally provides the best feedback from the network



Back propagation maths stuff

How it actually learns

– Once it has propagated, for each output node, it calculates the mean squared error.

– $E = \frac{1}{2}(t - y)^2$ t = the expected output (target) , y = the networks given output.

– For each node before the weights and bias is updated based on stochastic gradient descent to minimise its affect on the error.

– E.g. the weights between two nodes with respect to their output error.

– To achieve this the non-linear activation function (sigmoid function) must always be differentiable , in this case it is:

Example activation function: $\varphi(z) = \frac{1}{1 + e^{-z}}$ Derivative: $\frac{\partial \varphi}{\partial z} = \varphi(1 - \varphi)$

After some complicated partial derivative calculations using the chain rule, the weight update for a given node has the following form

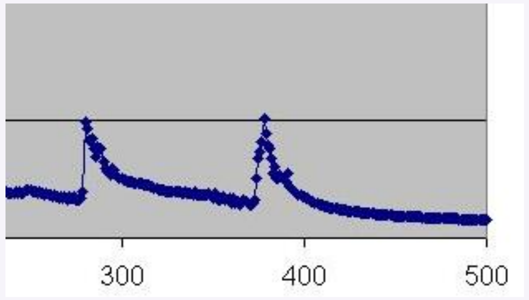
$\Delta w_i = -\alpha \frac{\partial E}{\partial w_i}$ (change in weight = learning rate (a small number) * the derivative of the error over the weight (delta error/delta weight))

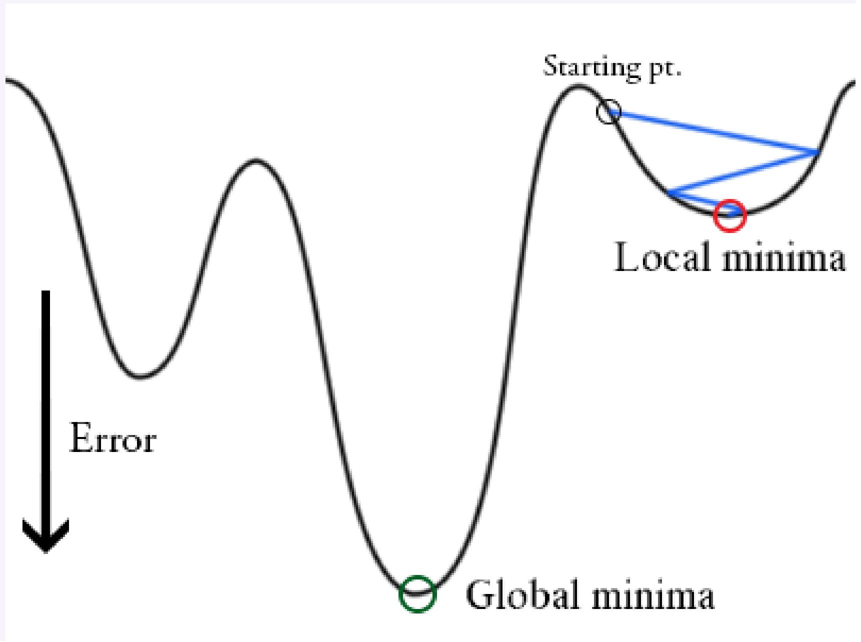
– Where *E* is the output error, and *w_i* is the weight of input *i* to the neuron.

– The learning rate is normally a small number (0.01) and makes sure the network does not vary massively and never settle in a minima of error.

Problems of back propagation

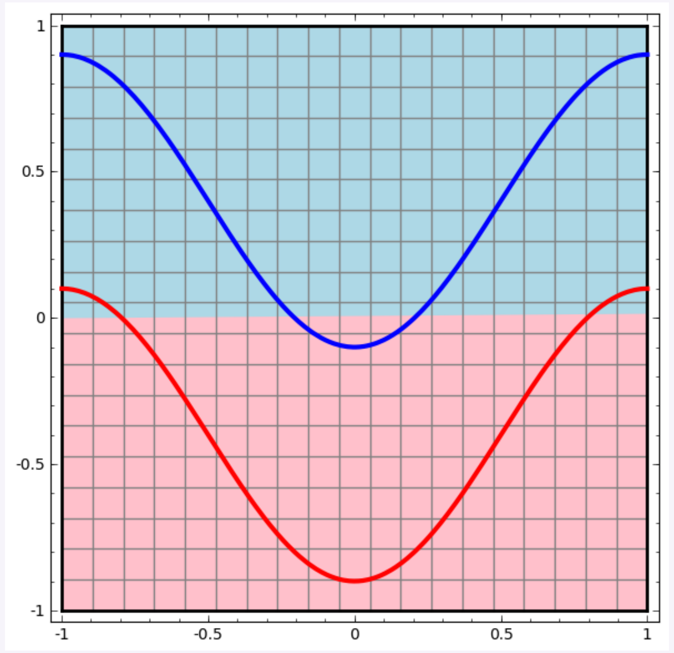
- Sometimes the values get ‘stuck’ in a local minima instead of the global minima
- The derivative displays that they are in a minima but it is impossible to know if this setup gives the lowest error or if there is a better setup (the global minima)
- A simple solution is to save the current network setup and then partially ‘reset’ the network if it reaches a minima
- And have changing learning rates based on previous errors
- This is the reason for the spikes on the original error graph



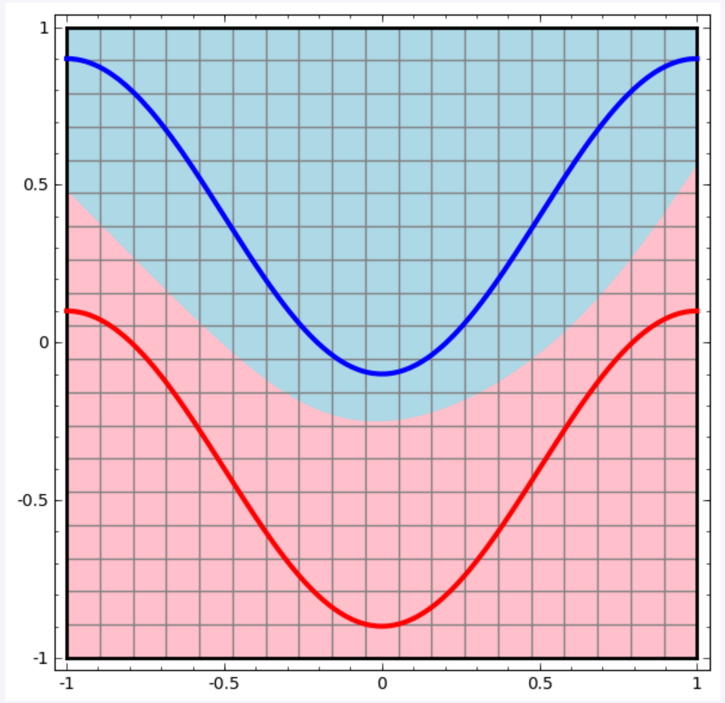


Topological representation of networks

Linear Approach



Neural Network Approach

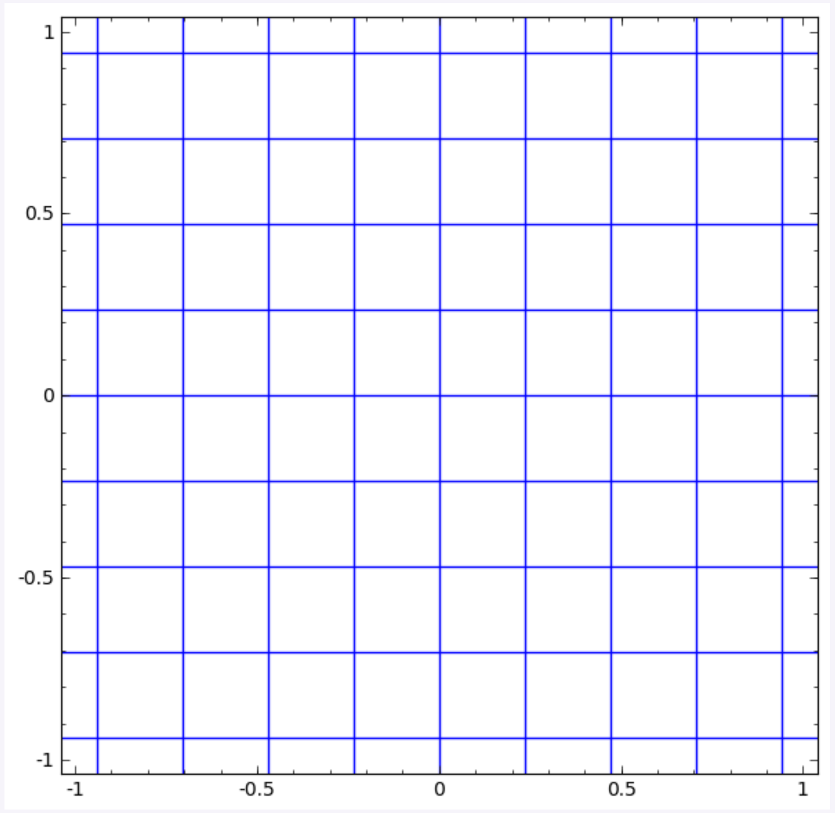


The lines represent the training data, the axis are the inputs and the colour is the expected output, so output is either red or blue and input is 2 numbers (x,y)

The background shows how the networks are best classifying them

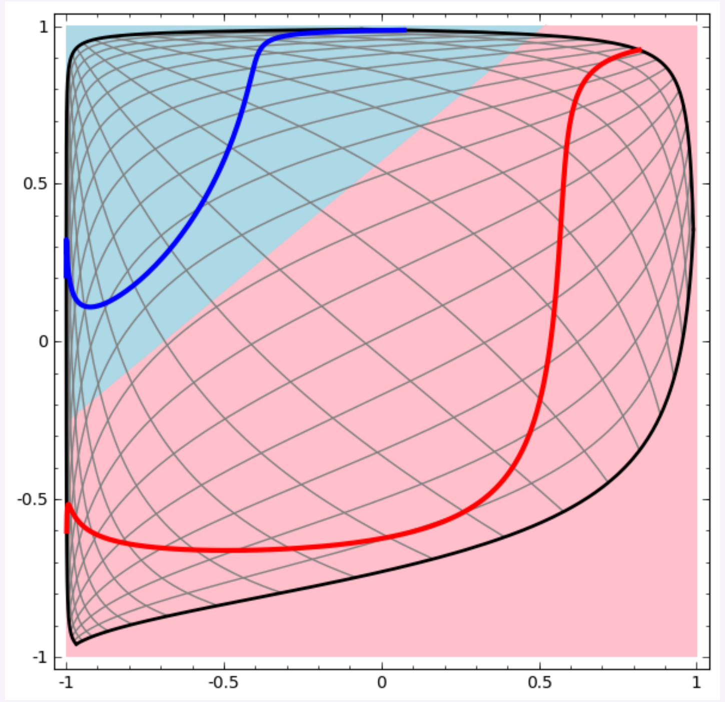
Topological transformations

- The changes that the hidden nodes make to the values can be represented as simple transformations
- Each node does the following transformations:
- The linear “stretching” of the input by the weight
- The linear offsetting of the values by adding the bias
- The linear summation of inputs
- The non-linear transformation of the tanh activation function
- We can attempt to visualize this with this gif

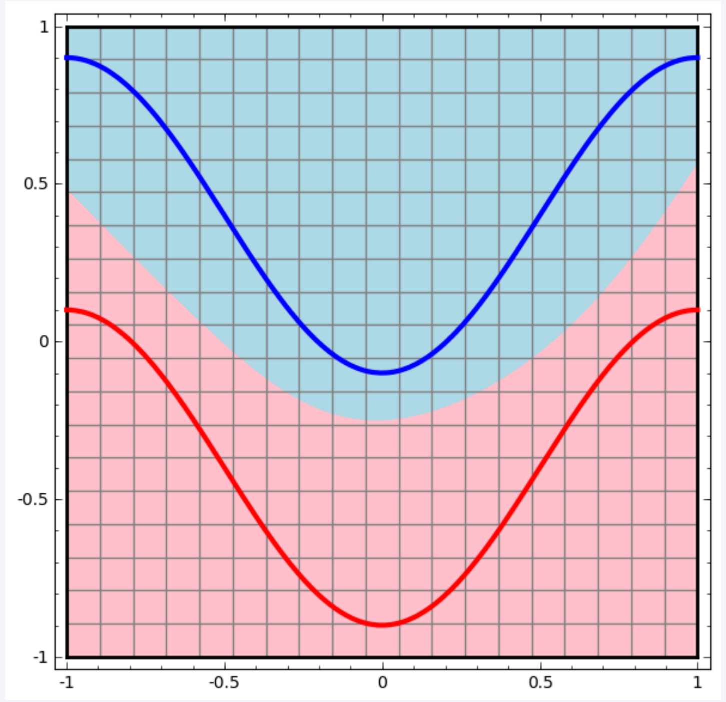


Topological representation of networks

Actual representation (output layer)



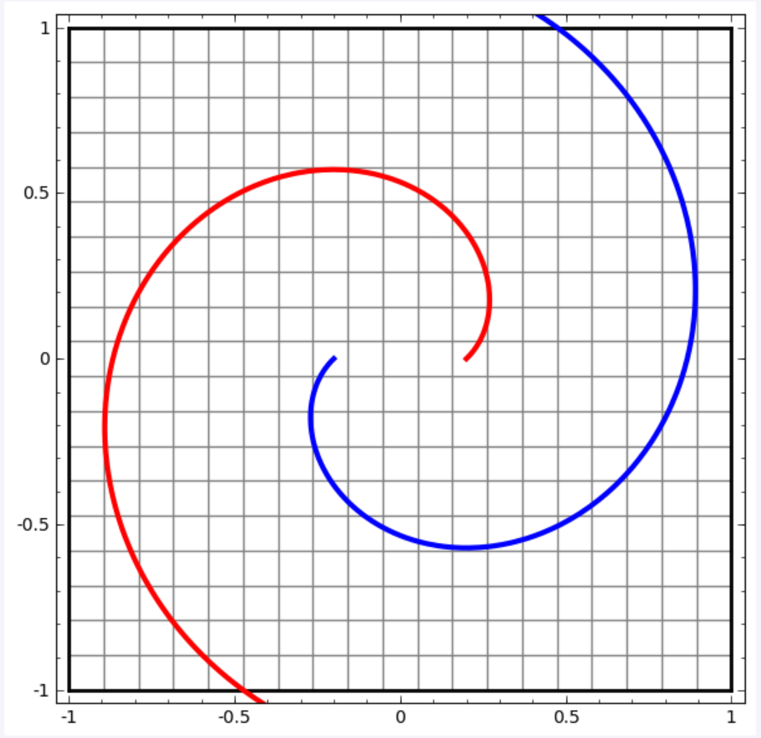
Raw representation (input layer) for comparison



The linear classifier of the final output node is visible here because the transformations done by the hidden nodes are shown as transformations of the actual space.

Topology

Demonstration of learning



This doesn’t show the linear classifier (it’s pretty easy to see where it would go though)

This shows how the network learns the best set of transformations to divide the two sets up for the output nodes.

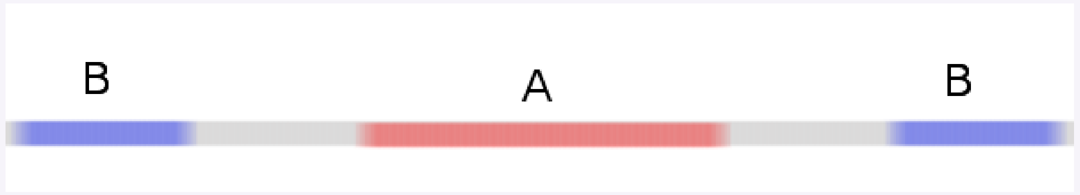
More nodes = better networks

- Every extra hidden node in a layer is a new dimension of transformation.
- So: 1 node = 1 dimensional transformations (not very useful)
- 2 node = 2 dimensional transformations …

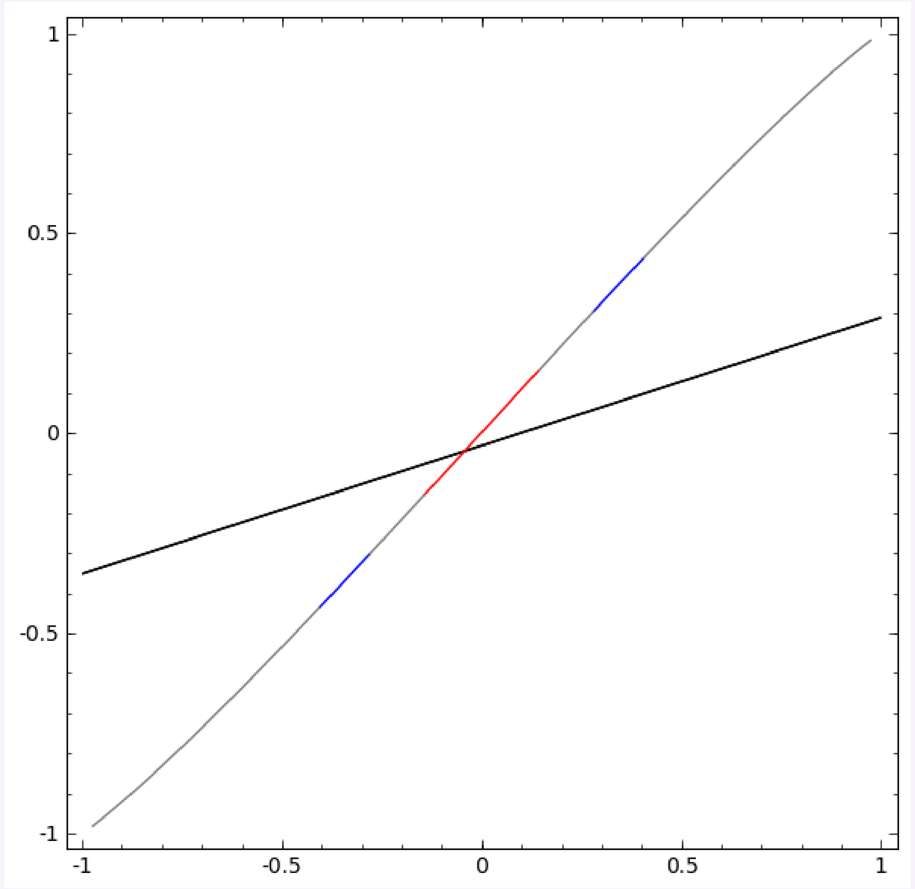
The more dimensions of transformation the easier it is to find a solution to classify data.

More nodes = better networks

- Some data types can’t be solved with 100% accuracy without extra dimensions
- E.G. This one dimensional data set (one input= x , one output = colour (red or blue))



- Its impossible to transform this along the x axis to separate the A and B (crossing over the points is not valid because the activation function is a one-one function). Therefore a neural network with 1 node cannot learn and classify this data.
- Instead 2 nodes/ 2 dimensions are needed to solve this

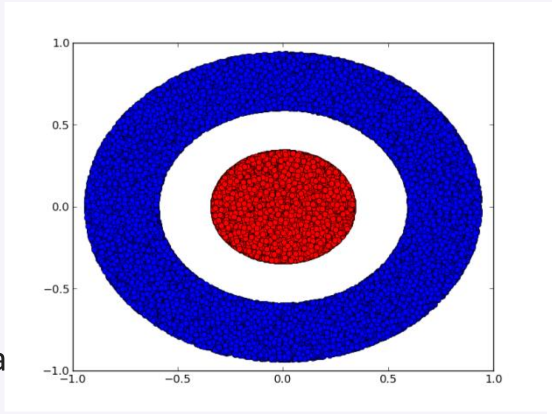


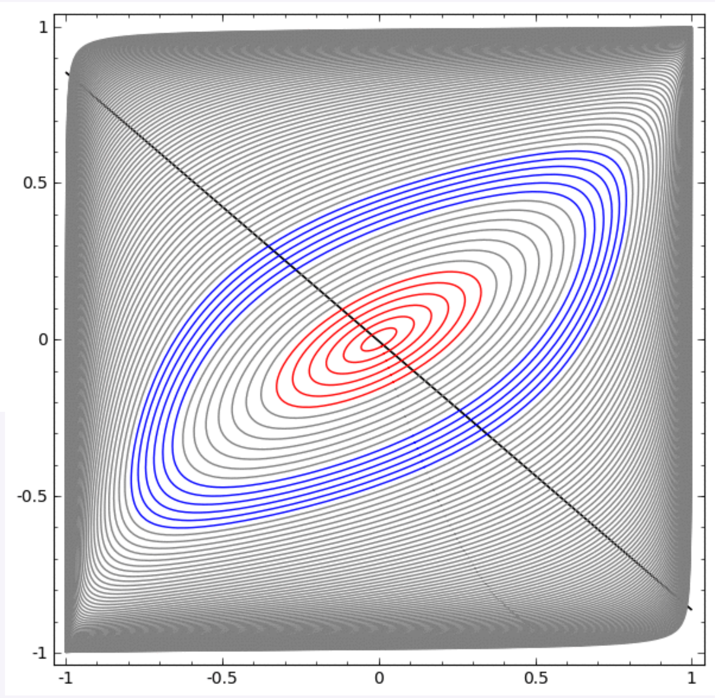
More nodes = better networks

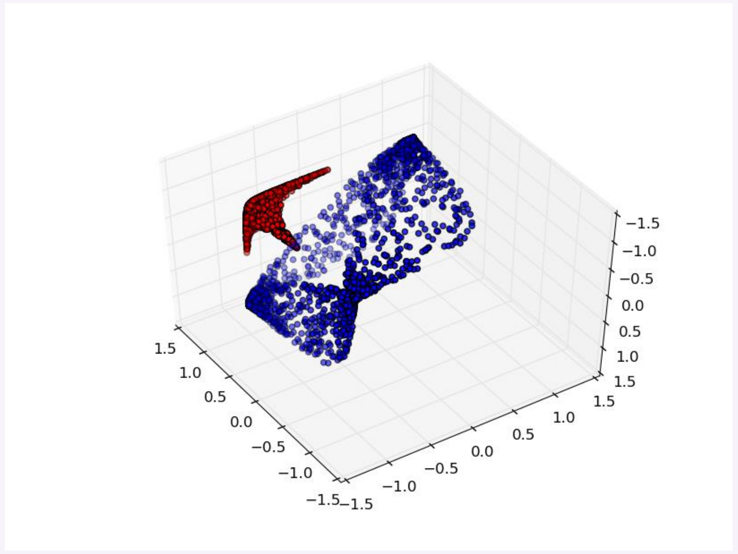
- Another example is a 2D (2 inputs) data set like this:
- This cannot be solved with just 2D transformations/2 nodes
- Here is the 2 node network reaching 80% accuracy. But still failing overall.

- With 3 nodes the solution is easy. The linear classifier forms a plane insteа

of a line

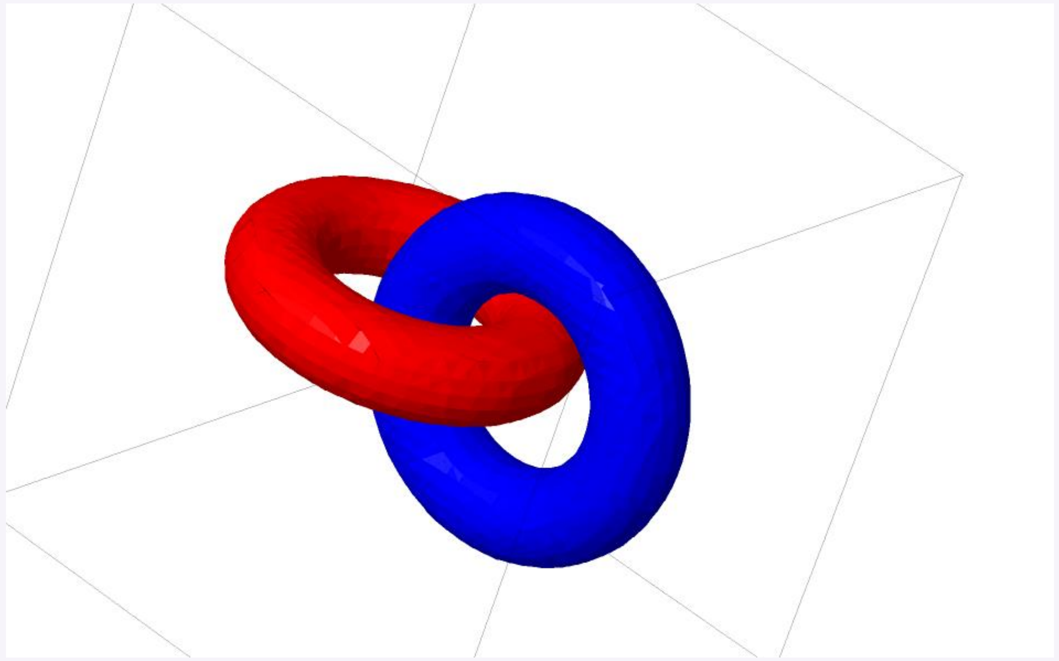






More nodes = better networks

- This 3 input dataset of two linked tori (donuts) requires 4 nodes/4 dimensions to solve
- Therefore, we cannot draw a graph of the solution :'(



Applications

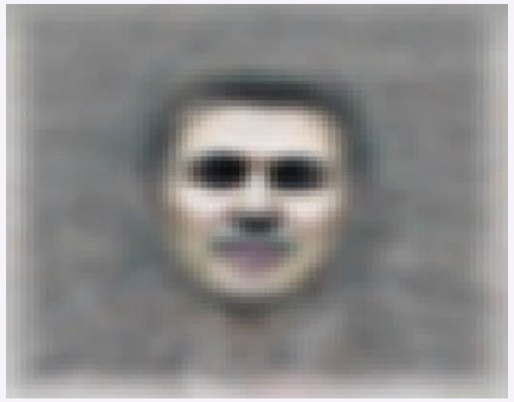
Rat mind reading

- Train a rat to press a lever, which activates a robotic arm. Robotic arm delivers reward to rat.
- Attach a 32-probe array to the rat's brain that can record the activity of parts of its brain at once.
- Train a neural network program to recognize brain-activity patterns during a lever press.
- Neural network can predict movement from the rat's brain activity alone, so when the rat's brain activity indicates that it is about to press the lever, robotic arm moves and rewards the rat - the rat does not need to press the lever, but merely needs to "think" about doing so.

Useful Applications

Image recognition

- Use low resolution image data as the input (it isn't practical to use a 1024x1024 image = 9,437,184 input nodes)
- Train it to recognise if an image contains a entity
- If you ‘reverse’ the neural network you can produce what the network thinks a cat looks like
- E.g. Give the trained network an output for a positive detection of a cat and propagate in reverse, the network then creates image data of what an optimal cat looks like.



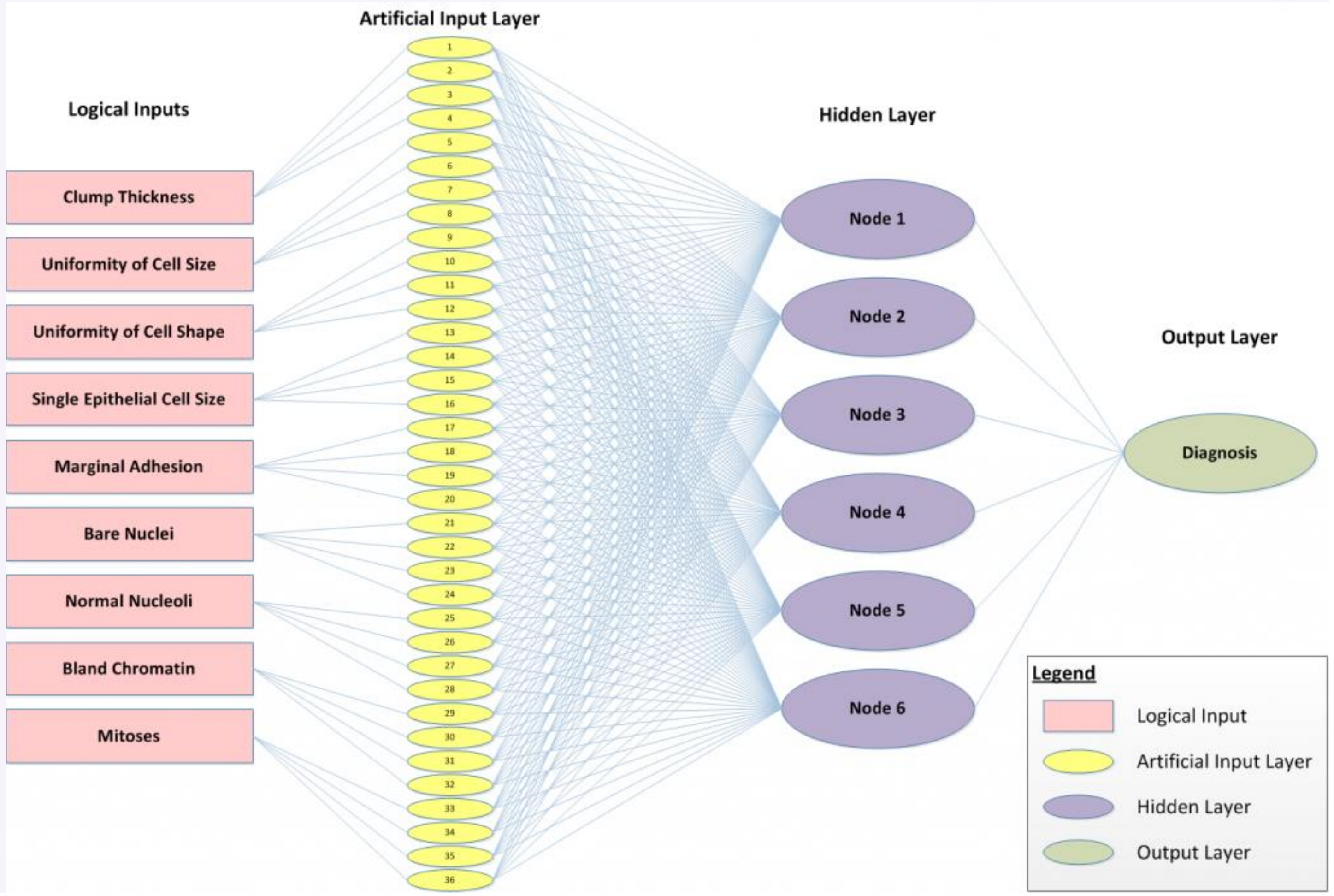
Useful Applications

Cancer detection

– A 17 year won $50,000 for designing this neural network that detects breast cancer from data from samples.

– It diagnoses to within 99.11% accuracy and can only improve.

– Soon super computers with incredibly complex neural networks will easily outperform human (doctor) diagnosis of diseases in terms of accuracy, speed and cost.



Future impact of A.I. (maybe)

- ~2020 Automatic self-driving cars using A.I. become more efficient, safer, faster and cheaper than humans. (e.g. google self driving car)
- ~2030 Exploration, recon, mining and other dangerous jobs become automated
- ~2040 Giant supercomputers with huge databases of knowledge to learn from help to diagnosis patients and other simple human interactions. (e.g. Watson)
- ~2050 Retail jobs replaced and automated (e.g. restocking shelves, taking orders, cooking food)
- ~2060 Simulation of rat or mouse brain occurs
- ~2070 Robots become as physically practical and able as humans. Therefore, vast majority of humans have no jobs as they are being outperformed by A.I. and robotics
- ~2080 Simulation of human brain
- >2100 A.I. becomes more intelligent than humans,
- >2100 A.I. becomes self improving, intelligence of the A.I. is now exponential.