OENG1116 – Modelling and Simulation of Engineering Systems

Artificial Neural Network

(ANN)

Course Lecturer:

Dr Hamid Khayyam

Office: 251-02-34

Phone: 03 9925 4630

Email: hamid.khayyam@rmit.edu.au

Agenda

- History of Artificial Neural Networks
- What Is an Artificial Neural Networks?
- How It Works?
- Learning
  - Learning Paradigms
    - Supervised Learning
    - Unsupervised Learning
    - Reinforcement Learning
- Neural Network Training Algorithms
- Example
- Conclusion and References

Agenda

- History of Artificial Neural Networks
- What Is an Artificial Neural Networks?
- How It Works?
- Learning
  - Learning Paradigms
    - Supervised Learning
    - Unsupervised Learning
    - Reinforcement Learning
- Neural Network Training Algorithms
- Example
- Conclusion and References

History of the Artificial Neural Networks

- 1940s, the decade of the first electronic computer.
- 1957 Rosenblatt introduced the perceptron. After this, the development of Artificial Neural Networks (ANNs) has proceeded. Rosenblatt's original perceptron model contained only one layer.
- 1960 the use of the Multi- Layer Perceptron (MLP) was complicated by the lack of a appropriate learning algorithm.
- 1972, Kohonen used Learning Vector Quantization (LVQ) for network

History of the Artificial Neural Networks

- 1974, Werbos introduced backpropagation algorithm for the three-layered perceptron.
- 1982, Kohonen introduced Self-Organizing Map (SOM) based on LVQ. SOM is a certain kind of topological map which organizes itself based on the input patterns.
- 1982, Hopfield brought a network consists of only one layer whose neurons are fully connected with each other.
- 1986, Rummelhart and Mclelland used the propagation algorithm for a MLP.

History of the Artificial Neural Networks

- 1988, Broomhead & Lowe introduced the Radial Basis Function (RBF) networks.

- 2007,Bengio & 2012, Krizhevsky Deep Neural Networks also called convolutional networks, are composed of multiple levels of nonlinear operations, such as neural nets with many hidden layers

- Since then, research on ANN has remained active, leading to many new network types, as well as hybrid algorithms and hardware for neural information processing.

Agenda

- History of Artificial Neural Networks
- What Is an Artificial Neural Networks?
- How It Works?
- Learning
  - Learning Paradigms
    - Supervised Learning
    - Unsupervised Learning
    - Reinforcement Learning
- Neural Network Training Algorithms
- Example
- Conclusion and References

Artificial Neural Network

- Artificial Neural Network (ANN) is a parallel distributed model consists of a pool of simple processing units which communicate by sending signals to each other over a large number of weighted connections.

- ANNs have the ability to learn and model of non-linear and complex systems.

- ANNs can making the model generalize and predict data.

Advantages / Disadvantages and ANN Applications

Advantages

- ✓ Adapt to unknown situations
- ✓ Powerful, it can model complex functions.
- ✓ Ease of use, learns by example, and very little user domain specific expertise needed

Disadvantages

- ➢ Forgets
- ➢ Not exact
- ➢ Large complexity of the network structure

A few application:

- ❖ Image Processing
- ❖ Forecasting
- ❖ Signal processing
- ❖ Industrial automation
- ❖ Energy system
- ❖ Control
- ❖ Robotics
- ❖ Pattern recognition
- ❖ Industrial optimization
- ❖ Etc

Artificial Neural Network

ANN:

- A set of processing units (cells).
- A state of activation for every unit.
- Connections between the units.
- A propagation rule.

- An activation function,
- An external input for each unit.
- A method for information gathering
- An environment within which the system must operate.

Computers vs. Neural Networks

"Standard" Computers

- one CPU
- fast processing units
- reliable units
- static infrastructure

Neural Networks

- highly parallel processing
- slow processing units
- unreliable units
- dynamic infrastructure

Why Artificial Neural Networks?

- There are two basic reasons why we are interested in building Artificial Neural Networks (ANNs):

- **Technical viewpoint:** Some problems such as character recognition or the prediction of future states of a system require massively parallel and adaptive processing.

- **Biological viewpoint:** ANNs can be used to replicate and simulate components of the human brain, thereby giving us insight into natural information processing.

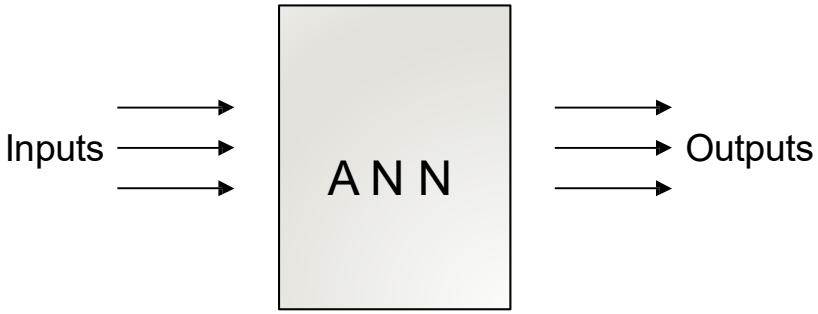
Agenda

- History of Artificial Neural Networks
- What Is an Artificial Neural Networks?
- How It Works?
- Learning
  - Learning Paradigms
    - Supervised Learning
    - Unsupervised Learning
    - Reinforcement Learning
- Neural Network Training Algorithms
- Example
- Conclusion and References

How Artificial Neural Network works

- Simplified model of the brain which is Artificial Neural Networks (ANN)

- Transforms inputs into outputs to the excellent of its ability



Brain Processes Information

In our brain, there are billions of cells called neurons, which processes data/ information in the form of electric signals.

External information/stimuli is received by the dendrites of the neuron, processed in the neuron cell body, converted to an output and passed through the Axon to the next neuron.

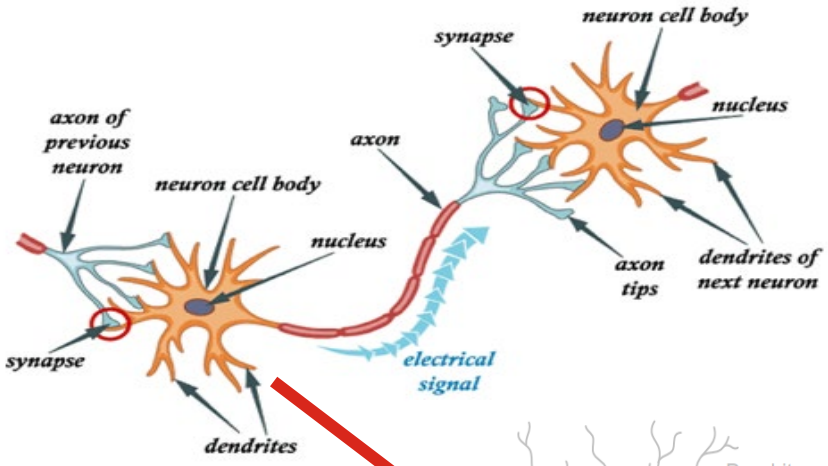
The next neuron can choose to either accept it or reject.

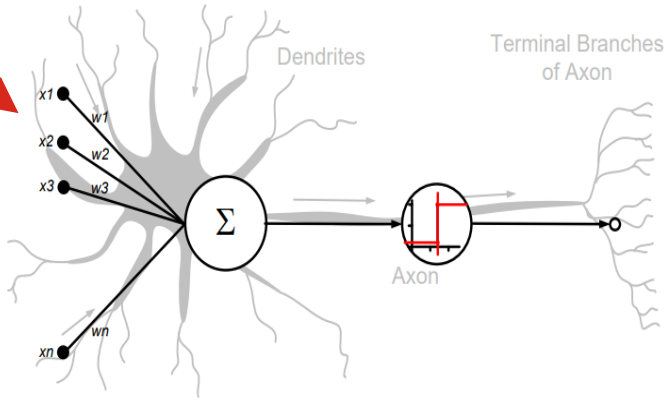
Step 1: External signal received by dendrites

Step 2: External signal processed in the neuron cell body

Step 3: Processed signal converted to an output signal and transmitted through the Axon

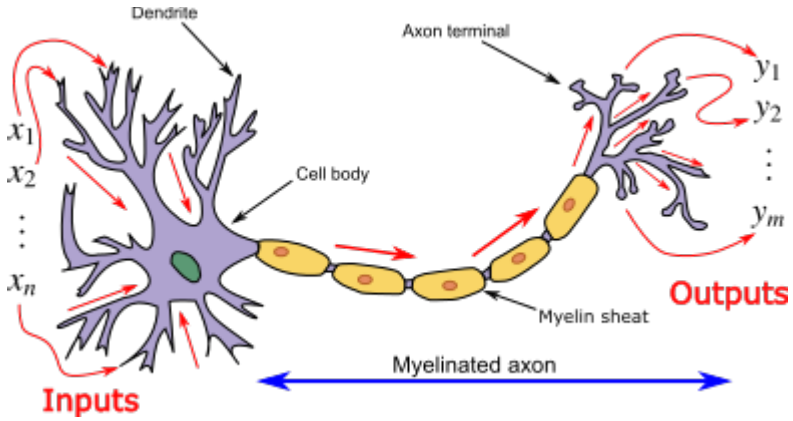
Step 4: Output signal received by the dendrites of the next neuron through the synapse



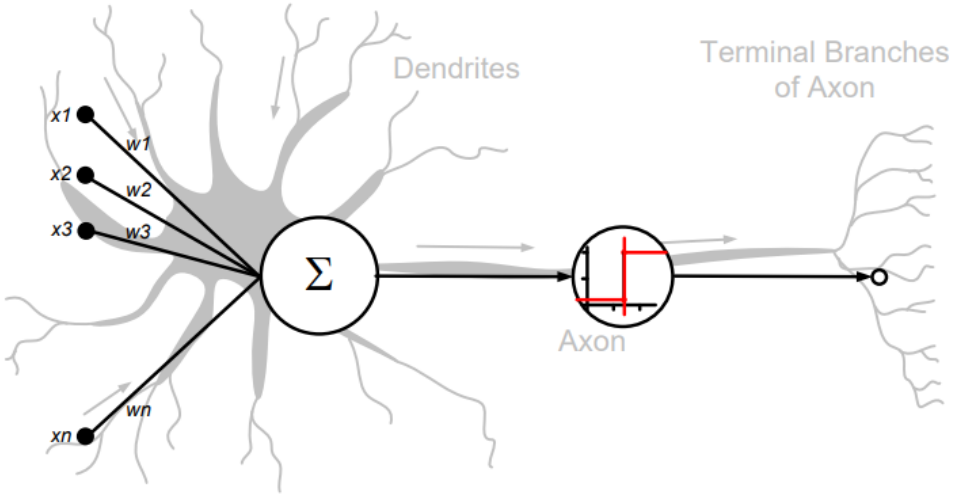


Artificial Neural Network processing

ANN uses the processing of the brain as a basis to develop algorithms that can be used to model complex patterns and prediction problems.

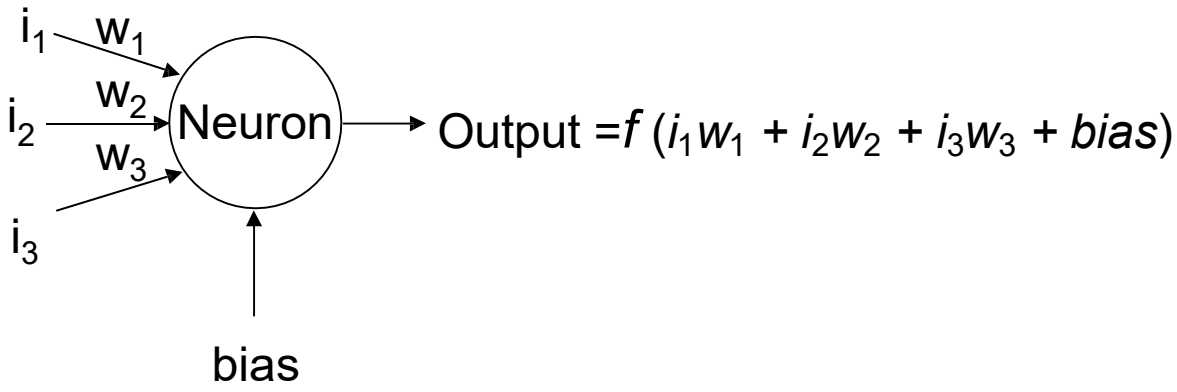


Neural cell	Artificial neuron
Soma	Neuron
Dendrite	Input
Synapse	Weights
Axon	Output



How Do ANN Work?

- The output of a neuron is a (activation) function of the weighted sum of the inputs plus a bias



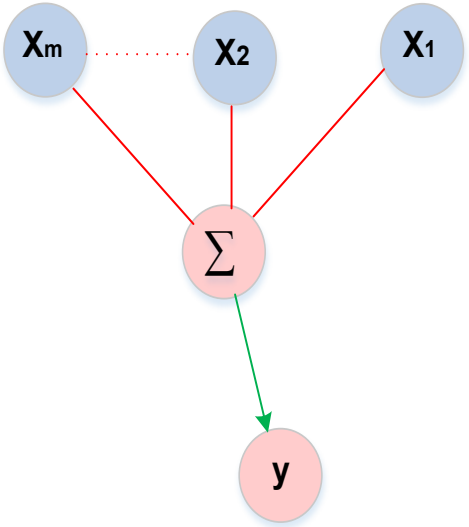
- The function of the entire neural network is simply the computation of the outputs of all the neurons
  - An entirely deterministic calculation

How Do ANN Work?

Input

Processing

Output



$$\sum = X_1 + X_2 + \ldots + X_m = y$$

How Do ANN Work?

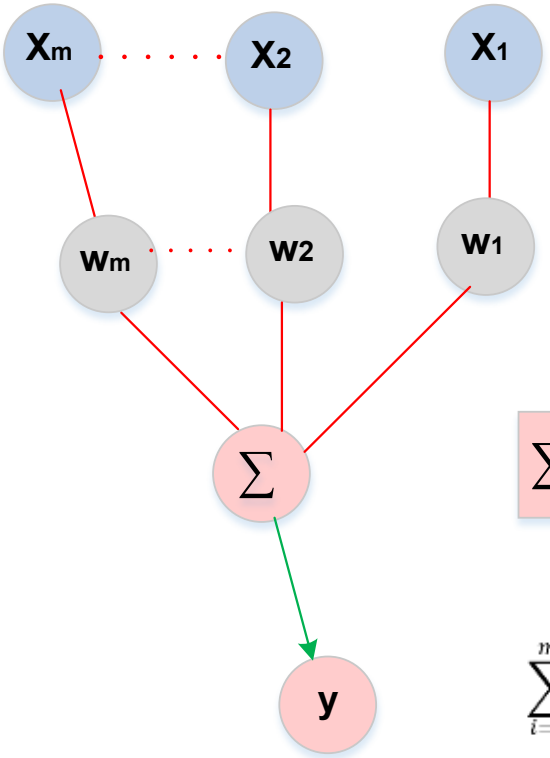
- Not all inputs are equal.

Input

Weight

Processing

Output



$$\sum = X_1w_1 + X_2w_2 + \ldots\ldots + X_mw_m = y$$

$$\sum_{i=1}^{m} X_i w_i = y$$

How Do ANN Work?

- The signal is not passed down to the next neuron

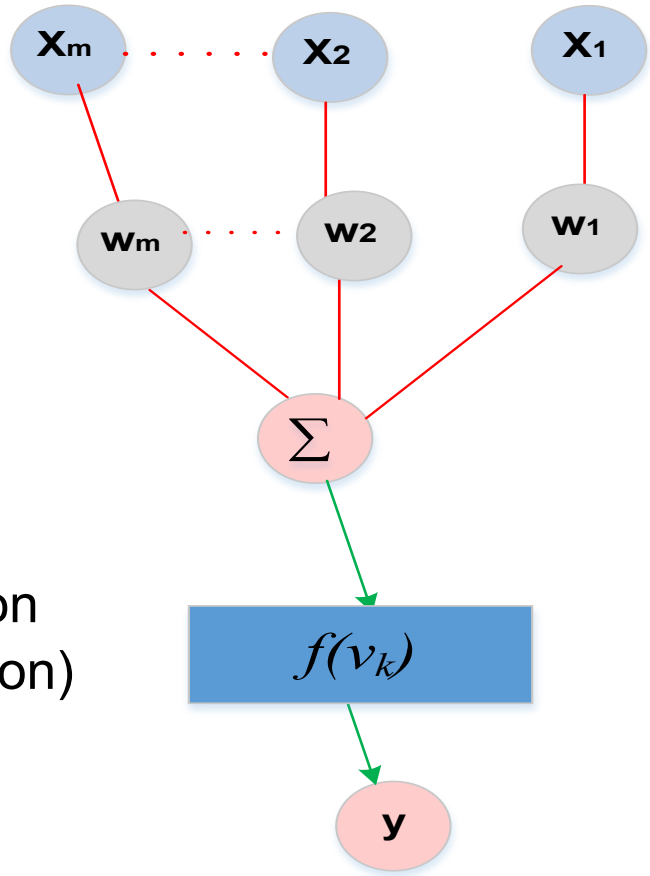
Input

Weight

Processing

Transfer function
(Activation function)

Output

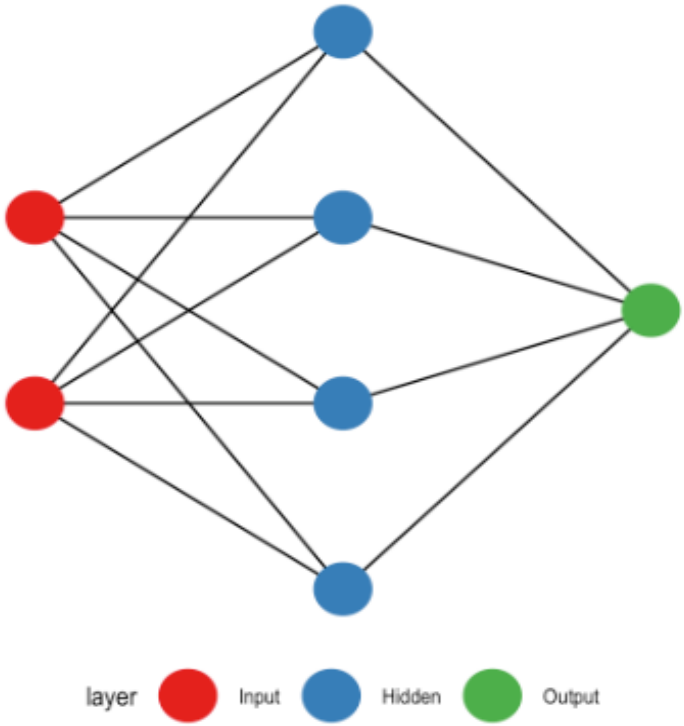


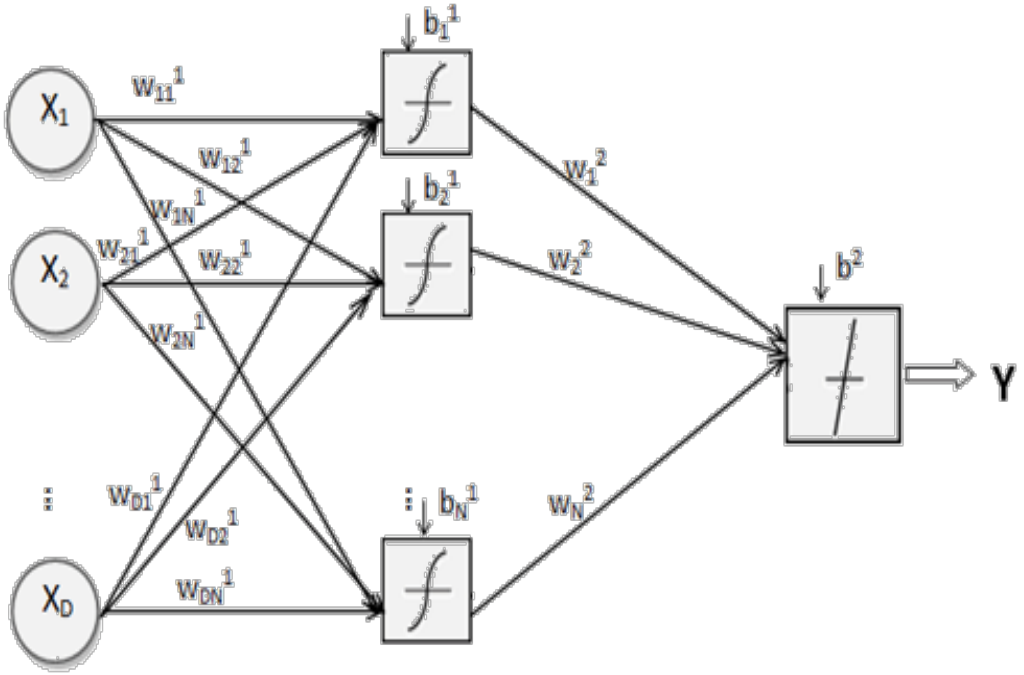
$$\sum_{i=1}^{m} X_i w_i = v_k$$

$$y_k = f(v_k)$$

How Do ANNs Work?

- Three types of layers: Input, Hidden, and Output
- The output is a function of the input, that is affected by the weights, and the transfer functions





Forward Propagation

Artificial Neural Networks

An ANN can:

- Compute any computable function, by the appropriate selection of the network topology and weights values.

- Learn from experience!
  - Specifically, by trial and error

  Training is transporting the information and knowledge, demonstration in a manner that instructs the trainee.

  Learning is the process of absorbing that information to increase skills and abilities and make use of it under a variety of contexts.

Learning by Trial and Error

Continuous process of:

- **Trial:** Processing an input to produce an output (in terms of ANN: compute the output function of a given input)
- **Evaluate:** Evaluating this output by comparing the actual output with the expected output.
- **Adjust:** Adjust the weights.

How Do ANN Work?

- Set initial values of the weights randomly.
- Input: truth table of the XOR

Input 1	Input 2	Output
0	0	0
0	1	1
1	1	0
1	0	1

- Do
  - Read input (e.g. 0, and 0)
  - Compute an output (e.g. 0.60543)
  - Compare it to the expected output. (Diff= 0.60543)
  - Modify the weights accordingly.
  - Loop until a condition is met
  - Condition: certain number of iterations
  - Condition: error threshold

ANN's Design Issues

- Initial weights (small random values$\in$ [1,1])
- Transfer function (how the inputs and the weights are combined to produce output?)
- Error estimation
- Weights adjusting
- Number of neurons
- Data representation
- Size of training set

Transfer (Activation) Functions

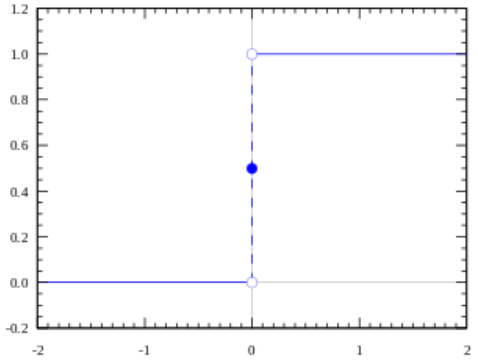
- Applied to the weighted sum of the inputs of a neuron to produce the output

$$\text{Output} = y = f(i_1 \times w_1 + i_2 \times w_2 + i_3 \times w_3 + bias)$$

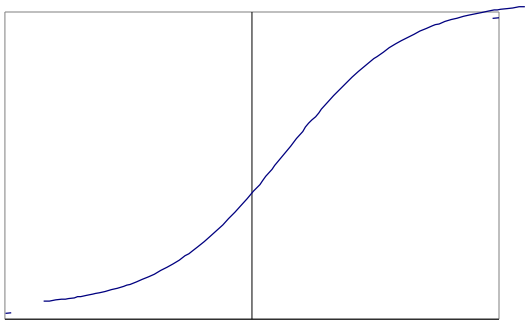
  - Linear: The output is proportional to the total weighted input.
  - Threshold: The output is set at one of two values, depending on whether the total weighted input is greater than or less than some threshold value.
  - Non linear: The output varies continuously but not linearly as the input changes.

Transfer (Activation) Functions

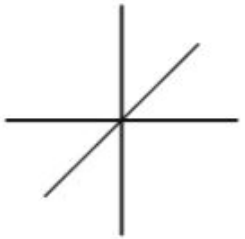
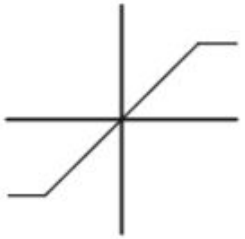
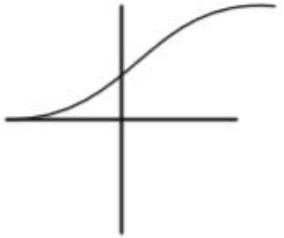
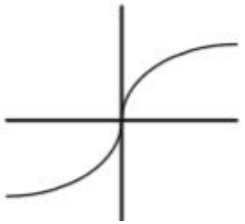
- Step (**linear**) function:
  - Alternatively, *A = 1* if *y>* threshold, *0* otherwise



- Majority of ANN's use Sigmoid (**non-linear**) functions
  - Smooth, continuous, and monotonically increasing (derivative is always positive)
  - Bounded range - but never reaches max or min
    - Consider "ON" to be slightly less than the max and "OFF" to be slightly greater than the min



Transfer (Activation) Functions

Name	Function	Graphs
Linear	$f(x) = x$	
Symmetric–saturating–linear	$f(x) = \begin{cases} \delta & x \geq \theta \\ x & -\theta \leq x \leq \theta \\ -\delta & x \leq -\theta \end{cases}$	
Log sigmoid	$f(x) = \frac{1}{1+e^{-\alpha x}}\ \alpha > 0$	
Tangent sigmoid	$f(x) = \left(\frac{2}{1+e^{-\alpha x}}\right) - 1\ \ \alpha > 0$	

ANN Weights and Training?

- The weights in a neural network are the most important factor in determining its function

- Training is the act of presenting the network with some sample data and modifying the weights to better approximate the desired function

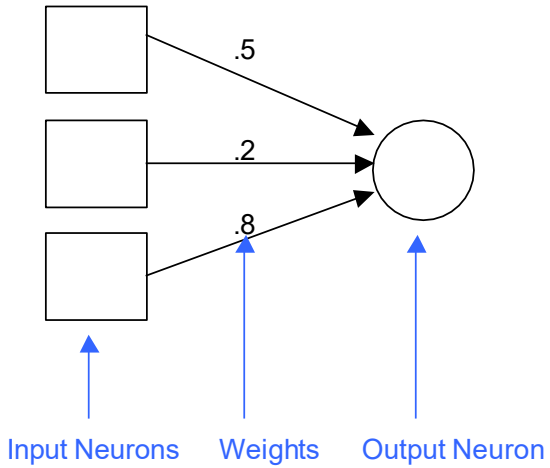
Iteration for ANN Weights ?

- Epoch
  - One iteration through the process of providing the network with an input and updating the network's weights
  - Typically many epochs are required to train the neural network

Perceptron

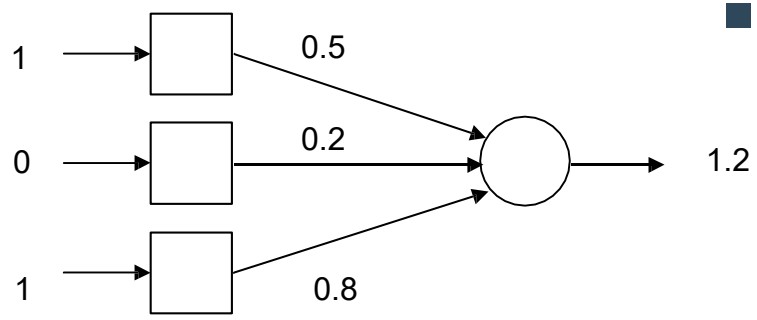
- First neural network with the ability to learn
- Made up of only input neurons and output neurons
- Input neurons typically have two states:

  ON and OFF
- Output neurons use a simple threshold activation function
- In basic form, can only solve linear problems
  - Limited applications



Perceptron Learning

- Uses supervised training
- If the output is not correct, the weights are adjusted according to the formula:



- $w_{new} = w_{old} + \alpha\ (desired - output)*input$

α is the learning rate

(1 * 0.5 + 0 * 0.2 + 1 * 0.8)= 1.3

Assuming Output Threshold = 1.2

1.3 > 1.2

Assume Output was supposed to be 0
➔update the weights

Assume $\alpha = 1$

$W_{1new} = 0.5 + 1*(0 - 1) * 1 = -0.5$
$W_{2new} = 0.2 + 1*(0 - 1) * 0 = 0.2$
$W_{3new} = 0.8 + 1*(0 - 1) * 1 = -0.2$

Multilayer Feedforward Networks

- Most common neural network

- An extension of the perceptron
  - Multiple layers
    - The addition of one or more “hidden” layers in between the input and output layers
  - Activation function is not simply a threshold
    - Usually a sigmoid function
  - A general function approximator
    - Not limited to linear problems

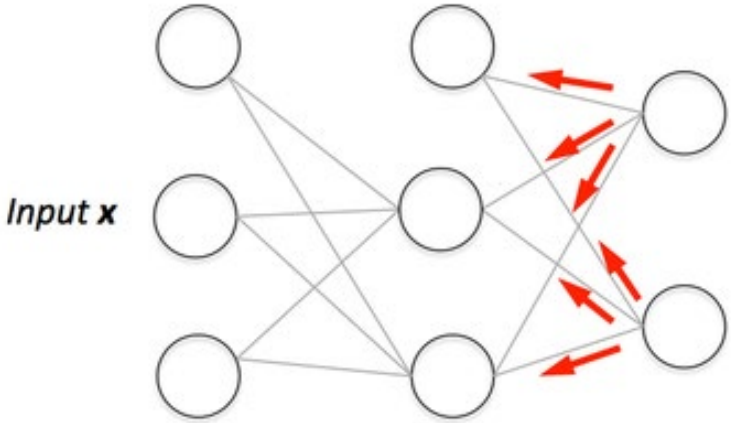
- Information flows in one direction
  - The outputs of one layer act as inputs to the next layer

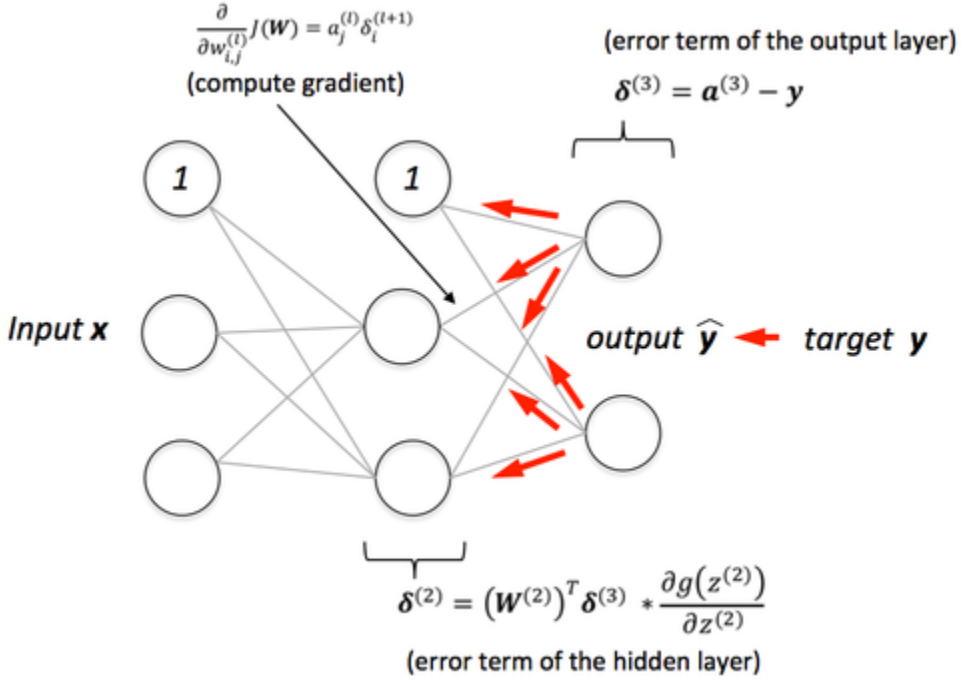
Back Propagation

- Most common method of obtaining the many weights in the network
  - A form of supervised training

Back Propagation

- Back-propagation is an example of supervised learning is used at each layer to minimize the error between the layer's response and the actual data
- The error at each hidden layer is an average of the evaluated error
- Hidden layer networks are trained this way





Back Propagation

- Most common measure of error is the Mean Square Error (MSE):

$$E = (target - output)^2$$

- Partial derivatives of the error (with respect to) w.r.t the weights:
  - Output Neurons:

$$\text{let: } \delta_j = f'(net_j)\,(target_j - output_j)$$
$$\partial E/\partial w_{ji} = -output_i\,\delta_j$$

j = output neuron
i = neuron in last hidden

  - Hidden Neurons:

$$\text{let: } \delta_j = f'(net_j)\,\Sigma(\delta_k w_{kj})$$
$$\partial E/\partial w_{ji} = -output_i\,\delta_j$$

j = hidden neuron
i = neuron in previous layer
k = neuron in next layer

Back Propagation

- Calculation of the derivatives flows backwards through the network, hence the name, back propagation
- These derivatives point in the direction of the maximum increase of the error function
- A small step (learning rate) in the opposite direction will result in the maximum decrease of the (local) error function:

$$w_{new} = w_{old} - \alpha \, \partial E / \partial w_{old}$$

where α is the learning rate

Back Propagation

- The learning rate is important
  - Too small
    - Convergence extremely slow
  - Too large
    - May not converge
- Momentum
  - Tends to aid convergence
  - Applies smoothed averaging to the change in weights:

$$\Delta_{new} = \beta\Delta_{old} - \alpha\, \partial E/\partial w_{old}$$

β is the momentum coefficient

$$w_{new} = w_{old} + \Delta_{new}$$

  - Acts as a low-pass filter by reducing rapid fluctuations

Back Propagation

- N is a neuron.
- N_w is one of N's inputs weights
- N_{out} is N's output.
- $N_w = N_w + \Delta N_w$
- $\Delta N_w = N_{out} * (1 - N_{out}) * N_{ErrorFactor}$
- $N_{ErrorFactor} = N_{ExpectedOutput} - N_{ActualOutput}$
- This works only for the last layer, as we can know the actual output, and the expected output.

Error Estimation

- The Root Mean Square Error (RMSE) is a frequently used measure of the differences between values predicted by a model or an estimator and the values actually observed from the thing being modelled or estimated

Weights Adjusting

- After each iteration, weights should be adjusted to minimize the error.
  - All possible weights
  - Back propagation

Number of Neurons

- Many neurons:
  - Higher accuracy
  - Slower
  - Risk of over-fitting
    - Memorizing, rather than understanding
    - The network will be useless with new problems.
- Few neurons:
  - Lower accuracy
  - Inability to learn at all
- Optimal neurons.

Data Representation

- Usually input/output data needs pre- processing
- Pictures
  - Pixel intensity
- Text:
  - A pattern

Size of Training Set

- No one fits-all formula
- Over fitting can occur if a “good” training set is not chosen
- What constitutes a “good” training set?
    - Samples must represent the general population.
    - Samples must contain members of each class.
    - Samples in each class must contain a wide range of variations or noise effect.
- The size of the training set is related to the number of hidden neurons

Agenda

- History of Artificial Neural Networks
- What Is an Artificial Neural Networks?
- How It works?
- Learning
  - Learning Paradigms
    - Supervised Learning
    - Unsupervised Learning
    - Reinforcement Learning
- Neural Network Training Algorithms
- Example
- Conclusion and References

Learning Paradigms

- **Learning:** is the adaptation of the network to better handle a task by considering sample observations. Learning involves adjusting the weights of the network to improve the accuracy of the result.

- **Learning Rate:** defines the size of the corrective steps that the model takes to adjust for errors in each observation.

- **Cost Function:** is a measure of "how good" a neural network did with respect to it's given training sample and the expected output. It also may depend on variables such as weights and biases.

Learning Paradigms

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

Supervised Learning

- This is what we have seen so far!
  - Only supplies inputs
  - The neural network adjusts its own weights so that similar inputs cause similar outputs
- A network is fed with a set of training samples (inputs and corresponding output), and it uses these samples to learn the general relationship between the inputs and the outputs.
- This relationship is represented by the values of the weights of the trained network.

Unsupervised Learning

- No desired output is associated with the training data!
  - Only supplies inputs
  - The neural network adjusts its own weights so that similar inputs cause similar outputs
    - The network identifies the patterns and differences in the inputs without any external assistance
- Faster than supervised learning
- Used to find out structures within data:
  - Clustering
  - Compression

Reinforcement Learning

- Like supervised learning, but:
  - Weights adjusting is not directly related to the error value.
  - The error value is used to randomly, mix weights!
  - Relatively slow learning due to ‘randomness’.

Reinforcement Learning (RL)

- RL use an agent that learn interactively with the environment through trial and error response from its experiences and own actions. RL can solve the decision problem by using a sequential decision problem through interactive to measure of the feedback performance.

Comparison of Learning Paradigms

- A brief comparison of unsupervised, supervised and reinforcement learning are:

	Unsupervised Learning	Supervised Learning	Reinforcement Learning
Model affection	Model does not affect the input data	Model does not affect the input data	Agent can affect it's own observations
Learning structure	Learning underlying data structure	Learning to approximate reference answers	Learning optimal strategy by trial and error
Feedback	No feedback required	Needs correct answers	Needs feedback on agent's own actions

Agenda

- History of Artificial Neural Networks
- What Is an Artificial Neural Networks?
- How It works?
- Learning
  - Learning Paradigms
    - Supervised Learning
    - Unsupervised Learning
    - Reinforcement Learning
- Neural Network Training Algorithms
- Example
- Conclusion and References

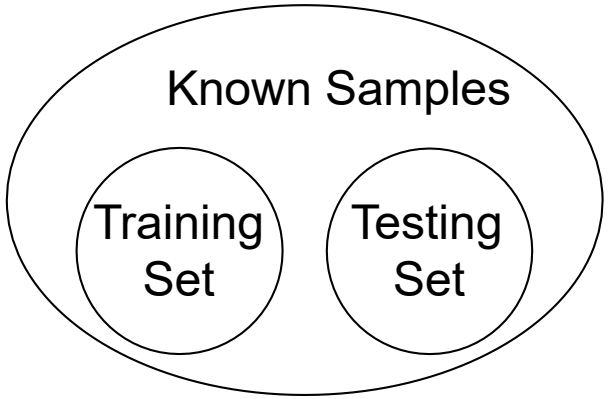
Set Chosen of Training

- Overfitting can also occur if a “good” training set is not chosen.

- What constitutes a “good” training set?
  - Samples must represent the general population
  - Samples must contain members of each class
  - Samples in each class must contain a wide range of variations or noise effect

Training and Verification

- Known samples is divided into two sets:
  - Training set
    - A group of samples used to train the neural network
  - Testing set
    - A group of samples used to test the performance of the neural network
    - Used to estimate the error rate



Verification

- Provides an unbiased test of the quality of the network
- Common error is to “test” the neural network using the same samples that were used to train the neural network
  - The network was optimized on these samples, and will obviously perform well on them
  - Doesn’t give any indication as to how well the network will be able to classify inputs that weren’t in the training set

Verification

- Various metrics can be used to grade the performance of the neural network based upon the results of the testing set
  - Mean Square Error (MSE), Signal-to-Noise Ratio (SNR), etc.
- Resampling is an alternative method of estimating error rate of the neural network
  - Basic idea is to iterate the training and testing procedures multiple times
  - Two main techniques are used:
    - Cross-Validation
    - Bootstrapping

Neural Network Training Algorithms

- Minimization of a loss (cost) function, *f*.

- Loss function is in general, composed of an error and a regularization (prevent overfitting) terms and fits the data set.

- The loss function depends on the adaptive parameters (biases and weights).

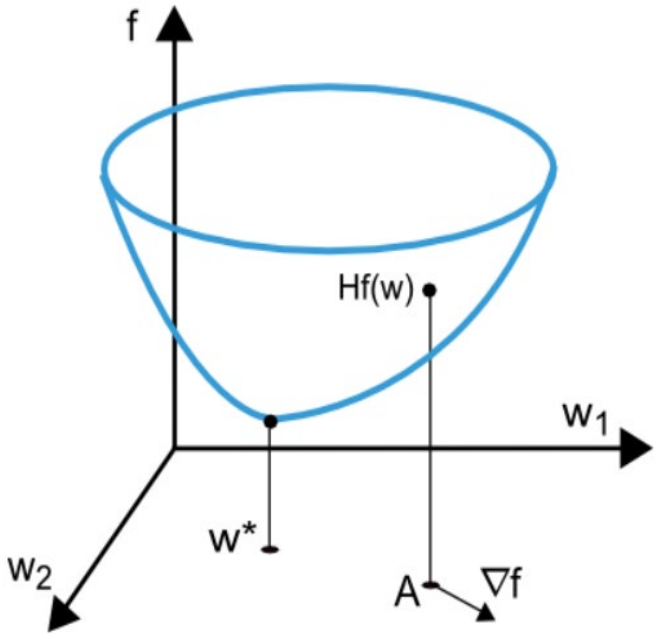
- Resampling is an alternative method of estimating error 1.

Neural Network Training Algorithms

- w^* is minima of the loss function.
- At point *A*, first and second derivatives:

$$\nabla_i f(w) = df/dw_i \ (i = 1,\ldots,n) \qquad (1)$$

$$H_{i,j} f(w) = d^2 f/dw_i \cdot dw_j \ (i,j = 1,\ldots,n) \qquad (2)$$

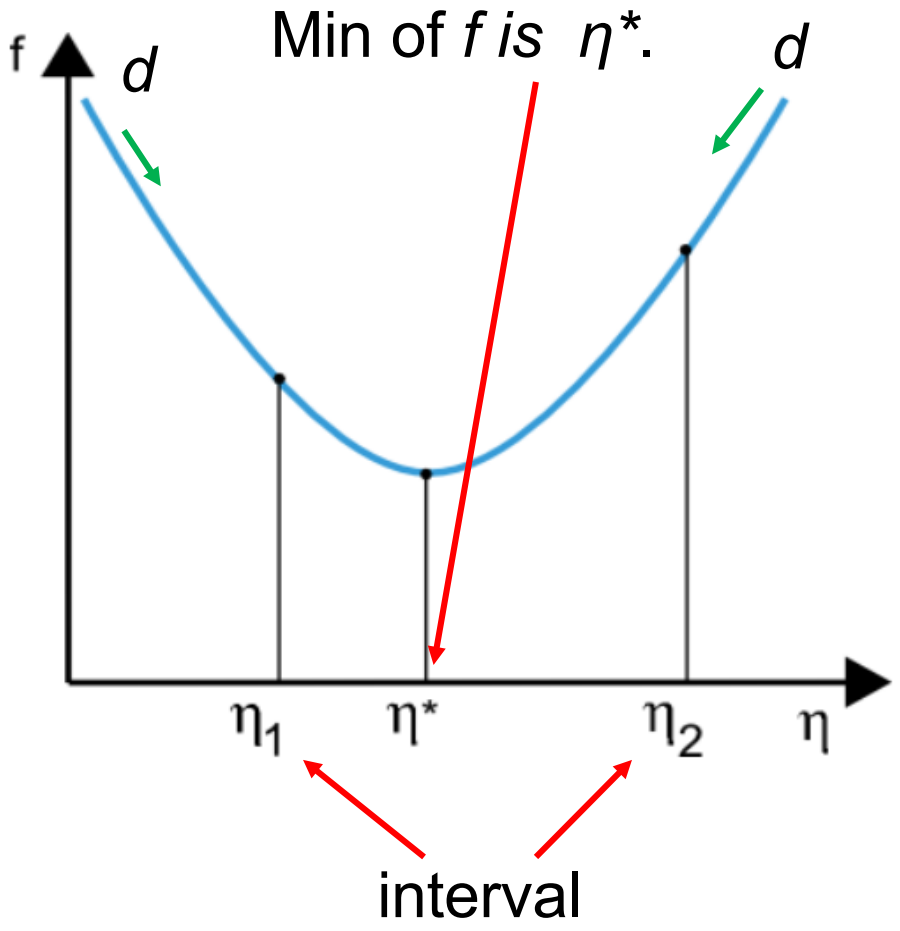


Problem: Min differentiable functions

Neural Network Training Algorithms

Certainly, many training algorithms first compute a training direction *d* and then a training rate η that minimizes the loss in that direction, $f(\eta)$.

- Usually $f(\eta)$ is a non-linear.
- To train a neural network first chosen at random parameters, then generate a sequence of parameters, so that the loss function is reduced at each iteration of the algorithm.

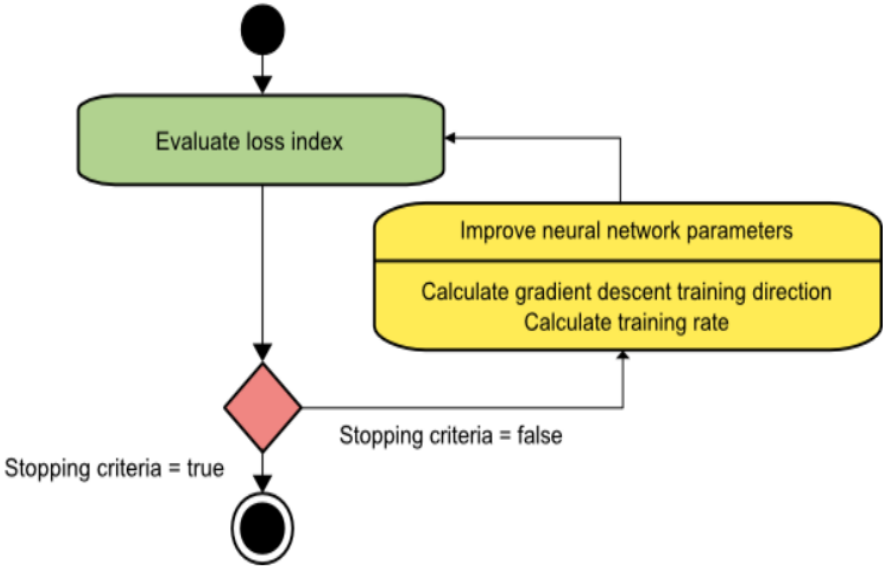


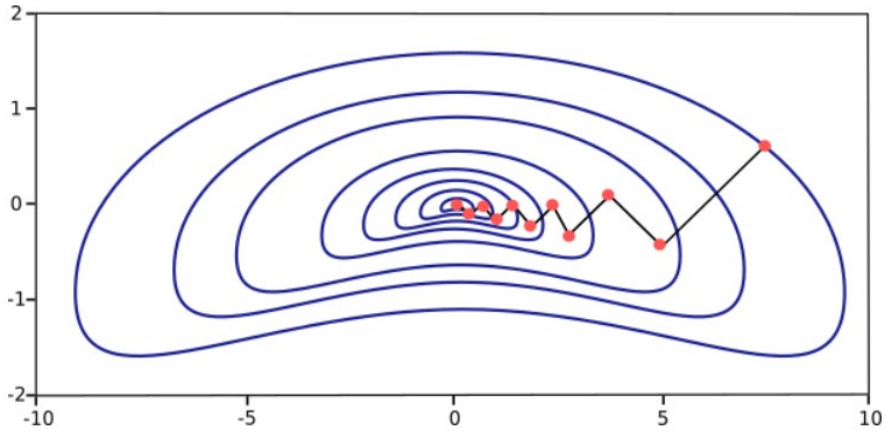
1. Gradient Descent (Steepest Descent)

- *Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function.*

- *Multi-variable function F(x) is defined and differentiable in a neighbourhood of a point then F(x) decreases fastest if one goes from a in the direction of the negative gradient of F(x) at a* $-\nabla F(a)$. *It follows that, if*

$$x^{+} = x + ts$$



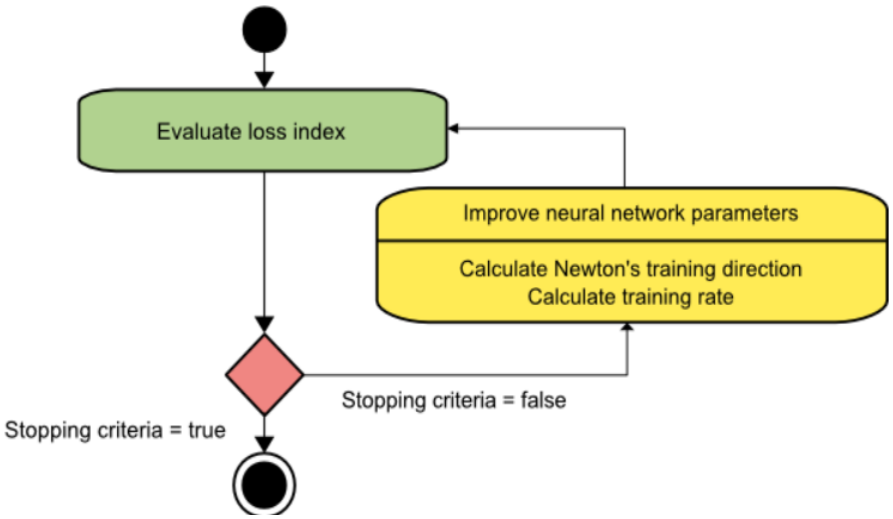


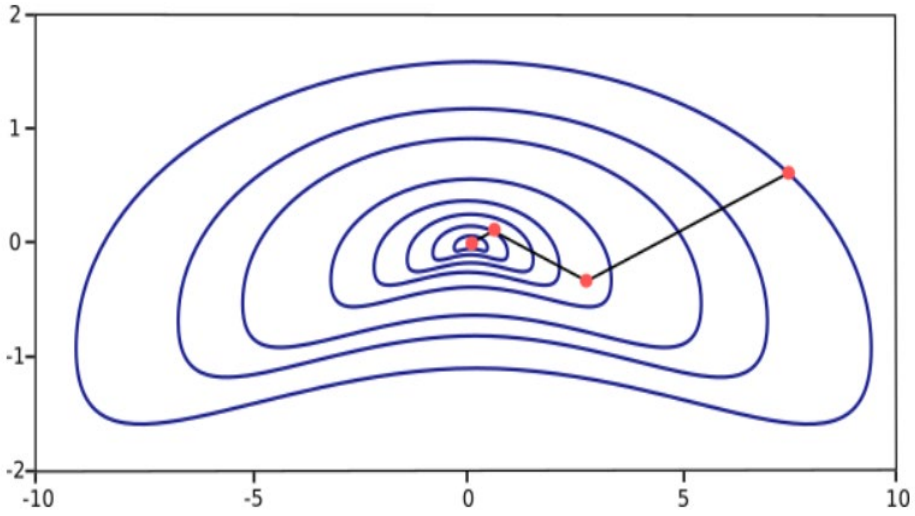
2. Newton's Method (Second Order)

- *Newton method is a root-finding algorithm which produces successively better approximations to the roots of a real-valued function.*

- The most basic version starts with a single-variable function f defined for a real variable x, the function's derivative f', and an initial guess x_0 for a root of f. If the function satisfies sufficient assumptions and the initial guess is close, then

$$x^+ = x - t(\nabla^2 f(x))^{-1} \nabla f(x))$$



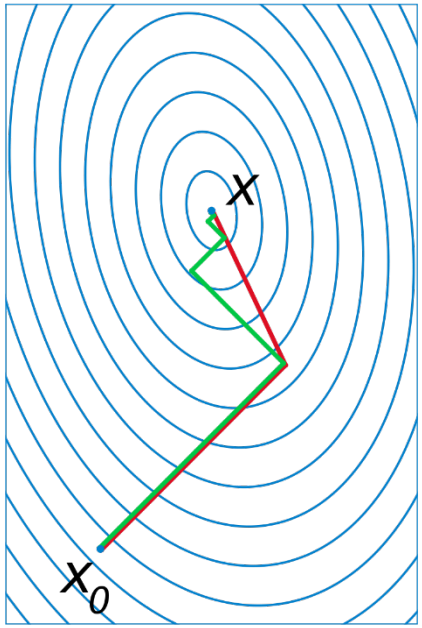


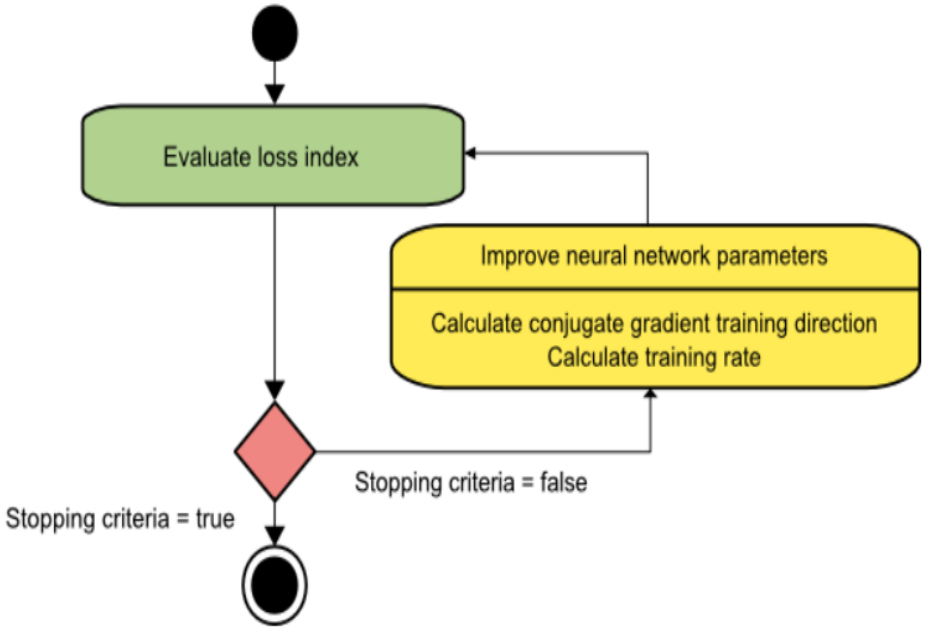
3. Conjugate Gradient

- *Conjugate direction methods can be regarded as being between the method of steepest descent (first-order method) and Newton's method (second-order method that uses Hessian as well).*

- *The conjugate gradient method aims to solve a system of linear equations, Ax=b, where A is symmetric, without calculation of the inverse of A.*

- *A comparison of * gradient descent (in green) and conjugate vector (in red) for minimizing a quadratic function.*





4. Quasi-Newton (Variable Metric)

Two main steps in Newton iteration:

- Compute Hessian $\nabla^2 f(x)$
- Solve the system $\nabla^2 f(x)\, s = -\nabla f(x)$

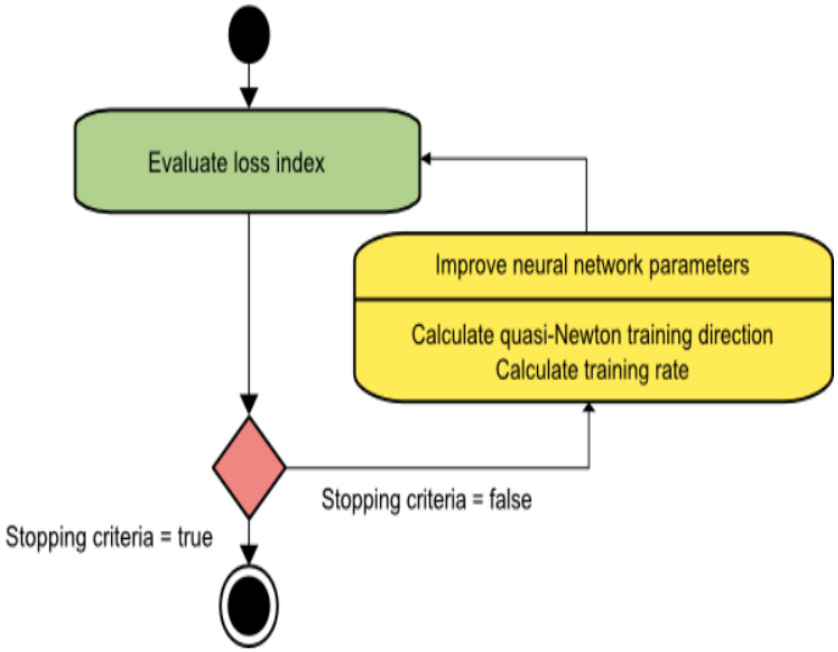
Each of these two steps could be expensive
Quasi-Newton methods repeat updates
of the form

$$x^+ = x + ts$$

where direction s is defined by linear

$$Bs = -\nabla f(x)$$

for some approximation B of $\nabla^2 f(x)$.
We want B to be easy to compute,
and $Bs = g$ to be easy to solve



5. Levenberg-Marquardt (Damped Least-Squares)

Levenberg-Marquardt steps are linear combination of Gradient descent and Gauss-Newton steps based on adaptive rules.

- Start with an initial guess x_0. x is adjusted by d only for downhill steps.

$$(J^T J + \lambda I)\, \delta = J^T r$$

J = jacobian matrix of derivatives

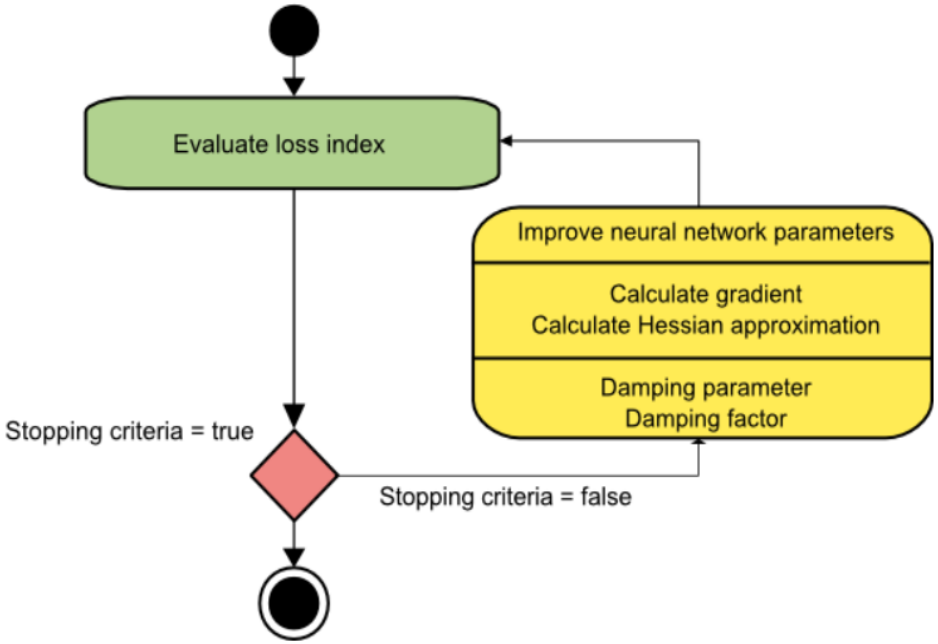
of the residuals with respect

to the parameters

λ = damping parameter

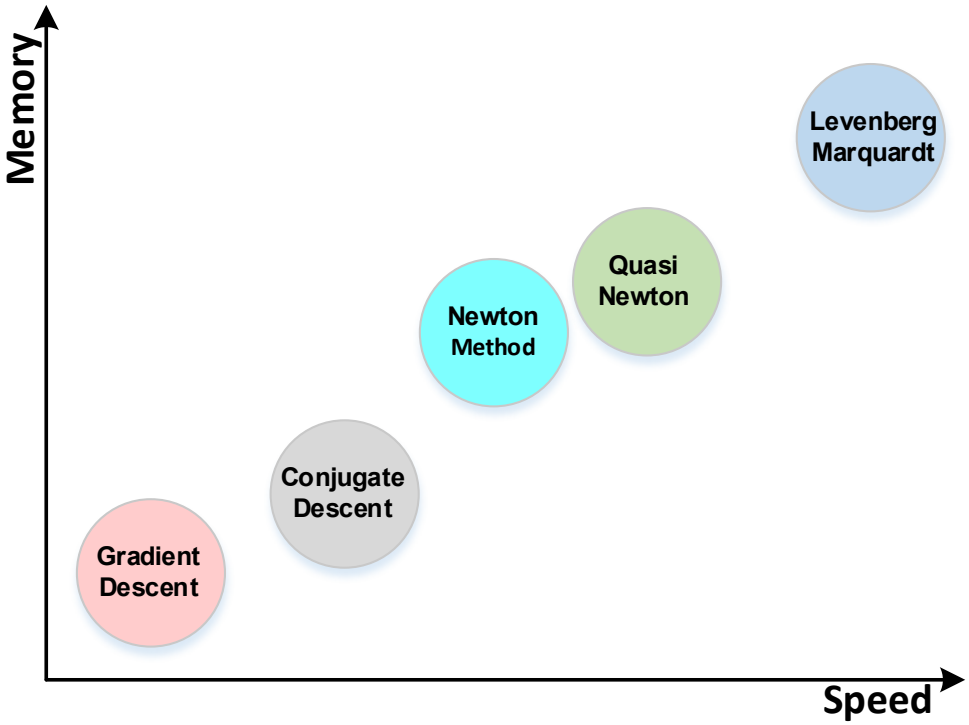
(adaptive balance between the 2 steps)

r = residual vector



Conclusions of Neural Network Training

- ✓ For many thousands of parameters, use Gradient Descent or Conjugate Gradient, in order to save memory.
- ✓ For a few hundreds of parameters, the best choice is Levenberg-Marquardt algorithm, for high speed.
- ✓ In the rest of situations, the Quasi-Newton method will work well.

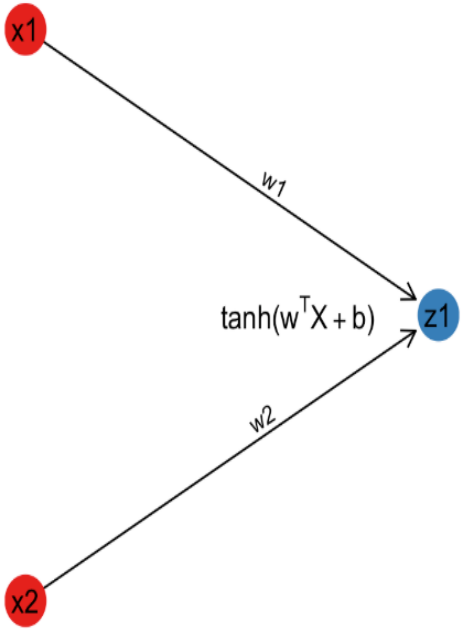


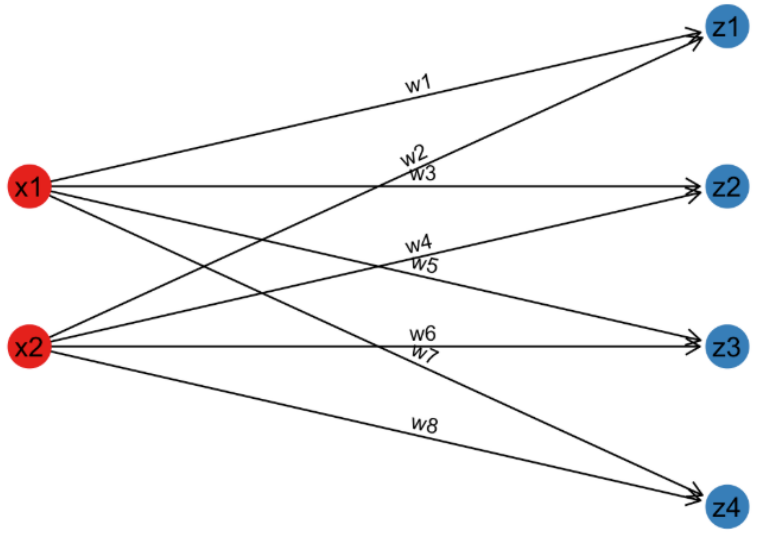
Agenda

- History of Artificial Neural Networks
- What is an Artificial Neural Networks?
- How it works?
- Learning
  - Learning paradigms
    - Supervised learning
    - Unsupervised learning
    - Reinforcement learning
- Neural Network Training Algorithms
- Example
- Conclusion and References

1. Forward Propagation

- Calculating an output based on the weights of each edge. The connections from the input layer to each of the hidden nodes is a linear function, followed by an activation function.
- The computational steps of forward propagations are Input−>Hidden−>Output .





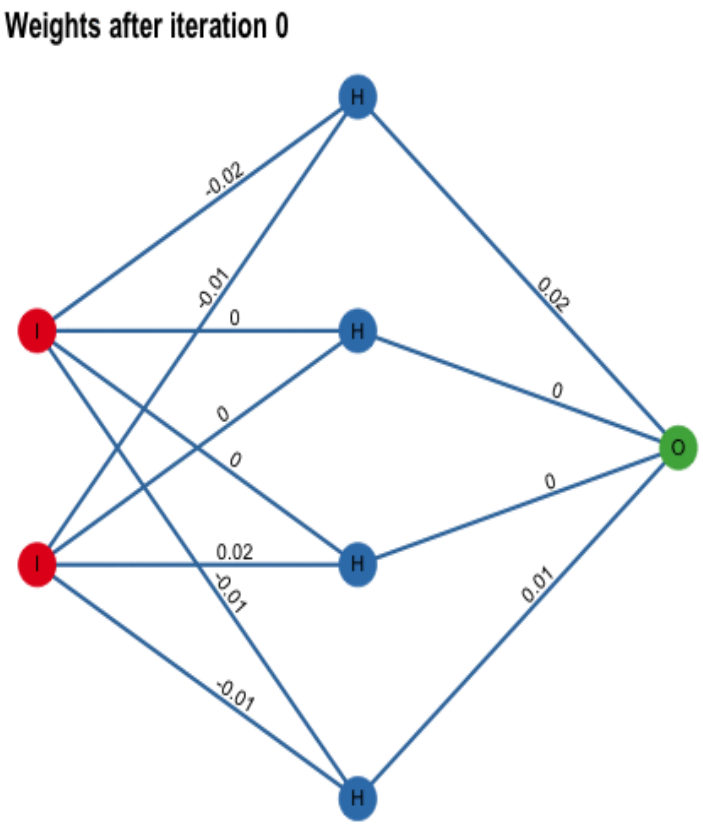
2. Computing the Cost

- As we have seen above, forward propagation is nothing more than a predict function. When the dataset has been passed through the network, we get an output that can be compared to the actual label. The purpose of the cost (loss) function is to inform the model how far the output is from the target value. One of the most popular cost functions is log loss, formally known as:

$$J = -\frac{1}{m}\sum_{i=0}^{m}\left(Y\log\left(A^{[2]}\right) + (1-Y)\log\left(1-A^{[2]}\right)\right)$$

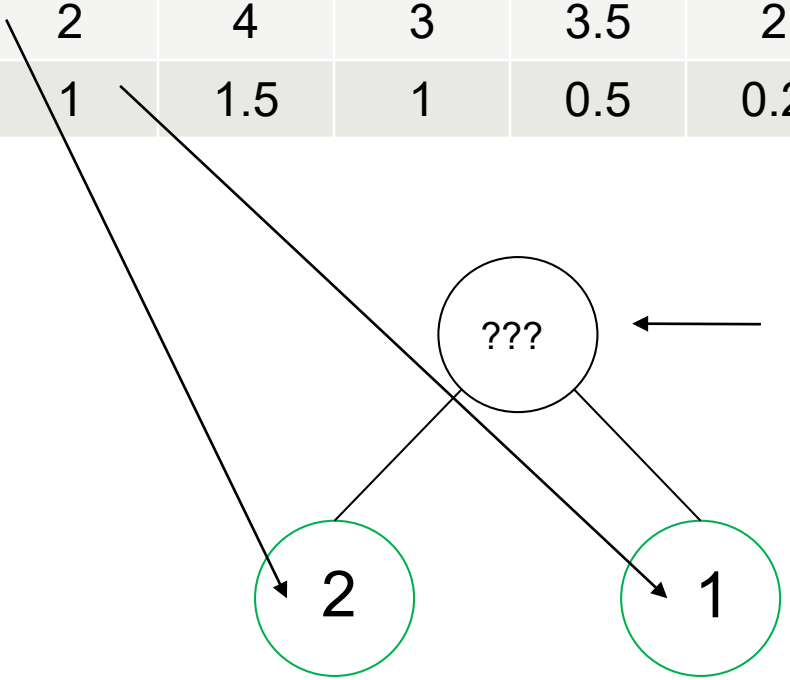
3. Backward Propagation

- By means of gradient descent, we calculate the partial derivatives of all computations with respect to what came after, alas we go backwards. First the derivatives of the weights of the hidden layer with respect to the output layer, and secondly those of the input layer with respect to the hidden layer. The gradients we obtain are then used to update the weights and start the process all over again. With each pass - also called epochs, we get closer to the optimal weights.



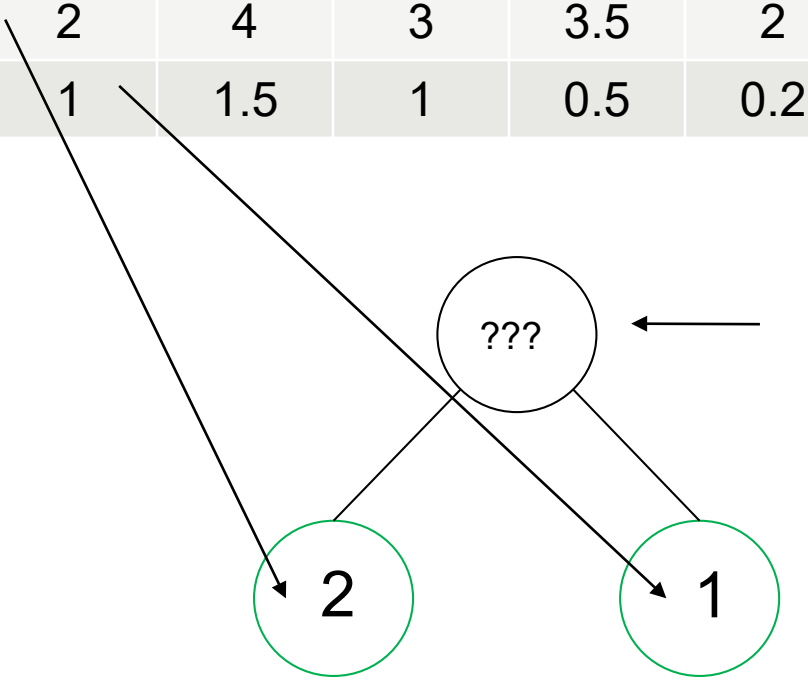
Example (2)- Colour Prediction

Binary Colour	1	0	1	0	1	0	1	0	???
Colour									???
Length	3	2	4	3	3.5	2	5.5	1	4.5
Width	1.5	1	1.5	1	0.5	0.2	1	1	1



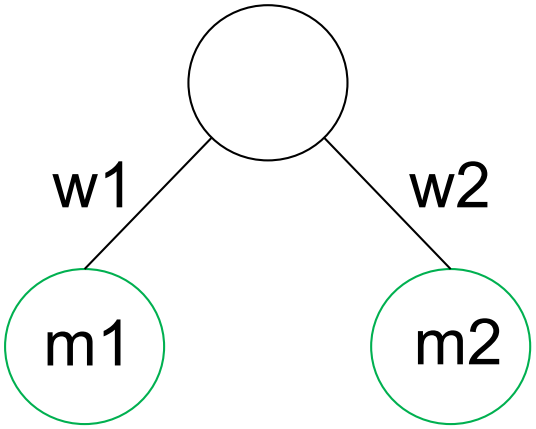
Example (2)- Colour Prediction

Binary Colour	1	0	1	0	1	0	1	0	???
Colour									???
Length	3	2	4	3	3.5	2	5.5	1	4.5
Width	1.5	1	1.5	1	0.5	0.2	1	1	1



Colour Prediction

(m1. w1 + m2.w2 + b) =???



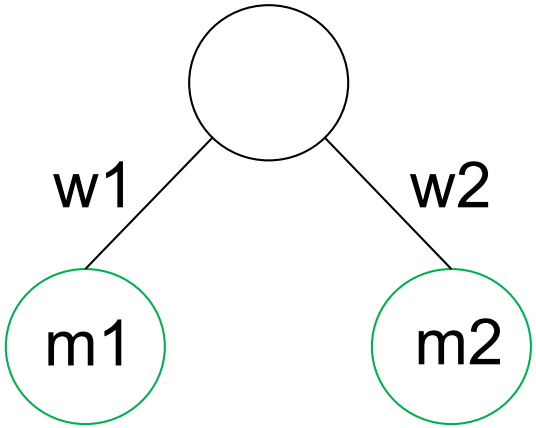
it should results between 0 and 1

The good function is sigmoid

m1 and m2 are variable names
w1 and w2 are weight
b is bias

Sigmoid Function

sigmoid(m1. w1 + m2.w2 + b)



m1 and m2 are variable names
w1 and w2 are weight

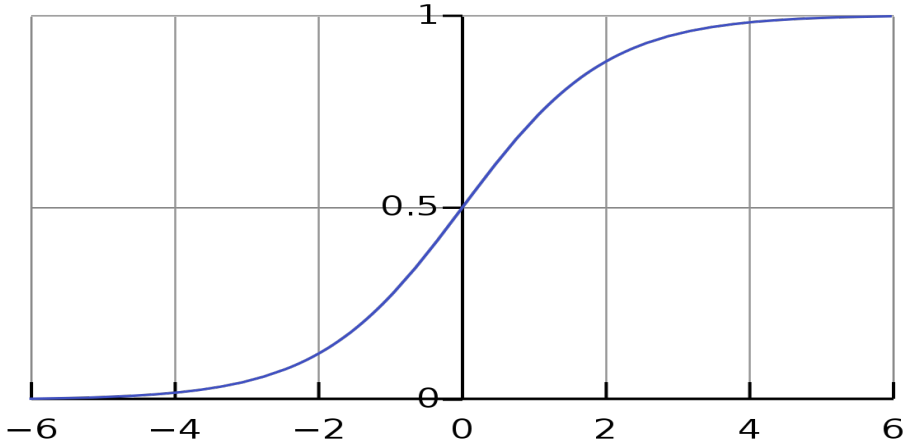
Sigmoid Function

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

$$sigmoid(-5) = \frac{1}{1+e^{-(-5)}} = 0.0067$$

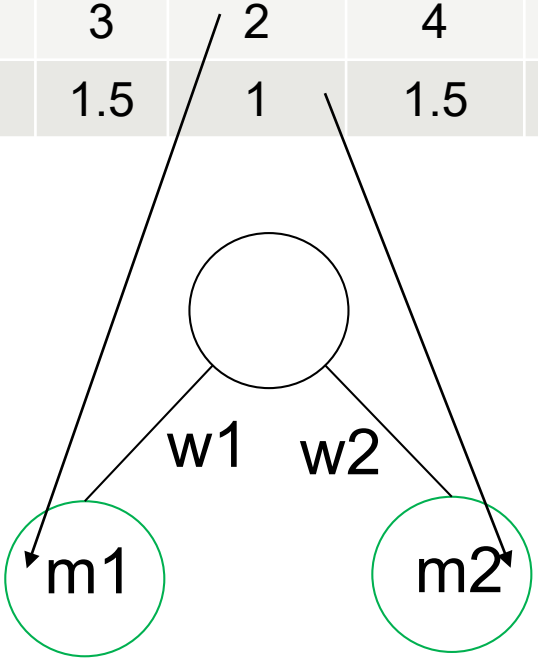
$$sigmoid(0) = \frac{1}{1+e^{-0}} = 0.5$$

$$sigmoid(5) = \frac{1}{1 + e^{-5}} = 0.9933$$



Sigmoid Function

Binary Colour	1	0	1	0	1	0	1	0	???
Colour									???
Length	3	2	4	3	3.5	2	5.5	1	4.5
Width	1.5	1	1.5	1	0.5	0.2	1	1	1

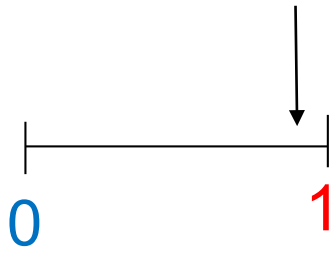


NN (m1,m2)= sigmoid (m1. w1 + m2.w2 + b)

NN (2,1)= sigmoid (2. w1 + 1.w2 + b)

? ? ?

NN (2,1)= sigmoid (1.5)= 0.82

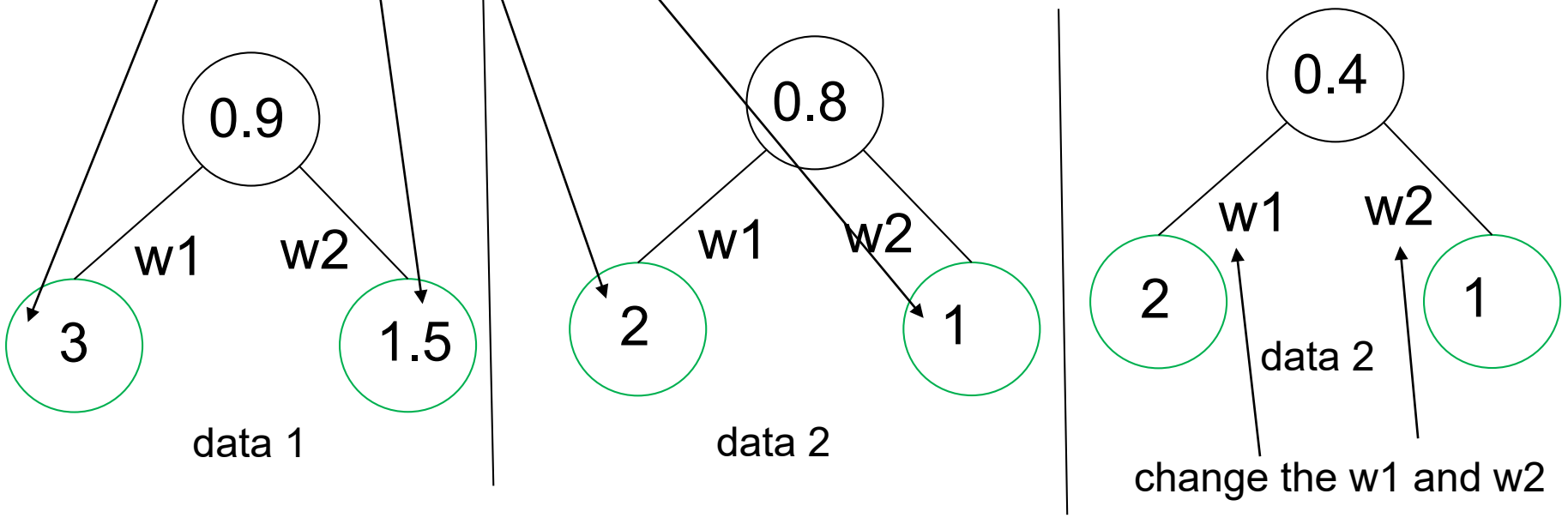


It is not correct with reality

Learn with new data (backpropagation)

Binary Colour	1	0	1	0	1	0	1	0	
Colour	○	○	○	○	○	○	○	○	???
Length	3	2	4	3	3.5	2	5.5	1	4.5
Width	1.5	1	1.5	1	0.5	0.2	1	1	1

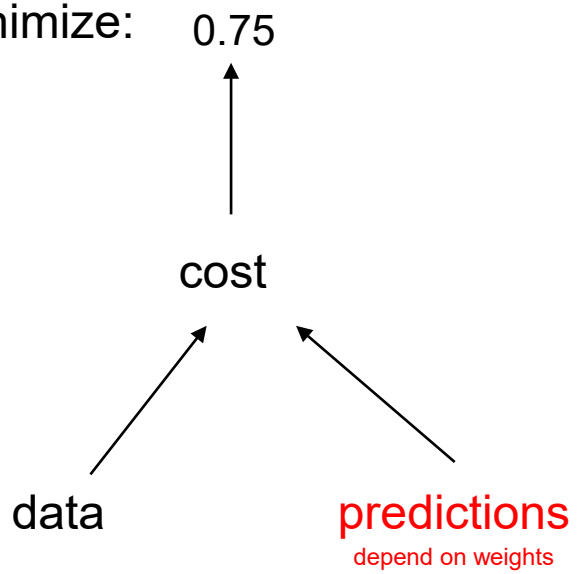
result is <0.5 and is blue



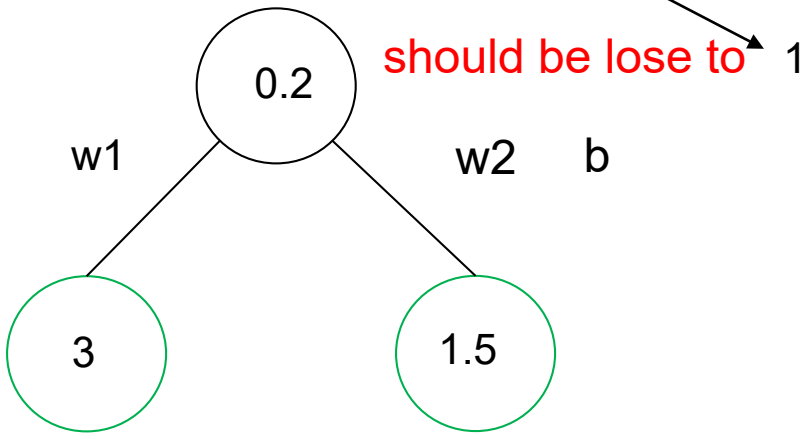
Cost Function

Aim

minimize:



Binary Colour	1	0	1
Length	3	2	4
Width	1.5	1	1.5



Cost Function gets 0.2 (prediction data) and 1 (target data):

squared error = (prediction-target) ^2 = (0.2 - 1) ^2 =0.64

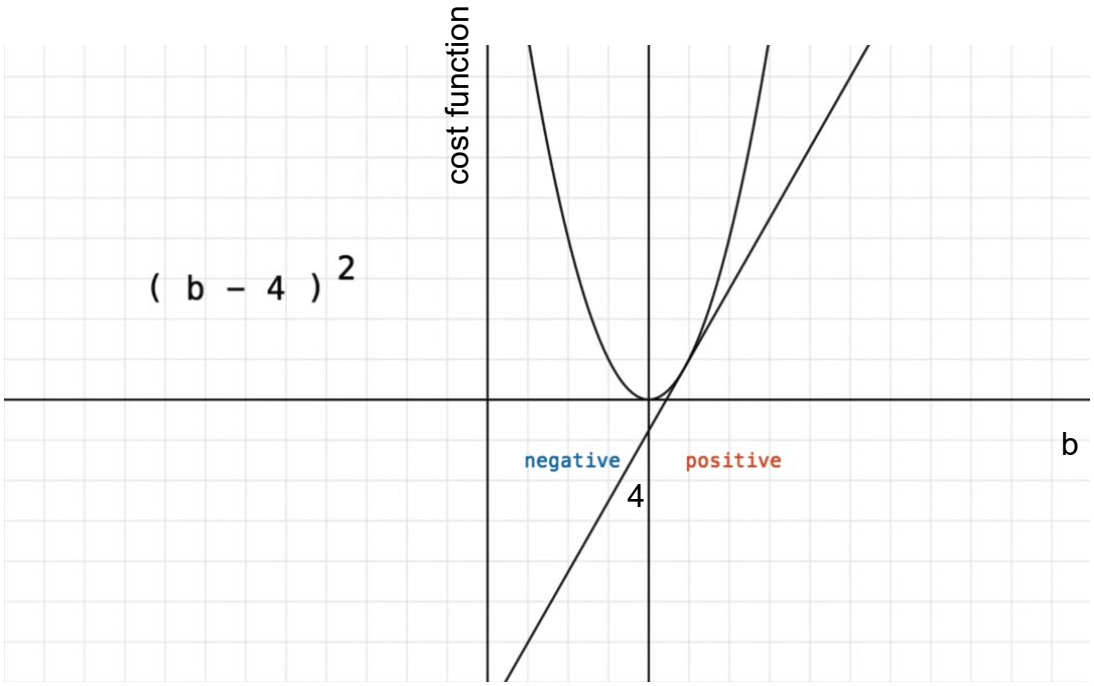
Note: we can not change the data (real world) but we can change the predictions

Cost Function

(prediction-target) ^2

NN() = b

(b – 4) ^2



Slope of the cost function at b ?

b=(b - 0.1 . slope(b))

1- Numerically Algebraically 2- Calculus- L y

$$slope(b) = \frac{rise}{run} \qquad cost(b) =$$

Cost Function

cost (b)= (b – 4) ^2

$$slope(b) \sim \frac{cost(b+h) - cost(b)}{h}$$

$$slope(b) \sim \frac{(b+h+4)^2 - (b^2 - 2.4b + 4^2)}{h}$$

$$slope(b) \sim 2(\mathrm{b} - 4) + h$$

looks derivative

$$slope(b) = 2\,.\,(\mathrm{b} - 4)$$

$$\mathrm{f(x)} = x^2$$

$$\frac{d}{dx}\mathrm{f(x)} = 2\,x$$

$$\frac{d}{dx}\mathrm{cost(b)} = 2(b - 4)$$

Example (2)- Colour Prediction

- Matlab Code

Inputs	Outputs
3,1.5	1
2,1	0
4,1.5	1
3,1	0
3.5,0.5	1
2 ,0.2	0
5.5,1	1
1,1	0

New Inputs	Prediction output
4.5, 1	?

Agenda

- History of Artificial Neural Networks
- What is an Artificial Neural Networks?
- How It Works?
- Learning
  - Learning Paradigms
    - Supervised Learning
    - Unsupervised Learning
    - Reinforcement Learning
- Neural Network Training Algorithms
- Example
- Conclusion and References

Conclusion

- Artificial Neural Networks are an imitation of the biological neural networks, but much simpler ones.
- Their ability to learn by example makes them very flexible and powerful furthermore there is need to device an algorithm in order to perform a specific task.
- Many factors affect the performance of ANNs, such as the transfer functions, size of training sample, network topology, weights adjusting algorithm, …

References

[1] H. Khayyam et al. "Limited data modelling approaches for engineering applications" 2018 Nonlinear Approaches in Engineering Applications Pp 345-379 , Springer.

[2] H. Khayyam et al. "Big data modelling approaches for engineering applications" 2019 Nonlinear Approaches in Engineering Applications Pp 345-379 , Springer.

[3] H. Abdi, et.al , *Neural Networks*, Thousand Oaks, CA: SAGE Inc., 1999.

[4] S. Haykin, *Neural Networks*, New York, NY: Nacmillan College Publishing Company, Inc., 1994. [Mast93] T. Masters, *Practial Neural Network Recipes in C++*, Toronto, ON: Academic Press, Inc., 1993.

[5] R. Schalkoff, *Artificial Neural Networks*, Toronto, ON: the McGraw-Hill Companies, Inc., 1997.

[WeKu91] S. M. Weiss and A. Kulikowski, Computer Systems That Learn, San Mateo, CA: Morgan Kaufmann Publishers, Inc., 1991.

[6] P. D. Wasserman, Neural Computing: Theory and Practice, New York, NY: Van Nostrand Reinhold, 1989.

[7] V. Cheung and K. Cannons "An Introduction to Neural Networks" University of Manitoba Winnipeg, Manitoba, Canada, (2002)

[8] Craig Heller, and David Sadava, Life: The Science of Biology, fifth edition, Sinauer Associates, INC, USA, 1998.

[9] Introduction to Artificial Neural Networks, Nicolas Galoppo von Borries

[10] Tom M. Mitchell, Machine Learning, WCB McGraw-Hill, Boston, 1997.

[11] http://tamaszilagyi.com/blog/2017/2017-11-11-animated_net/