Data Mining II: Advanced Methods and Techniques

Lecture 2

Natasha Balac, Ph.D.

Review

Are Our Brains Computers?

- Human brain is merely a very complex computer?
- The human brain is definitely not the type of computer with which most people are familiar
- Computers found in our homes and offices are *serial*
  - they only do one computation at a time
  - execute sequences of computations very quickly

Are Our Brains Computers?

- To perform even the simplest functions a serial computer requires hundreds or thousands of computations
  - Can be very fast (trillions of computations per second)
- They have limitations
- Very complex activity such as driving a car through a large city during rush hour

Brain vs. Serial Computer

- Our neurons do not transmit information very fast
- Even for a reflex that only involves three neurons, it may take several tenths of a second
- Somehow, we are able to process the information that it takes to drive a car using our relatively slow neurons

Human Brain

- Our nervous system does not process serially
- Human brain is an example of a *massively-parallel system*
  - rather than executing computations one-by-one in series
  - the human brain makes numerous computations simultaneously when performing a neural process

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

About Neural Networks

- Most NNs have some sort of "training" rule
  - weights of connections are adjusted based on data
- NNs "learn" from examples
  - as children learn to distinguish dogs from cats based on examples of dogs and cats
- NNs are capable of exhibiting capability for generalization beyond the training data
  - to produce approximately correct results for new, unseen cases that were not used for training

About Neural Networks

- NNs normally have great potential for parallelism
  - Computations of the components are largely independent of each other
- Are massive parallelism and high connectivity are some of defining characteristics

Neural nets origin - The human brain

- Consists of more than a billion neural cells that process information
- Each cell works like a simple processor
  - the massive interaction between all cells and their parallel processing makes the brain's abilities possible
- On average neuron is connected to other neurons through about 10 000 **synapses**

Brain as an Information Processing System

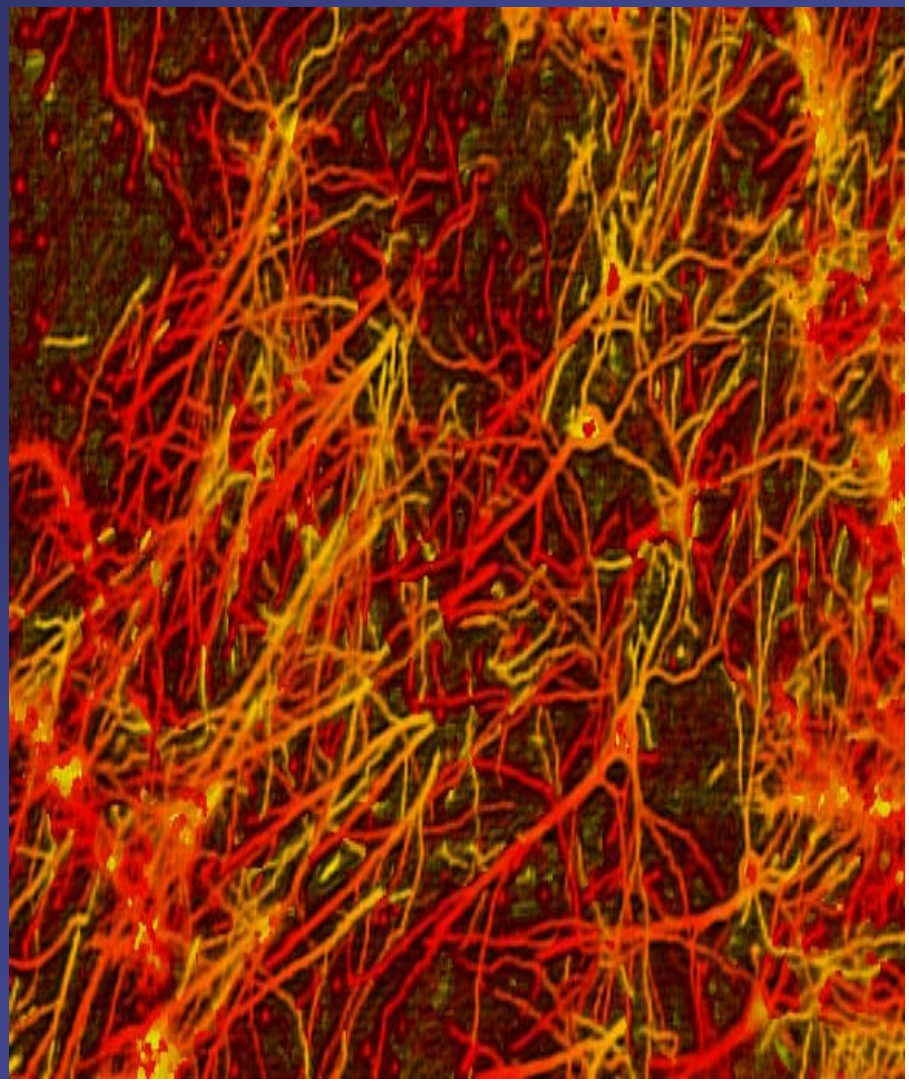
- Despite of being built with very slow hardware, the brain has quite remarkable capabilities
  - its performance tends to degrade gracefully under partial damage
- In contrast most programs and engineered systems are brittle
  - if you remove some arbitrary parts, very likely the whole will cease to function previously carried out by the damaged areas

Brain as an Information Processing System

- It can learn from experience
- Partial recovery from damage is possible if healthy units can learn to take over the functions previously carried out by the damaged areas
- Performs massively parallel computations extremely efficiently
- It supports our intelligence and self-awareness
  - How? Nobody knows yet

Neurons

- In addition to these long-range connections, neurons also link up with many thousands of their neighbors
- To form very dense, complex local networks



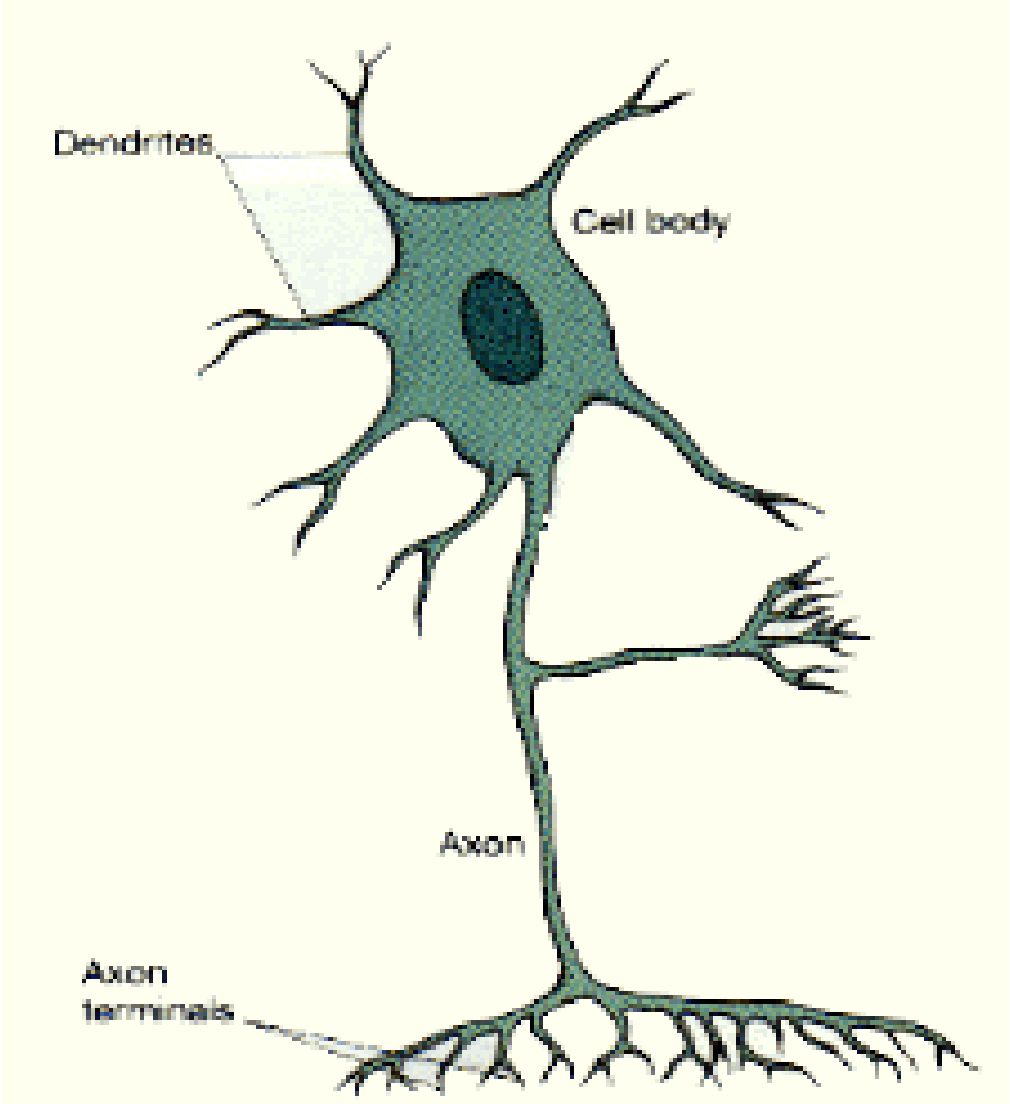
Human Brain

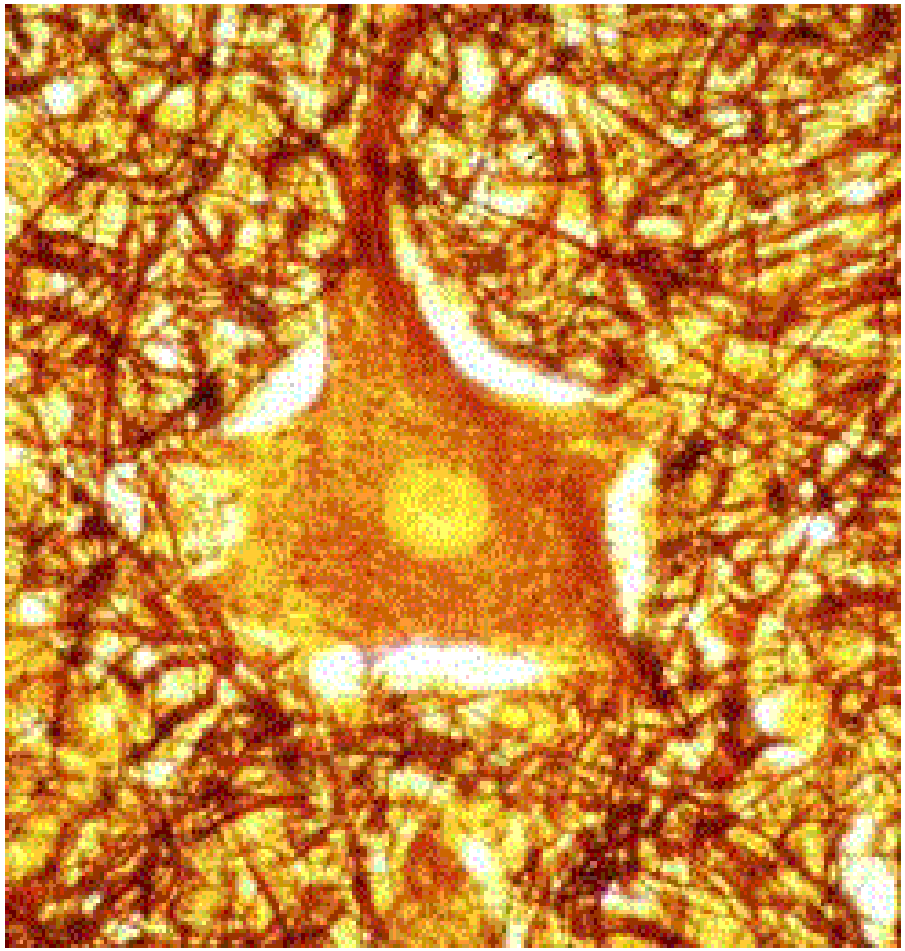
- Our brains are made up of about 100 billion tiny units called *neurons*
- Each neuron is connected to thousands of other neurons and communicates with them via electrochemical signals
- Signals coming into the neuron are received via junctions called *synapses*
- Synapses are located at the end of branches of the neuron cell called *dendrites*

Human Brain

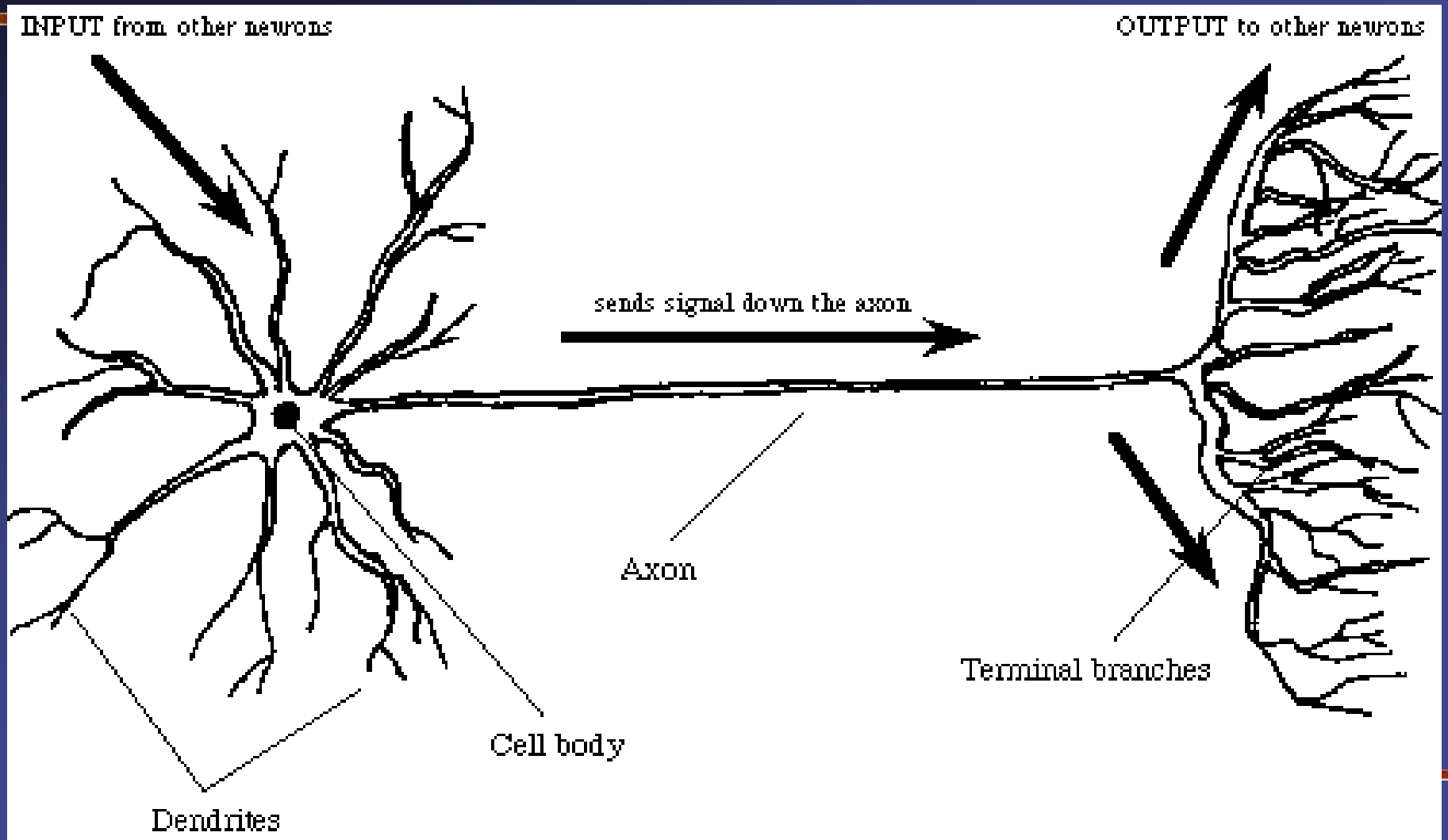
- The neuron continuously receives signals from inputs and then performs a little bit of “magic”
- Neuron sums up the inputs to itself (in some way) and then, if the end result is greater than some threshold value, the neuron fires
- It generates a voltage and outputs a signal along an *axon*

A sketch a neural cell or neuron





Neuron



Biological Neuron

- A biological neuron may be
  - connected to other neurons
  - accepting connections from other neurons
- Through those connections
  - electrical pulses are transmitted
  - information is carried in the timing and the frequency with which these pulses are emitted

More About Neurons

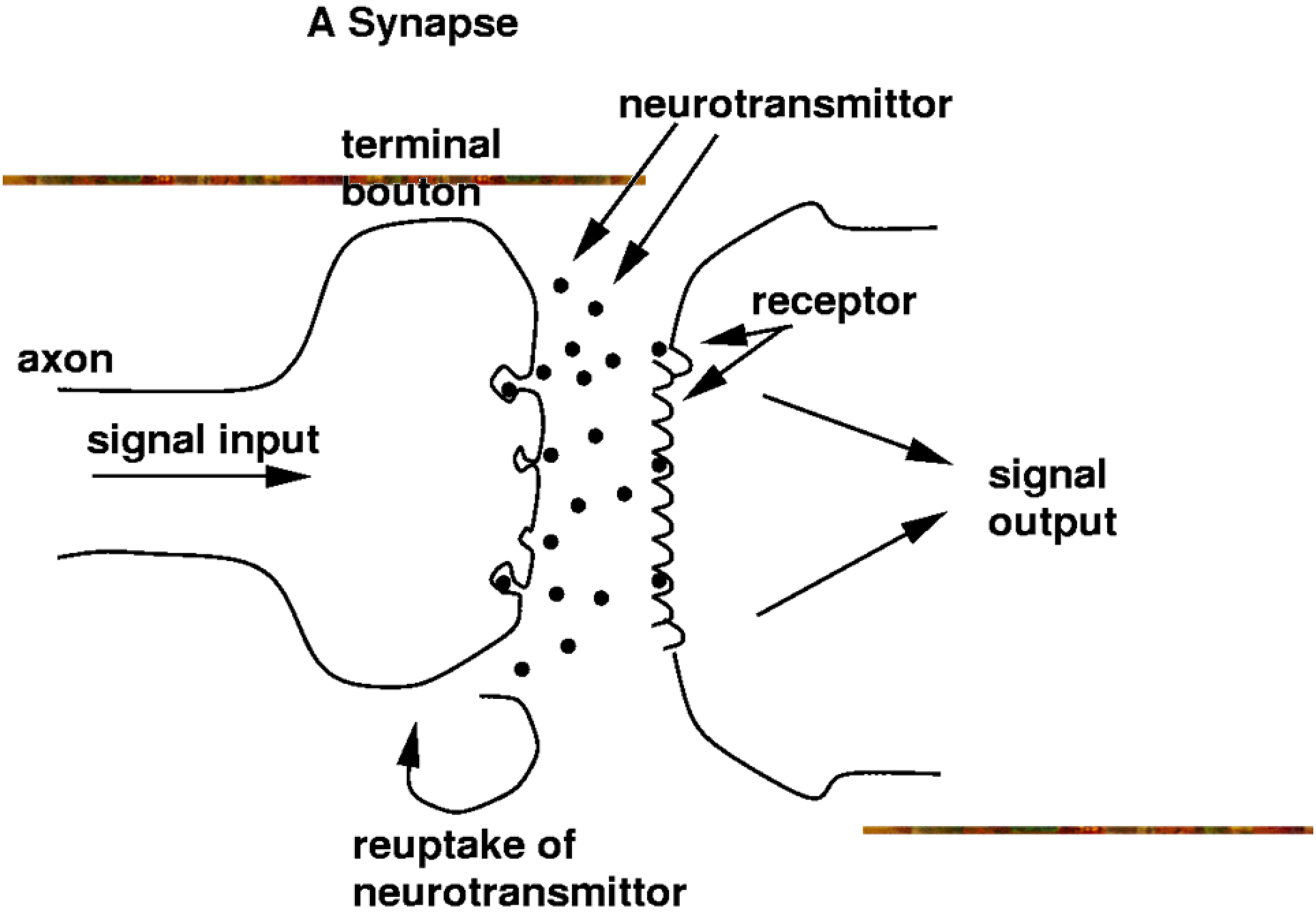
- Neuron receives information from other neurons, processes it and then relays this information to other neurons
- What form does this processing take?
- Clearly the neuron must generate some kind of output based on the cumulative input
- Neuron integrates the pulses that arrive and when this integration exceeds a certain limit, neuron in turn emits a pulse

Neurons and Synapses

- A neuron receives input from other neurons (typically many thousands)
- Inputs sum (approximately)
- Once input exceeds a critical level, the neuron discharges a **spike** - an electrical pulse that travels from the body, down the axon, to the next neuron(s) (or other receptors)

Brains Learn - Of Course How?

- One way brains learn is by altering the strengths of connections between neurons and by adding or deleting connections between neurons
- They learn "on-line”
  - based on experience
  - and typically without the benefit of a teacher


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

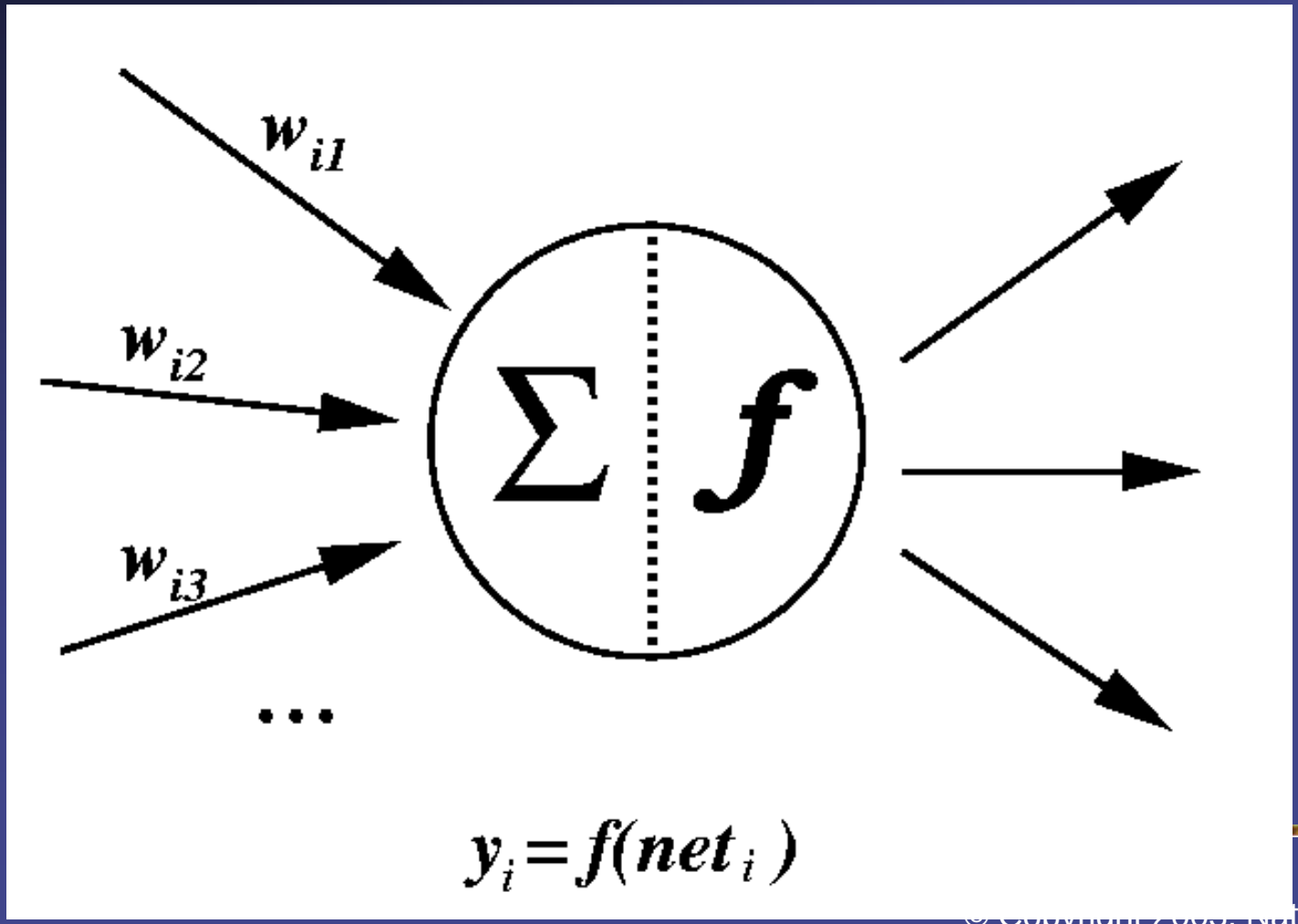
A Simple Artificial Neuron

- 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



Activation Function

- Identity Function
- Step Function
- Logistic Function (Sigmoid)
- Symmetric Sigmoid
- Radial Basis Functions
- Derivatives

The Loss Function

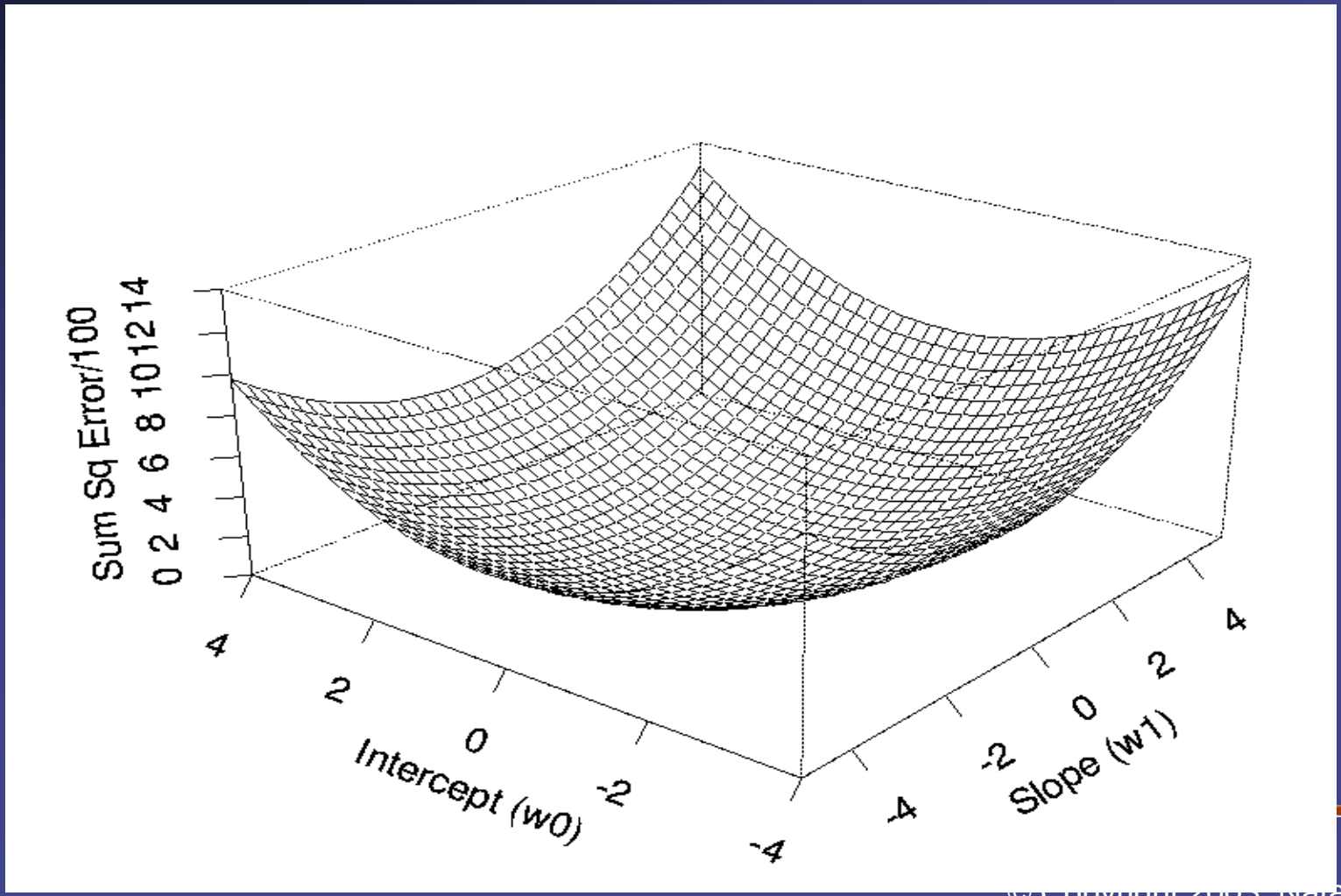
- In order to make precise what we mean by being a "good predictor”
- Define a **loss** (**objective** or **error**) function *E* over the model parameters
- Popular choice for *E* is the **sum-squared error**

$$E = \frac{1}{2}\sum_{p}(t_p - y_p)^2$$

$$E = \frac{1}{2} \sum_p (t_p - y_p)^2$$

- Sum over all points *i* in our data set of the squared difference between
  - **target** value t_i (actual fuel consumption) and
  - the model's prediction y_i, calculated from the input value x_i (weight of the car)
    - $y = w_1 x + w_0$
  - For a linear model, the sum-sqaured error is a quadratic function of the model parameters

Figure showing E for a range of values of w_0 and w_1



Minimizing the Loss

- The loss function *E* provides an objective measure of predictive error for a specific choice of model parameters
- We can thus restate our goal of finding the best (linear) model as finding the values for the model parameters that minimize *E*

Gradient Descent

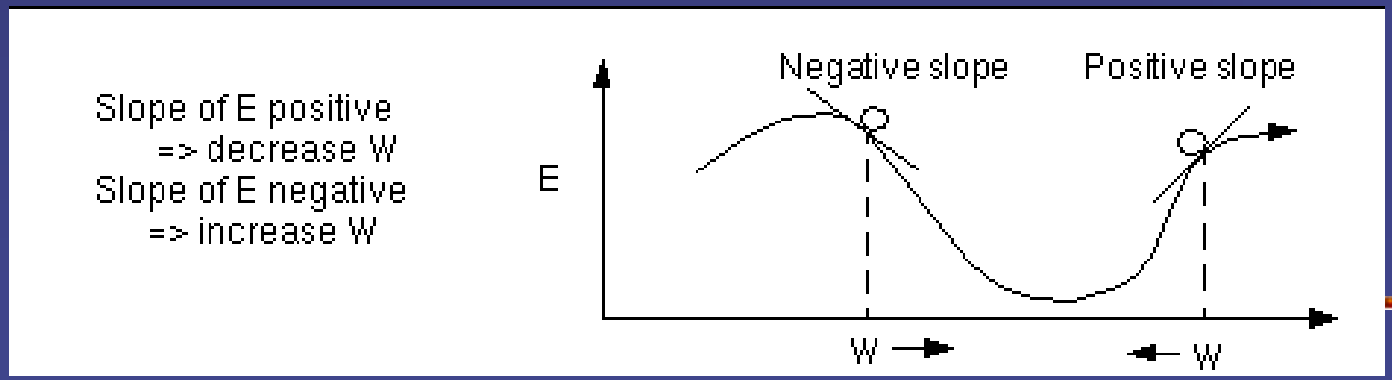
- For linear models
  - **linear regression** provides a direct way to compute these optimal model parameters
- Does not generalize to **nonlinear** models
- Even though the solution cannot be calculated explicitly in that case
  - the problem can still be solved by an iterative numerical technique called **gradient descent**

Gradient Descent

- Choose some (random) initial values for the model parameters
- Calculate the gradient G of the error function with respect to each model parameter
- Change the model parameters so that we move a short distance in the direction of the greatest rate of decrease of the error, i.e., in the direction of -G
- Repeat steps 2 and 3 until G gets close to zero

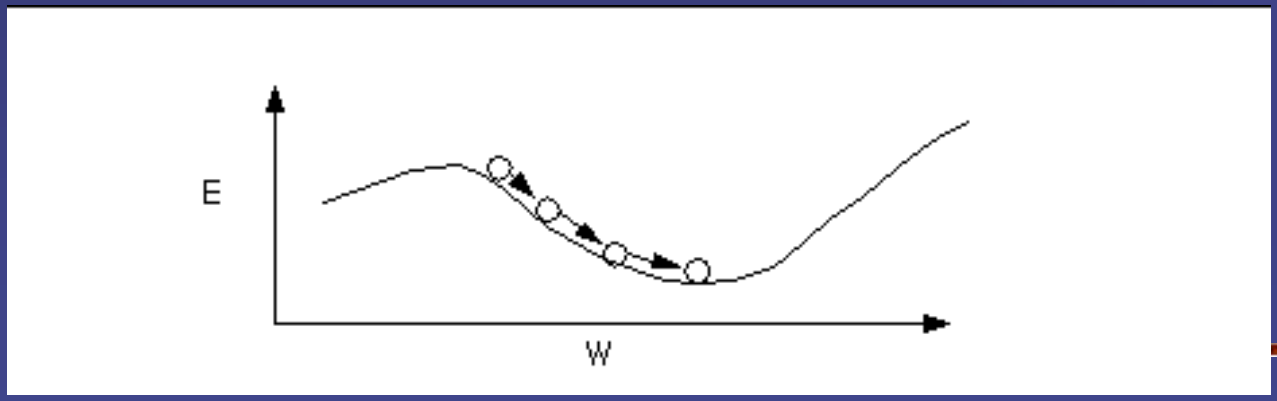
How does this work?

- The gradient of E gives us the direction in which the loss function at the current setting of the *w* has the steepest **slope**
- In order to decrease *E* take a small step in the opposite direction or *-G*



Gradient Descent

- By repeating this over and over, we move "downhill" in *E* until reach a minimum where *G* = 0
- so that no further progress is possible



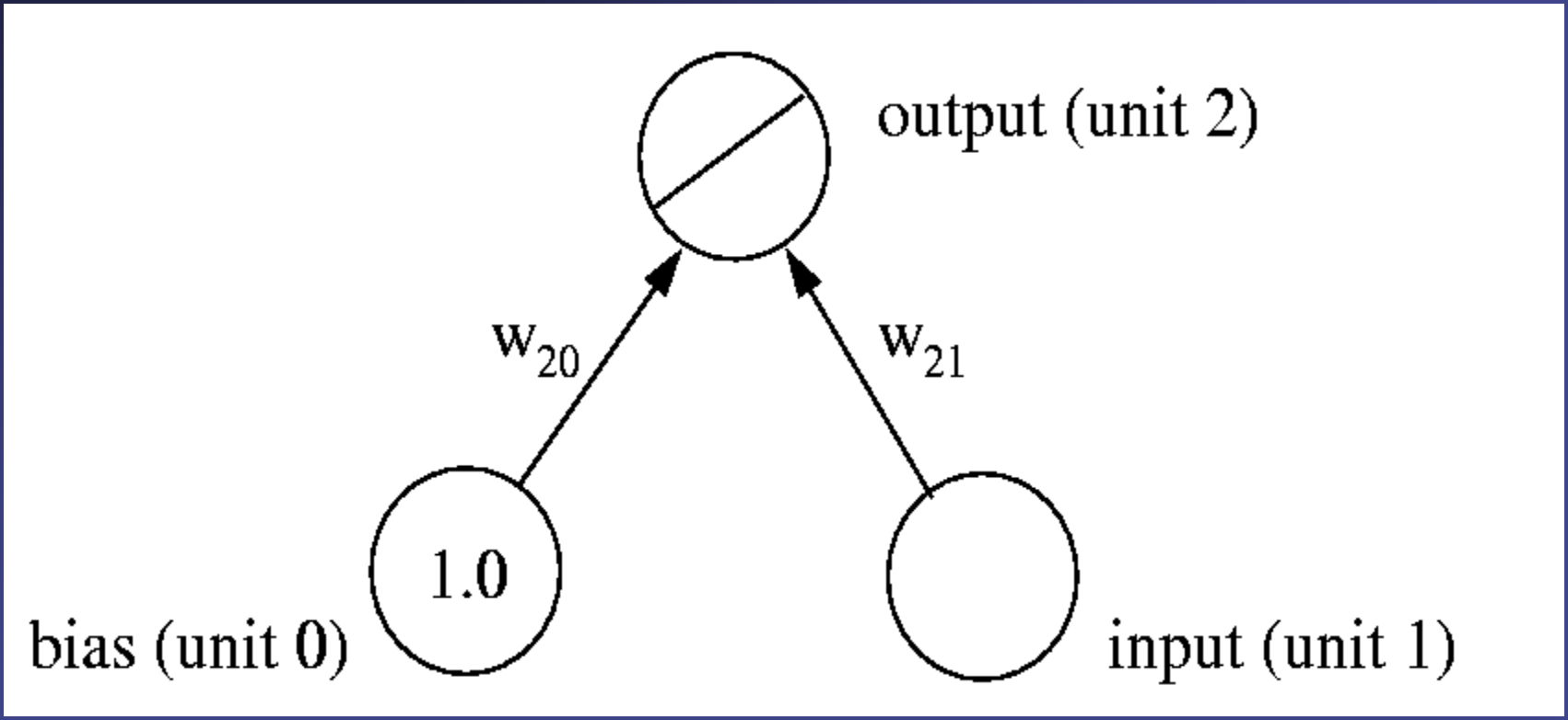
Simple Neural Network

- Linear model of equation $y = w_1 x + w_0$ can be implemented by the simple neural network
  - It consists of a
    - **bias** unit
    - an **input** unit
    - a linear **output** unit
- The input unit makes external input x (the weight of a car) available to the network
- Bias unit always has a constant output of 1

Simple Neural Network

- The output unit computes the sum
  - $y_2 = y_1 w_{21} + 1.0 w_{20}$
- This is equivalent to the previous equation $y = w_1 x + w_0$
  - with w_{21} implementing the slope of the straight line
  - and w_{20} its intercept with the y-axis

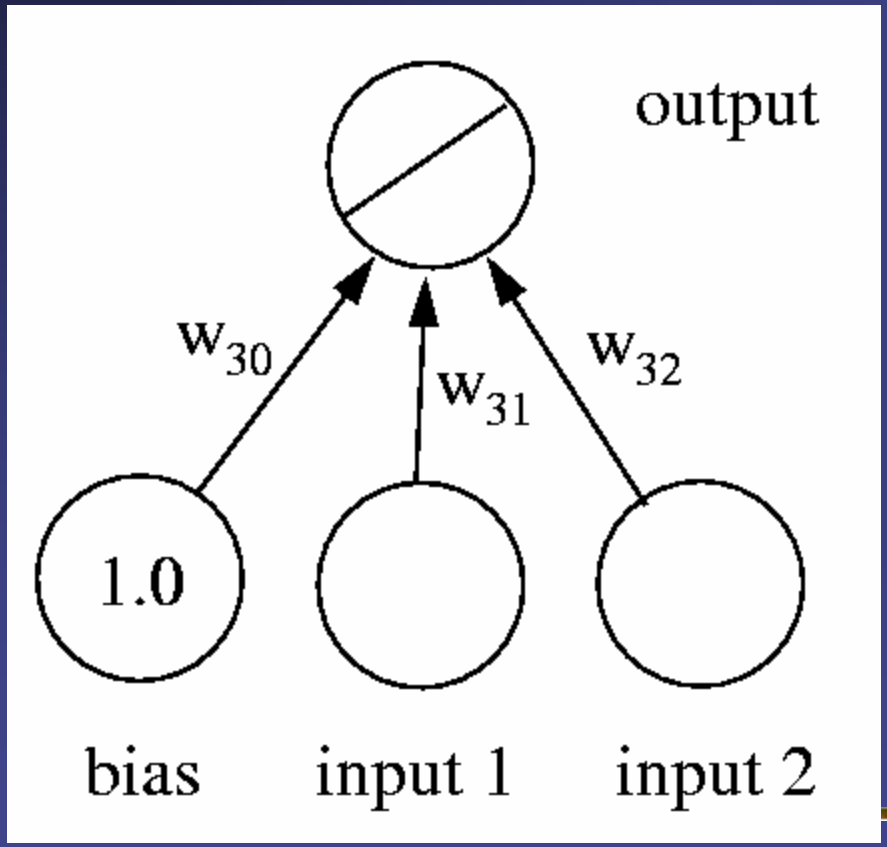
Simple Neural Network



Multiple regression

- Car example showed how we could discover an optimal linear function for predicting one variable (fuel consumption) from one other (weight)
- What if we are also given one or more additional variables which could be useful as predictors?
- Simple neural network model can be extended by adding more input units

Multiple inputs

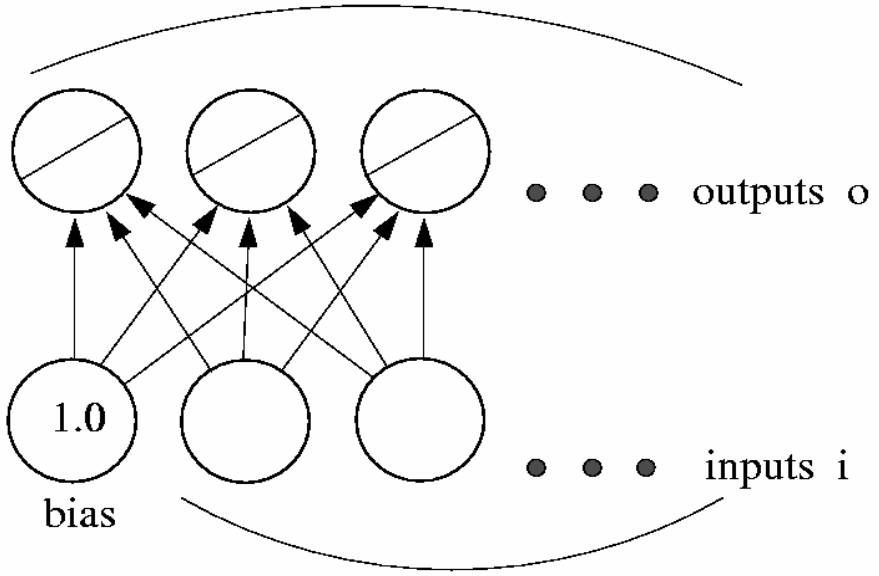


Multiple Outputs

- To predict more than one variable from the given data
  - Adding more output units
- The loss function for a network with multiple outputs is obtained simply by adding the loss for each output unit together

Network Structure

- The network now has a typical layered structure
  - a layer of input units (and the bias)
  - connected by a layer of weights to
  - a layer of output units



Computing the gradient

- Compute the gradient *G* of the loss function with respect to each weight w_{ij} of the network
- How a small change in that weight will affect the overall error *E*
- We begin by splitting the loss function into separate terms for each point *p* in the training data:

$$E = \sum_p E^p, \qquad E^p = \frac{1}{2}\sum_o (t_o^p - y_o^p)^2$$

Computing the gradient

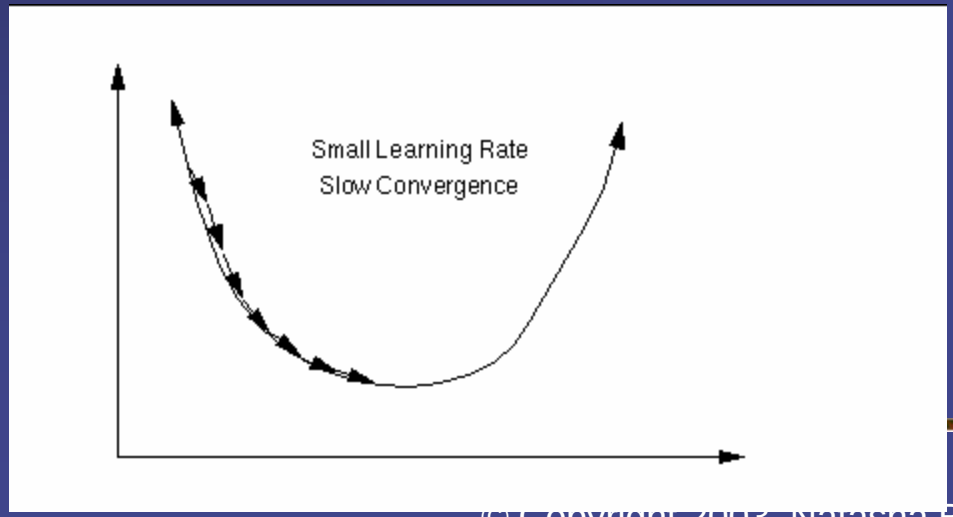
- To find the gradient *G* for the entire data set
- Sum at each weight the contribution given by equation $\frac{\partial E}{\partial w_{oi}} = -(t_o - y_o)\, y_i$ over all the data points
- We can then subtract a small proportion μ (called the **learning rate**) of *G* from the weights to perform gradient descent

The Gradient Descent Algorithm

- Initialize all weights with small random values
- REPEAT until done
  - For each weight w_{ij} set $\Delta w_{ij} := 0$
  - For each data point $(\mathbf{x}, \mathbf{t})^p$
    - set input units to **x**
    - compute value of output units
    - For each weight w_{ij} set $\Delta w_{ij} := \Delta w_{ij} + (t_i - y_i)\, y_j$
  - For each weight w_{ij} set $w_{ij} := w_{ij} + \mu\, \Delta w_{ij}$

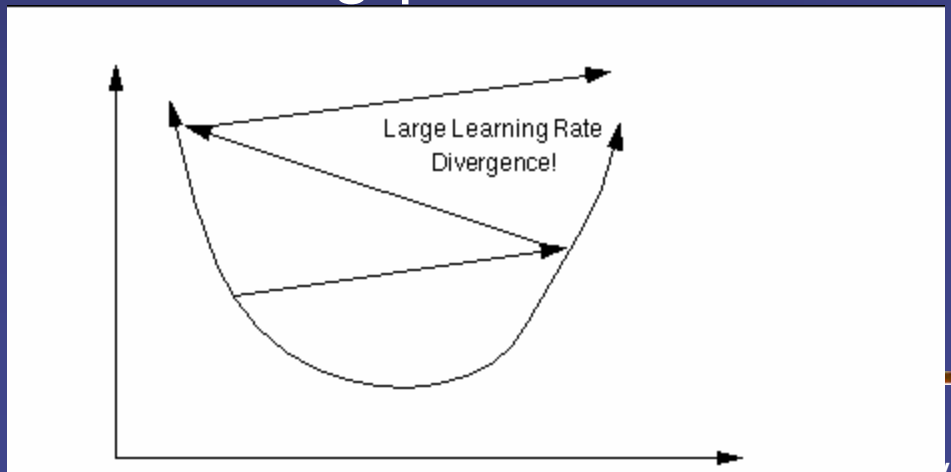
The Learning Rate

- Important consideration is the learning rate μ
  - determines by how much we change the weights w at each step
- If μ is too small the algorithm will take a long time to converge



The Learning Rate

- If μ is too large
  - we may end up bouncing around the error surface out of control - the algorithm **diverges**
  - This usually ends with an overflow error in the computer's floating-point arithmetic



Batch vs. Online Learning

- **Batch** learning
  - Accumulate the gradient contributions for all data points in the training set before updating the weights
- **Online** learning
  - weights are updated immediately after seeing each data point

Online Learning Advantages

- Often much faster, especially when the training set is **redundant**
- Can be used when there is no fixed training set
- Better at tracking **non-stationary** environments
- Noise in the gradient can help to escape from **local minimum**

Online Learning Disadvantages

- Many powerful optimization techniques (conjugate and second-order gradient methods, support vector machines, Bayesian methods) are batch methods that cannot be used online
- A compromise between batch and online learning is the use of "mini-batches"
  - the weights are updated after every *n* data points
    - where *n* is greater than 1 but smaller than the training set size

Today

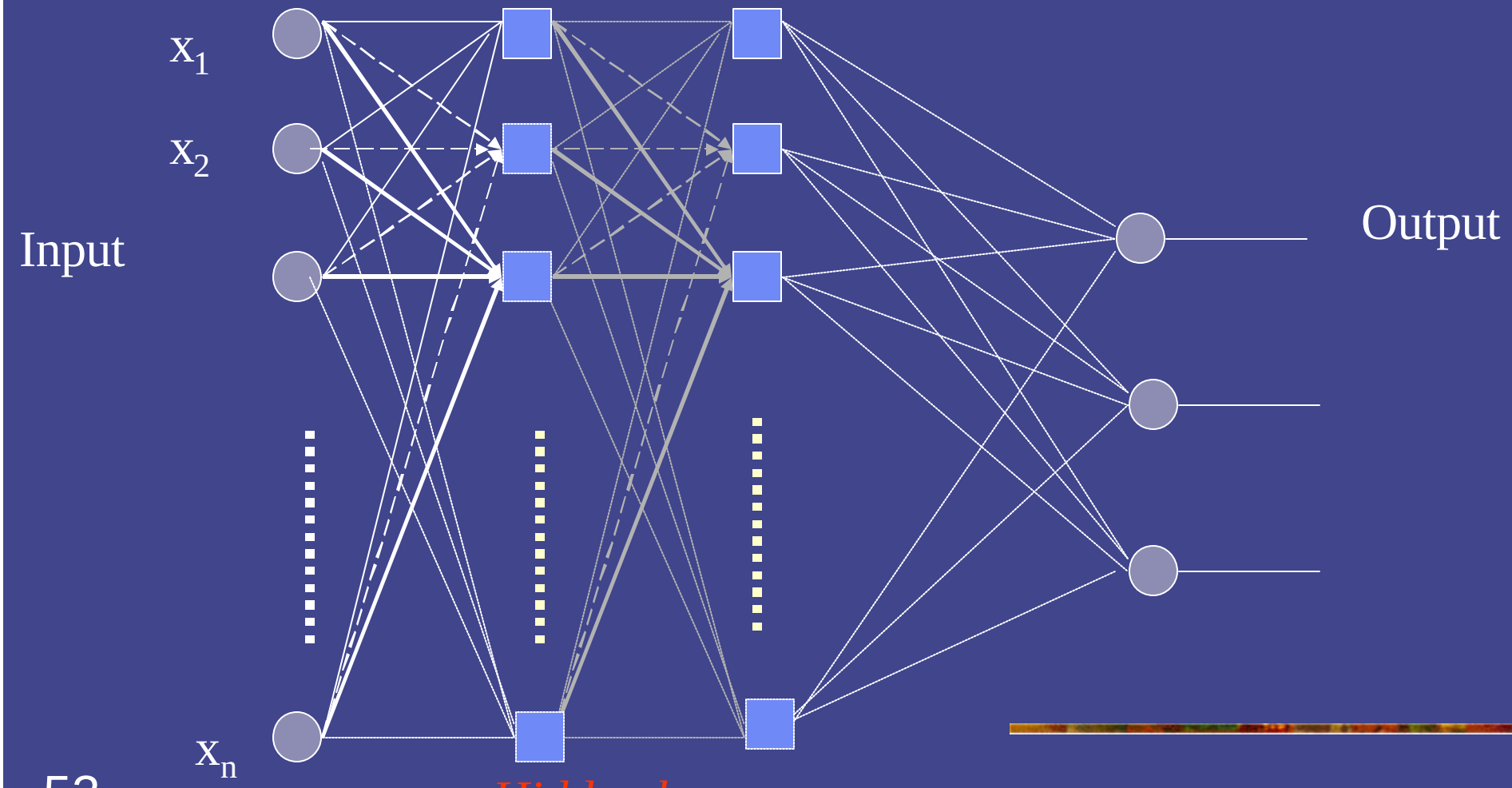
- McCullogh-Pitts Neuron
- Artificial Neuron
- Generalization
- Classification
- Backpropagation
- Lab 1

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



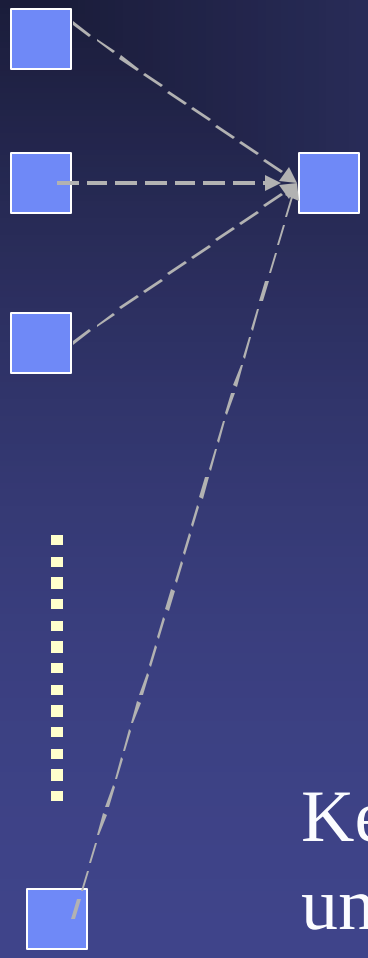
General Artificial Neuron Model

- 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 generalise

NNs for function approximation

- Network learns a ('correct') mapping from inputs to outputs
- NNs can be seen as being a multivariate non-linear mapping and are often used for function approximation
- 2 main categories:
  - Classification: given an input say which class it is in
  - Regression: given an input what is the expected output

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***

Pattern recognition

- **Pattern: an entity, vaguely defined, that could be given a name or a classification**

- **Examples:**
  - **Fingerprints**
  - **Hand-written characters**
  - **Human face**
  - **Speech (or deer/whale/bat etc) signals**
  - **Medical imaging (various screening procedures)**
  - **Remote sensing etc.**

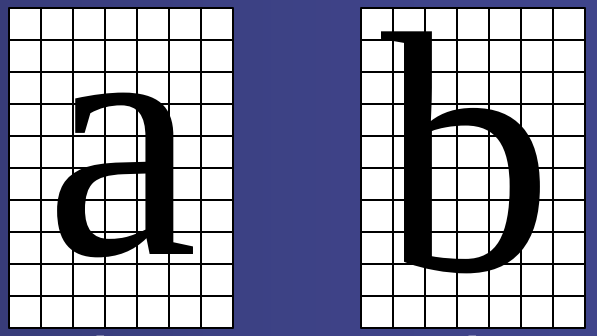
Classification

a. supervised classification (discriminant analysis) in which the input pattern is identified as a member of a predefined class

b. unsupervised classification (e.g. clustering) in which the patter is assigned to a unknown class

Example: Handwritten digit classification

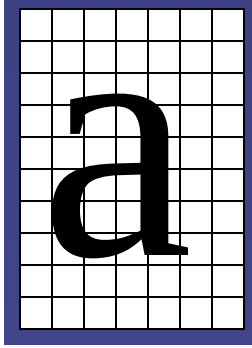
- First need a data set to learn from: sets of characters

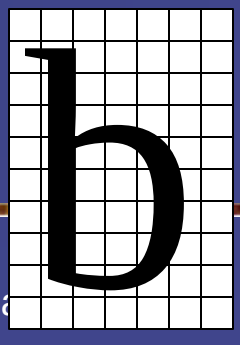


- How are they represented?
  - Input vector $\underline{x} = (x_1, \ldots, x_n)$ to the network
    - vector of ones and zeroes for each pixel according to whether it is black/white

Example: Hand-written digit classification

- Set of input vectors is our **Training Set** X which has already been classified into a's and b's
- Given a training set X, our goal is to tell if a new image is an a or b
  - Classify it into one of 2 classes C_1 or C_2
  - in general one of k classes C_1.. C_k





Generalization

Q. How do we tell if a new unseen image is an a or b?

A. Brute force: have a library of all possible images

There are:

$$256 \text{ x } 256 \text{ pixels} => 2^{256 \text{ x } 256} = 10^{158,000} \text{ images}$$

Impossible! Typically have less than a few thousand images in training set

Generalization Problem

- **System must be able to classify UNSEEN patterns from the patterns it has seen**
  - **I.e. Must be able to generalise 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

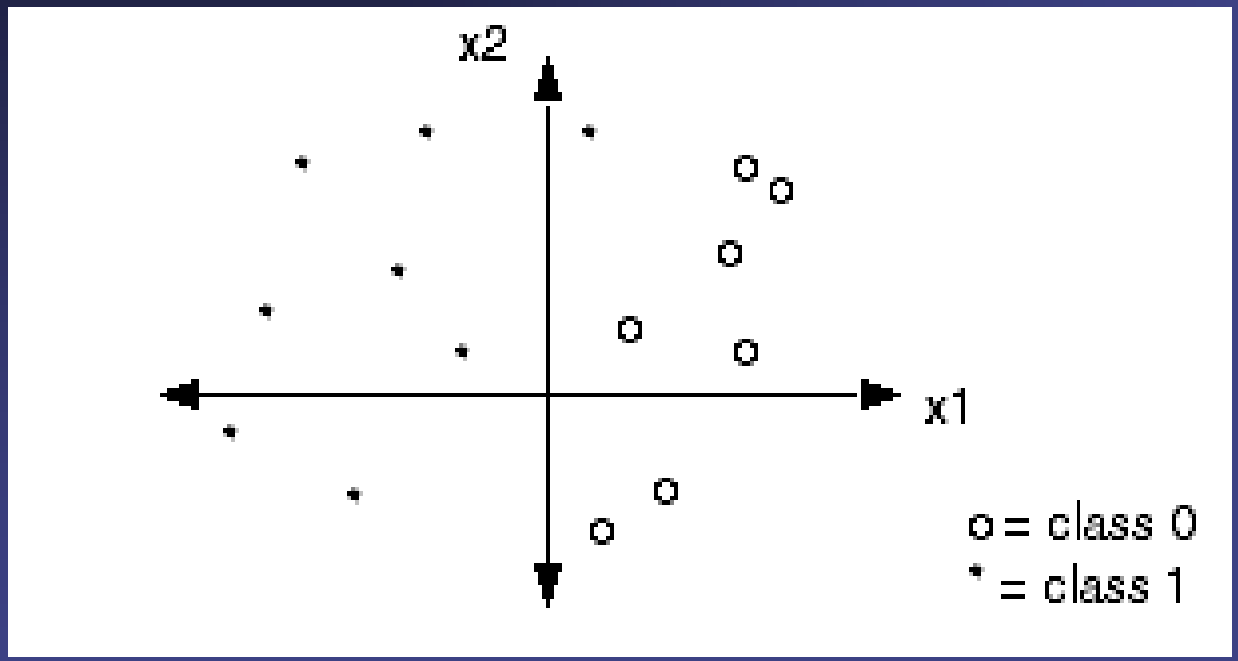
- **For 2 class classification we want network output y (function of inputs and network parameters) to be:**

  $y(\underline{x}, \underline{w}) = 1$ if $\underline{x}$ is an a

  $y(\underline{x}, \underline{w}) = -1$ if $\underline{x}$ is a b

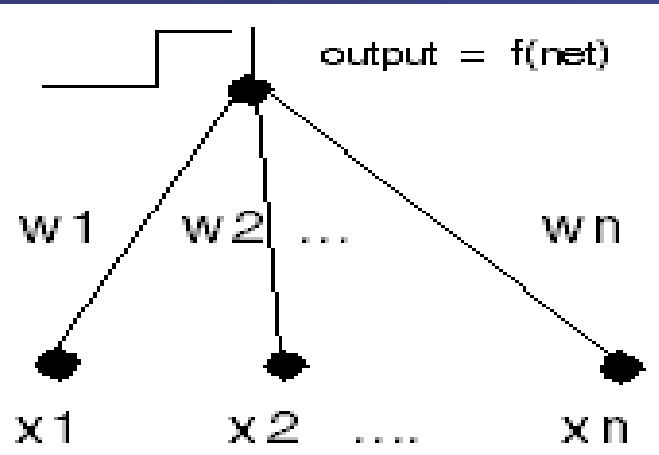
- **where $\underline{x}$ is an input vector and the network parameters are grouped as a vector $\underline{w}$**
- **y is known as a discriminant function**
  - **it discriminates between 2 classes**

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



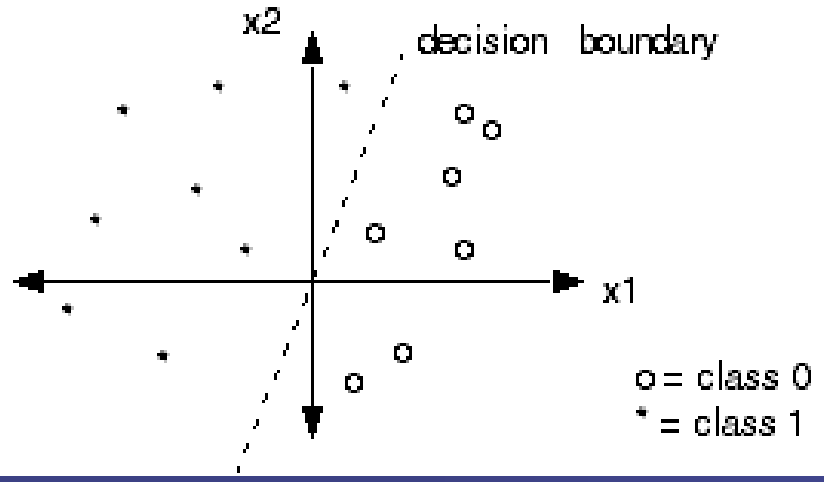
Training Methods

- Two kinds of methods for training single-layer networks that do pattern classification
- Perceptron - guaranteed to find the right weights if they exist
- The Adaline (uses Delta Rule) - can be generalized to multi-layer nets (solving nonlinear problems)
- But how do we know if the right weights exist at all?

Network with 2 inputs and 1 output node (2 classes)

- **The net output of the network is a linear function of the weights and the inputs**
- **net = W X = x1 w1 + x2 w2**
  y = f(net)

Suppose we want the output of the net to be
0 for class 0 and
1 for class 1.
This means that we want
net > 0 for class 1
net < 0 for class 0
=> The dividing line between two classes is defined by net = 0



- **x1 w1 + x2 w2 = 0 defines a straight line through the input space**
- **x2 = - w1/w2 x1 <- this is line through the origin with slope -w1/w2**

What if line dividing 2 classes does not go through the origin?

To describe this line we need an extra constant b

$y = a\,x + b$

We can incorporate this into the net by using a bias.

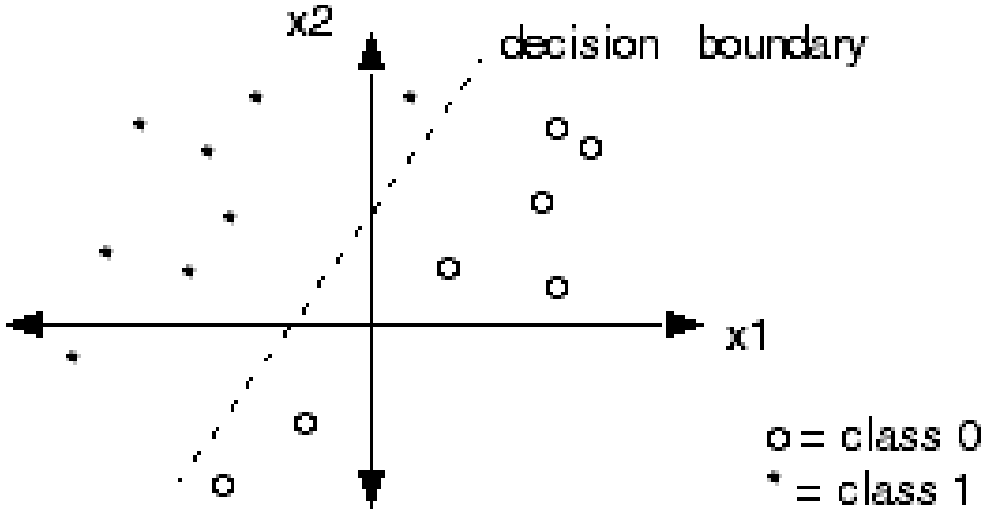
We already saw the bias in the case of regression. We do the same thing here by adding a fictitious input clamped at 1. The weight associated with this weight is called the bias weight.

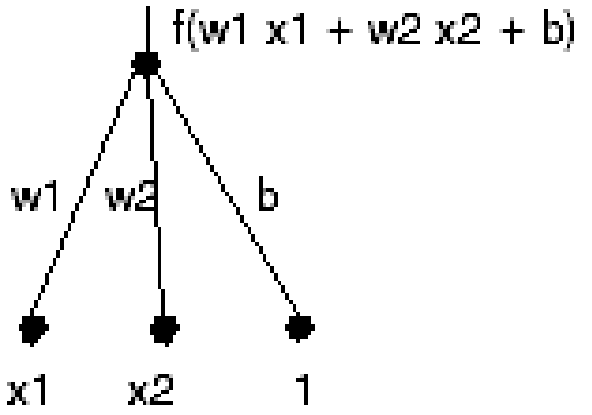
Now, we have

$net = 0 = w1\,x1 + w2\,x2 + b$

$= W\,X + b$

or





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

Multivariate regression

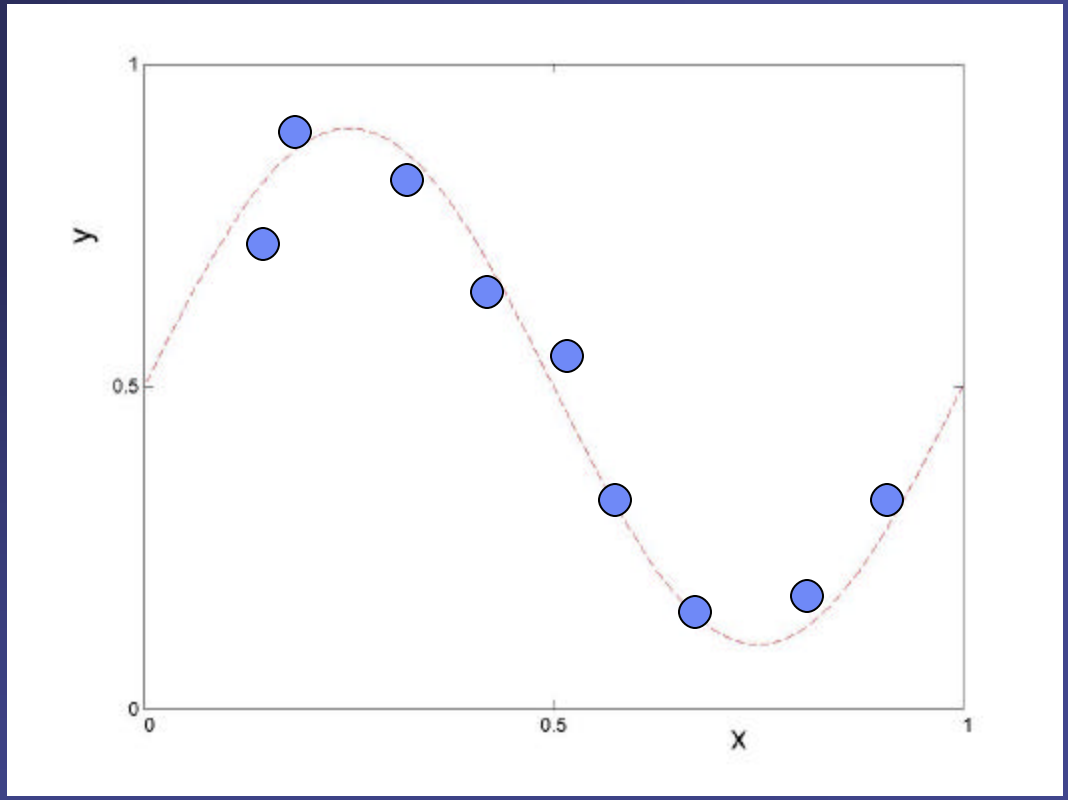
- **Type of function approximation: try to approximate a function from a set of (noisy) training data**
- **Example: suppose we have the function**

$$y = 0.5 + 0.4 \sin(2\pi x)$$

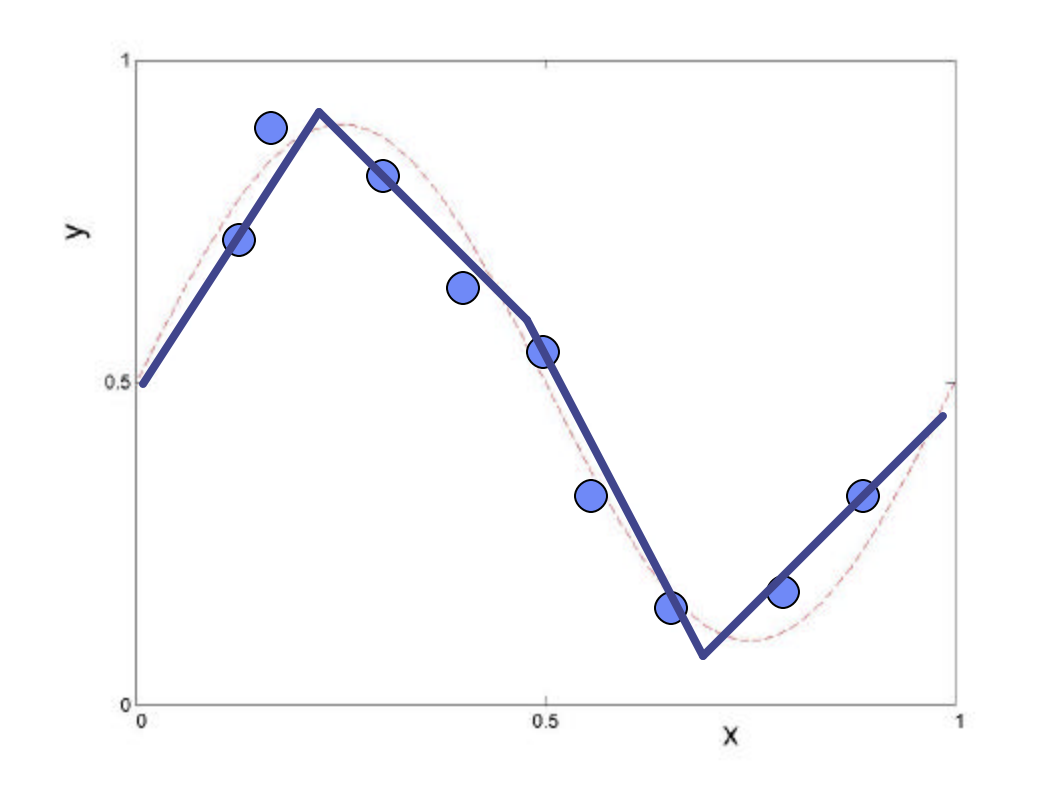
- **Generate training data at equal intervals of x and add a little random Gaussian noise with s.d. 0.05**
- **Add noise since in practical applications data will inevitably be noisy**

Example

$y = 0.5 + 0.4 \sin(2\pi x)$



Example



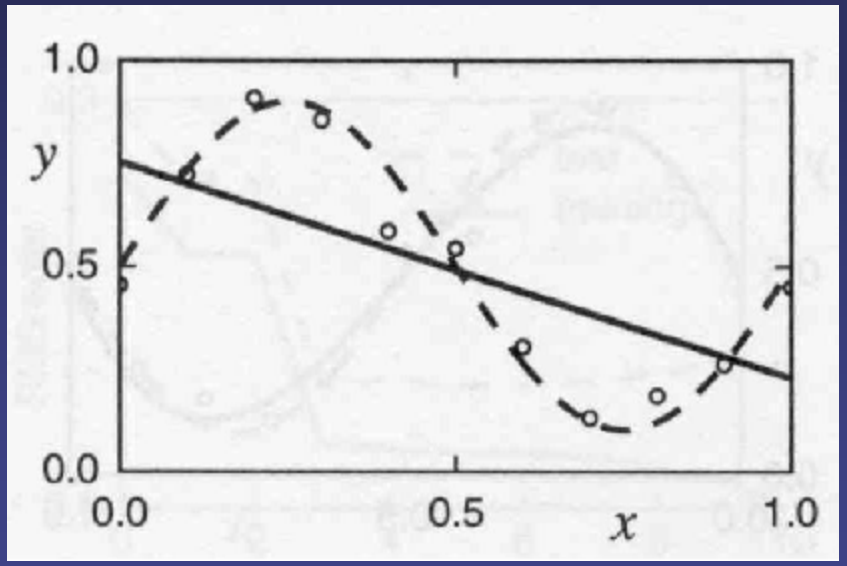
This gives an idea of the *Generalization* performance of the model

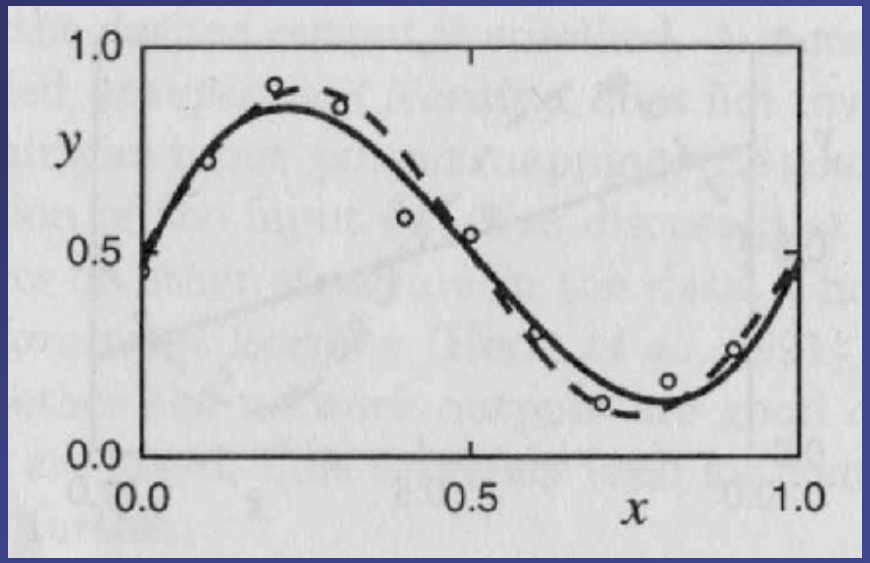
Model Complexity

- **Piecewise linear function to approximate the data**
- **Better use a polynomial $y = ? a_i x^i$ to approximate**

$y = a_0 + a_1 x$	**1st order (straight line)**
$y = a_0 + a_1 x + a_2 x^2$	**2nd order (quadratic)**
$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3$	**3rd order**
$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_n x^n$	**nth order**

Model Complexity

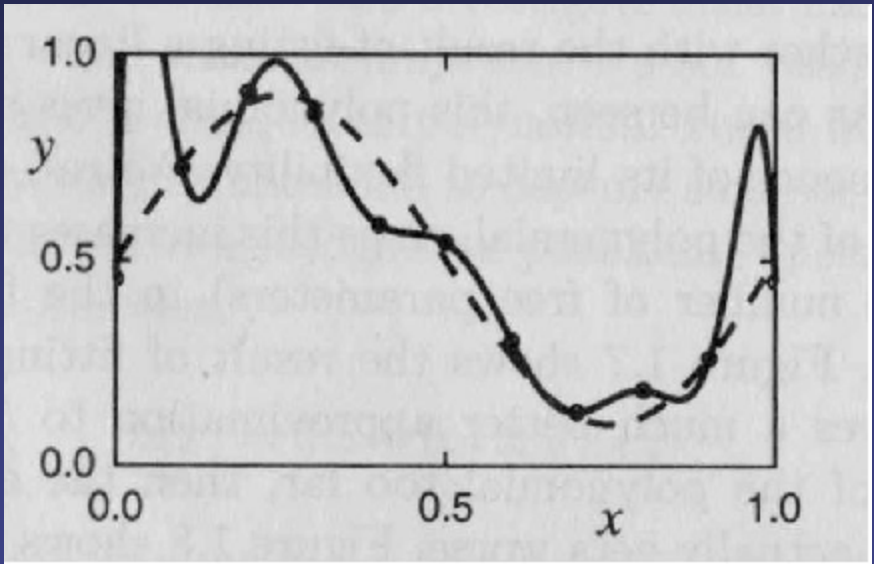




1st order model too simple

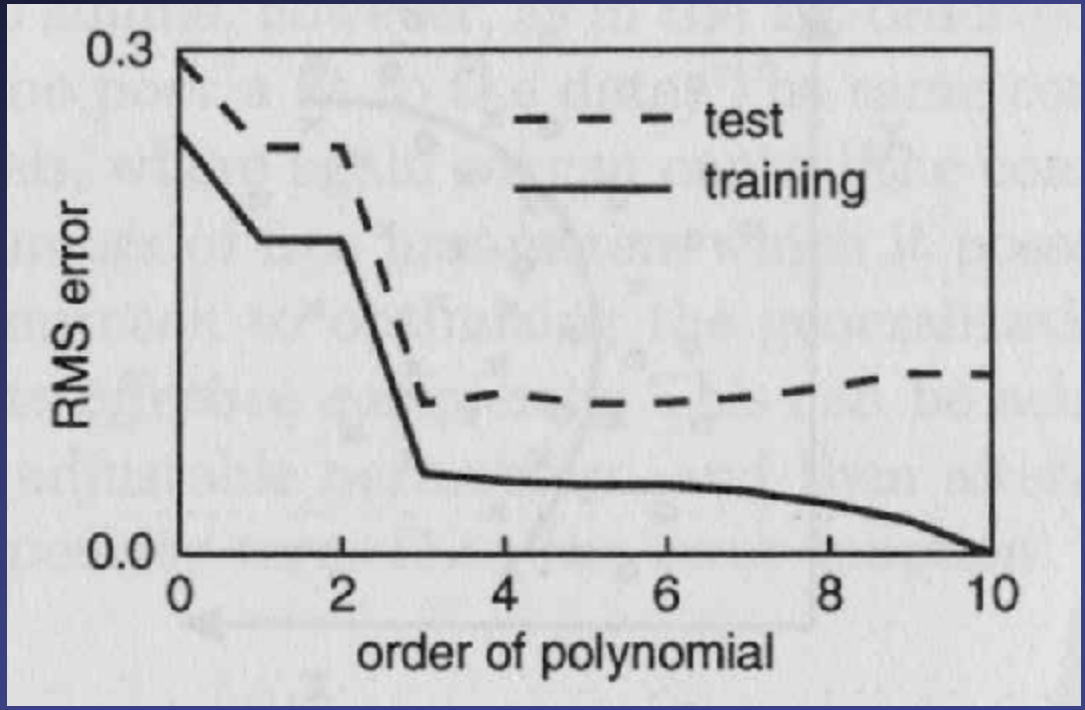
3rd order: models underlying function well

Model Complexity



10th order: more accurate in terms of passing through data points but is too complex and non-smooth (curvy)

Model Complexity

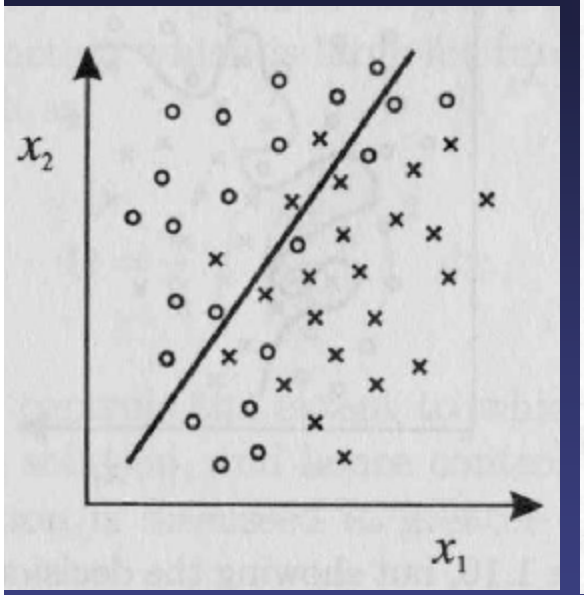


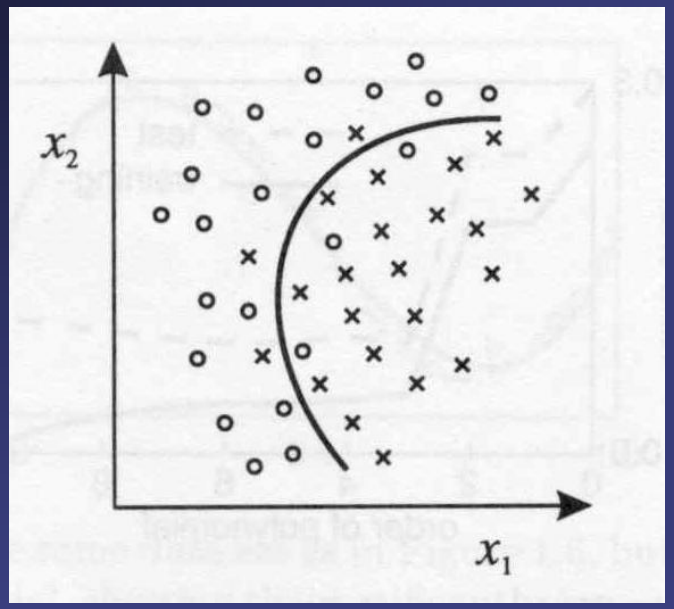
As the model complexity grows performance improves for a while but starts to degrade after reaching an optimal level

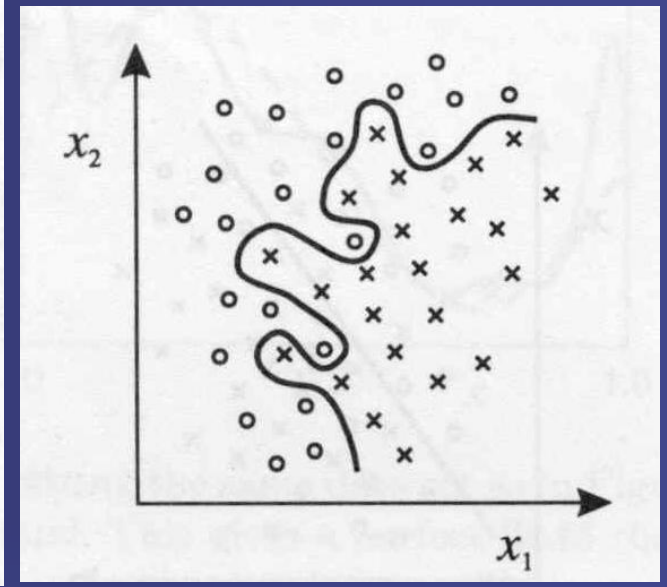
Model Complexity

- 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**

Bias-Variance trade-off

- Somewhat like giving a system enough capacity to 'remember' all training point
  - no need to generalise
- Less memory => it must generalise to be able to model training data

- Trade-off between being a good fit to the training data and achieving a good generalisation
  - ***Bias-Variance trade-off***

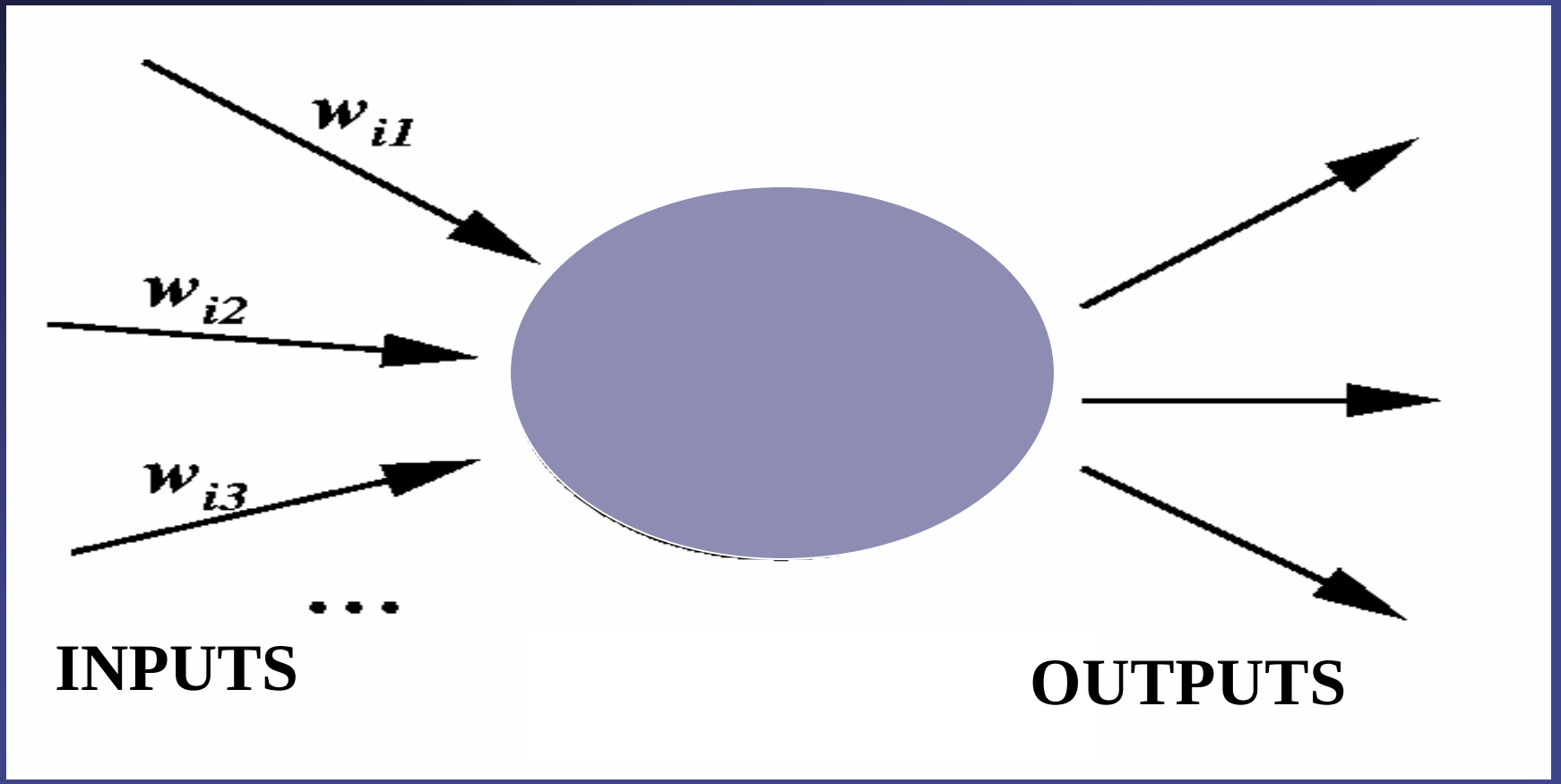
Neural Networks

- Made up of many artificial neurons
- An artificial neuron is simply an electronically modeled biological neuron
- How many neurons are used depends on the task at hand
- It could be as few as three or as many as several thousand

Neural Networks

- One optimistic researcher has even hard wired 2 million neurons together
- There are many different ways of connecting artificial neurons together to create a neural network
- Most common which is called a *feedforward* network

What Does An Artificial Neuron Look Like?

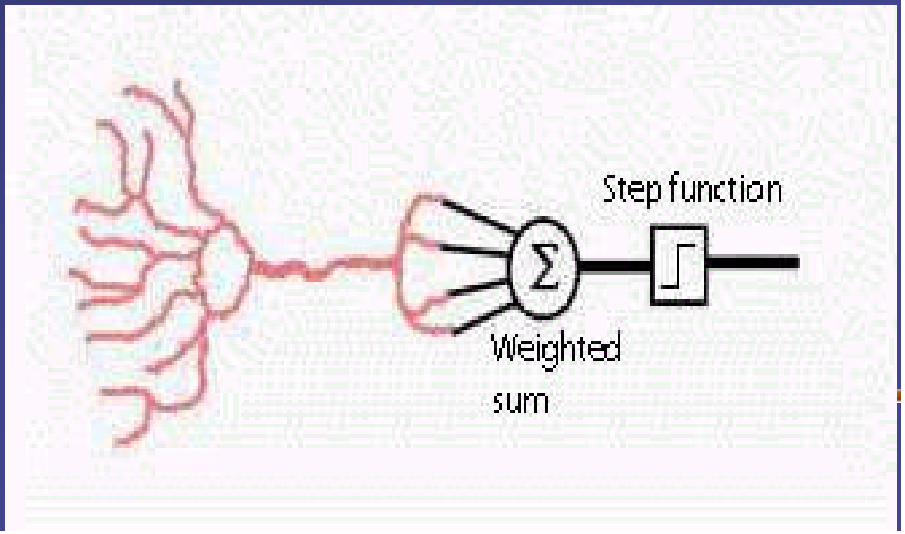


Neuron

- 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

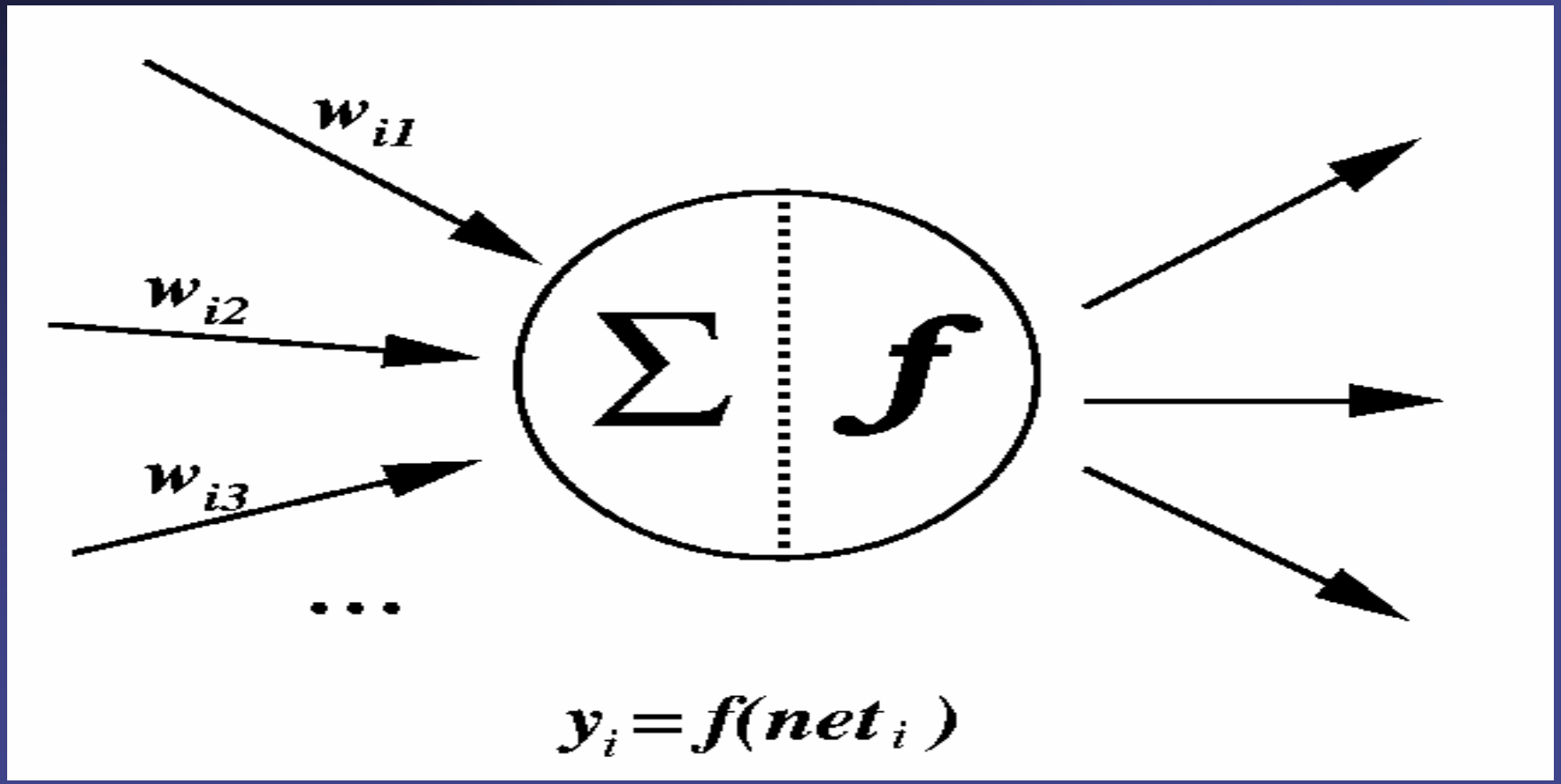
- The nucleus then sums all these new input values which gives us the ***activation***
  - floating point number which can be negative or positive
- If the activation is greater than a threshold value
  - the neuron outputs a signal
  - If the activation is less than
    - 1 the neuron outputs zero
-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



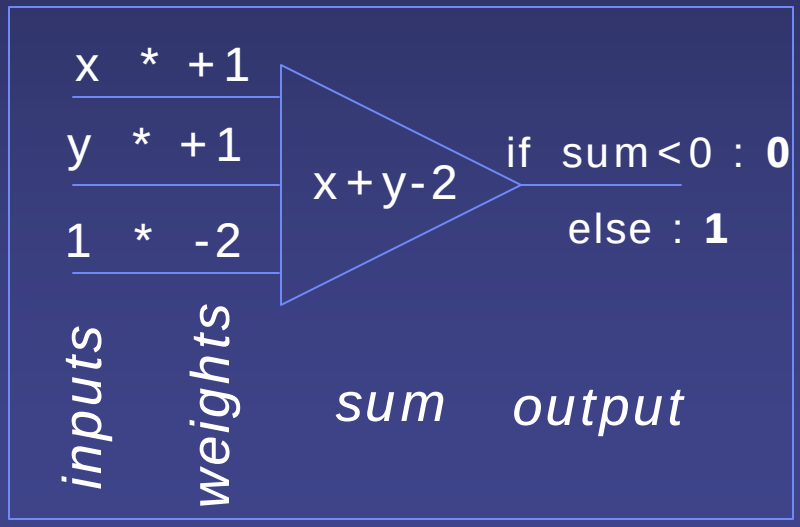
Neural Network History

- **McCulloch-Pitts neurons – 1943**
- **• Widrow's ADALINE and MADALINE – 1960/62**
- **• Nilsson's "Learning Machines" (perceptrons) – 1965**
- **• Minsky and Papert book – 1969**
- **• Kohonen's Self Organized Feature Maps - 1982**
- **• Hopfield's recurrent networks – 1982/84**
- **• Rumelhart's Error Backpropagation – 1986**
- **• Zurada's textbook - 1992**

The Perceptron

History

W.S. McCulloch & W. Pitts (1943). "A logical calculus of the ideas immanent in nervous activity", *Bulletin of Mathematical Biophysics*, 5, 115-137.



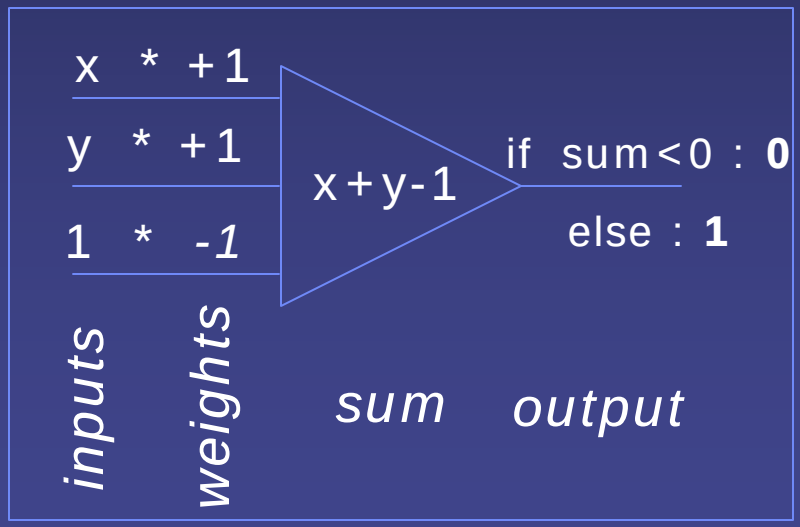
Truth Table for Logical AND

x	y	x & y
0	0	0
0	1	0
1	0	0
1	1	1

inputs *output*

Nervous Systems as Logical Circuits

- Could computers built from these simple units reproduce the computational power of biological brains?
- Were *biological* neurons performing logical operations?



Truth Table for Logical OR

x	y	x | y
0	0	0
0	1	1
1	0	1
1	1	1

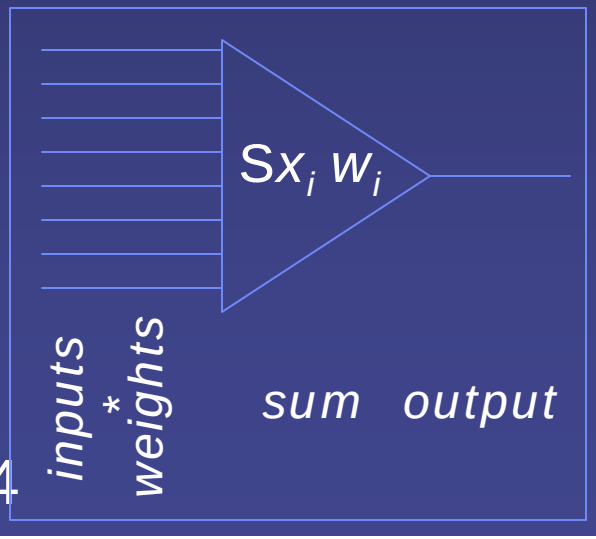
inputs *output*

The Perceptron

- **Rosenblatt's *Perceptron***

could automatically learn to categorize or classify input vectors into types

- **If the sum of the weighted inputs exceeds a threshold**
  - **output 1**
  - **else output -1**

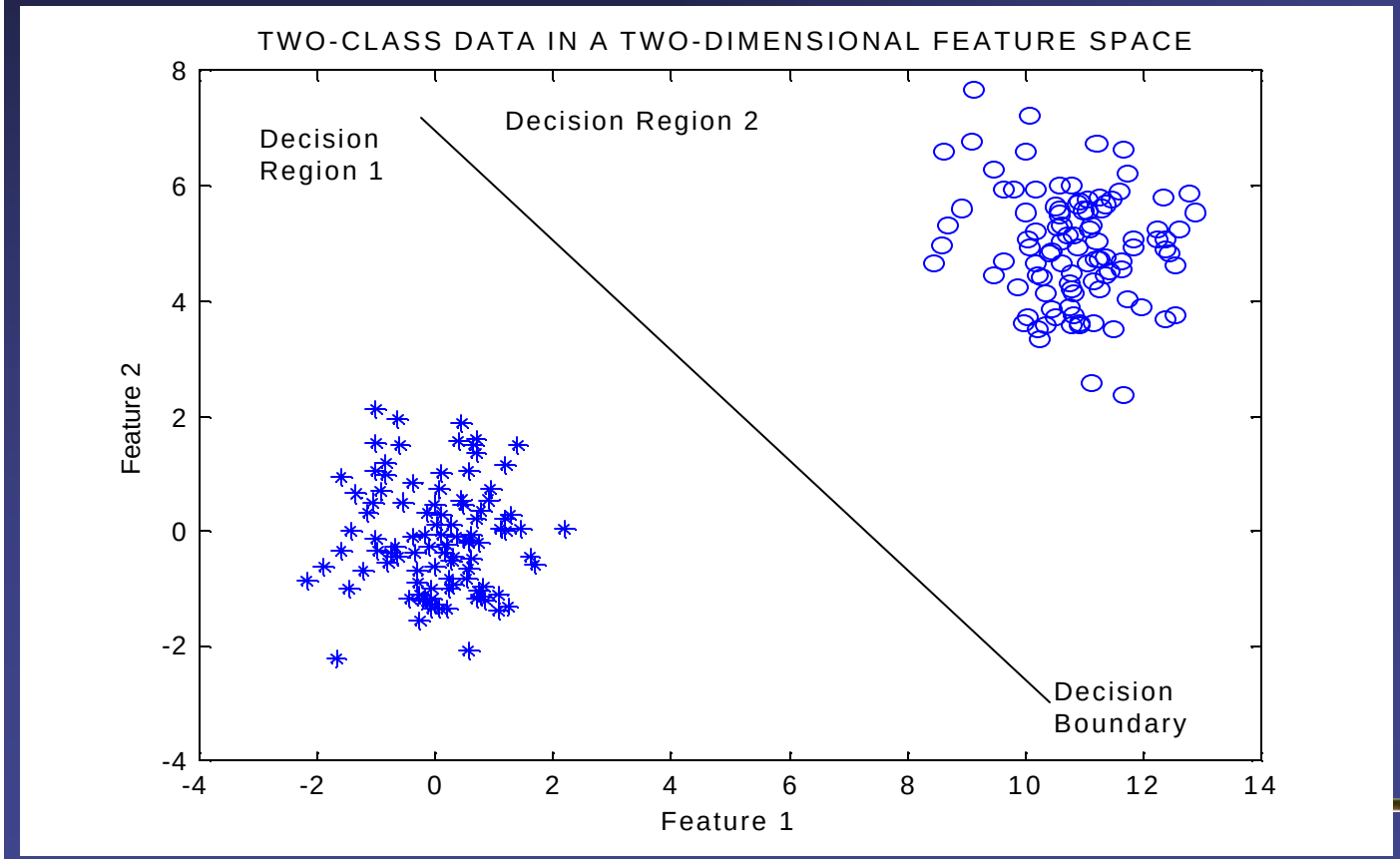


1 if S *input*$_i$ * *weight*$_i$ > *threshold*

-1 if S *input*$_i$ * *weight*$_i$ < *threshold*

Linear discriminant functions

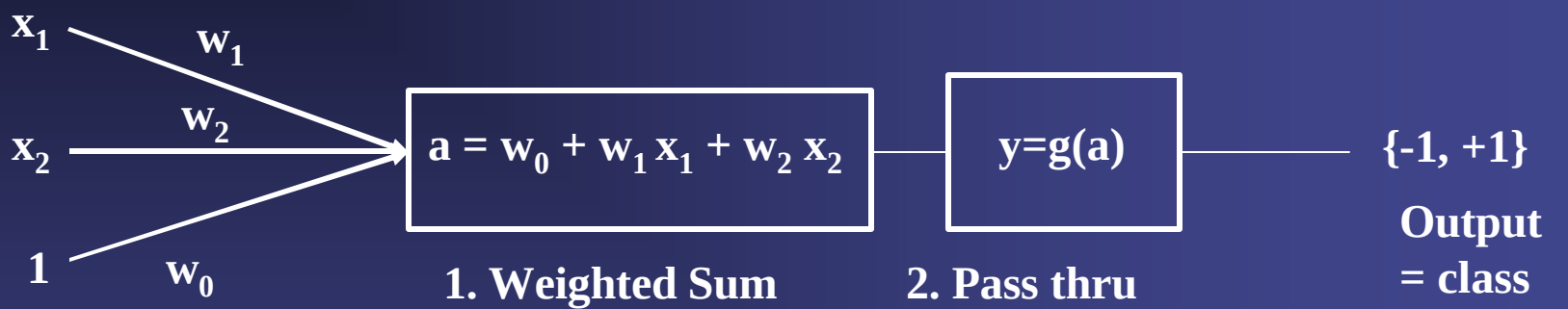
In 2 dimensions the decision boundary is a straight line



Simple form of classifier:

"separate the two classes using a straight line in feature space"

The Perceptron as a Classifier



Output = class decision

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

1. Weighted Sum of the inputs

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

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

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

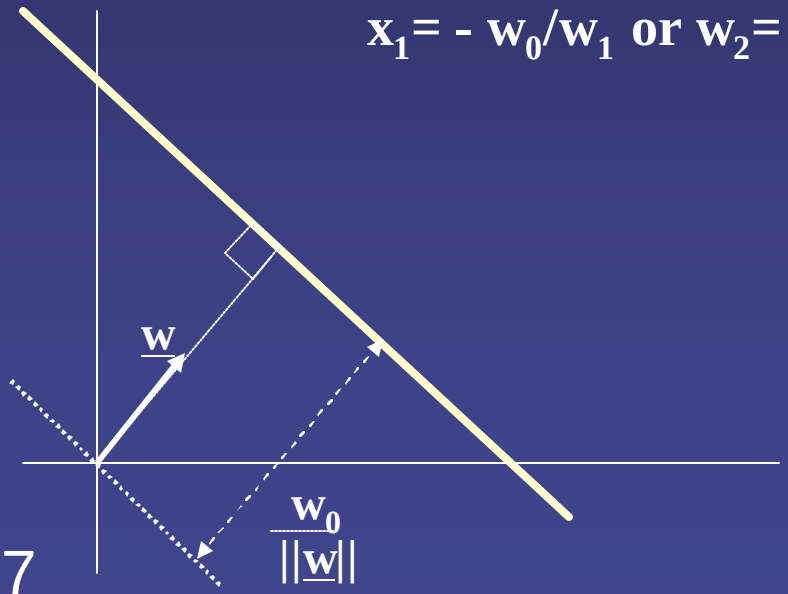
Interpretation of weights

- **Heavy side function is thresholded on 0**
- **Decision boundary is**

$$a = \underline{w} \cdot \underline{x} + w_0 = w_0 + w_1 x_1 + w_2 x_2 = 0$$

$x_2 = -(w_0 + w_1 x_1)/w_2$ unless $w_2 = 0$ when

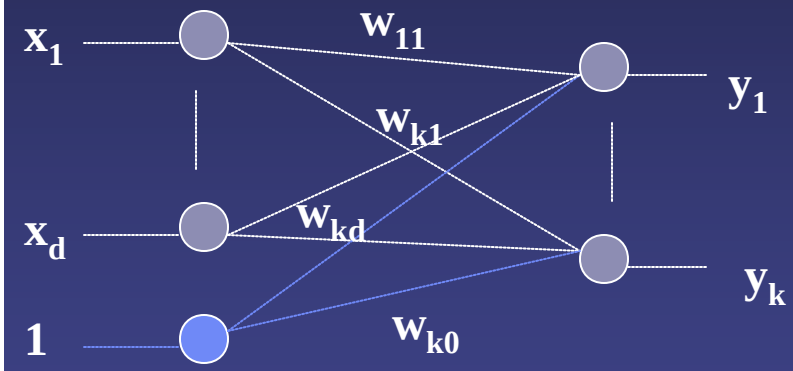
$x_1 = -w_0/w_1$ or $w_2 = w_1 = 0$ -> classification depends on sign of w_0



So perceptron functions as a linear discriminant function

Perceptron

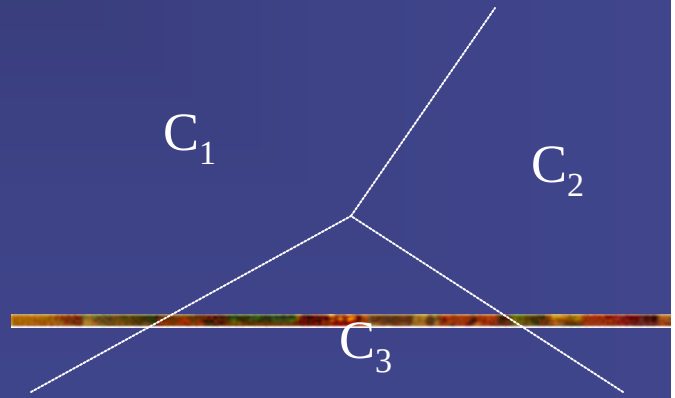
- **Perceptron can be extended to discriminate between k classes by having k output nodes:**
  - **$\underline{x}$ is in class C_j if $y_j(\underline{x}) >= y_k$ for all k**



$$y_j = g(? w_{jk} x_k + w_{k0})$$

Weight to output j from input k is w_{jk}

Resulting decision boundaries divide the feature space into convex decision regions



Sigmoid logistic activation function

- **Sigmoid logistic activation function:**

$$g(a) = 1/(1 + e^{-a})$$

together with data drawn from Gaussian or Bernoulli class-conditional distributions ($P(\underline{x} \mid C_k)$) means that the network outputs can be interpreted as the posterior probabilities $P(C_k \mid \underline{x})$

- **Generalized linear discriminants**

- **Linear discriminants can be made more general by including non-linear functions (*basis functions)* which to transform the input data**

Activation Functions

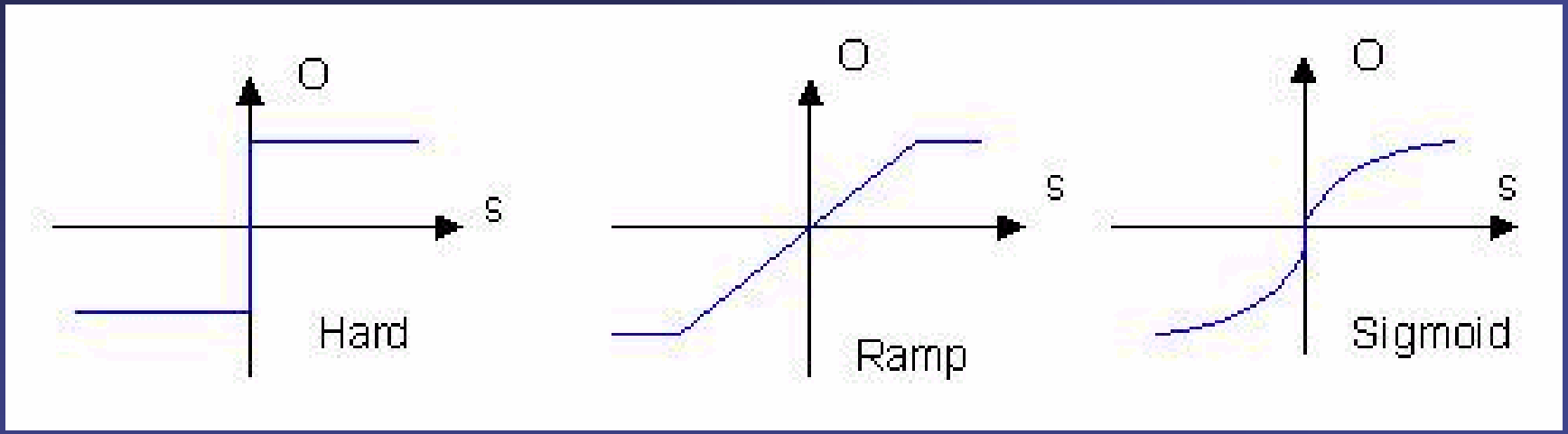


Figure from Answermath.com

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

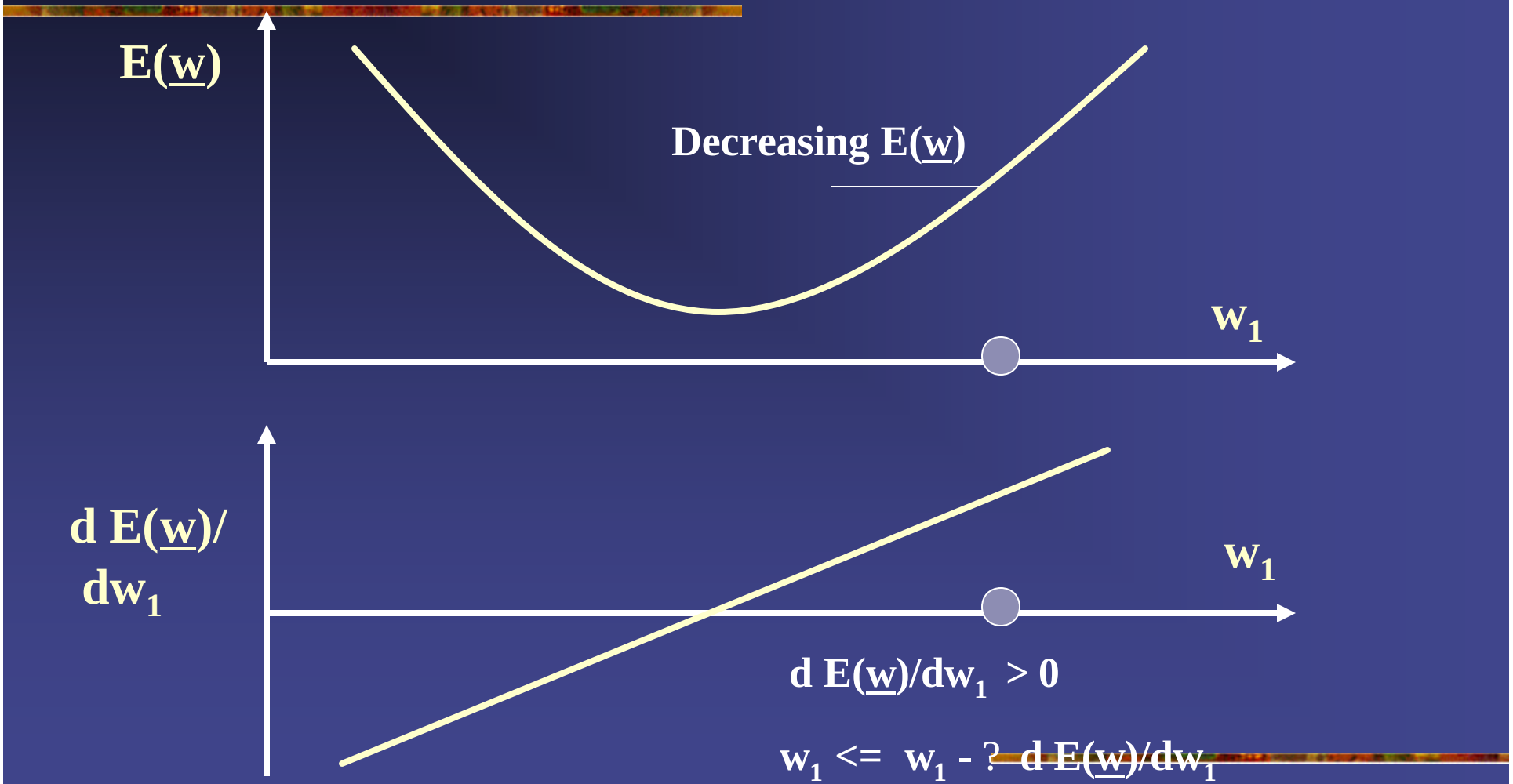
- So

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

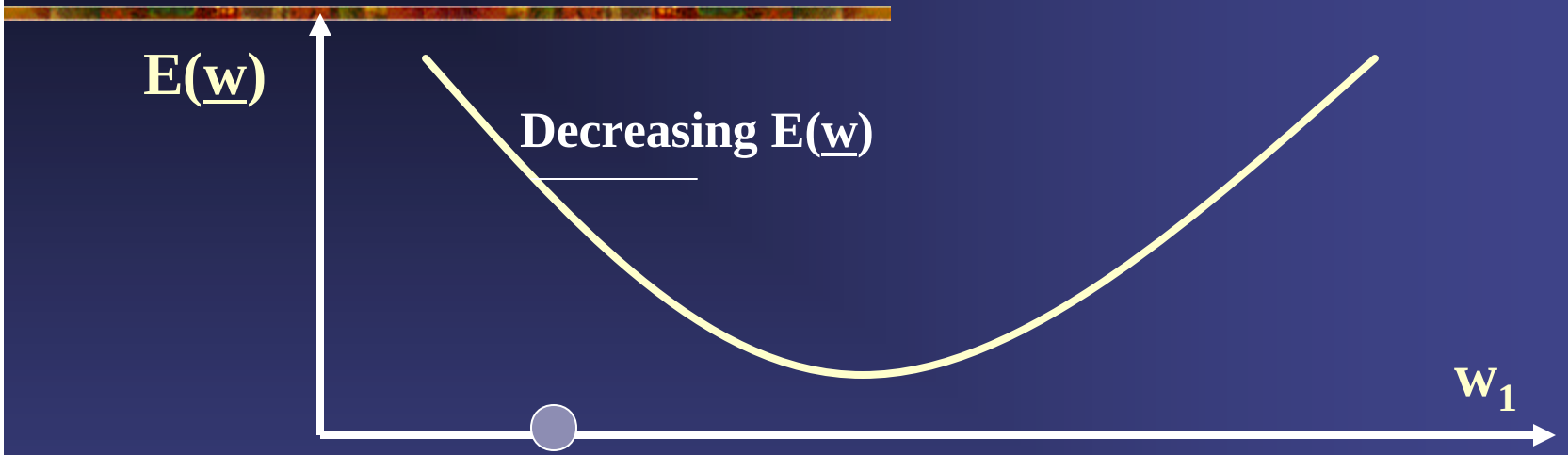
- where $\nabla E(\underline{w})$ is a vector representing the gradient and η is the learning rate (small, positive)

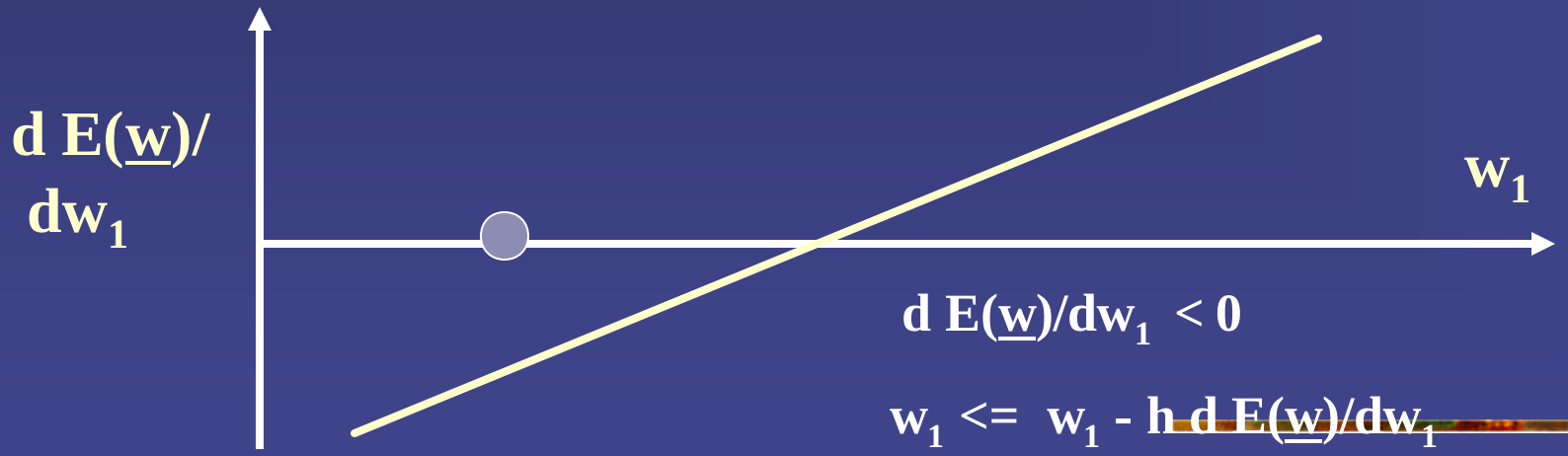
1. Moves downhill in direction $-\nabla E(\underline{w})$ (steepest downhill since $\nabla E(\underline{w})$ is the direction of steepest increase)
2. How far to go at each step is determined by the value of η

Moving Downhill: Move in direction of negative derivative



Moving Downhill: Move in direction of negative derivative



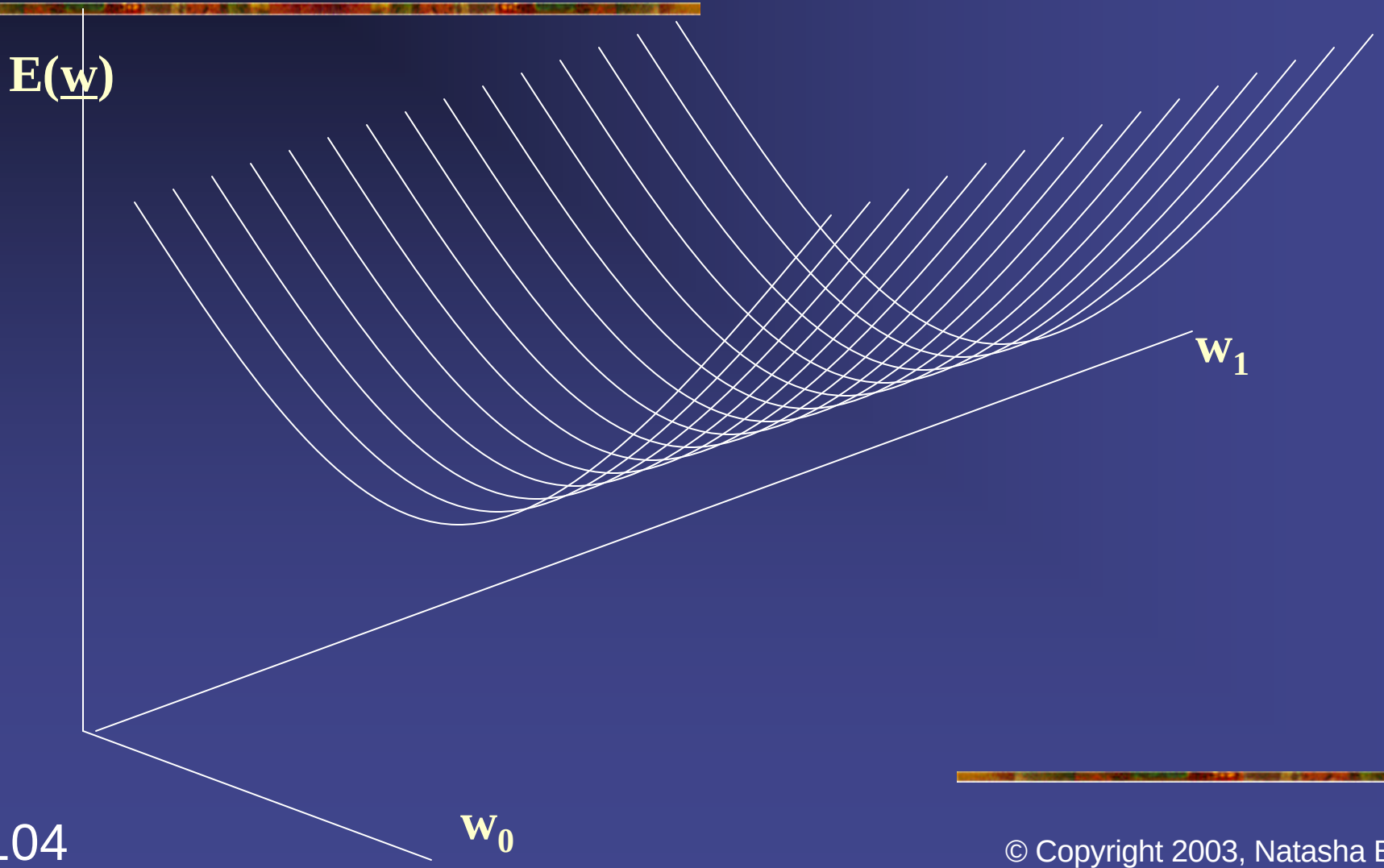


$d\,E(\underline{w})/dw_1 < 0$

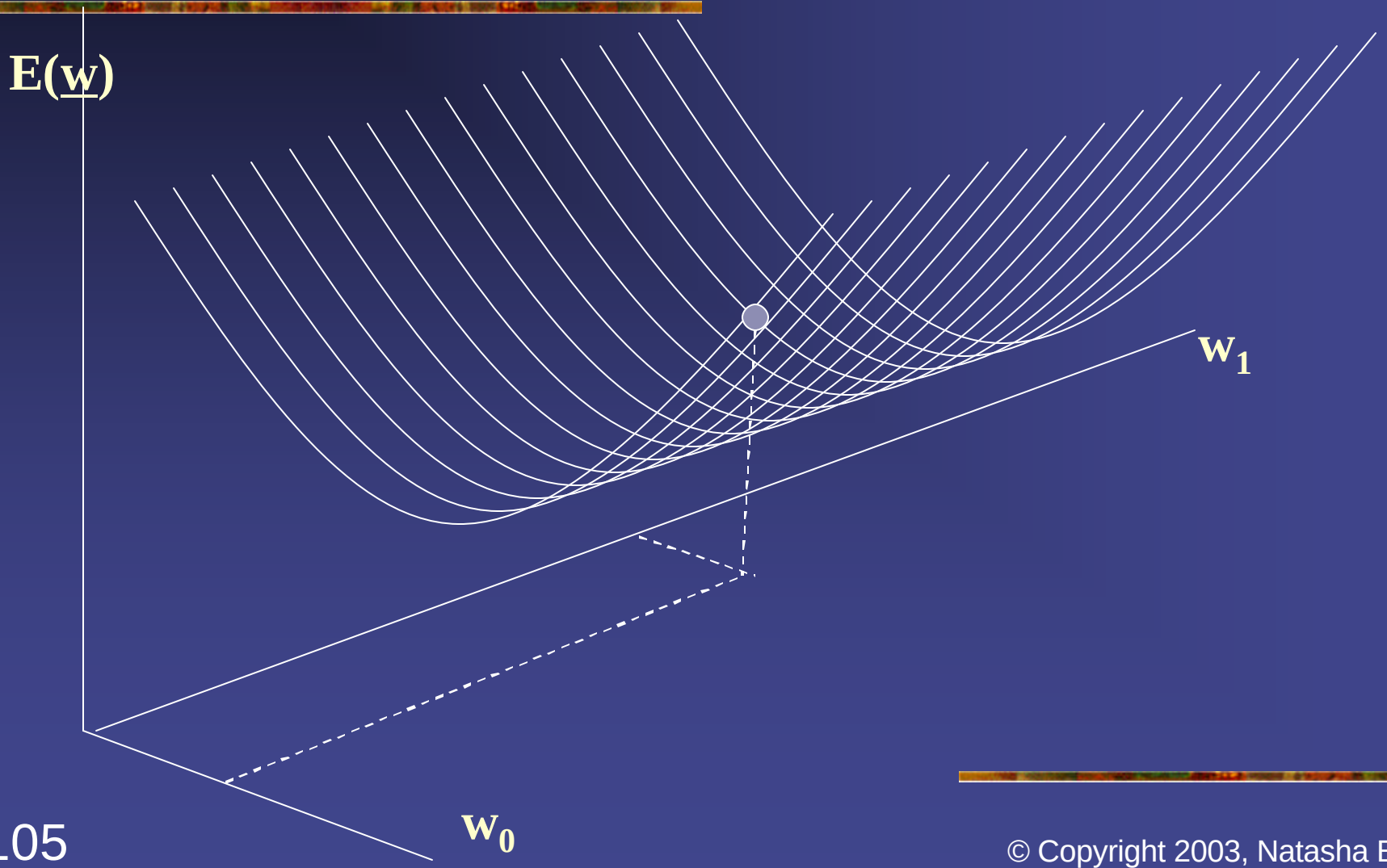
$w_1 <= w_1 - h\,d\,E(\underline{w})/dw_1$

i.e., the rule increases w_1

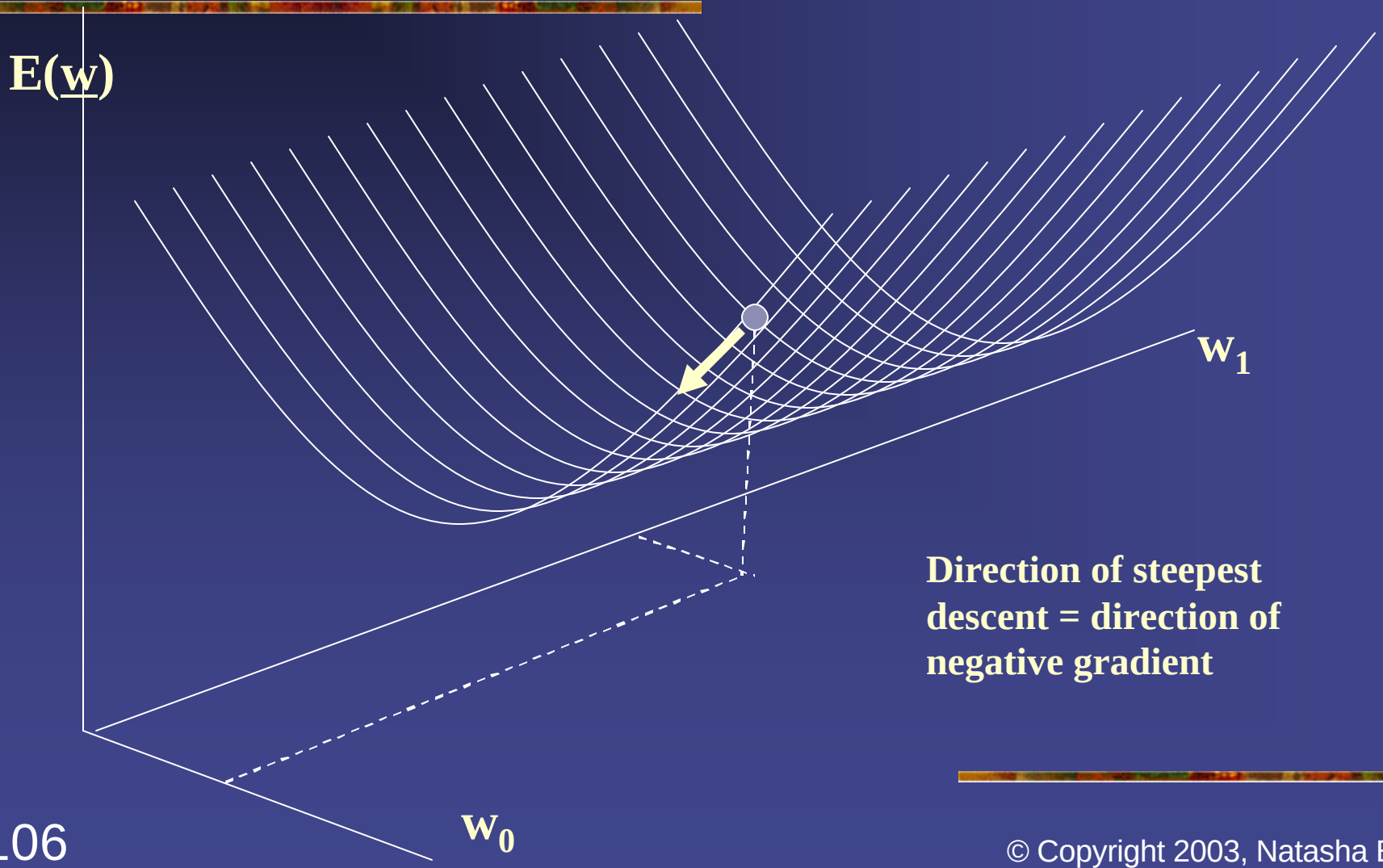
Gradient Descent



Gradient Descent

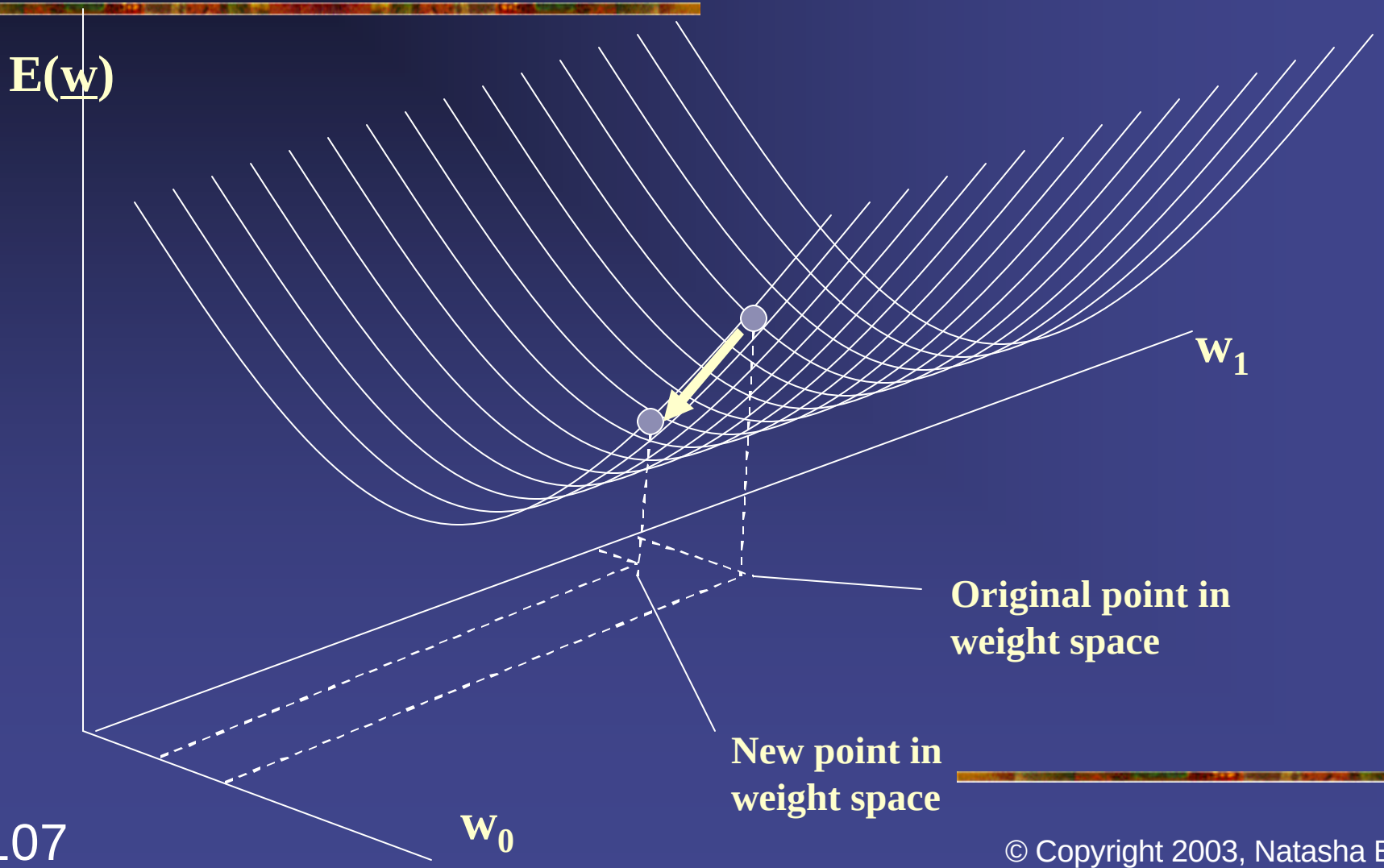


Gradient Descent



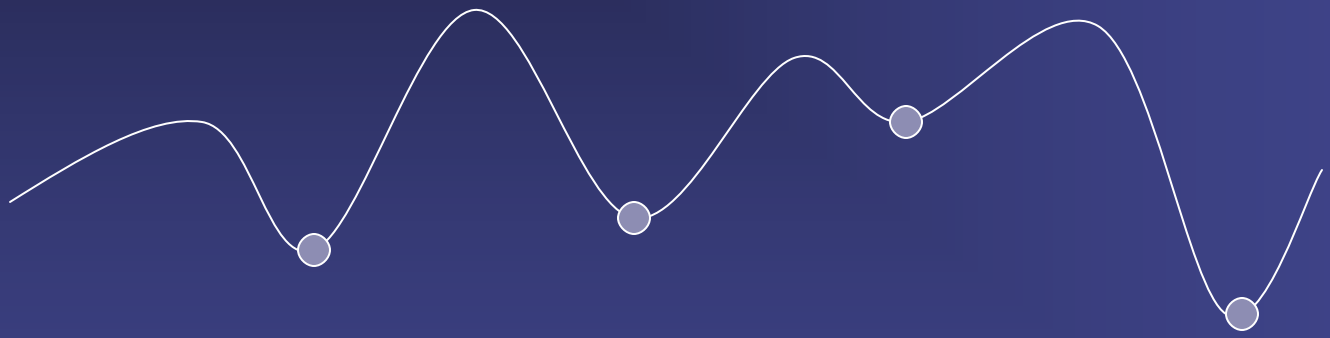
Direction of steepest descent = direction of negative gradient

Gradient Descent



Gradient Descent

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



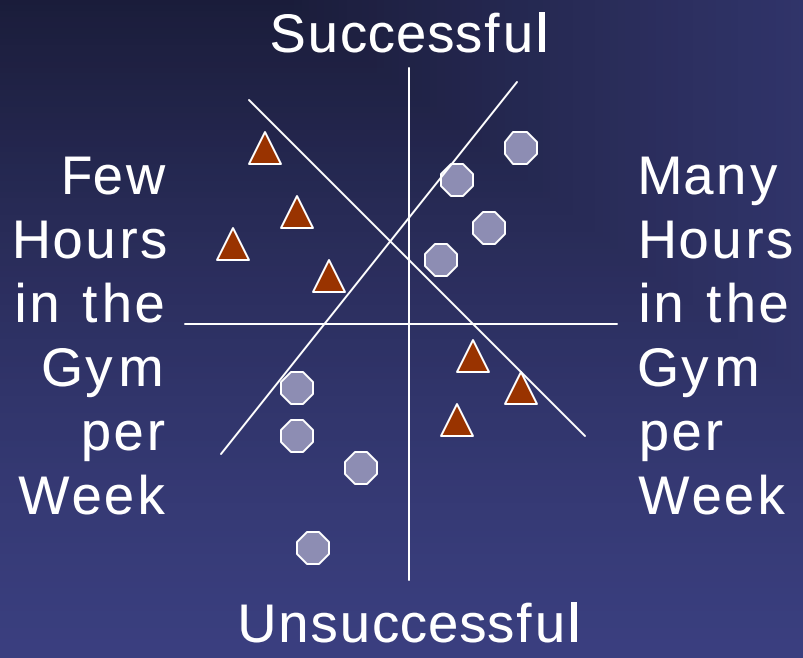
  - can have multiple local minima (note: for perceptron, E(w) only has a single global minimum, so this is not a problem)
  - gradient descent goes to the closest local minimum:
    - solution: random restarts from multiple places in weight space

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)

The Fall of the Perceptron



- Footballers
- Academics

...despite the simplicity of their relationship:

Academics = Successful XOR Gym

In this example, a perceptron would not be able to discriminate between the footballers and the academics…

This failure caused the majority of researchers to walk away

Perceptron Problems

- The simple XOR example masks a deeper problem

- Consider a perceptron classifying shapes as connected or disconnected and taking inputs from the dashed circles in 1

- In going from 1 to 2, change of right hand end only must be sufficient to change classification (raise/lower linear sum thru 0)

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

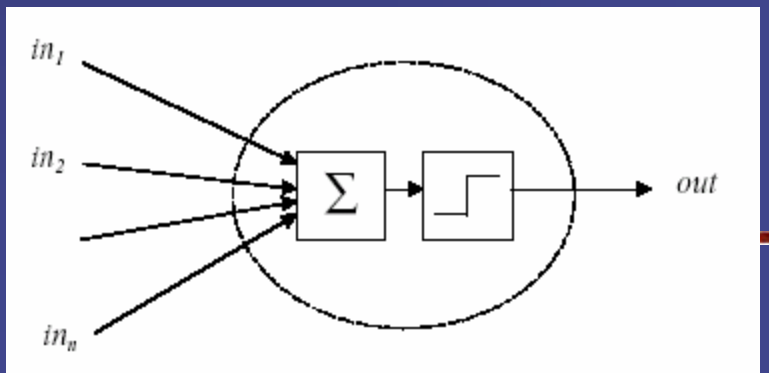
The weight vector

- 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***:

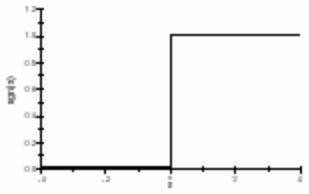
1. A set of synapses (connections) brings in activations from other neurons
2. A processing unit sums the inputs, and then applies a non-linear activation function
3. An output line transmits the result to other neurons



Functions

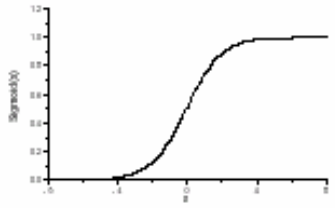
- A function $y = f(x)$ describes a relationship (mapping) from x to y.
- **Example 1** The sign function $\mathrm{sgn}(x)$ is defined as

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



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

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



Vectors

A vector consists of a number of related components, e.g.

$$\boldsymbol{x} = (x_1, x_2, x_3, \ldots, x_n) \quad , \quad \boldsymbol{y} = (y_1, y_2, y_3, \ldots, y_m)$$

Vector components can be added to give a scalar (number), e.g.

$$s = x_1 + x_2 + x_3 + \ldots + x_n = \sum_{i=1}^{n} x_i$$

Two vectors of the same length may be added to give another vector, e.g.

$$\boldsymbol{z} = \boldsymbol{x} + \boldsymbol{y} = (x_1 + y_1, x_2 + y_2, \ldots, x_n + y_n)$$

Two vectors of the same length may be multiplied to give a scalar, e.g.

$$p = \boldsymbol{x}.\boldsymbol{y} = x_1 y_1 + x_2 y_2 + \ldots + x_n y_n = \sum_{i=1}^{n} x_i y_i$$

Matrices

The concept of vectors can be extended to more dimensions/indices. For two indices we have matrices, e.g. an $m \times n$ matrix

$$\mathbf{w} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{1n} \\ \vdots & \vdots & & \vdots \\ w_{m1} & w_{m1} & \cdots & w_{mn} \end{pmatrix}$$

Matrices of the same size can be added or subtracted component by component.

An $m \times n$ matrix **a** can be multiplied with an $n \times p$ matrix **b** to give an $m \times p$ matrix **c**, such that in terms of components:

$$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}$$

A n component vector can be regarded as a $1 \times n$ or $n \times 1$ matrix

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

What can a perceptron do?

- Perceptron calculates the weighted sum of the input values
- Two input values, x and y for a certain perceptron P
- Let the weights for x and y be A and B for respectively
- Weighted sum could be represented as
  - A x + B y

What can a perceptron do?

- Since the perceptron outputs an non-zero value only when the weighted sum exceeds a certain threshold C

- Output of P = { 1 if A x + B y > C
  { 0 if A x + B y < = C

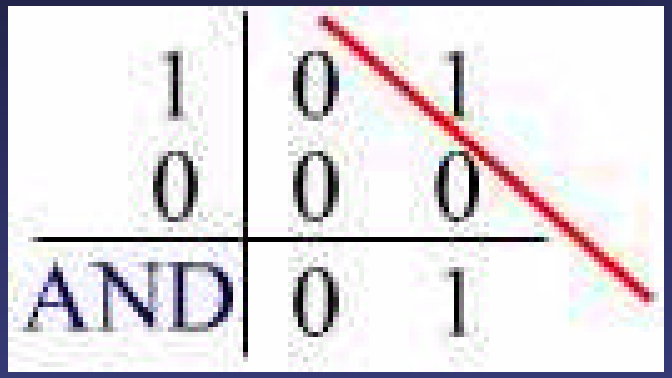
Linearly Separable Regions

- Since A x + B y > C and A x + B y < C are the two regions on the xy plane separated by the line A x + B y + C = 0
- Consider the input (x, y) as a point on a plane
  - perceptron actually tells us which region on the plane to which this point belongs
- Such regions separated by a single line are called linearly separable regions

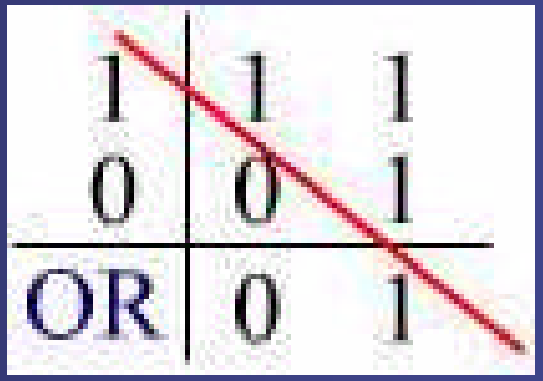
Linearly Separable Regions Example

- This result is useful because some logic functions
  - such as the boolean AND, OR and NOT operators
  - are linearly separable
- They can be performed using a single perceptron

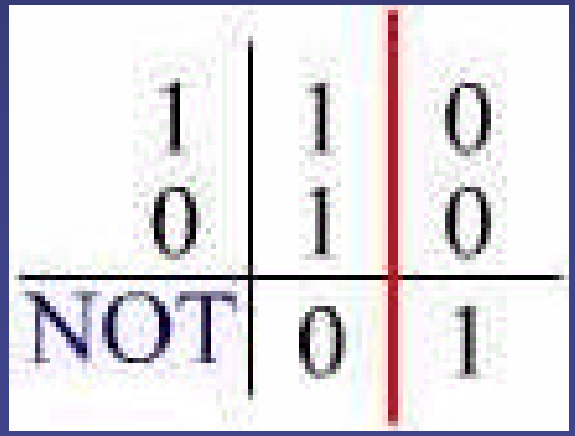
Linearly Separable Logic Functions



	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

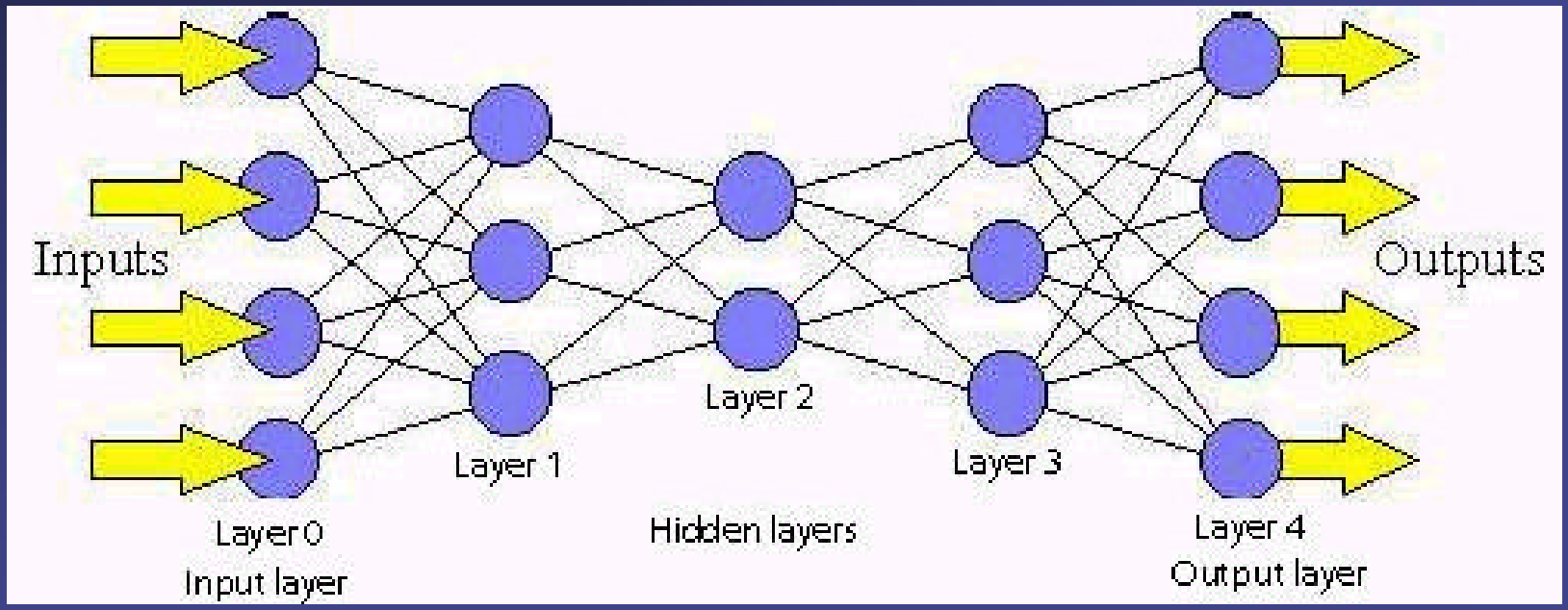
1	1	0
0	0	1
XOR	0	1

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

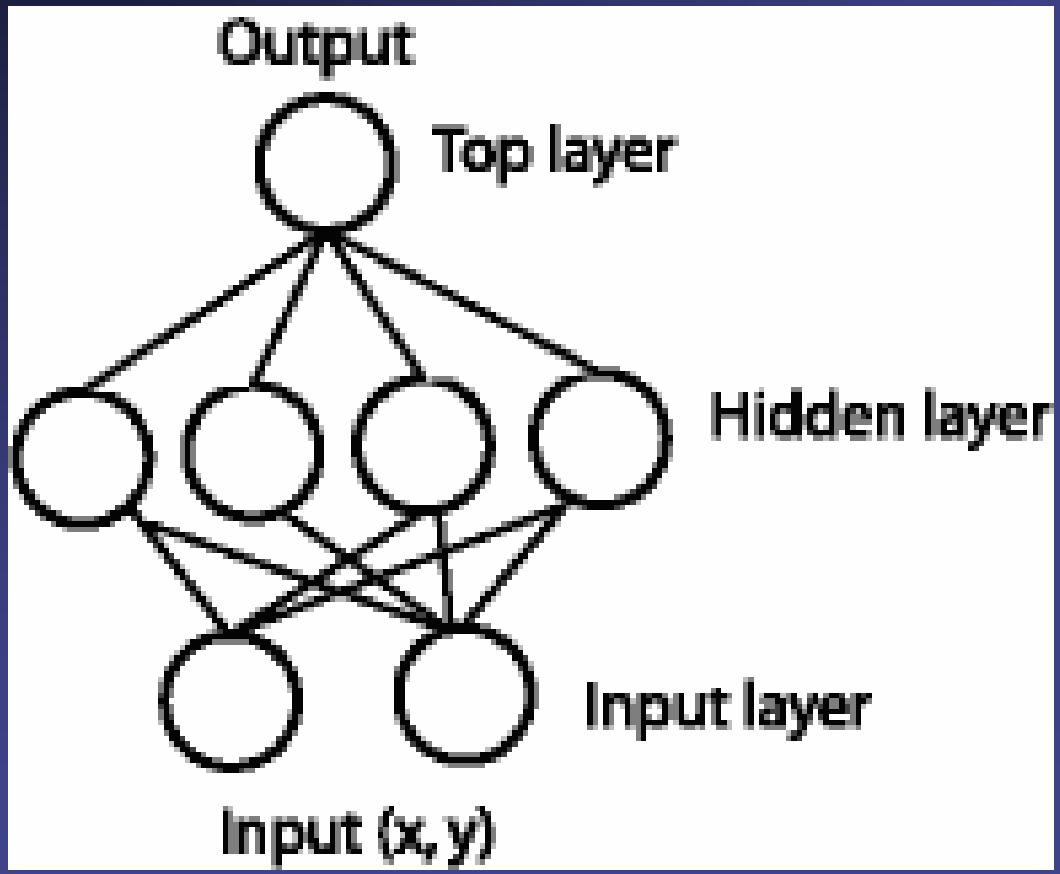
- Feed-Forward network



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?

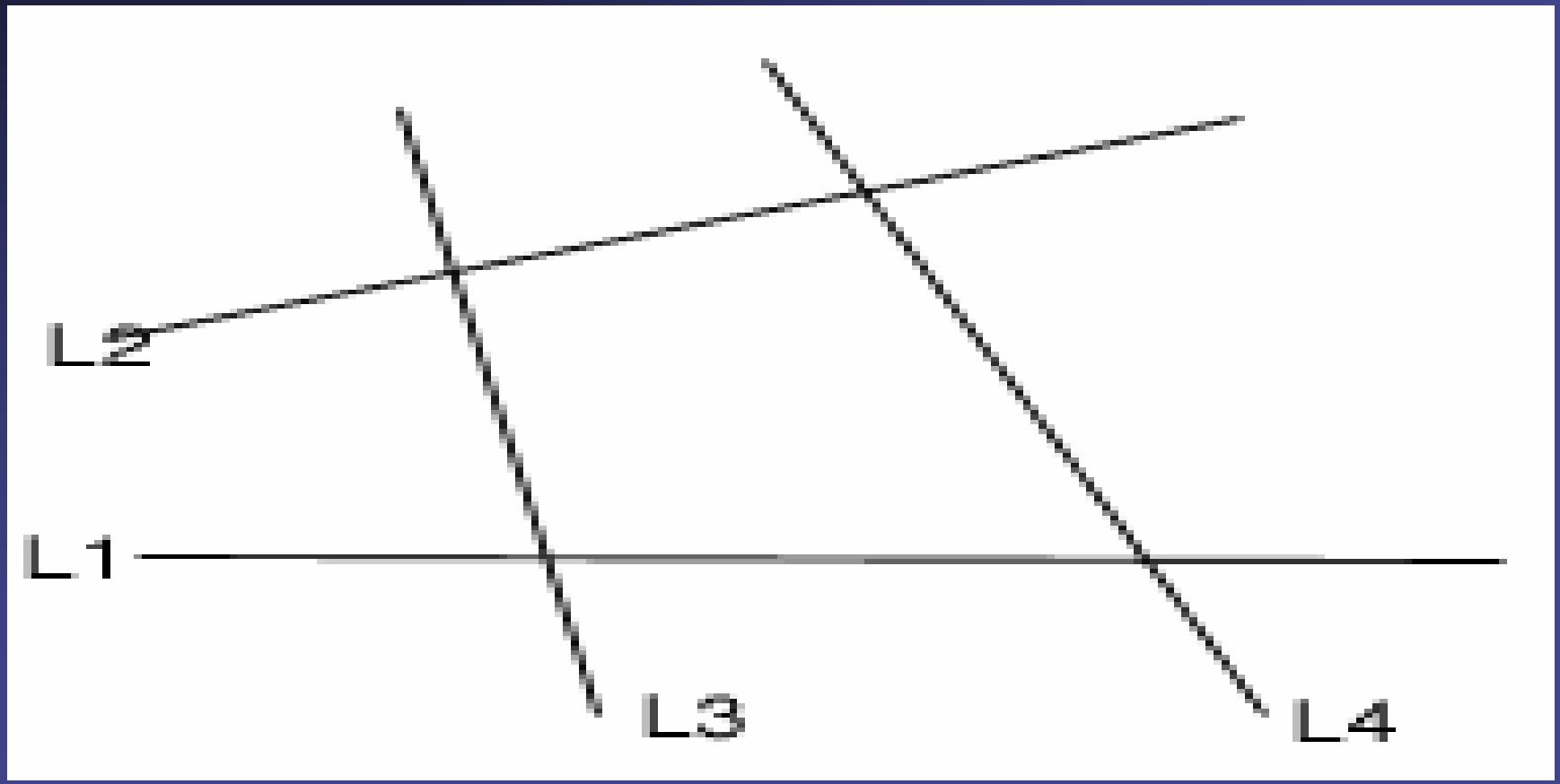
A feed-forward network with one hidden layer



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

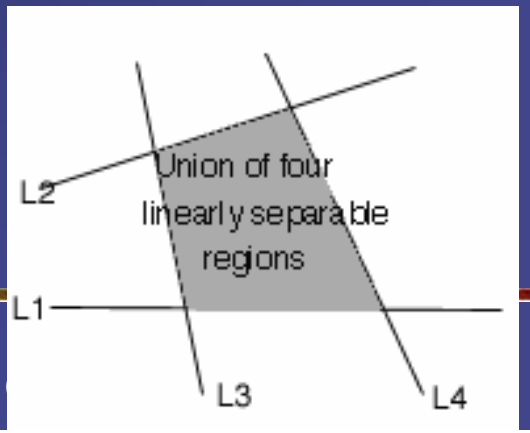
Four lines each dividing the plane into 2 linearly separable regions



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



Feed-forward networks

- By varying the
  - number of nodes in the hidden layer
  - number of layers
  - number of input and output nodes
- We can perform classification of points in arbitrary dimension into an arbitrary number of groups
- Feed-forward networks are commonly used for classification

Learning in feed-forward networks

- Supervized 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

Example

- **Points (1, 2) and (1, 3) belonging to group 0**
- **points (2, 3) and (3, 4) belonging to group 1**
- **(5, 6) and (6, 7) belonging to group 2**
- **Then for a feed-forward network with 2 input nodes and 2 output nodes, the training set would be:**

{ i = (1, 2) , o =(0, 0)
i = (1, 3) , o = (0, 0)
i = (2, 3) , o = (1, 0)
i = (3, 4) , o = (1, 0)
i = (5, 6) , o = (0, 1)
i = (6, 7) , o = (0, 1) }

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 the following formula:

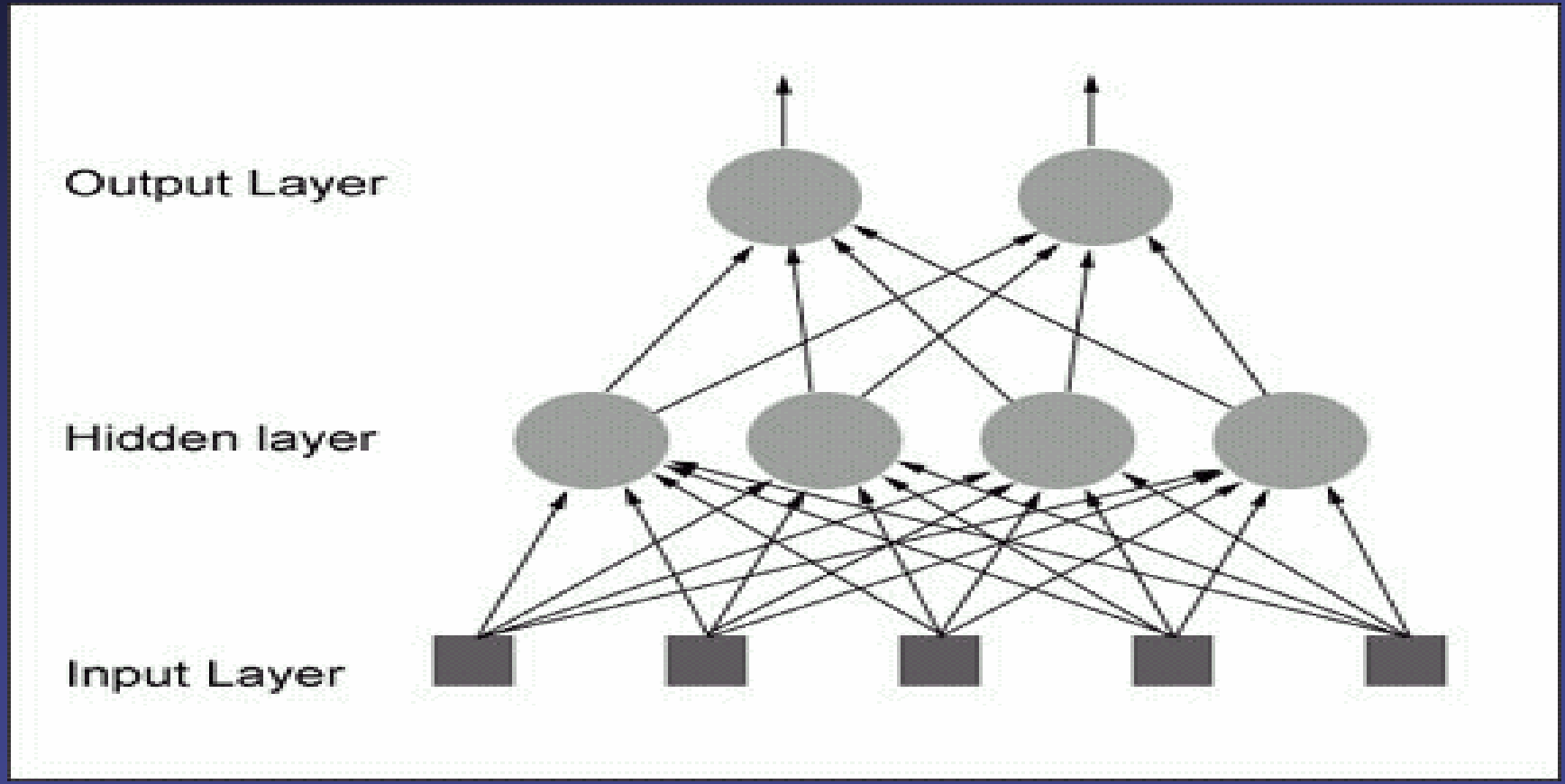
$$w_{new} = w_{old} - n \frac{\delta Err}{\delta w}$$

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

Feedforward Network

- In order to use the neuron we link several of these neurons up in some way
- One way of doing this is by organizing the neurons into a design called a *feedforward network*
- Neurons in each layer feed their output forward to the next layer until we get the final output from the neural network

Feedforward Network



Feedforward Network

- Each input is sent to every neuron in the hidden layer and then each hidden layer's neuron's output is connected to every neuron in the next layer
- There can be any number of hidden layers within a feedforward network

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 Methods

- This type of training is called ***supervised learning*** and the data we feed it is called a ***training set***
- There are many different ways of adjusting the weights - ***Learning rules***
- The most common for this type of problem is called ***backpropagation***
- ***Others are : Hebb's rule, Delta rule***

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

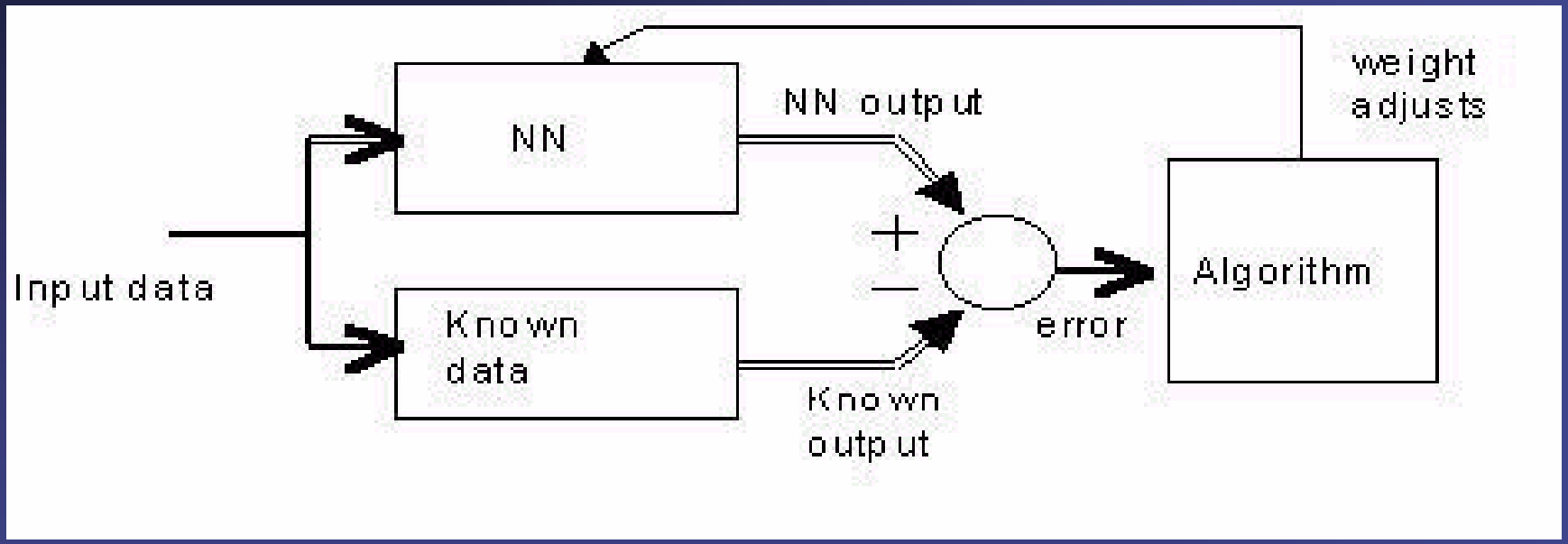


Figure from Answermath.com

Executing: Final behavior of the NN

- Have a trained network
- Feed the system with new input data
  - and NN will produce a reasonable or intelligent response in its outputs
- Application may be
  - stock value prediction for certain circumstances
  - risk for a new loan application
  - local weather warning
  - identification of a person in a new picture

Applets:

- http://euclid.ii.metu.edu.tr/emkodtu/met901/lectures/ch1/sec2.html
- Artificial neuron Applet:
- http://home.cc.umanitoba.ca/~umcorbe9/neuron.html
- http://home.cc.umanitoba.ca/~umcorbe9/anns.html
- Handwriting recognizer:
- http://members.aol.com/Trane64/java/JRec.html
- XOR Applet: http://www.patol.com/java/NN/index.html
- Ball balancing:
- http://neuron.eng.wayne.edu/bpBallBalancing/ball5.htm

- http://www.qub.ac.uk/mgt/intsys/feedforw.html