Data Mining II: Advanced Methods and Techniques

Lecture 3

Natasha Balac, Ph.D.

Review

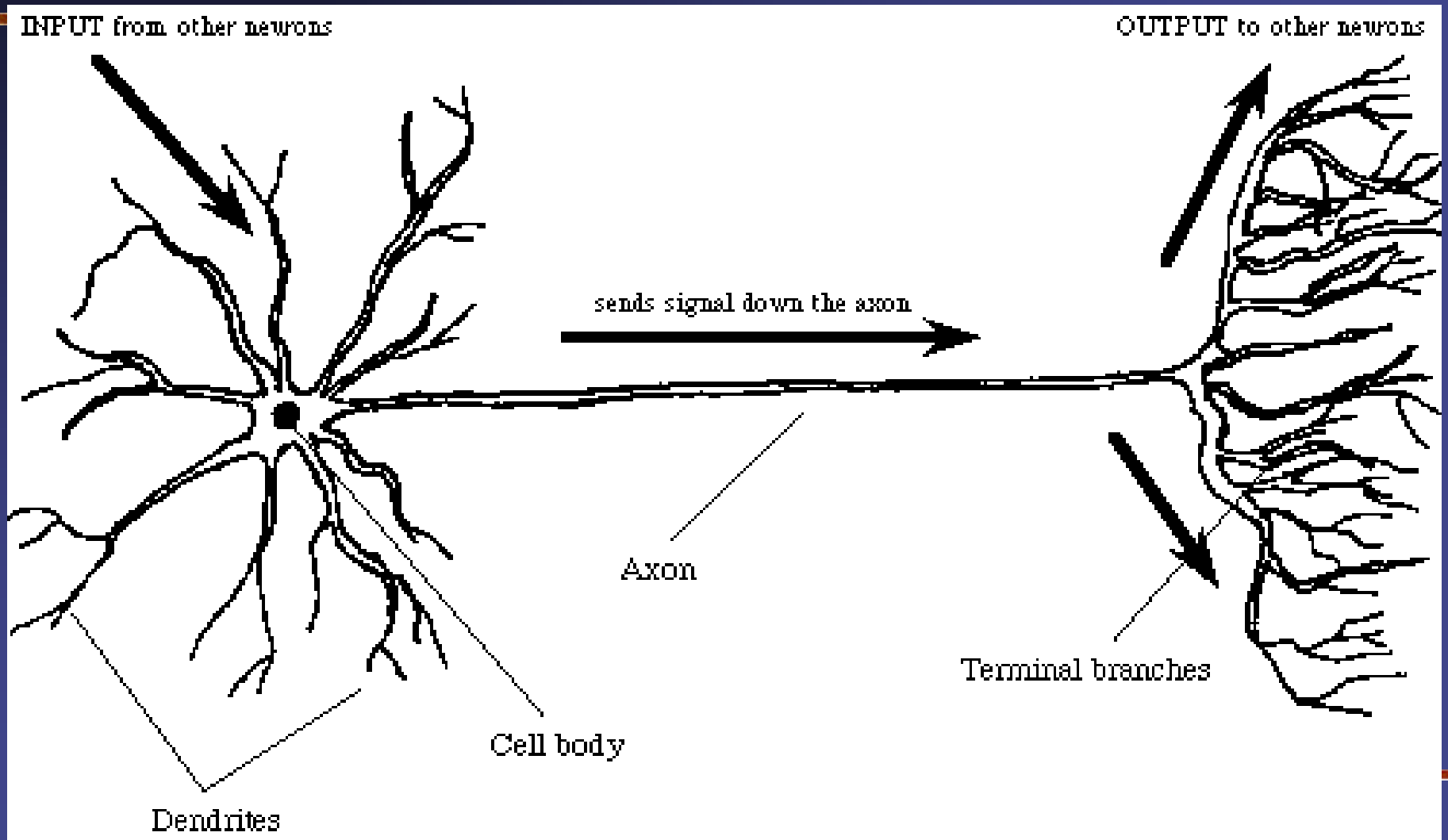
NN Definition

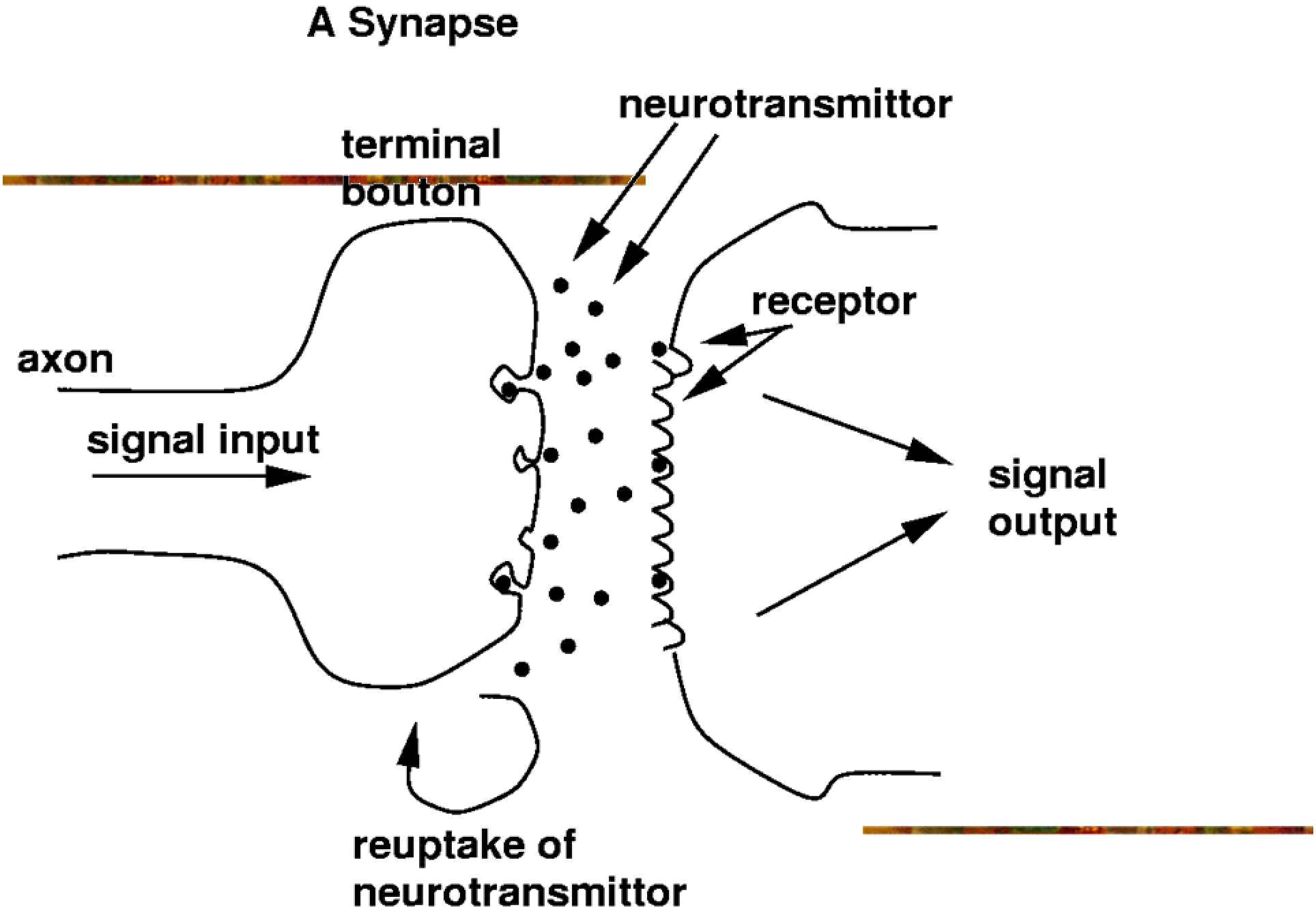
- NN is a network of many simple processors (*units*), each possibly having a small amount of local memory
- The units are connected by communication channels (*connections*) which usually carry numeric data of various kinds
- The units operate only on their local data and on the inputs they receive via the connections

About Neural Networks

- Some NNs are models of biological neural networks and some are not
- Historically, much of the inspiration for the field of NNs came from the desire to produce artificial systems capable of sophisticated, "intelligent", computations similar to those that the human brain routinely performs
- Possibly to enhance our understanding of the human brain

Neuron




A Synapse
neurotransmittor
terminal
bouton
receptor
axon
signal input
signal
output
reuptake of
neurotransmittor

A Simple Artificial Neuron

- Basic computational element (model neuron) is often called a **node** or **unit**
- It receives input from some other units, or perhaps from an external source
- Each input has an associated **weight** *w*, which can be modified so as to model synaptic learning

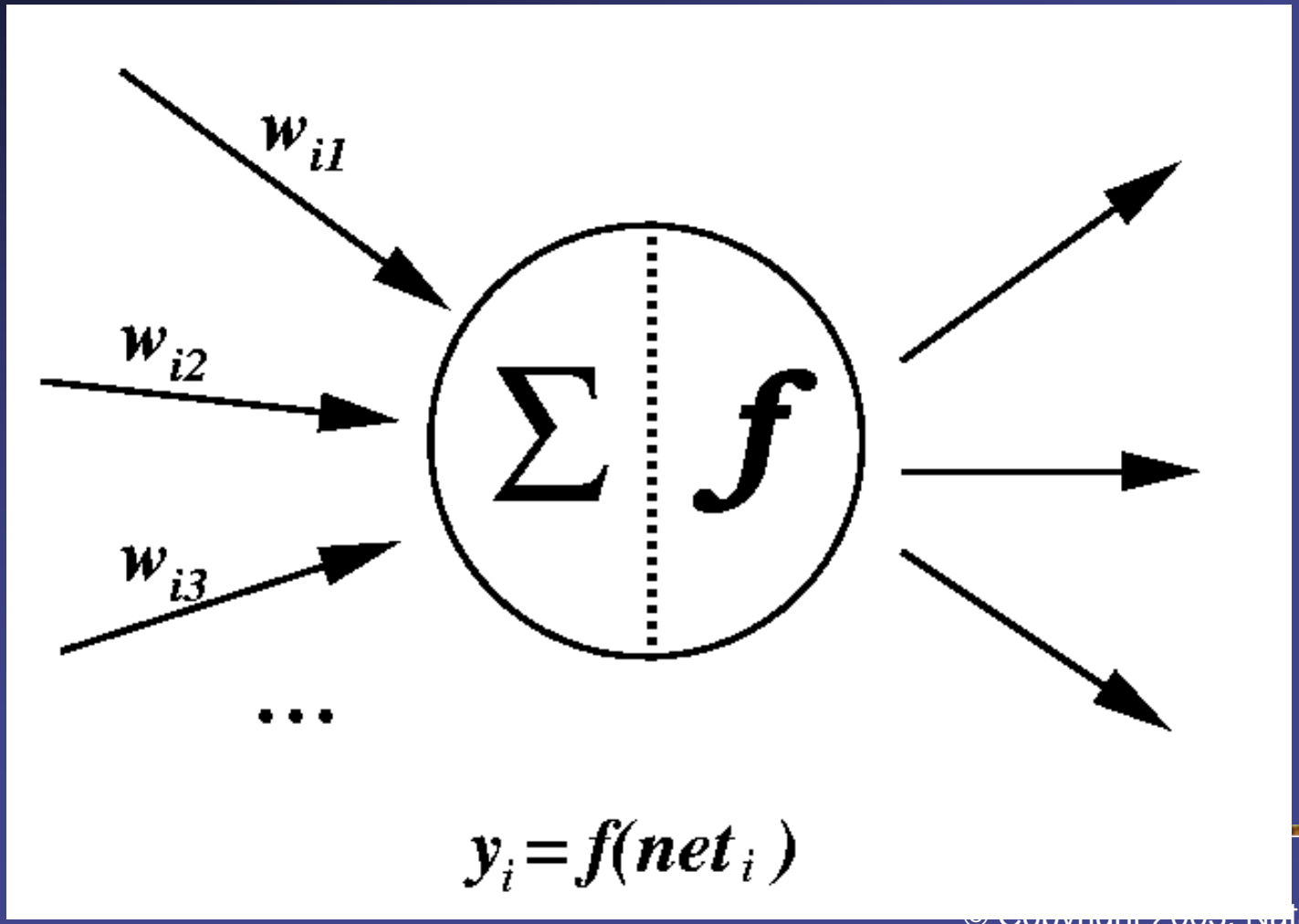
A Simple Artificial Neuron

- The unit computes some function *f* of the weighted sum of its inputs:

$$y_i = f\left(\sum_j w_{ij}\, y_j\right)$$

- Its output, in turn, can serve as input to other units

Linear unit

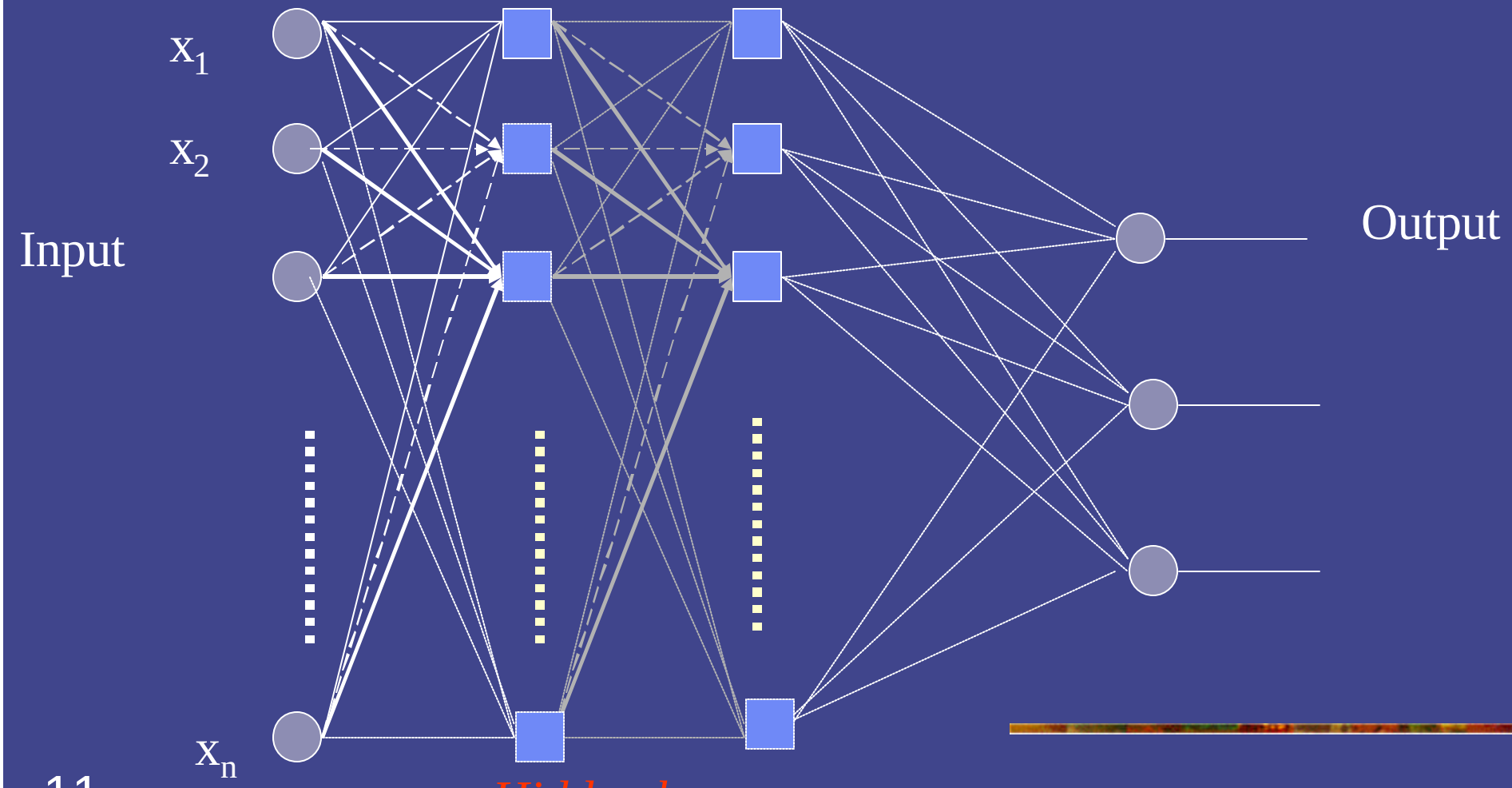


Artificial Neural Networks (ANNs)

A network with interactions mimicking the brain functionality

- **UNITs**: artificial neuron (linear or nonlinear input-output unit), small numbers, typically less than a few hundred
- **INTERACTIONs**: encoded by weights, how strong a neuron affects other neurons
- **STRUCTUREs**: can be feedforward, feedback or recurrent

Example
Four-layer network



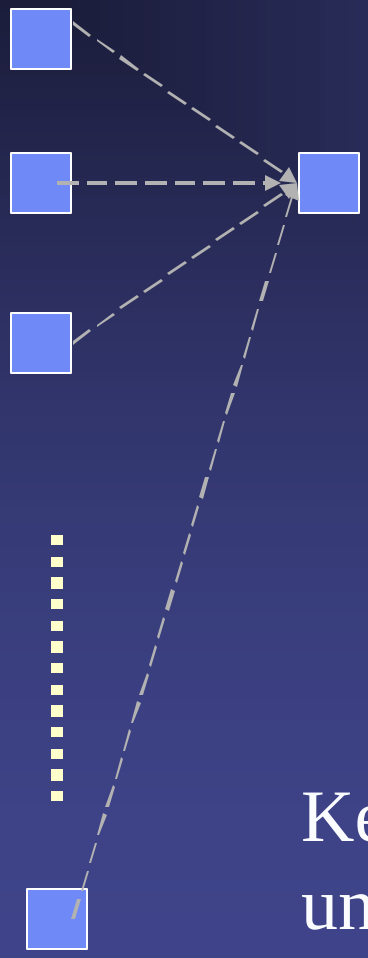
General Artificial Neuron Model

- Has five components, shown in the following list

  The subscript i indicates the i-th input or weight

  1. **A set of inputs, x_i**
  2. **A set of weights, w_i**
  3. **A bias, u**
  4. **An activation function, f**
  5. **Neuron output**, y

General Artificial Neuron Model



$$y_i \;?\; f\left(\sum_{j?1}^{m} w_{ij}x_j \;?\; b_i\right)$$

Key to understanding ANNs is to understand/generate the local input-output relationship

Network as a data model

- We can view a network as a model
  - which has a set of parameters associated with it
- Networks transform input data into an output
  - Transformation is defined by the network parameters
- Parameters set/adapted by optimisation/adaptive procedure: 'learning'
- Given a set of data points network (model) can be trained so as to generalize

LEARNING: extracting principles from data

- **Mapping/function needs to be learnt**
  - **various methods available**
- **Supervised learning: have a teacher, telling you what is correct/incorrect answer**
- **Unsupervised learning: no teacher, net learns by itself**
- **Reinforcement learning: have a critic, *wrong or correct***

Generalization Problem

- System must be able to classify **UNSEEN** patterns from the patterns it has seen
  - I.e. Must be able to **generalize** from the data in the training set
- Intuition: biological neural networks do this well, so maybe artificial ones can do the same?
- As they are also shaped by experiences maybe we'll also learn about how the brain does it

Two Class Classification

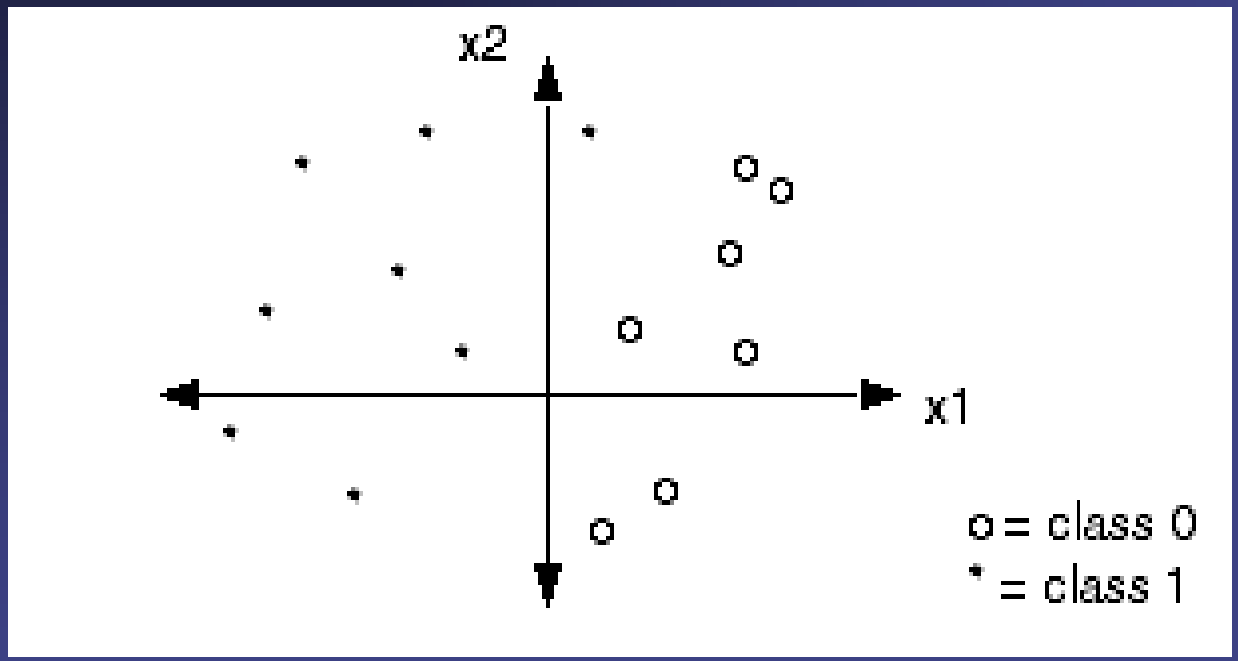
- **For 2 class classification we want network output y (function of inputs and network parameters) to be:**

  y(x, w) = 1 if x is an a

  y(x, w) = -1 if x is a b

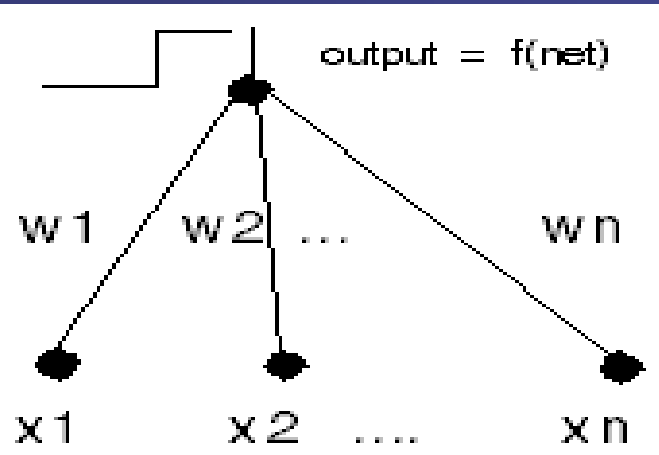
- **where x is an input vector and the network parameters are grouped as a vector w**
- **y is known as a discriminant function**
  - **it discriminates between 2 classes**

Two Classes 0 and 1
Two Inputs x1 and x2



What does the network look like?

- If there are just 2 classes we only need 1 output node
- The target is 1 if the example is in class 1
- and the target is 0 (or -1) if the target is in class 0
- Use a binary step function to guarantee an appropriate output value



Learning

- **As the network mapping is defined by the parameters we use the data set to perform learning:**

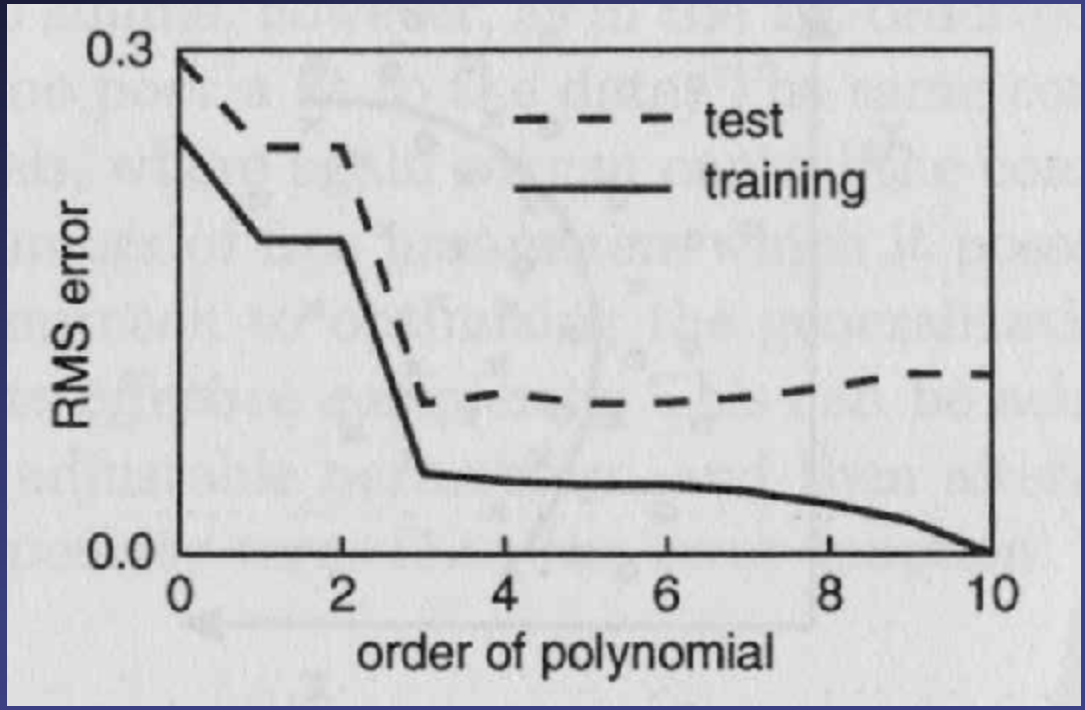
change weights or interaction between neurons according to the training examples (and possibly prior knowledge of the problem)

- **The purpose of learning is to minimize**
  - **training errors on learning data**
    - **learning error**
  - **prediction errors on new, unseen data**
    - **generalization error**

Learning

- When the errors are minimized, the network discriminates between the 2 classes
- **Error function** measures the network performance based on the training error
- Optimisation algorithms are then used to minimize the learning errors and train the network

Model Complexity

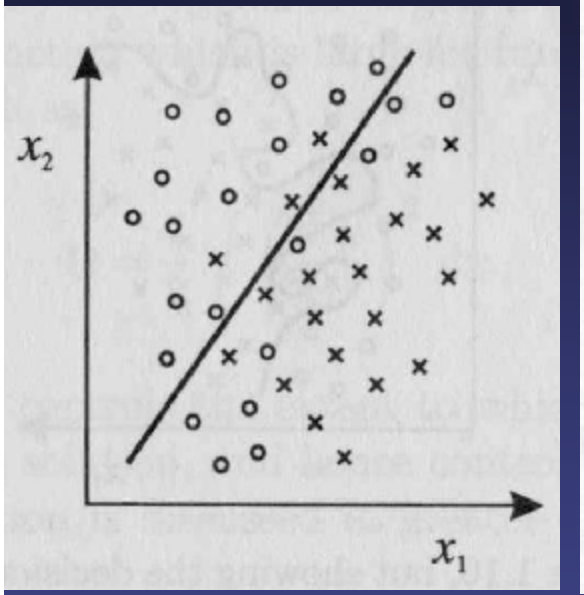


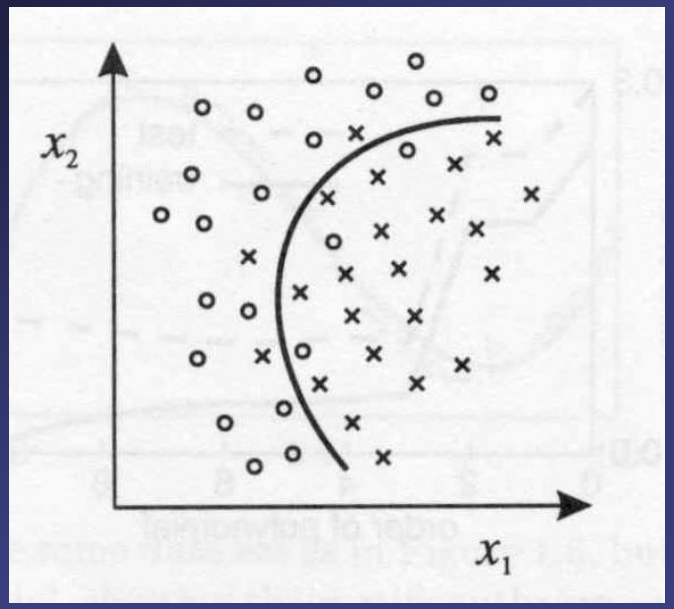
As the model complexity grows performance improves for a while but starts to degrade after reaching an optimal level

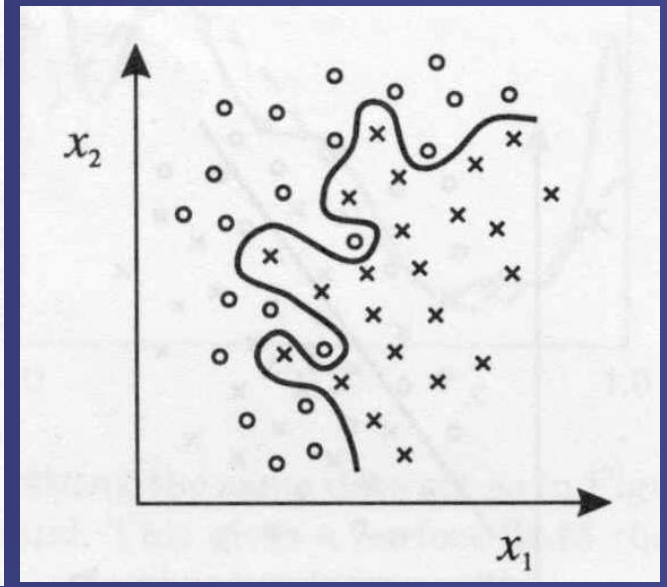
Model Complexity

- Note that training error continues to go down as model matches the fine-scale detail of the data (the noise)
- We want to model the ***intrinsic dimensionality*** of the data otherwise - problem of ***overfitting***
- Problem of over-training
  - where a model is trained for too long and models the data too exactly and loses its generality

Generalisation Problem

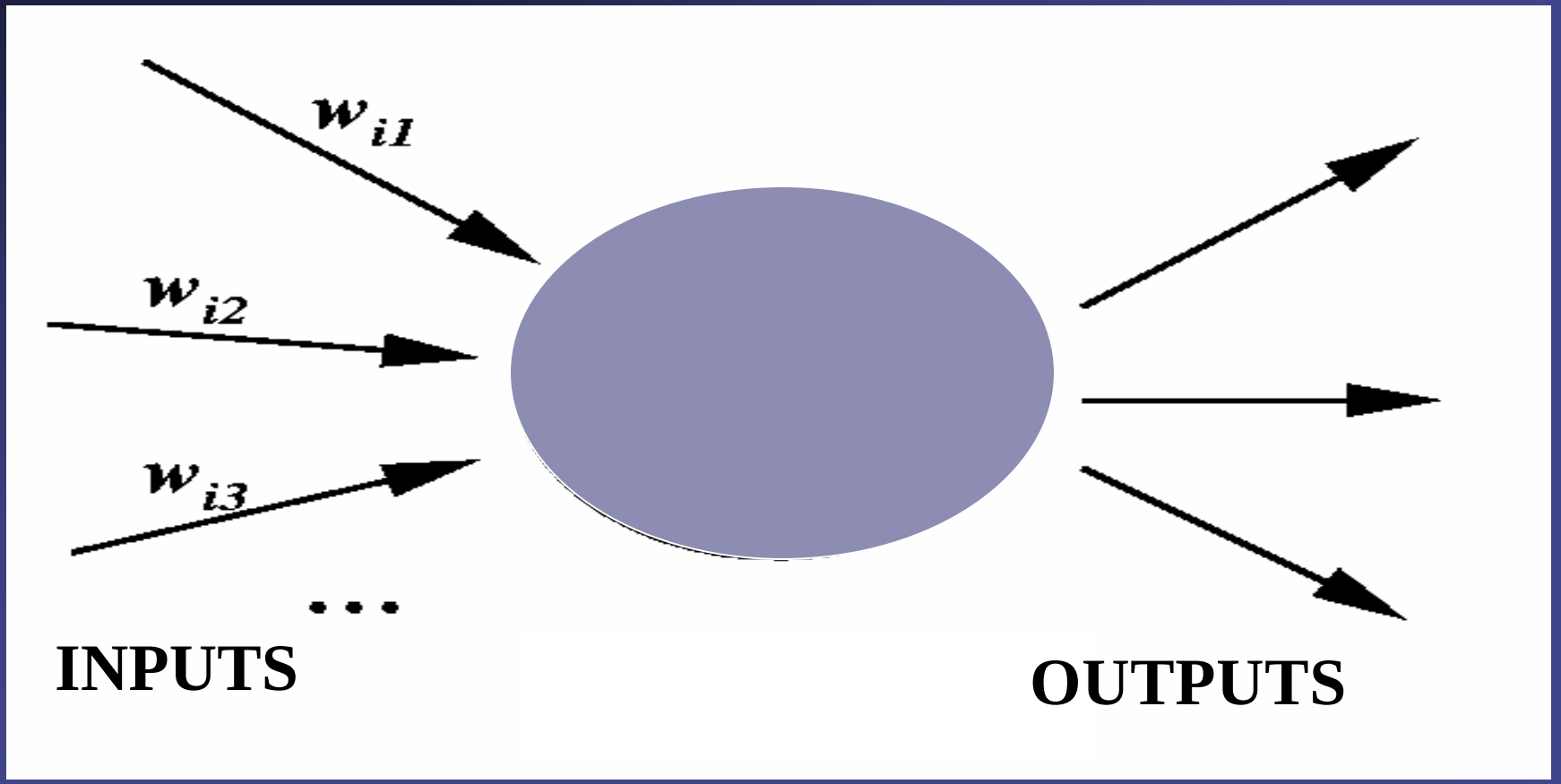






- **A model with too much flexibility does not generalize well resulting in a non-smooth decision boundary**

What Does An Artificial Neuron Look Like?

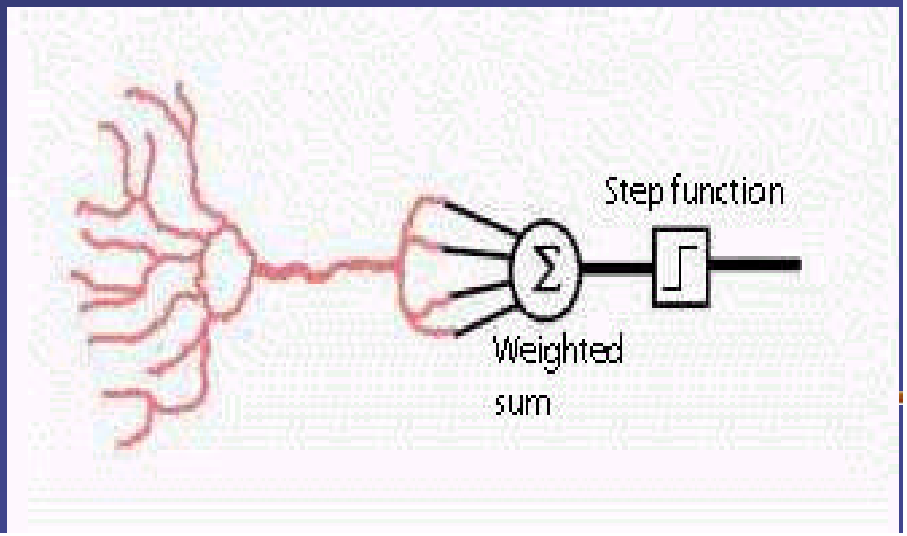


Neuron

- Each input into the neuron has its own weight
- Weight is a floating point number
  - will be adjust when training the network
- Weights in most neural nets can be both negative and positive
  - providing excitory or inhibitory influences to each input
- Each input enters the nucleus it's multiplied by its weight

Perceptron Activation

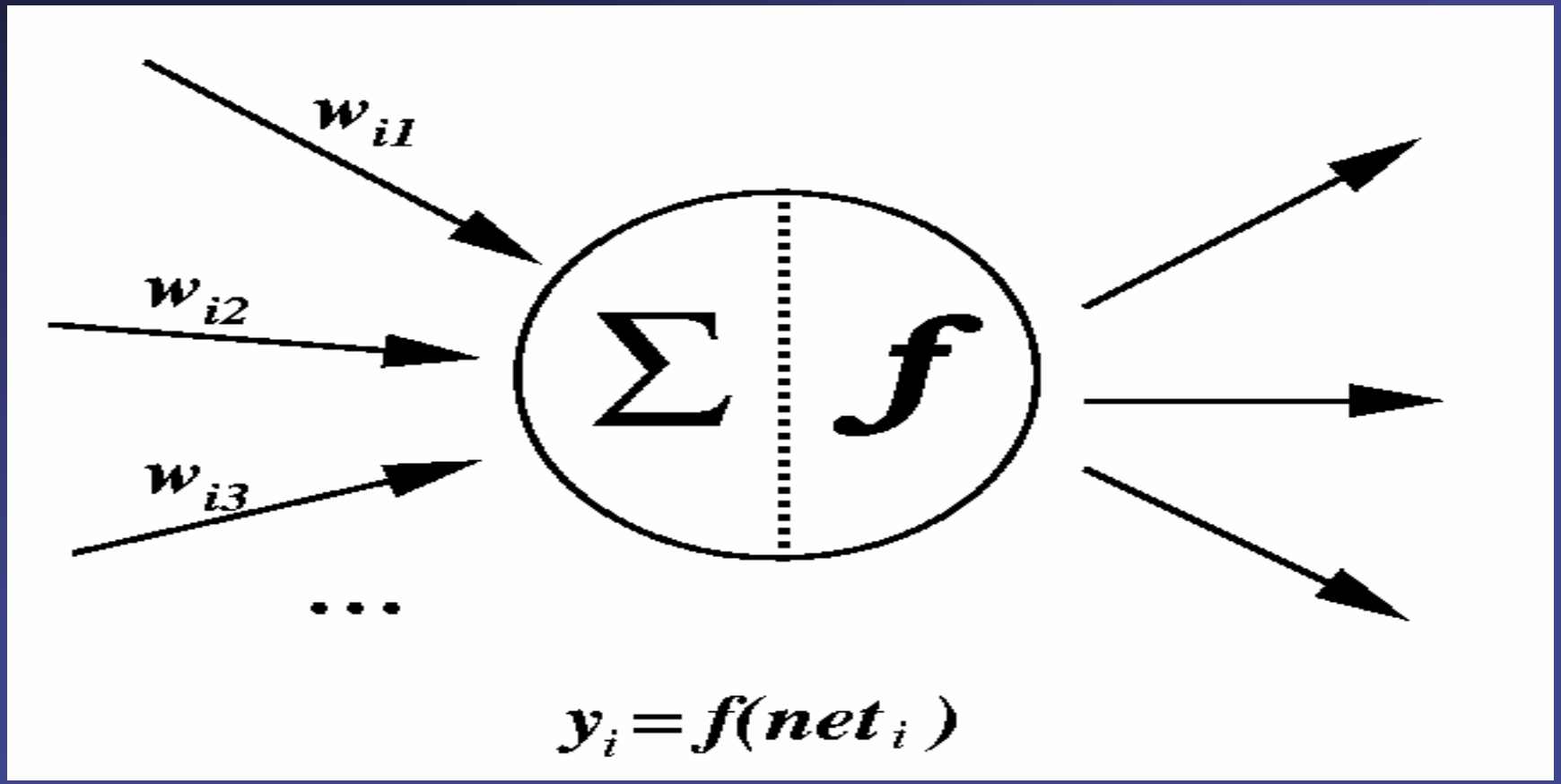
- The nucleus then sums all these new input values which gives us the ***activation***
  - floating point number which can be negative or positive
- If the activation is greater than a threshold value
  - the neuron outputs a signal
  - If the activation is less than
    - 1 the neuron outputs zero
- Called a ***step*** function



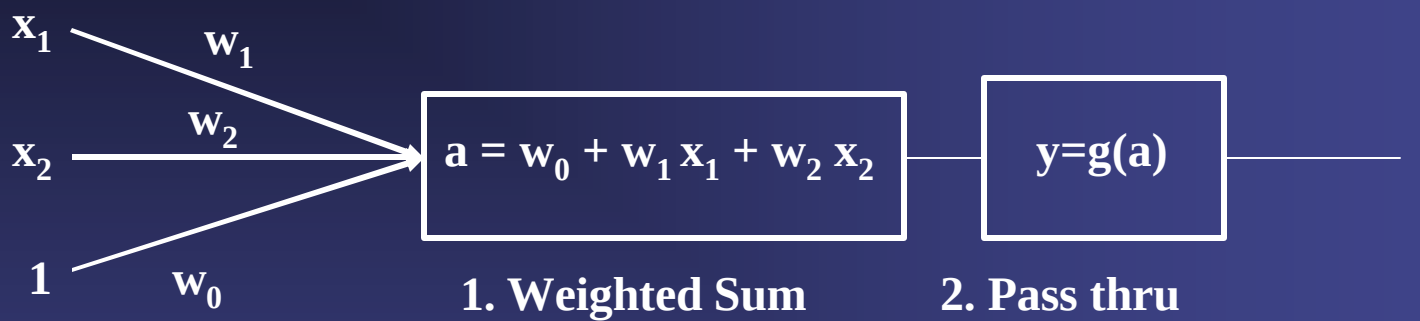
Structure

- A neuron can have any number of inputs 1-> n
- The inputs represented as: $x_1, x_2, x_3 \ldots x_n$
- Corresponding input weights : $w_1, w_2, w_3 \ldots w_n$
- Summation of the weights multiplied by the inputs: $x_1w_1 + x_2w_2 + x_3w_3 \ldots + x_nw_n$
  - called activation value
  - $a = x_1w_1+x_2w_2+x_3w_3 \ldots +x_nw_n$
    - $a=? \ w_ix_i$

Artificial Neuron



The Perceptron as a Classifier



Output = class decision

1. Weighted Sum of the inputs

2. Pass thru Heaviside function:
T(a)= -1 if a < 0
T(a)= 1 if a >= 0

View the bias as another weight from an input which is constantly on

If we group the weights as a vector $\underline{w}$ we therefore have the net output y given by:

$$y = g(\underline{w} \cdot \underline{x} + w_0)$$

Activation Functions

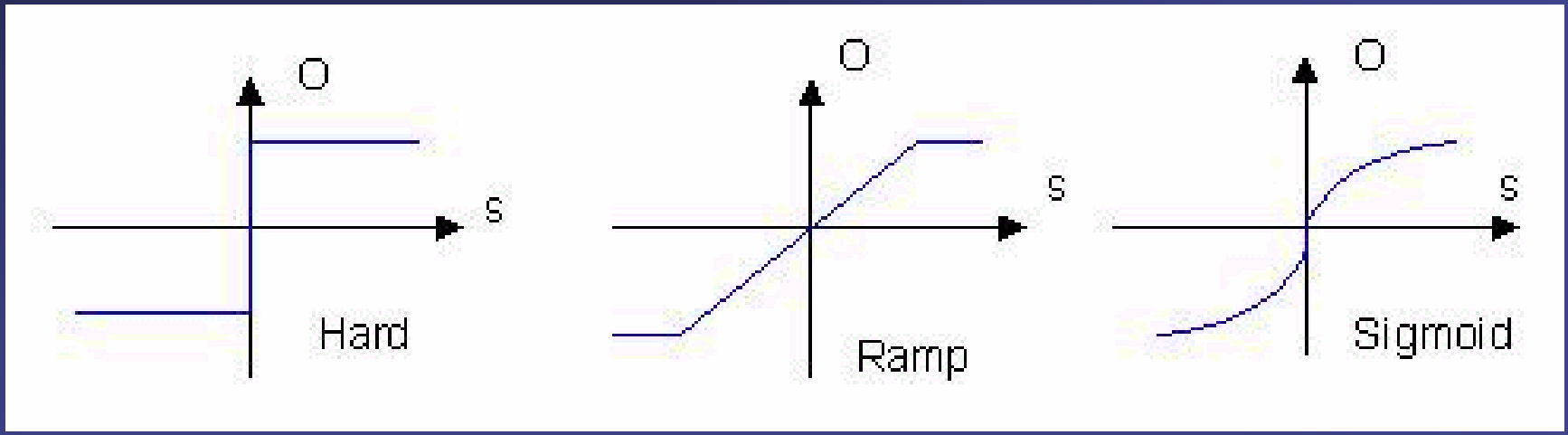


Figure from Answermath.com

Network Learning

training the weights by gradient descent

- Set of training data from known classes to be used in conjunction with an error function $E(\underline{w})$ (eg sum of squares error) to specify an error for each instantiation of the network

- Then $\underline{w}_{new} = \underline{w}_{old} - \eta \nabla E(\underline{w})$

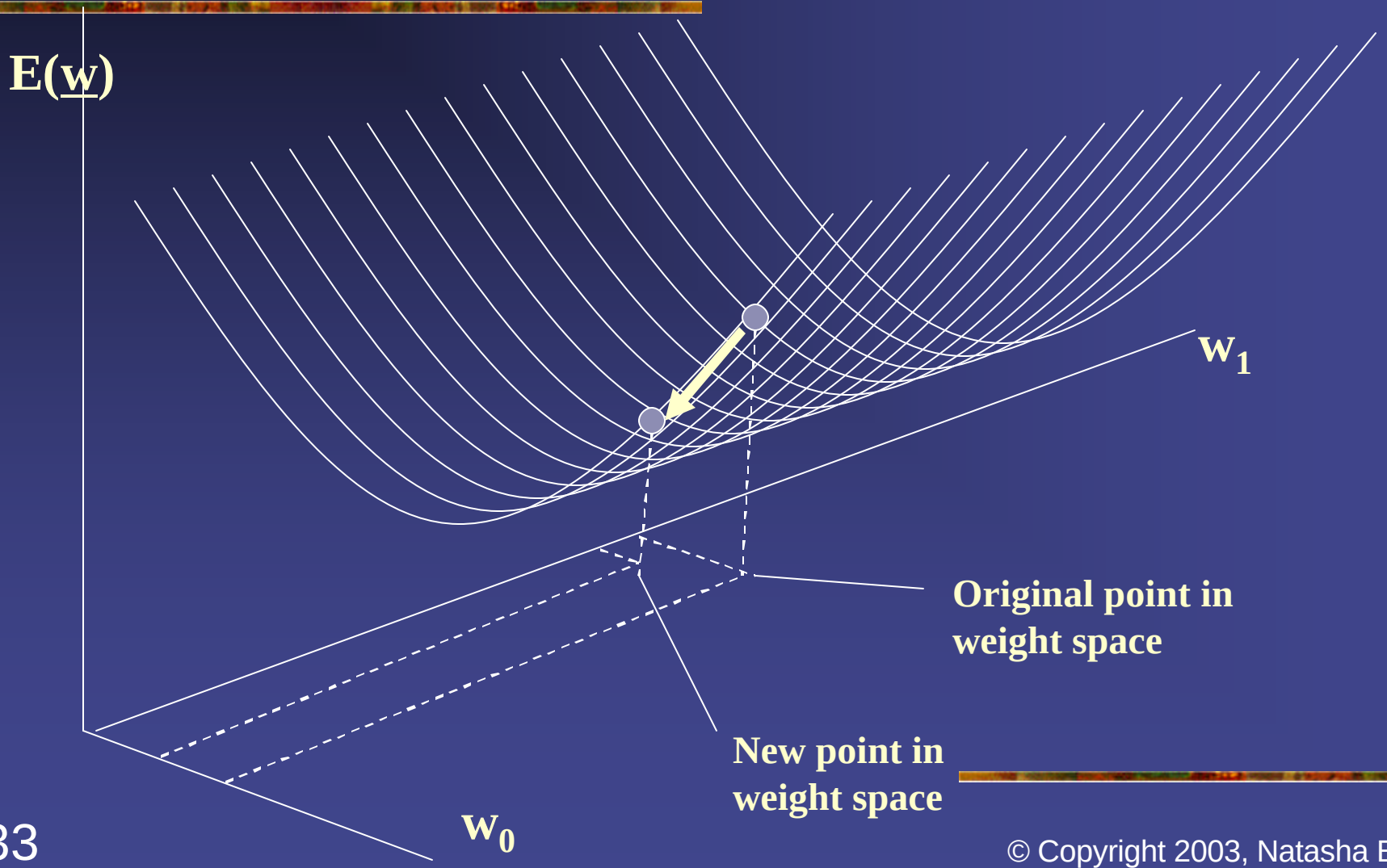
- So

$$w_{jk}^{t+1} = w_{jk}^{t} - \eta \frac{\partial E}{\partial w_{jk}}$$

- where $\nabla E(\underline{w})$ is a vector representing the gradient and η is the learning rate (small, positive)

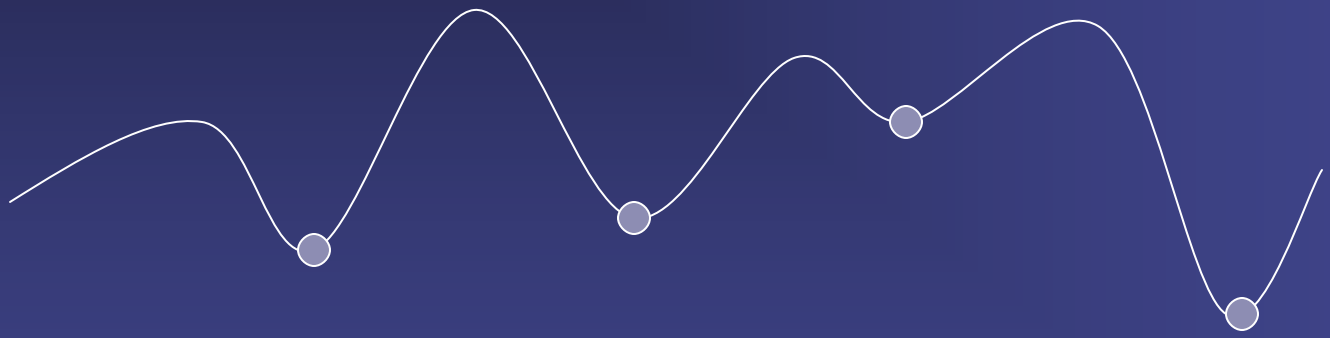
1. Moves downhill in direction $-E(\underline{w})$ (steepest downhill since $E(\underline{w})$ is the direction of steepest increase)
2. How far to go at each step is determined by the value of η

Gradient Descent



Gradient Descent

- Equivalent to hill-climbing
- Can be problems knowing when to stop
- Local minima



  - can have multiple local minima (note: for perceptron, E(w) only has a single global minimum, so this is not a problem)
  - gradient descent goes to the closest local minimum:
    - solution: random restarts from multiple places in weight space

The Fall of the Perceptron

- Marvin Minsky & Seymour Papert (1969). *Perceptrons*, MIT Press, Cambridge, MA.

- **Before long researchers had begun to discover the Perceptron's limitations**
- **Unless input categories were "linearly separable"**
  - **a perceptron could not learn to discriminate between them**
- **Unfortunately it appeared that many important categories were not linearly separable**
- **Example**
  - **inputs to an XOR gate that give an output of 1 (10 & 01) are not linearly separable from those that do not (00 & 11)**

Perceptron Disadvantages

- Similarly, the change in left hand end only must be sufficient to change classification
- Therefore changing both ends must take the sum even further across threshold
- Problem is because of single layer of processing local knowledge cannot be combined into global knowledge
- Add more layers!

The Perceptron Controversy

- **Minsky and Papert's book was a block to the funding of research in neural networks for more than ten years**

- **The book was widely interpreted as showing that neural networks are basically limited and fatally flawed**

- **What IS controversial is whether Minsky and Papert shared and/or promoted this belief ?**

- **Following the rebirth of interest in artificial neural networks, Minsky and Papert claimed that they had not intended such a broad interpretation of the conclusions they reached in the book Perceptrons**

Terminology

The input vector

- All the input values of each perceptron are collectively called the input vector of that perceptron

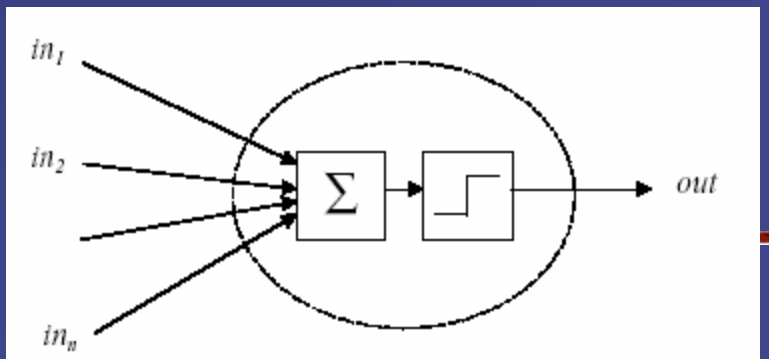
The weight vector

- Similarly, all the weight values of each perceptron are collectively called the weight vector of that perceptron

The McCulloch-Pitts Neuron

- A vastly simplified model of a real neuron known as a ***Threshold Logic Unit***:

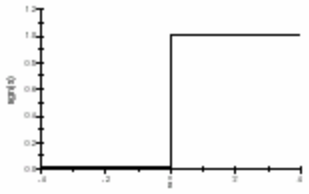
1. A set of synapses (connections) brings in activations from other neurons
2. A processing unit sums the inputs, and then applies a non-linear activation function
3. An output line transmits the result to other neurons



Functions

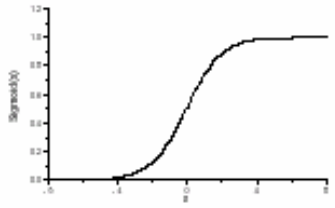
- A function $y = f(x)$ describes a relationship (mapping) from x to y.
- **Example 1** The sign function $\mathrm{sgn}(x)$ is defined as

$$\mathrm{sgn}(x)=\begin{cases}1 & \text{if } x\geq 0\\ 0 & \text{if } x<0\end{cases}$$



- **Example 2** The sigmoid function $\mathrm{Sigmoid}(x)$ is defined as

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



McCulloch-Pitts neuron

- **Output out of a McCulloch-Pitts neuron is related to its n inputs in by**

$$out = \mathrm{sgn}(\sum_{i=1}^{n} in_i - \theta)$$

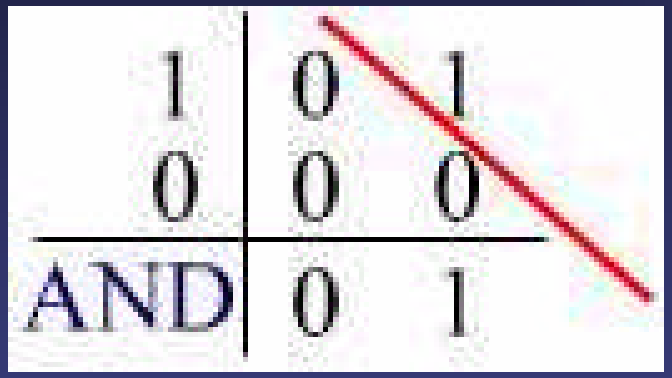
- **Where ? is the threshold**

$$out = 1 \quad \text{if} \quad \sum_{k=1}^{n} in_k \geq \theta \qquad out = 0 \quad \text{if} \quad \sum_{k=1}^{n} in_k < \theta$$

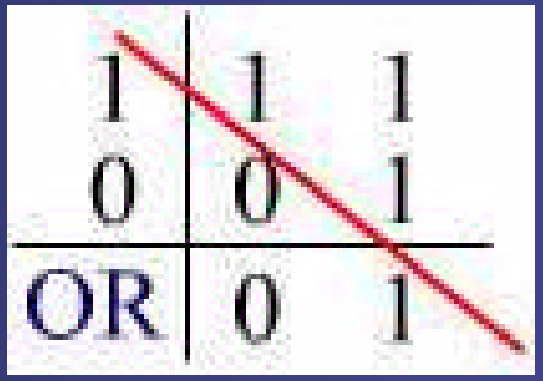
McCulloch-Pitts neuron

- McCulloch-Pitts neuron is an extremely simplified model of real biological neurons
- McCulloch-Pitts neurons are computationally very powerful
  - Synchronous assemblies of such neurons are capable, in principle, of universal computation
    - as powerful as our ordinary computers

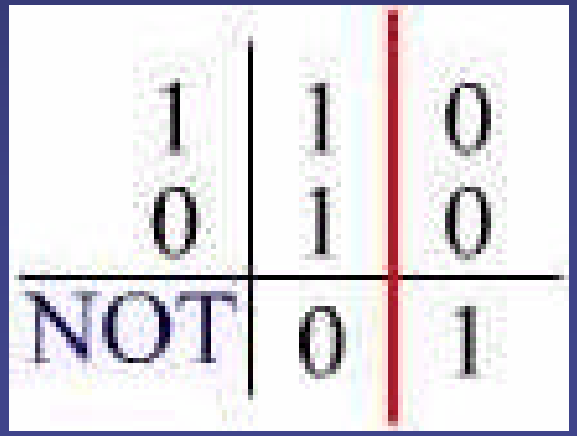
Linearly Separable Logic Functions



	1	0	1
	0	0	0
AND		0	1



	1	1	1
	0	0	1
OR		0	1



	1	1	0
	0	1	0
NOT		0	1

Non - Linearly Separable Functions

- Not all logic operators are linearly separable
- XOR operator is not linearly separable and cannot be achieved by a single perceptron
- This problem could be overcome by using more than one perceptron arranged in feedforward network

Non - Linearly Separable Functions

- Since it is impossible to draw a line to divide the regions containing either 1 or 0
  - XOR function is not linearly separable

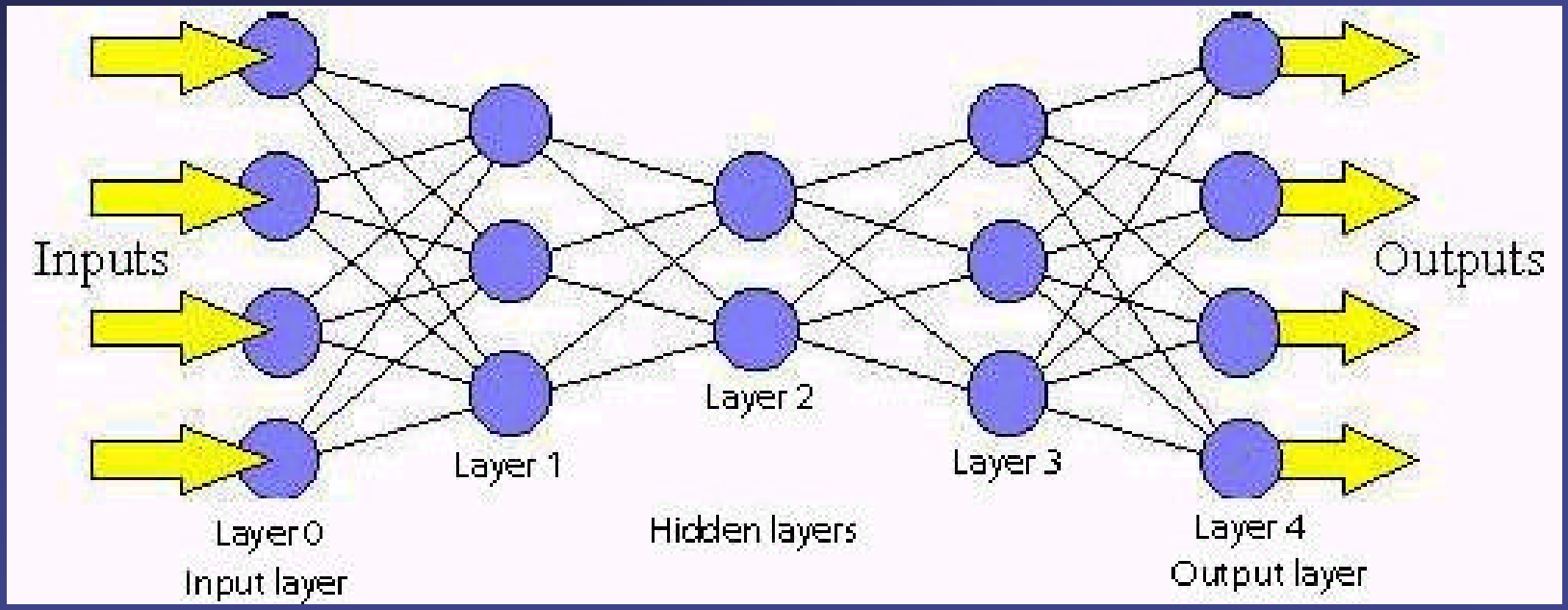
1	1	0
0	0	1
XOR	0	1

Feed-forward networks characteristics

- Perceptrons are arranged in layers
  - first layer taking in inputs and the last layer producing outputs
  - The middle layers have no connection with the external world - called hidden layers
- Each perceptron in one layer is connected to every perceptron on the next layer
- Information is constantly "fed forward" from one layer to the next
- There is no connection among perceptrons in the same layer

Feed-Forward networks

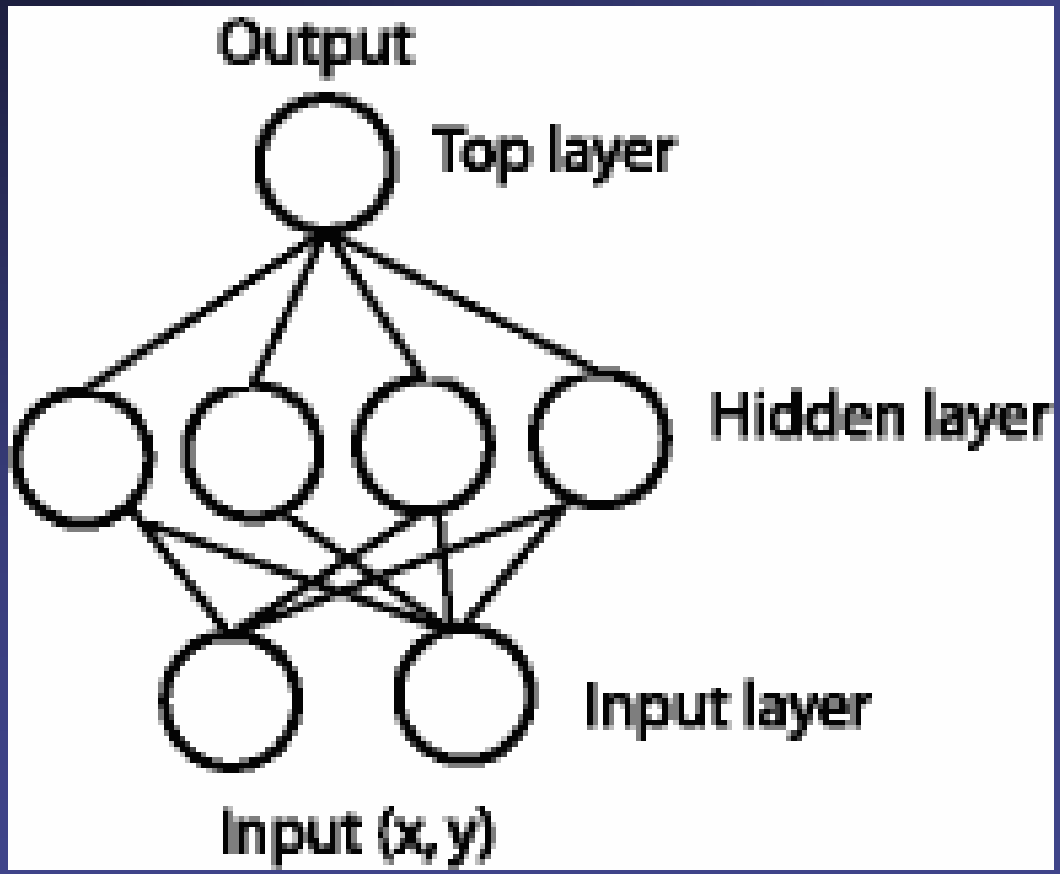
- Feed-Forward network



Why Are Feedforward Networks Interesting?

- Single perceptron can classify points into two regions that are linearly separable
- What about separation of points into two regions that are not linearly separable?

A feed-forward network with one hidden layer



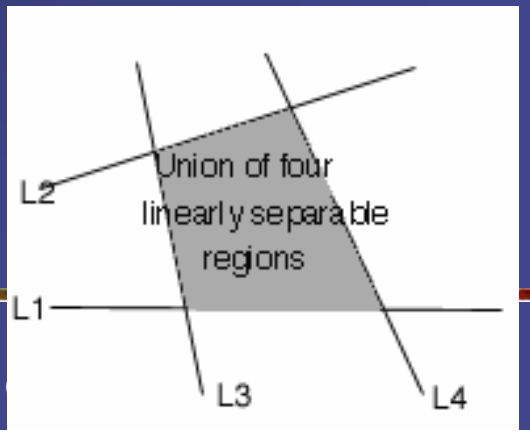
Example: Feed-forward Network at Work

- The same (x, y) is fed into the network through the perceptrons in the input layer
- With four perceptrons that are independent of each other in the hidden layer
- The point is classified into 4 pairs of linearly separable regions
  - each of which has a unique line separating the region

Intersection of 4 linearly separable regions

- The top perceptron performs logical operations on the outputs of the hidden layers so that the whole network classifies input points in 2 regions that might not be linearly separable
- Using the AND operator on these four outputs
- Intersection of the 4 regions

  that forms the center region



Learning in feed-forward networks

- Supervised learning
  - in which pairs of input and output values are fed into the network for many cycles, so that the network 'learns' the relationship between the input and output
- We provide the network with a number of training samples, which consists of an input vector i and its desired output o

Backpropagation Learning

- In backpropagation learning, every time an input vector of a training sample is presented, the output vector o is compared to the desired value d
- The comparison is done by calculating the squared difference of the two:

$$Err = (d-o)^2$$

Backpropagation Learning

- The value of Err tells us how far away we are from the desired value for a particular input
- The goal of backpropagation is to minimize the sum of Err for all the training samples
  - so that the network behaves in the most "desirable" way
  - Minimize:

$$\sum \text{Err} = (d-o)^2$$

Backpropagation Learning

- We can express Err in terms of
  - input vector (i)
  - weight vectors (w)
  - threshold function of the perceptions
- Using a continuous function (instead of the step function) as the threshold function
  - we can express the gradient of Err with respect to the w in terms of w and i

Backpropagation Learning

- Given the fact that decreasing the value of w in the direction of the gradient leads to the most rapid decrease in Err
- We update the weight vectors every time a sample is presented using

$$w_{new} = w_{old} - n \frac{\delta Err}{\delta w}$$

  - where n is the learning rate (a small number ~ 0.1)

Training

- Once the neural network has been created it needs to be trained
- Initialize the neural net with random weights and then feed it a series of inputs
- For each input we check to see what its output is and adjust the weights accordingly so that whenever it sees a positive example it outputs 1 otherwise 0

Learning:
The training process

- Progressive adaptation of the synaptic connection values to let the NN learn the desired behavior
  - Feed the NN with an input from training data
  - Compare the NN's outputs with the training data's output
  - The differences are used to compute the error of the NN's response

Learning:
The training process

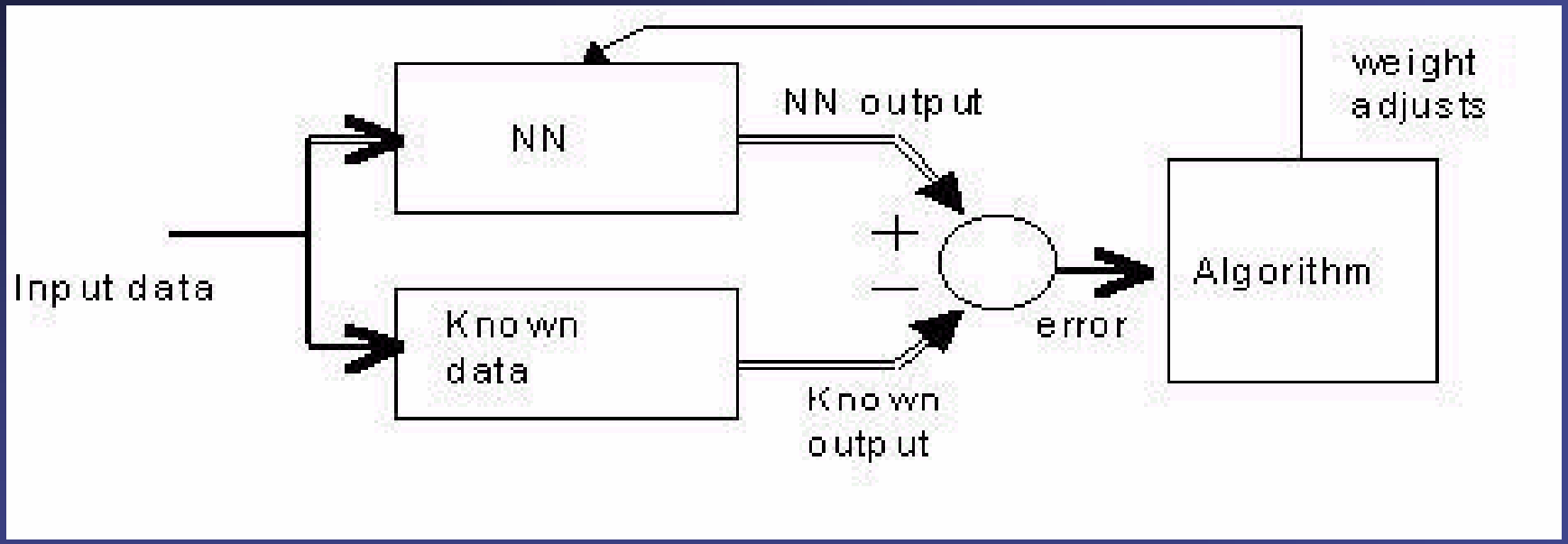
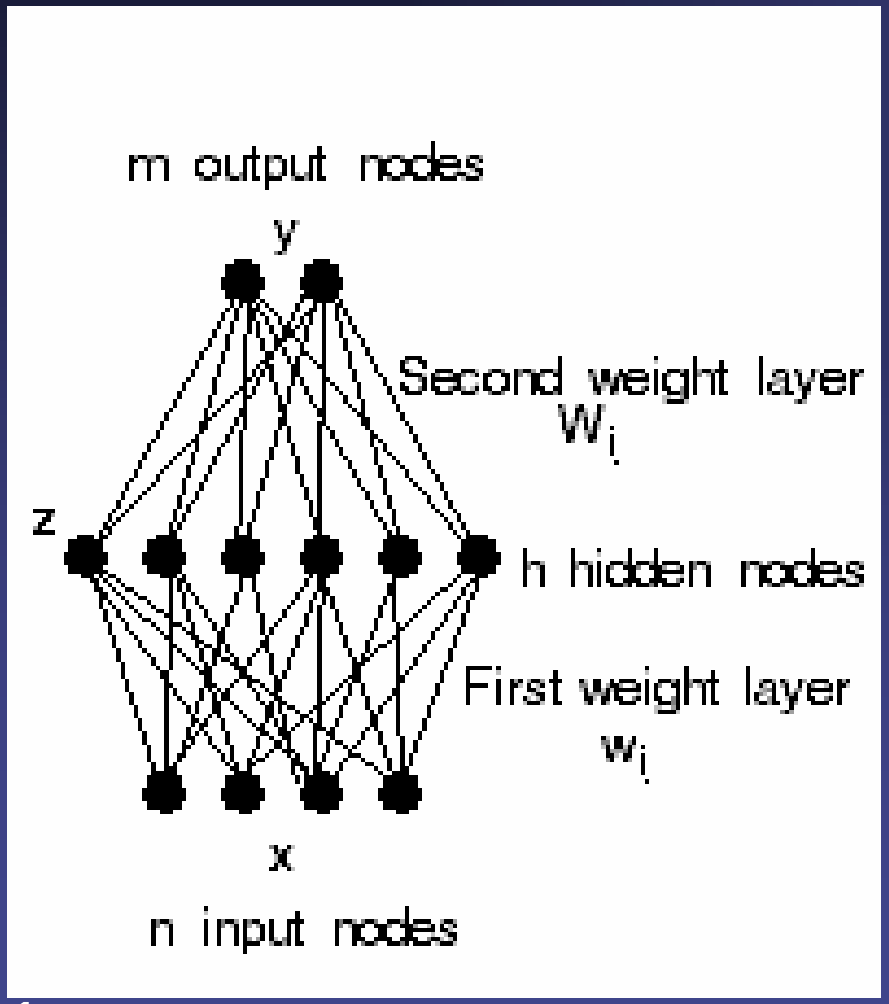


Figure from Answermath.com

Today

- Backpropagation Learning!

Multilayer Network



w_{ij} = weight connecting input j -> hidden i
W_{ij} = weight connecting hidden j -> output i

- **Theorem: a 2-layer network can approximate ANY function arbitrarily closely as long as *there are enough hidden nodes***
- **How many hidden nodes are enough? No one can say exactly**

About Backpropagation

- **Much early research in networks was abandoned because of the severe limitations of single layer linear networks**
- **It was not until the 1980s that backpropagation became widely known**
- **Backpropagation also refers to the very efficient method that was discovered for computing the gradient**
- ***Multilayer nets are much harder to train than single layer networks***
  - ***convergence is much slower***

What is backprop?

- Short for "backpropagation of error“
- *Backpropagation* refers to the method for computing the gradient of the error function with respect to the weights for a feedforward network
- Straightforward but elegant application of the chain rule of elementary calculus
- *Backpropagation* or *backprop* often refers to a training method that uses backpropagation to compute the gradient
- *Backprop* network is a feedforward network trained by backpropagation

What is backprop?

- "Standard backprop" is a euphemism for the *generalized delta rule*, the training algorithm that was popularized by Rumelhart, Hinton, and Williams in 1986
- Remains the most widely used supervised training method for neural nets
- The generalized delta rule (including momentum) is called the "heavy ball method" in the numerical analysis literature

What is backprop?

- Standard backprop can be used for both batch training
  - in which the weights are updated after processing the entire training set
- and incremental training
  - in which the weights are updated after processing each case

Backprop using batch vs. incremental

- For batch training, standard backprop usually converges (eventually) to a local minimum - if one exists
- For incremental training, standard backprop does not converge to a stationary point of the error surface
  - To obtain convergence the learning rate must be slowly reduced
  - Methodology called "stochastic approximation" or "annealing"

What is backprop?

- A large area of NN research is devoted to attempts to speed up backprop
- Incremental backprop can be highly efficient for some large data sets if you select a good learning rate
  - but that can be difficult to do

What learning rate should be used for standard backprop?

- Too low a learning rate makes the network learn very slowly
- Too high a learning rate makes the weights and objective function diverge
  - so there is no learning at all
- If the objective function is quadratic - as in linear models
  - good learning rates can be computed from the Hessian matrix

What learning rate should be used for backprop?

- In a typical feedforward NNs with hidden units (where objective function has many local and global optima) the optimal learning rate often changes dramatically during the training process
- Trying to train a NN using a constant learning rate is usually a tedious process requiring much trial and error

What learning rate should be used for backprop?

- Many other variants of backprop have been invented
- Most suffer from the same theoretical flaw as standard backprop
  - the magnitude of the change in the weights (the step size) should NOT be a function of the magnitude of the gradient
- In some regions of the weight space, the gradient is small and you need a large step size
  - this happens when you initialize a network with small random weights

What learning rate should be used for backprop?

- In other regions of the weight space
  - the gradient is small and you need a small step size
    - this happens when you are close to a local minimum
- Similarly large gradient may call for either a small step or a large step
- Many algorithms try to adapt the learning rate
  - but any algorithm that multiplies the learning rate by the gradient to compute the change in the weights is likely to produce erratic behavior when the gradient changes abruptly

How to count layers?

- Matter of considerable dispute
- Some people count layers of *units*
  - Some count the input layer and some don't
- Some people count layers of *weights*
  - How they count skip-layer connections?
- To avoid ambiguity we will use
  - 2-hidden-layer network
  - Not a 4-layer network - as some would call it
  - Nor 3-layer network - as others would call it

What are cases, variables, vectors, features?

- A vector of values presented at one time to all the input units of a neural network is called a **case**, **example**, **pattern**, **sample**, etc.
  - Case may include not only input values, but also target values and possibly other information
  - A vector of values presented at different times to a single input unit is often called an **input variable** or **feature**
  - To a statistician, it is a **predictor**, **regressor**, **covariate**, **independent variable**, **explanatory variable**, etc.

What are vectors and variables?

- Vector of target values associated with a given output unit of the network during training is called a *target variable*
- To a statistician it is usually a response or dependent variable
- *Data set* is a matrix containing one or (usually) more cases
- We assume that cases are rows of the matrix and variables are columns
- Often term *input vector* is ambiguous
  - it can mean either an input case or an input variable

What are population and sample?

- There seems to be no term in the NN literature for the set of all cases that you want to be able to generalize to
- Statisticians call this set the "population"
- There is no consistent term for the set of cases that are available for training and evaluating an neural network
- Statisticians call this set the "sample“
  - Sample is usually a subset of the population

What are training, validation and test set in NN?

- Distinctions is crucial
  - Terms "validation" and "test" sets are often confused
- Training set
  - A set of examples used for learning, that is to fit the parameters (weights) of the classifier
- Validation set
  - A set of examples used to tune the parameters (architecture - not weights) of a classifier
    - to choose the number of hidden units in a neural network
- Test set
  - A set of examples used only to assess the performance (generalization) of a fully designed and trained classifier

Evaluation

- Simplest approach to the comparison of different networks
  - evaluate the error function using data which is independent of that used for training
- *Hold out method*
- Various networks are trained using *training* data set
- The performance of the networks is compared by evaluating the error function using an independent *validation* set
  - the network having the smallest error based on validation set is selected

Evaluation

- This procedure can lead to some overfitting to the validation set
  - Performance of the selected network is confirmed by measuring its performance on a third independent set of data - *test* set
- Test set - by definition is *never* used to choose among two or more networks
  - Error on the test set provides an unbiased estimate of the generalization error
- Any data set used to choose the best of two or more networks is - by definition - a validation set
  - Error of the chosen network on the validation set is optimistically biased

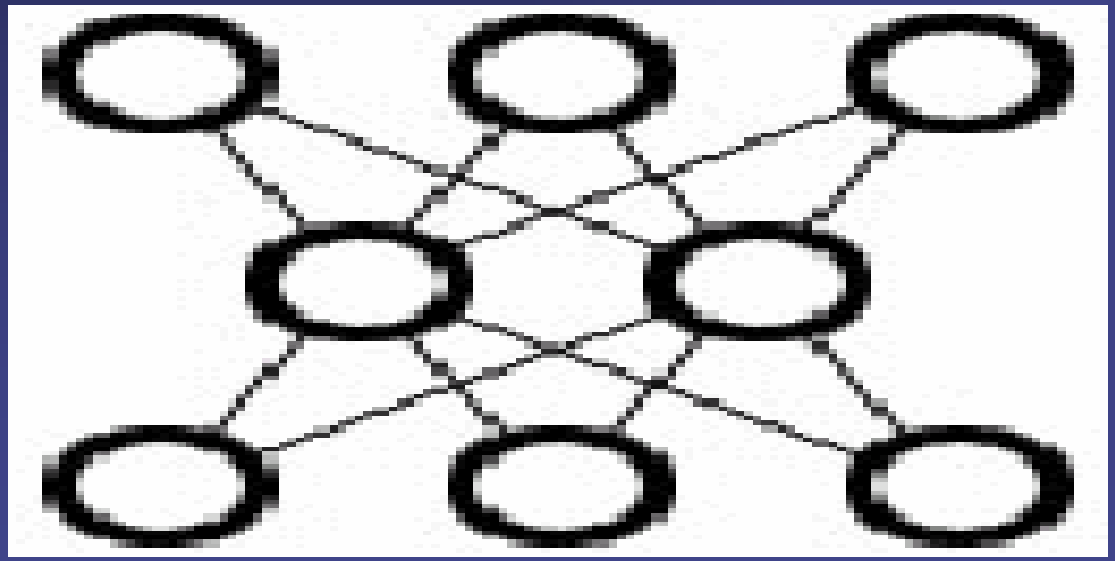
Backpropagation Algorithm

- The backpropagation algorithm is a multi-layer network using a weight adjustment based on the sigmoid function
- The backpropagation method is one of the examples of supervised learning
  - where the target of the function is known
- Let's look at an example of the backpropagation algorithm working on a small Artificial Neural Network
- The Network has a single hidden layer of size two and input and output nodes of size 3

Example

Initialize the weighted links

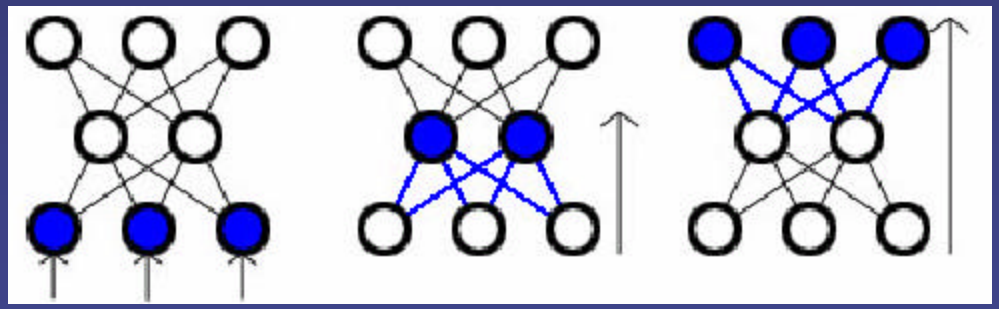
Typically initialized to a small random number



Example

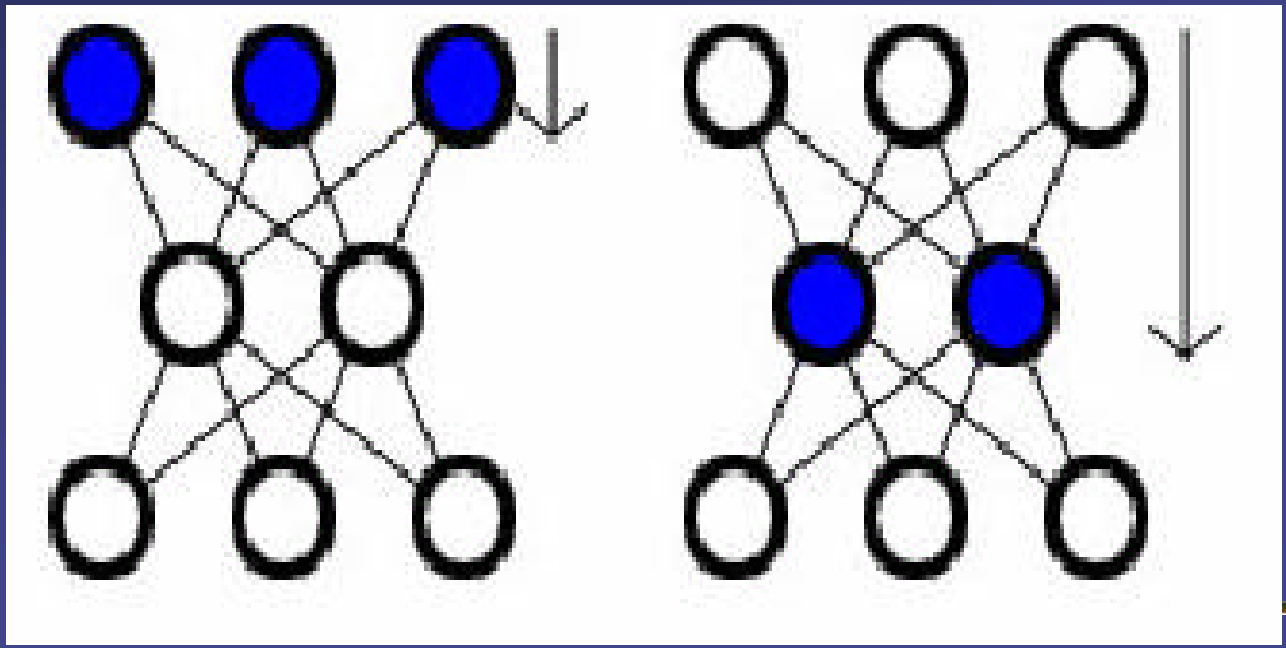
For each training example in the testing set:

- Input the training data to the input nodes
- calculate O_k - which is the output of node k
  - This is done for each node in the hidden layer(s) and output layer



Example

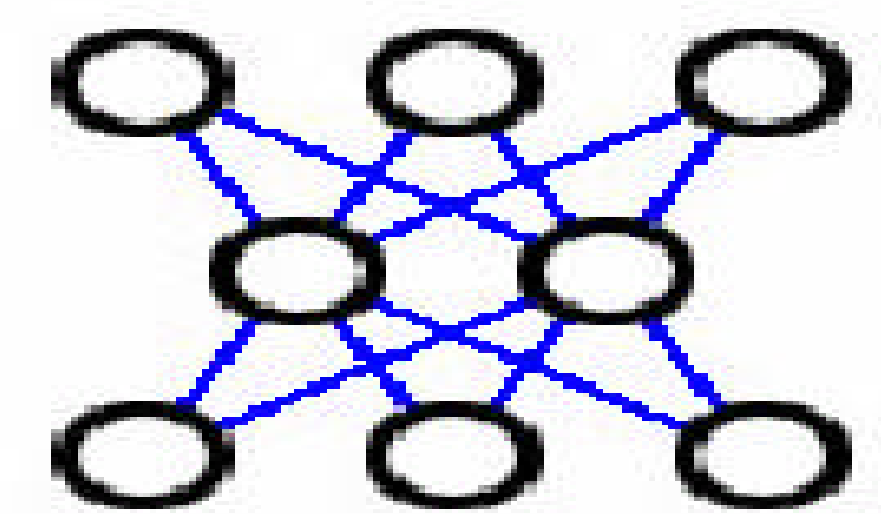
Then calculate δ_k for the each output node, where t_k is the target of the node:
$\delta_k \leftarrow O_k(1 - O_k)(t_k - O_k)$



Example

Now calculate δ_k for the each hidden node:

$$\delta_k \leftarrow O_k(1 - O_k) \sum_{k \in \text{child}(h)} w_{h,k} \bullet \delta_k$$



Finally adjust the weights of all the links, where x_i is the activation and η is the learning rate:

$$W_{i,j} \leftarrow W_{i,j} + \eta \delta_j x_i$$

Practical Example of Backpropagation

- Problem: Learning XOR
- Solution: one-hidden-layer network with two hidden and one output unit
- Such a multi-layer network has capability to correctly classify the XOR examples with the following weights

$w_{10} = -3/2$ $w_{20} = -½$ $w_{30} = -1/2$
$w_{11} = 1$ $w_{21} = 1$ $w_{31} = -2$
$w_{21} = 1$ $w_{22} = 1$ $w_{32} = 1$

XOR Problem

- **x1,x2 | Y**

  (0,0) | 0 The output *unit 3 is off* because the input units 1 and 2 are off
  (0,1) | 1 The output *unit 3 is on* due to the positive excitation from unit 2
  (1,0) | 1 The output *unit 3 is on* due to the positive excitation from unit 2
  (1,1) | 0 The output *unit 3 is off* because of the inhibitory effect from unit 1

Backpropagation Application

- Even though we can get excellent fits for training data, the application of backpropagation becomes more difficult for the prediction of the performance on independent test data
- *There are a lot of choices when applying this method*

Backpropagation Application

- ***Learning rate and local minima***
  - **the selection of a learning rate is of critical importance in finding the true global minimum of the error distance**
- ***Backpropagation training with too small a learning rate will make agonizingly slow progress***
  - ***Too large a learning rate will proceed much faster, but may simply produce oscillations between relatively poor solutions***
- ***Both of these conditions are generally detectable through experimentation and sampling of results after a fixed number of training epochs***

Learning Rate

- Typical values for the learning rate parameter are numbers between 0 and 1:
  - $0.05 < ? < 0.75$
- We would like to use the largest learning rate that still converges to the minimum solution

Momentum

- Empirical evidence shows that the use of momentum in the backpropagation algorithm can be helpful in speeding the convergence and avoiding local minima
- The idea is to stabilize the weight change by making non-radical revisions using a combination of the gradient decreasing term with a fraction of the previous weight change:
  - ? w(t) = -?Ee / ?w(t) + ? ? w(t-1)
- where ? is taken 0? ? ? 0.9, and t is the index of the current weight change

Momentum

- This gives the system a certain amount of inertia since the weight vector will tend to continue moving in the same direction unless opposed by the gradient term
- The effects of momentum
  - smooths the weight changes and suppresses cross-stitching
    - cancels side-to-side oscillations across the error valley
  - when all weight changes are all in the same direction
    - momentum amplifies the learning rate causing a faster convergence
  - enables to escape from small local minima on the error surface

Momentum

- The hope is that the momentum will allow a larger learning rate and that this will speed convergence and avoid local minima
- On the other hand a learning rate of 1 with no momentum will be much faster when no problem with local minima or non-convergence is encountered

Sequential or Random Presentation

- Epoch is the fundamental unit for training
  - length of training often is measured in terms of epochs
- During a training epoch with revision after a particular example
  - examples can be presented in the same sequential order
  - or examples could be presented in a different random order for each epoch
- The random representation usually yields better results

The Randomness

- *Advantages*
  - Gives the algorithm some stochastic search properties
  - Weight state tends to jitter around its equilibrium and may visit occasionally nearby points
  - It may escape trapping in sub-optimal weight configurations
- *Disadvantages*
  - The weight vector never settles to a stable configuration
  - Having found a good minimum it may then continue to wander around it

Random Initial State

- Neural network begin in a random state
- Network weights are initialized to some choice of random numbers with a range typically between -0.5 and 0.5
- Even with identical learning conditions
  - the random initial weights can lead to results that differ from one training session to another
- Training sessions may be repeated until we get the best results

Issues in Learning by Backpropagation

- *While it may be tempting to specify more than one layer of hidden units*
  - *additional layers do not add representational power to the discrimination*
- *Two-hidden-layer networks are more powerful*
  - *but one-hidden-layer networks may be sufficiently accurate for many tasks encountered in practice*
- *Certain real world function can be modeled exactly by one-hidden-layer networks with prohibitively large number of hidden units*
- *One-hidden layer networks assume faster training*

Issues in Learning by Backpropagation

- Training may require thousands of backpropagations
- Backpropagation can get stuck or become unstable when varying the learning rate parameter
  - increasing too much of the learning parameter leads to unstable learning- errors decrease as well as increase during the training process

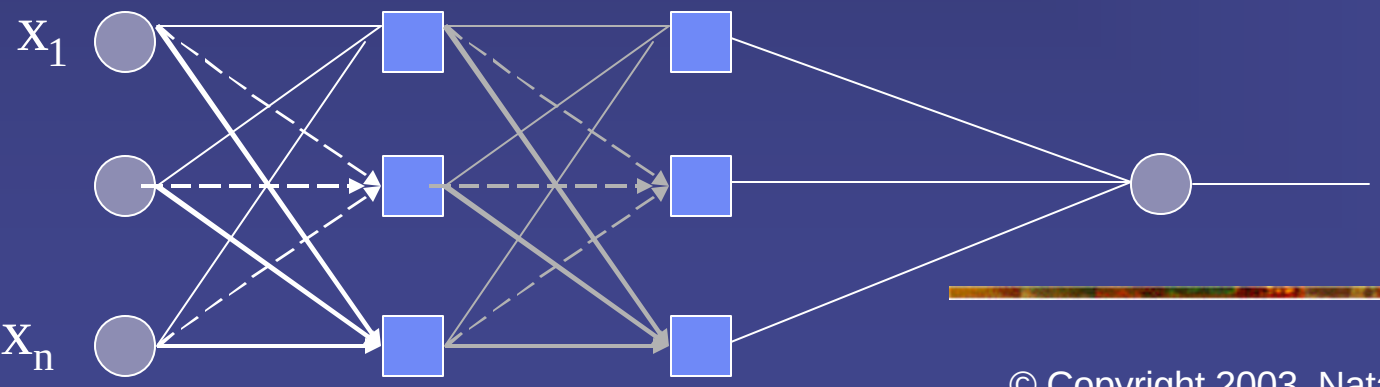
Issues in Learning by Backpropagation

- Excess weights lead to **overfitting**, which may be prevented by
- *early stopping*
- *network pruning*
- *network growing*
- applying *regularization techniques*

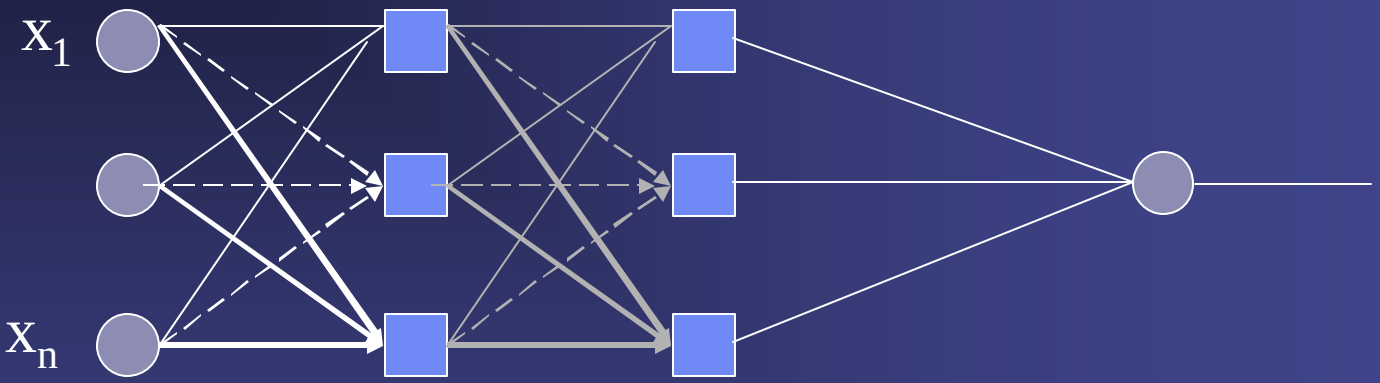
Multi-Layer Perceptron (MLP)

Multi-Layer Perceptron (MLP)

- MLP and the backpropagation algorithm which is used to train it
- MLP used to describe any general feedforward (no recurrent connections) network
- Nets with units arranged in layers



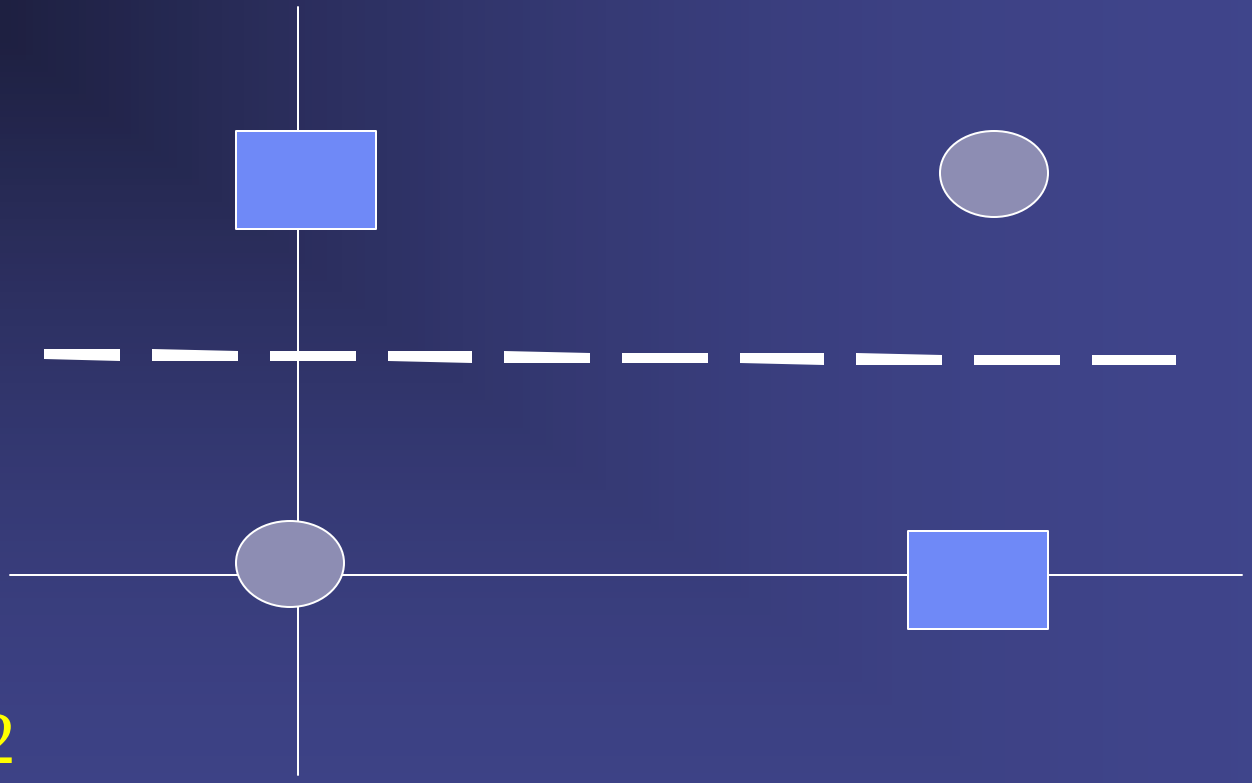
What do the extra layers gain us?



What would you call this network?

- 4 layer (no. of layers of neurons)?
- 3 layer (no. of layers of adaptive weights)?
- Something else?

XOR problem
Perceptron does not work here



Exclusive OR problem

0+0=0

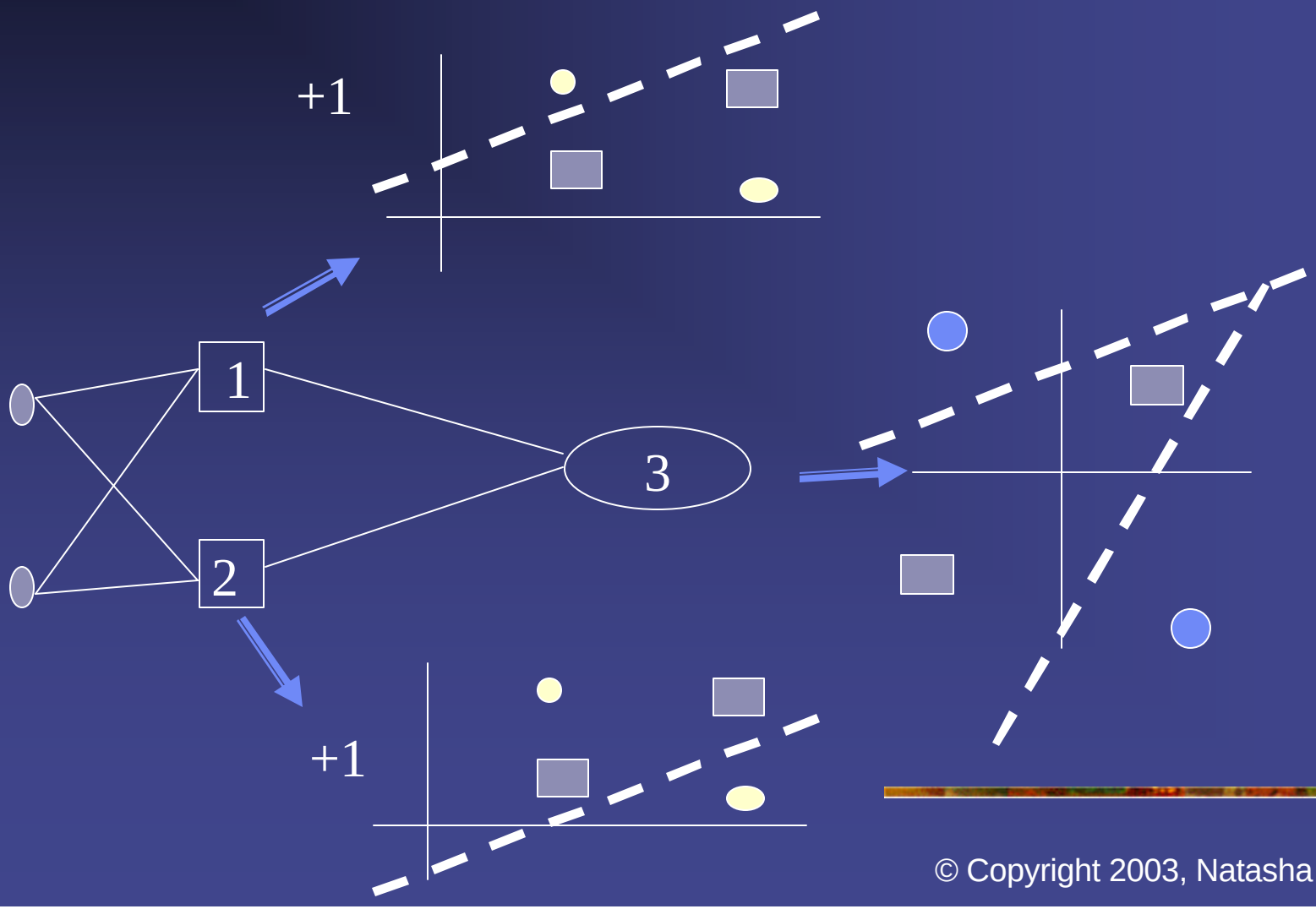
1+1=2=0 mod 2

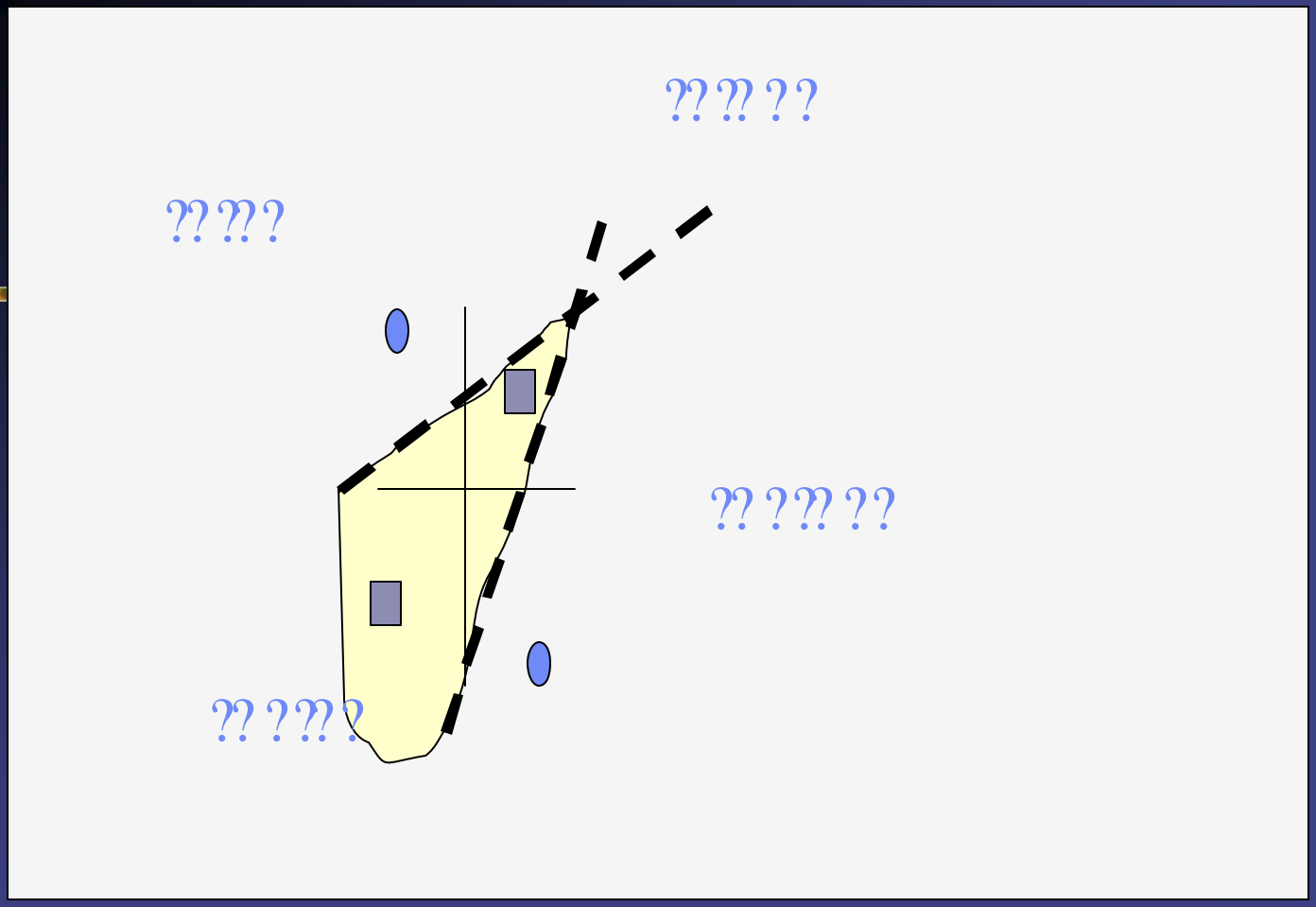
1+0=1

0+1=1

Single layer generates a linear decision boundary

XOR Solution

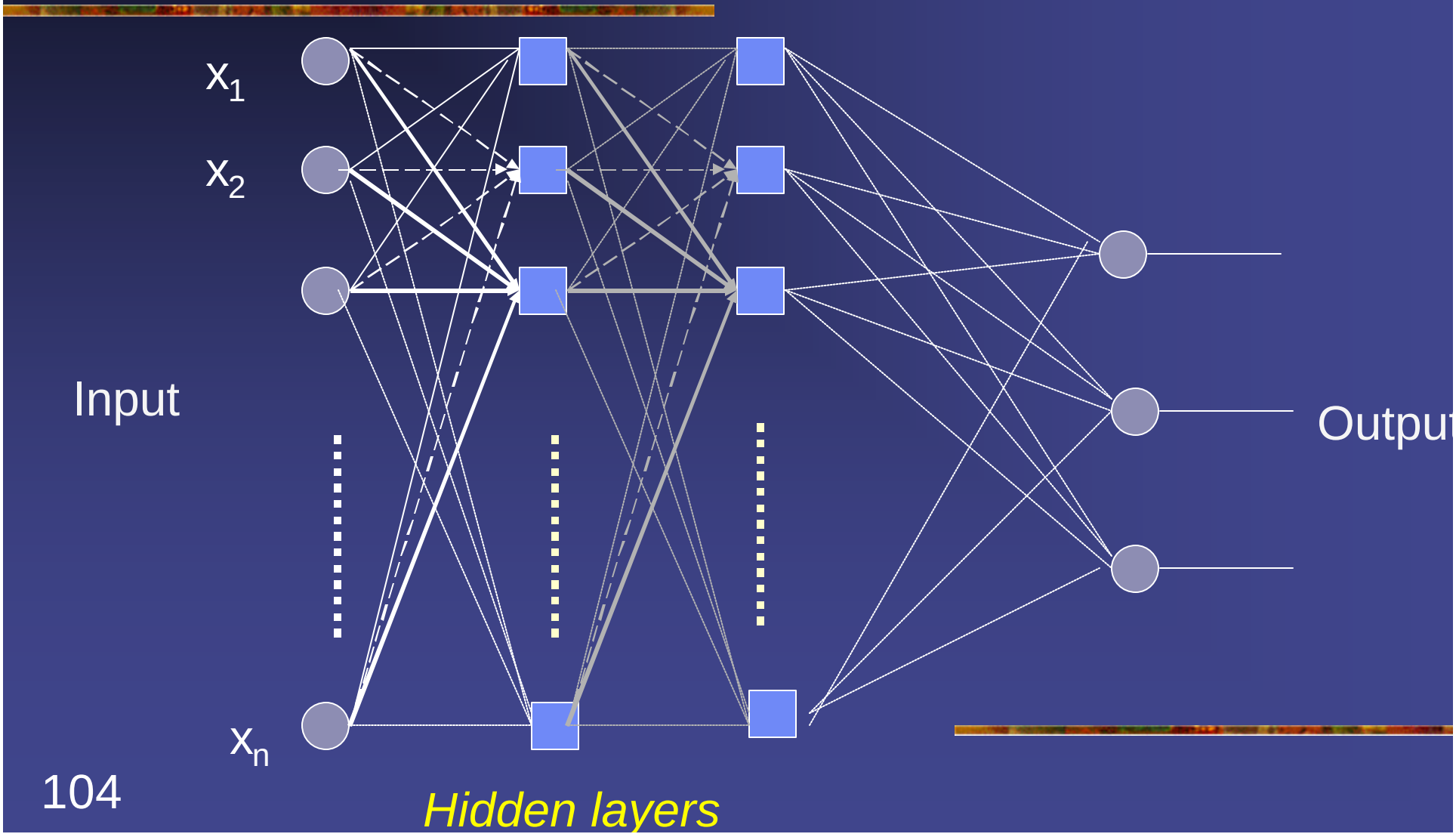




This is a linearly separable problem!

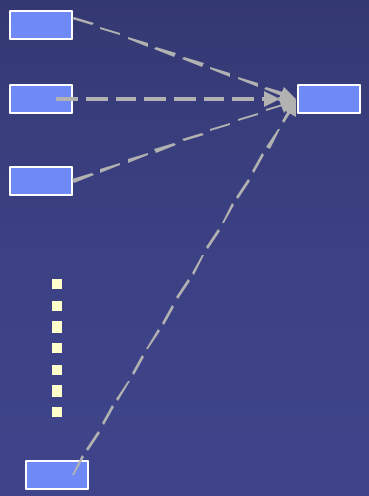
Since for 4 points *{ (-1,1), (-1,-1), (1,1),(1,-1) }* it is always linearly separable if we want to have three points in a class

Multi-layer networks



Properties of architecture

- No connections within a layer

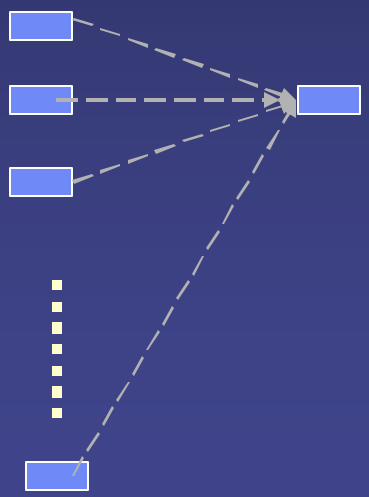


Each unit is a perceptron

$$y_i = f\left(\sum_{j=1}^{m} w_{ij} x_j - b_i\right)$$

Properties of architecture

- No connections within a layer
- No direct connections between input and output layers
-

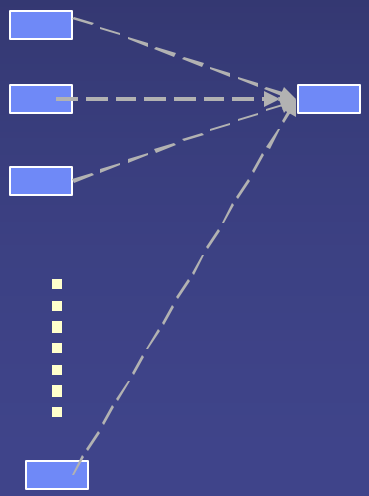


Each unit is a perceptron

$$y_i = f\left(\sum_{j=1}^{m} w_{ij} x_j - b_i\right)$$

Properties of architecture

- No connections within a layer
- No direct connections between input and output layers
- Fully connected between layers
-

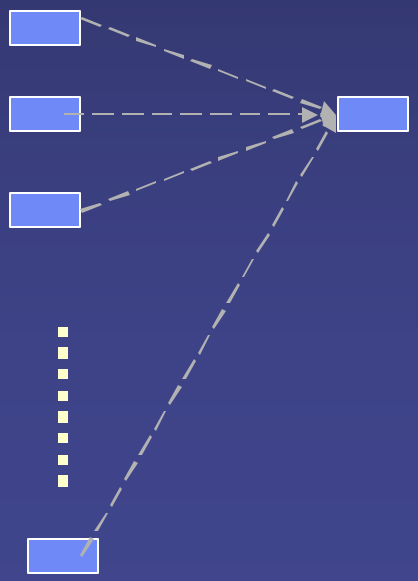


Each unit is a perceptron

$$y_i = f\left(\sum_{j=1}^{m} w_{ij} x_j - b_i\right)$$

Properties of architecture

- No connections within a layer
- No direct connections between input and output layers
- Fully connected between layers
- Often more than 3 layers
- Number of output units need not equal number of input units
- Number of hidden units per layer can be more or less than input or output units

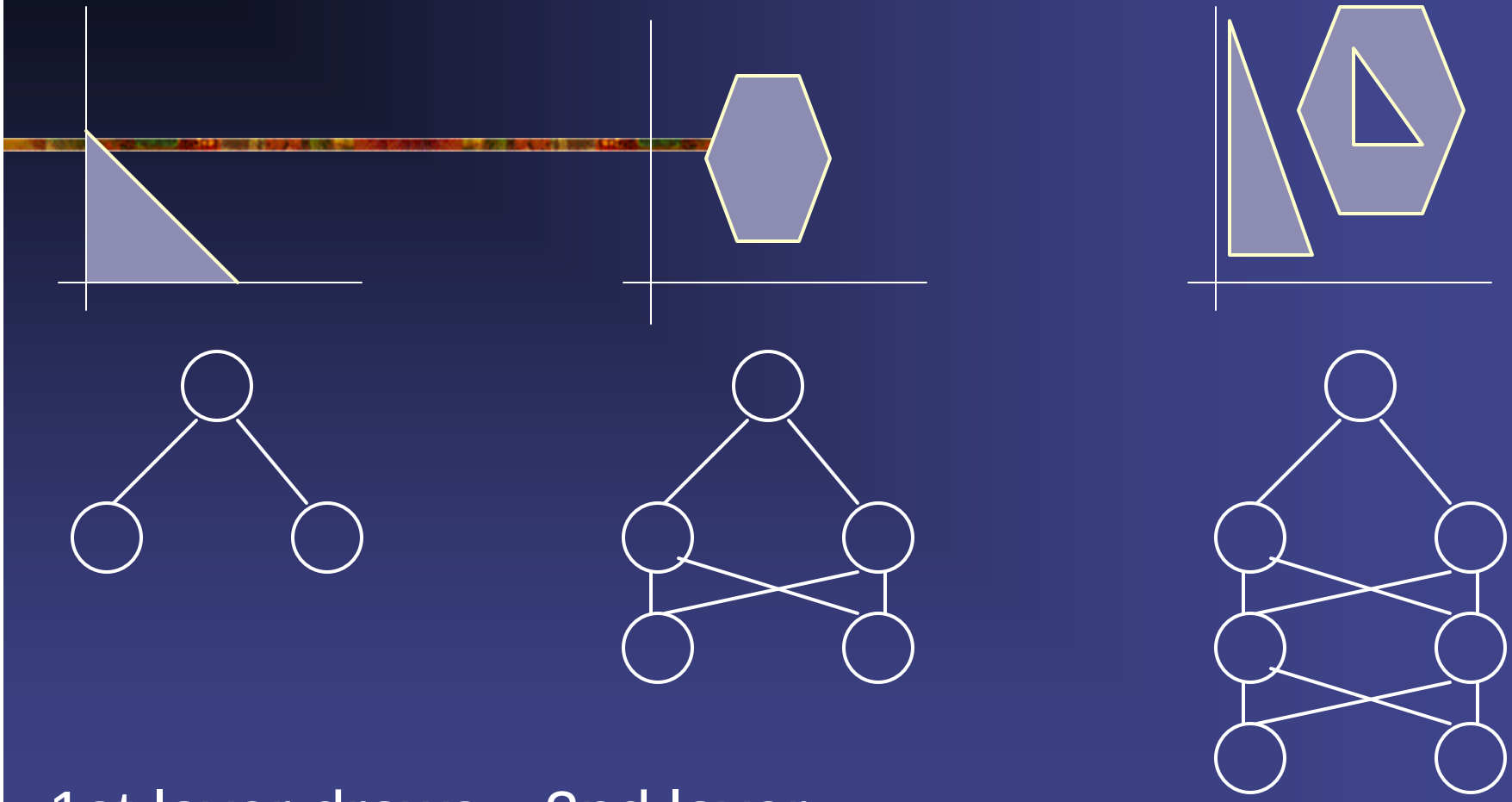


Each unit is a perceptron

$$y_i = f\left(\sum_{j=1}^{m} w_{ij} x_j - b_i\right)$$

Include bias as an extra weight

What do each of the layers do?



1st layer draws linear boundaries

2nd layer combines the boundaries

3rd layer can generate arbitrarily complex boundaries

What do each of the layers do?

- Can also view 2nd layer as using local knowledge while 3rd layer does global
- With sigmoidal activation functions theory says that a 3 layer net can approximate any function to arbitrary accuracy
  - property of Universal Approximation
- Not practically useful
  - need arbitrarily large number of units
  - more of an existence proof

What do each of the layers do?

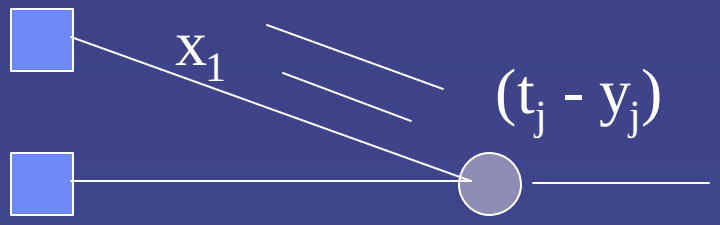
- Can also view 2nd layer as using local knowledge while 3rd layer does global
- With sigmoidal activation functions theory says that a 3 layer net can approximate any function to arbitrary accuracy
  - property of Universal Approximation
- Not practically useful
  - need arbitrarily large number of units
  - more of an existence proof

Backpropagation

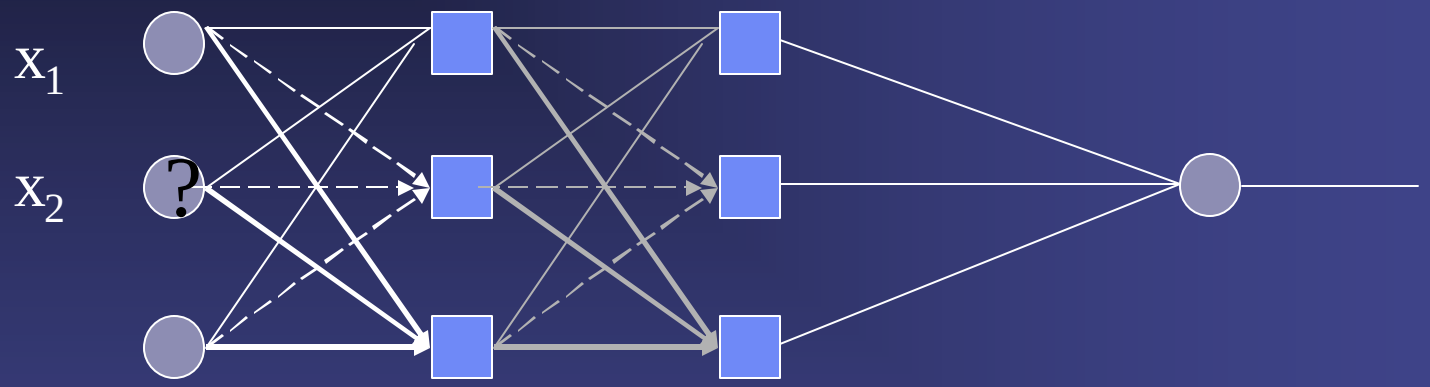
Gradient Decent Method &
Multilayer Networks

Updating Errors
In a single perceptron

- Perceptron/single layer nets used gradient descent on the error function to find the correct weights
- $\Delta w_{ji} = (t_j - y_j) x_i$
- Errors/updates are local to the node
- The change in the weight from node i to output j (w_{ji}) is controlled by the input that travels along the connection and the error signal from output j



Updating Errors
In a multilayer network



- *With more layers how are the weights for the first 2 layers found when the error is computed for layer 3 only?*
- *There is no direct error signal for the first layers*

Credit assignment problem

- Problem of assigning ‘credit’ or ‘blame’ to individual units involved in forming overall response of a learning system - hidden units
- How to decide which weights should be altered, by how much and in which direction
- Analogous to deciding how much a weight in the early layer contributes to the output and thus the error
- We therefore want to find out how weight w_{ij} affects the error
- We want

$$\frac{?\,E\,(t)}{?\,w_{ij}\,(t)}$$

Backpropagation (BP) learning algorithm

Two Phases:

Forward pass phase: computes ‘functional signal’, feedforward propagation of input pattern signals through network

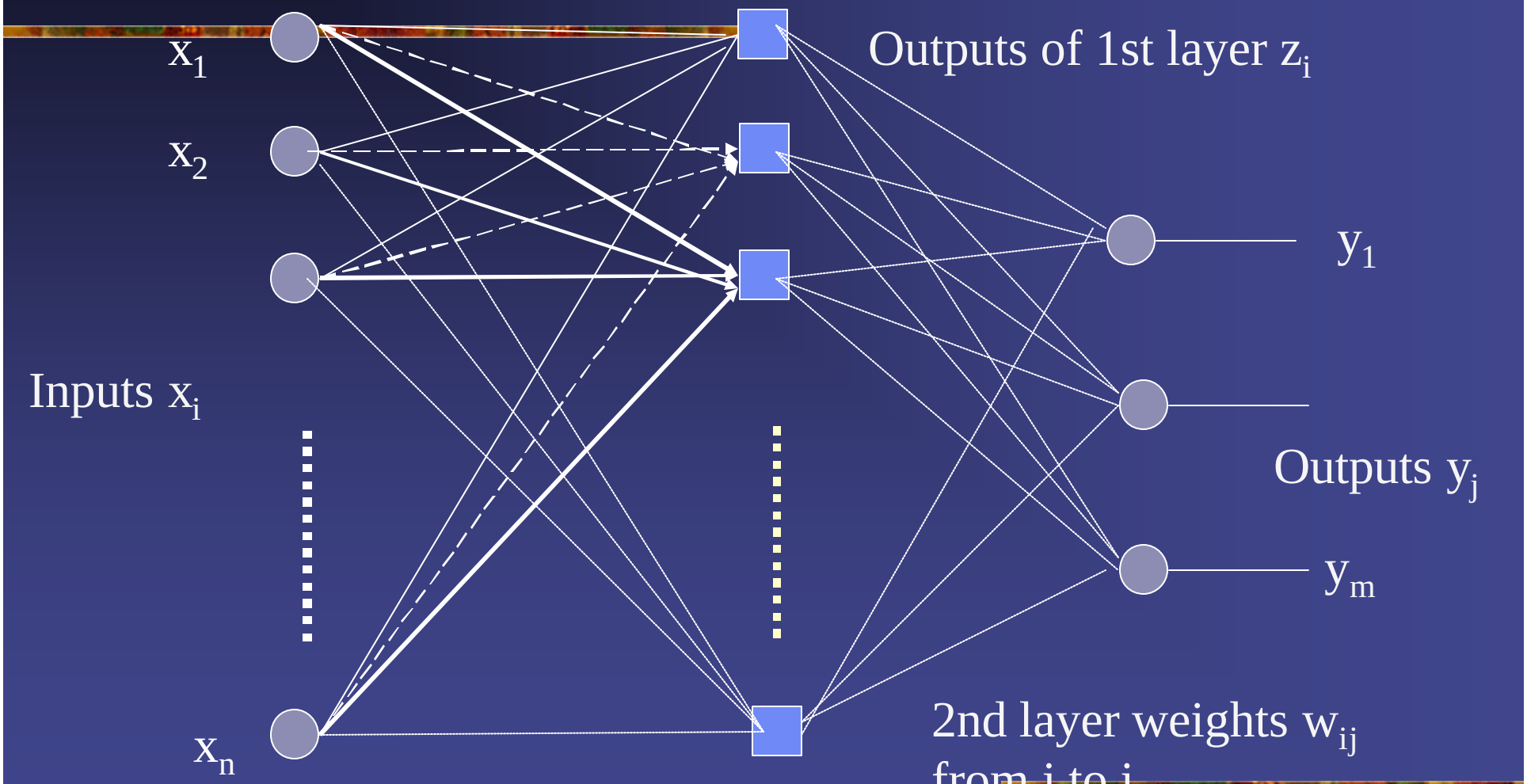
Backpropagation (BP) learning algorithm

Two Phases:

Forward pass phase: computes 'functional signal', feedforward propagation of input pattern signals through network

Backward pass phase: computes 'error signal', *propagates* the error *backwards* through network starting at output units

One-hidden Layer Network



Notation

$$z_i(t) = g(\sum_j v_{ij}(t)\, x_j(t)) \quad \text{at time t}$$
$$= g(u_i(t))$$

$$y_i(t) = g(\sum_j w_{ij}(t)\, z_j(t)) \quad \text{at time t}$$
$$= g(a_i(t))$$

a & u known as activation

g the activation function

Weights are fixed during forward and backward pass at time *t*

Forward pass

1. Compute hidden unit values

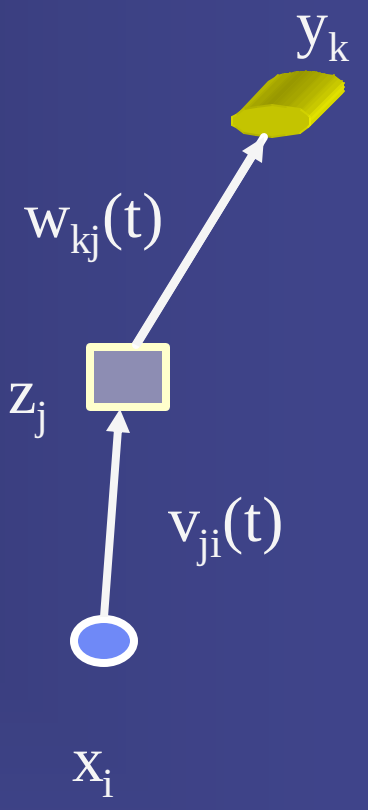
$$u_j(t) = \sum_i v_{ji}(t)\, x_i(t)$$

$$z_j = g(u_j(t))$$

2. Compute output unit values

$$a_k(t) = \sum_j w_{kj}(t)\, z_j$$

$$y_k = g(a_k(t))$$



Backward Pass

- **Use a sum of squares error measure**
- **For each training pattern**

$$E(t) ? \frac{1}{2} ? \sum_{k ? 1} (d_k(t) ? y_k(t))^2$$

where d_k is the target value for dimension k

- **How to modify weights in order to decrease E?**
- **Use gradient descent**

$$w_{ij}(t ? 1) ? w_{ij}(t) ? ? \frac{? E(t)}{? w_{ij}(t)}$$

both for hidden units and output units

Partial derivative rewritten as product of two terms using chain rule

$$\frac{?E(t)}{?w_{ij}(t)} = \frac{?E(t)}{?a_i(t)} \cdot \frac{?a_i(t)}{?w_{ij}(t)}$$

How error for pattern changes as function of change in network input to unit *j*

How net input to unit j changes as a function of change in weight *w*

Using chain rule

Second Term

$$\frac{?u_i(t)}{?v_{ij}(t)} ? \; x_j(t) \qquad \frac{?a_i(t)}{?w_{ij}(t)} ? \; z_j(t)$$

Using chain rule

For Output Units

First Term

$$?_i(t) ? ? \frac{?E(t)}{?u_i(t)}, ?_i(t) ? ? \frac{?E(t)}{?a_i(t)}$$

Evaluate by using the chain rule

$$?_i(t) ? ? \frac{?E(t)}{?a_i(t)} ? ? g'(a_i(t)) \frac{?E(t)}{?y_i(t)}$$

$$?_i(t) ? ? g'(a_i(t))(d_i(t) ? y_i(t))$$

Using chain rule

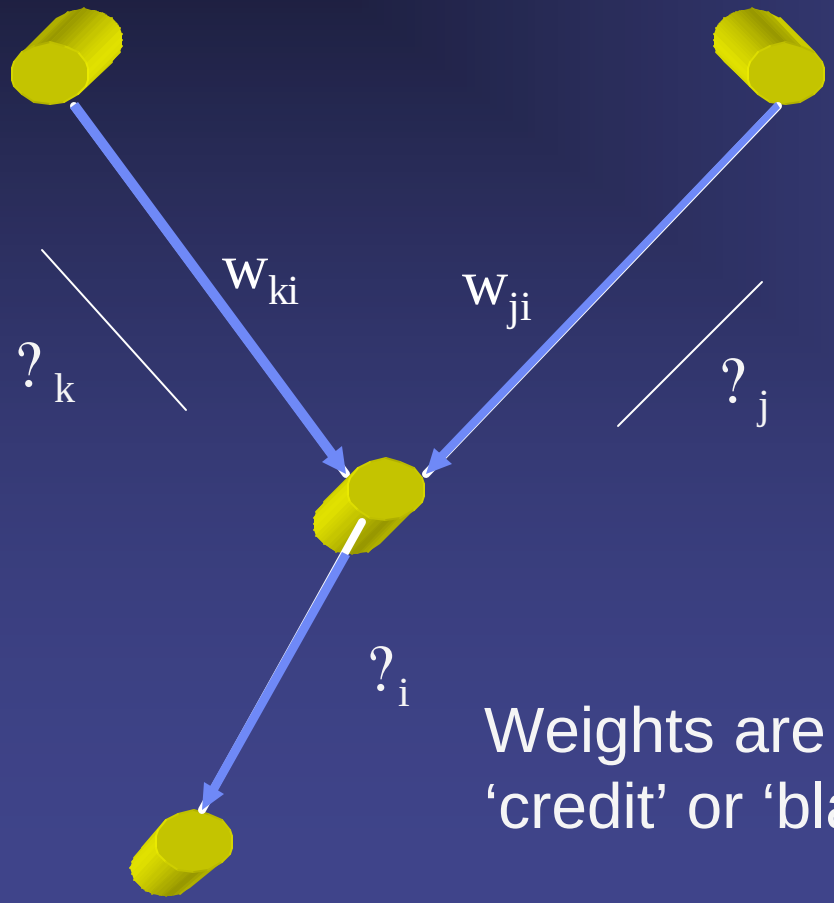
For Hidden Units

Use the chain rule

$$?_i(t) \; ? \; ? \; \frac{?E(t)}{?u_i(t)} \; ? \; \sum_j \frac{?E(t)}{?a_j(t)} \frac{?a_j(t)}{?u_i(t)}$$

$$?_i(t) \; ? \; g'(u_i(t)) \sum_j w_{ji} ?_j$$

Backward Pass



$$?_i = g'(a_i)\ ?_j\ w_{ji}\ ?_j$$

Weights are providing degree of ‘credit’ or ‘blame’ to hidden units

Achieving Gradient Decent

$$? \frac{? E(t)}{? v_{ij}(t)} ? ? _i(t) x_j(t)$$

$$? \frac{? E(t)}{? w_{ij}(t)} ? ? _i(t) z_j(t)$$

So to achieve gradient descent in E should change weights by

$$v_{ij}(t+1)-v_{ij}(t) = h d_i(t) x_j(n)$$

$$w_{ij}(t+1)-w_{ij}(t) = h D_i(t) z_j(t)$$

h is the learning rate parameter ($0 < h <= 1$)

Updating the weights

Wight updates are local

$$v_{ij}(t ? 1) ? v_{ij}(t) ? ? ? _i(t)x_j(t)$$

$$w_{ij}(t ? 1) ? w_{ij}(t) ? ? ? _i(t)z_j(t)$$

Output Units

$$w_{ij}(t ? 1) ? w_{ij}(t) ? ? ? _i(t)z_j(t)$$

$$? ? (d_i(t) ? y_i(t))\, g'(a_i(t))\, z_j(t)$$

Hidden Units

$$v_{ij}(t ? 1) ? v_{ij}(t) ? ?? _i(t)x_j(t)$$

$$? ? g'(u_i(t))\, x_j(t) ? \sum_k ? _k(t)w_{ki}$$

Backpropagation at Work

1. Apply an input vector and calculate all activations, a and u
2. Evaluate D_k for all output units

$$? _i(t) ? (d_i(t) ? y_i(t)) g'(a_i(t))$$

3. Backpropagate D_ks to get error terms d for hidden layers

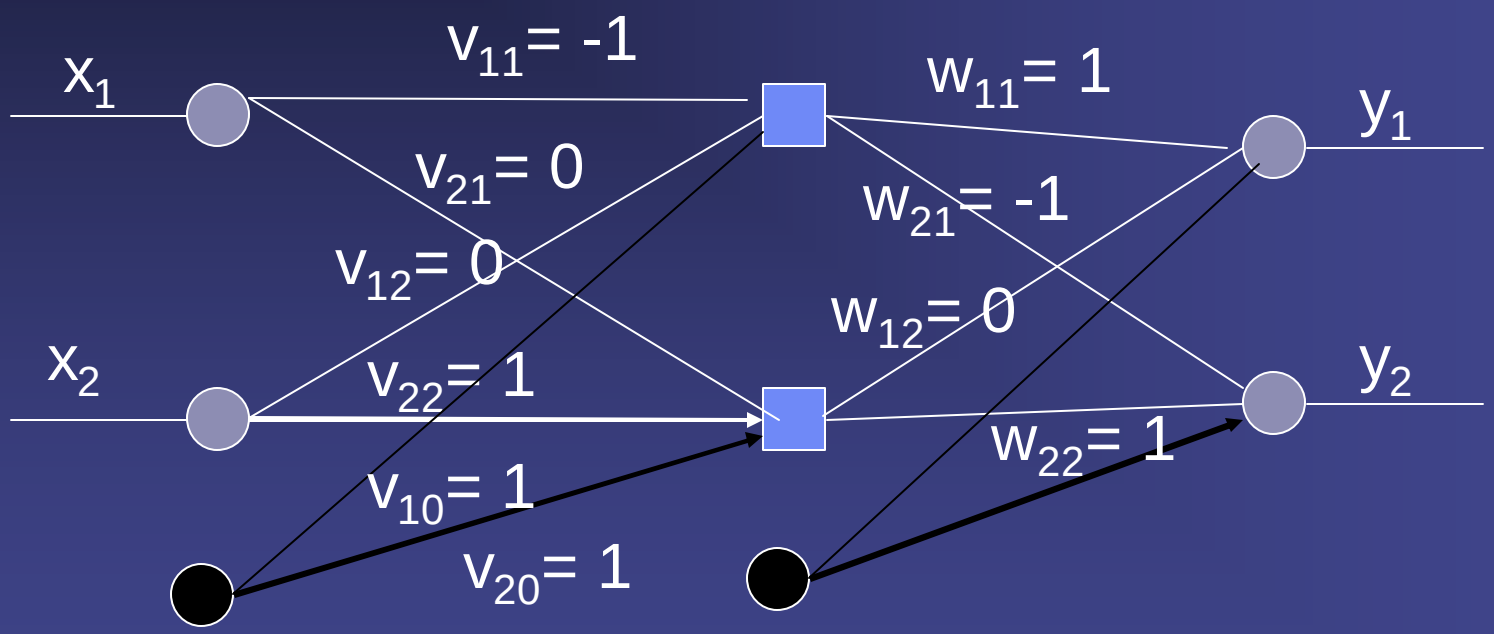
$$? _i(t) ? g'(u_i(t)) ? _k ? _k(t) w_{ki}$$

4. Evaluate changes

$$v_{ij}(t ? 1) ? v_{ij}(t) ? ? ? _i(t) x_j(t)$$

$$w_{ij}(t ? 1) ? w_{ij}(t) ? ? ? _i(t) z_j(t)$$

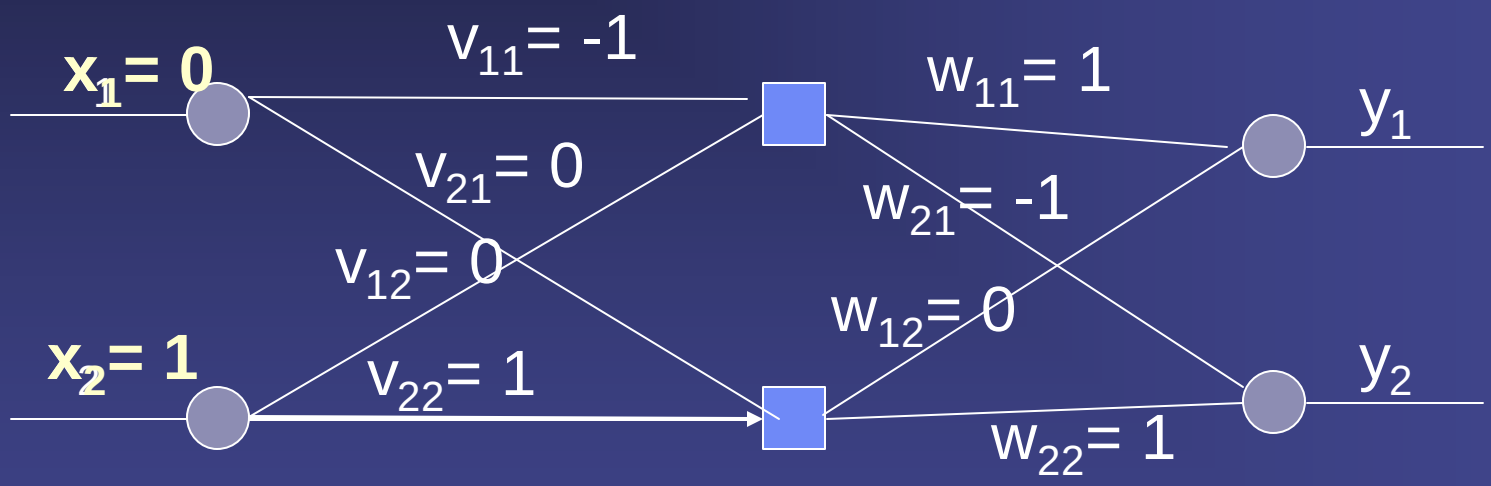
Example



Example

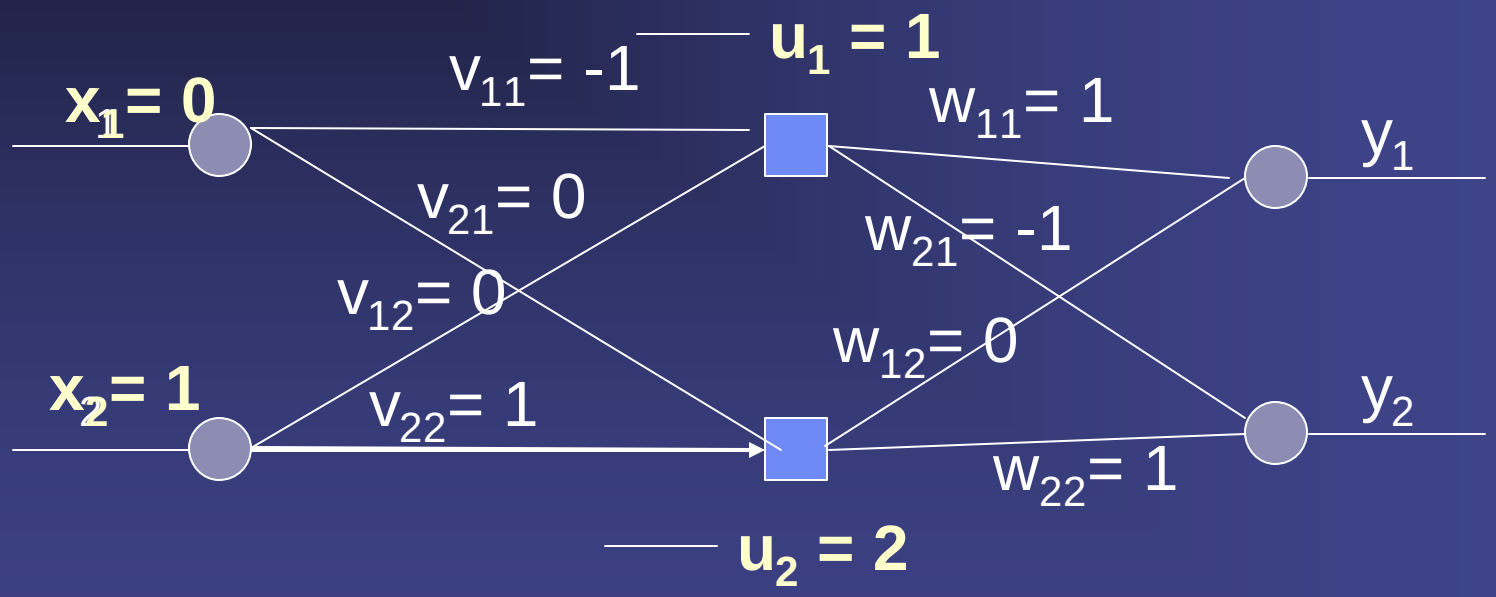
- Using identity activation function
  - g(a) = a
- All biases set to 1
  - Will not draw them in the example but will use it's value in the calculations
- Learning rate ? = 0.1
- Start with input [0 1]
- Learn the target [1 0]

Example



Forward pass

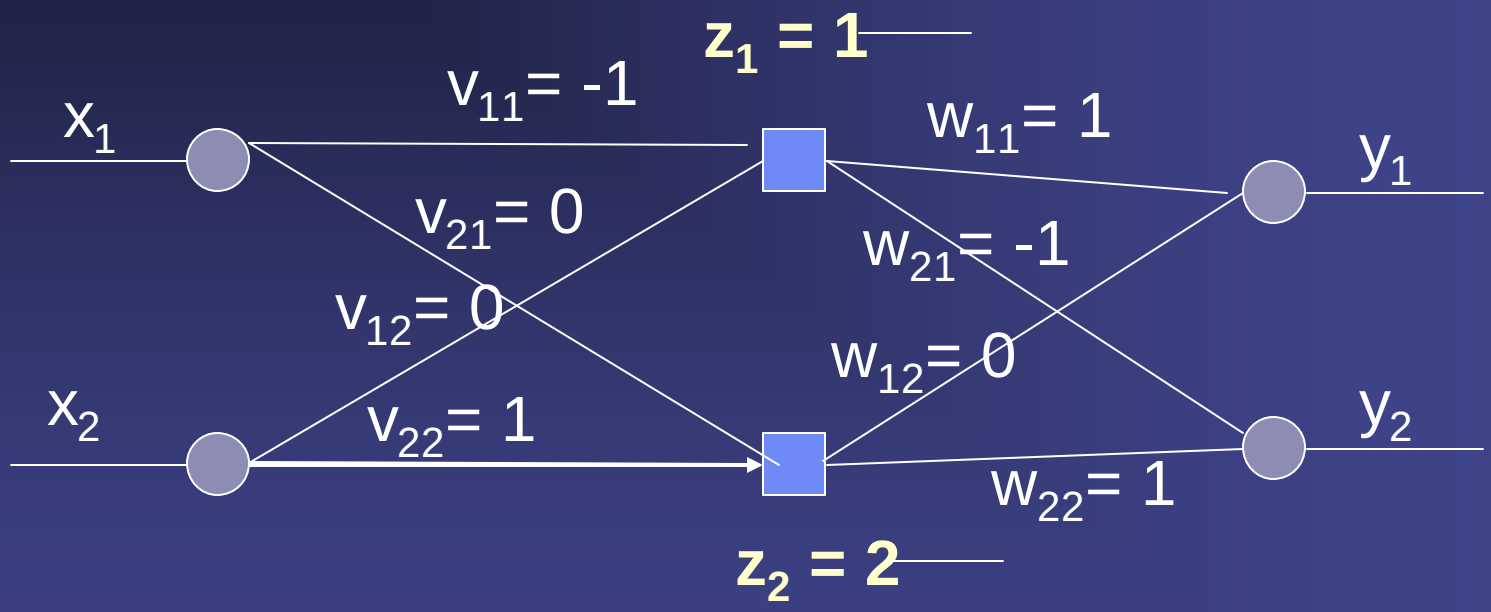
Calculating 1st layer activation values



$u_1 = -1x0 + 0x1 + 1 = 1$

$u_2 = 0x0 + 1x1 + 1 = 2$

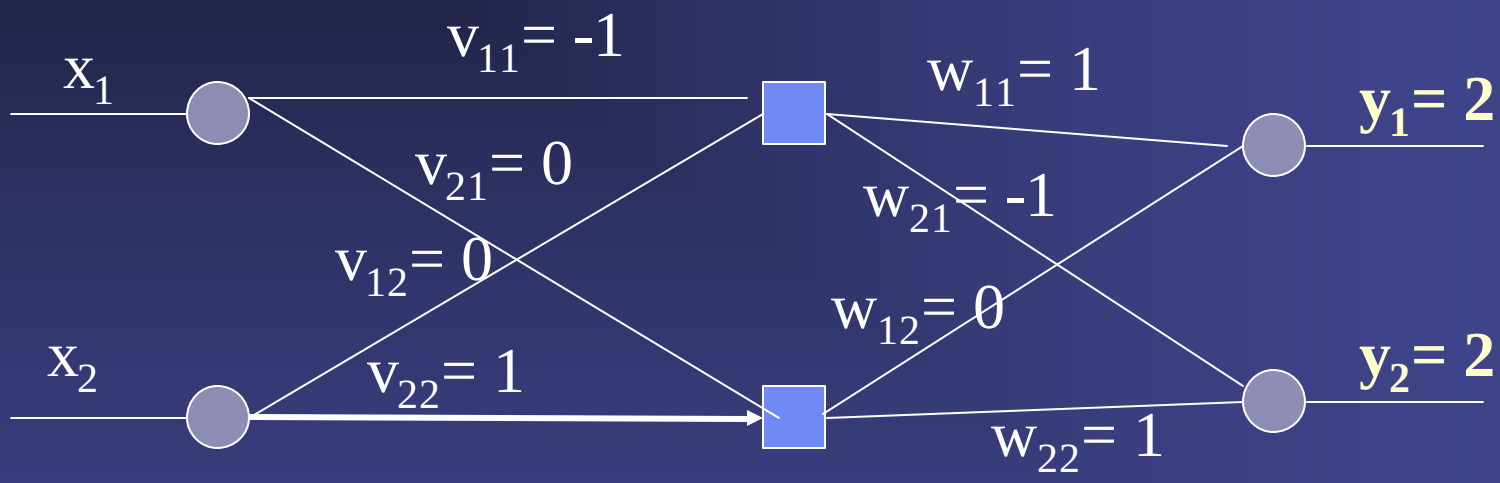
Example
Calculating first layer outputs



$z_1 = g(u_1) = 1$

$z_2 = g(u_2) = 2$

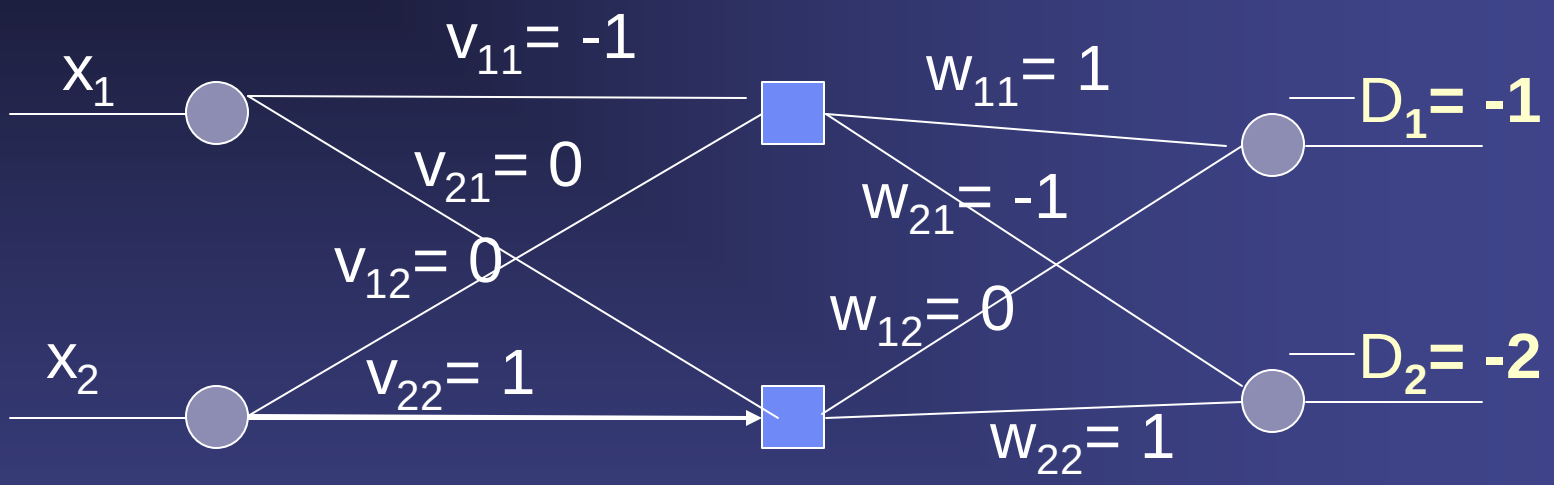
Example
Calculating 2nd layer outputs



$y_1 = a_1 = 1x1 + 0x2 + 1 = 2$

$y_2 = a_2 = -1x1 + 1x2 + 1 = 2$

First Backward Pass



$$w_{ij}(t+1) = w_{ij}(t) + \eta\,\delta_i(t)\,z_j(t)$$

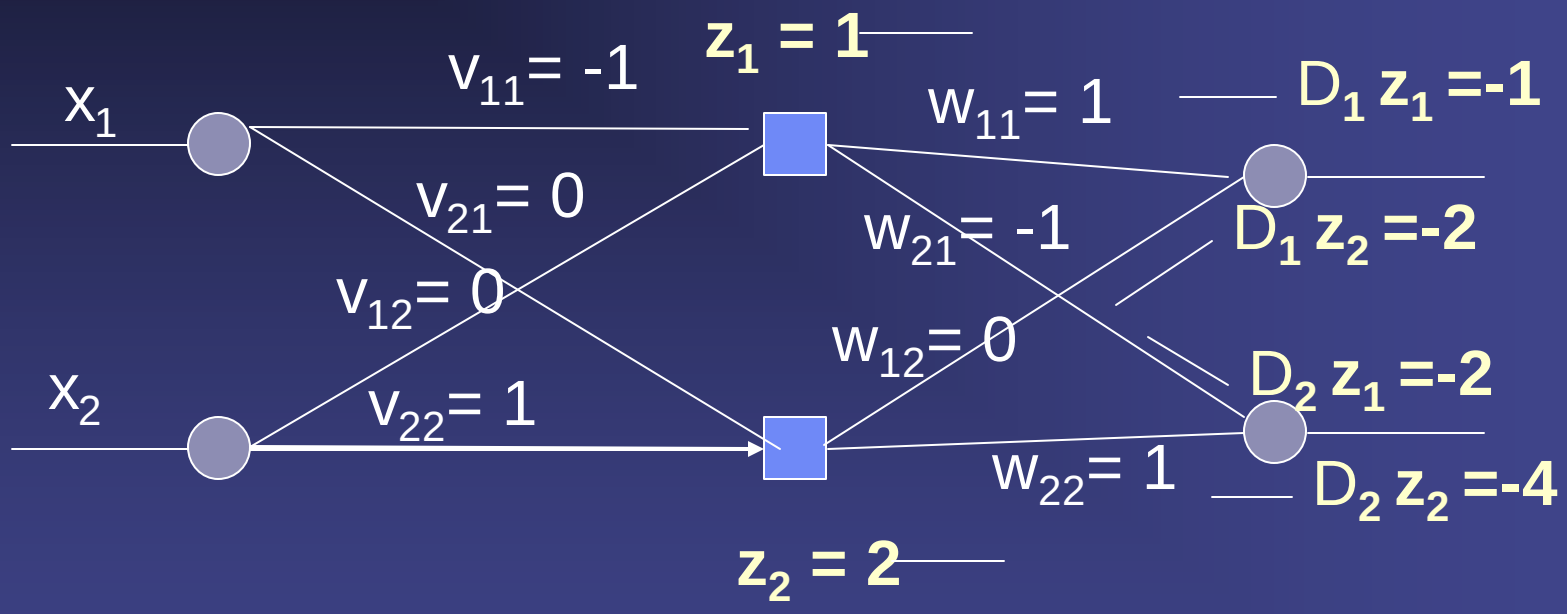
$$\delta_i(t) = (d_i(t) - y_i(t))\,g'(a_i(t))\,z_j(t)$$

Target =[1, 0] so $d_1 = 1$ and $d_2 = 0$

$D_1 = (d_1 - y_1) = 1 - 2 = -1$

$D_2 = (d_2 - y_2) = 0 - 2 = -2$

Calculating weight changes for 1st layer

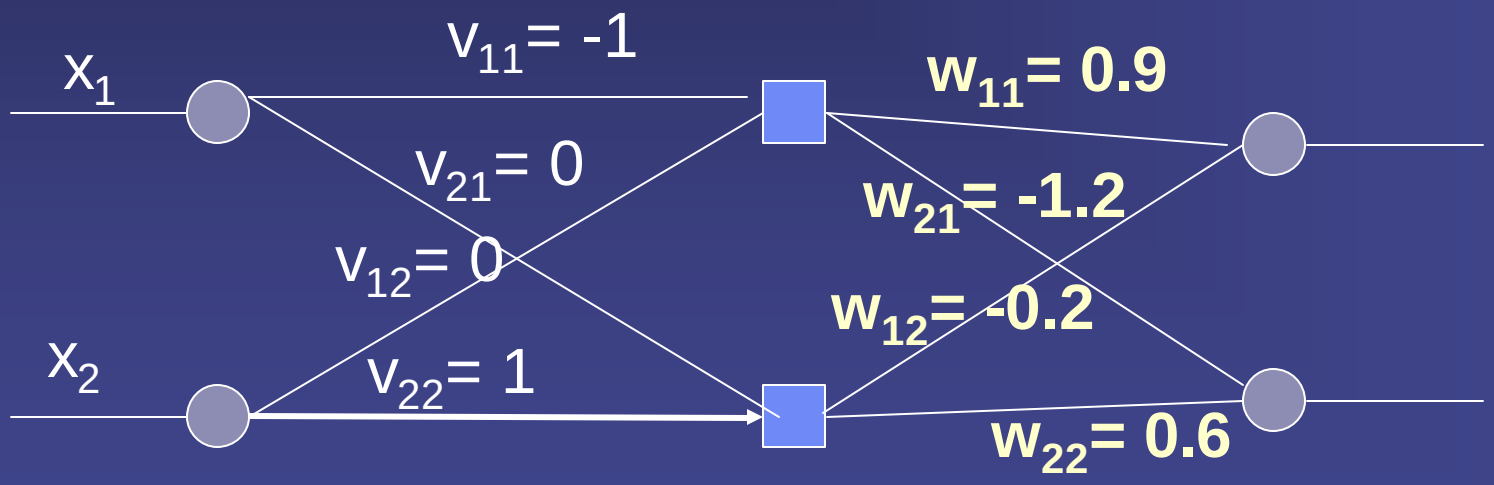


$$w_{ij}(t ? 1) ? w_{ij}(t) ? ? ? {}_{i}(t) z_{j}(t)$$

Calculating weight changes for 1st layer

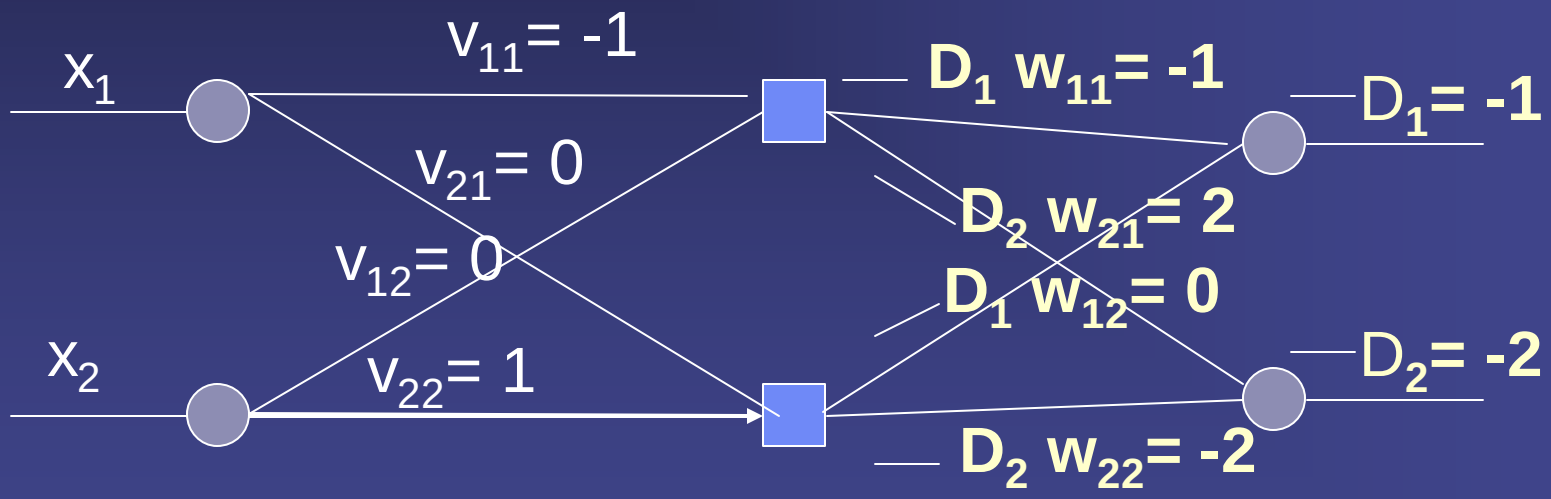
Weight changes as

$$w_{ij}(t ? 1) ? w_{ij}(t) ? ? ? {}_{i}(t) z_{j}(t)$$



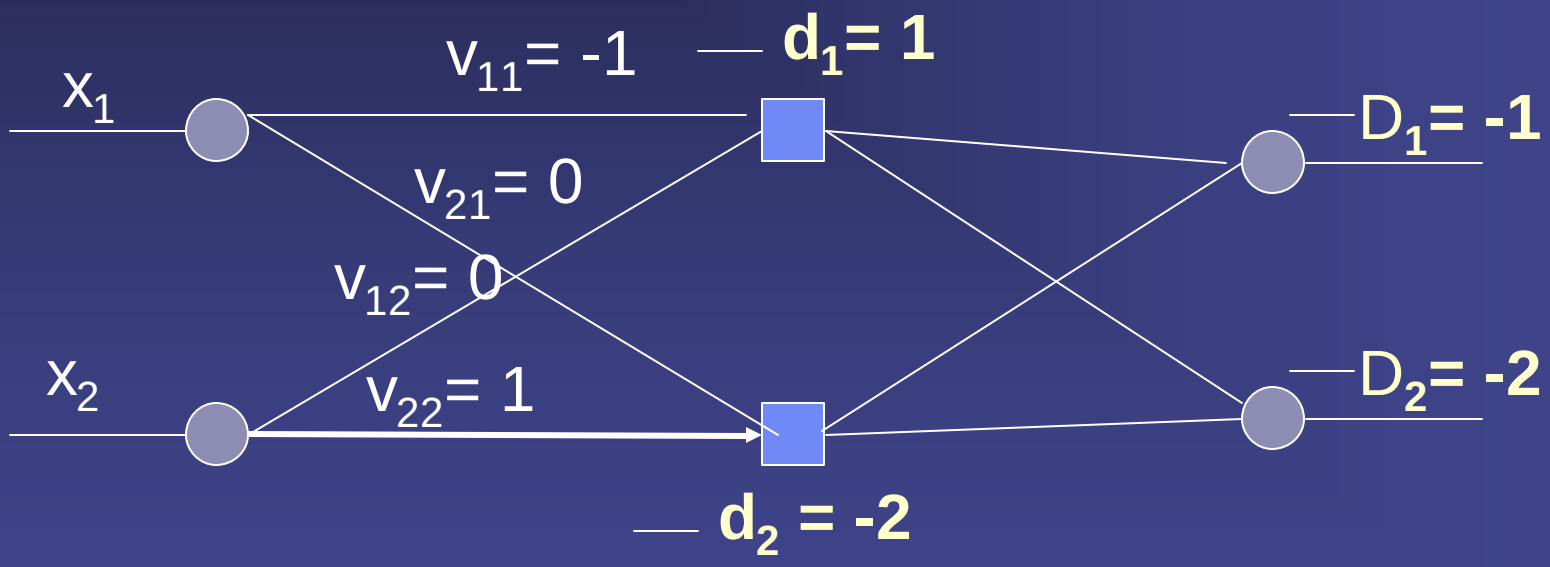
Calculating ?'s

$$? _i (t) \; ? \; g'(u_i(t)) \sum_k ? _k (t) w_{ki}$$



Propagating ?'s back

$$? _i (t) \; ? \; g'(u_i(t)) \sum_k ? _k (t) w_{ki}$$

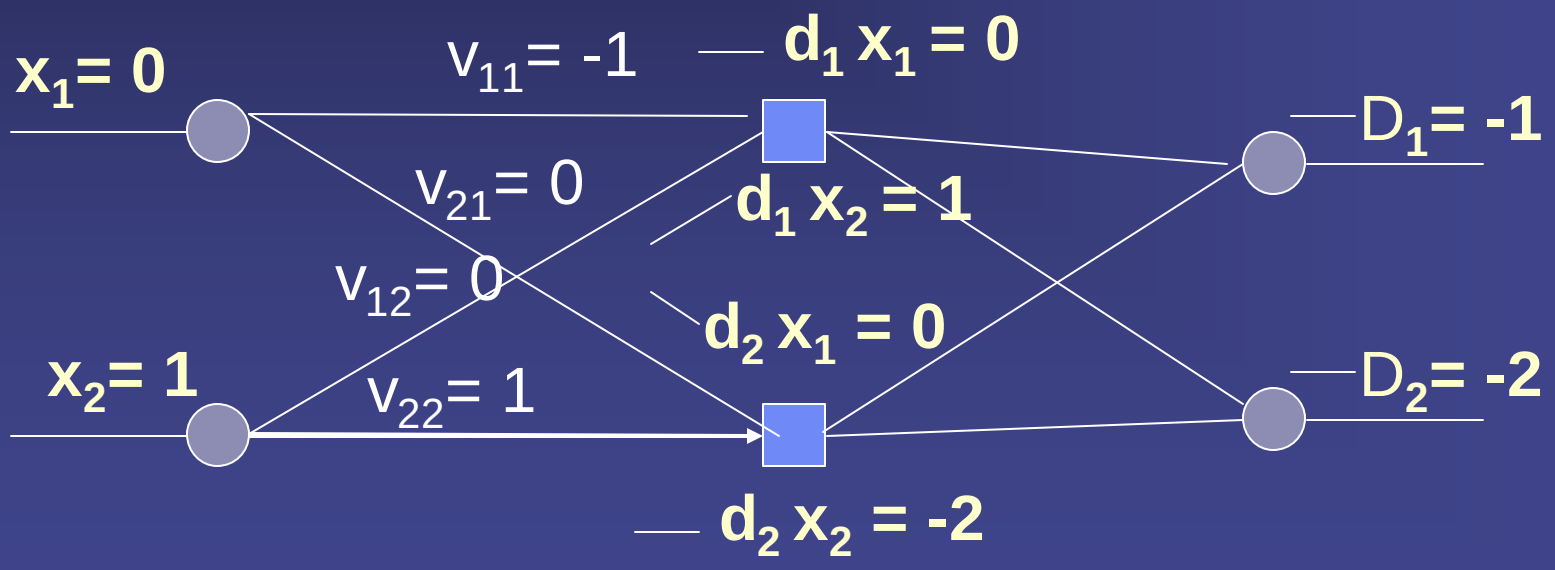


$d_1 = -1 + 2 = 1$

$d_2 = 0 - 2 = -2$

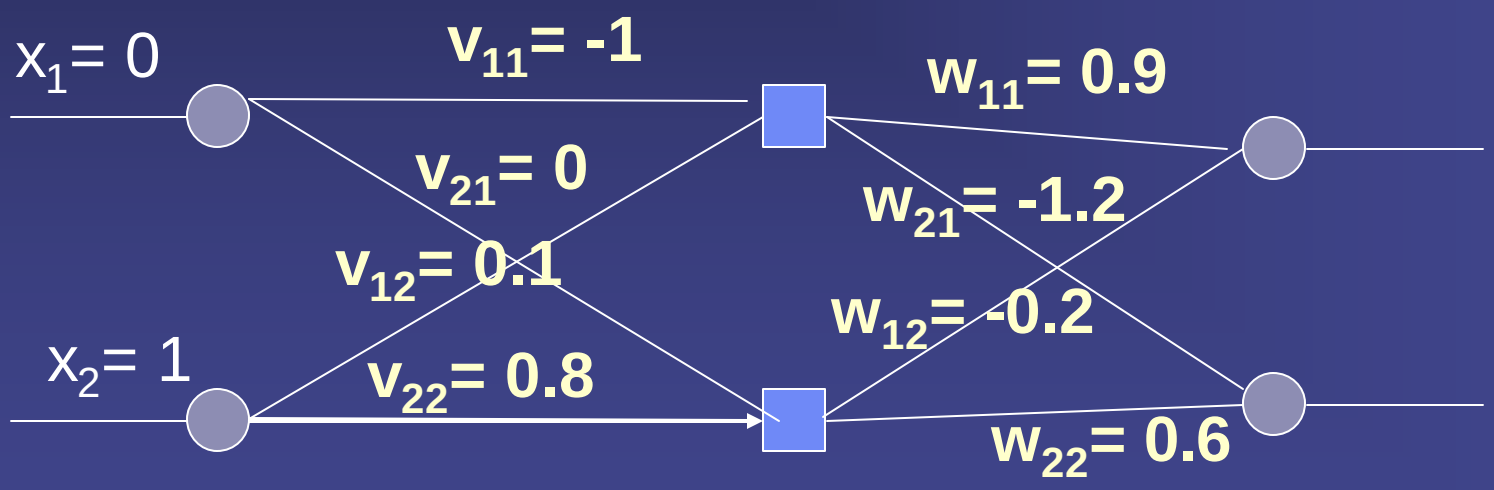
Multiplying ?'s by inputs

$$v_{ij}(t ? 1) ? v_{ij}(t) ? ?? _i(t) x_j(t)$$



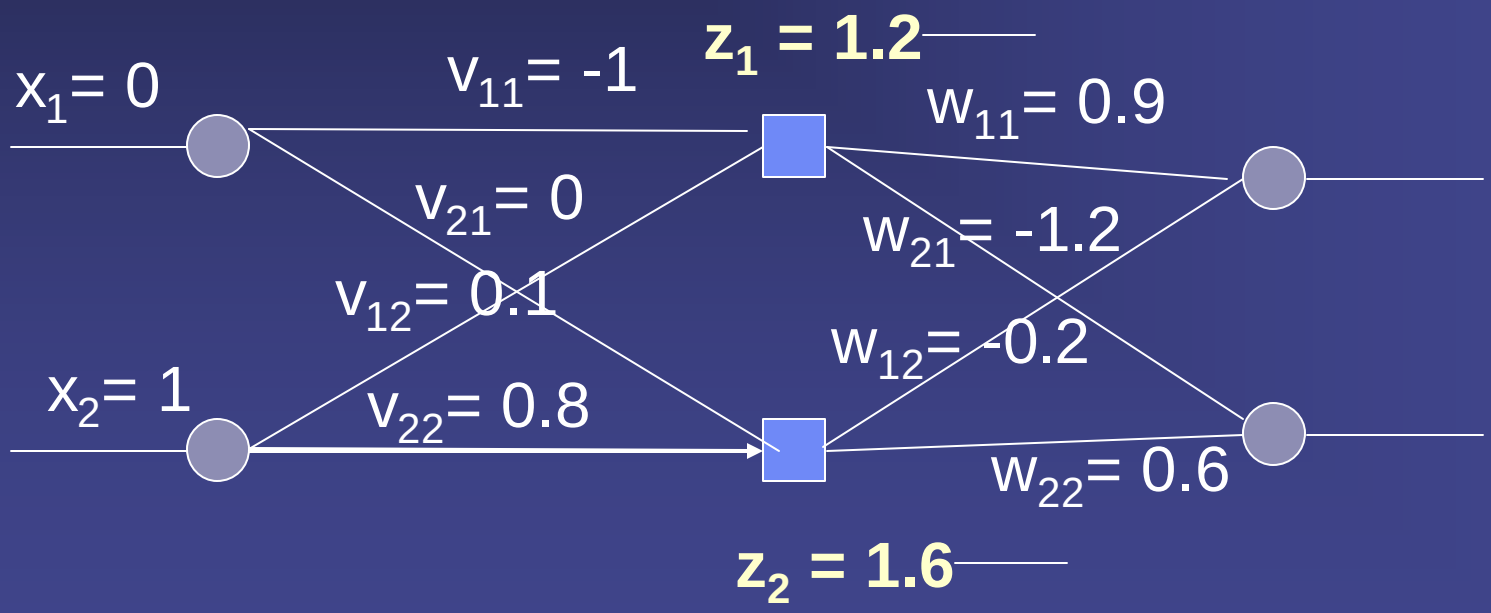
Changing the weights

$$v_{ij}(t ? 1) ? v_{ij}(t) ? ?? {}_{i}(t) x_{j}(t)$$



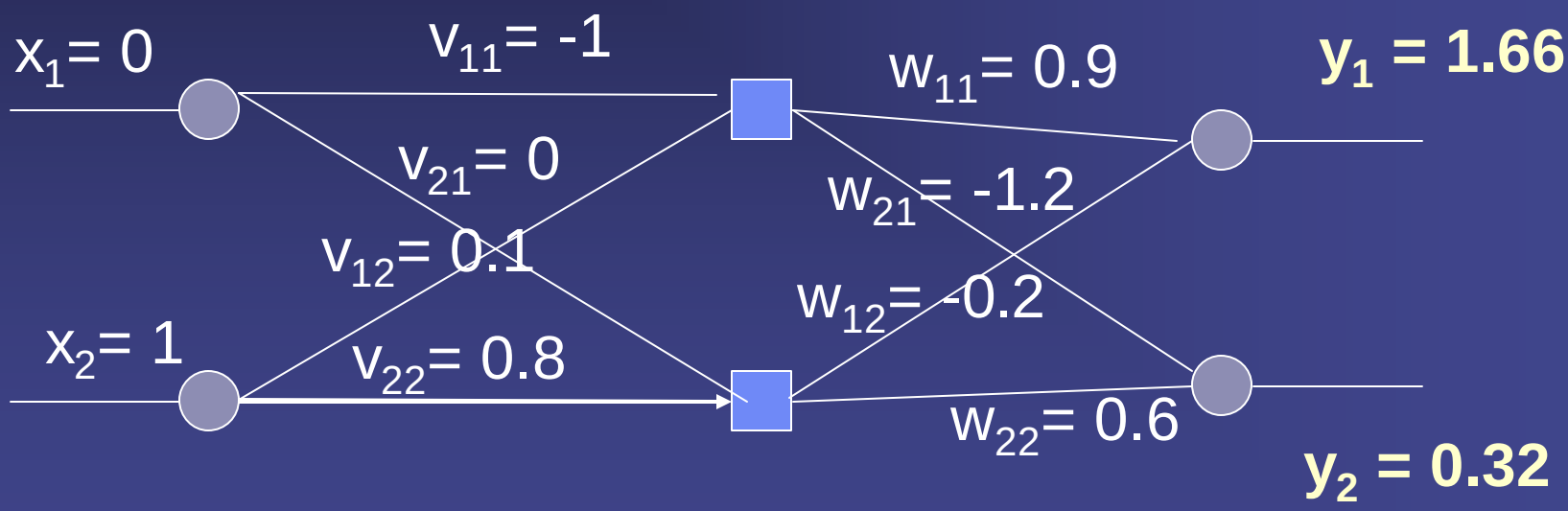
Another Forward Pass

$$v_{ij}(t ? 1) ? v_{ij}(t) ? ?? _{i}(t) x_{j}(t)$$



Another Forward Pass

$$v_{ij}(t+1) = v_{ij}(t) + \eta \delta_i(t) x_j(t)$$



Activation Functions

- How does the activation function affect the changes?

$$? _i(t) ? (d_i(t) ? y_i(t)) g'(a_i(t))$$

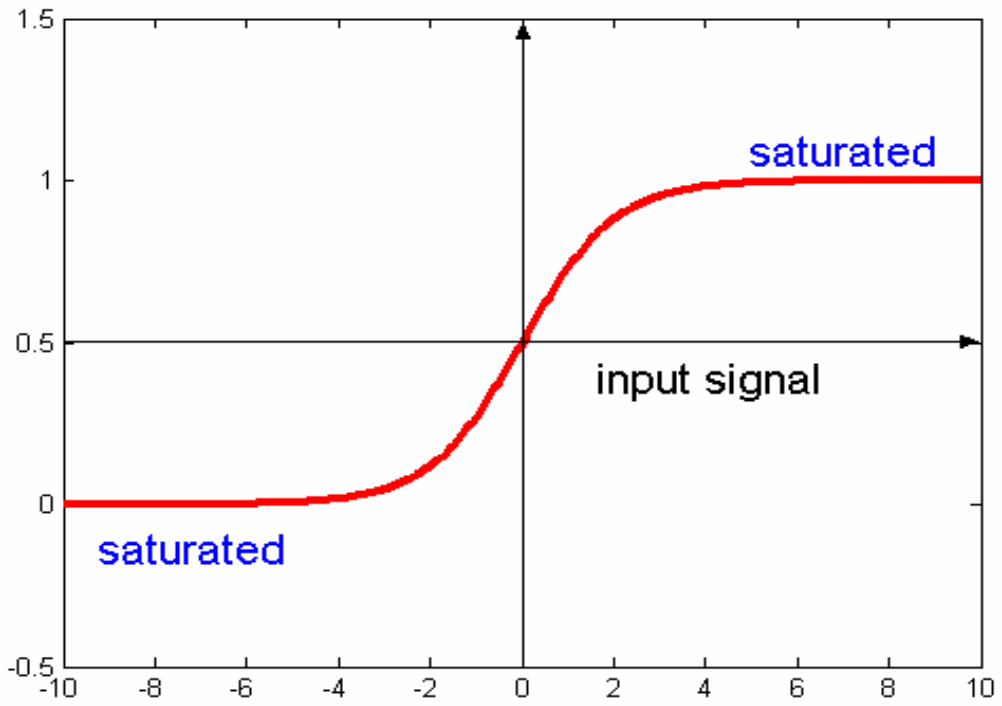
$$? _i(t) ? g'(u_i(t)) \sum_k ? _k(t) w_{ki}$$

Where $g'(a_i(t)) ? \frac{dg(a)}{da}$

- Need to compute the derivative of activation function g
- To find derivative the activation function must be smooth (differentiable)

Sigmoidal (logistic) function

$$g(a_i(t)) \; ? \; \frac{1}{1 \; ? \; \exp(\; ? \; k a_i(t))} \; ? \; \frac{1}{1 \; ? \; e^{\; ? \; k a_i(t)}}$$



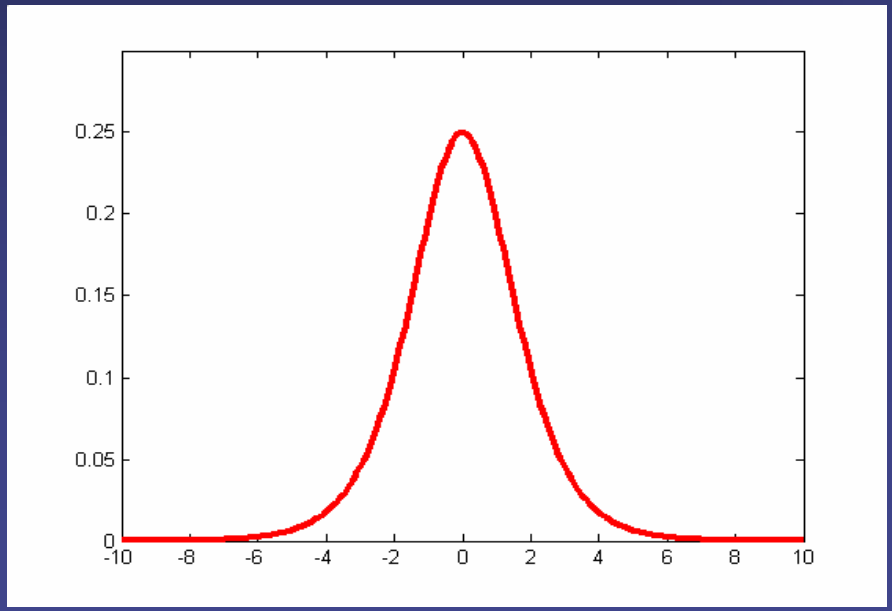
- Where k is a positive constant
- The sigmoidal function gives a value in range of 0 to 1
- Alternative is tanh(ka) which is same shape but in range –1 to 1

When net = 0, f = 0.5

Derivative of sigmoidal function

$$g'(a_i(t)) ? \frac{k \exp(? ka_i(t))}{[1 ? k \exp(? ka_i(t))]^2} ? kg(a_i(t))[1 ? g(a_i(t))]$$

since : $y_i(t) ? g(a_i(t))$ we have : $g'(a_i(t)) ? ky_i(t)(1 ? y_i(t))$



How does the activation function affect the changes?

• Degree of weight change is proportional to derivative of activation function

$$? _i(t) \; ? \; (d_i(t) \; ? \; y_i(t)) \, g'(a_i(t))$$

$$? _i(t) \; ? \; g'(u_i(t)) \sum_k ? _k(t) w_{ki}$$

•weight changes will be greatest when
  - •units receives mid-range functional signal
  - •and 0 (or very small) extremes

•**Saturating** a neuron (making the activation large) the weight can be forced to be static

Backpropagation learning algorithm

Set learning rate ?
Set initial weight values (incl. biases): w, v

Loop until stopping criteria satisfied:

present input pattern to input units
compute functional signal for hidden units
compute functional signal for output units

present Target response to output units
computer error signal for output units
compute error signal for hidden units
update all weights at same time
increment n to n+1 and select next input and target
end loop

Training the Network

- Training set presented repeatedly until stopping criteria is met
- ‘Epoch’ - each full presentation of all patterns
- Randomize order of training patterns presented for each epoch in order to avoid order effects
- Two types of network training
  - Sequential mode (on-line/ stochastic/or per-pattern)
    - Weights updated after each pattern is presented
  - Batch mode (off-line/ per -epoch)
    - Calculate the derivatives/wieght changes for each pattern in the training set
    - Calculate total change by summing individual changes

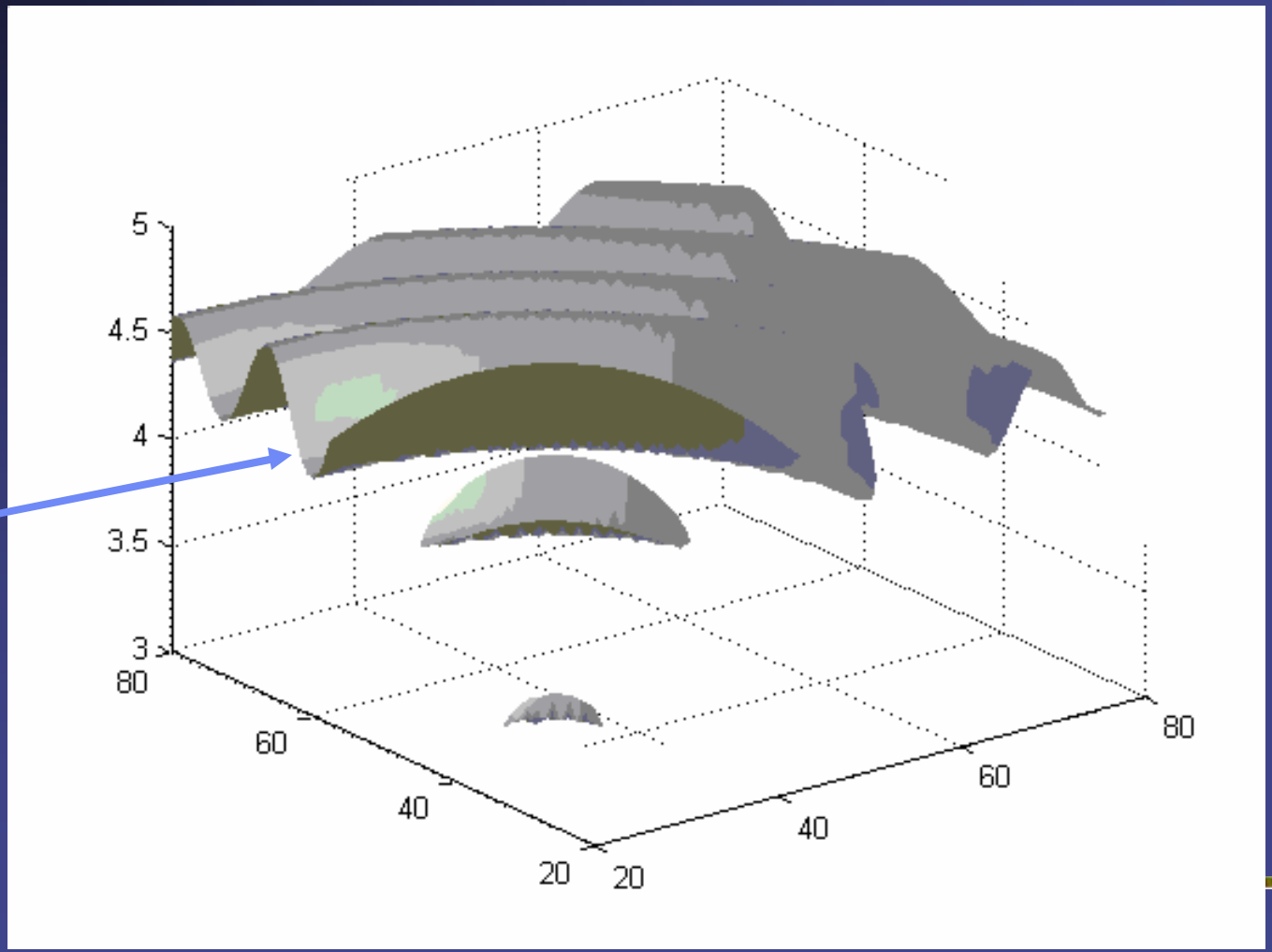
Sequential vs. Batch mode

- **Sequential mode**
  - Less storage for each weighted connection
  - Random order of presentation and updating per pattern means search of weight space is stochastic -reducing risk of local minima
  - Able to take advantage of any redundancy in training set
  - Simpler to implement
- **Batch mode:**
  - Faster learning than sequential mode
  - Easier from theoretical viewpoint
  - Easier to implement as parallelism

More on Backpropagation learning

- Minimising error function over all training patterns by adapting weights in MLP
- Mean squared error is
- E(t)= $\frac{1}{2} \sum_{k=1}^{p} (d_k(t) - O_k(t))^2$
- Trying to reduce E
- In a single layer network with linear activation functions error function is simple
  - described by a smooth parabolic surface with a single minimum

Nonlinear activation functions with complex error surfaces



Selecting initial weight values

- Choice of initial weight values is important
  - decides starting position in weight space
  - How far away from global minimum
- Aim is to select weight values which produce midrange function signals
- Select weight values randomly form uniform probability distribution
- Normalise weight values so number of weighted connections per unit produces midrange function signal

Momentum

- Method for reducing problems of instability while increasing the rate of convergence

- Adding term to weight update equation term
  - effectively exponentially holds weight history of previous weights changed

- Modified weight update equation is

$$w_{ij}(n ? 1) ? w_{ij}(n) ? ? ? _j(n)y_i(n)$$

$$? ? [w_{ij}(n) ? w_{ij}(n ? 1)]$$

Momentum

- ? is momentum constant and controls how much notice is taken of recent history
- Effect of momentum term
- If weight changes tend to have same sign
  - momentum terms increases and gradient decrease
  - speed up convergence on shallow gradient
- If weight changes tend have opposing signs
  - momentum term decreases
  - and gradient descent slows to reduce oscillations (stabilises)
- Can help escape being trapped in local minima

Stopping criteria

- Assesses train performance using

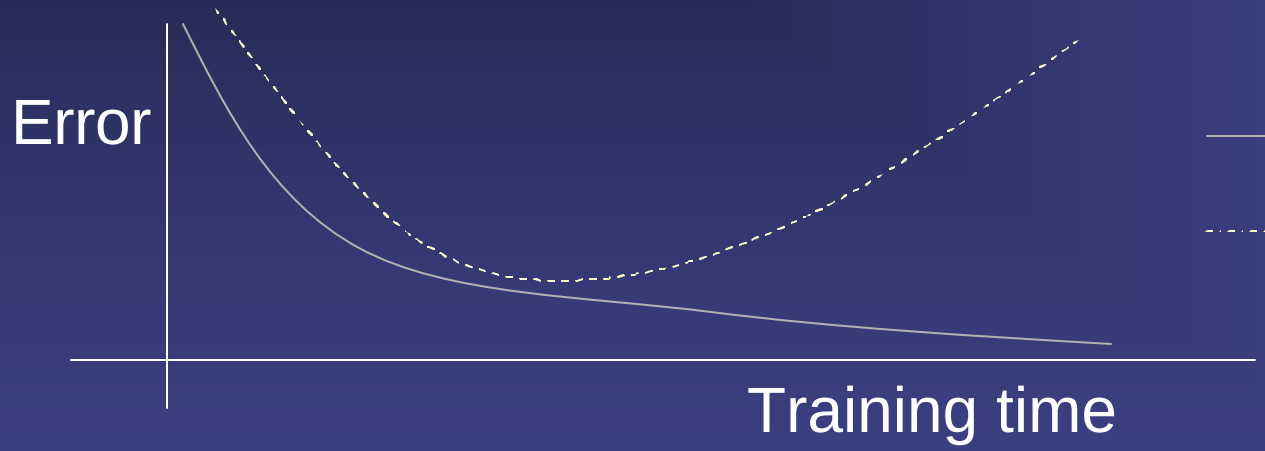
$$E ? \sum_{i?1}^{p} \sum_{j?1}^{M} [d_j(n) ? y_j(n)]_i^2$$

where p=number of training patterns

M=number of output units

- Could stop training when rate of change of E is small - suggesting convergence
- Aiming for new patterns to be classified correctly

Stopping criteria



Evaluation Methods

- **Cross-validation**
- ***Hold-out method***
  - Divide available data into sets
    - Training data set
      - used to obtain weight and bias values during network training
    - Validation data
      - used to periodically test ability of network to generalize
      - suggest ‘best’ network based on smallest error
    - Test data set
      - Evaluation of generalisation error (network performance)
  - Early stopping of learning to minimise the training error and validation error

Lab 2

- Part A
- Part B