Data Mining

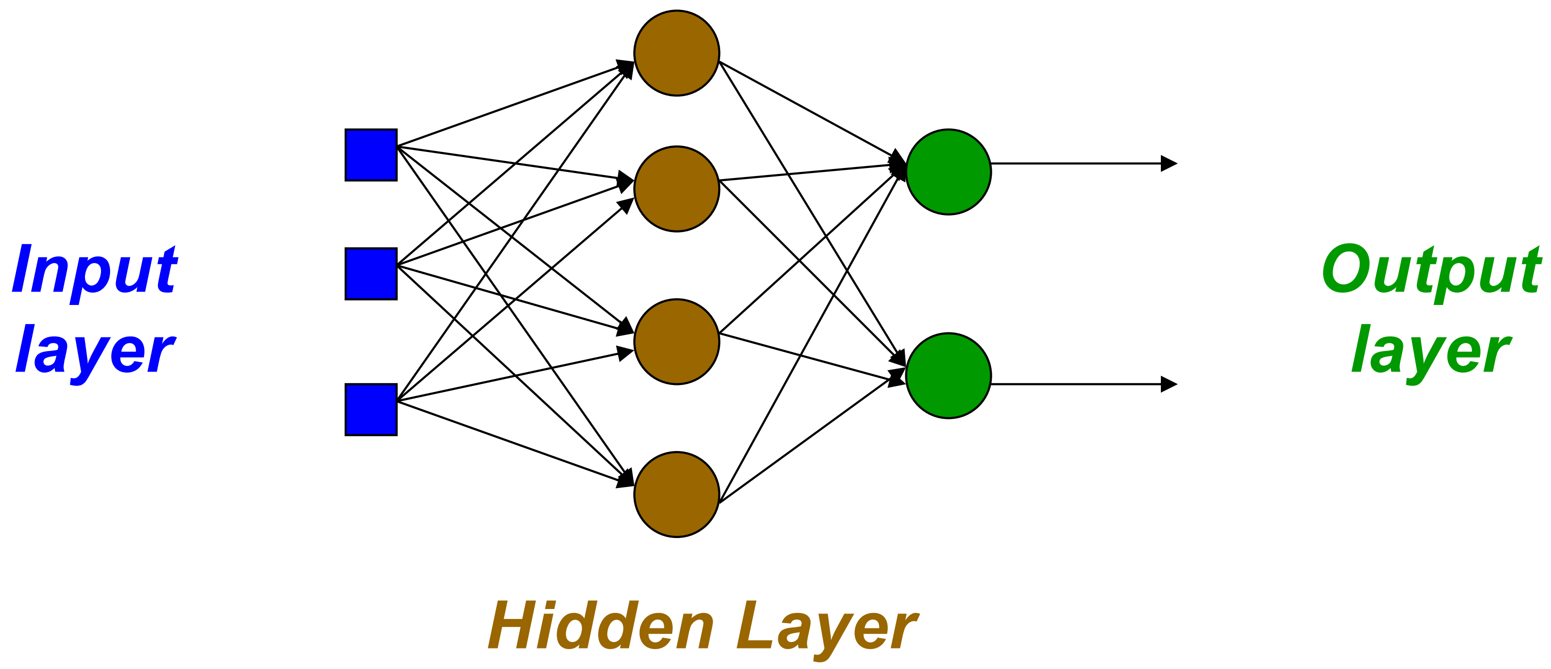
Feed Forward Neural Networks

Outline

- Multi-layer Neural Networks
- Feedforward Neural Networks
  - FF NN model
  - Backpropogation (BP) Algorithm
  - Practical Issues of FFNN

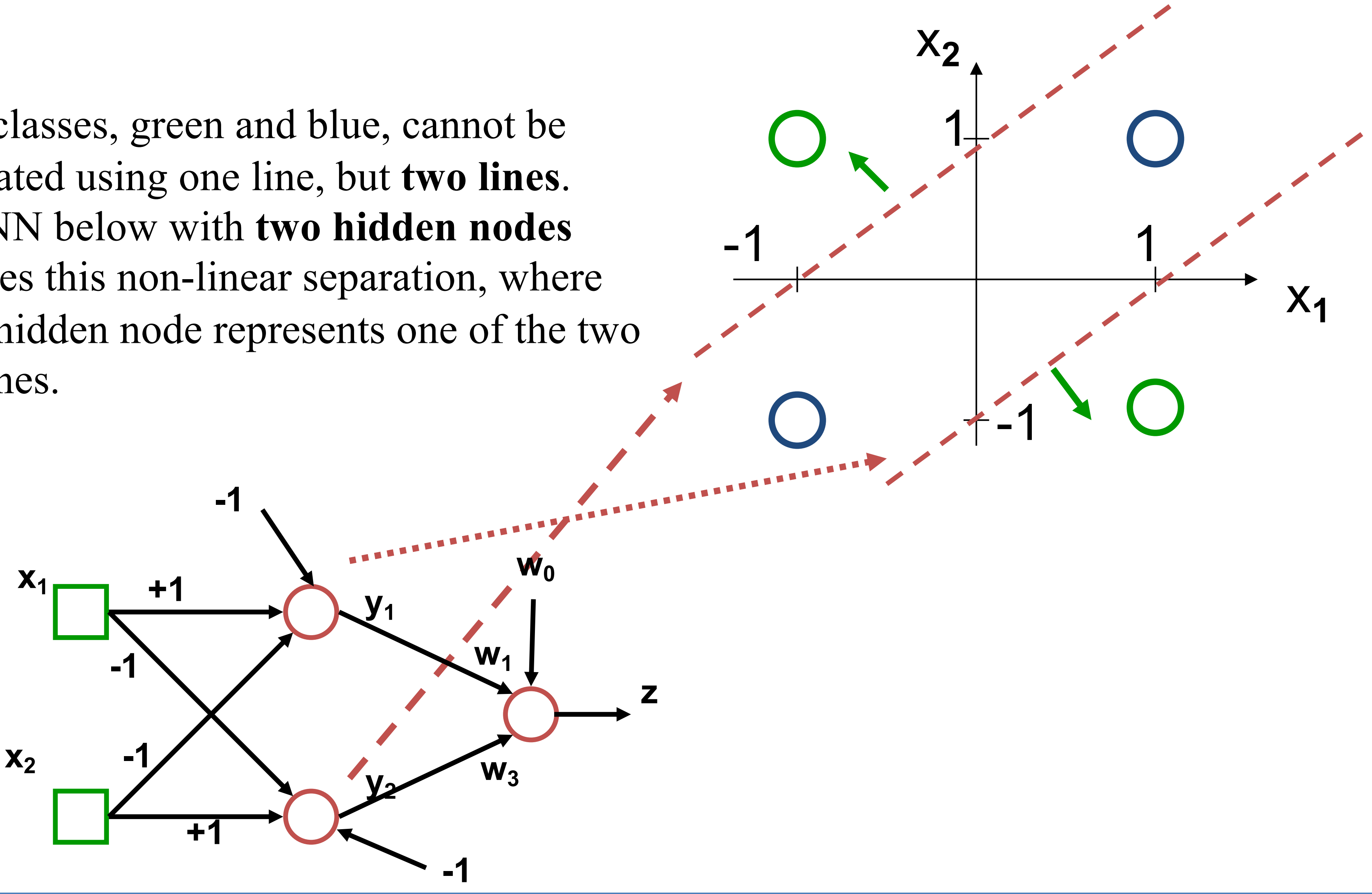
Multi-layer NN

- Between the input and output layers there are hidden layers, as illustrated below.
  - Hidden nodes do not directly send outputs to the external environment.
- Multi-layer NN overcome the limitation of a single-layer NN
  - they can handle non-linearly separable learning tasks.

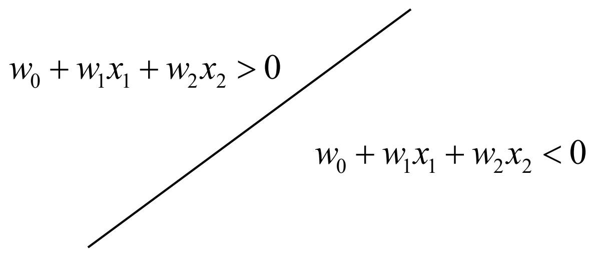


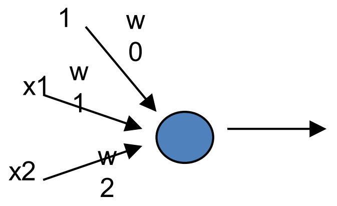
XOR problem

Two classes, green and blue, cannot be separated using one line, but **two lines**. The NN below with **two hidden nodes** realizes this non-linear separation, where each hidden node represents one of the two red lines.

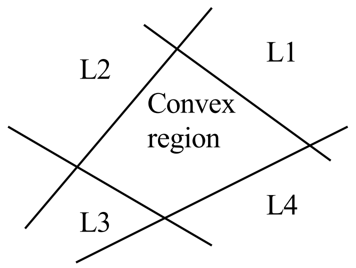


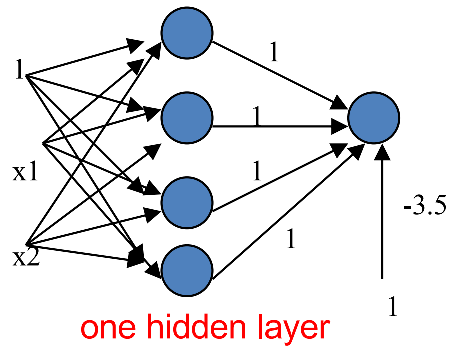
Types of decision regions



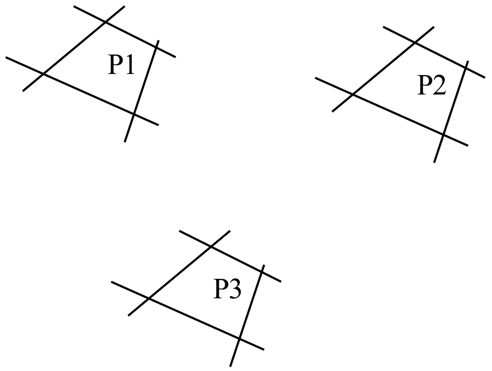


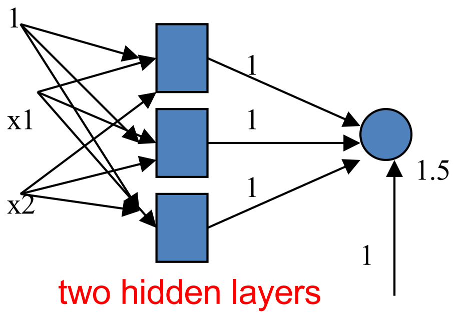
Network with a single node





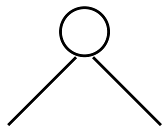
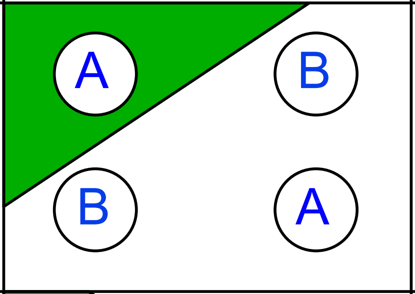
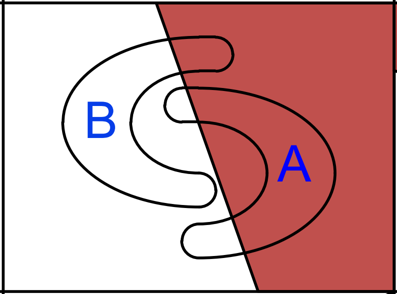
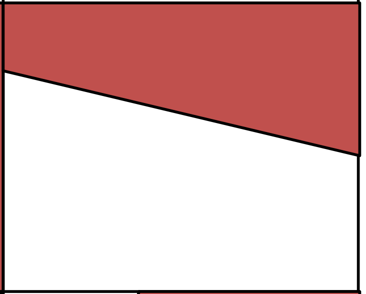
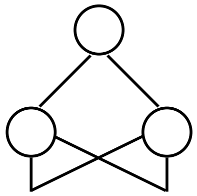
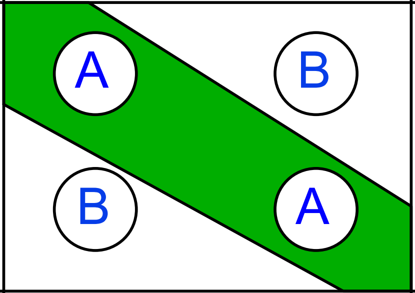
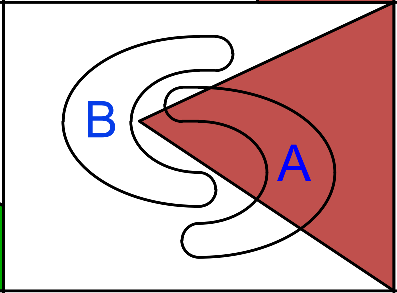
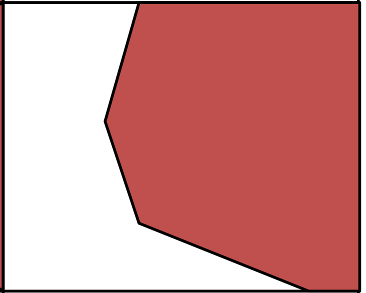
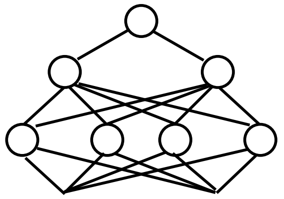
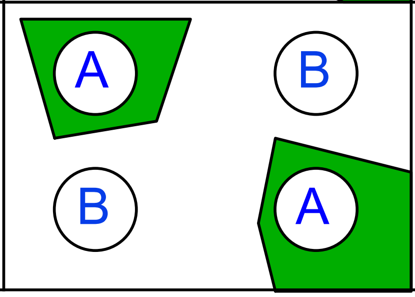
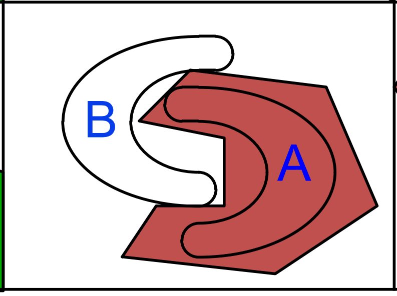
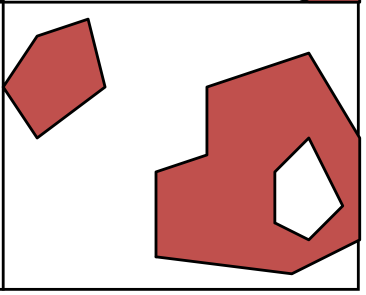
One-hidden layer network that realizes the convex region: each hidden node realizes one of the lines bounding the convex region





two-hidden layer network that realizes the union of three convex regions: each box represents a one hidden layer network realizing one convex region

Types of decision regions

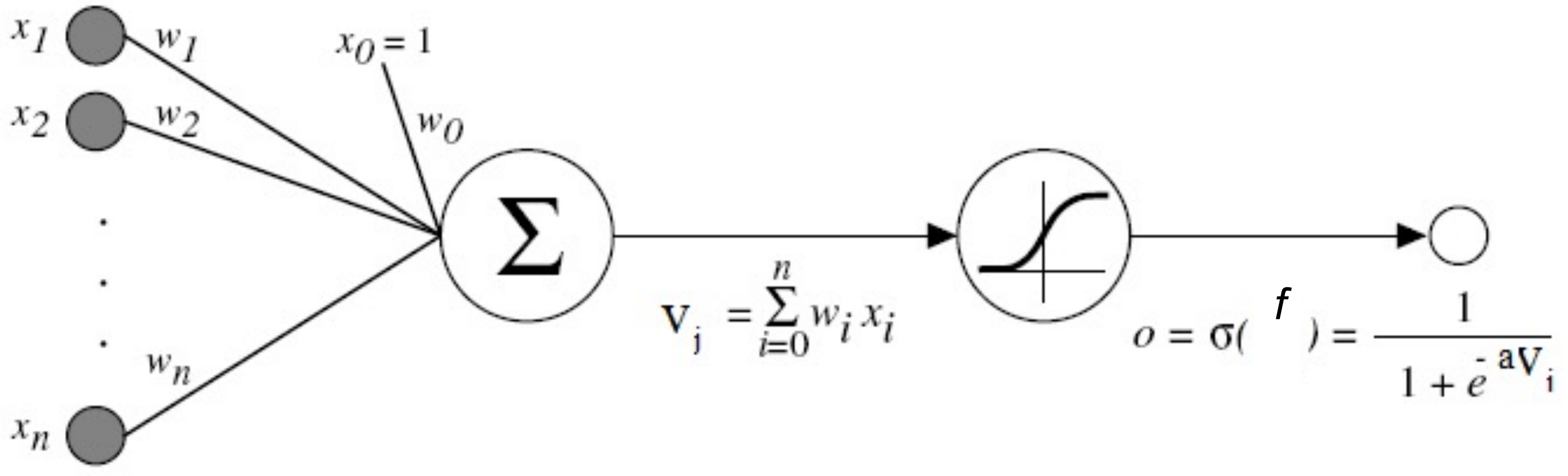
Structure	*Types of Decision Regions*	*Exclusive-OR Problem*	*Class Separation*	*Most General Region Shapes*
Single-Layer 	*Half Plane Bounded By Hyperplane*			
Two-Layer 	*Convex Open Or Closed Regions*			
Three-Layer 	Arbitrary (Complexity Limited by No. of Nodes)			

Outline

- Multi-layer Neural Networks
- Feedforward Neural Networks
  - FF NN model
  - Backpropogation (BP) Algorithm
  - BP rules derivation
  - Practical Issues of FFNN

FFNN NEURON MODEL

- The classical learning algorithm of FFNN is based on the gradient descent method.
- The activation function used in FFNN are continuous functions of the weights, differentiable everywhere.
  - A typical activation function is the Sigmoid Function
  - Nowadays, Rectified Linear activation function (ReLU)



x_1
x_2
x_n
w_1
w_2
w_n
$x_0 = 1$
w_0
Σ
$V_j = \sum_{i=0}^{n} w_i x_i$
$o = \sigma(f) = \frac{1}{1 + e^{-aV_i}}$

Activation Function

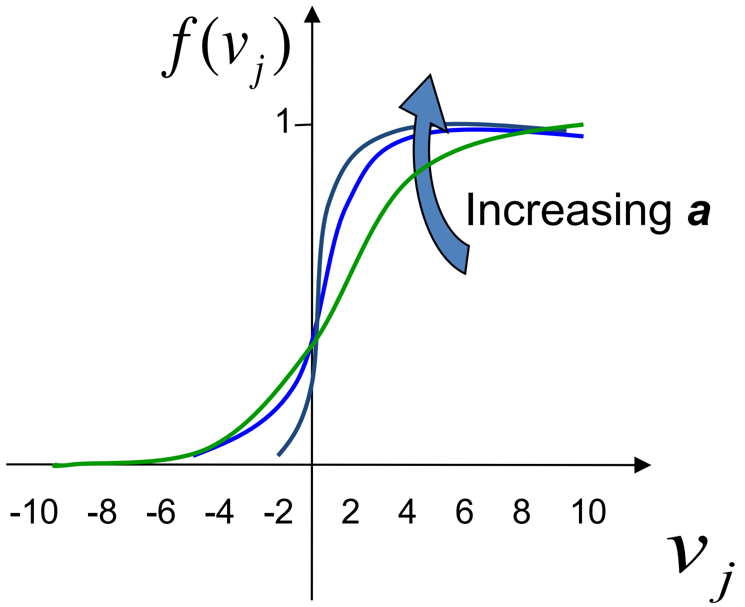
- A typical activation function is the Logistic Sigmoid Function:

$$f(\mathrm{v}_\mathrm{j}) = \frac{1}{1+e^{-av_j}} \quad \text{with } a > 0$$

where $\mathrm{v}_\mathrm{j} = \sum_i w_{\mathrm{ji}} y_{\mathrm{i}}$

with w_{ji} weight of link from node i

to node j and y_{i} output of node i



  - when **a** approaches to 0, f tends to a linear function
  - when **a** tends to infinity then f tends to the step function

Binary Sigmoid Activation Function

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

$$f'(x) = f(x)[1 - f(x)]$$ backward

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

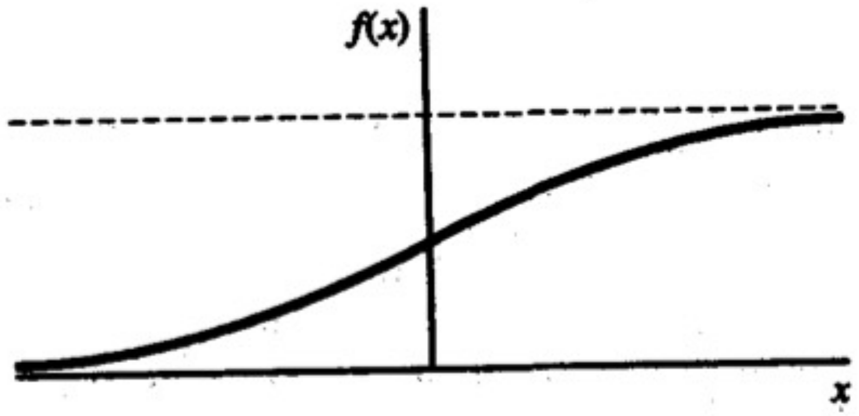
$$= \left(\frac{1}{1+e^{-x}}\right)^2 \frac{d}{dx}(1+e^{-x})$$

$$= \left(\frac{1}{1+e^{-x}}\right)^2 e^{-x}(-1)$$

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

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

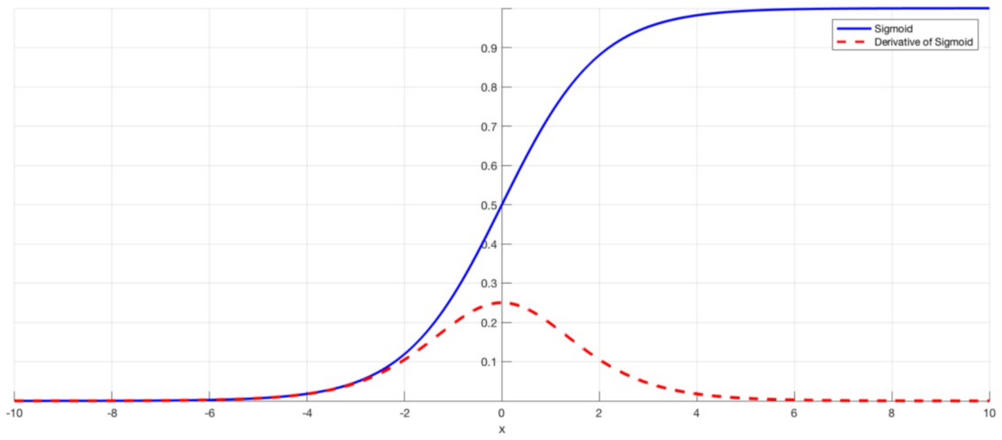
$$= s(x)(1 - s(x))$$



Binary Sigmoid Activation Function

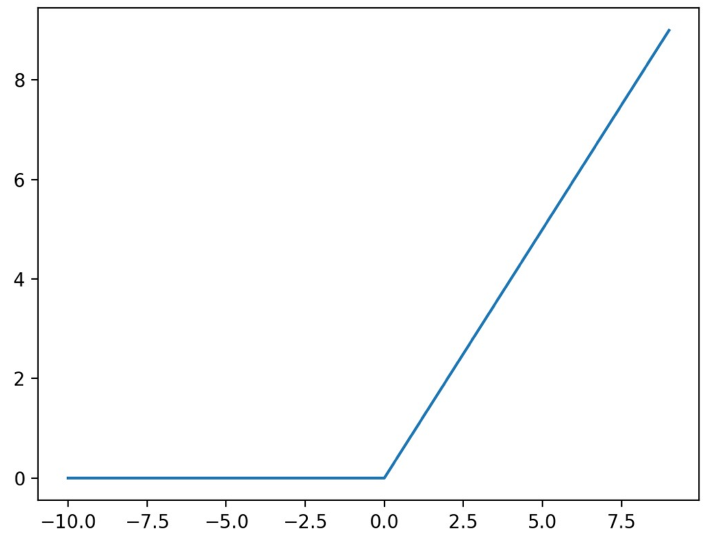
- Problems with sigmoid (and tanh) function:
  - Limited sensitivity
  - Saturation of values
  - Vanishing gradient problem

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$



Rectified Linear Activation Function

- ReLU
  - f(x) = max{0, x}
  - Derivative:
    - 0 for x <= 0; 1 for x>0
- Non-linear function
- Linear behavior : easier to optimize for NN
- Computational simplicity
- Works well in training deep multi-layered NN with backpropagation

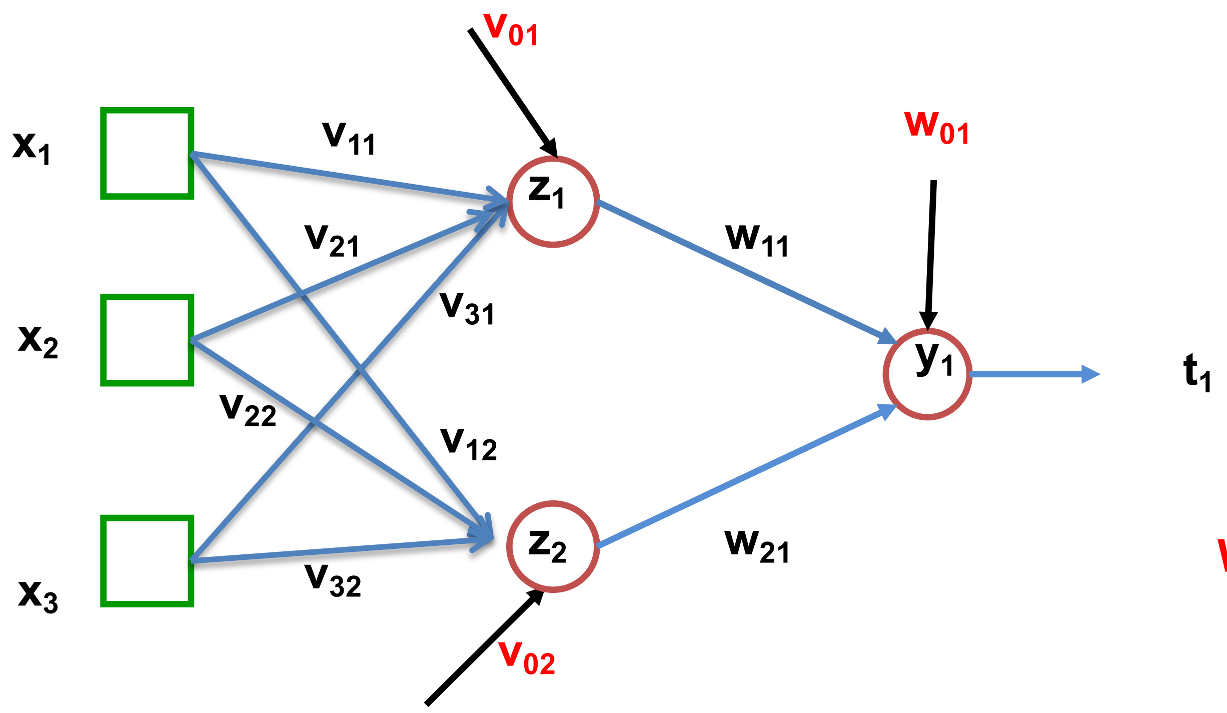


piece-wise linear

FFNN MODEL

- v_{ij} : The weight from input node *i* to hidden layer node *j*
  - Δv_{ij} : The weight updating amount from node *i* to node j
- w_{ij} : The weight from hidden layer node *i* to output node *j*
  - Δw_{ij} : The weight updating amount from node *i* to node *j*
- t_k : The output from output node *k*

V_{0j} is the bias on hidden unit j



W_{0j} is the bias on output unit j

Assess multi-layer NN

- The **error of output neuron *j*** after the activation of the network on the *n-th* training example $(x(n), d(n))$ is:

$$e_j(n) = d_j(n) - o_j(n)$$

$o_j(n)$: estimated output

- The **network error** is the sum of the squared errors of the output neurons:

$$E(n) = \frac{1}{2} \sum_{j \text{ output node}} e_j^2(n)$$

- *The **total mean squared error** is the average of the network errors over the training examples.*

Minimize

$$E(W) = \frac{1}{N} \sum_{n=1}^{N} E(n)$$

$$E(W) = \frac{1}{2N} \sum_n \sum_j (d_j(n) - o_j(n))^2$$

Feed Forward NN

Idea: Credit assignment problem

- Problem of assigning ‘credit’ or ‘blame’ to individual elements involved in forming the overall response of a learning system (hidden units)

- In neural networks, the problem relates to distributing the network error to the weights.

Outline

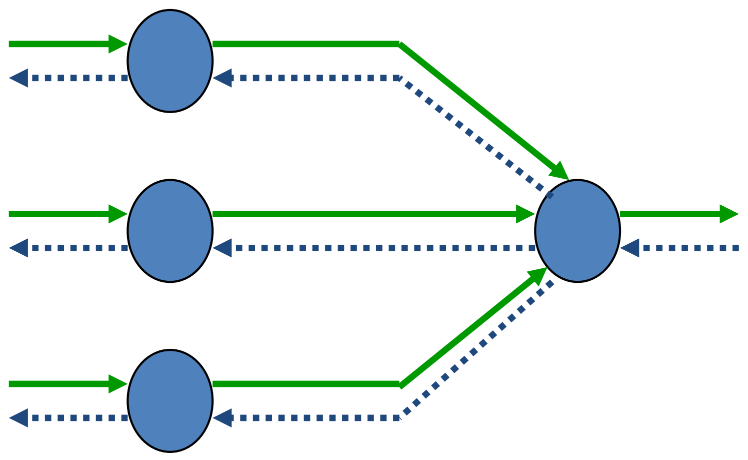
- Multi-layer Neural Networks
- Feedforward Neural Networks
  - FF NN model
  - Backpropogation (BP) Algorithm
  - Practical Issues of FFNN

Training: Backprop algorithm

- Searches for weight values that **minimize the total error of the network** over the set of training examples.
- **Repeated** procedures of the following two passes:
  - **Forward pass**: Compute the **outputs** of all units in the network, and the **error** of the output layers.
  - **Backward pass**: The network error is used for updating the weights (**credit assignment problem**).
    - Starting at the output layer, **the error is propagated backwards through the network, layer by layer**. This is done by recursively computing the local gradient of each neuron.

Backprop

- Back-propagation training algorithm illustrated:



→ *Network activation*
Error computation
Forward Step

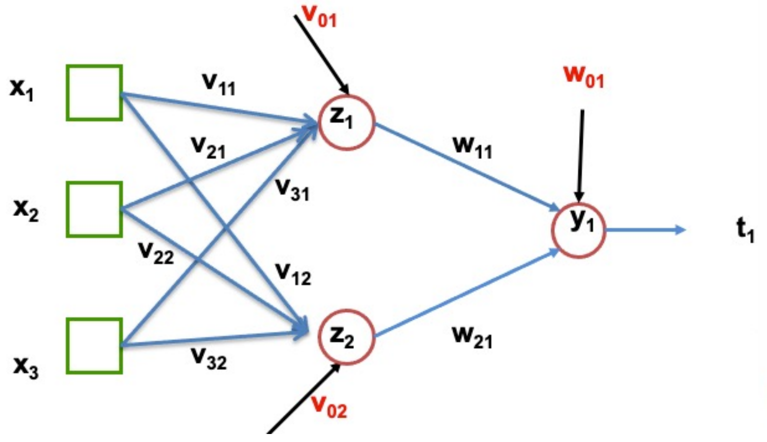
⇠ *Error propagation*
Backward Step

- Backprop adjusts the weights of the NN in order to minimize the network total mean squared error.

Back Propagation

$$o_x = 1/(1+e^{-v_x})$$

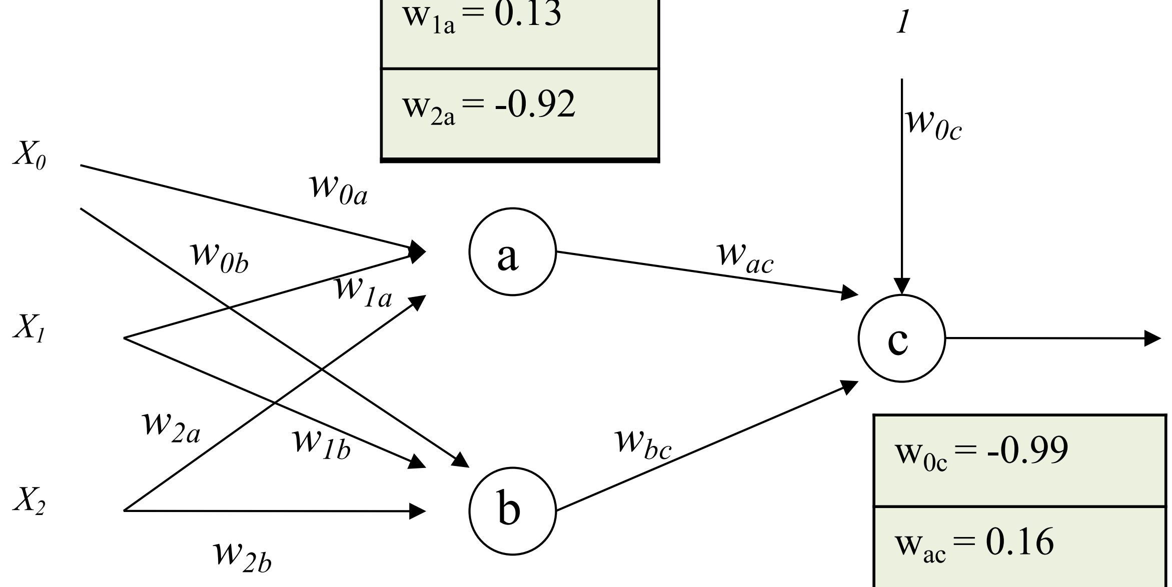
```
Initialize all weights to small random numbers.
While (E(W) unsatisfactory & Itera < Max_Iteration)
  • For each training example, Do
      1. Input the training example to the network
         and compute the network outputs
      2. For each output unit $k$
             $\delta_k \leftarrow o_k(1-o_k)(d_k-o_k)$
      3. For each hidden unit $h$
             $\delta_h \leftarrow o_h(1-o_h)\sum_{k\in outputs} w_{h,k}\delta_k$
      4. Update each network weight $w_{i,j}$
             $w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$
         where
             $\Delta w_{i,j} = \eta\delta_j x_{i,j}$
    EndFor
EndWhile
```



Back Propagation Example

- XOR
  -

X_0	X_1	X_2	Y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0



$w_{oa} = 0.34$
$w_{1a} = 0.13$
$w_{2a} = -0.92$

$w_{0b} = -0.12$
$w_{1b} = 0.57$
$w_{2b} = -0.33$

$w_{0c} = -0.99$
$w_{ac} = 0.16$
$w_{bc} = 0.75$

Initial weights

Back Propagation Example

η=0.5; $o_x = 1/(1+e^{-v_x})$; **For instance {(1, 0, 0), 0}**

Neuro a		Neuro b		Neuro c	
$w_{oa} = 0.34$	$v_a = 0.34$ $o_a = 0.58$	$w_{0b} = -0.12$	$v_b = -0.12$ $o_b = 0.47$	$w_{0c} = -0.99$	$v_c = -0.54$ $o_c = 0.37$
$w_{1a} = 0.13$		$w_{1b} = 0.57$		$w_{ac} = 0.16$	
$w_{2a} = -0.92$		$w_{2b} = -0.33$		$w_{bc} = 0.75$	
$\delta_a = o_a(1-o_a)\Sigma_k w_{ak} \cdot \delta_k$ $= 0.58*(1-0.58)*0.16*(-0.085)$ $= -0.003$				$\delta_c = o_c(1-o_c)(d_c - o_c)$ $= 0.37*(1-0.37)*(0-0.37)$ $= -0.085$	
$\Delta w_{oa} = \eta\delta_a x_{oa} = 0.5*(-0.003)*1$ $= -0.015$				$\Delta w_{oc} = \eta\delta_c x_{oc} = 0.5*(-0.085)*1$ $= -0.043$	

Back Propagation Example

Neuro a	Neuro b	Neuro c
$w_{oa} = w_{oa} + \Delta w_{oa} = 0.34 - 0.015$ $= 0.325$		
$w_{1a} = w_{1a} + \Delta w_{1a} = 0.13 + 0$		
$w_{2a} = w_{2a} + \Delta w_{2a} = -0.92 + 0$		
$\Delta w_{oa} = \eta \delta_a x_{oa} = 0.5*(-0.003)*1$ $= -0.015$		$\Delta w_{oc} = \eta \delta_c x_{oc} = 0.5*(-0.085)*1$ $= -0.043$
$\Delta w_{1a} = \eta \delta_a x_{1a} = 0.5*(-0.003)*0 = 0$		
$\Delta w_{2a} = \eta \delta_a x_{2a} = 0.5*(-0.003)*0 = 0$		

Outline

- Multi-layer Neural Networks
- Feedforward Neural Networks
  - FF NN model
  - Backpropogation (BP) Algorithm
  - Practical Issues of FFNN

Network training:

- Two types of training:

➢ **Incremental mode** (on-line, stochastic, or per-observation) Weights updated after each instance is presented

➢ **Batch mode** (off-line or per -epoch)
Weights updated after all the patterns are presented

Stopping criteria

- Sensible stopping criterions:
  - total mean squared error change: Back-prop is considered to have converged when the absolute rate of change in the average squared error per epoch is sufficiently small (in the range [0.01, 0.1]).
  - generalization based criterion: After each epoch the NN is tested for generalization using a different set of examples (validation set). If the generalization performance is adequate then stop.

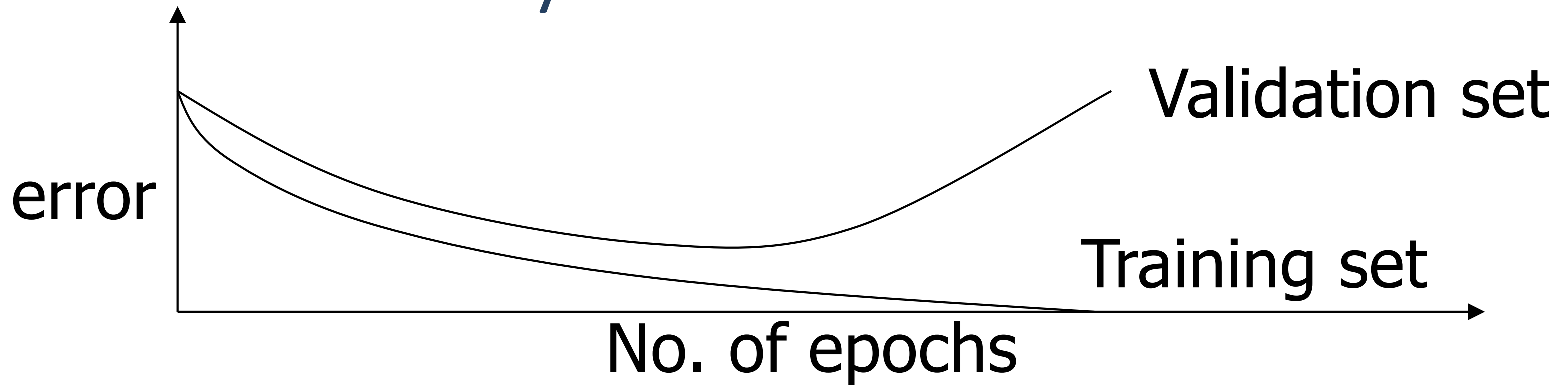
Use of Available Data Set for Training

- The available data set is normally split into three sets as follows:
  - Training set – use to update the weights. Patterns in this set are repeatedly in random order. The weight update equation are applied after a certain number of patterns.
  - Validation set – use to decide when to stop training only by monitoring the error.
  - Test set – Use to test the performance of the neural network. It should not be used as part of the neural network development cycle.

Earlier Stopping - Good Generalization

- Running too many epochs may overtrain the network and result in overfitting and perform poorly in generalization.

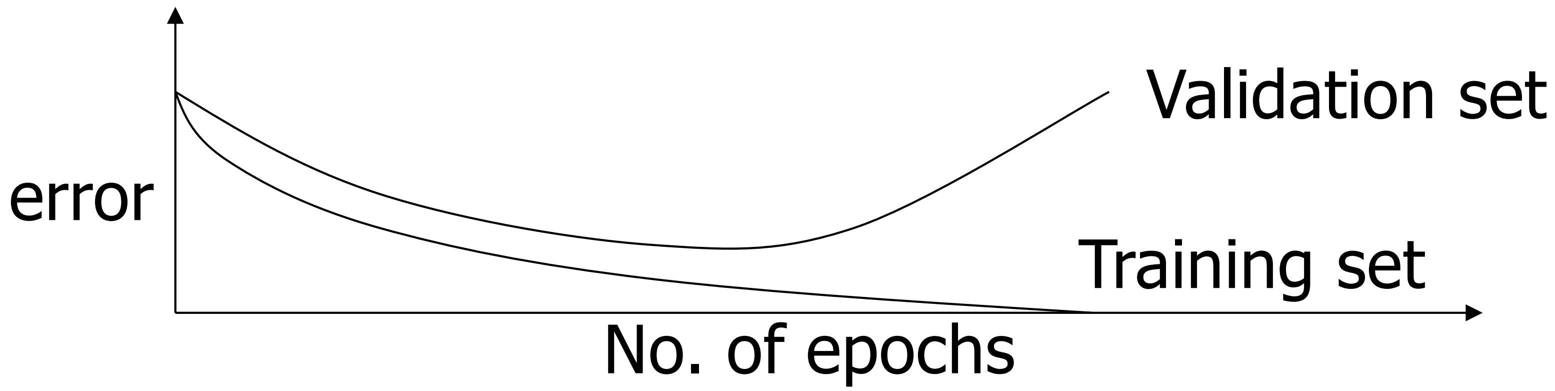
➢ Keep a hold-out validation set and test accuracy after every epoch. Maintain weights for best performing network on the validation set and stop training when error increases beyond this.



Model Selection by Cross-validation

- Too few hidden units prevent the network from learning adequately fitting the data and learning the concept.
- Too many hidden units leads to overfitting.

➢ Similar cross-validation methods can be used to determine an appropriate number of hidden units by using the optimal test error to select the model with optimal number of hidden layers and nodes.



NN DESIGN

- Data representation
- Network Topology
- Network Parameters
- Training

Data Representation

- Data representation **depends on the problem**. In general NNs work on **continuous (real valued) attributes**. Therefore symbolic attributes are encoded into continuous ones.
- Attributes of different types may have different ranges of values which affect the training process. **Normalization** may be used, like the following one which scales each attribute to assume values between 0 and 1.

$$x_i = \frac{x_i - \min_i}{\max_i - \min_i}$$

for each value x_i of attribute i , where $\min_i$ and $\max_i$ are the minimum and maximum value of that attribute over the training set.

Network Topology

- The number of layers and neurons depend on the specific task. In practice this issue is solved by trial and error.
- Two types of adaptive algorithms can be used:
  - start from a large network and successively remove some neurons and links until network performance degrades.
  - begin with a small network and introduce new neurons until performance is satisfactory.

Network parameters

- How are the weights initialized?
- How is the learning rate chosen?
- How many hidden layers and how many neurons?
- How many examples in the training set?

Initialization of weights

- Weight initializing affects:
  - the speed of learning/convergence
  - the quality of the local minimum point reached
  - the generalization performance of the learned NN
- In general, initial weights are randomly chosen, with between -1.0 and 1.0 or -0.5 and 0.5.
- Weight initialization method:
  - singular value decomposition, genetic algorithm, simulated annealing, factoring, least square method, pre-training step, …

Choice of learning rate

- The right value of η depends on the application. Values between 0.1 and 0.9 have been used in many applications.

Size of Training set

- Rule of thumb:
  - the number of training examples should be at least five to ten times the number of weights of the network.
- Other rule:

$$N > \frac{|W|}{(1-a)}$$

|W|= number of weights
a=expected accuracy

Applications of FFNN

Classification, pattern recognition:

- FFNN can be applied to tackle non-linearly separable learning tasks.
  - Recognizing printed or handwritten characters
  - Face recognition
  - Classification of loan applications into credit-worthy and non-credit-worthy groups
  - Analysis of sonar radar to determine the nature of the source of a signal
  - Speech Recognition

Regression and forecasting:

- FFNN can be applied to learn non-linear functions (regression) and in particular functions whose inputs is a sequence of measurements over time (time series).