



Introduction to Neural Networks

Johns Hopkins University
Engineering for Professionals Program
605-447.71/625-438.71

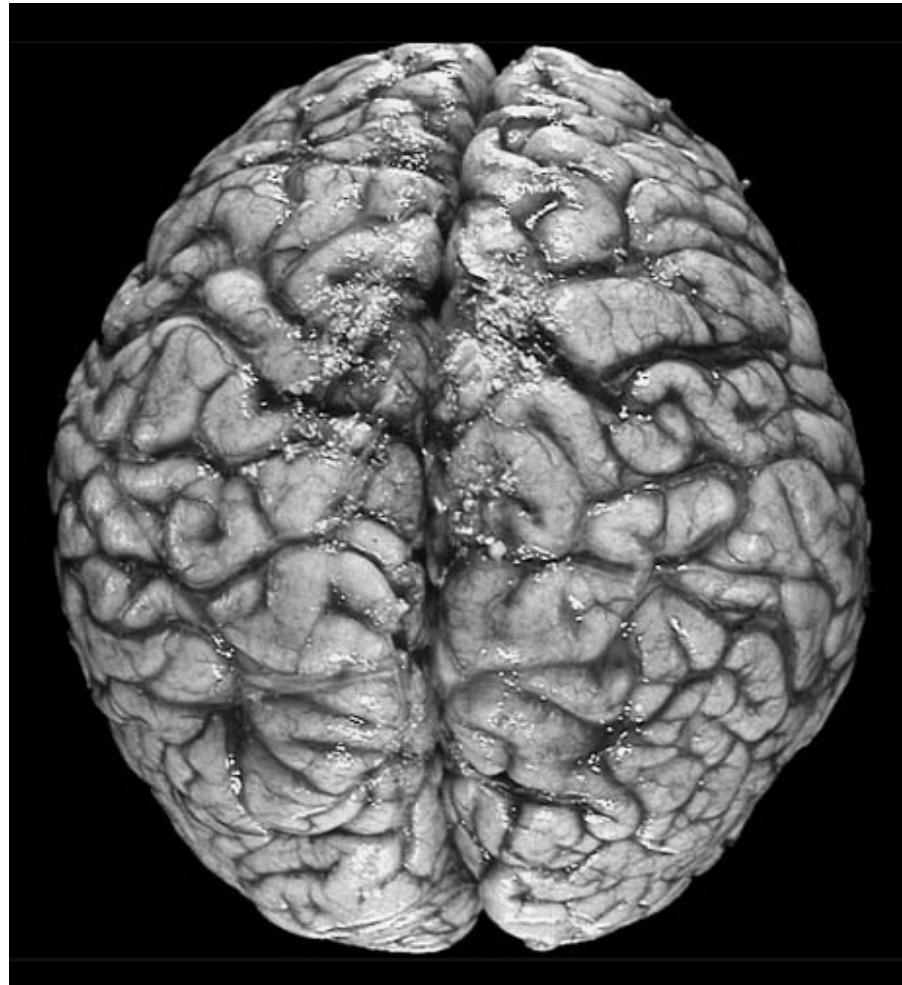
Dr. Mark Fleischer

Copyright 2013 by Mark Fleischer

Module 1.1: Course Overview



My Brain





The Hemispheres of the Brain

- Brain lateralization: certain functions seem to be specialized into one hemisphere. Some interesting symmetries!

LEFT BRAIN FUNCTIONS

Small Picture
Verbal Communication
Small Muscle Control
Intelligence Quotient
Word Reading
Math Calculations
Processing Information
Conscious Actions
Positive Emotions
Receiving Auditory Input
Linear and Logical Thinking
Curious and Impulsive Actions
Like Routine/Sameness
Activates Immunity

RIGHT BRAIN FUNCTIONS

Big Picture
Nonverbal Communication
Large Muscle Control
Emotional Quotient
Comprehension
Math Reasoning
Interpreting Information
Unconscious Actions
Negative Emotions
Interpreting Auditory Input
Gets Abstract Concepts
Cautious and Safe Actions
Likes Newness, Novelty
Suppresses Immunity



What are Neural Networks?

- Originated as mathematical models of biological neurons.
- Evolved into large scale, massively parallel collections of computing entities.
- Single neurons are *relatively simple*.
- Many interconnected neurons are **complex**.
- Simple computation.
- Complex computation.



A bit of History

- McCulloch and Pitts (1943) devised **neural network models** in the context of automata and computation;
- Donald Hebb (1940) established concepts now referred to as *Hebbian learning* covered later in the course.
- Alan Turing (1940) developed the foundation of **computing theory** with his Turing machine.
- John von Neumann (1940) devised other **computing paradigms**.
- Stanislaw Ulam (1940) worked on **cellular automata** and computing theory.
- Developments in electronics, nuclear science, radar, computers and cryptanalysis.

A great deal of intellectual ferment during World War II.



What are Neural Networks?

- Neural networks are an attempt to mimic the behavior, in some respects, of the brain—a network of nerve tissues.
- Neural networks are **mathematical models** inspired by biological neurons.
- Each nerve or neuron is considered to be a relatively simple device.
- When combined with other similar neurons in a network, they become an exceedingly complex system that seem to give rise to '**emergent**' behavior.
- Mathematical capabilities to perform **logic**.
- Mathematical properties of **dynamical systems**.



Why do people study them?

- Curiosity about how brains work.
- Curiosity about consciousness, thinking and computing.
- Curiosity about ‘complex systems’ and ‘emergent’ behavior.
- Many recent advances and insights into ‘network theory’.
 - Keywords: clustering, network analysis, pathways, degrees of contact, data-mining, complexity theory ...
 - Maximal Information-based Nonparametric Exploration---data mining.



But, What are Neural Networks?!

- Essentially, they are simple, **mathematical models** of neurons.
- Have roots in
 - Neuroscience
 - Mathematics and statistics
 - Physics
 - Computer science
 - Engineering
- Requires use of **mathematical tools** to study and analyze them and **gain insights into what they CAN and CANNOT do.**
 - Linear algebra, Calculus, Vector Calculus, metric spaces and dynamical systems, optimization methods.



Application Areas

- Modeling systems.
- Time series processing.
 - Forecasting, prediction.
- Signal processing.
 - Filtering, noise reduction...
- Pattern recognition.
 - Speech recognition,
 - Natural language processing,
 - Biometrics,
 - Classification
- Control system engineering.



From Simple to Complex

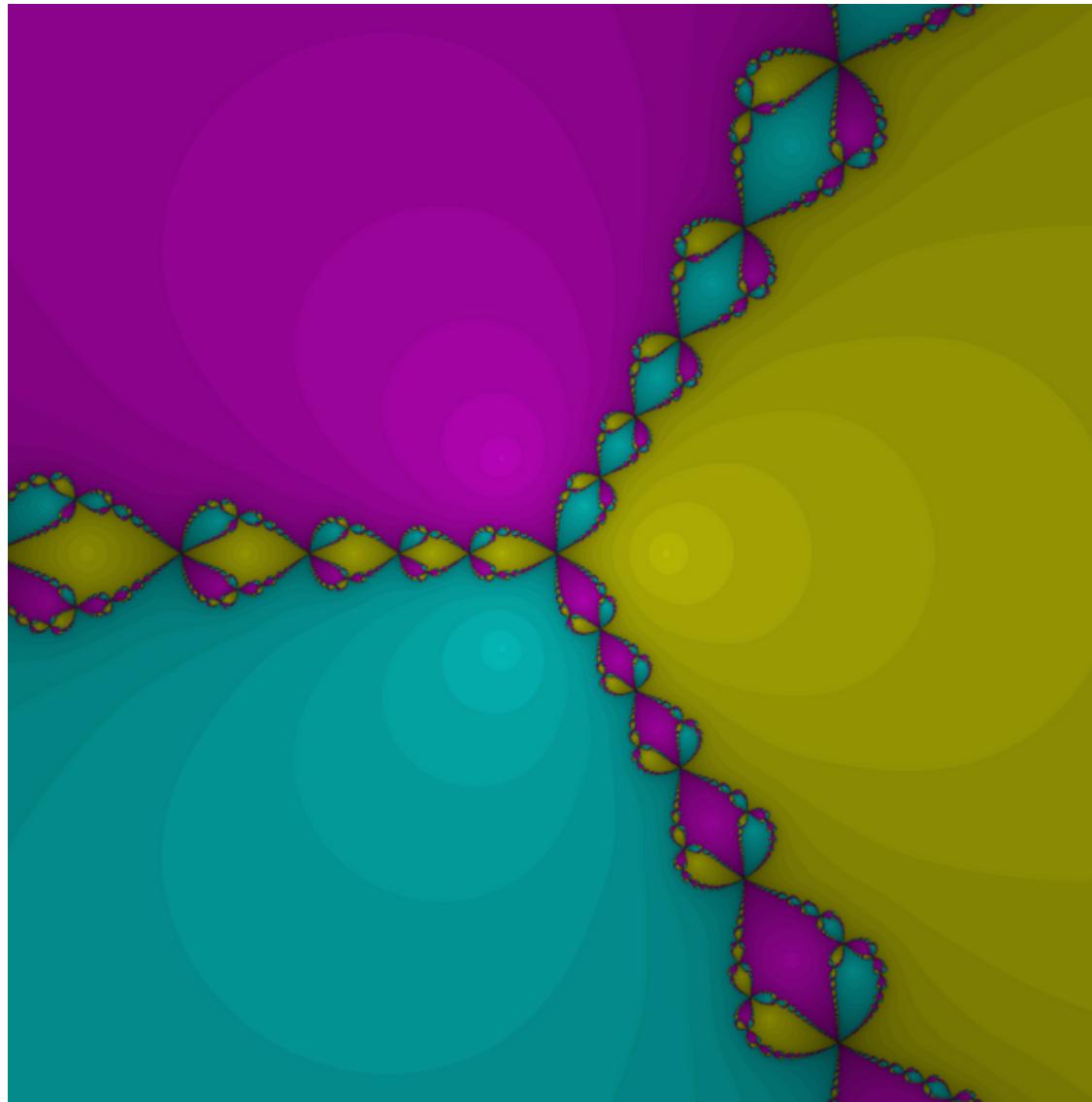
- Let's start with a simple polynomial --- how complex can that be? $f(z) = z^3 - 1$
- Let's explore something where a mathematical quantity changes over time according to some formula.



Newton's Method

- In complex plane, what are the roots of:

$$f(z) = z^3 - 1$$





A Simple Dynamical System

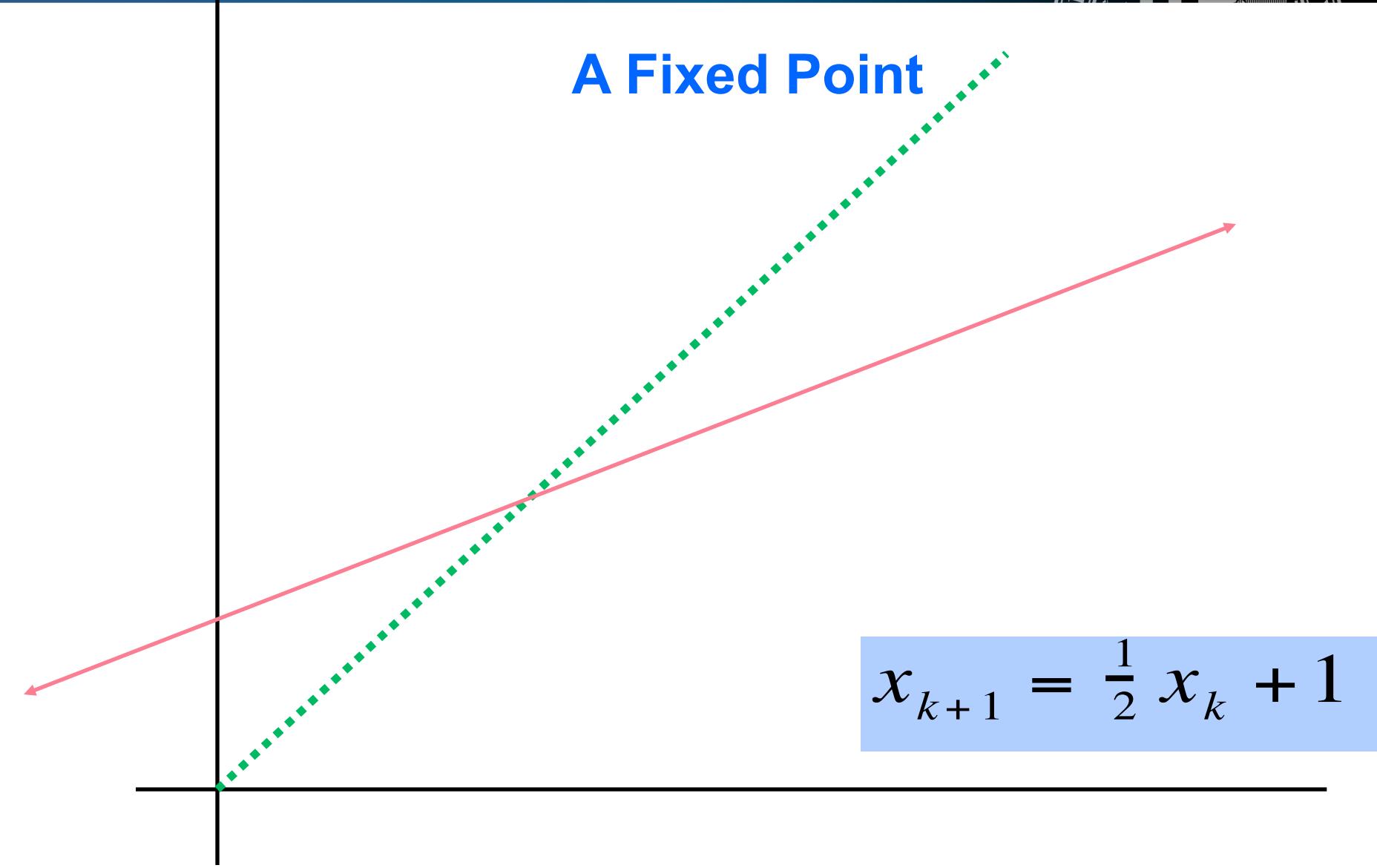
$$x_{k+1} = f(x_k)$$

$$\lim_{k \rightarrow \infty} x_k = x^* \implies x^* = f(x^*)$$

A Fixed - Point



A Fixed Point



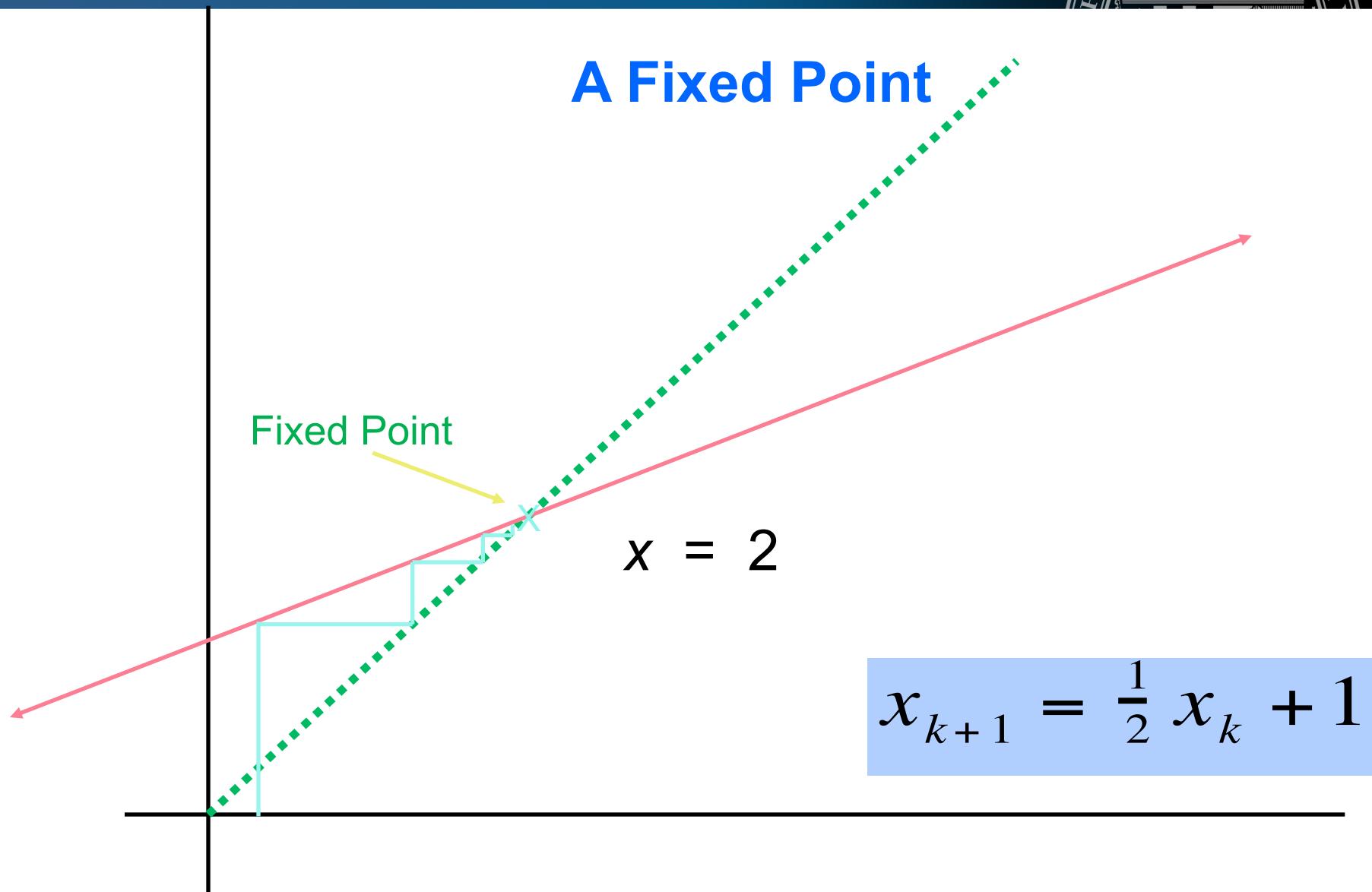


A Fixed Point

Fixed Point

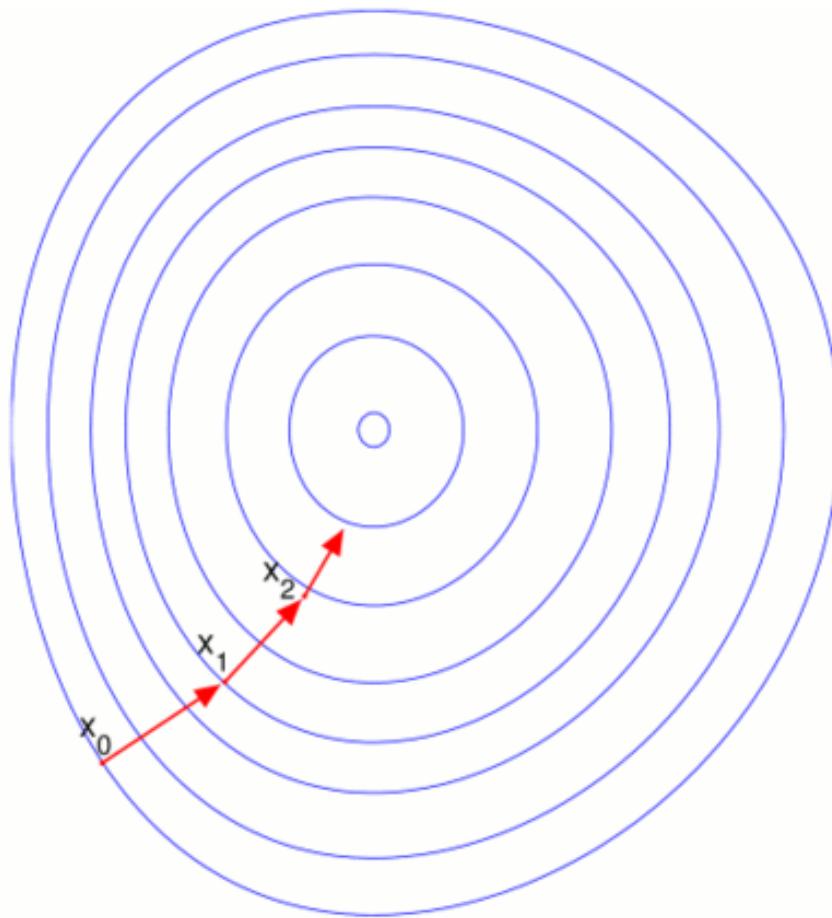
$$x = 2$$

$$x_{k+1} = \frac{1}{2} x_k + 1$$



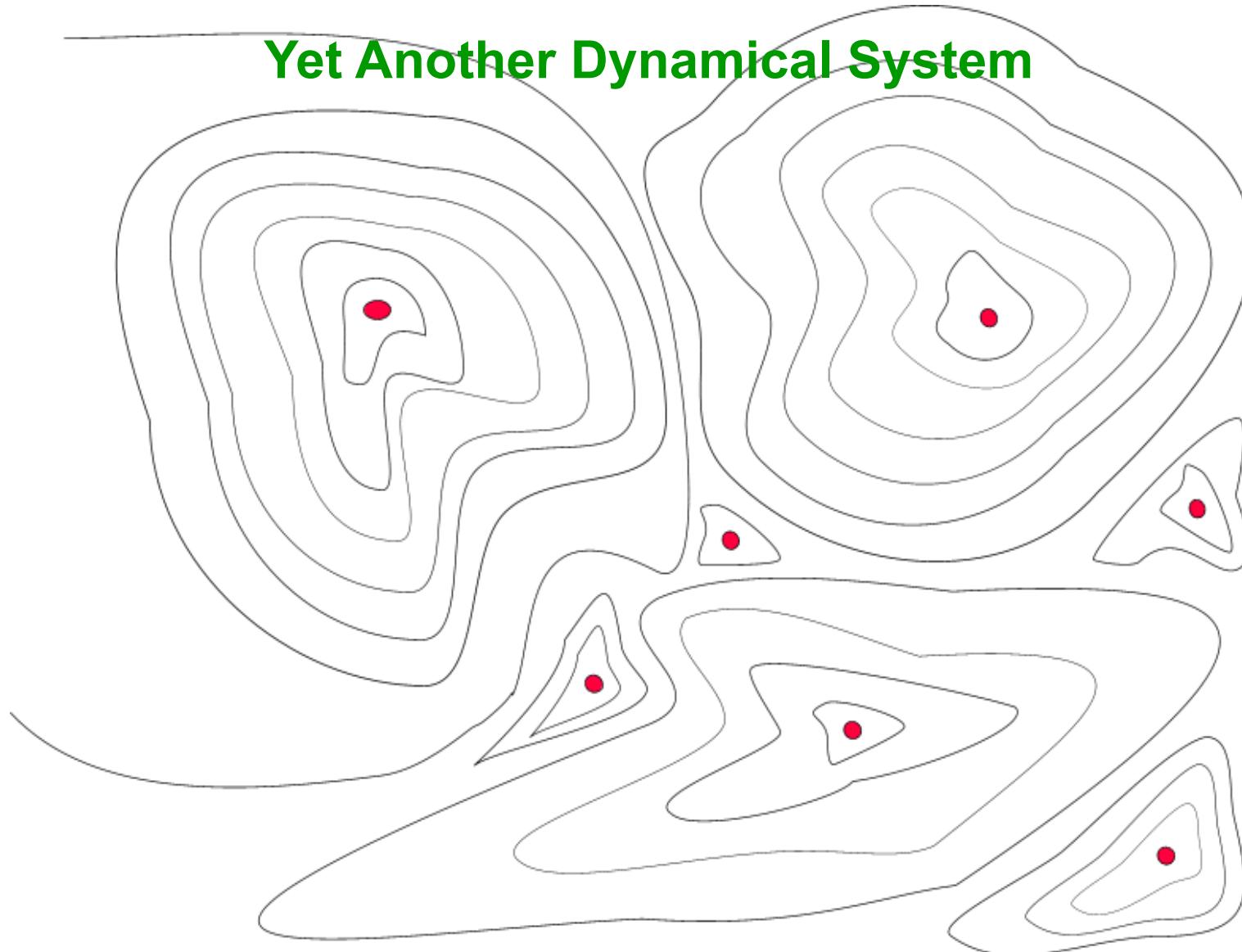


Another Dynamical System



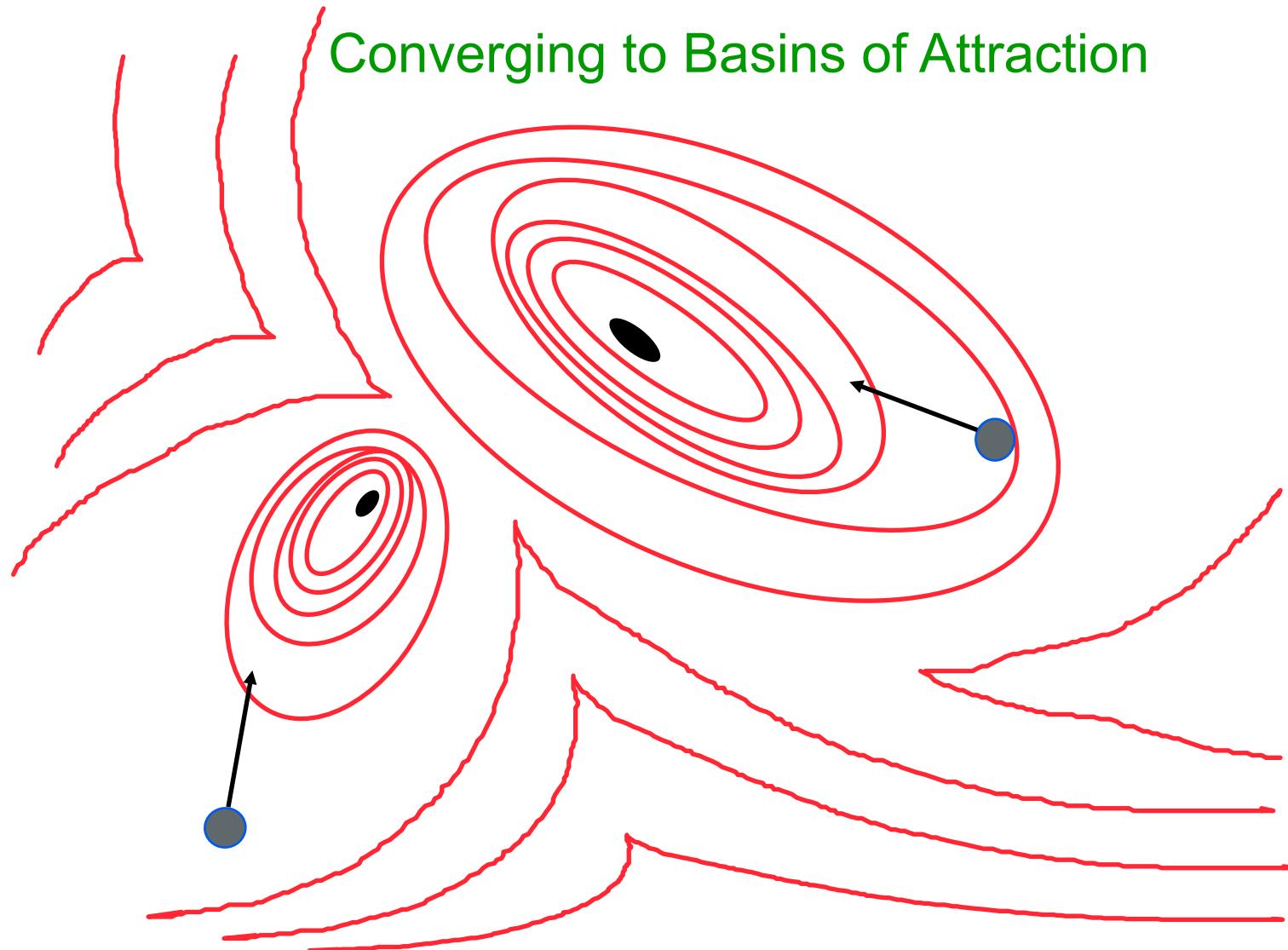


Yet Another Dynamical System





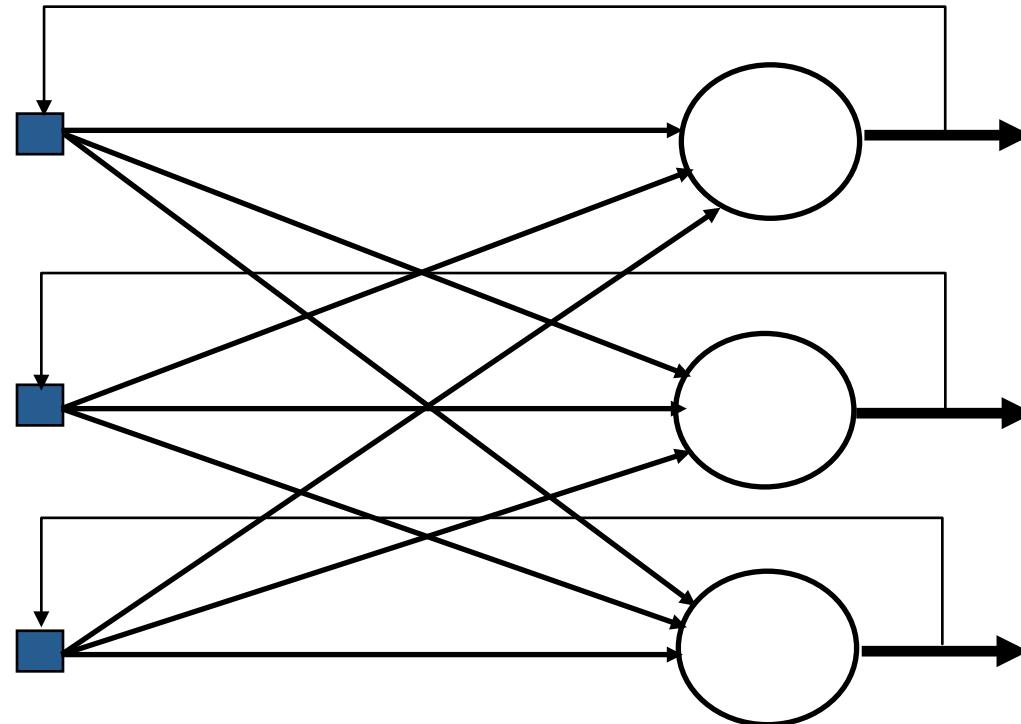
Converging to Basins of Attraction





Attractor Neural Nets

Hopfield Net





Some topics we'll cover:

- Simple mathematical relationships and theory
- Some 'systems' theory
- Some numerical methods and algorithms
- Some optimization methods

The Principle Theme

- **Using computers to perform experiments, investigate ideas and develop and test new theories!**



Introduction to Neural Networks

Johns Hopkins University
Engineering for Professionals Program
605-447.71/625-438.71

Dr. Mark Fleischer

Copyright 2013 by Mark Fleischer

Module 1.2: The Biological Neuron



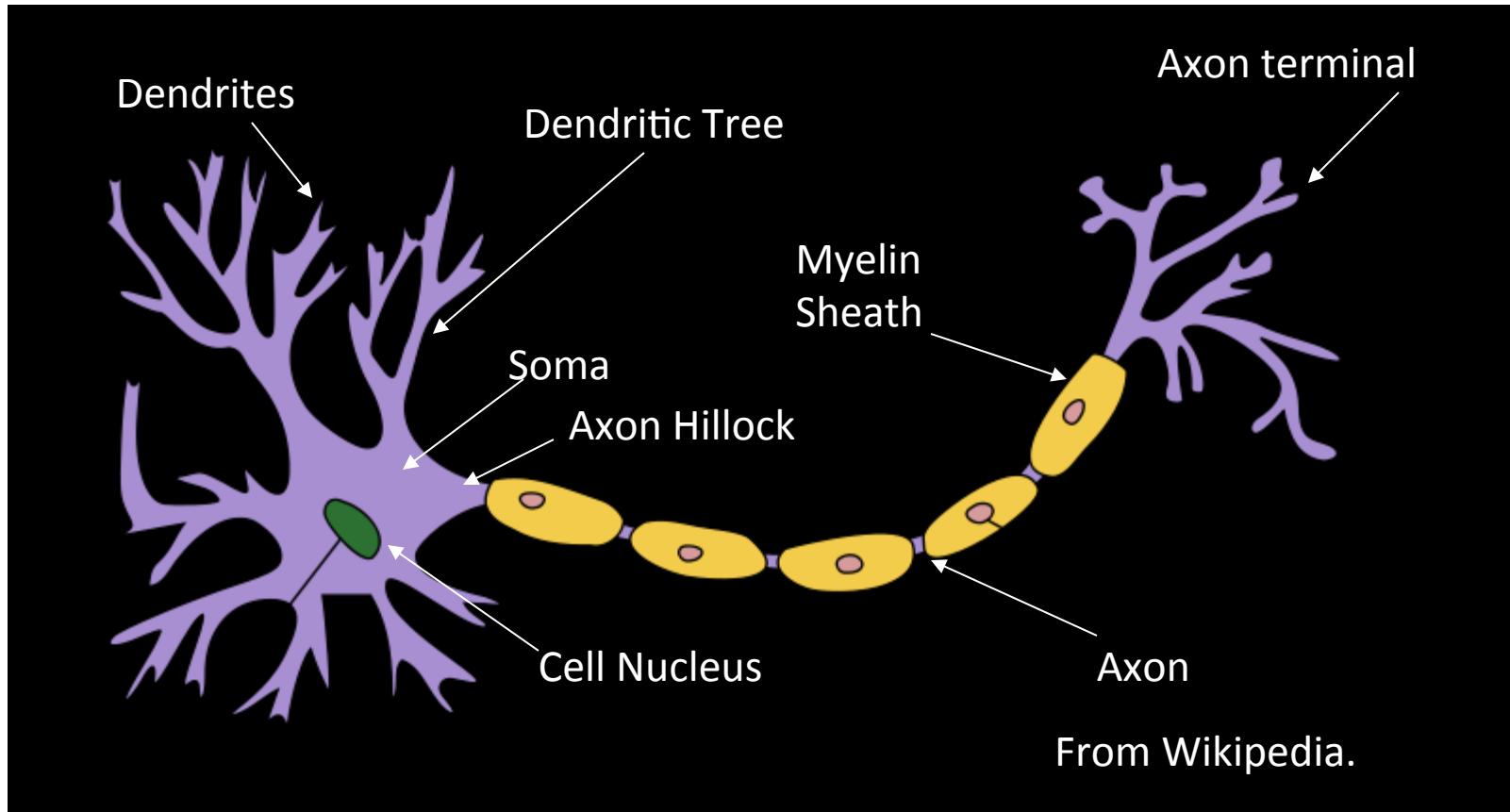
This Sub-Module Covers ...

- The basic elements/components of biological neurons and serves as the foundation of our modeling efforts. We will cover:
 - The basic structure of the biological neuron.
 - Some of the electro-chemical properties of the neuron.
 - Mechanisms for signaling between and among neurons.
 - Mechanisms associated with neuronal excitation and inhibition.
- This sub-module is then followed by a short quiz.



A Biological Neuron

Nature's Communication Mechanism





The Ion Pump

- A basic mechanism of a living neuron.
- ‘Pumps out’ sodium ions from inside of cell, pumps in potassium ions.
- 3:2 --- 3 out for every 2 in.
- Results in a net positive charge on the outside of the cell membrane.
- Some ions randomly cross membrane.
- Ion movement attempts to neutralize charge.
- Various types of ‘channels’ that are open or close and let ions move through the membrane more easily.



Neuronal Connections

- Axon terminals have ‘synaptic buttons’ at a ‘synapse’.
- The synapse ‘connects’ to dendrites of other cells and so one cell can connect to many other cells.
- The length of the axon is relatively long compared to the dimensions of the cell body --- long distance communication!

So how do the cells ‘communicate’?



Open the Flood Gates!

- Sometimes a channel is opened causing sodium ions to flood into the cell. E.g., ligand-gated channels near synapses. Causes local depolarization.
- This can cause nearby sodium channels to open. E.g., voltage-gated channels. Causes further depolarization.



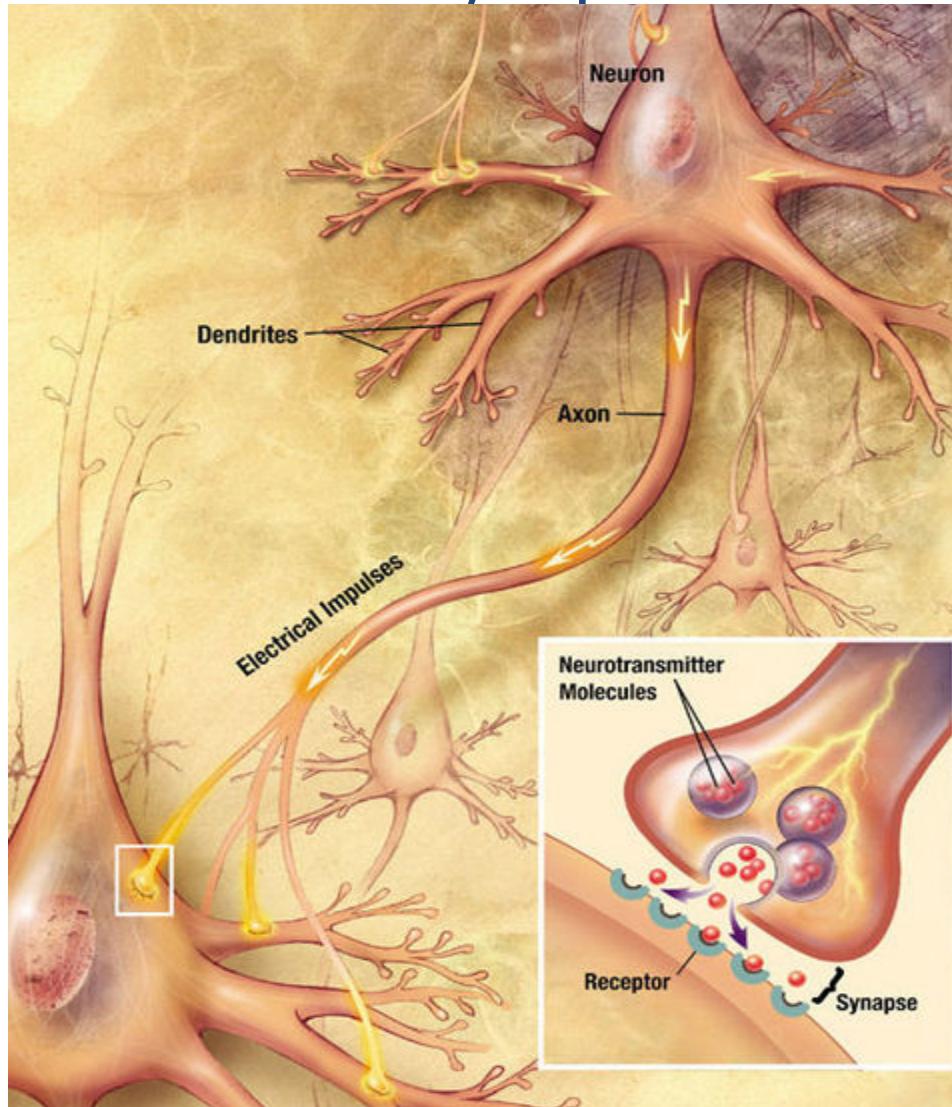
The Action Potential

A simplified view

- The ‘action potential’ is essentially a **pulse** of electric charge that travels down an axon.
- It cascades down the axon changing the state of channels which cause an electric waveform to expand causing further channels to change, etc. Just like dominoes.
- This pulse is triggered by electrical changes in the cell body.
- Inputs at synapses, affected by release of neuro-transmitters, trigger electrical changes (increase/decrease electric charge) in the neuron by changing the number of ions in the cell.
- These changes affect voltage dependent sodium channels through which ions can enter or leave a cell.



The Synapse



From Wikipedia



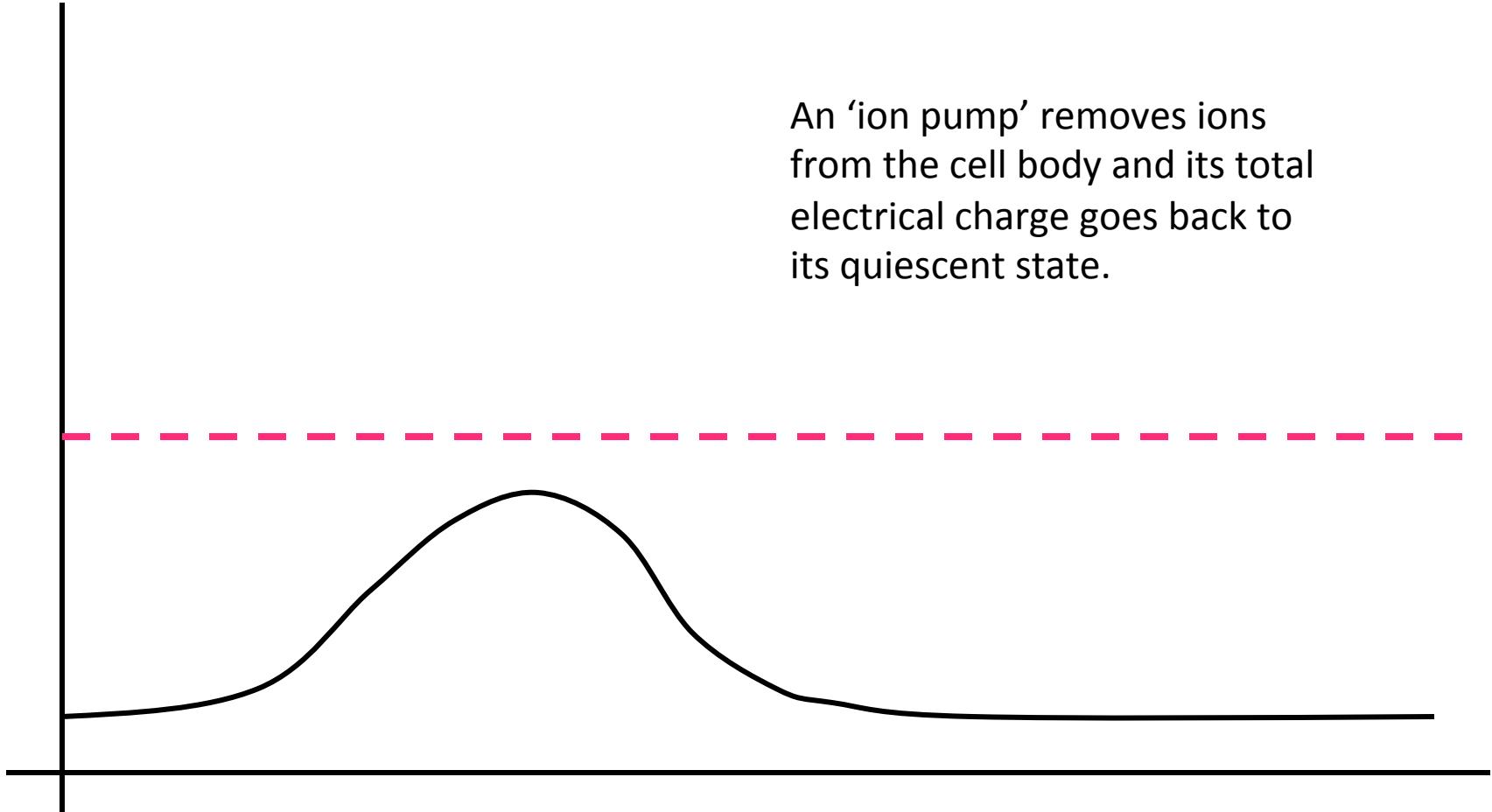
The Action Potential

- Charge builds up and/or decays in a neuron.
- If the charge continues to build up and reaches a threshold value, the cell begins to discharge... i.e., the action potential is triggered.



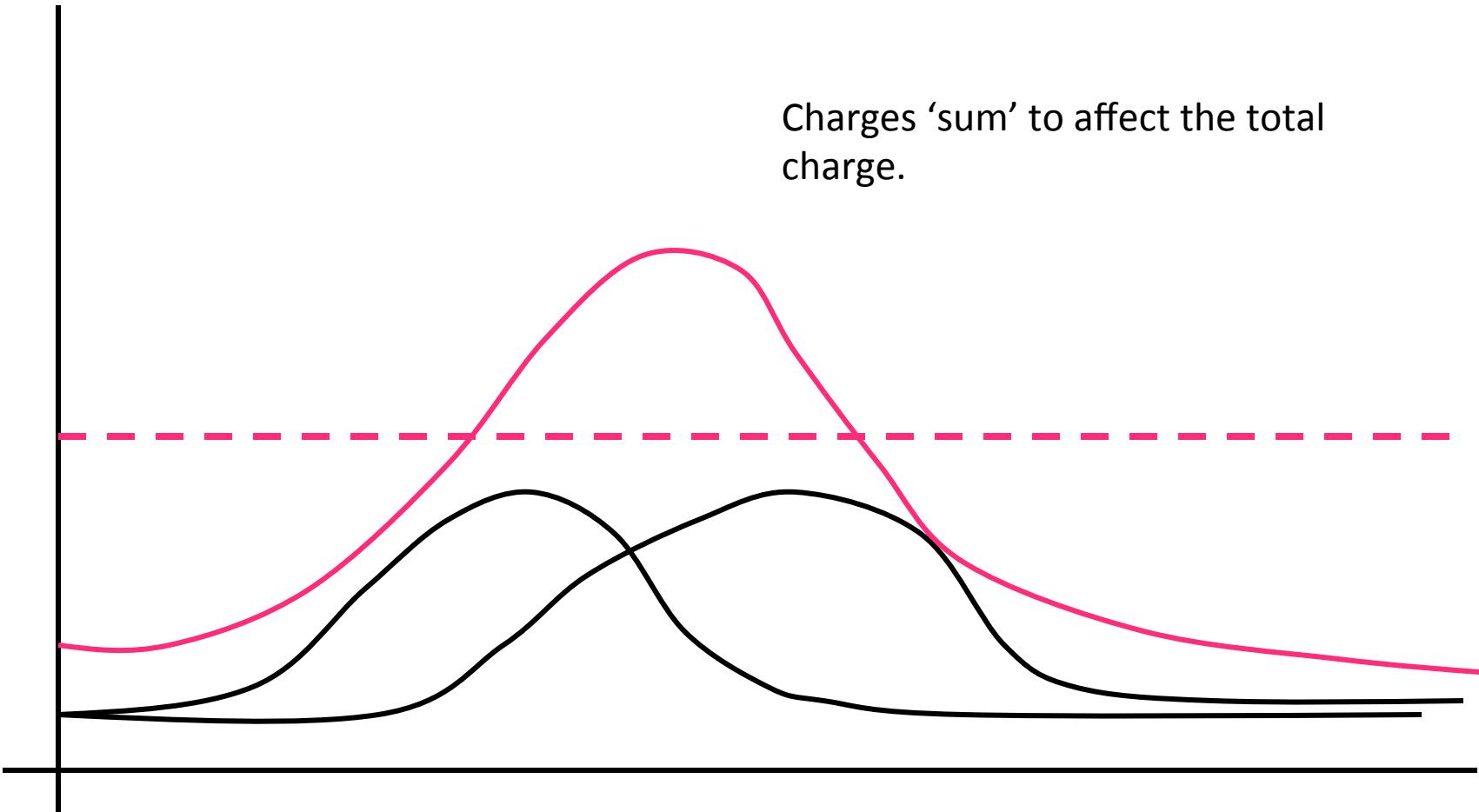
The Action Potential

An ‘ion pump’ removes ions from the cell body and its total electrical charge goes back to its quiescent state.





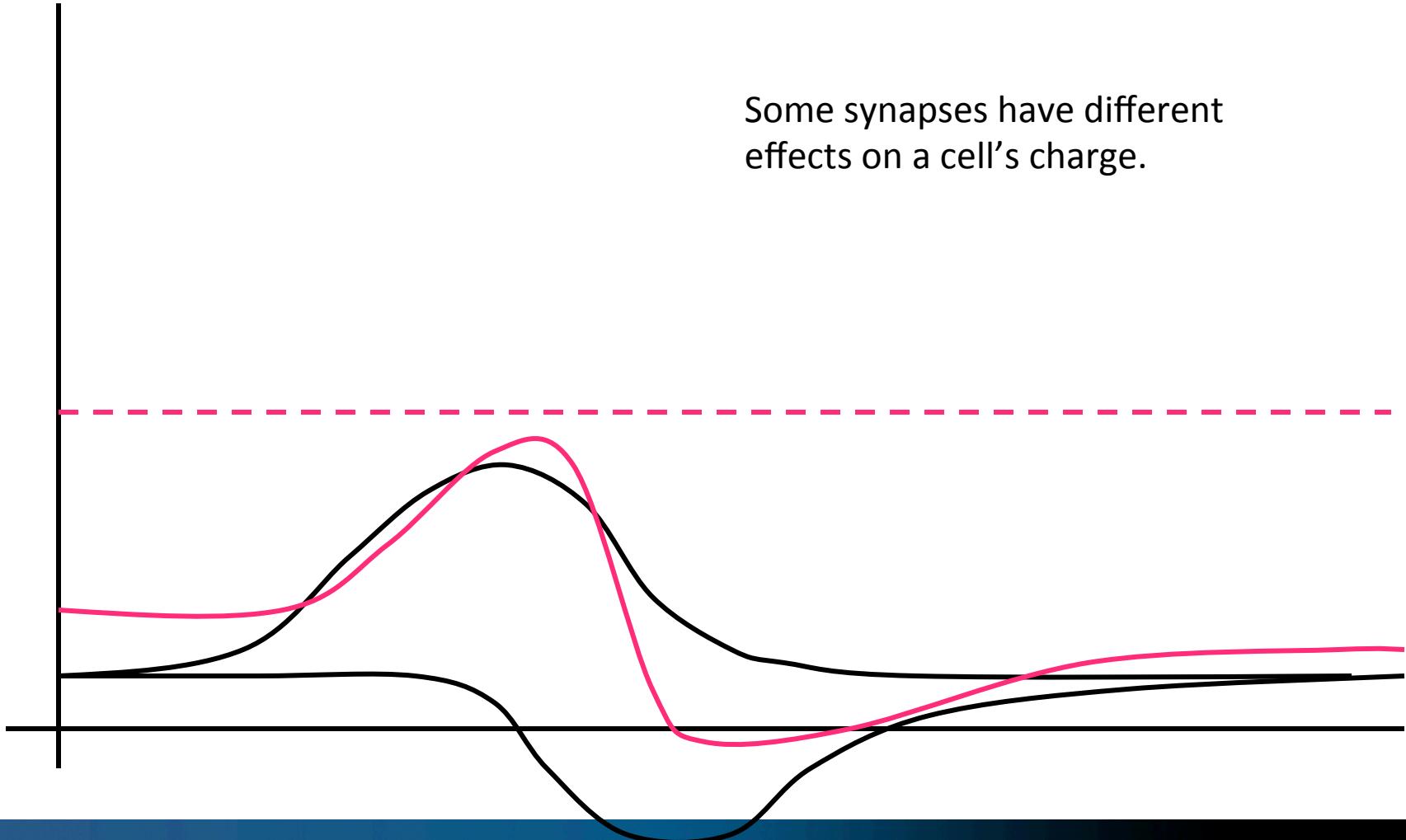
The Action Potential





The Action Potential

Some synapses have different effects on a cell's charge.





The Meaning of the Action Potential

- What does it ‘mean’ when it ‘fires’? What good is that?
- Think in terms of evolution.
- What does it mean when several action potential impinge on a given neuron?
- What does the fact that the effect of an action potential ‘decays’?



In the next sub-module...

- We will cover some of the issues surrounding the art and science of modeling.
- Before viewing the next sub-module, take the online quiz using the link following this presentation in the Module Content page.



Introduction to Neural Networks

Johns Hopkins University
Engineering for Professionals Program
605-447.71/625-438.71

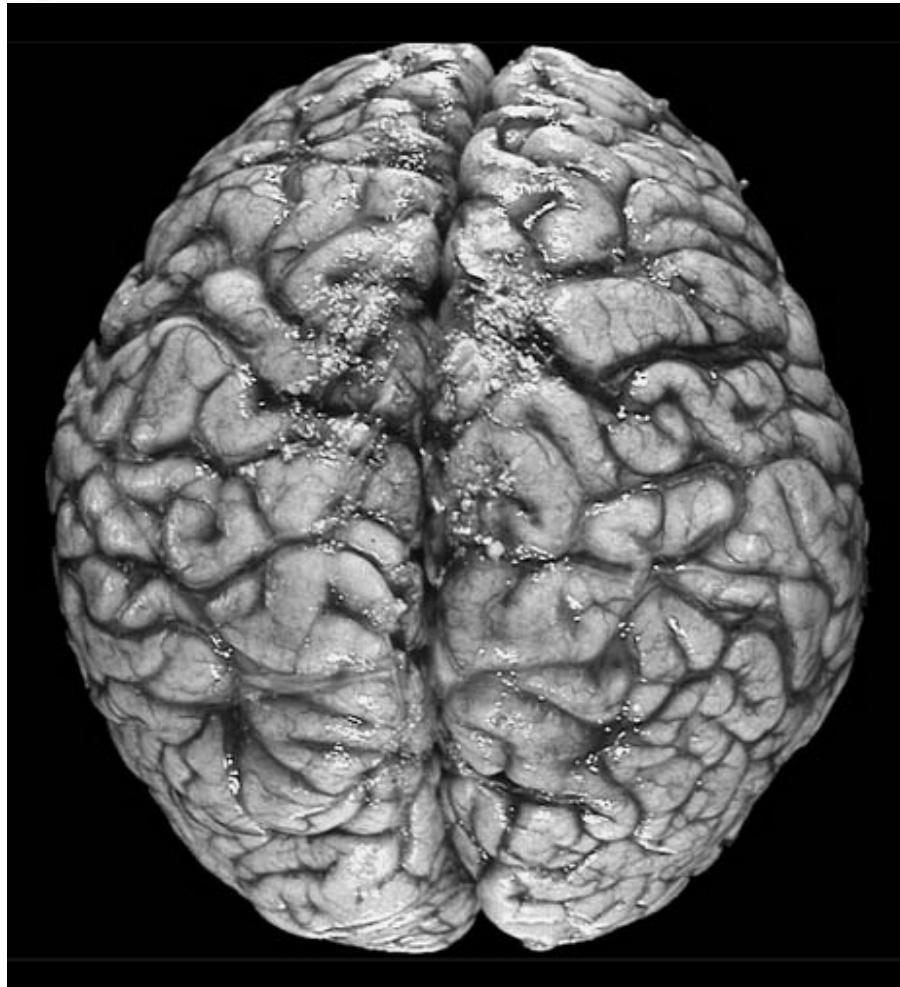
Dr. Mark Fleischer

Copyright 2013 by Mark Fleischer

Module 1.3: Modeling Considerations



My Brain, Your Brain





In this sub-module...

- We will cover some of the issues surrounding the art and science of modeling.
- These issues involve a tradeoff between detail and simplicity.
 - Want enough detail to capture important phenomena.
 - Want to keep things simple enough so that we can analyze our model and gain insight.

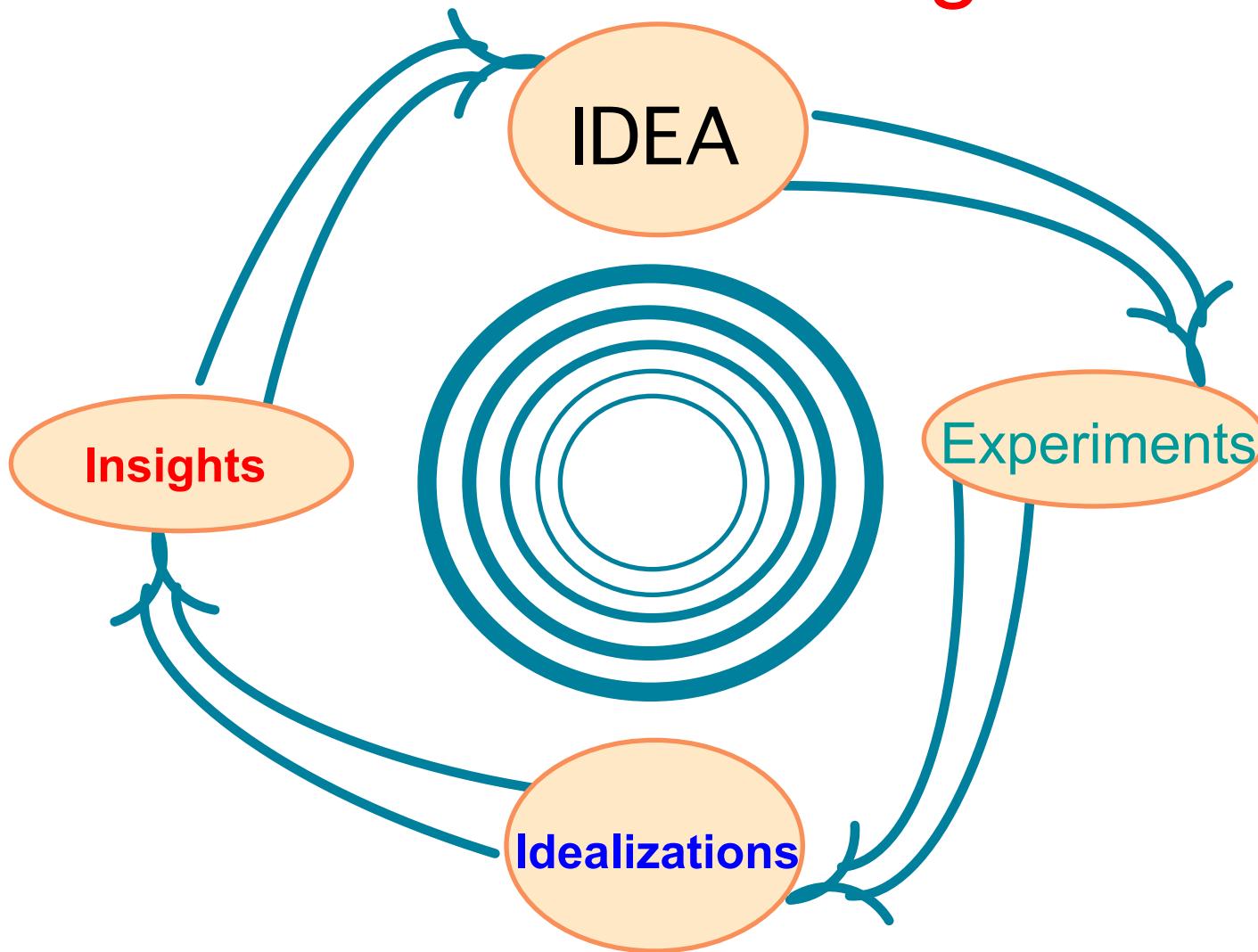


Modeling Reality

- Can't model all details. Too difficult and misses the mark.
- Don't want it too simple...also misses the mark.
- Have to strike a balance so that some interesting phenomena can be illuminated and possibly analyzed and studied.
- Capture the essential and interesting features.



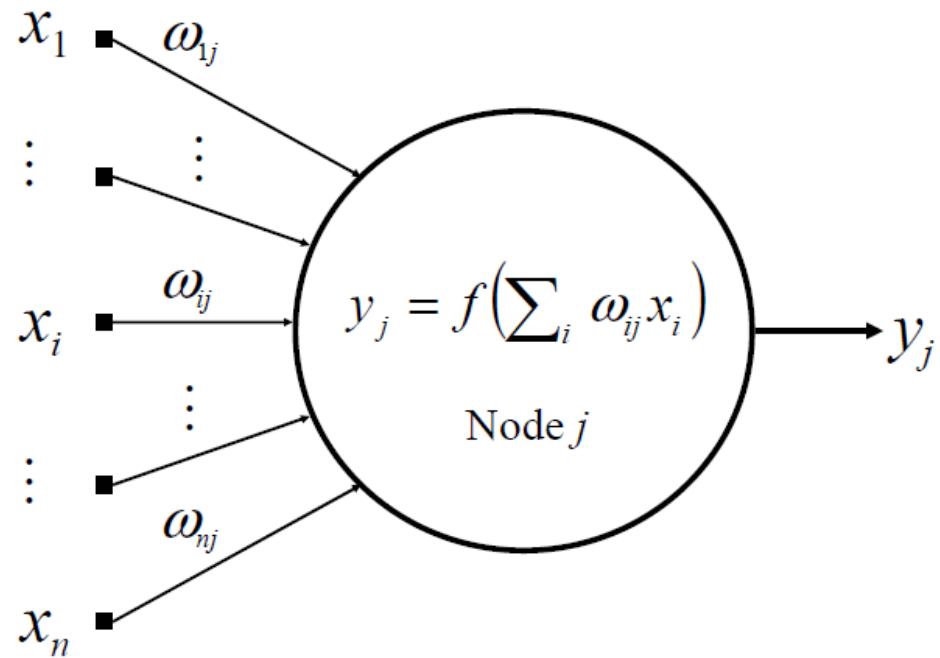
The Vortex of Progress





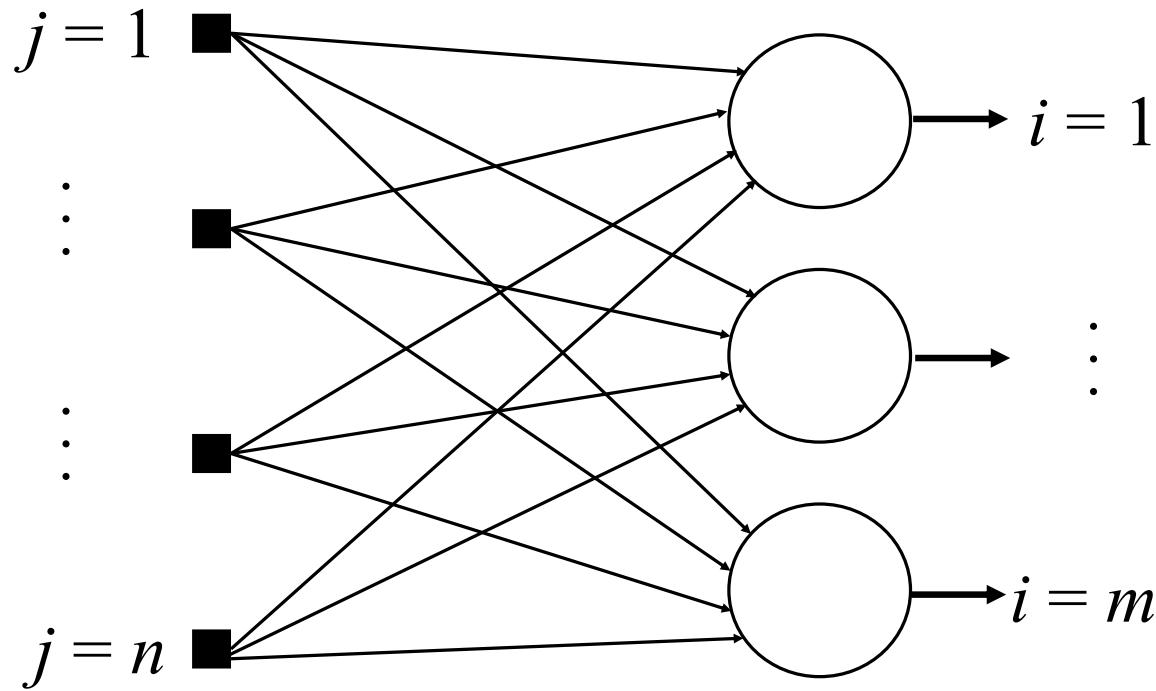


The Perceptron





The Multi-perceptron





The Activity Function

- The output of the Perceptron y is given by the equation:

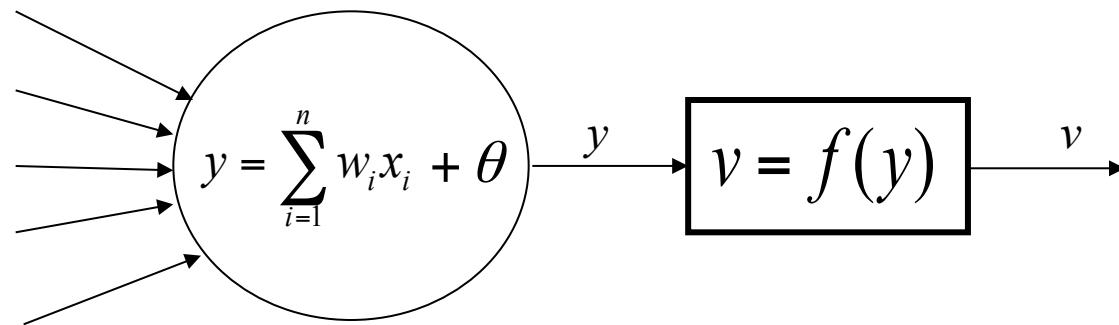
$$y = \sum_{i=1}^n w_i x_i + \theta$$

- This Activity Function is sometimes referred to as a *linear basis function*.



The Activation Function

- The Activation Function provides a one to one mapping between the Activity Function as input and some value for the output.
- It attempts to provide further flexibility in modeling biological neurons.



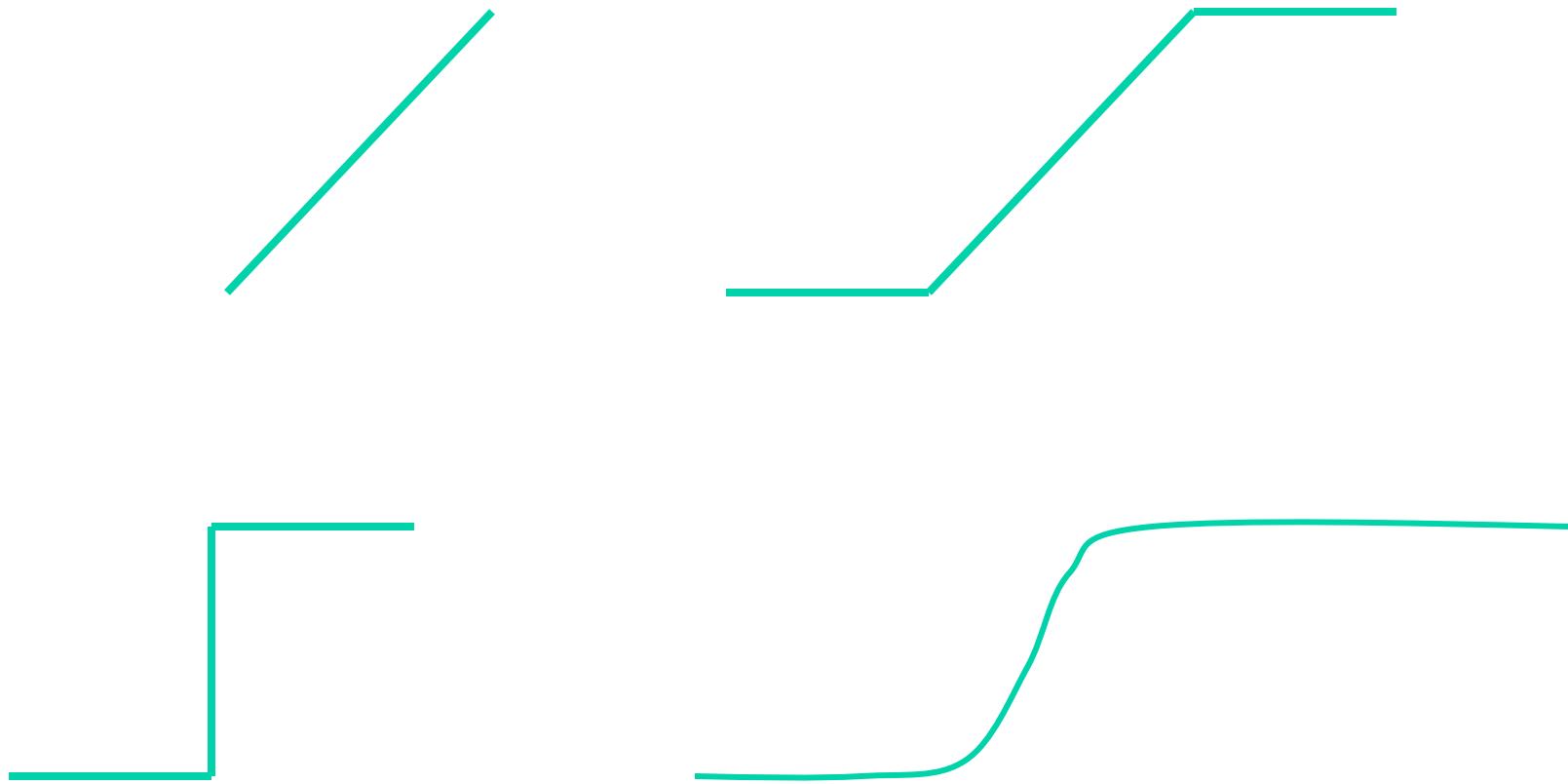


Features of the Perceptron

- Weighted inputs --- correspond to synaptic inputs.
 - Can involve both **positive and negative values**.
 - Provides an analogy to **excitation and inhibition**.
- Activation function --- corresponds to action potential.
 - We have great flexibility in choosing the activation function.
 - We choose one that satisfies our needs.



Activation Functions





An Important Activation Function

The Sigmoid Function

$$v = \frac{1}{1 + e^{-y}} = \frac{1}{1 + e^{-(\sum_{i=1}^n w_i x_i + \theta)}}$$



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING



Introduction to Neural Networks

Johns Hopkins University
Engineering for Professionals Program
605-447.71/625-438.71

Dr. Mark Fleischer

Copyright 2013 by Mark Fleischer

Module 2.1: Mathematical Review-Linear Algebra



This Sub-Module Covers ...

- Some mathematical review of Linear Algebra.
- Some of the essential elements of matrix/vector algebra.
- This sub-module is then followed by a short quiz.



Mathematical Review

- Preliminaries: Notational conventions

Summation

$$\sum_{j=1}^n a_{ij} = a_{i1} + a_{i2} + \cdots + a_{in}$$

Product

$$\prod_{j=1}^n a_{ij} = a_{i1} \cdot a_{i2} \cdot \cdots \cdot a_{in}$$

Vectors

If a row vector $\vec{a} = (a_1, a_2, \dots, a_n)$ then $\vec{a}^T = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}$



Vector Operations

Vector Addition:

Given two vectors **a** and **b** *of the same size*, $\mathbf{a} + \mathbf{b} = (a_1 + b_1, a_2 + b_2, \dots, a_n + b_n)$

For subtraction, the “+” is substituted with a “-”.

Vector Multiplication: (inner product, the dot product)

$$\vec{a} \cdot \vec{b} = (\vec{a}, \vec{b}) = \langle \vec{a}, \vec{b} \rangle = \sum_{i=1}^n a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

= some scalar quantity.



Matrix-Vector Operations

The inner product ---reprise:

$$\mathbf{ab}^T = (a_1, a_2, \dots, a_n) \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

This operation results in a single scalar value. If $\mathbf{a}^T\mathbf{b} = 0$ when both vectors **a** and **b** are non-zero vectors, then the vectors **a** and **b** are said to be *orthogonal*.

A *non-zero vector* is a vector in which **at least one element** is not zero.

$$(0, 0, 0, 1.3, 0, 0)$$



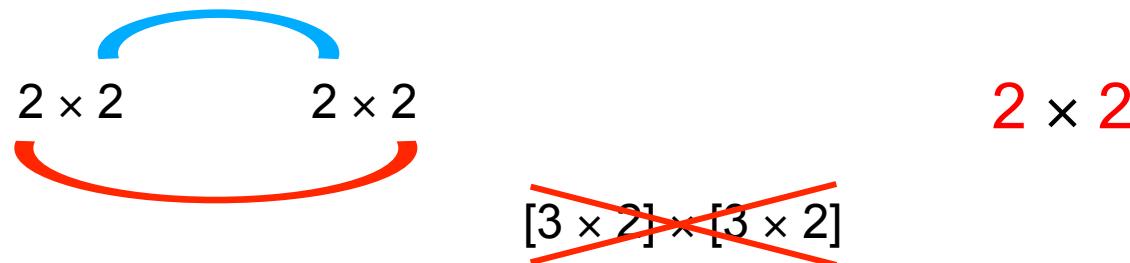
The Outer Product

$$\mathbf{a}^T \mathbf{b} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} (b_1, b_2, \dots, b_n) = \begin{pmatrix} a_1 b_1 & a_1 b_2 & \cdots & a_1 b_n \\ a_2 b_1 & a_2 b_2 & \cdots & a_2 b_n \\ \vdots & \vdots & \ddots & \vdots \\ a_n b_1 & a_n b_2 & \cdots & a_n b_n \end{pmatrix}$$



Matrix Multiplication

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$



$$[4 \times 10] \times [10 \times 2] = [4 \times 2]$$

$$\begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \end{bmatrix} \begin{bmatrix} \mathbf{b}_1 & \mathbf{b}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1 \cdot \mathbf{b}_1 & \mathbf{a}_1 \cdot \mathbf{b}_2 \\ \mathbf{a}_2 \cdot \mathbf{b}_1 & \mathbf{a}_2 \cdot \mathbf{b}_2 \end{bmatrix}$$



Linear Independence

A set of **non-zero vectors** \mathbf{v}_i , $i = 1, \dots, n$ is said to be

linearly independent where $\sum_{i=1}^n a_i \mathbf{v}_i = \mathbf{0}$ if and only if
 $a_i = 0$ for all i .

$$a_1 \begin{bmatrix} v_{11} \\ v_{12} \\ v_{13} \end{bmatrix} + a_2 \begin{bmatrix} v_{21} \\ v_{22} \\ v_{23} \end{bmatrix} + a_3 \begin{bmatrix} v_{31} \\ v_{32} \\ v_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

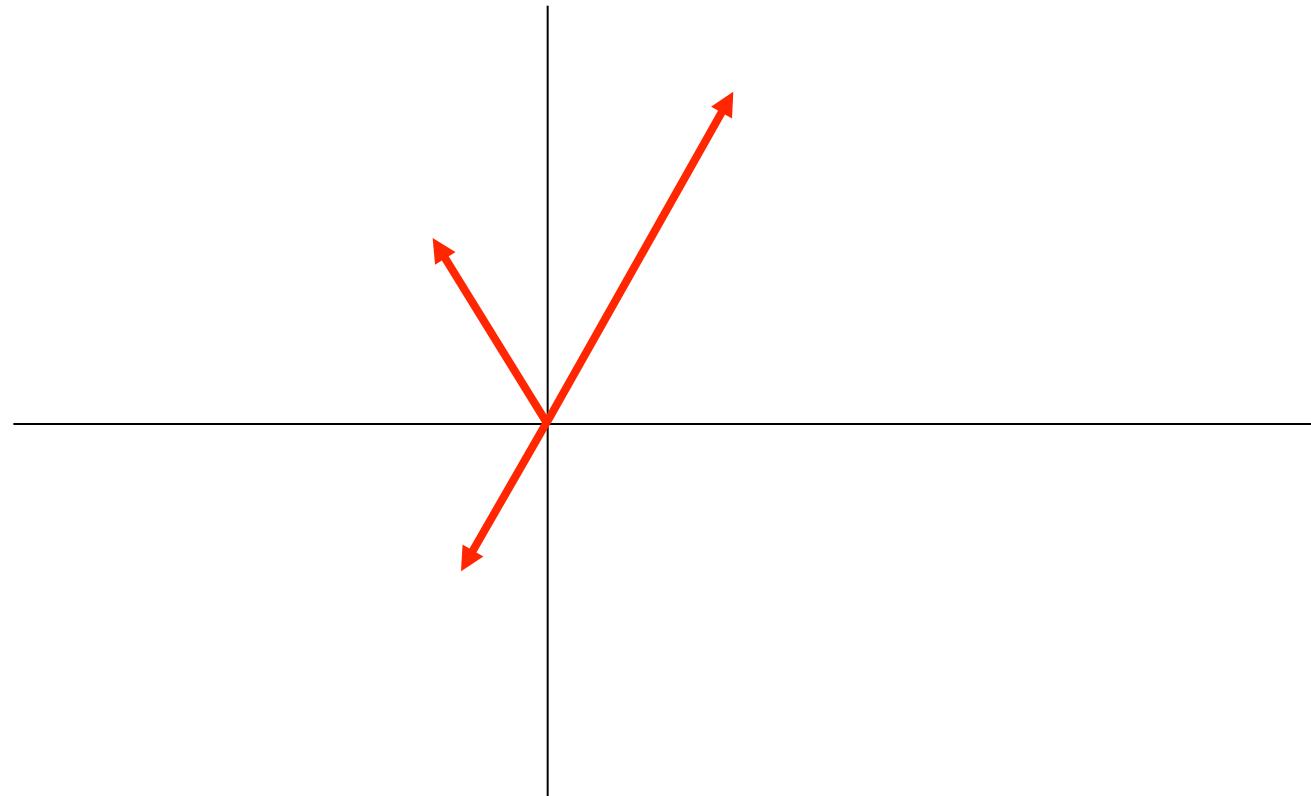
$$a_1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + a_2 \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING



Linear Independence





Introduction to Neural Networks

Johns Hopkins University
Engineering for Professionals Program
605-447.71/625-438.71

Dr. Mark Fleischer

Copyright 2013 by Mark Fleischer

Module 2.2: Mathematical Review-Differential Calculus



This Sub-Module Covers ...

Review of differential calculus:

- derivatives, partial derivatives, gradients
- directional derivatives

The next sub-module covers:

- Calculus-based optimization methods and related material:
 - First order necessary conditions.
 - Second order sufficiency conditions.
 - Definition of convexity.



Optimization and Learning

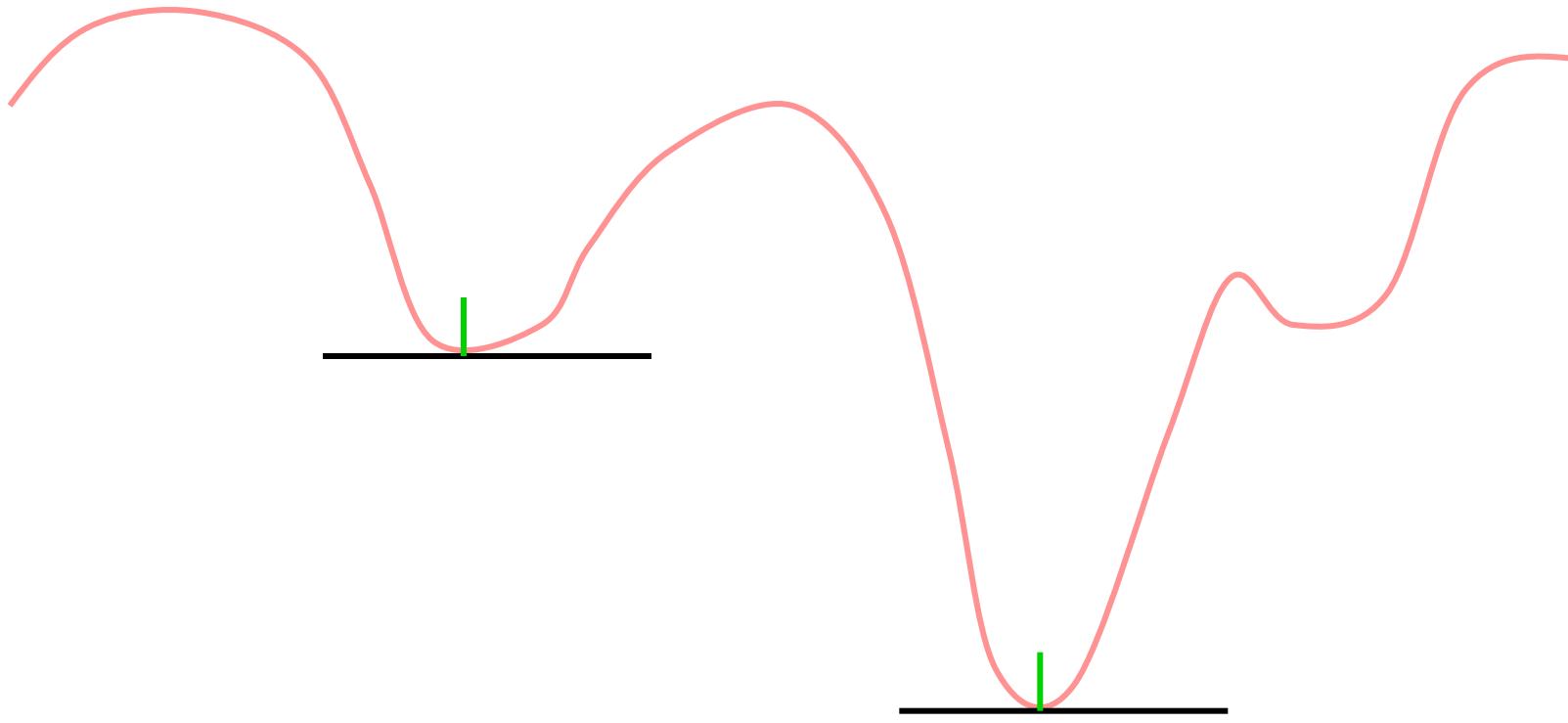
Many types of optimization problems.

- For continuous functions, we use calculus-based methods.
- Problems with linear objective functions, linear constraint equations can be solved using **Linear Programming** methods
- Problems with non-linear objective functions, non-linear constraint equations can be solved using **Non-Linear Programming** methods.
- For now, we will not worry about constraints.

Training a neural network involves solving an optimization problem!

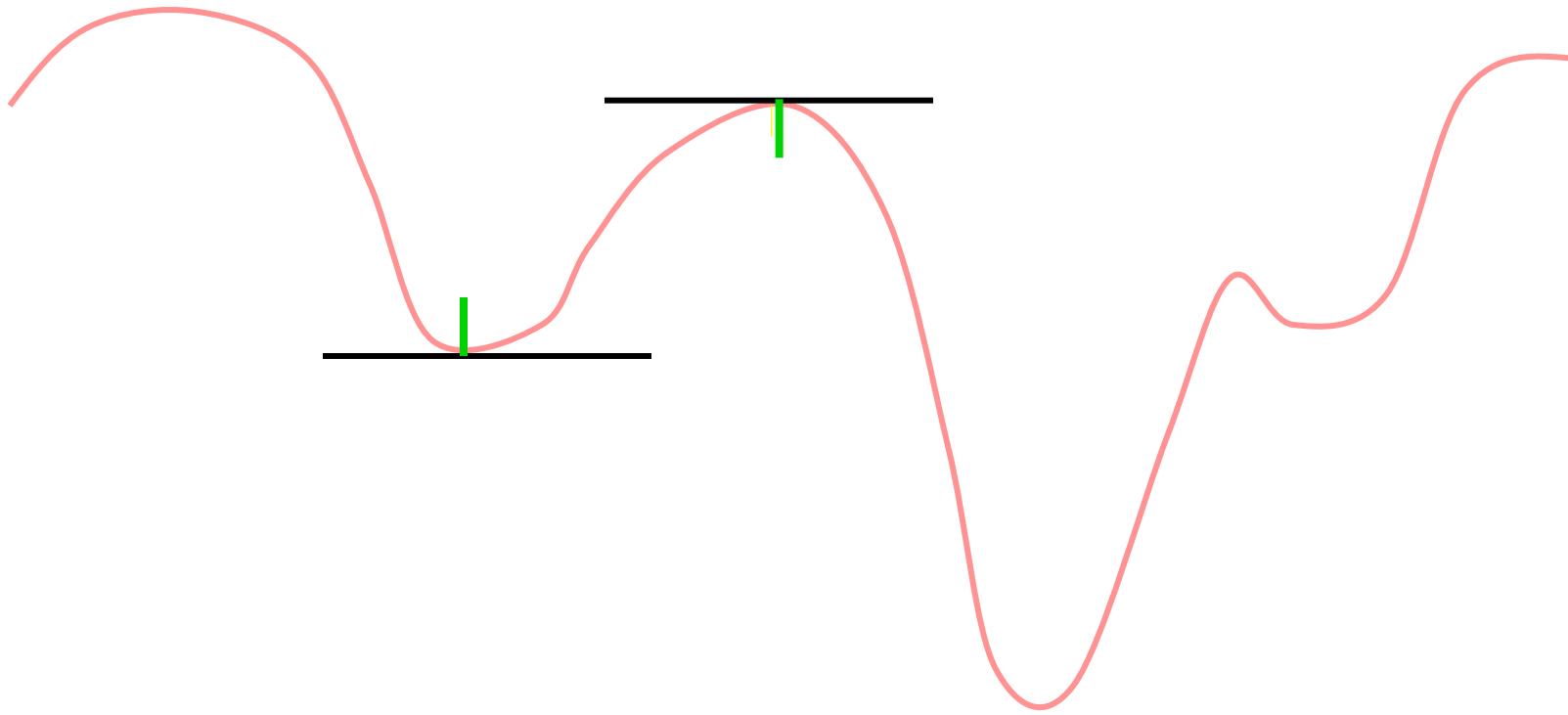


Calculus Based Optimization





Calculus Based Optimization





Derivatives

$$\frac{dy}{dx} = \frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

$$\frac{\partial f(x)}{\partial x_i} = \frac{f(x_1, \dots, x_i + h, \dots, x_n) - f(x_1, \dots, x_i, \dots, x_n)}{h}$$

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f(x_1)}{\partial x_1}, \frac{\partial f(x_2)}{\partial x_2}, \dots, \frac{\partial f(x_n)}{\partial x_n} \right)$$



Derivatives

$$f(x) = ax^n \Rightarrow \frac{df(x)}{dx} = f'(x) = nax^{n-1}$$

$$f(x_1, x_2) = ax_1 x_2^n \Rightarrow \frac{\partial f(x_1, x_2)}{\partial x_2} = nax_1 x_2^{n-1}$$



Directional Derivatives

- In partial derivatives, we consider the slope of a surface in the direction along one variable axis.
- In directional derivatives, we consider the slope of a surface in a specified direction.

Important for training neural networks!



Directional Derivatives

$$\frac{df(x, y, z)}{d\mathbf{d}} = \lim_{t \rightarrow 0^+} \frac{f(x + td_1, y + td_2, z + td_3) - f(x, y, z)}{t}$$

Each independent variable x, y and z is perturbed by a component of the direction vector \mathbf{d} and multiplied by a factor t .



Directional Derivatives

$$\begin{aligned}\frac{d f(x, y, z)}{d\mathbf{d}} &= \frac{\partial f}{\partial x} \frac{dx}{dt} + \frac{\partial f}{\partial y} \frac{dy}{dt} + \frac{\partial f}{\partial z} \frac{dz}{dt} \\ &= \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right) \cdot \left(\frac{dx}{dt}, \frac{dy}{dt}, \frac{dz}{dt} \right) \\ &= \nabla f(x, y, z) \cdot \mathbf{d}\end{aligned}$$



Directional Derivatives





JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING



Introduction to Neural Networks

Johns Hopkins University
Engineering for Professionals Program
605-447.71/625-438.71

Dr. Mark Fleischer

Copyright 2013 by Mark Fleischer

Module 2.3: Mathematical Review-Calculus Based Optimization



This Sub-Module Covers ...

- Calculus-based optimization methods and related material:
 - First order necessary conditions.
 - Second order sufficiency conditions.
 - Definition of convexity.
- Sets the stage for further mathematical review by exploring Metric Spaces in the next sub-module.



First-Order Necessary Conditions

TFAE

$$\frac{df(x^*)}{dx} = 0$$

$$\nabla f(\mathbf{x}^*) = \mathbf{0} = (0, 0, \dots, 0)$$

$$\forall \mathbf{d}, \quad \nabla f(\mathbf{x}^*) \cdot \mathbf{d} = 0$$

There exists a $t' > 0$, such that for all t , where $0 < t < t'$, and for all non - zero vectors \mathbf{d}

$$(\exists t' > 0, \exists \forall 0 < t < t' \wedge \mathbf{d} > \mathbf{0}),$$

$$f(\mathbf{x}^*) < f(\mathbf{x}^* + t\mathbf{d})$$

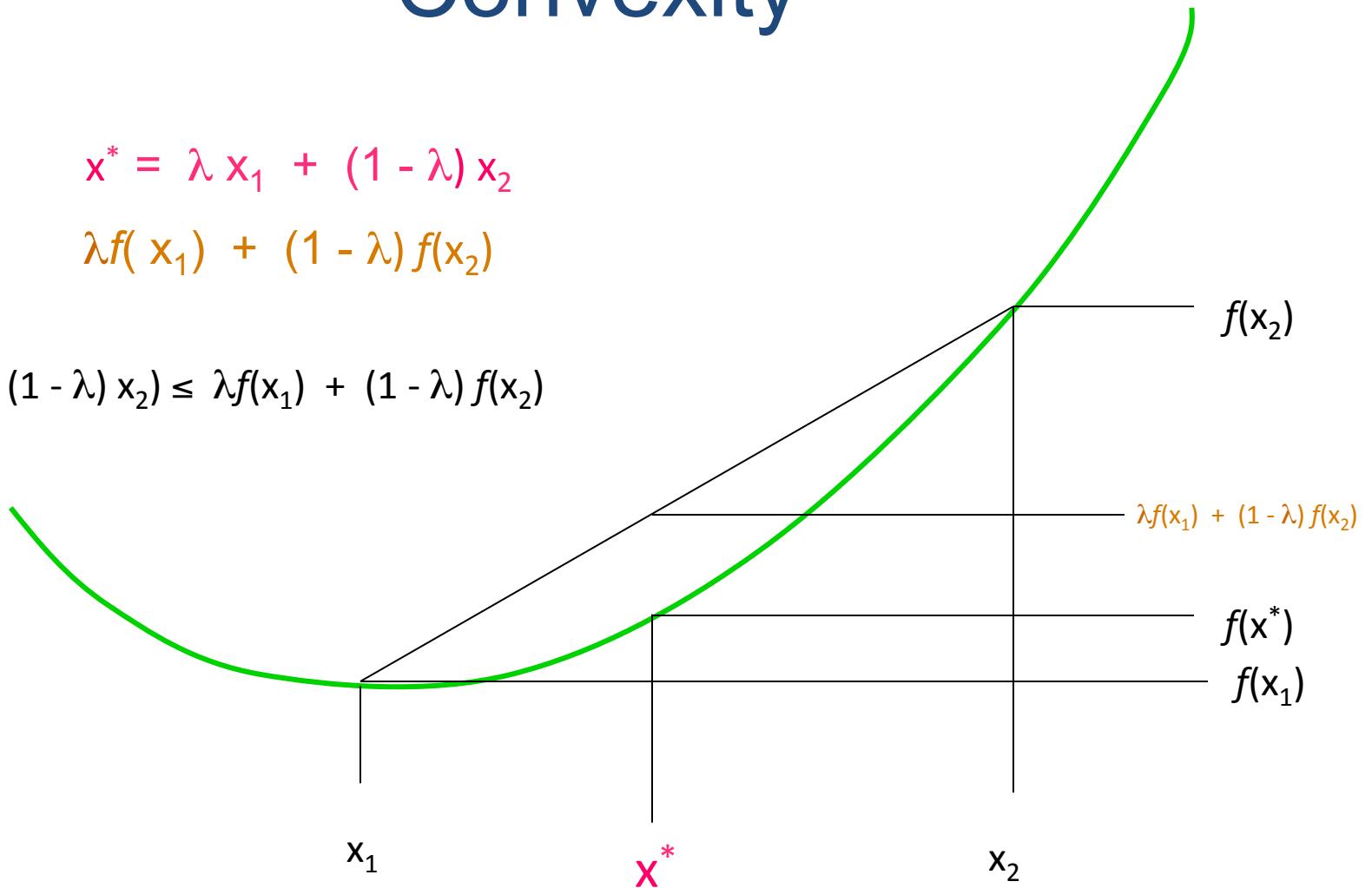


Convexity

$$x^* = \lambda x_1 + (1 - \lambda) x_2$$

$$\lambda f(x_1) + (1 - \lambda) f(x_2)$$

$$f(\lambda x_1 + (1 - \lambda) x_2) \leq \lambda f(x_1) + (1 - \lambda) f(x_2)$$

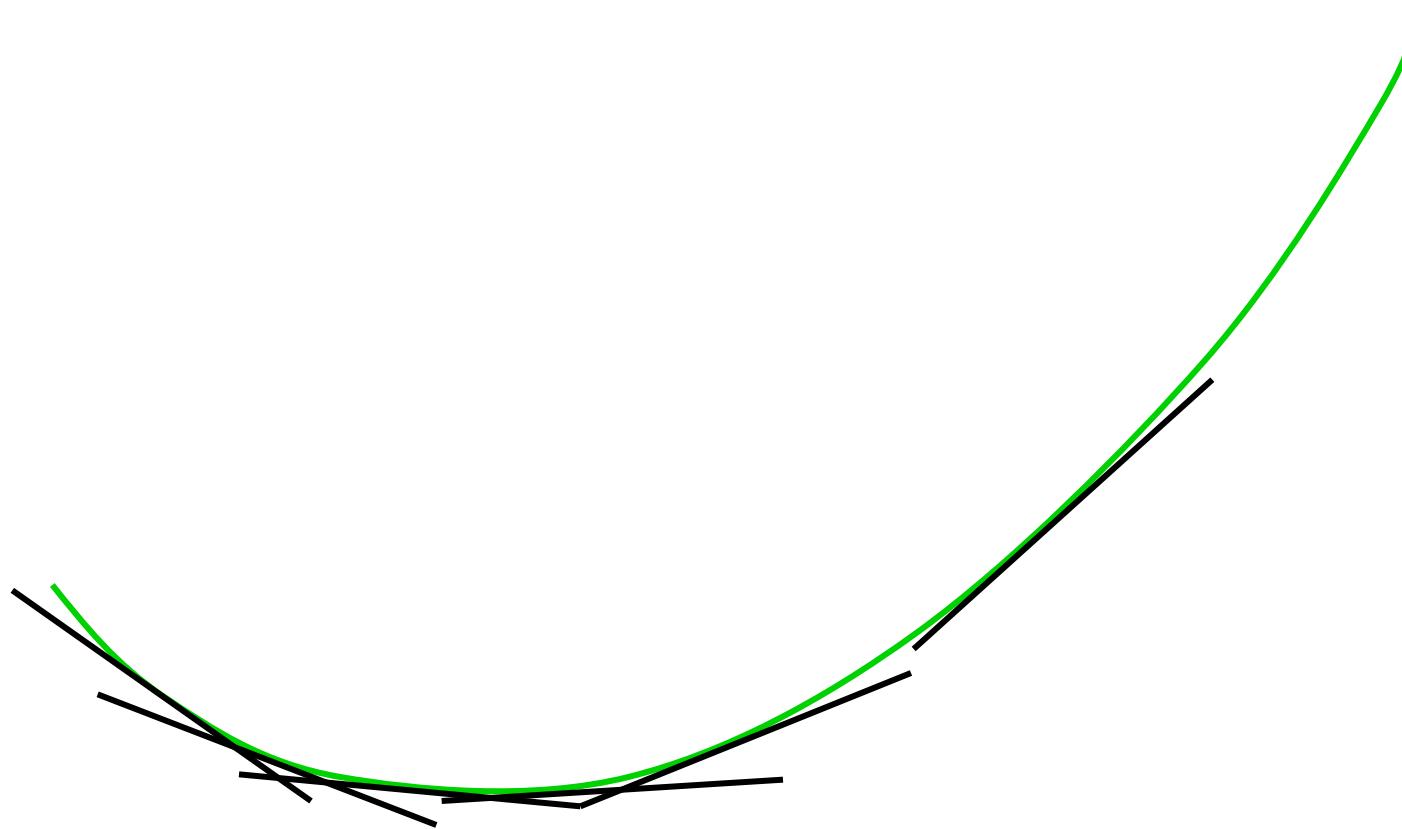




JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING



Second Derivatives





Second-Order Sufficiency Conditions

TFAE (for determining minima)

1. All **second** derivatives are positive
2. All points in a tangent plane (or hyperplane) have function values less than or equal to the objective function value.
3. The function is convex.

Item 1:

$$\frac{d^2y}{dx^2} = f''(x^*) > 0$$

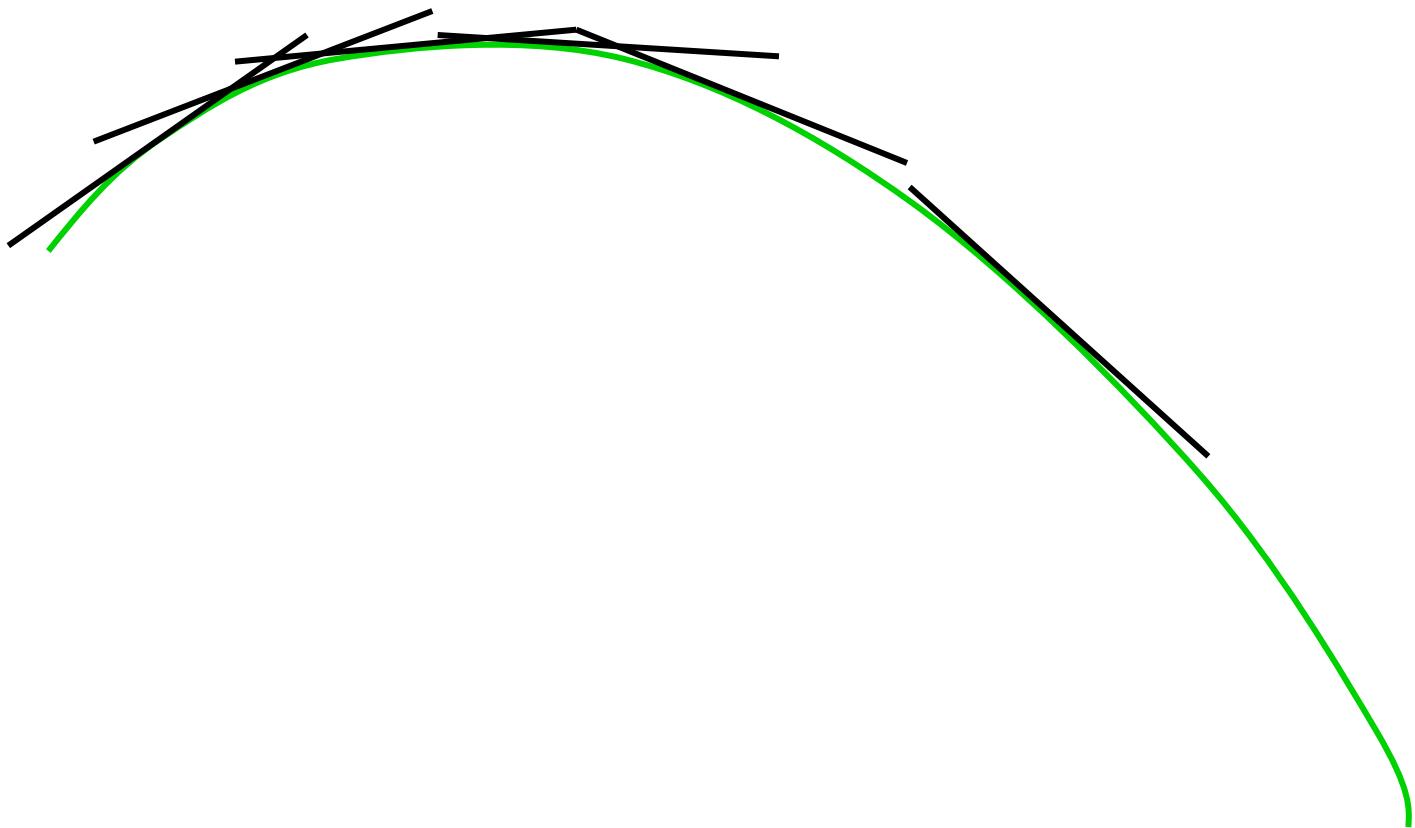
The Hessian Matrix $\mathbf{H}(\mathbf{x})$ is positive definite,
i.e., for all R^n , $\mathbf{x}^T \mathbf{H} \mathbf{x} \geq 0$:



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING



Second Derivatives





Second-Order Sufficiency Conditions

TFAE (for determining minima)

1. All second derivatives are positive
2. All points in a tangent plane (or hyperplane) have function values less than or equal to the objective function value.
3. The objective function is convex.

Item 2:

$$2. \forall \mathbf{x}, \mathbf{x}^* \in R^n,$$

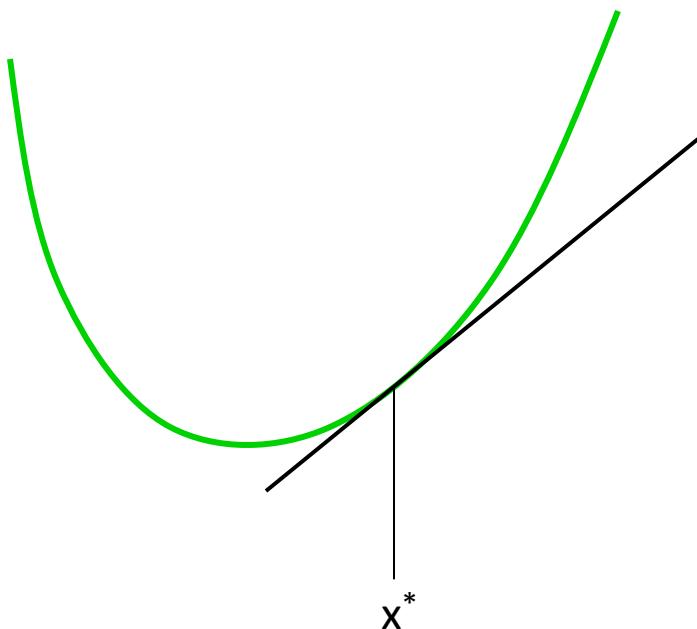
$$f(\mathbf{x}) \geq f(\mathbf{x}^*) + \nabla f(\mathbf{x}^*)(\mathbf{x} - \mathbf{x}^*).$$



Item 2: Tangent Plane

2. $\forall \mathbf{x}, \mathbf{x}^* \in R^n,$

$$f(\mathbf{x}) \geq f(\mathbf{x}^*) + \nabla f(\mathbf{x}^*)(\mathbf{x} - \mathbf{x}^*).$$



$$f(\mathbf{x}^*) + \nabla f(\mathbf{x}^*)(\mathbf{x} - \mathbf{x}^*)$$

$$\begin{aligned}&= f(\mathbf{x}^*) + m(\mathbf{x} - \mathbf{x}^*) \\&= f(\mathbf{x}^*) + mx - mx^* \\&= mx + [f(\mathbf{x}^*) - mx^*] \\&= mx + b\end{aligned}$$



Second-Order Sufficiency Conditions

TFAE (for determining minima)

1. All second derivatives are positive
2. All points in a tangent plane (or hyperplane) have function values less than or equal to the objective function value.
3. The objective function is convex.

Item 3:

3. $\forall \mathbf{x}, \mathbf{x}^* \in R^n$ and $0 \leq \lambda \leq 1$,

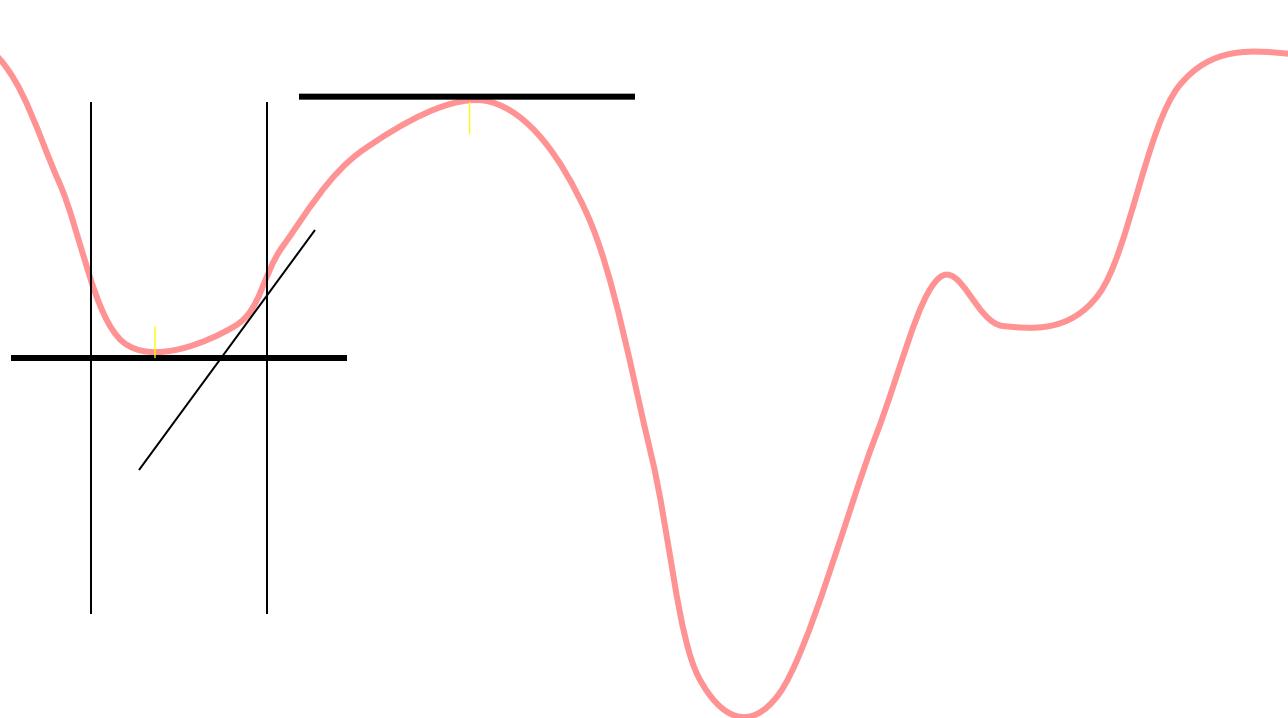
$$\lambda f(\mathbf{x}) + (1 - \lambda)f(\mathbf{x}^*) \geq f(\lambda\mathbf{x} + (1 - \lambda)\mathbf{x}^*).$$



JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING



Calculus Based Optimization





Example

Suppose we want to minimize the function:

$$f(x_1, x_2) = x_1 x_2 - 6x_1 - 3x_2 + 18$$

Taking all partial derivatives, we see that:

$$\frac{\partial f}{\partial x_1} = x_2 - 6 = 0$$

$$\frac{\partial f}{\partial x_2} = x_1 - 3 = 0$$

$$x_1 = 3; x_2 = 6$$



Example

To show that it is a minimum, it is *sufficient* to show that one of the three second-order sufficiency conditions holds. Using the condition in item 2, we see that the equation for the tangent plane at say point (4,3) is:

Remembering that

$$\frac{\partial f}{\partial x_1} = x_2 - 6 = 0$$

$$\frac{\partial f}{\partial x_2} = x_1 - 3 = 0$$

$$\begin{aligned}f_P(x_1, x_2) &= f(4, 3) + \nabla f(4, 3) \begin{pmatrix} x_1 - 4 \\ x_2 - 3 \end{pmatrix} \\&= -3 + (-3, 1) \begin{pmatrix} x_1 - 4 \\ x_2 - 3 \end{pmatrix} \\&= -3 - 3(x_1 - 4) + 1(x_2 - 3) \\&= -3x_1 + x_2 + 6\end{aligned}$$



Example

Now consider any point (x_1, x_2) and compare the value of f_P with the objective function value f . For example, at point $(0,0)$ the value of $f_p = 6$. For the objective function, $f(0,0) = 18$.

Since the relationship between the objective function and tangent plane holds as item 2 above, it *suggests* that the second-order conditions hold. To establish this however requires that we prove this relation holds for *all* points (x_1, x_2) .

Can you prove that they do?

Remembering that

$$f(x_1, x_2) = x_1 x_2 - 6x_1 - 3x_2 + 18$$

$$\begin{aligned} f_P(x_1, x_2) &= f(4, 3) + \nabla f(4, 3) \begin{pmatrix} x_1 - 4 \\ x_2 - 3 \end{pmatrix} \\ &= -3 + (-3, 1) \begin{pmatrix} x_1 - 4 \\ x_2 - 3 \end{pmatrix} \\ &= -3 - 3(x_1 - 4) + 1(x_2 - 3) \\ &= -3x_1 + x_2 + 6 \end{aligned}$$



Taylor's Theorem

Single variable case

$$\begin{aligned} f(x) &= a_0 + a_1(x - x_0) + a_2(x - x_0)^2 \dots \\ &= \boxed{f(x_0) + (x - x_0)f'(x_0)} + \frac{1}{2!}(x - x_0)^2 f''(x_0) \dots \end{aligned}$$

Multi - variable case

$$\begin{aligned} f(\mathbf{x}) &= a_0 + a_1(\mathbf{x} - \mathbf{x}_0) + a_2(\mathbf{x} - \mathbf{x}_0)^2 \dots \\ &= \boxed{f(\mathbf{x}_0) + (\mathbf{x} - \mathbf{x}_0)\nabla f(\mathbf{x}_0)} + \frac{1}{2!}(\mathbf{x} - \mathbf{x}_0)^T \mathbf{H}(\mathbf{x} - \mathbf{x}_0) \dots \end{aligned}$$



Mathematical Review

So far we've reviewed:

- Basic Vector/Matrix operations: inner, outer products
- Linear Independence
- Differential calculus
- Partial Differentiation,
- Directional Derivatives $\nabla f(x, y, z) \cdot \mathbf{d} = \|\nabla f(x, y, z)\| \times \|\mathbf{d}\| \times \cos \theta$
- Gradient vector $\nabla f(\mathbf{x}) = (\partial f(\mathbf{x})/\partial x_1, \partial f(\mathbf{x})/\partial x_2, \dots, \partial f(\mathbf{x})/\partial x_n)$
- First order necessary conditions, Second order sufficiency conditions

In the next sub-modules we will review:

- Metric Spaces:
 - Distance and Magnitude of vectors, matrices
 - Definitional requirements of “norms”
 - **Positivity, homogeneity, Triangle Inequality**



Introduction to Neural Networks

Johns Hopkins University
Engineering for Professionals Program
605-447.71/625-438.71

Dr. Mark Fleischer

Copyright 2013 by Mark Fleischer

Module 2.4: Mathematical Review-Metric Spaces



This Sub-Module Covers ...

- Some mathematical review of Metric Spaces and will set the stage for using:
 - directional derivatives in conjunction with calculus based optimization to define the
 - Method of Steepest Descent (MOSD)---used to train Perceptrons.
- Also provides additional insights into dynamical systems.



What is ‘Distance’ and ‘Magnitude’?

- Need capability to rank and/compare objects using a simple criterion.
- Want a flexible yet abstract notion of distance or length or magnitude.
- The following principles provide a very **abstract, general** way of defining the essential properties of a ‘length’.



Length

Desirable properties of a “length”(magnitude or norm):

1. **Positivity:** $\|\mathbf{y}\| \geq 0$ for all \mathbf{y} and $\|\mathbf{y}\| = 0$ if and only if $\mathbf{y} = \mathbf{0}$.
2. **Homogeneity:** $\|c\mathbf{y}\| = |c| \|\mathbf{y}\|$ for all scalars c and vectors \mathbf{y} .
3. **The Triangle Inequality:** $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$ for all vectors \mathbf{x} and \mathbf{y} .



Examples of a Norm

$$|\vec{a}| \equiv \|\vec{a}\| = \left[\sum_{i=1}^n a_i^2 \right]^{\frac{1}{2}}$$

This is called the **Euclidean Norm**.

$$|\vec{a}|_p \equiv \|\vec{a}\|_p = \left[\sum_{i=1}^n a_i^p \right]^{\frac{1}{p}}$$

This is the *p*-norm.

If $\mathbf{y} = \sum_{i=1}^n \alpha_i \mathbf{v}_i$ for some set of n linearly independent vectors \mathbf{v}_i . Then

$$\|\mathbf{y}\|_{\mathbf{V}} \equiv \sum_{i=1}^n |\alpha_i| \quad \text{is a norm with respect to the matrix } (\mathbf{v}_1 | \mathbf{v}_2 | \cdots | \mathbf{v}_n).$$



Metric Spaces

We've already alluded to some issues for measuring mathematical things.
We need tools that allow us to *rank* mathematical objects and *rank relationships* between and among mathematical objects.

1. $\rho(x, y) = \rho(y, x) \geq 0$. **Positivity**
2. $\rho(x, x) = 0$; $\rho(x, y) = 0 \Leftrightarrow x = y$. **Homogeneity**
3. $\rho(x, z) + \rho(z, y) \geq \rho(x, y)$. **Triangle Inequality**

where ρ is a function defined on $M \times M$, hence (M, ρ) is called a Metric Space

The metric function ρ can take on many forms!

We define ρ based on what is necessary and convenient!



Various Forms of Metrics

$$\rho_{\infty}(x, y) = \operatorname{Max}_i |x_i - y_i|$$

$$|x_i - y_i| \leq |x_i - z_i| + |z_i - y_i|$$

$$\rho(f(x), g(x)) = \operatorname{Max}_{x \in [a, b]} |f(x) - g(x)|$$

$$\rho(\vec{a}, \vec{b}) = \|\vec{a} - \vec{b}\| = \left[\sum_{i=1}^n (a_i - b_i)^2 \right]^{\frac{1}{2}} = (\vec{a} - \vec{b}, \vec{a} - \vec{b})^{\frac{1}{2}}$$



Orthogonality and Angles

We can posit that for any two mathematical objects, there corresponds a number between -1 and 1 that conveys how these objects are related.

$$\cos \theta = \frac{\langle \mathbf{a}, \mathbf{b} \rangle}{\|\mathbf{a}\| \|\mathbf{b}\|}$$

We can say that this corresponds to the angle between two vectors.



Convergent Sequences

Limit Points: A sequence $\{x_n\}$ is convergent and has a limit point

$x^* \in M$, if $\rho(x_n, x^*) \rightarrow 0$ as $n \rightarrow \infty$,

$$x_n \rightarrow x^*$$

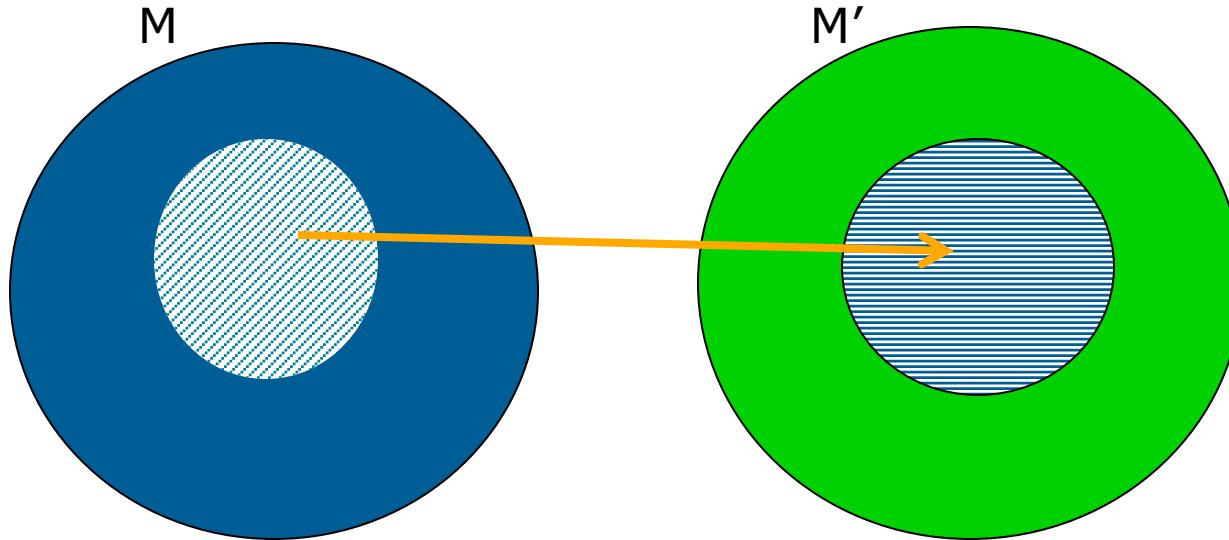


Mappings in Metric Spaces

- Define two sets and two Metric Spaces:

$$D(f) = \{x : x \in M \wedge f(x) \text{ is defined}\}$$

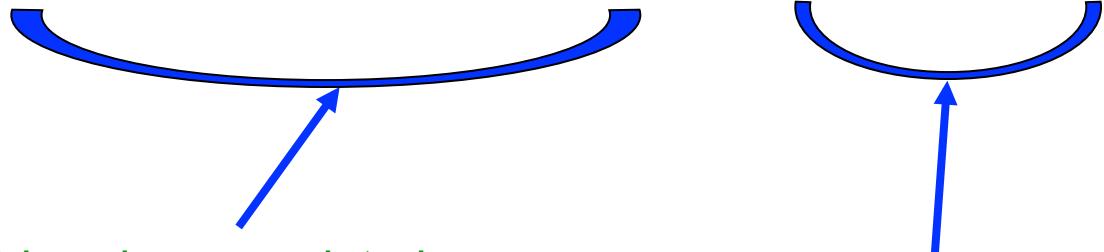
$$R(f) = \{x' : x' \in M' \wedge \exists x \in D(f) \ni x' = f(x)\}$$





Boundedness

$$\exists k \geq 0, \exists \forall x, y \in D(f) \\ \rho'(f(x), f(y)) \leq k \rho(x, y)$$



Metric value associated
with the range.

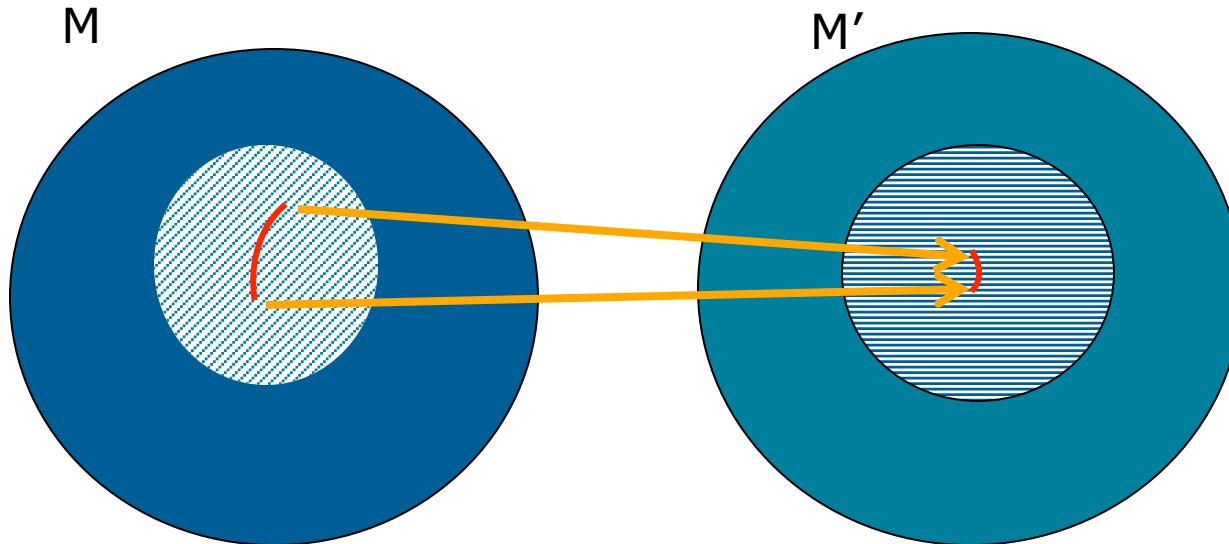
Metric value associated
with the domain.



Contraction Mapping

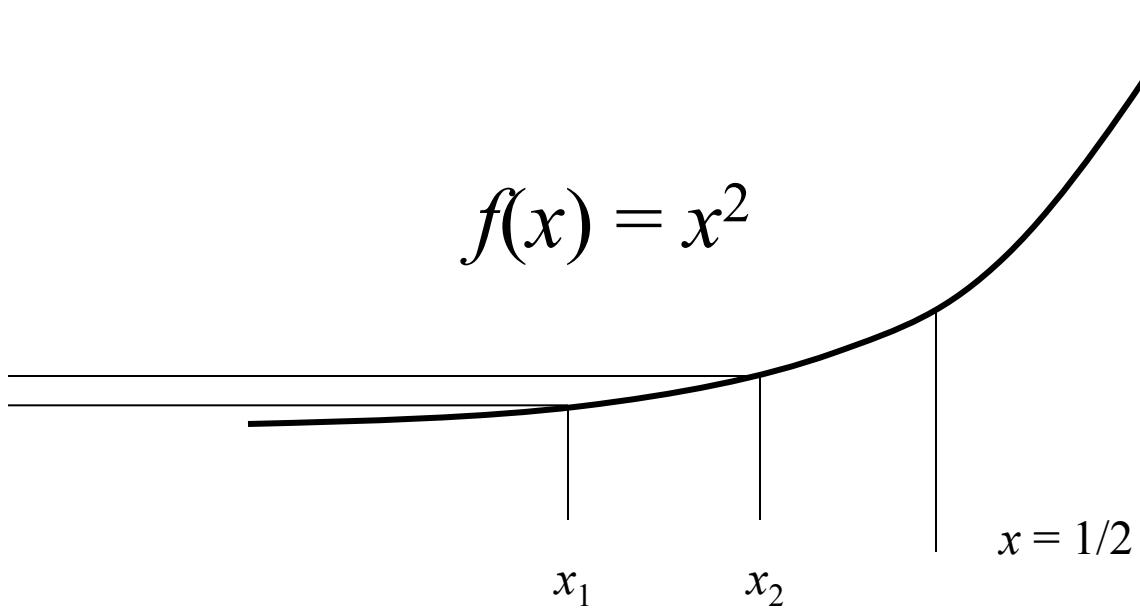
A Mapping is a contraction if it is bounded and

$$\begin{aligned} \exists k \geq 0 \wedge 0 \leq k < 1, \exists \forall x, y \in D(f) \\ \rho'(f(x), f(y)) \leq k\rho(x, y) \end{aligned}$$





An Example



Define $\rho(x_1, x_2) = |x_1 - x_2|$



A Useful Fact

$\Delta y = f'(\xi)\Delta x$ where ξ is some "intermediate" value.

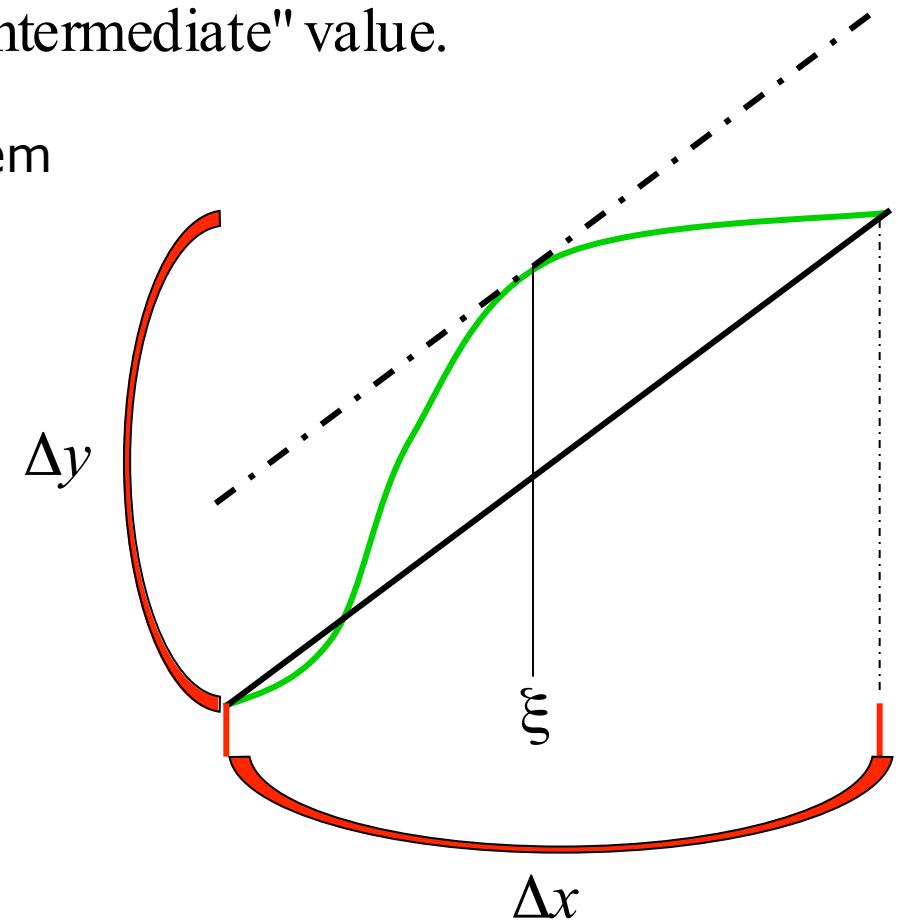
The Mean Value Theorem

$$|\Delta y| = |f(x_1) - f(x_2)|$$

$$= |f'(\xi)\Delta x|$$

$$\leq |f'(\xi)| |\Delta x|$$

$$= f'(\xi) |x_1 - x_2|$$





Putting It All Together

$$|\Delta y| = |f(x_1) - f(x_2)| = \rho(f(x_1), f(x_2))$$

Therefore,

$$\rho(f(x_1), f(x_2)) \leq f'(\xi) \rho(x_1, x_2)$$

and since $0 \leq x_1, x_2 < \frac{1}{2}$, and $f'(x) = 2x$

then it follows that $f'(\xi) < 1$

$\forall x, y \in D(f)$ and $0 \leq k < 1$

$$\rho(f(x), f(y)) \leq k \rho(x, y)$$

is a Contraction Mapping



A Numerical Example

$$\rho\left(\frac{1}{4}, \frac{1}{2}\right) = \frac{1}{4}$$

$$\begin{aligned}\rho\left(f\left(\frac{1}{4}\right), f\left(\frac{1}{2}\right)\right) &= \rho\left(\left(\frac{1}{4}\right)^2, \left(\frac{1}{2}\right)^2\right) \\ &= \rho\left(\frac{1}{16}, \frac{1}{4}\right) \\ &= \frac{3}{16}\end{aligned}$$

Note that $\frac{3}{16} < \frac{1}{4}$



How Would You Use This in The Context of Fixed Points?

$\forall x_1, x_2 \in D(f)$ and $0 \leq k < 1$

$$\rho'(f(x_1), f(x_2)) \leq k\rho(x_1, x_2)$$

$\forall x_1, x_2 \in D(f)$ and $0 \leq k < 1$

$$\rho'(x^*, f(x_2)) \leq k\rho(x^*, x_2)$$

because by definition, a fixed point x^* is such that

$$f(x^*) = x^*$$



Dynamical Systems

- Neural Networks can be fashioned into dynamical systems.
- Feeding outputs back as inputs and cycling them ala fixed point exercise.
- How does such a system behave?
- Stay tuned.

A SIMPLE PROOF OF THE CHAIN RULE

© 2015 by Mark Fleischer

1 Let's Start at the Very Beginning

Let's look at the basic definition of the derivative where we'll emphasize certain, specific patterns that will help us navigate through a more complicated-looking derivative. First, the derivative a function $g(x)$ should be quite familiar:

$$\frac{dg}{dx} = \lim_{h \rightarrow 0} \frac{g(x+h) - g(x)}{h} \quad (1)$$

which tells us what the relative change is of the function value g compared to changes in the value of x . In other words, we want to know how fast g changes if we slightly change the value of its argument by a small amount h . Notice that the increment h is thus *added to the argument of the function* which is why we refer to this as the derivative of g *with respect to x* — because x is the variable we are *perturbing* with h . The value of h simply represents the difference between $x + h$ and x — the “run” and $g(x+h) - g(x)$ is the “rise” and their ratio corresponds to the slope and in the limit corresponds to the derivative. When we add the increment h directly to the argument of the function, as is the case here, we can write the derivative as $dg(x)/dx$ or simply dg/dx — again, the derivative of g *with respect to x* which tells us the proportional change in g when we change x by adding h to x . Also notice that the increment h in the numerator of (1) is the very same increment that's in the denominator of (1). This is necessary so that we can correctly compute the rise over the run. If the h in the numerator was different than the one in the denominator, we wouldn't actually be calculating the slope would we.

2 What About a Function of a Function?

Now let's look at a compound function: $f(g(x))$. In this case, as before, the argument or variable of the function g is x , but the argument or variable of the function f is g ! So changing x changes g and changes in g change the value of f . So if we want to define the derivative of f *with respect to x* , we have to add the increment h to the variable x and so by the definition of the derivative and using the same general form as in (1),

$$\frac{df}{dx} = \lim_{h \rightarrow 0} \frac{f(g(x+h)) - f(g(x))}{h} \quad (2)$$

So this is the derivative of f with respect to x based on the definition of a derivative. The problem with (2) however is that it's hard to actually use and work with. This is due to several issues. For one thing, the argument of the function f is the value of g and *not* the value x . Remember, that in (1) we added the increment h directly to the argument of that function. In this case, we're not exactly doing that. Instead of adding the increment h to the argument of f , we're adding h to the argument of g !

It might be helpful therefore if we could somehow modify the numerator of (2) so that the increment h is added to the argument of f as this is the function we're taking the derivative of and since the argument of f is g , we want to add the increment h to g and not to x . Thus, we need to somehow change the expression thusly:

$$f(g(x + h)) \longrightarrow f(g(x) + h)$$

or, ignoring the argument x since it is the argument of g (think of g as the argument or variable for f), we want to somehow establish a conversion

$$f(g(x + h)) \longrightarrow f(g + h)$$

in the numerator of (2). In this way, we would be adding the increment h directly to the argument of f (which is g) and not to the argument of the function g (which is x)! Do you see the difference? So the question is: is there a way to change $f(g(x + h))$ to look something more like $f(g(x) + h)$ or $f(g + h)$ in a legitimate way? To do this requires that we find some way of relating or converting $g(x + h)$ to $g(x) + h$ or something similar to it. Do you see a way to connect these two expressions? (*hint*: look at equation (1)).

From (1) we see quite directly (after doing just a little algebra) that

$$g(x + h) = g(x) + h \frac{dg}{dx} \quad (3)$$

Now we're getting somewhere. Note also that in (3) as $h \rightarrow 0$ so does $h \frac{dg}{dx}$ which means the increment goes to 0 just like it's supposed to in the definition of a derivative like in (1). So let's simplify some notation and write $h \frac{dg}{dx}$ as \hat{h} so that (3) looks like

$$g(x + h) = g(x) + \hat{h} \quad (4)$$

which looks much cleaner and simpler. Now, we can substitute (4) for $g(x + h)$ in equation (2) which now looks like

$$\frac{df}{dx} = \lim_{h \rightarrow 0} \frac{f(g(x) + \hat{h}) - f(g(x))}{h} \quad (5)$$

or, ignoring the argument x of g to emphasize that it is g that is the argument of f , (5) can be rewritten as

$$\frac{df}{dx} = \lim_{h \rightarrow 0} \frac{f(g + \hat{h}) - f(g)}{h} \quad (6)$$

Now we see very clearly that g is the variable or argument of f and that we've added a type of increment \hat{h} directly to the argument of f just like in (1). Thus, the right-hand side of (6) is beginning to look a lot like df/dg !

The only problem in (6) is that the increment \hat{h} in the numerator of (6) is not the same as the increment h in the denominator of (6). These two increments must be the same for there to be a proper evaluation of the slope and to correspond to the definition of a derivative as in (1). How can we get these two increments to be the same?

Well, if we multiply the quantity in (6) by h/\hat{h} then the h 's would cancel as in

$$\frac{df}{dx} = \lim_{h \rightarrow 0} \left[\frac{f(g + \hat{h}) - f(g)}{\kappa} \right] \cdot \frac{\kappa}{\hat{h}}$$

and we'd end up with an \hat{h} in both the numerator and denominator and that really would be df/dg ! But we can't just willy-nilly multiply the right-hand side of (6) by h/\hat{h} just to make it look like we want it to look — doing that could change its value. The only multiplication that we can do to (6) *without changing its value* is to multiply (somehow) by a factor that is *always* 1! So instead of just multiplying (6) by h/\hat{h} , we're going to multiply by

$$\frac{h}{\hat{h}} \cdot \frac{\hat{h}}{h}$$

which is just another way of writing 1. Thus, (6) becomes

$$\frac{df}{dx} = \lim_{h \rightarrow 0} \left[\frac{f(g + \hat{h}) - f(g)}{h} \right] \cdot \frac{h}{\hat{h}} \cdot \frac{\hat{h}}{h} \quad (7)$$

and the value is still the same. Notice that we can now cancel the h 's thusly:

$$\frac{df}{dx} = \lim_{h \rightarrow 0} \left[\frac{f(g + \hat{h}) - f(g)}{\kappa} \right] \cdot \frac{\kappa}{\hat{h}} \cdot \frac{\hat{h}}{h} \quad (8)$$

and then moving the \hat{h} in the denominator into the denominator of the bracketed quantity we get

$$\frac{df}{dx} = \lim_{h \rightarrow 0} \left[\frac{f(g + \hat{h}) - f(g)}{\hat{h}} \right] \cdot \frac{\hat{h}}{h} \quad (9)$$

and now we see that the quantity in brackets is actually $\frac{df}{dg}$ because g is the argument of f to which the increment \hat{h} has been added, the increments in the numerator and denominator of the bracketed quantity are the same, and as $h \rightarrow 0$ so does \hat{h} . Thus, everything in the brackets corresponds to the definition of a derivative of the function f with respect to g !

So what about the fraction \hat{h}/h on the right of the brackets in (9)? Well, recall from (3) and (4) that $\hat{h} = h \frac{dg}{dx}$ therefore

$$\frac{\hat{h}}{h} = \frac{h(dg/dx)}{h} = \frac{dg}{dx} \quad (10)$$

because the h 's cancel and so (9) becomes

$$\frac{df}{dx} = \lim_{h \rightarrow 0} \left[\frac{f(g + \hat{h}) - f(g)}{\hat{h}} \right] \cdot \frac{dg}{dx} \quad (11)$$

or

$$\frac{df}{dx} = \frac{df}{dg} \cdot \frac{dg}{dx} \quad (12)$$

3 Final Thoughts

In our quest to understand the chain rule, we first had to use symbols and abstractions to clearly state what we meant by a derivative of some function with respect to some variable. Such a quantity represents the rate of change of a function's value as its argument is changed and we were able to write (1) to symbolize this concept. Then using good 'ole algebra and our knowledge of limits, we manipulated the symbols, all the while adhering to fundamental rules of mathematics to show a *chain* of reasoning and logic to arrive at the simple rule that you will use a lot — the chain rule — a multiplication of rates of change where one variable affects another. Remember, a chain is only as strong as its weakest link, so learn this rule well!

◇ ◇ ◇



Introduction to Neural Networks

**Johns Hopkins University
Engineering for Professionals Program**

605-447/625-438

Dr. Mark Fleischer

Copyright 2013 by Mark Fleischer

Module 3.1: Basic Symbolic Logic



This Sub-Module Covers ...

- Basic review of Symbolic Logic and Truth Tables.
- Rules of Inference.
- The Truth Value of Compound Statements
- Perceptrons and Logic.



What is ...

Truth?



... information or statements on which we can **act** with confidence.

Meaningful ... but vague!



Learning Truth

Young children and babies often learn how to assess things in their brand new world,
but often in a very **dualistic** fashion!





Let's keep things simple...

- Avoid all the vagaries of the human condition.
- Simplify issues ... make them amenable to analysis.
- Provide for a rich set of possibilities ... allow encoding of all the shades of gray.



Logic-Truth Tables

True = 1

False = 0

AND

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

OR

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1



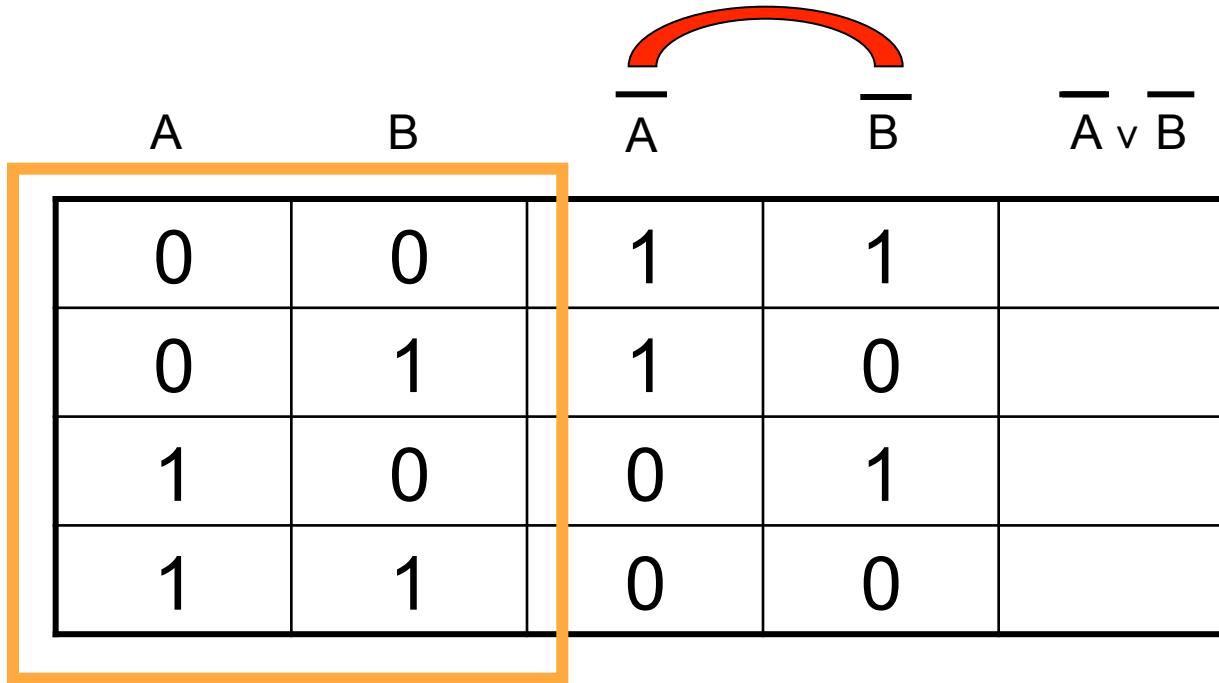
Logic-Truth Tables

NAND

A	B	$\overline{A \wedge B}$
0	0	1
0	1	1
1	0	1
1	1	0



Logic-Truth Tables



A	B	\overline{A}	\overline{B}	$\overline{A} \vee \overline{B}$
0	0	1	1	
0	1	1	0	
1	0	0	1	
1	1	0	0	



Logic-Truth Tables

A	B	\bar{A}	\bar{B}	$\bar{A} \vee \bar{B}$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0



Logic-Truth Tables

NAND

A	B	$\overline{A \wedge B}$
0	0	1
0	1	1
1	0	1
1	1	0



Not A OR Not B

A	B	$\overline{A} \vee \overline{B}$
0	0	1
0	1	1
1	0	1
1	1	0

Logically Equivalent

$$\overline{A \wedge B} \equiv \overline{A} \vee \overline{B}$$



Rules of Inference

What does $A \Rightarrow B$ mean?

Recited often as

A implies B... or

If A then B...

But what does this mean?

Answer:

If A is a true statement, then B is a true statement.

Sometimes stated as:

A is sufficient for B, or

B is a necessary consequence of A.



Truth Table of $A \Rightarrow B$

A	B	$A \Rightarrow B$
0	0	1
0	1	1
1	0	0
1	1	1



Logic-Truth Tables

NAND

A	B	$\overline{A \wedge B}$
0	0	1
0	1	1
1	0	1
1	1	0

XOR

A	B	$A \otimes B$
0	0	0
0	1	1
1	0	1
1	1	0



Evaluating Compound Statements

NAND

A	B	$\overline{A \wedge B}$
0	0	1
0	1	1
1	0	1
1	1	0

Not A OR Not B

A	B	$\overline{A} \vee \overline{B}$
0	0	1
0	1	1
1	0	1
1	1	0

How would we determine the truth value of this statement?

$$\overline{A \wedge B} \Rightarrow \overline{A} \vee \overline{B}$$

A' B'



Compound Statements

A	B	$A \Rightarrow B$
0	0	1
0	1	1
1	0	0
1	1	1

A' B'

A	B	$\overline{A \wedge B}$	$\overline{\overline{A} \vee \overline{B}}$	$\overline{\overline{A \wedge B}} \Rightarrow \overline{\overline{\overline{A} \vee \overline{B}}}$
0	0	1	1	1
0	1	1	1	1
1	0	1	1	1
1	1	0	0	1

Tautology



Really Compound Statements!

$$[(A \Rightarrow B) \wedge (B \Rightarrow C)] \Rightarrow (A \Rightarrow C)$$

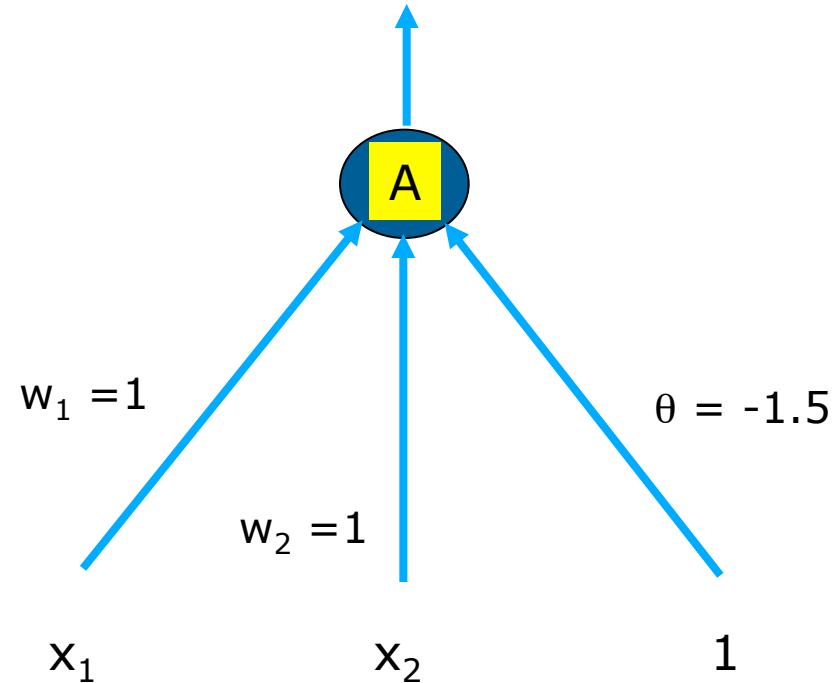
Rule of Inference
Basis of Deductive Reasoning

A	B	C	$A \Rightarrow B$	$B \Rightarrow C$	$(A \Rightarrow B) \wedge (B \Rightarrow C)$	$A' \quad B'$ 1 1	$A \Rightarrow C$	$[(A \Rightarrow B) \wedge (B \Rightarrow C)] \Rightarrow (A \Rightarrow C)$	1
0	0	0	1	1	1				1



Can Perceptrons Model AND?

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

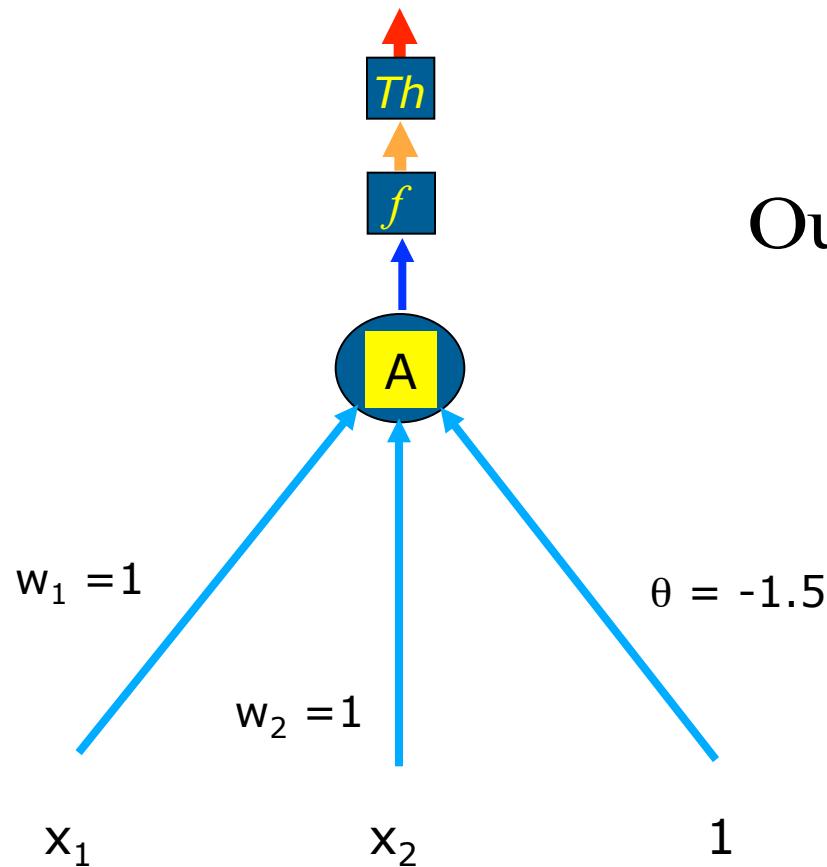


$$A = w_1x_1 + w_2x_2 + \theta = 1 + 1 - 1.5 = 0.5$$

We can now input this value into the activation function.



Threshold Logic

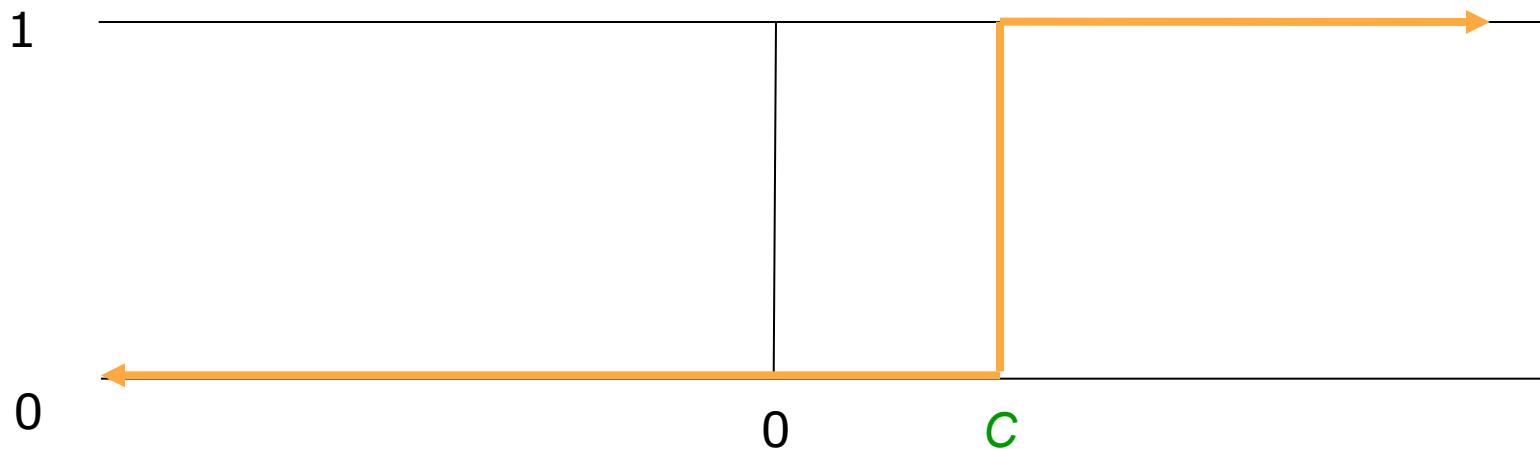


$$\text{Output} = \begin{cases} 1 & \text{if } f(A) \geq C \\ 0 & \text{otherwise} \end{cases}$$



Threshold Logic

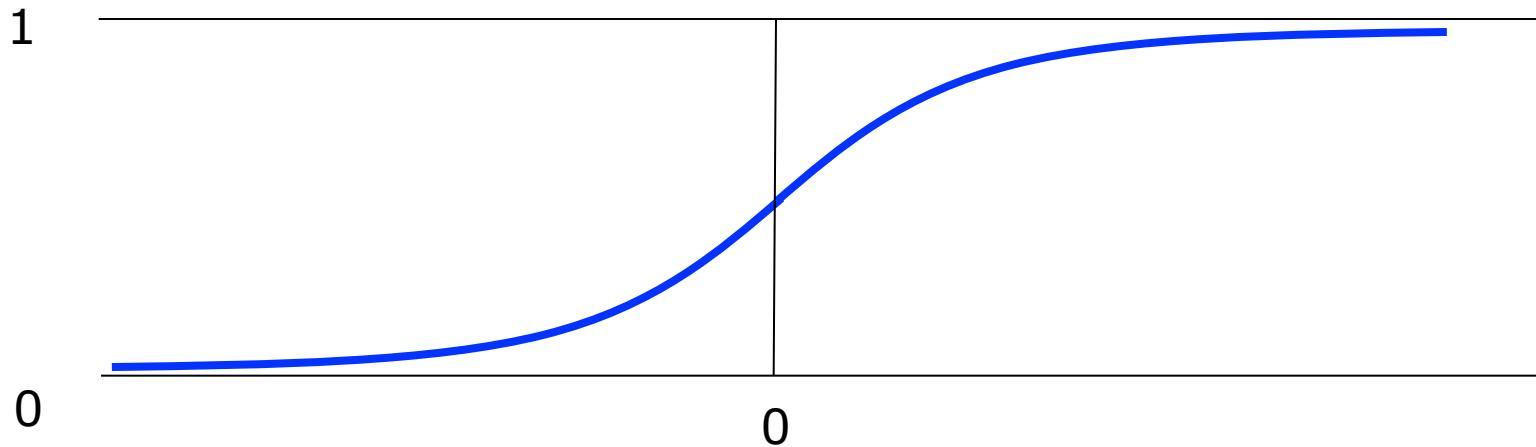
$$\text{Output} = \begin{cases} 1 & \text{if } f(A) \geq C \\ 0 & \text{otherwise} \end{cases}$$





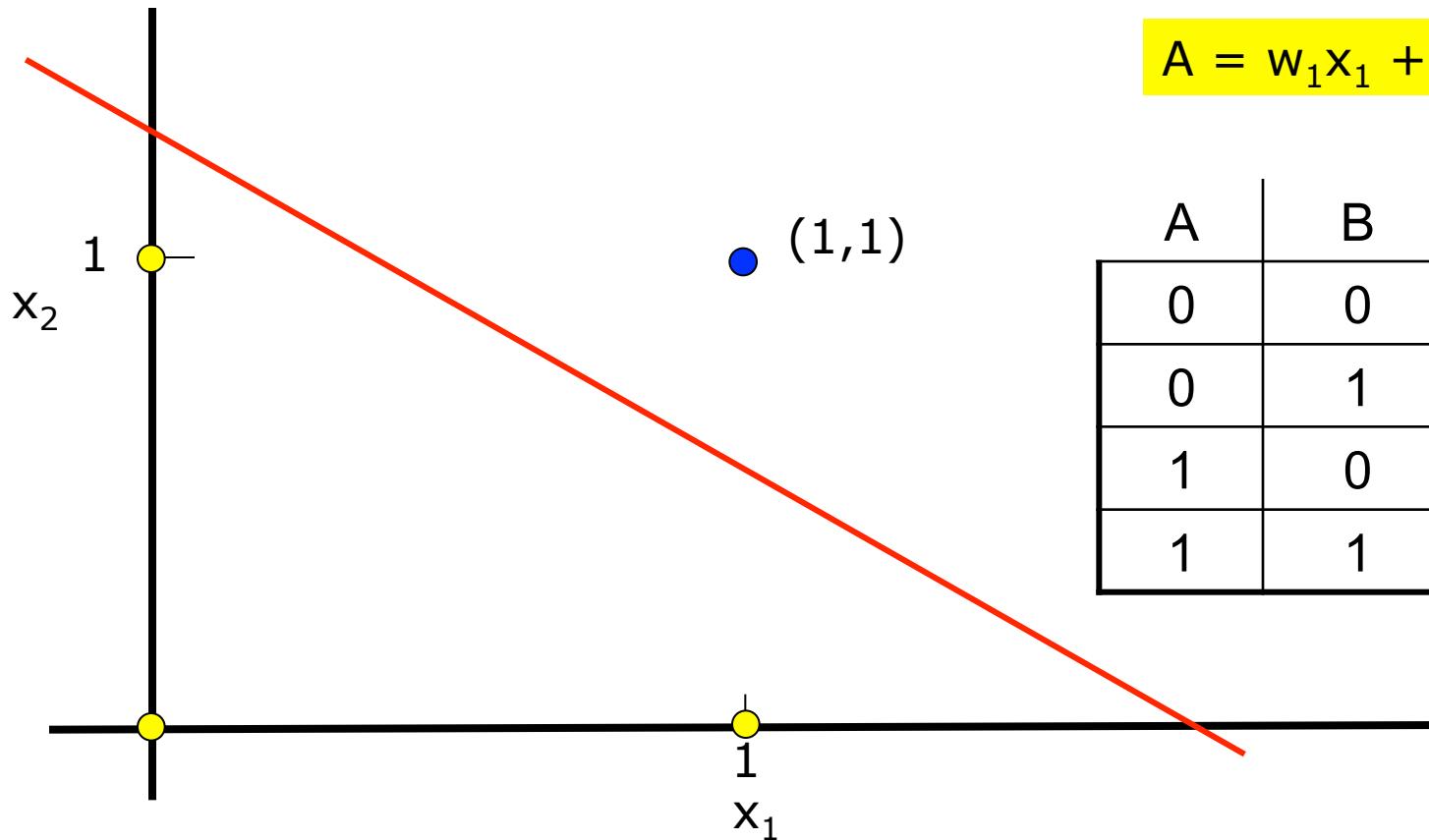
Activation Function

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





A New Angle to Perceptrons





Introduction to Neural Networks

**Johns Hopkins University
Engineering for Professionals Program**

605-447/625-438

Dr. Mark Fleischer

Copyright 2013 by Mark Fleischer

Module 3.2: Perceptrons and Logic

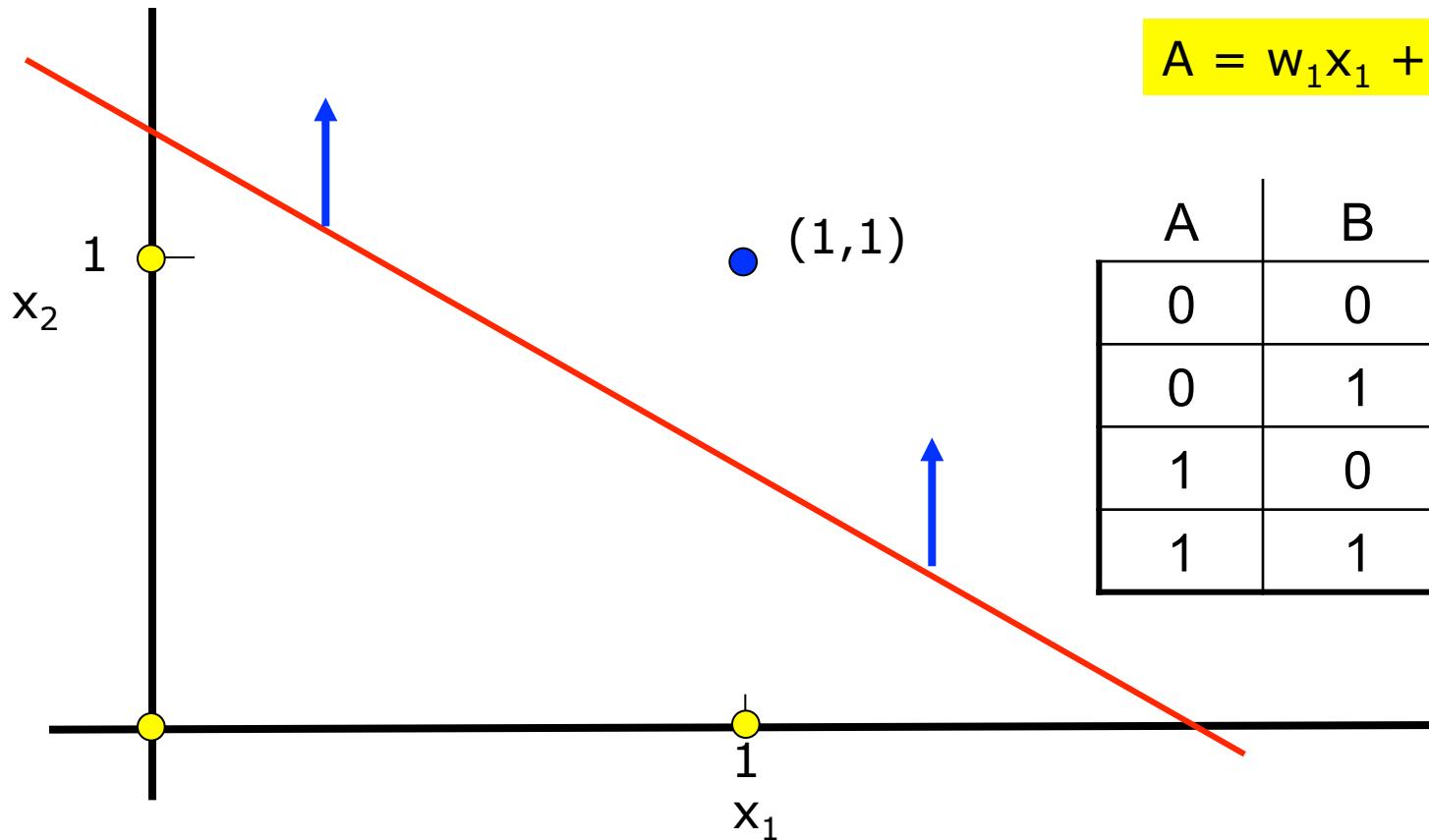


This Sub-Module Covers ...

- How Perceptrons can model logic statements.
- How Perceptron networks can model compound statements.
- Limitations on Perceptrons: The XOR problem.
- Second Order Perceptrons and the XOR problem.



A New Angle to Perceptrons



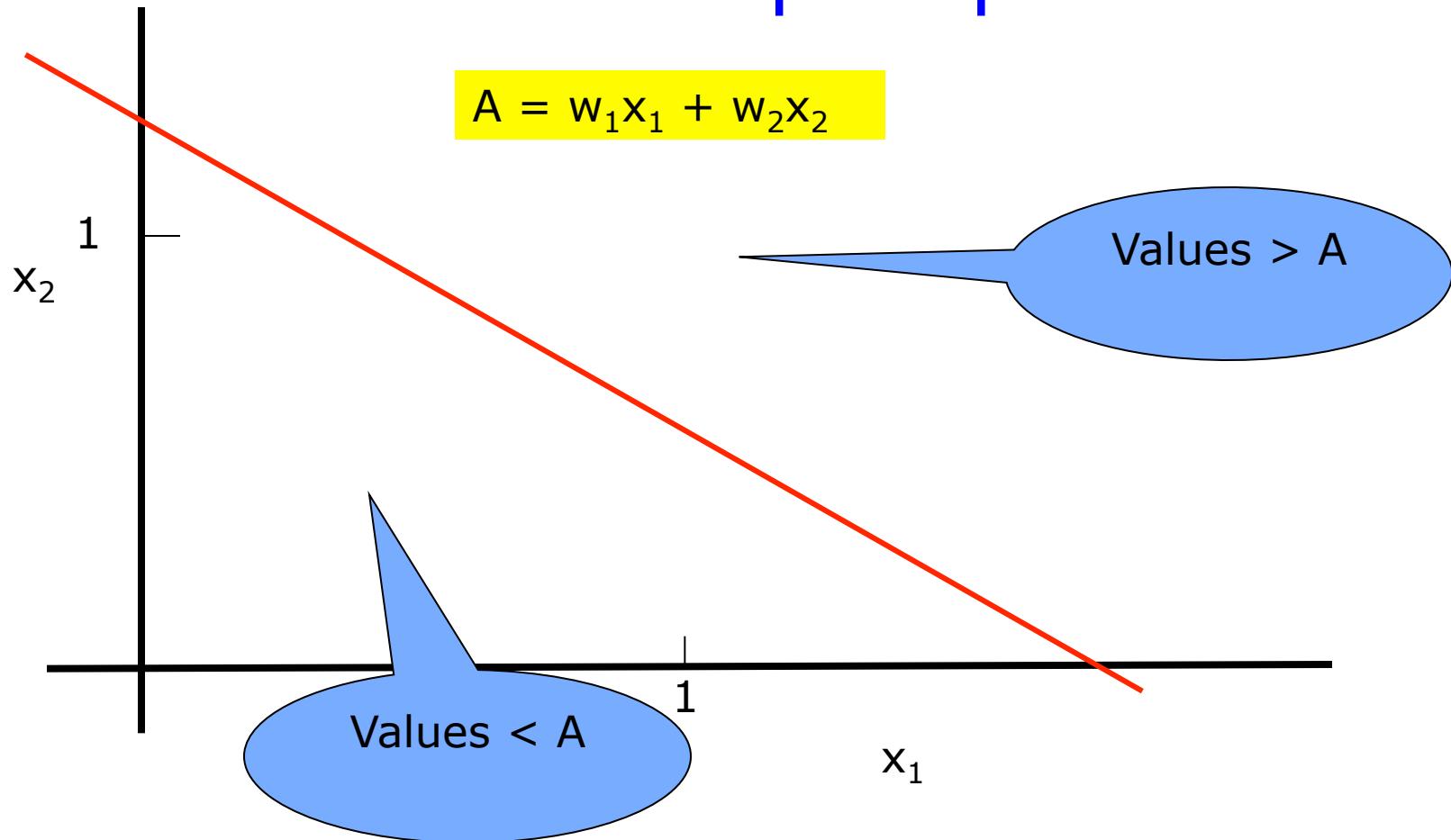


Linear Separability & Perceptrons

- **Inputs** $x_1, x_2 \dots$ values we use or control
- **Activity** $A = w_1x_1 + w_2x_2 + \theta$, a weighted function of the inputs
- A Monotonically increasing **Activation Function**, possibly coupled to some ‘threshold logic’ function.

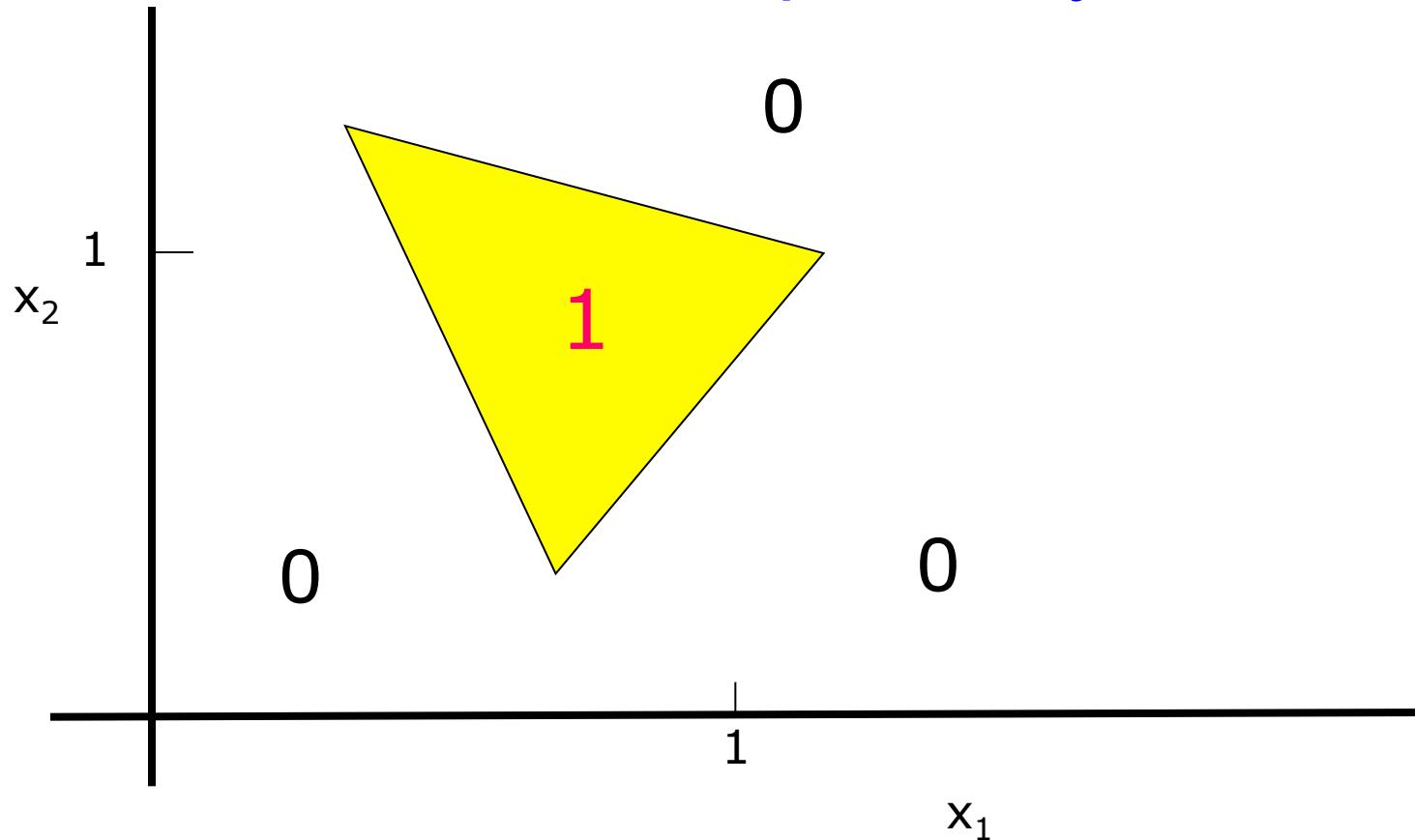


Linear Separability Bisects The Input Space



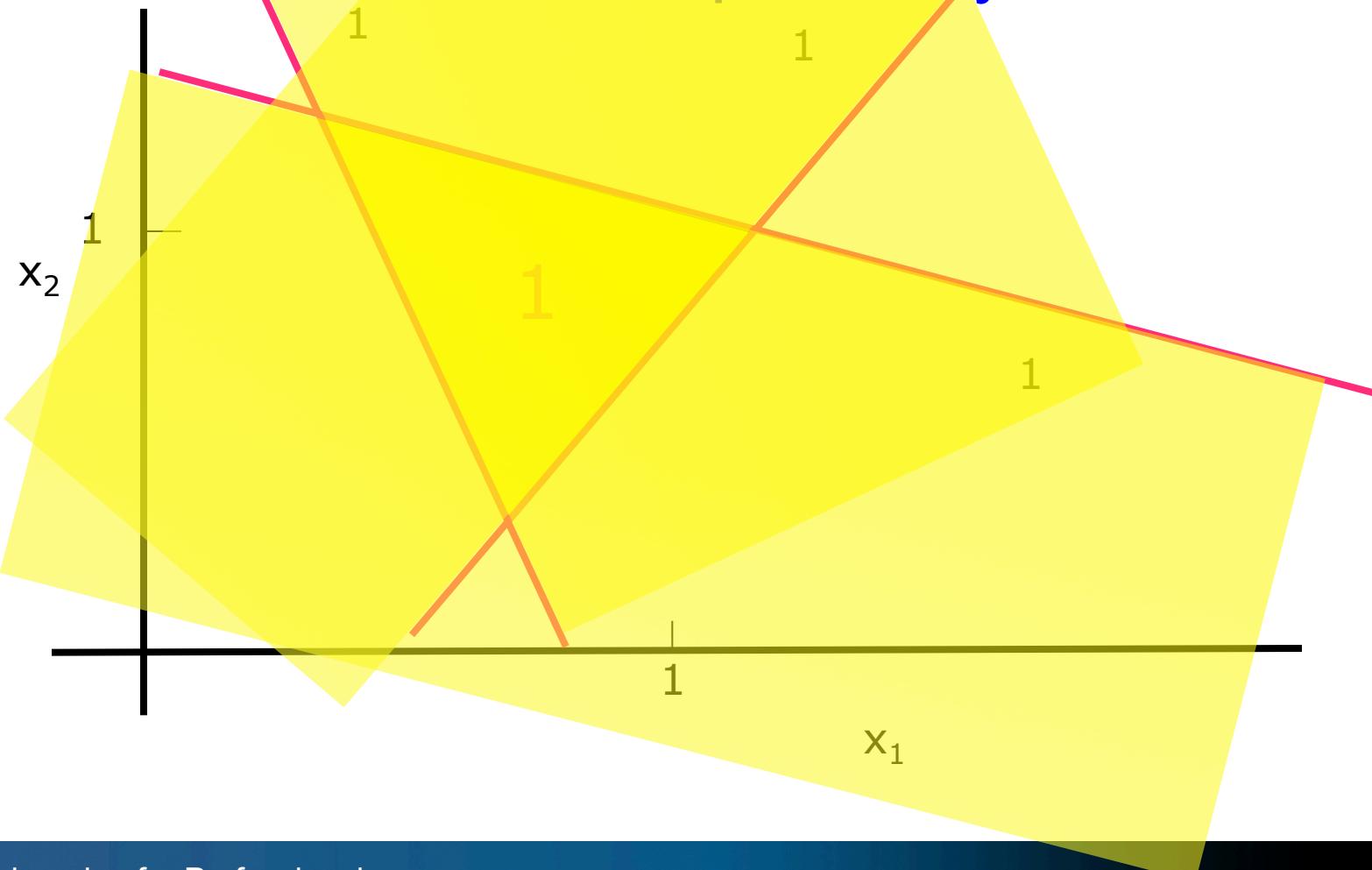


What Can We Do With Linear Separability?



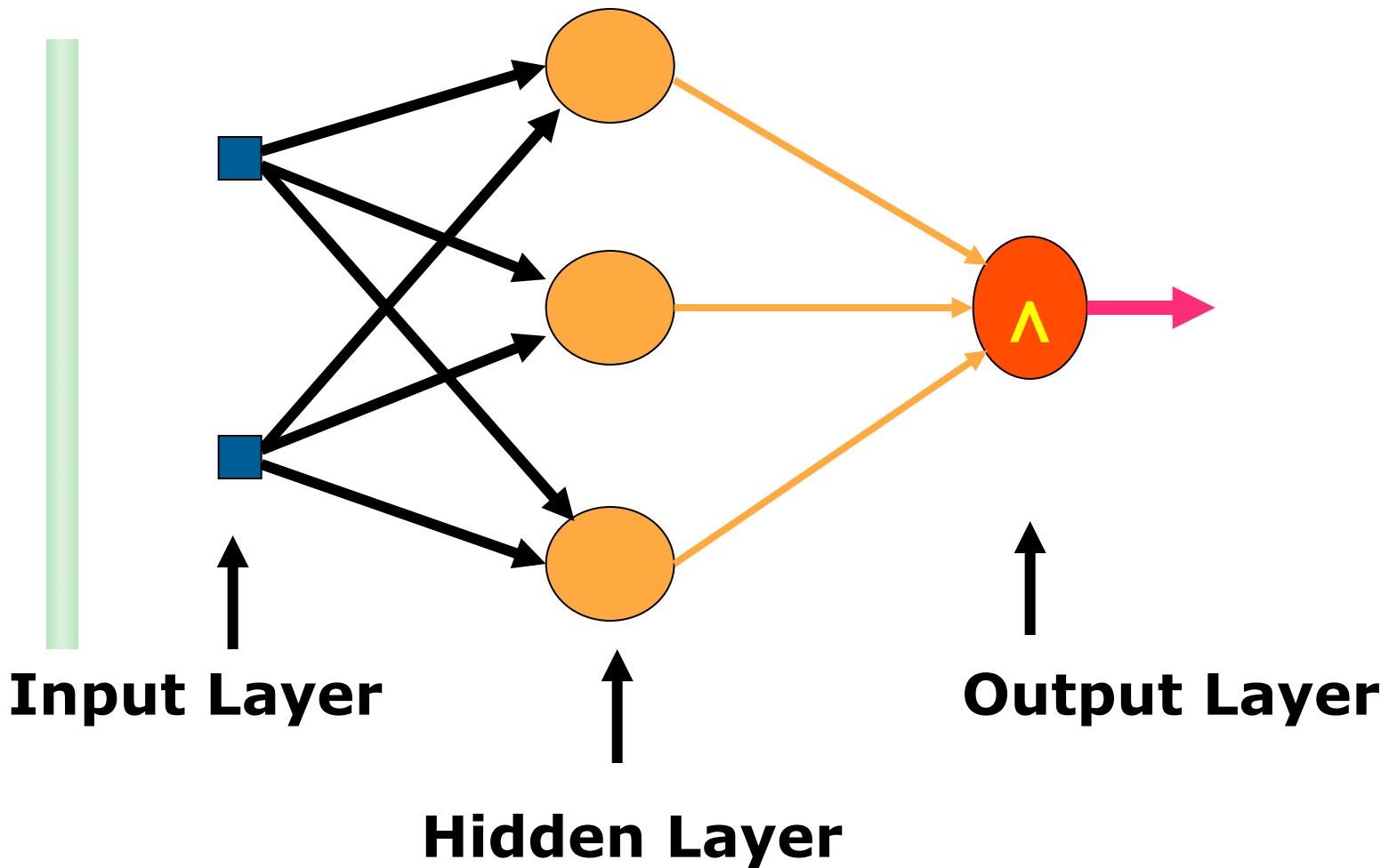


What Can We Do With Linear Separability?



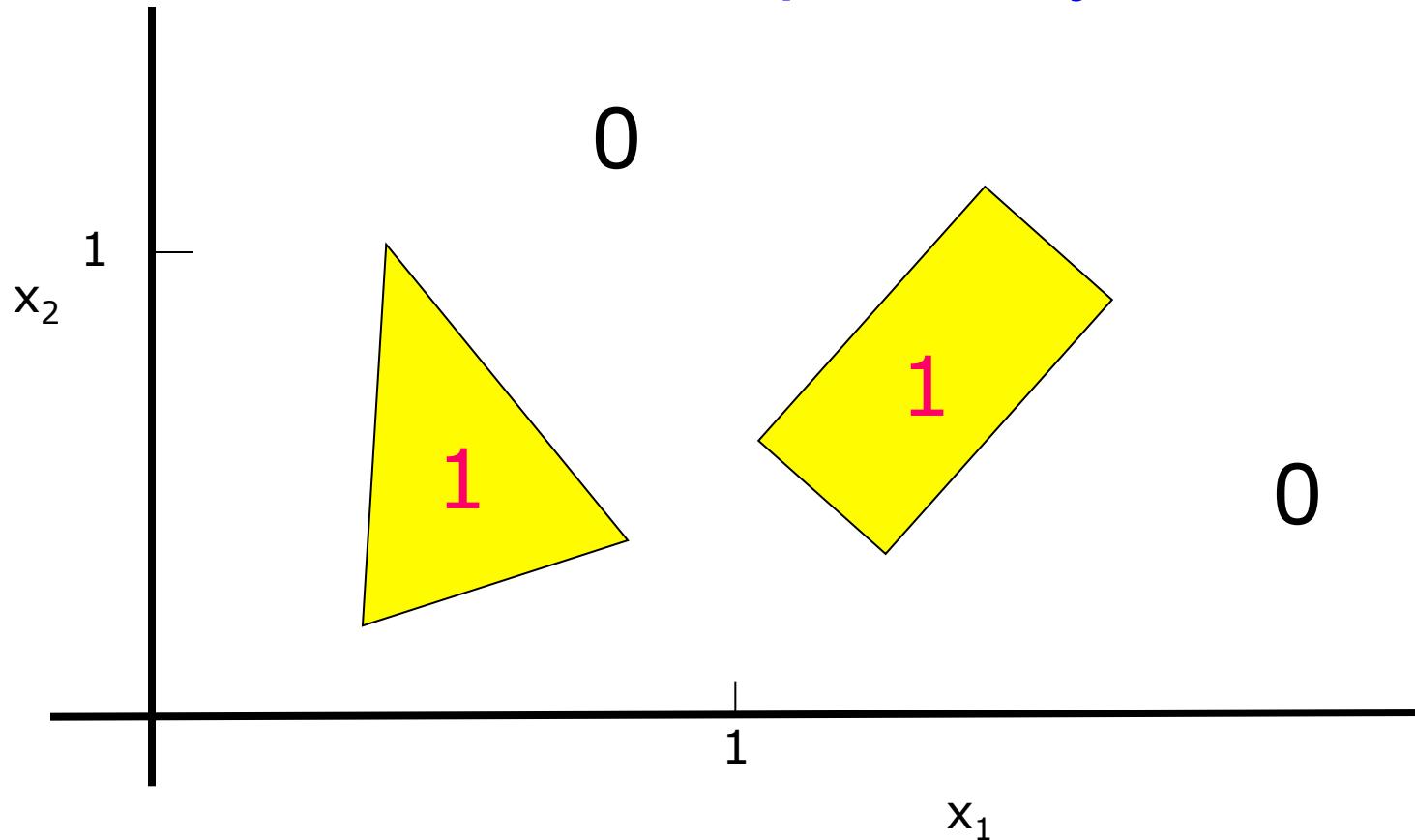


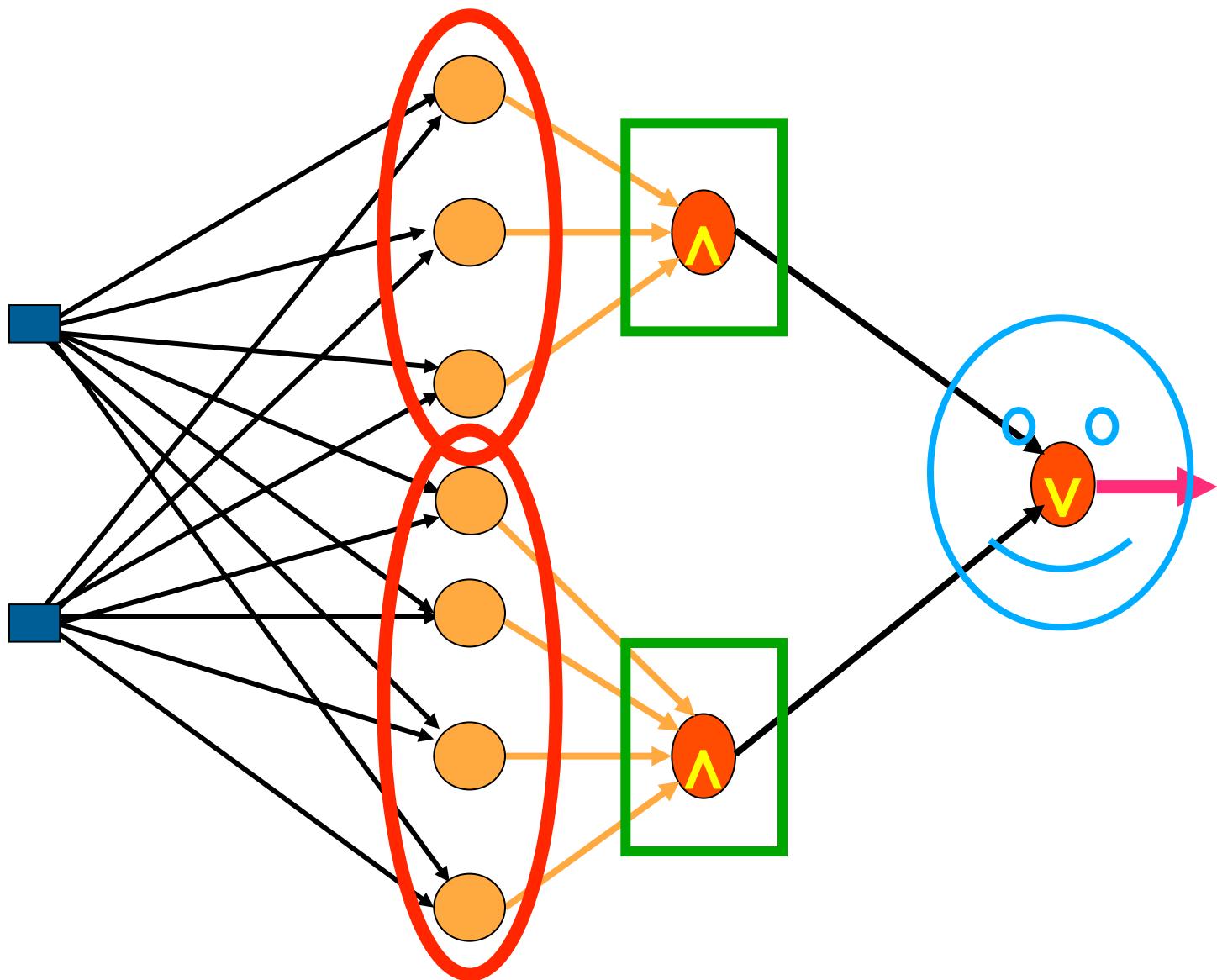
A Multi-layered Network





What Can We Do With Linear Separability?





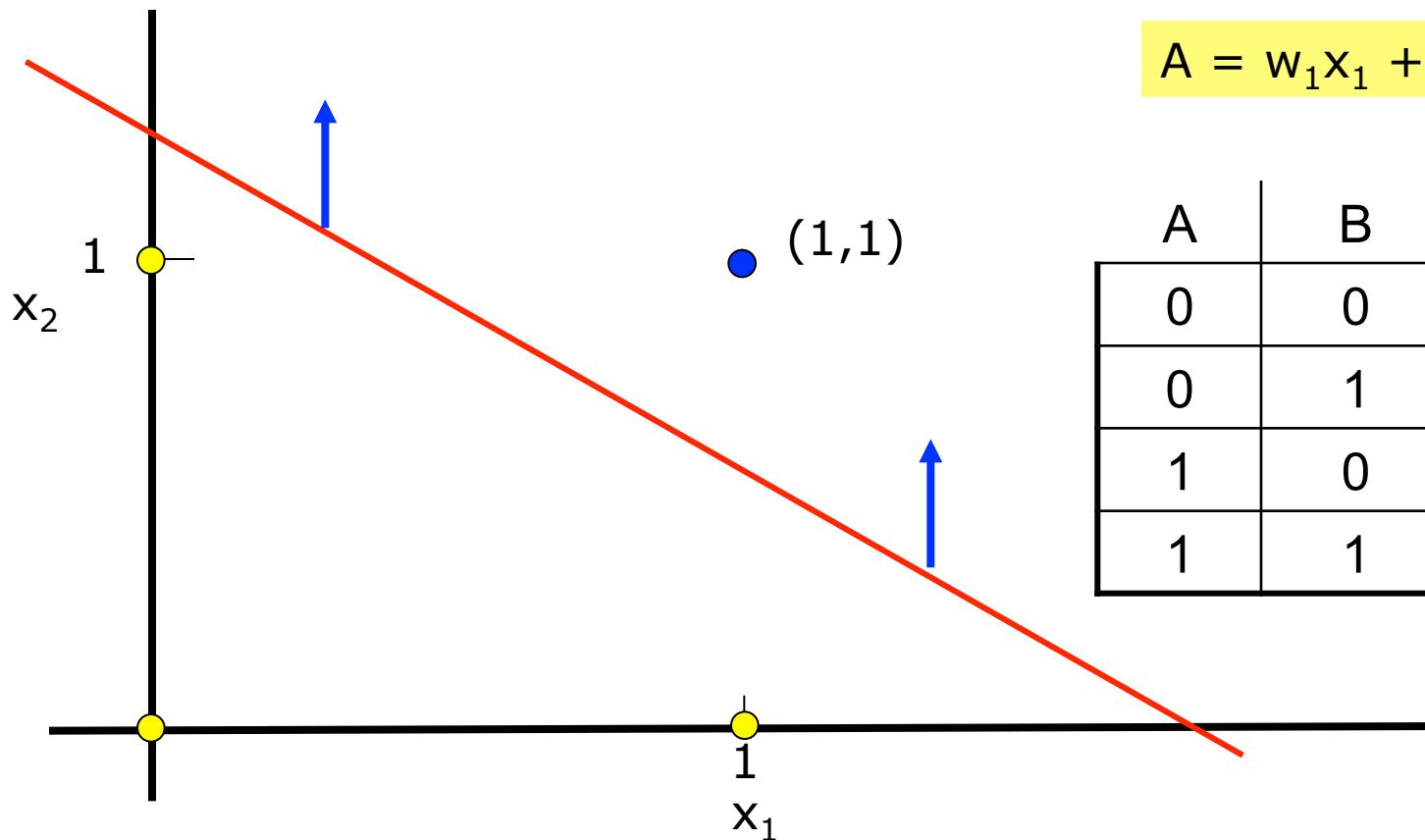


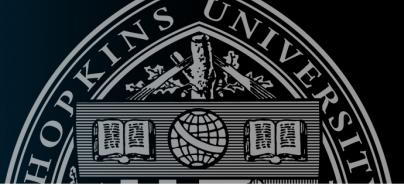
What Can We Do With Linear Separability?

- Segregate regions of the input-space
- Classification, categorization, labeling, etc.
- What do we need to do to enable this?
- Determine the weights!
- Is that all we need to do?

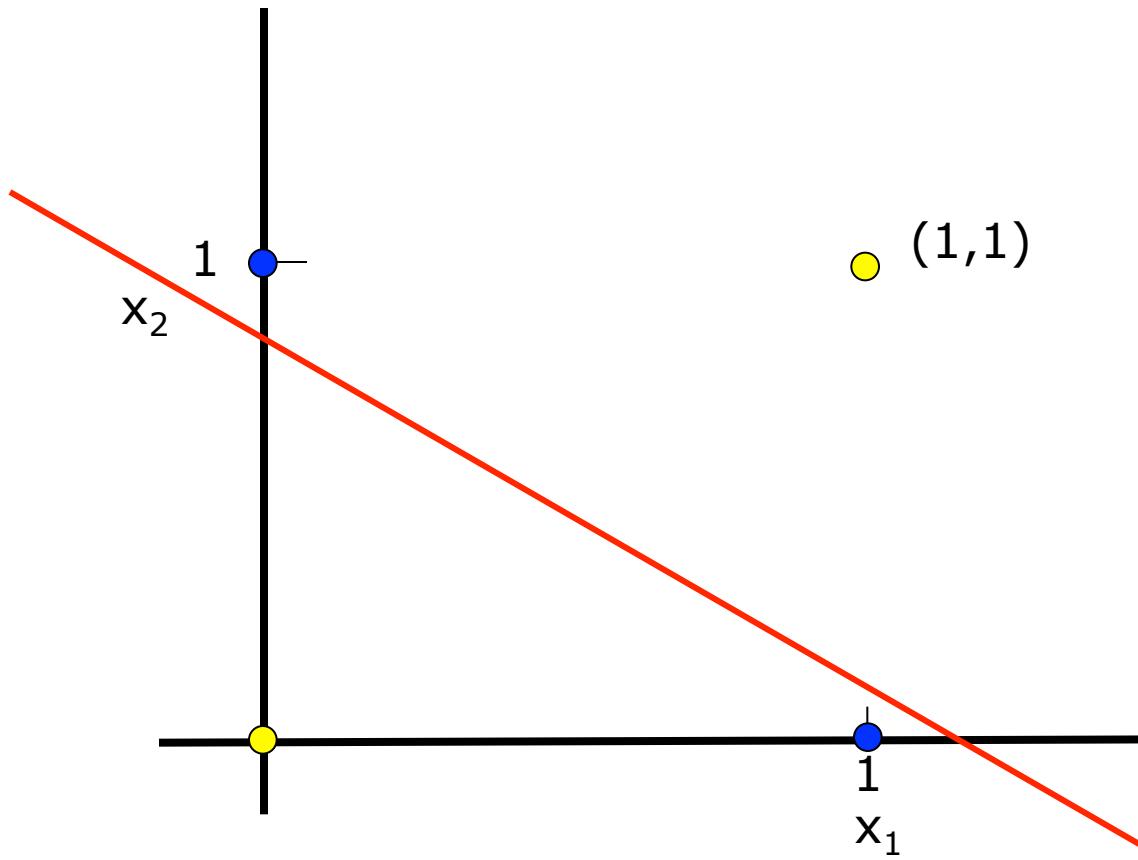


Can Do the AND and NAND





Can We Do XOR?



$$A = w_1x_1 + w_2x_2$$

A	B	$A \otimes B$
0	0	0
0	1	1
1	0	1
1	1	0



Still, We Can't Solve XOR

With a Single Perceptron

$$w_1x_1 + w_2x_2 + B = A$$

x_1	x_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

$$0 + 0 + B < 0$$

$$0 + w_2x_2 + B \geq 0$$

$$w_1x_1 + 0 + B \geq 0$$

$$w_1x_1 + w_2x_2 + B < 0$$

Adding together the two middle rows we get:

$$w_1x_1 + w_2x_2 + 2B \geq 0$$

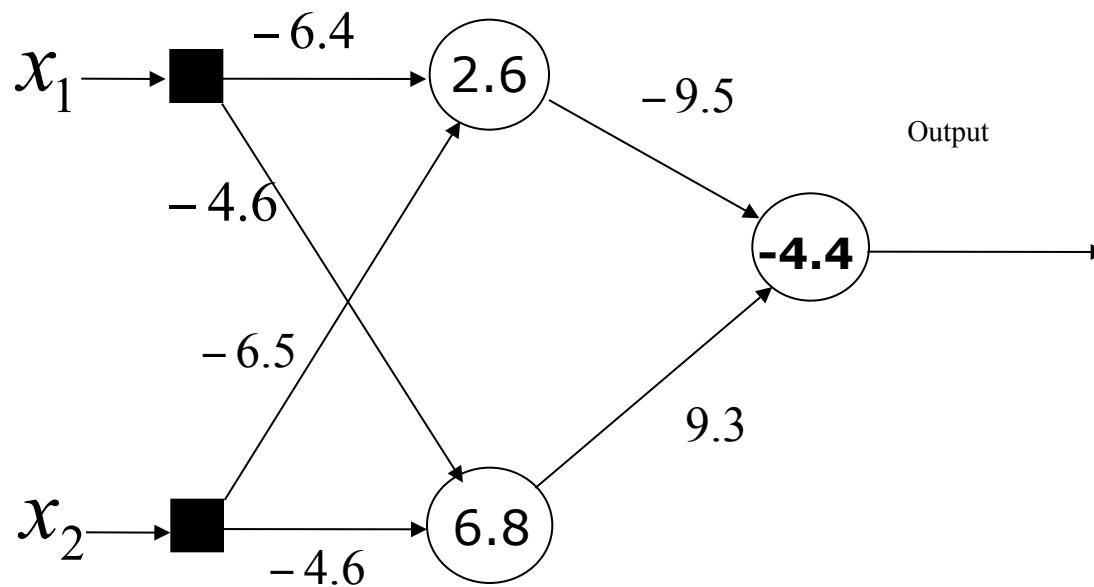
Adding together the first and last rows we get:

$$w_1x_1 + w_2x_2 + 2B < 0$$

Does there exist values of w_1 and w_2 that can yield this?
Is there any combination of values for w_1 and w_2 ?



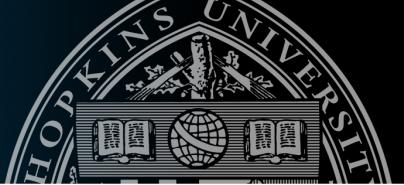
An Example of the XOR Problem



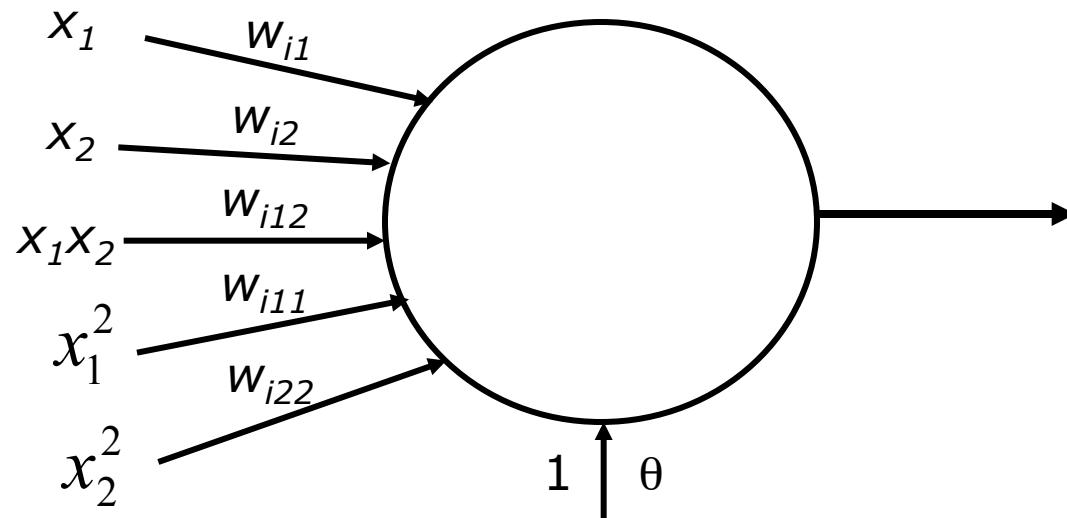


An Example of the XOR Problem

x_1	x_2	I to HN	$I + b$	HN out	Input to Output Node	$I + b$	Output
0	0	(0,0)	(2.6,6.8)	(0.93, 1.0)	0.465	-4	0.2->0
0	1	(-6.4,-4.6)	(-3.8,2.2)	(0.02,0.9)	8.37	3.96	0.98->1
1	0	(-4.6,-6.4)	<->	<->	<->	<->	<->
1	1	(-12.9,-9.2)	(-10.3,-2.4)	(0.0,0.08)	0.77	.77-4.4 =-3.6	0.03->0



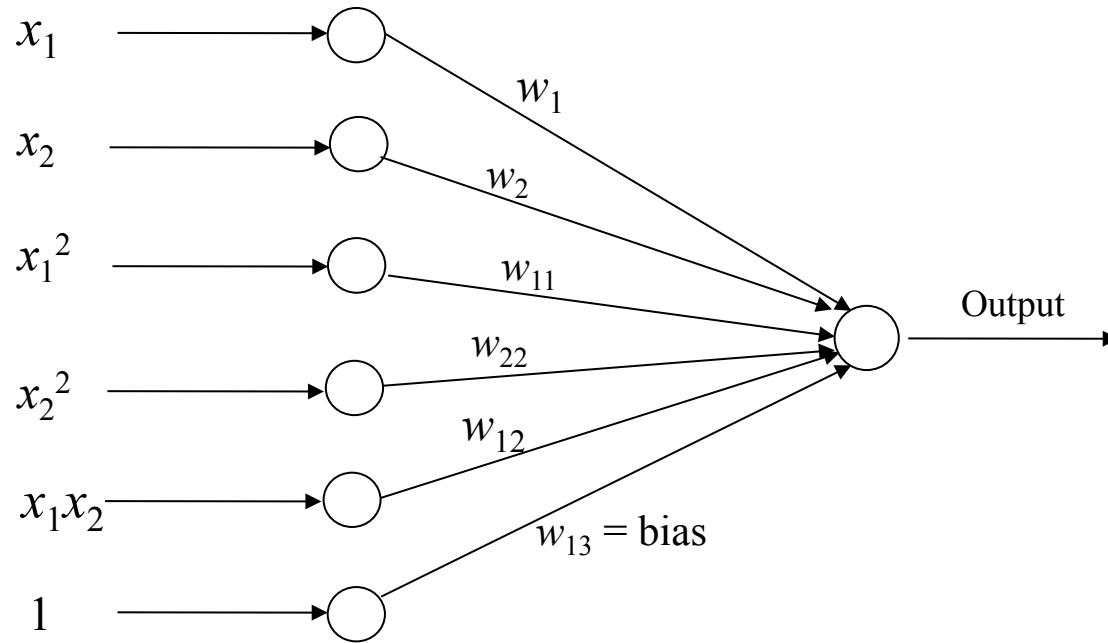
A Second-Order Perceptron



$$y_i = \sum_{j=1} w_{ij} x_j + \sum_{\substack{j=1 \\ k=1}} w_{ikj} x_k x_j + \theta$$



XOR and 2nd Order Perceptron



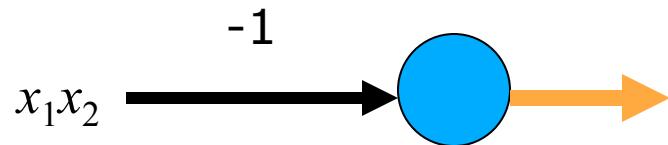
$$A = \sum_{j=1} w_{ij} x_j + \sum_{\substack{j=1 \\ k=1}} w_{ikj} x_k x_j + \theta$$

If we change the alphabet to 'bipolar' values of -1 and 1 AND set $w_{12} = -1$, then this can solve XOR.



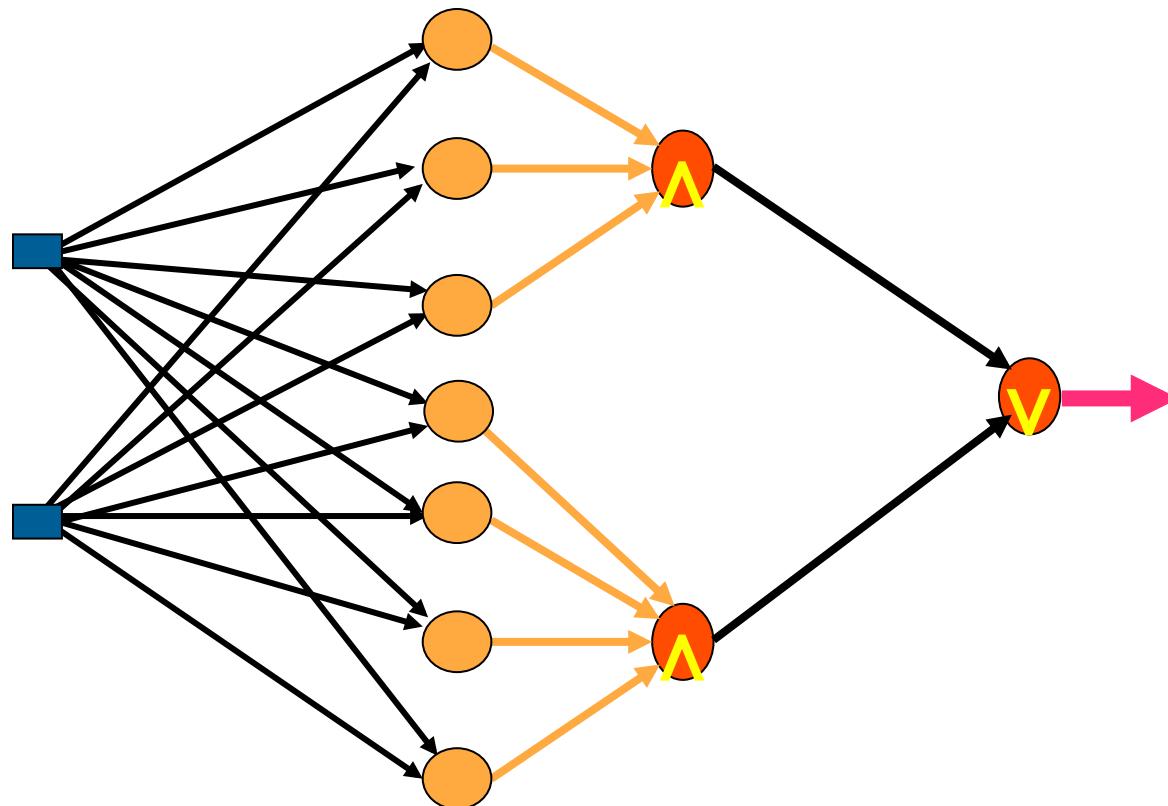
XOR

Input		Output
x_1	x_2	
-1	-1	-1
-1	1	1
1	-1	1
-1	-1	-1





How do we set the weights in a complicated network like this?





Introduction to Neural Networks

**Johns Hopkins University
Engineering for Professionals Program**

605-447/625-438

Dr. Mark Fleischer

Copyright 2013 by Mark Fleischer

Module 4.1: Training Perceptrons



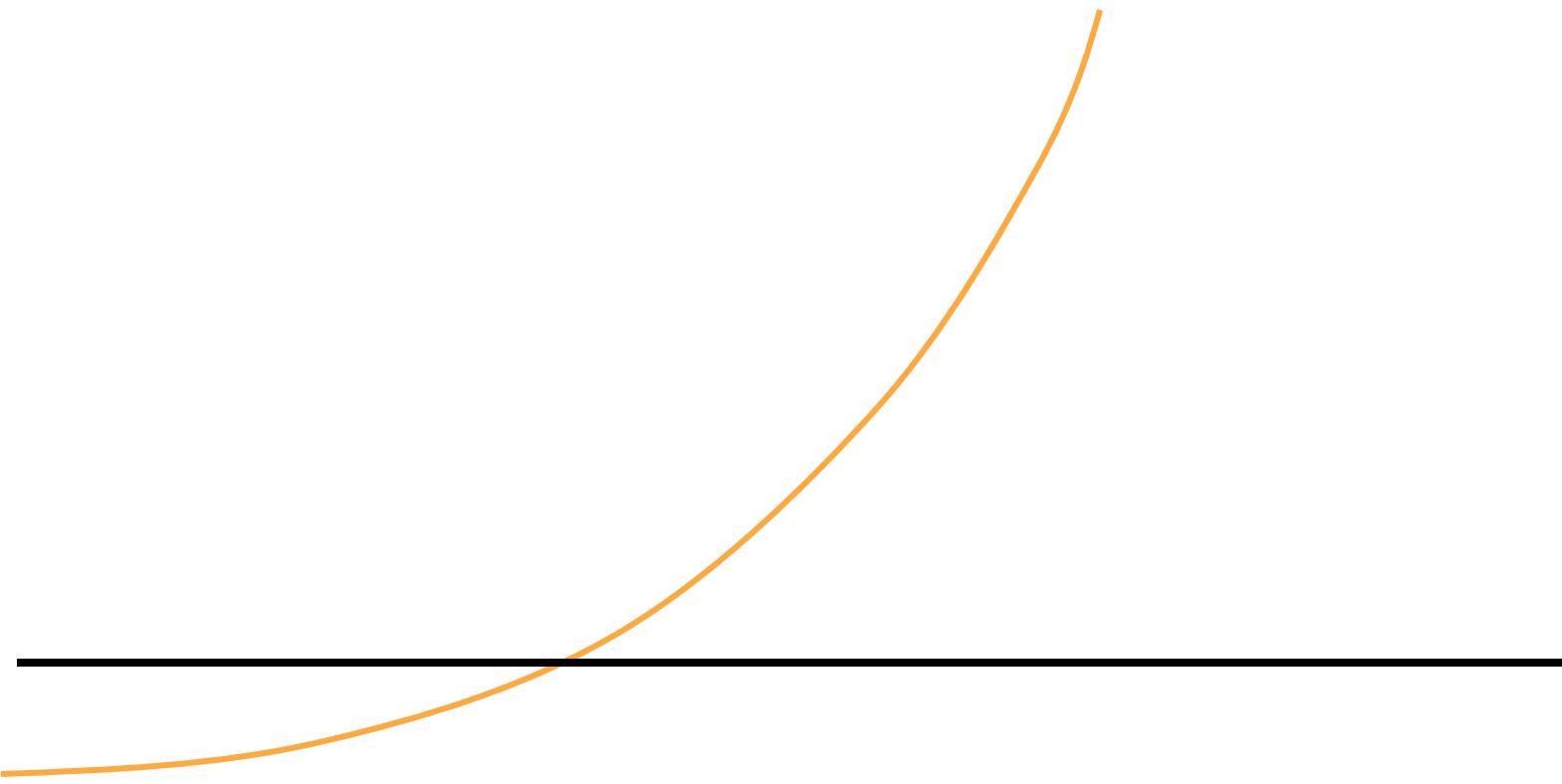
This Sub-Module Covers ...

- A dynamical systems approach and iterative method for solving root finding problems.
- The Method of Steepest Descent.
- A similar dynamical systems approach for training a Perceptron.



Look at Taylor Series

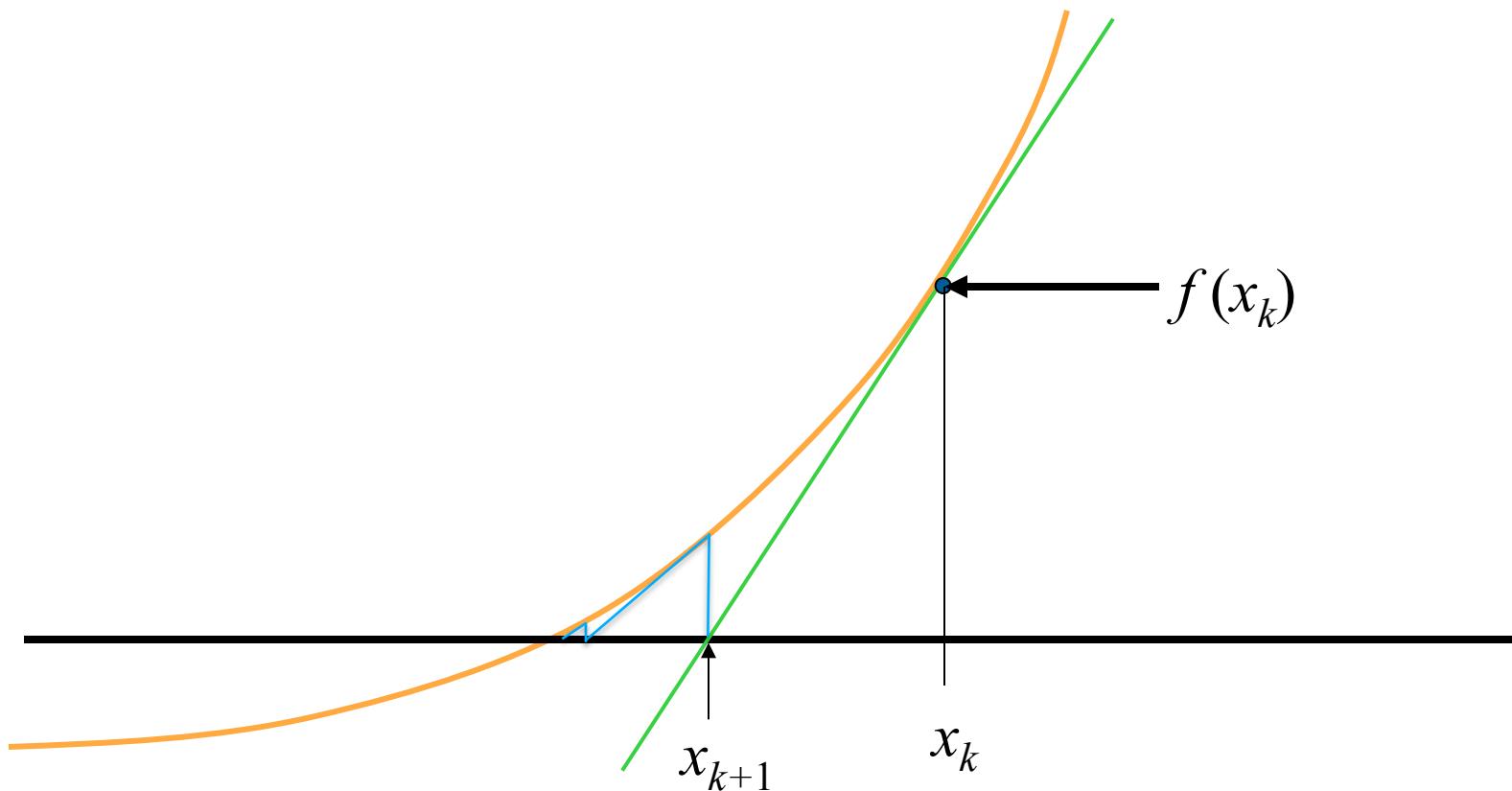
$$f(x) = f(x_0) + f'(x_0)(x - x_0)$$





Look at Taylor Series

$$0 = f(x_k) + f'(x_k)(x_{k+1} - x_k)$$





A Little Geometry and Algebra

$$\frac{(f(x_k) - 0)}{(x_k - x_{k+1})} = f'(x_k)$$

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Newton's Method



An Example of Using an Iterative Method

$$f(x) = x^n - C$$

$$f(x) = x^2 - 2$$

$$0 = x^2 - 2$$

$$x_{k+1} = x_k - \left(\frac{x_k^2 - 2}{2x_k} \right)$$

$$x^2 = 2$$

$$x = \pm\sqrt{2}$$

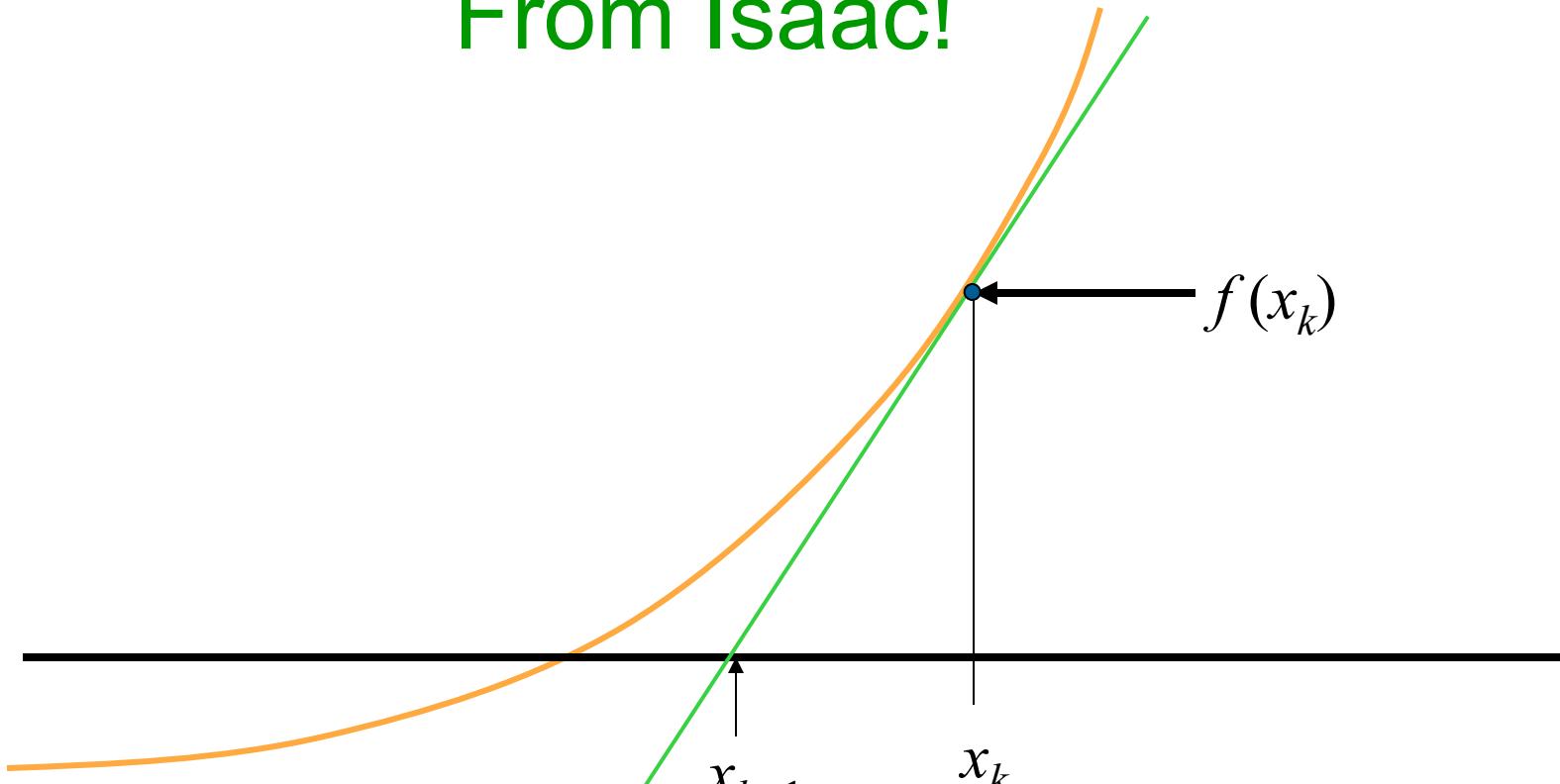


A Numerical Example

			100	50.01
5	2.7		50.01	25.024996
2.7	1.72037037		25.024996	12.55245805
1.72037037	1.441455368		12.55245805	6.355894695
1.441455368	1.414470981		6.355894695	3.335281609
1.414470981	1.414213586		3.335281609	1.967465562
1.414213586	1.414213562		1.967465562	1.49200089
1.414213562	1.414213562		1.49200089	1.416241332
1.414213562	1.414213562		1.416241332	1.414215014
			1.414215014	1.414213562
			1.414213562	1.414213562
			1.414213562	1.414213562



Let Takes Some Inspiration From Isaac!



$$x_{k+1} = x_k - \eta f'(x_k)$$

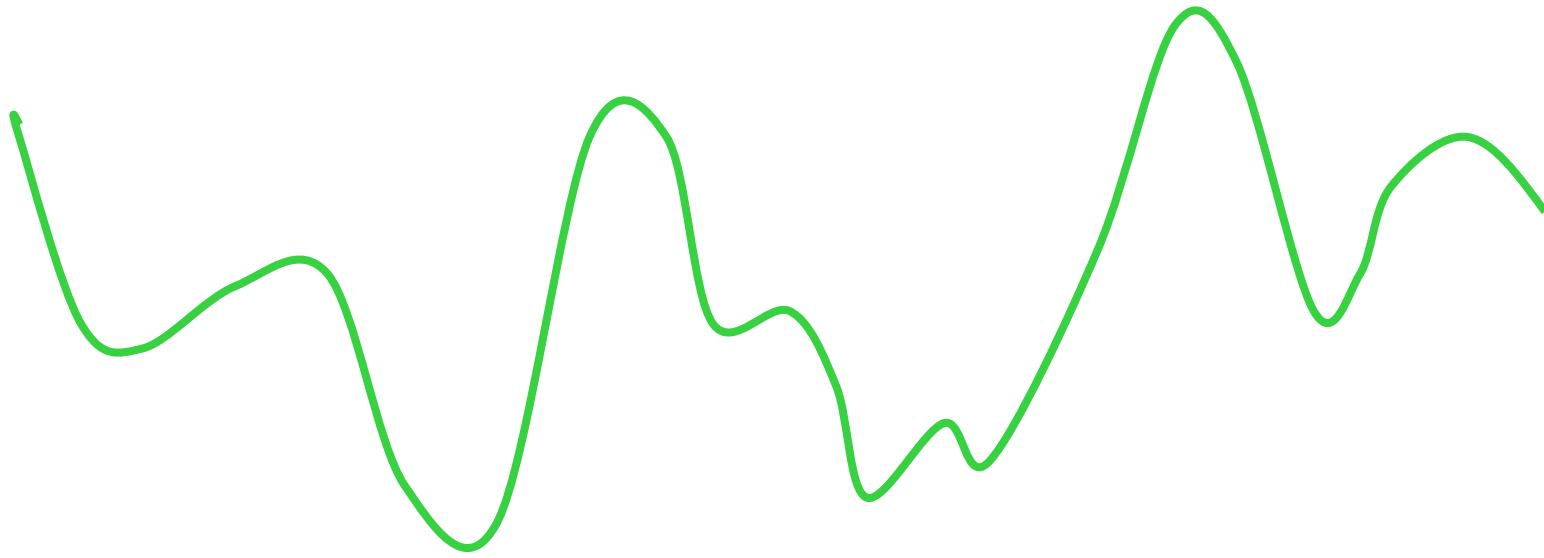


Finding a Minimum

- In Neural Networks, we want to set the weights so the Activity has the correct slope/orientation so we can take advantage of linear separability.
- We can define some ‘function’ that is affected by the weights.
- Such a function can relate what we want the output to be with what the output is ... an **error function**.
- But assuming such a function is possible, we cannot simply set derivatives to zero.
- Why?



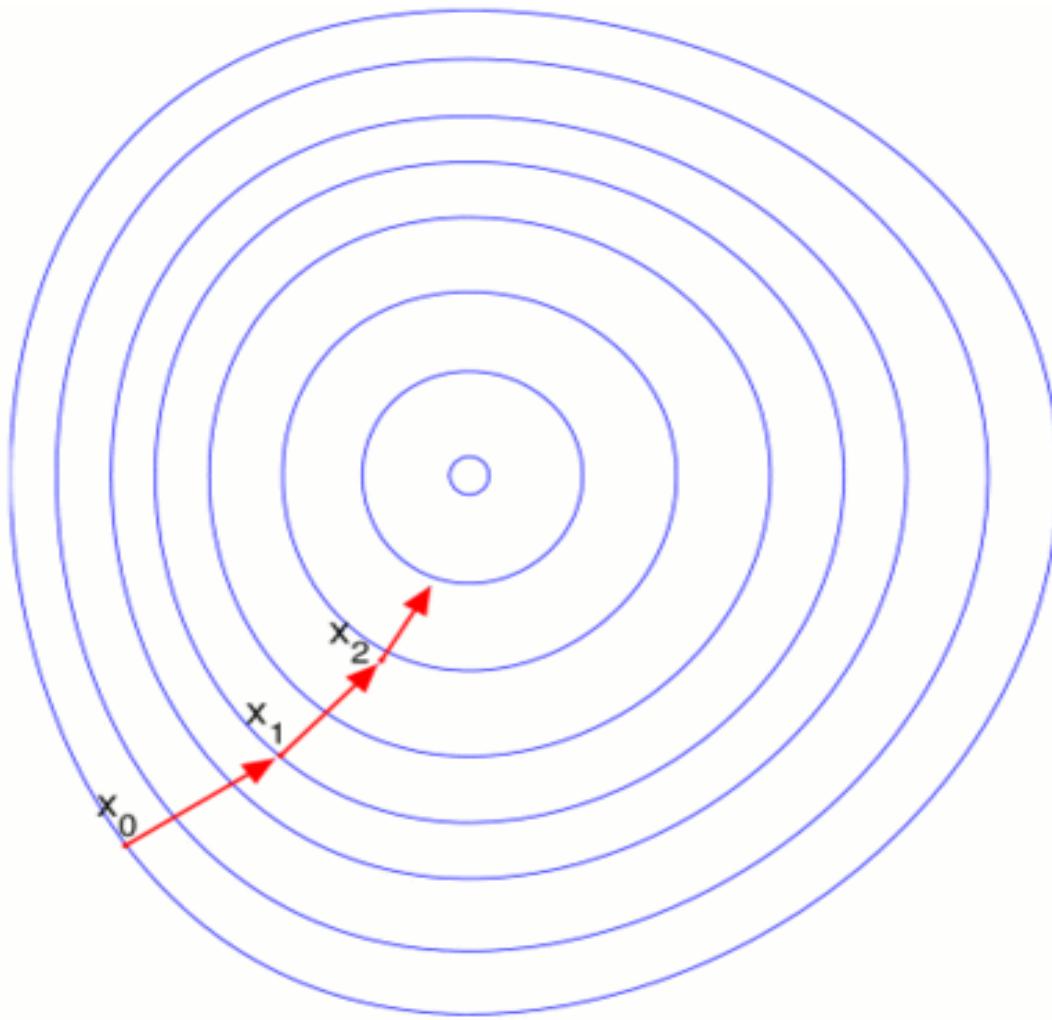
Minima, where for art thou?



The number of minima is large.

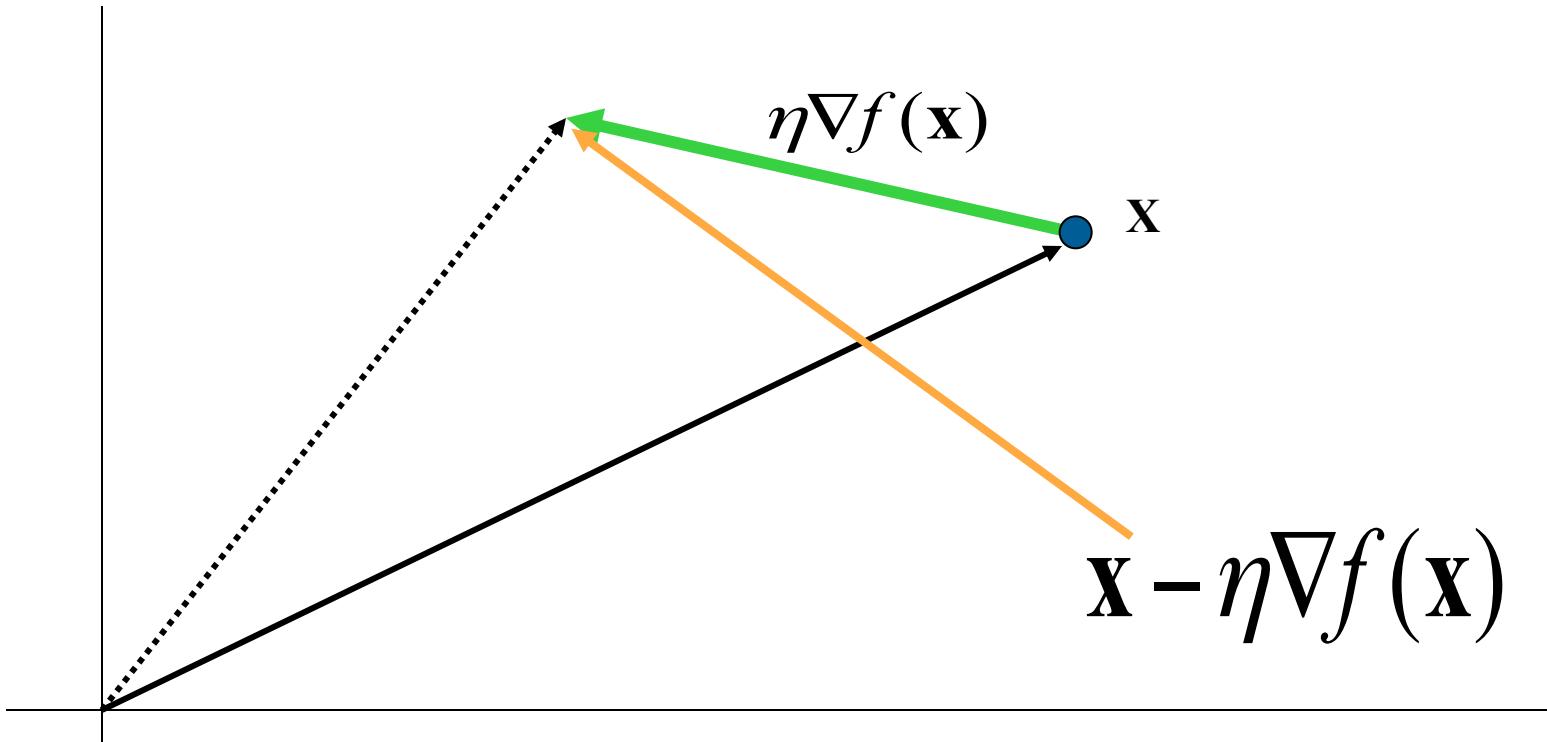
The equations for this landscape is quite complicated.

Finding the arguments that correspond to the derivatives equaling zero is not realistic.





Can Only Go From Where We Are





Method of Steepest Descent

Single Variable Case:

$$x_{k+1} = x_k - \eta f'(x_k)$$

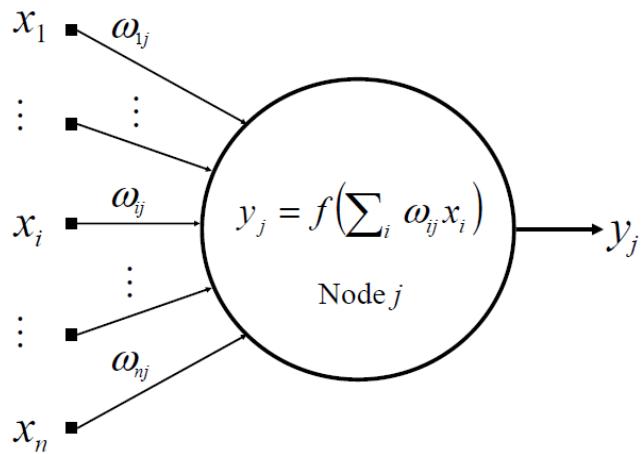
Multi-Variate Case:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta \nabla f(\mathbf{x}_k)$$





Getting Back to The Perceptron



Every node computes an output signal where $v_j = \sum_{i=1}^n x_i \omega_{ij} - \theta$ and is referred to as the *local field* impinging on node j . The output of node j is

$$y_j = f(v_j)$$

where

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



Want to do something useful!

- Let's assume we can train the perceptron to produce a desired output value.
- Define:

$$e_j = d_j - y_j$$



Let's do something even better!

- Define:

$$E_j = \frac{1}{2} (d_j - y_j)^2$$

If we have m output nodes, define:

$$E = \frac{1}{2} \sum_{j=1}^m e_j^2 = \frac{1}{2} \sum_{j=1}^m (d_j - y_j)^2$$



The Perceptron Delta Function

$$w_j(k+1) = w_j(k) - \eta \nabla E_j$$

$$\Delta \omega_{ij}(t+1) = \omega_{ij}(t+1) - \omega_{ij}(t) = \eta \frac{\partial E}{\partial \omega_{ij}(t)}$$

or in vector form

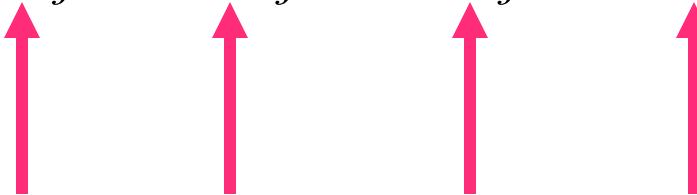
$$(\Delta \omega_{i1}, \Delta \omega_{i2}, \dots, \Delta \omega_{in}) = \eta \nabla(E)$$



The Perceptron Delta Function

Using the Chain Rule, we get:

$$\frac{\partial E_j}{\partial w_i} = \frac{\partial E_j}{\partial e_j} \times \frac{\partial e_j}{\partial y_j} \times \frac{\partial y_j}{\partial A_j} \times \frac{\partial A_j}{\partial w_i}$$



Factors:

1, 2, 3, 4



Factor 1

Recall that

$$E_j = \frac{1}{2} e_j^2, \text{ so } \frac{\partial E_j}{\partial e_j} = e_j$$



Factor 2

Recall that $e_j = d_j - y_j$

$$\text{so } \frac{\partial e_j}{\partial y_j} = -1$$



Factor 3

Recall that $y_j = \frac{1}{1 + e^{-A_j}}$ So,

$$\begin{aligned}
 \frac{\partial y_j}{\partial A_j} &= \frac{\partial}{\partial A_j} \left(\frac{1}{1 + e^{-A_j}} \right) \\
 &= \frac{f'g - fg'}{g^2} = \frac{-e^{-A_j}(-1)}{(1 + e^{-A_j})^2} \\
 &= \frac{e^{-A_j}}{(1 + e^{-A_j})} \times \left(\frac{1}{1 + e^{-A_j}} \right) \\
 &= [1 - y_j]y_j
 \end{aligned}$$



Factor 4

Recall that

$$A_j = \sum_{i=1}^n w_i x_i$$

So,

$$\begin{aligned}\frac{\partial A_j}{\partial w_k} &= \frac{\partial}{\partial w_k} \sum_{i=1}^n w_i x_i \\ &= \sum_{i=1}^n \frac{\partial}{\partial w_k} (w_i x_i) \\ &= x_k \text{ for } i = k, 0 \text{ otherwise}\end{aligned}$$



The Perceptron Delta Function

Putting it all together,

$$\frac{\partial E_j}{\partial w_i} = \frac{\partial E_j}{\partial e_j} \times \frac{\partial e_j}{\partial y_j} \times \frac{\partial y_j}{\partial A_j} \times \frac{\partial A_j}{\partial w_i}$$

$$\frac{\partial E}{\partial \omega_{ij}} = -e_j[1 - y_j]y_jx_i$$



The Perceptron Delta Function

Putting it all together,

$$\frac{\partial E}{\partial \omega_{ij}} = -e_j[1 - y_j]y_jx_i$$

and letting $\delta_j = -e_j[1 - y_j]y_j$ then

$$\Delta\omega_{ij} = \eta \frac{\partial E}{\partial \omega_{ij}} = \eta \delta_j x_i.$$



Introduction to Neural Networks

**Johns Hopkins University
Engineering for Professionals Program**

605-447/625-438

Dr. Mark Fleischer

Copyright 2013 by Mark Fleischer

Module 4.2: MOSD Example



This Sub-Module Covers ...

- A closer look at the Method of Steepest Descent.
- Provides an example of how it performs using an Excel spreadsheet.
- Illustrates various performance issues.



A Quick Rehash

$$f(x) = f(x_0) + f'(x_0)(x - x_0)$$

$$f(x_{k+1}) = f(x_k) + f'(x_k)(x_{k+1} - x_k)$$

$$0 = f(x_k) + f'(x_k)(x_{k+1} - x_k)$$

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

$$x_{k+1} = x_k - \frac{f(x_k)}{\left[f'(x_k)\right]^2} f'(x_k)$$

$$x_{k+1} = x_k - \eta f'(x_k)$$

$$\underline{\mathbf{w}_{k+1} = \mathbf{w}_k - \eta \nabla E} \quad \Rightarrow \quad \underline{w_{ij}(k+1) = w_{ij}(k) + \eta e_j [1 - y_j] y_j x_i}$$

First 2 terms of Taylor Series

Substitute x with x_{k+1} and x_0 with x_k

Setting $f(x_{k+1}) = 0$

Rearranging and solving for x_{k+1}

Equivalently

Method of Steepest Descent

The Perceptron Delta Method



The Perceptron Delta Function

Putting it all together,

$$\frac{\partial E}{\partial \omega_{ij}} = -e_j[1 - y_j]y_jx_i$$

and letting $\delta_j = -e_j[1 - y_j]y_j$ then

$$\Delta\omega_{ij} = \eta \frac{\partial E}{\partial \omega_{ij}} = \eta \delta_j x_i.$$



Let's Use the PDF in Excel

0.71095	
0.71095	
0.8	0.8
0.8	0.8
1.137519	
0.757224	
0.142776	



Introduction to Neural Networks

Johns Hopkins University
Engineering for Professionals Program
605-447/625-438

Dr. Mark Fleischer

Copyright 2013 by Mark Fleischer

Module 5.1: The Feed-forward, Back Propagation Algorithm



This Sub-Module Covers ...

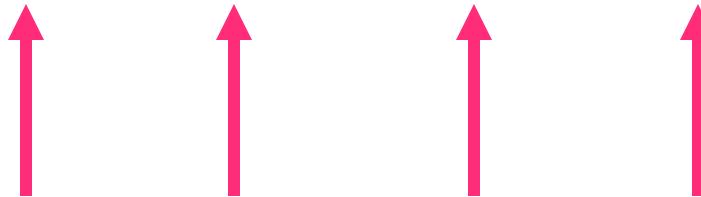
- Extends the Perceptron Delta Function to handle multi-layer networks.
- We will derive the feed-forward, back-propagation algorithm.
 - Similar in spirit to the Perceptron Delta Function.
 - Calculus based optimization technique.
- Next video we go through a computational example.



The Perceptron Delta Function

Using the Chain Rule, we get:

$$\frac{\partial E_j}{\partial w_{ij}} = \frac{\partial E_j}{\partial e_j} \times \frac{\partial e_j}{\partial y_j} \times \frac{\partial y_j}{\partial A_j} \times \frac{\partial A_j}{\partial w_{ij}}$$



Factors:

1,

2,

3,

4



The Perceptron Delta Function

$$\frac{\partial E_j}{\partial w_{ij}} = \frac{\partial E_j}{\partial e_j} \times \frac{\partial e_j}{\partial y_j} \times \frac{\partial y_j}{\partial A_j} \times \frac{\partial A_j}{\partial w_{ij}}$$

$$\frac{\partial E}{\partial \omega_{ij}} = -e_j[1 - y_j]y_jx_i$$

From input i to output j .



The Perceptron Delta Function

$$\frac{\partial E}{\partial \omega_{ij}} = -e_j[1 - y_j]y_jx_i$$

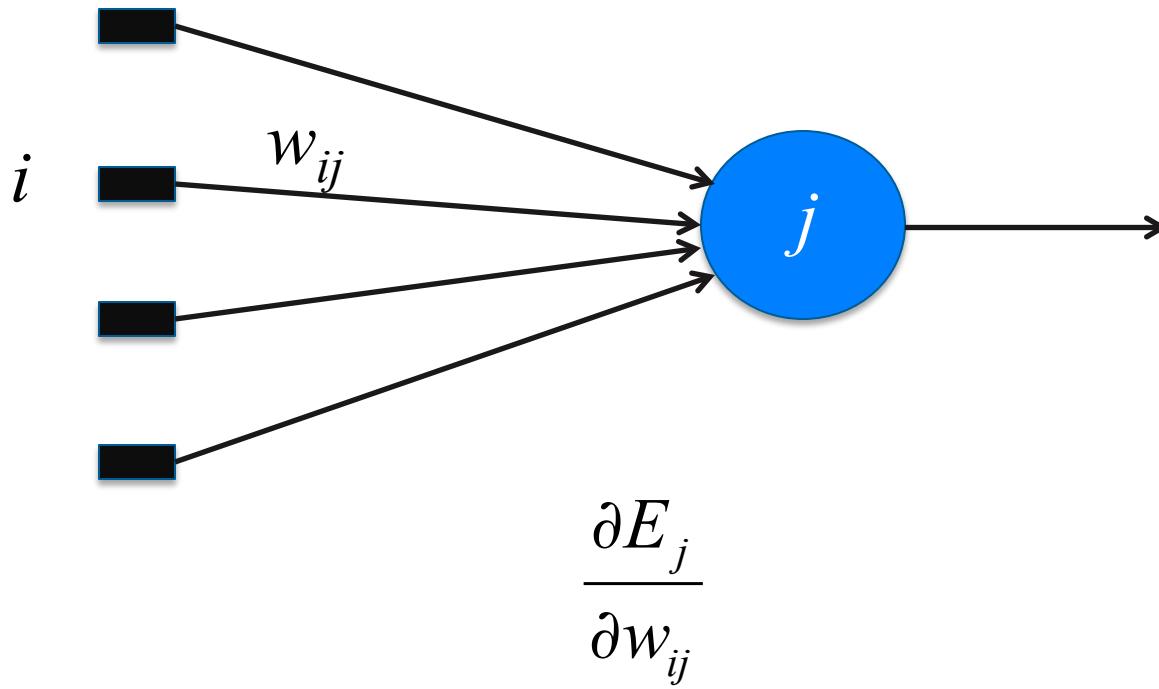
and letting $\delta_j = -e_j[1 - y_j]y_j$ then

$$\Delta\omega_{ij} = \eta \frac{\partial E}{\partial \omega_{ij}} = \eta \delta_j x_i.$$

From input i to output j .

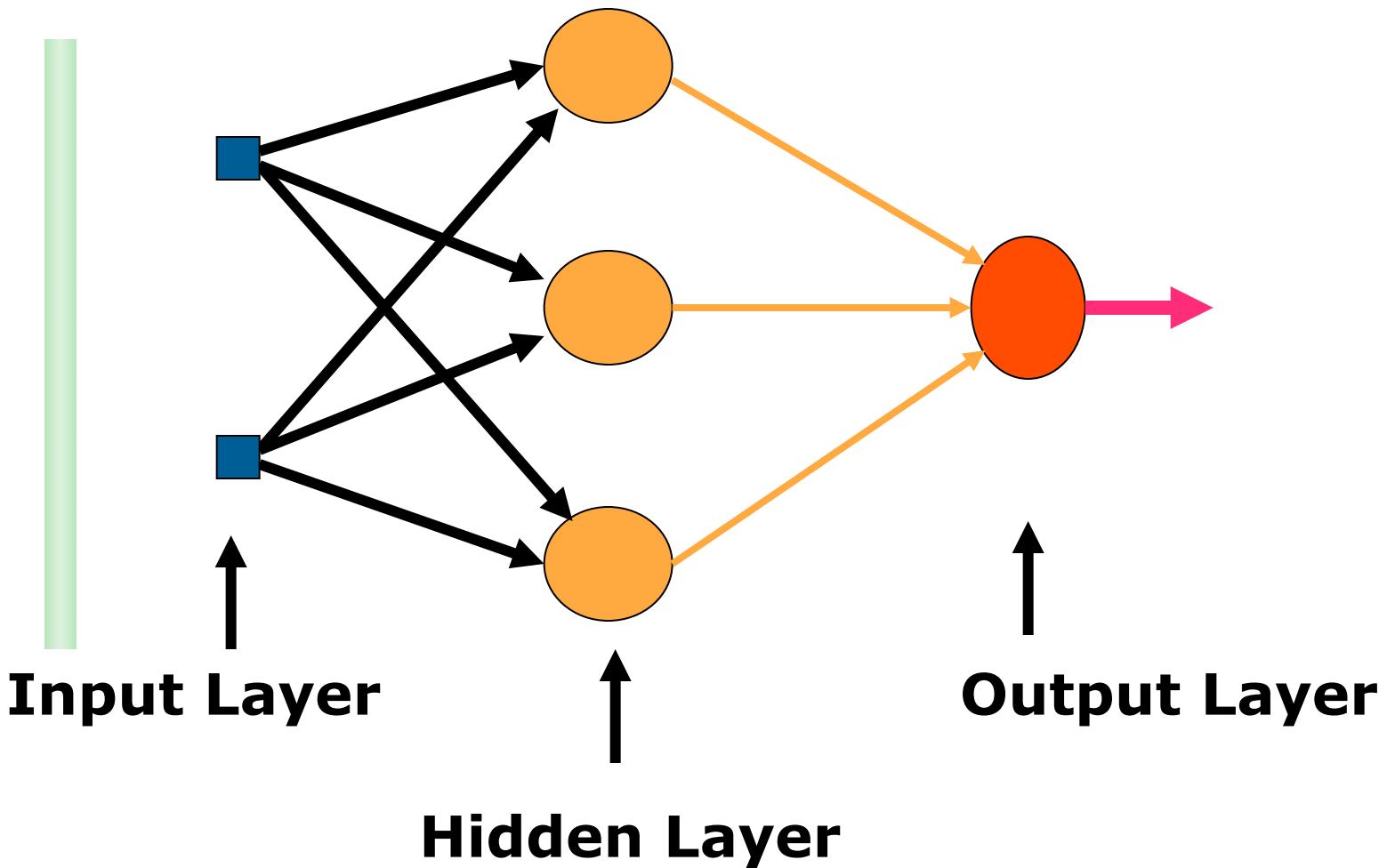


The Perceptron





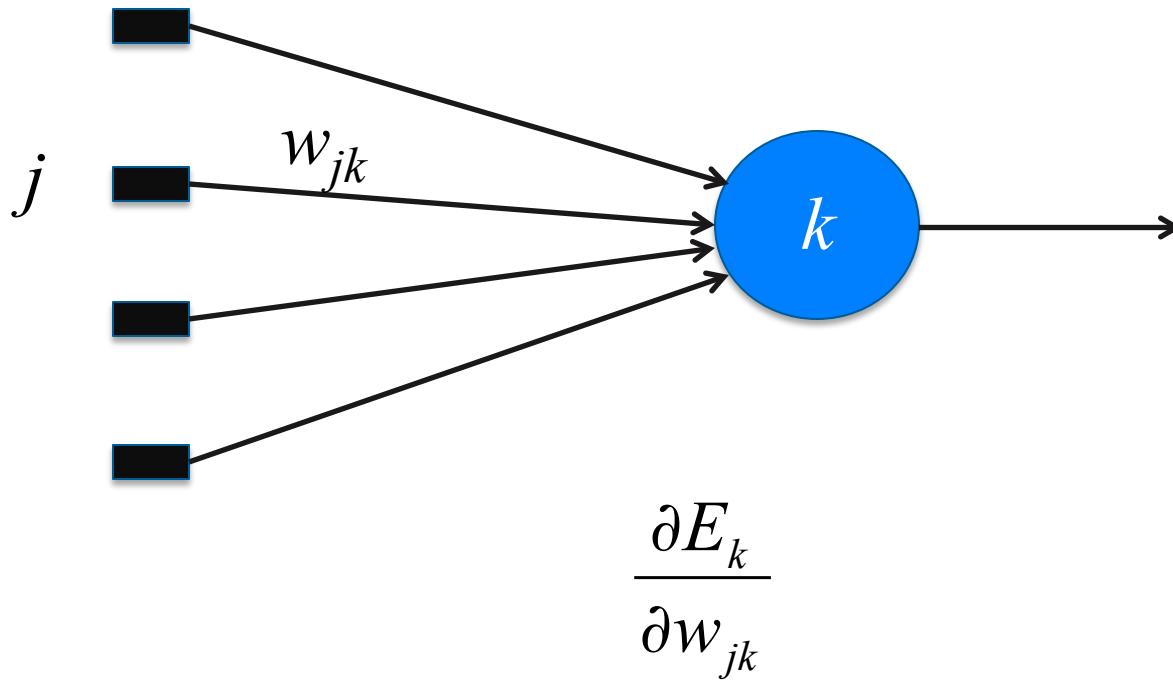
A Multi-layered Network





The Perceptron

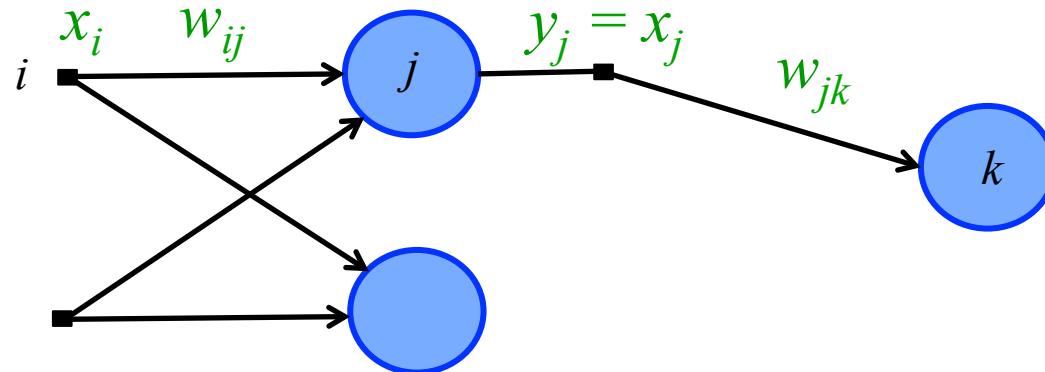
New naming conventions for the output nodes.





The Feed-forward Back-propagation Algorithm

Notational Conventions for a multi-layered network





The Gradient Vector

was just

$$\left(\frac{\partial E_k}{\partial w_{1k}}, \frac{\partial E_k}{\partial w_{2k}}, \dots, \frac{\partial E_k}{\partial w_{nk}} \right)$$

but in a multi-layer network, it becomes

$$\left(\underbrace{\frac{\partial E_k}{\partial w_{1k}}, \frac{\partial E_k}{\partial w_{2k}}, \dots, \frac{\partial E_k}{\partial w_{nk}}}_{\text{Output Layer Node}}, \underbrace{\frac{\partial E_k}{\partial w_{1j_1}}, \frac{\partial E_k}{\partial w_{2j_1}}, \dots, \frac{\partial E_k}{\partial w_{mj_1}}}_{\text{Hidden Layer Node 1}}, \underbrace{\frac{\partial E_k}{\partial w_{1j_2}}, \frac{\partial E_k}{\partial w_{2j_2}}, \dots, \frac{\partial E_k}{\partial w_{mj_2}}}_{\text{Hidden Layer Node 2}}, \dots \right)$$

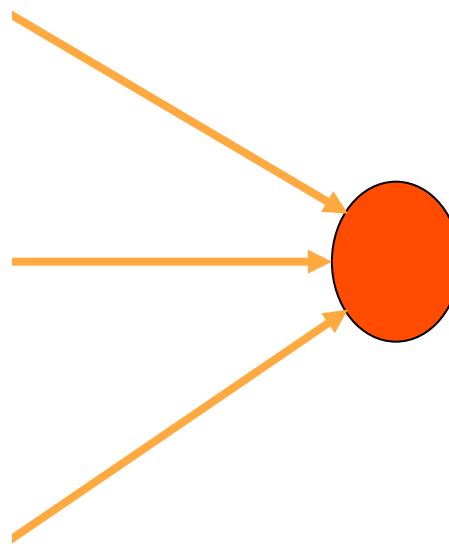
Output Layer Node

Hidden Layer Node 1

Hidden Layer Node 2



A Multi-layered Network





The Feed-forward Back-Propagation Algorithm

From this

$$\frac{\partial E_k}{\partial w_{jk}} = \frac{\partial E_k}{\partial e_k} \times \frac{\partial e_k}{\partial y_k} \times \frac{\partial y_k}{\partial A_k} \times \frac{\partial A_k}{\partial w_{jk}}$$

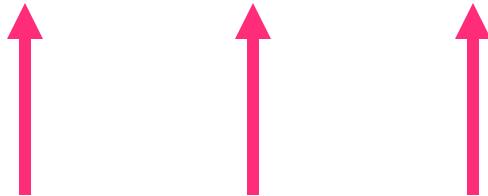
to this

$$\frac{\partial E_k}{\partial w_{ij}} = \frac{\partial E_k}{\partial x_j} \times \frac{\partial x_j}{\partial A_j} \times \frac{\partial A_j}{\partial w_{ij}}$$



The Feed-forward Back-Propagation Algorithm

$$\frac{\partial E_k}{\partial w_{ij}} = \frac{\partial E_k}{\partial x_j} \times \frac{\partial x_j}{\partial A_j} \times \frac{\partial A_j}{\partial w_{ij}}$$



Factors: 1, 2, 3



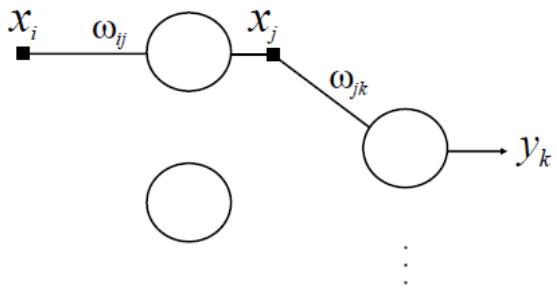
The Feed-forward Back-Propagation Algorithm

Factor 1:

$$\begin{aligned}\frac{\partial E_k}{\partial x_j} &= \frac{\partial}{\partial x_j} \frac{1}{2} \sum_k e_k^2 \\ &= \frac{1}{2} \sum_k \frac{\partial}{\partial x_j} e_k^2 \\ &= \sum_k e_k \frac{\partial e_k}{\partial x_j} \\ &= \sum_k e_k \frac{\partial e_k}{\partial A_k} \cdot \frac{\partial A_k}{\partial x_j}\end{aligned}$$



The Feed-forward Back-propagation Algorithm



$$\begin{aligned}
 e_k &= d_k - y_k \\
 &= d_k - f_k(A_k) \\
 \therefore \frac{\partial e_k}{\partial A_k} &= -f'_k(A_k)
 \end{aligned}$$

$$\sum_k e_k \frac{\partial e_k}{\partial A_k} \cdot \frac{\partial A_k}{\partial x_j}$$

$$\text{Since } A_k = \sum_{j=1}^M x_j w_{jk}$$

$$\begin{aligned}
 \text{then } \frac{\partial A_k}{\partial x_j} &= \frac{\partial \sum_{j=1}^M x_j w_{jk}}{\partial x_j} \\
 &= w_{jk}
 \end{aligned}$$



The Feed-forward Back-propagation Algorithm

Factor 1:

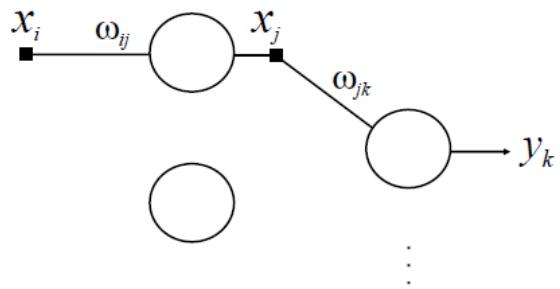
$$\begin{aligned}
 \frac{\partial E_k}{\partial x_j} &= \frac{\partial}{\partial x_j} \frac{1}{2} \sum_k e_k^2 \\
 &= \frac{1}{2} \sum_k \frac{\partial}{\partial x_j} e_k^2 \\
 &= \sum_k e_k \frac{\partial e_k}{\partial x_j} \\
 &= \sum_k e_k \frac{\partial e_k}{\partial A_k} \cdot \frac{\partial A_k}{\partial x_j}
 \end{aligned}$$

$$\begin{aligned}
 \frac{\partial E_k}{\partial x_j} &= - \sum_k e_k f'_k(A_k) w_{jk} \\
 &= \sum_k \delta_k w_{jk}
 \end{aligned}$$



FFBP: Factors 2 and 3

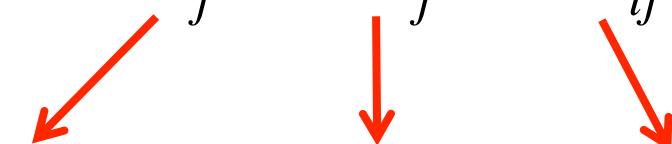
$$\frac{\partial x_j}{\partial A_j} \times \frac{\partial A_j}{\partial w_{ij}} = [1 - x_j]x_jx_i$$



$$A_j = \sum_i w_{ij}x_i$$



FFBP --- All Together Now

$$\frac{\partial E_k}{\partial w_{ij}} = \frac{\partial E_k}{\partial x_j} \times \frac{\partial x_j}{\partial A_j} \times \frac{\partial A_j}{\partial w_{ij}}$$

$$\sum_k \delta_k w_{jk} \quad \times \quad [1 - x_j] x_j \quad \times \quad x_i$$



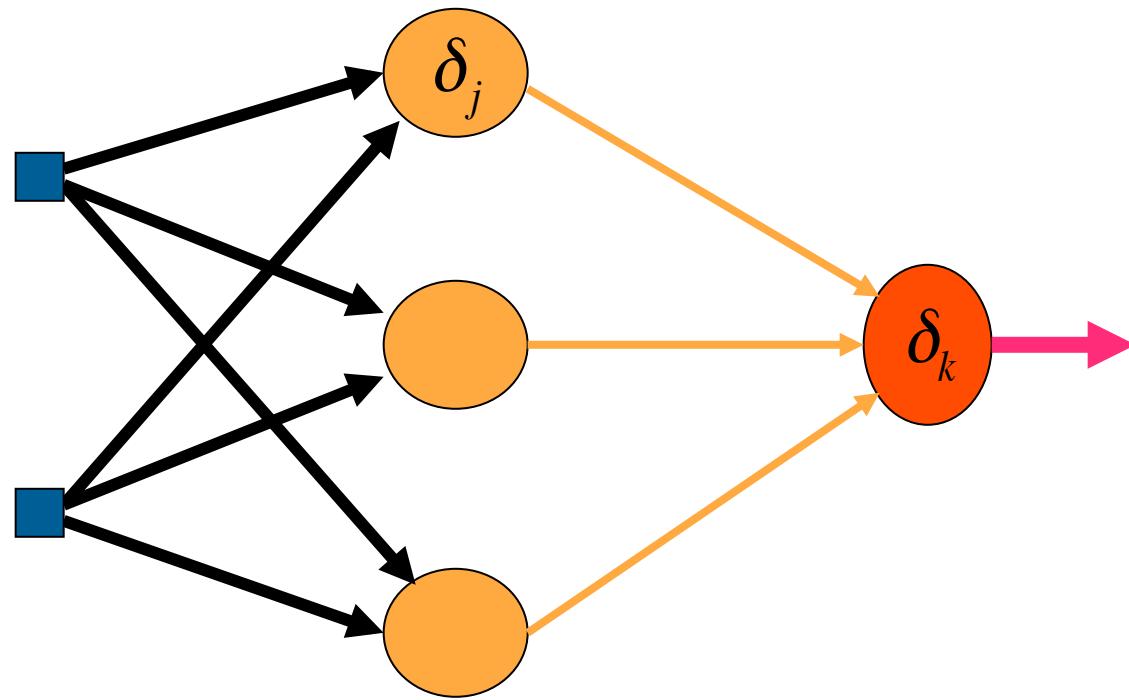
FFBP --- All Together Now

$$\frac{\partial E}{\partial w_{ij}} = [1 - x_j]x_j \left(\sum_k \delta_k w_{jk} \right) x_i = \delta_j x_i$$

$$\Delta w_{ij} = \eta \delta_j x_i$$



A Multi-Layered Network





Introduction to Neural Networks

**Johns Hopkins University
Engineering for Professionals Program**

605-447/625-438

Dr. Mark Fleischer

Copyright 2013 by Mark Fleischer

Module 5.2: An Example of the FFBP

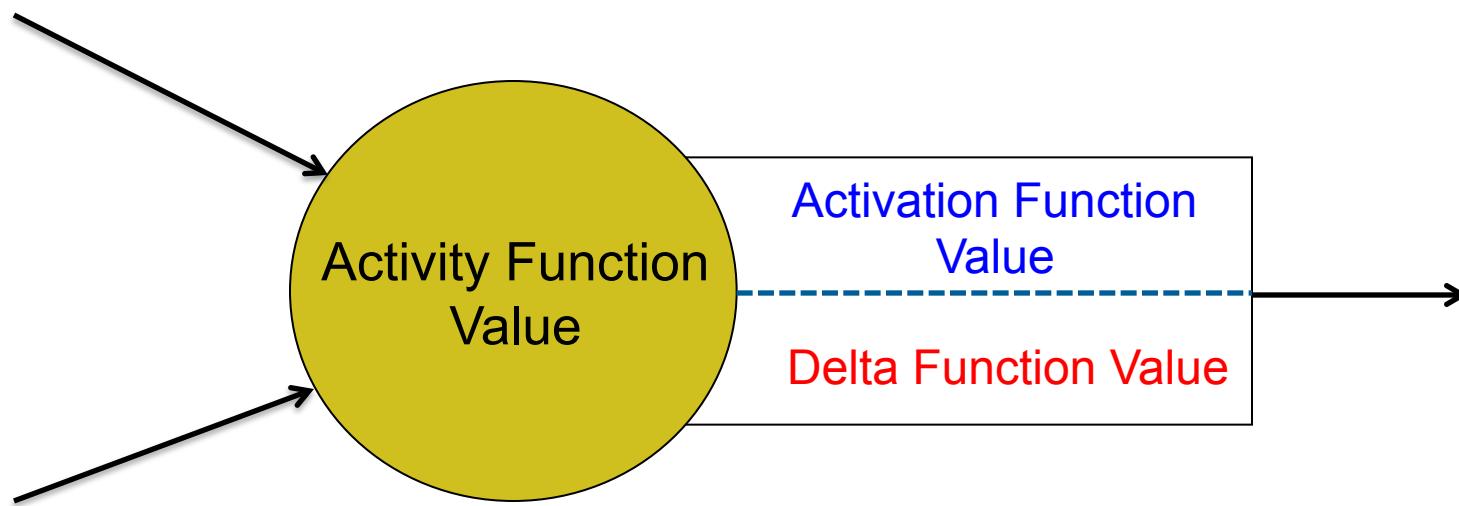


This Sub-Module Covers ...

- An example of applying the FFBP algorithm to a simple, multi-layer network.
- It will demonstrate the structure and behavior of the FFBP.



Conventions for Our Perceptron





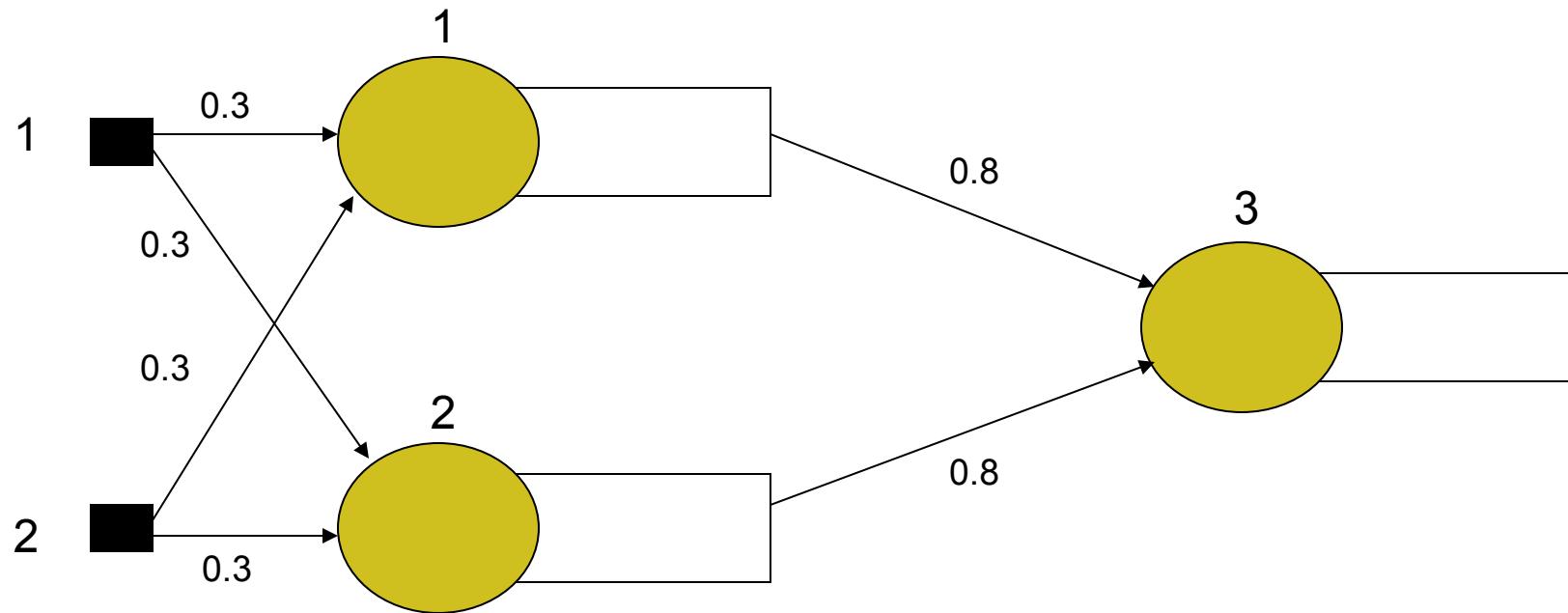
An FFBP Example

- A simple two input, single output, two layer network
- Inputs are 1 and 2: $I = \{1,2\}$
- Want to train the network so the inputs produce an output value of 0.7. Thus, $\{1,2\} \rightarrow \{0.7\}$
- Arbitrarily set initial weights for all links.
- For convenience, we'll set the biases all to 0.
- Set all the weights of the hidden layer to 0.3.
- Set all the weights in the output layer to 0.8.
- Set the value of $\eta = 1.0$ (that's convenient!).



Topology of Net

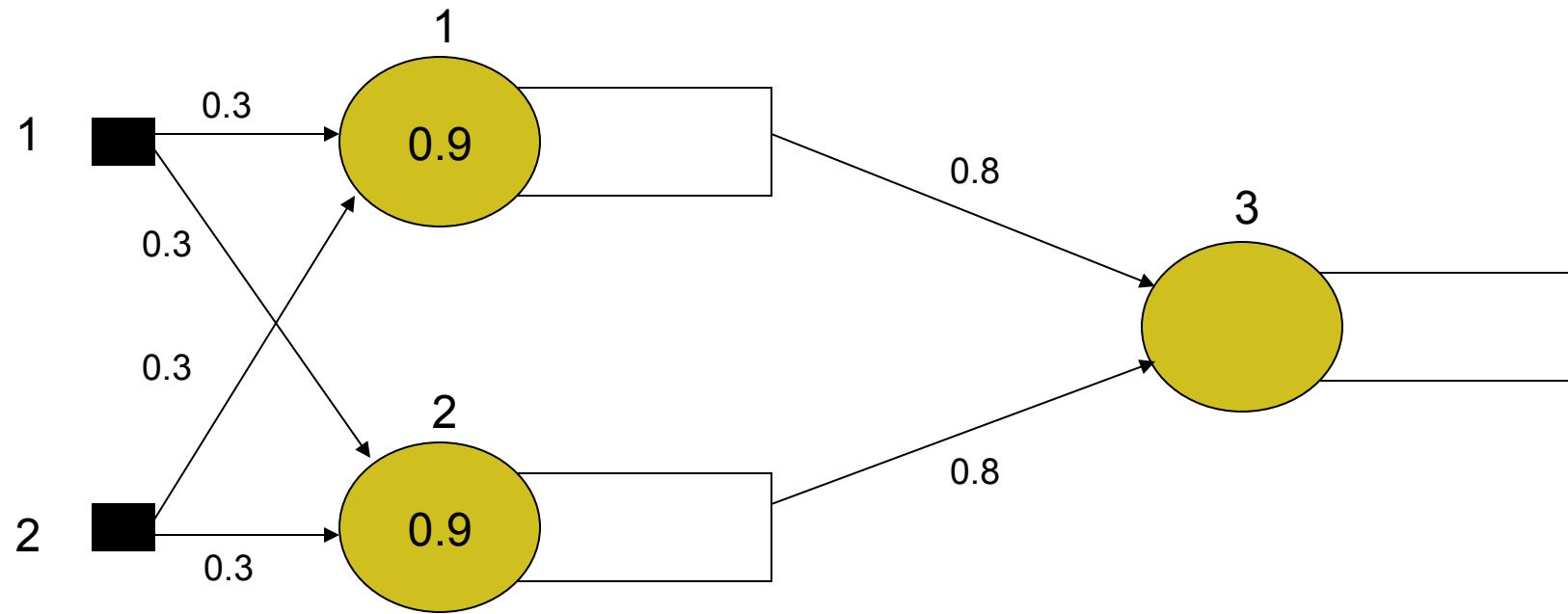
After Initialization of Weights





Topology of Net

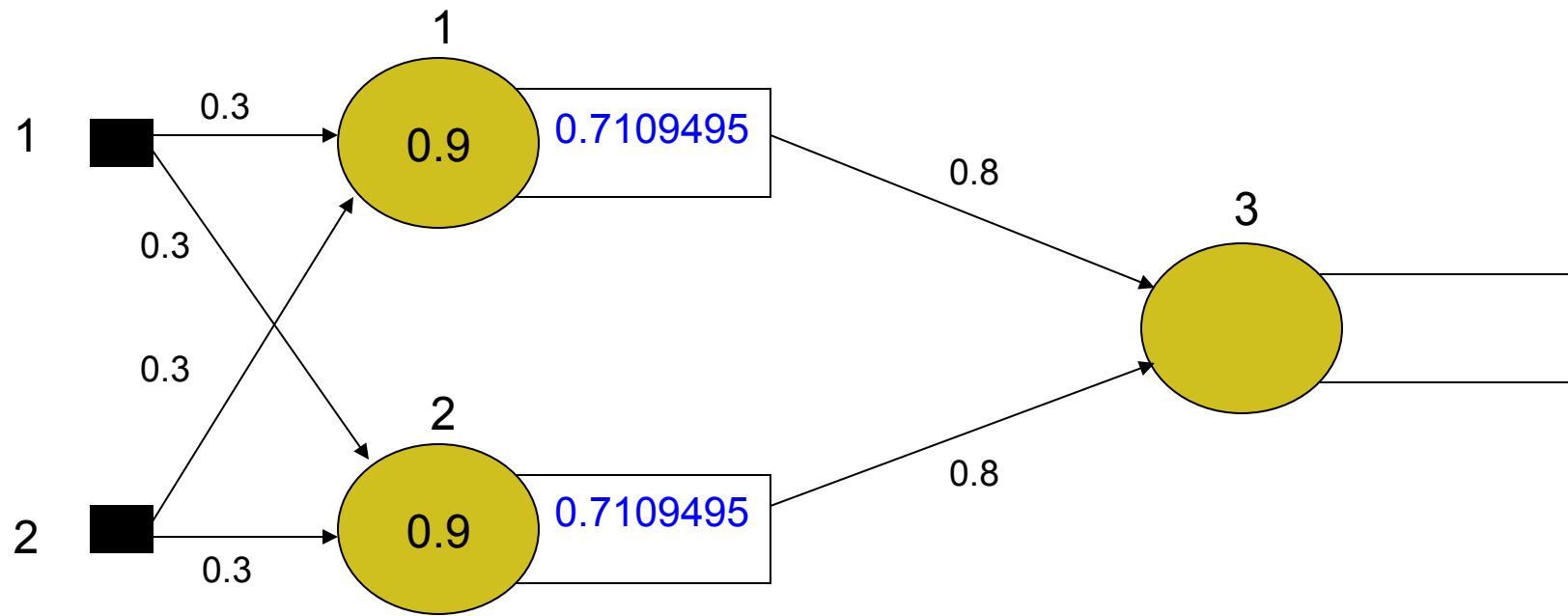
Feed-forward Epoch





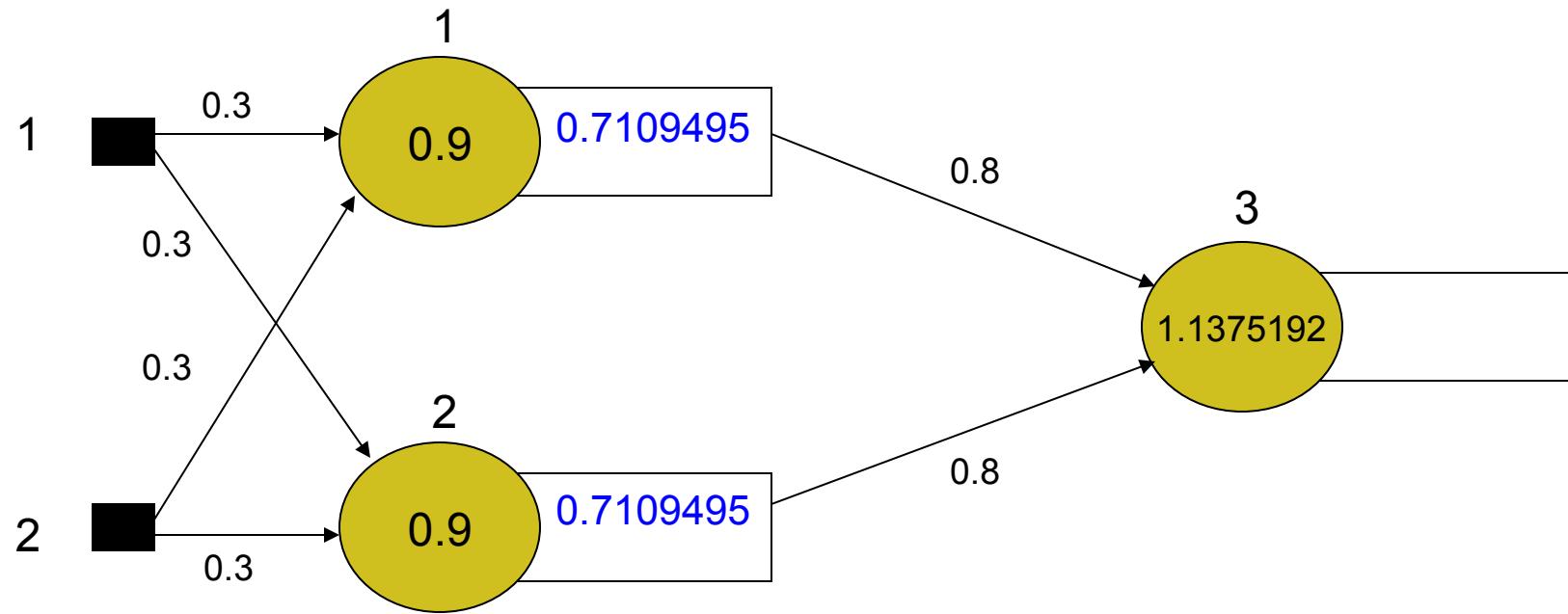
Topology of Net

Feed-forward Epoch



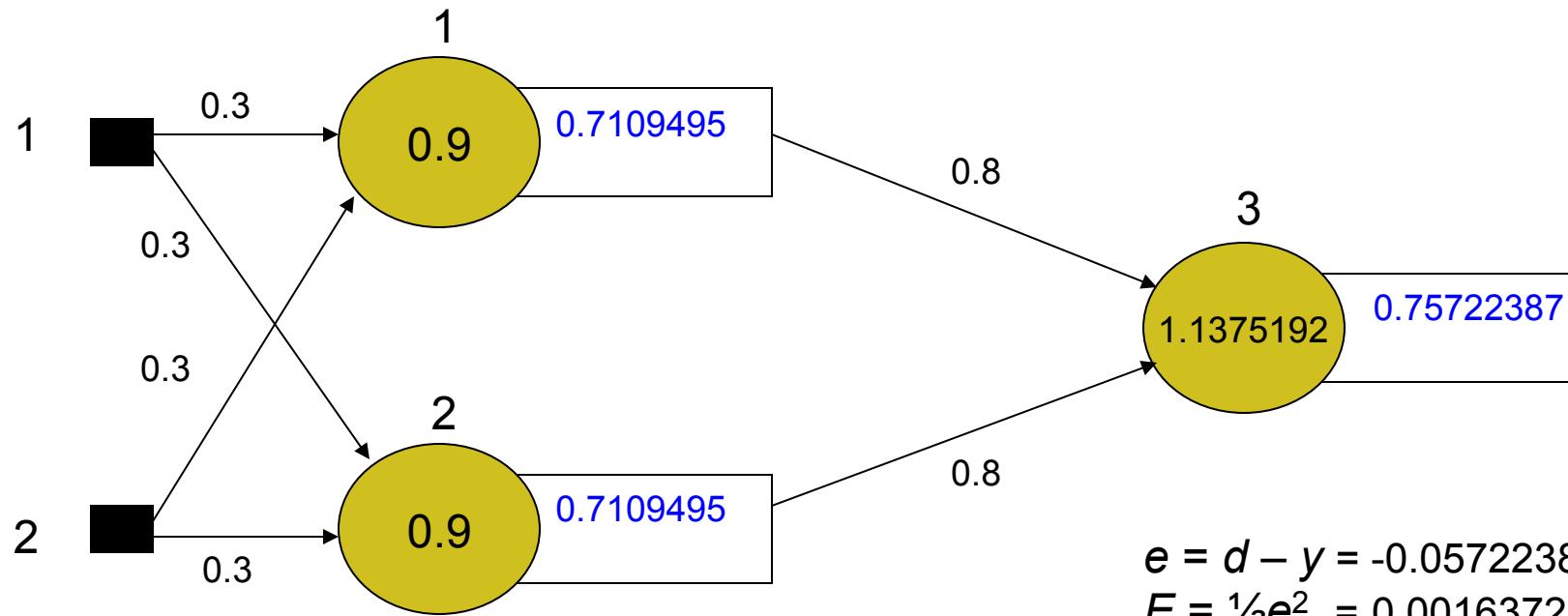


Topology of Net Feed-forward Epoch





Topology of Net Feed-forward Epoch





Calculate the Deltas for the Output Layer

$$w_{jk}(t+1) = w_{jk}(t) - \eta(-e_k[1 - y_k]y_k x_j)$$

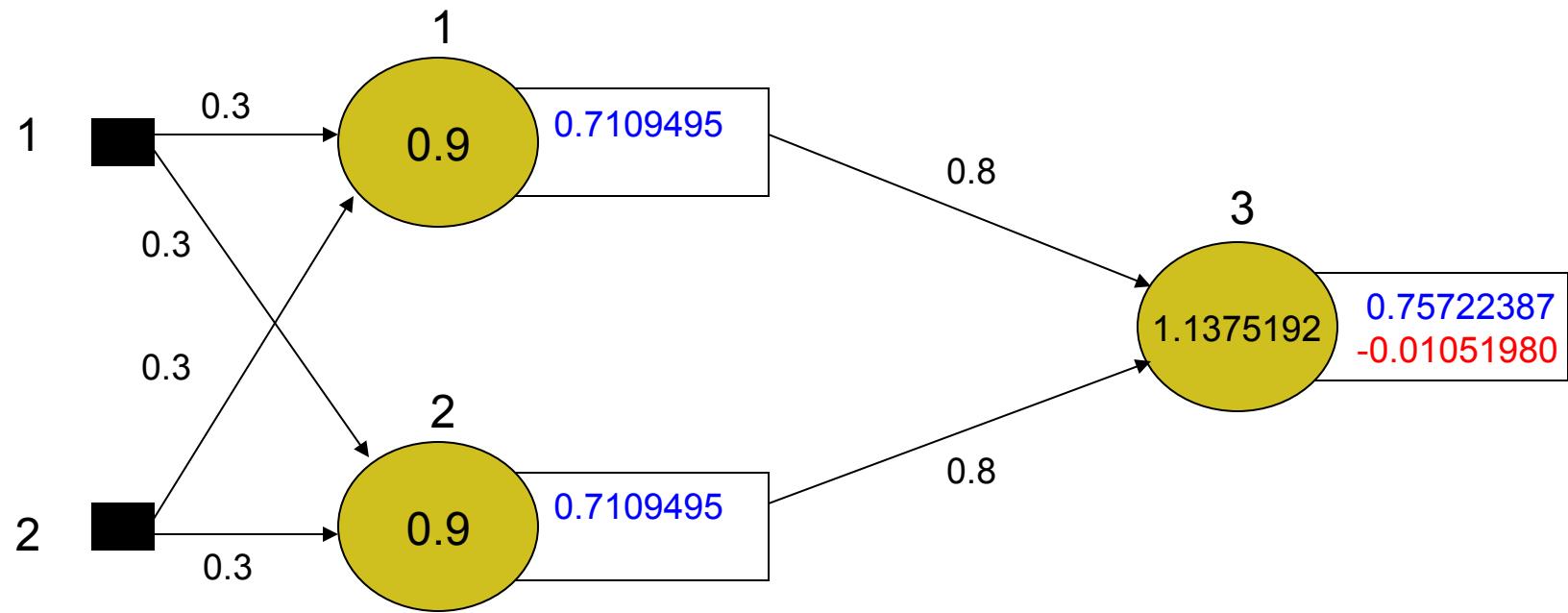
$$w_{jk}(t+1) = w_{jk}(t) + \eta(\delta_k x_j)$$

$$\delta_k = e_k [1 - y_k] y_k$$

$$\begin{aligned}\text{Delta (node 3)} &= (0.7 - 0.757224)(1 - 0.757224) 0.757224 \\ &= -0.0105198\end{aligned}$$



Back Propagation Epoch





Calculate the New Weights for the Output Layer

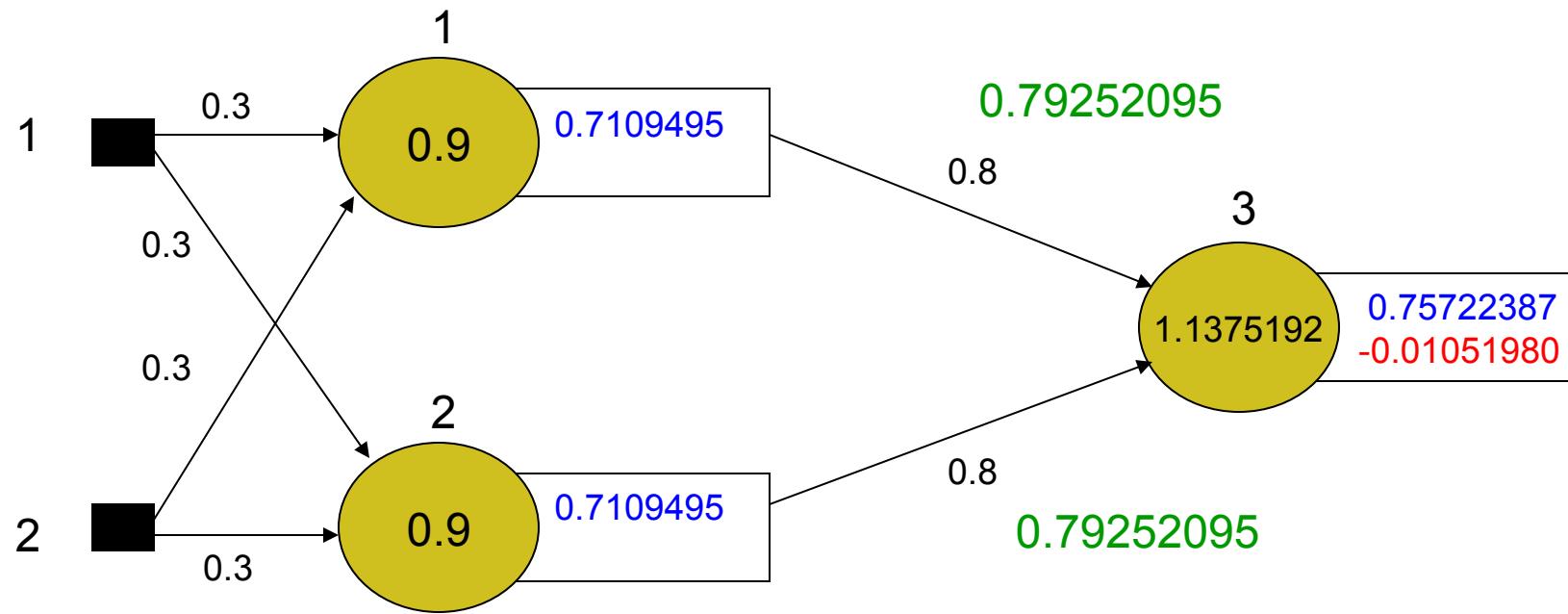
$$w_{jk}(t+1) = w_{jk}(t) - \eta(-e_k[1 - y_k]y_kx_j)$$

$$w_{jk}(t+1) = w_{jk}(t) + \eta(\delta_k x_j)$$

$$\begin{aligned} w_{jk}(t+1) &= (0.8) + 1 * (-0.0105198) * 0.7109495 \\ &= 0.79252095 \end{aligned}$$



Back Propagation Epoch





Calculate the Deltas for the Input Layer

Recall that for the input layer, the gradient vector element

$$\frac{\partial E}{\partial w_{ij}} = -[1 - x_j]x_j \left(\sum_k \delta_k w_{jk} \right) x_i = \delta_j x_i$$

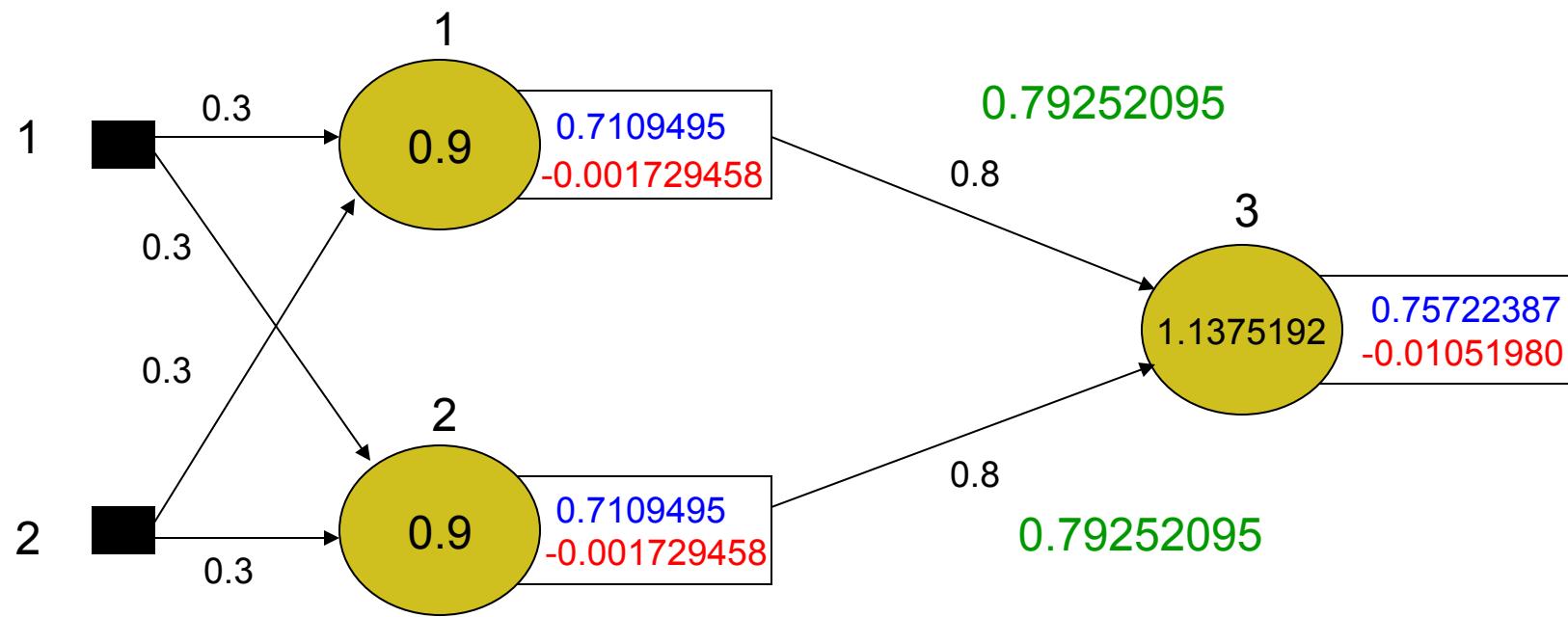
where

$$\delta_j = [1 - x_j]x_j \left(\sum_k \delta_k w_{jk} \right)$$

$$\begin{aligned}\delta_j &= (1 - 0.7109495) * 0.7109495 * (-0.01051980 * 0.8) \\ &= -0.001729458\end{aligned}$$



Back Propagation Epoch





Calculate the New Weights for the Hidden Layer

$$w_{ij}(t+1) = w_{ij}(t) - \eta \left(\frac{\partial E}{\partial w_{ij}} \right)$$

But now

$$\frac{\partial E}{\partial w_{ij}} = -[1 - x_j]x_j \left(\sum_k \delta_k w_{jk} \right) x_i = -\delta_j x_i$$

Thus, the updated weights for the input layer are $w_{ij}(t+1) = w_{ij}(t) + \eta(\delta_j x_i)$

$$w_{11}(t+1) = (0.3) + 1 * (-0.001729458) * 1 = 0.298270542$$

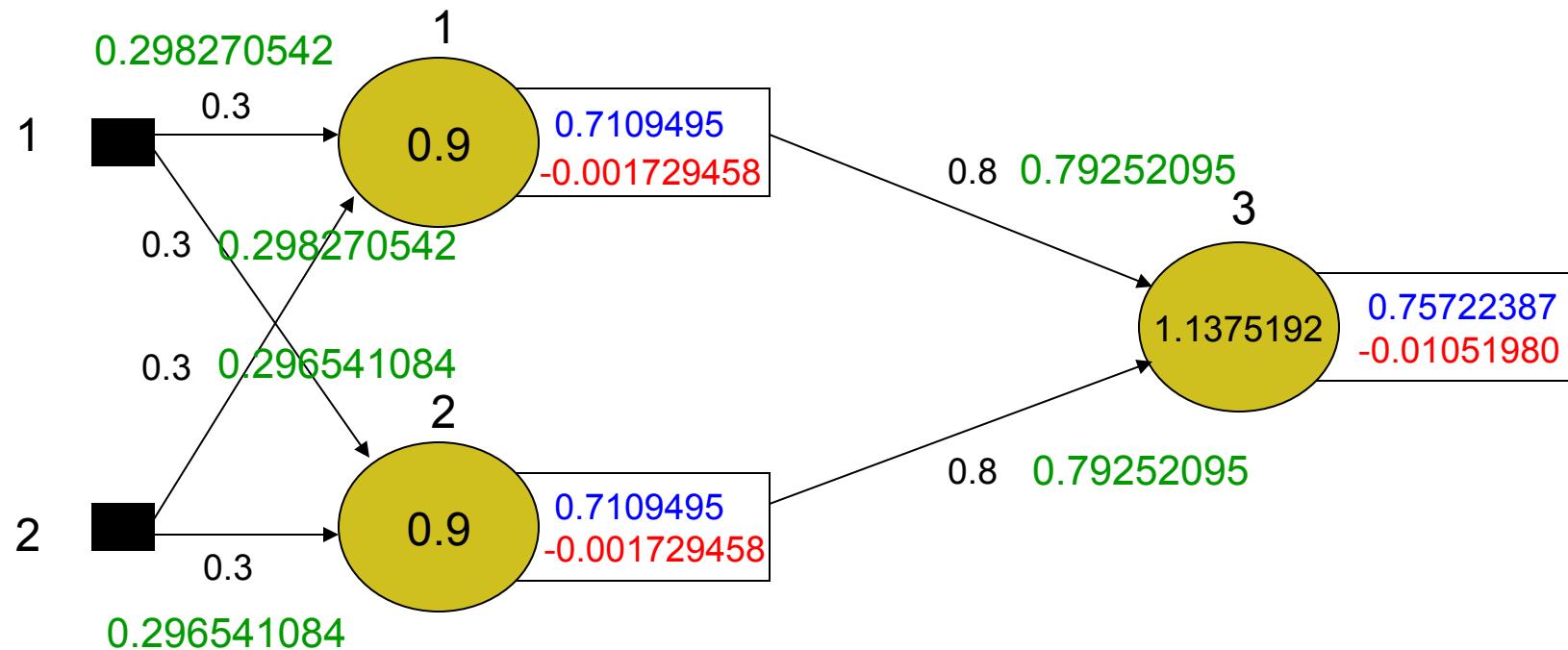
$$w_{12}(t+1) = (0.3) + 1 * (-0.001729458) * 1 = 0.298270542$$

$$w_{21}(t+1) = (0.3) + 1 * (-0.001729458) * 2 = 0.296541084$$

$$w_{22}(t+1) = (0.3) + 1 * (-0.001729458) * 2 = 0.296541084$$

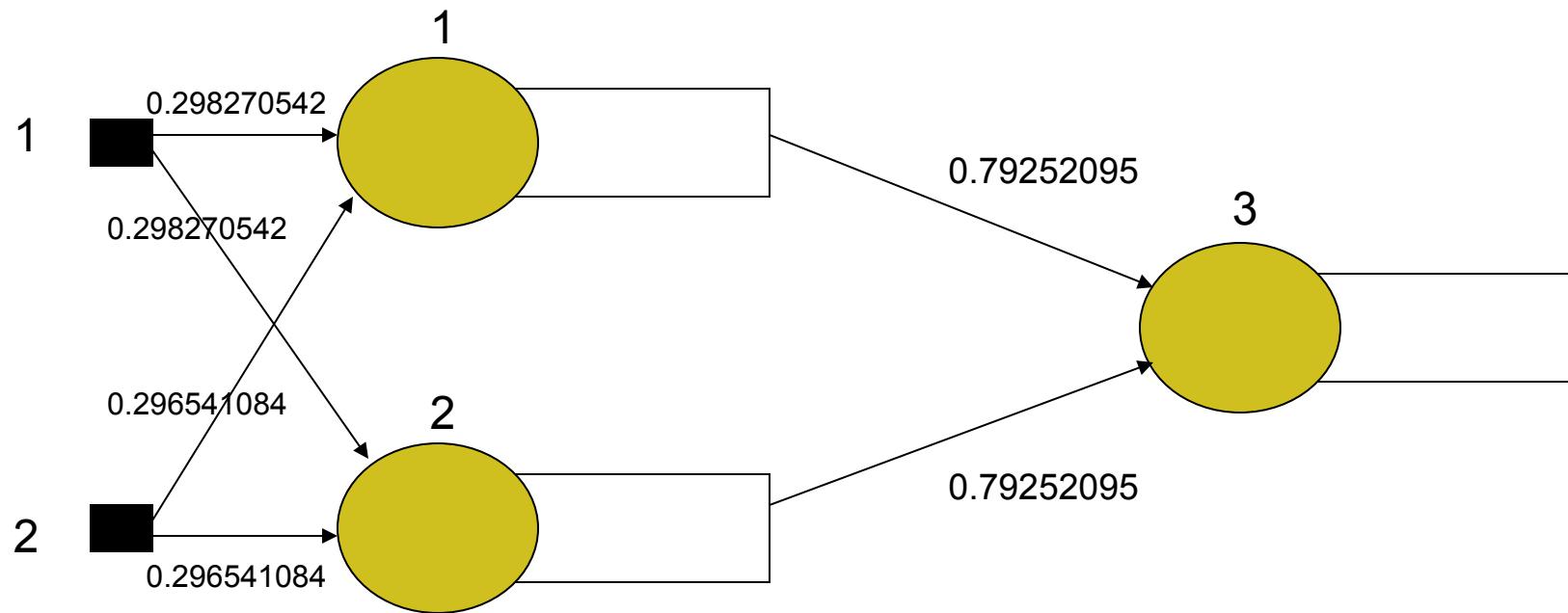


Back Propagation Epoch



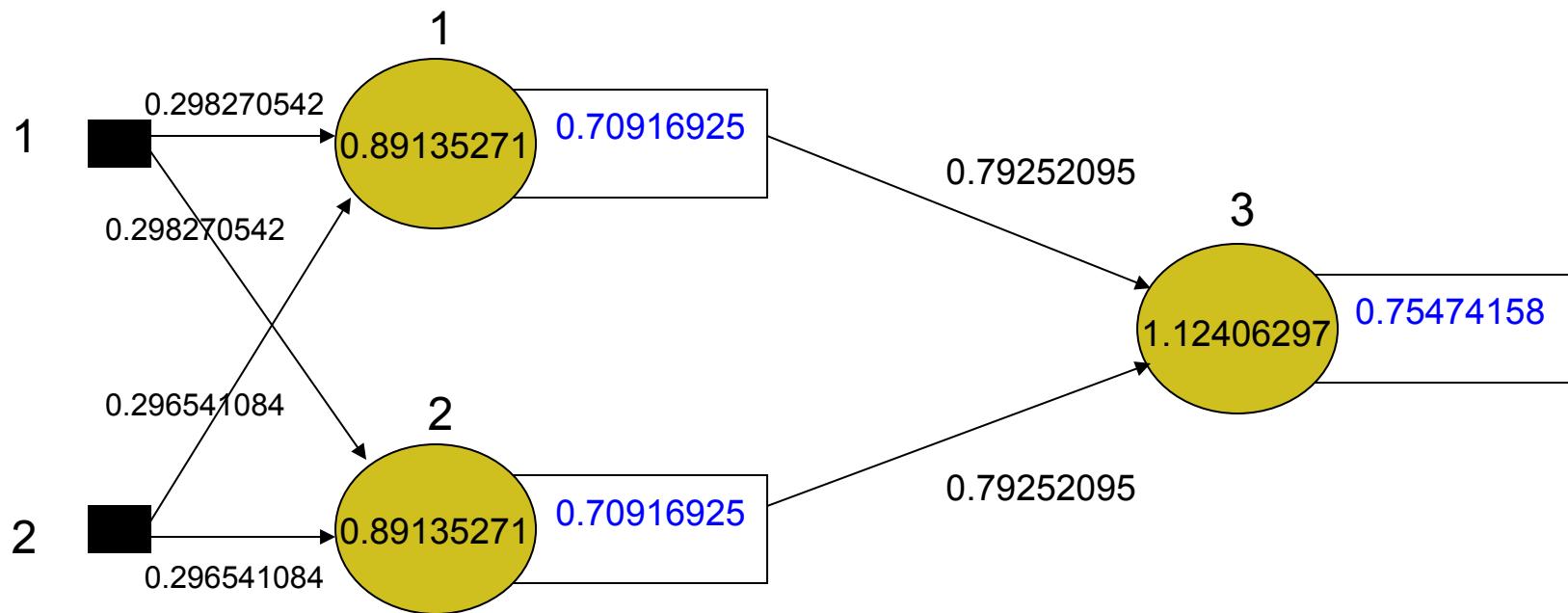


Start of a New Day





2nd Feed-forward Epoch



First Epoch

$$e = d - y = -0.05722387$$

$$E = \frac{1}{2}e^2 = 0.001637286$$

Second Epoch

$$e = d - y = -0.05474158$$

$$E = \frac{1}{2}e^2 = 0.00149832$$



The FF BP Summary

Feed-forward Epoch:

- We presented input values at the input layer to the hidden layer nodes.
- Computed activity and activation values in the hidden layer.
- Used those activation function values as input values for the output layer node(s).
- Computed the output error values.

Back-Propagation Epoch:

- Used the output error values in conjunction with the inputs to and the outputs of the output layer node(s) to compute the delta value(s) (gradient value(s)) for the output layer node(s).
- Used these delta value(s) to compute updated weights for the output layer node(s).
- Used these delta values in conjunction with the output and input values of the hidden layer nodes along with the original weights in the output layer to compute delta values for the hidden layer nodes.
- Used the delta values associated with the hidden layer nodes to compute updated weights for the hidden layer.



Introduction to Neural Networks

**Johns Hopkins University
Engineering for Professionals Program**

605-447/625-438

Dr. Mark Fleischer

Copyright 2014 by Mark Fleischer

**Module 6.1: Other Optimization Techniques—Theoretical Foundations
of the Simulated Annealing Algorithm**



This Sub-Module Covers ...

- Describe the history and mathematical foundations of the Simulated Annealing Program.
- Provides a foundation for study of stochastic neural networks which we will cover in later modules.
- Subsequent sub-modules will describe implementation issues and other optimization techniques.

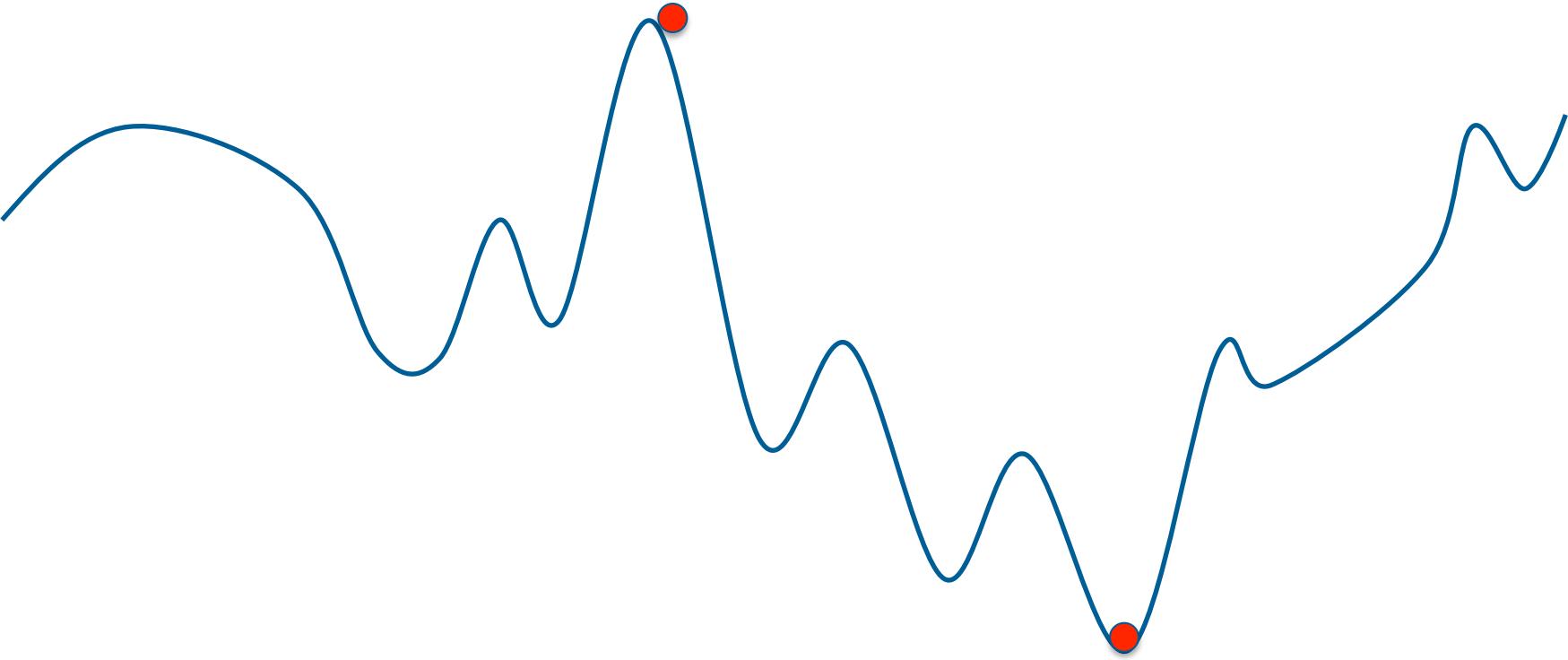


Overview

- Why Global Optimization?
- Metaheuristics
 - Simulated Annealing
 - Genetic Algorithms
 - Others as well
- Research Issues

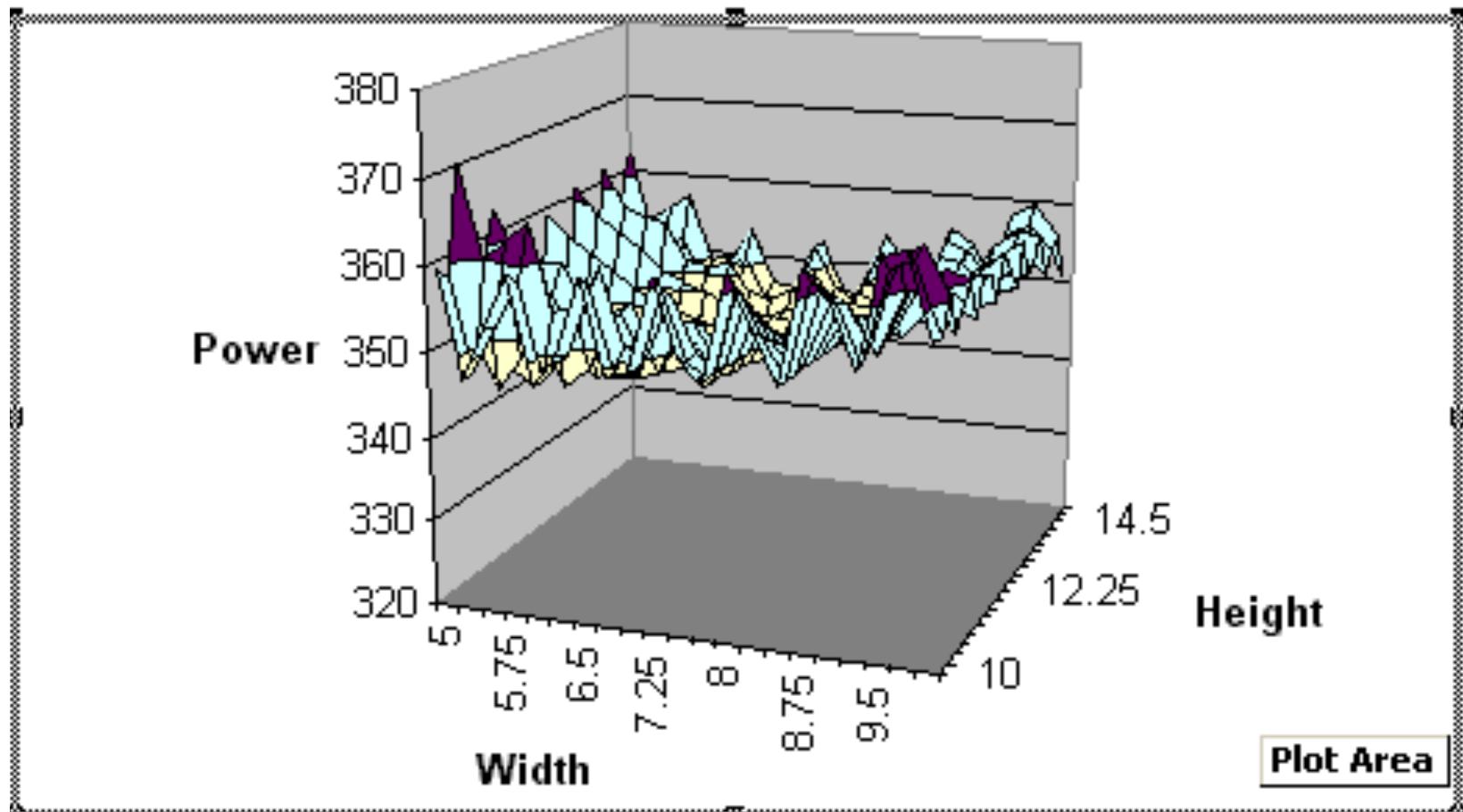


A Nasty Landscape





A Really Nasty Landscape





What is Visible to an Algorithm?

- Can't see the entire landscape.
 - Doing so equivalent to total enumeration.
- Only a neighborhood is visible. Why?
 - A tractable computational effort required.



Search Themes

- Directed e.g. gradient based.
 - Not applicable to most real-world problems. Why?
- Deterministic.
- Random.
- Random with some determinism.
- Memory dependent – avoid where we have already been.



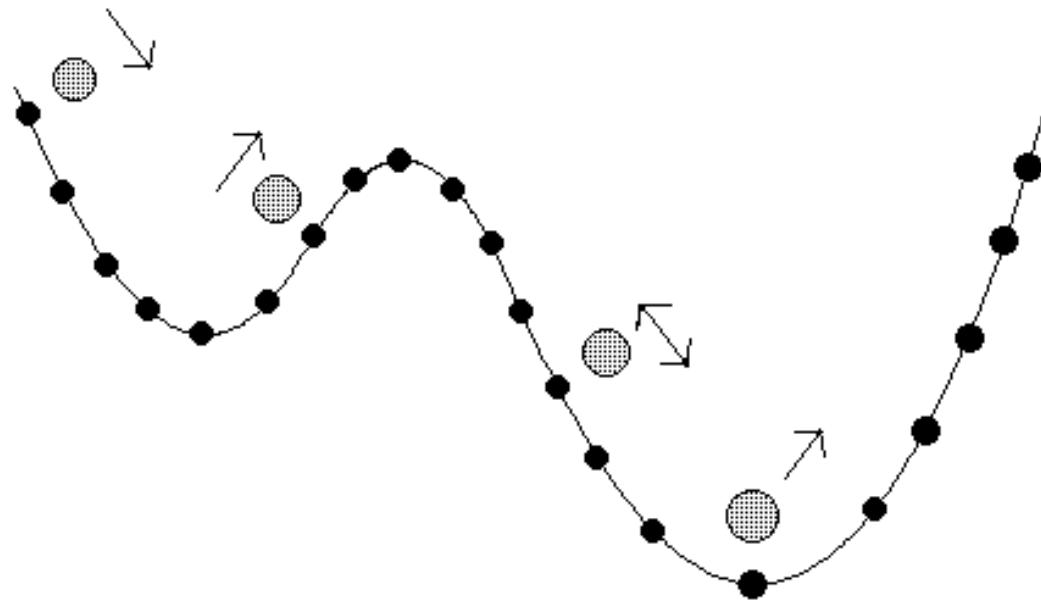
Search Paradigms

Rules for Path-Finding

- Thermodynamics
 - Simulated Annealing is based on the laws of thermodynamics
- Biological
 - Genetic algorithms are based on the paradigms of evolution:
natural selection, survival of the fittest, etc.
- Common-Sense/Intelligent



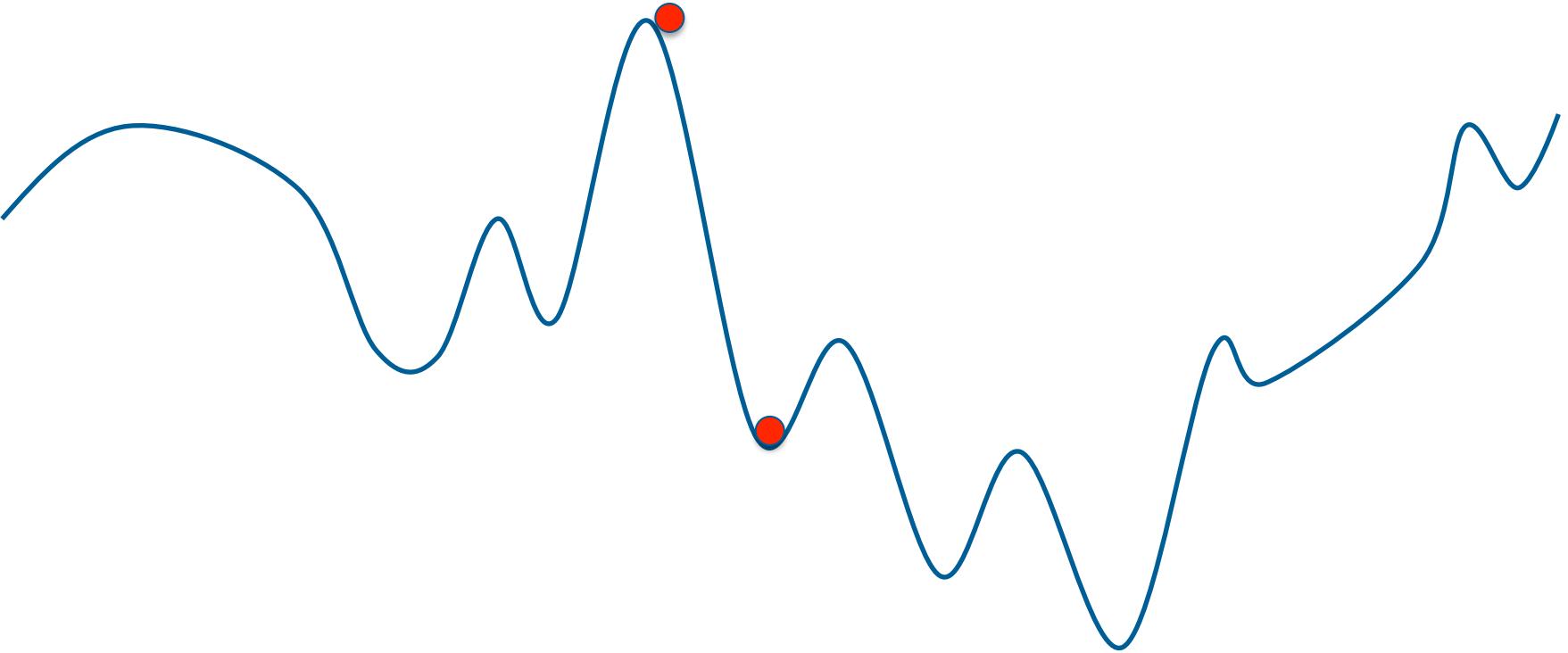
Simulated Annealing



Hill Climbing ---> Getting to the
Global Optima



A Nasty Landscape





Modeling Thermodynamic Systems

- a large (really large) population (ensemble) of physical entities, e.g., molecules of a gas, liquid, or solid.
- obeys the laws of thermodynamics and statistical mechanics.
- entities subject to collisions.
- entities have mass, position, velocity (kinetic energy)



Statistical Mechanics
Diffusion, entropy, temperature



Thermodynamics/Statistical Mechanics

- Particles change energy levels sporadically because of collisions.
- Energy level of system proportional to the **mean distance** between particles.
- Want to facilitate a simulation of such a system!

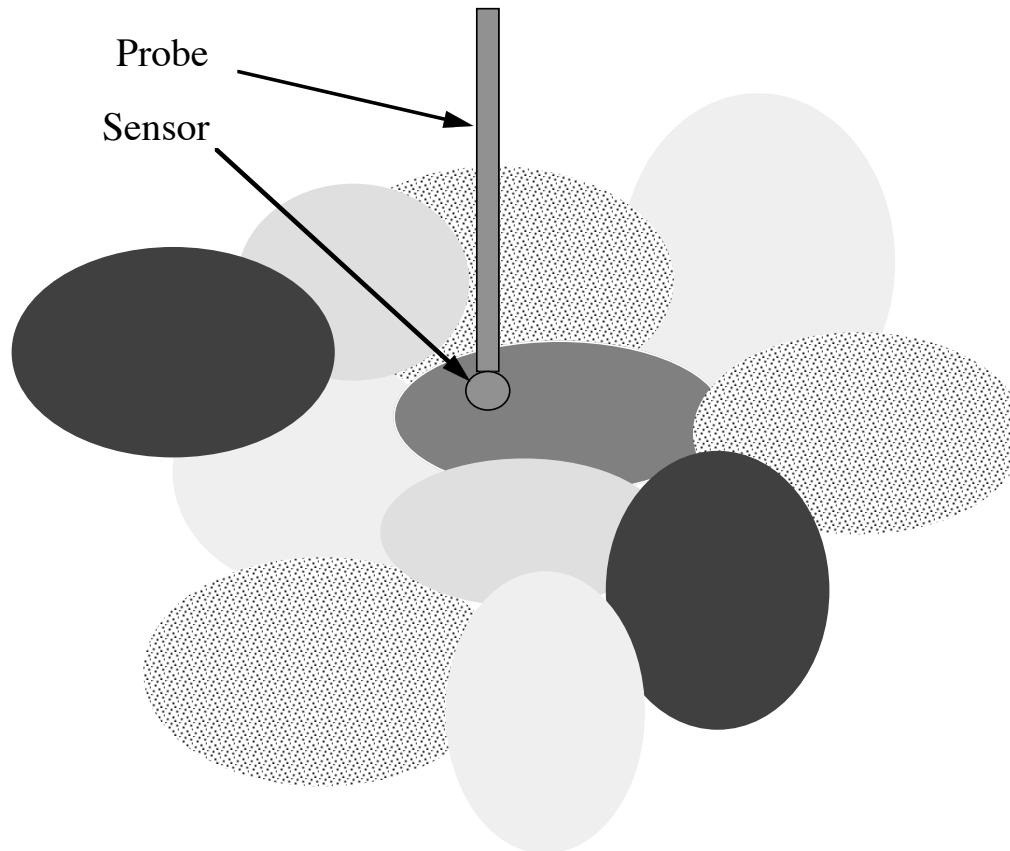


Thermodynamics/Statistical Mechanics

- Kinetic energy of particles is virtually a **continuous function** because of the number of particles involved.
- Define a **finite** set of states i each with an associated energy level E_i .
- Consider the evolution of "states" at a given temperature.
 - Some regions become more/less dense randomly over time.
 - Some regions gain/lose energy over time.



Gas with Varying Densities



Temperature at sensor fluctuates.
Energy states change over time.



Definition of Terms

t = an estimate of the mean kinetic energy of a substance, gas, etc.

$\pi_i(t)$ the stationary probability of energy state i at temperature t .

E_i the energy level of partition i .



Modeling the System of Particles

- An ensemble of particles seeks the maximum state of disorder.
- Statistical Mechanics translates this to a probability distribution that maximizes entropy.



What is Entropy?

A measure of the total ‘uncertainty’ associated with an ensemble of possibilities.

A single, non-negative scalar value that is associated with uncertainty.

We want it to encompass many different possible events.

We want it to be additive.

$$S_i \propto \frac{1}{p_i}, \quad S_{ij} \propto S_i + S_j$$

$$S_i = k \log \frac{1}{p_i} = -k \log p_i$$

$$S_{ij} = k \log \left(\frac{1}{p_i p_j} \right) = k \log \frac{1}{p_i} + k \log \frac{1}{p_j}$$



What is ‘Uncertainty’

A	B
0.01	0.99

Uncertainty = Expected Surprisal

$$U = p_A S_A + p_B S_B$$

$$U = -k \sum_i p_i \log p_i$$

A	B
0.5	0.5



An NLP

$$\text{Max} \quad -k \sum_{i=1}^n \pi_i(t) \log \pi_i(t)$$

$$\text{s.t.} \quad \sum_{i=1}^n \pi_i(t) E_i = k t$$

$$\sum_{i=1}^n \pi_i(t) = 1$$

$$t, \pi_i(t), E_i \geq 0 \quad \forall i$$



The Boltzmann Distribution

$$\pi_i(t) = \frac{e^{-E_i/kT}}{B(t)}$$

$$B(t) \equiv \sum_{i=1}^n e^{-E_i/kT}$$

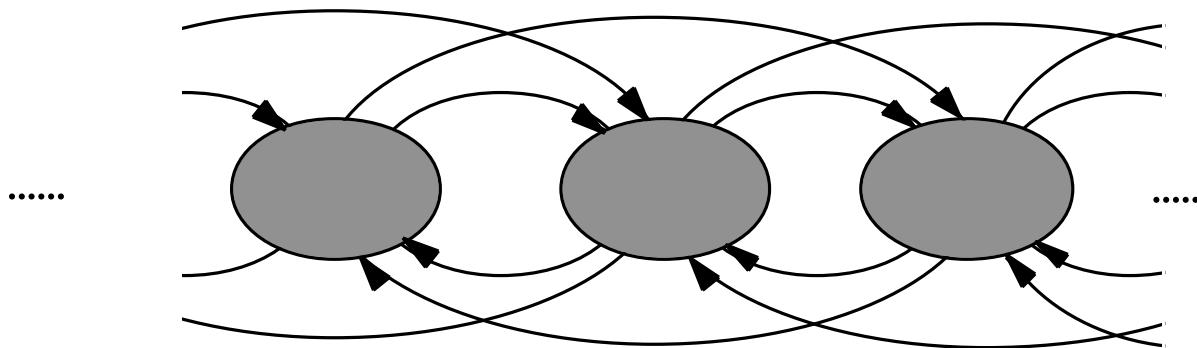


State Transitions

(Metropolis, *et al.* 1953)

Detailed balance requires that

$$\pi_i(t) p_{ij}(t) = \pi_j(t) p_{ji}(t)$$



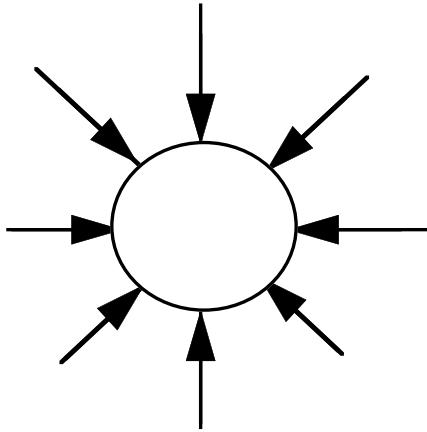


State Transitions

(Metropolis, *et al.* 1953)

Global balance requires that

$$\pi_i(t) = \sum_j \pi_j(t) p_{ji}(t)$$





Equations of State

(Metropolis Acceptance Criterion)

- Regions (states) move spontaneously to different energy levels.
- Given current energy level i ,
 - if $\Delta E = E_j - E_i \leq 0$ then that region moves to the new energy level j .
 - if $\Delta E = E_j - E_i > 0$ then that region moves to the new energy level j with the following acceptance probability

$$\Pr\{\text{Accept state } j\} = e^{-\Delta E/t}$$



Transition Probability

Balance conditions lead to the form for the transition probabilities:

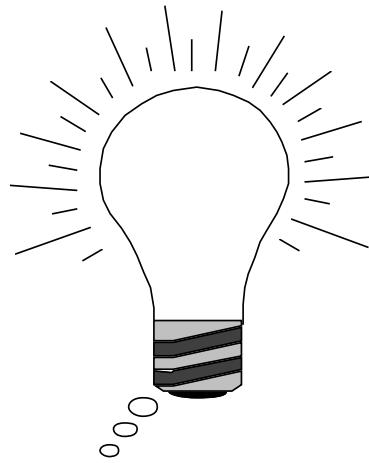
$$p_{ij}(t) = \begin{cases} G_{ji} e^{-\Delta E^+ t} & \Delta E_{ji} > 0, j \neq i \\ G_{ji} & \Delta E_{ji} \leq 0, j \neq i \\ 1 - \sum_k p_{ik}(t) & j = i \\ 0 & \text{otherwise} \end{cases}$$



Kirkpatrick, et al. (1983), Cerny (1985)

The annealing process lowers
the energy content of a solid.

Energy content of a solid \Leftrightarrow Objective Function of an optimization problem



The Simulated Annealing Algorithm



Mathematical Properties of SA

Convergence in Probability (stationary probability)

$$\lim_{k \rightarrow \infty} \Pr\{S_k \in S_{\text{OPT}}\} = 1$$

Convergence in Distribution

$$\lim_{t \rightarrow 0} \pi_i(t) = \begin{cases} \frac{1}{|S_{\text{OPT}}|} & \text{if } i \text{ is in the set of optima} \\ 0 & \text{otherwise} \end{cases}$$



Summarizing Simulated Annealing

- The Simulated Annealing is based on modeling a thermodynamic system.
 - As in thermodynamics, we model large-scale behavior instead of modeling the dynamics of each involved entity.
 - Metropolis formulated ‘Equations of State’ as a non-linear mathematical program.
 - Led to the mathematical form of transition probabilities between states.
- Provides equivalent state probabilities as in thermodynamic systems.
- Kirkpatrick and Cerny found a way to apply these equations of state to simulate the annealing process.



Conclusion

- More metaphors exist
 - Tabu Search, Adaptive Memory Programming and Genetic Algorithms
- Active area of research
- Many overlooked/ignored possibilities
- Balance accuracy, search time
- Structure, properties, computer architecture.



Introduction to Neural Networks

**Johns Hopkins University
Engineering for Professionals Program**

605-447/625-438

Dr. Mark Fleischer

Copyright 2014 by Mark Fleischer

Module 6.2: Implementation of the Simulated Annealing Algorithm



This Sub-Module ...

- describes approaches for implementing the Simulated Annealing Program.
- Implementation depends on the type of problem we're trying to solve.
 - We'll look at Combinatorial Optimization Problems (COPs)
 - Continuous variable problems.
 - Touch on approaches for parallelizing SA.



Metropolis Acceptance Criterion

States i and j
Objective Function Values f_i and f_j
with $\Delta f_{ji} = f_j - f_i$

$$\Pr \{ \text{Accept Cand. } J \} = \begin{cases} e^{-\Delta f_{ji}^+ / t_k} & \Delta f_{ji} > 0, j \in N(i) \\ 1 & \Delta f_{ji} \leq 0, j \in N(i) \\ 0 & \text{otherwise} \end{cases}$$



The Simulated Annealing Algorithm

1. Select a new state with a new objective function value.
2. Calculate the value of Δf .
3. If $\Delta f \leq 0$ (for a minimization problem) then the new state becomes the current state. If $\Delta f > 0$ then the new state becomes the current state via the Metropolis Acceptance Criterion. If state J is not accepted, keep state I as the current state.
4. Lower temperature and goto 1.



Requirements for SA

1. Define an appropriate set of states.
2. Define a suitable neighborhood structure.
 This entails the candidate generation mechanism.
3. Define a suitable cost/objective function.
 This entails a method for calculating the change in objective function value.
4. Define a cooling schedule.
5. Define stopping criteria.



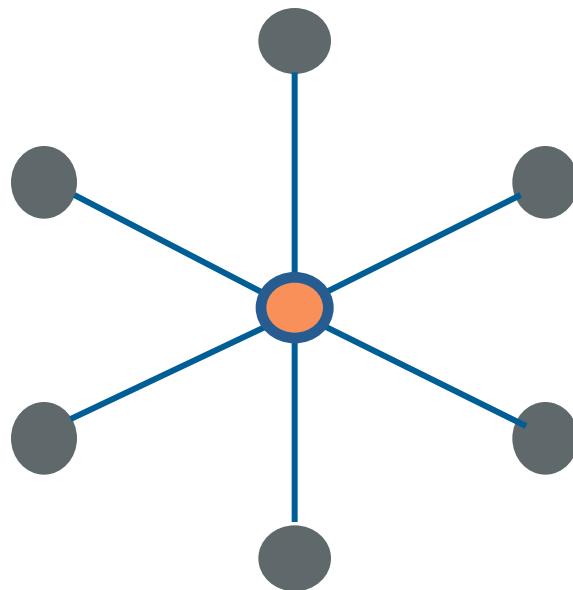
Cooling Schedules

$$t_k = \frac{\gamma}{\log(c + k)}$$

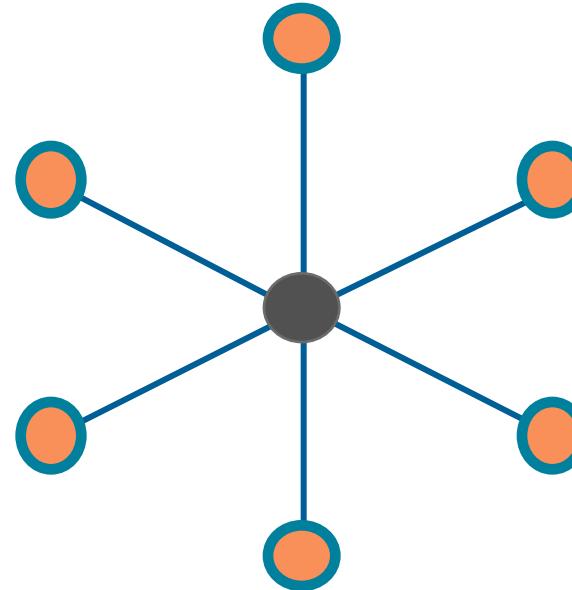
$$t_k = \frac{\gamma}{c + k}$$



Combinatorial Optimization Problems (COPs)



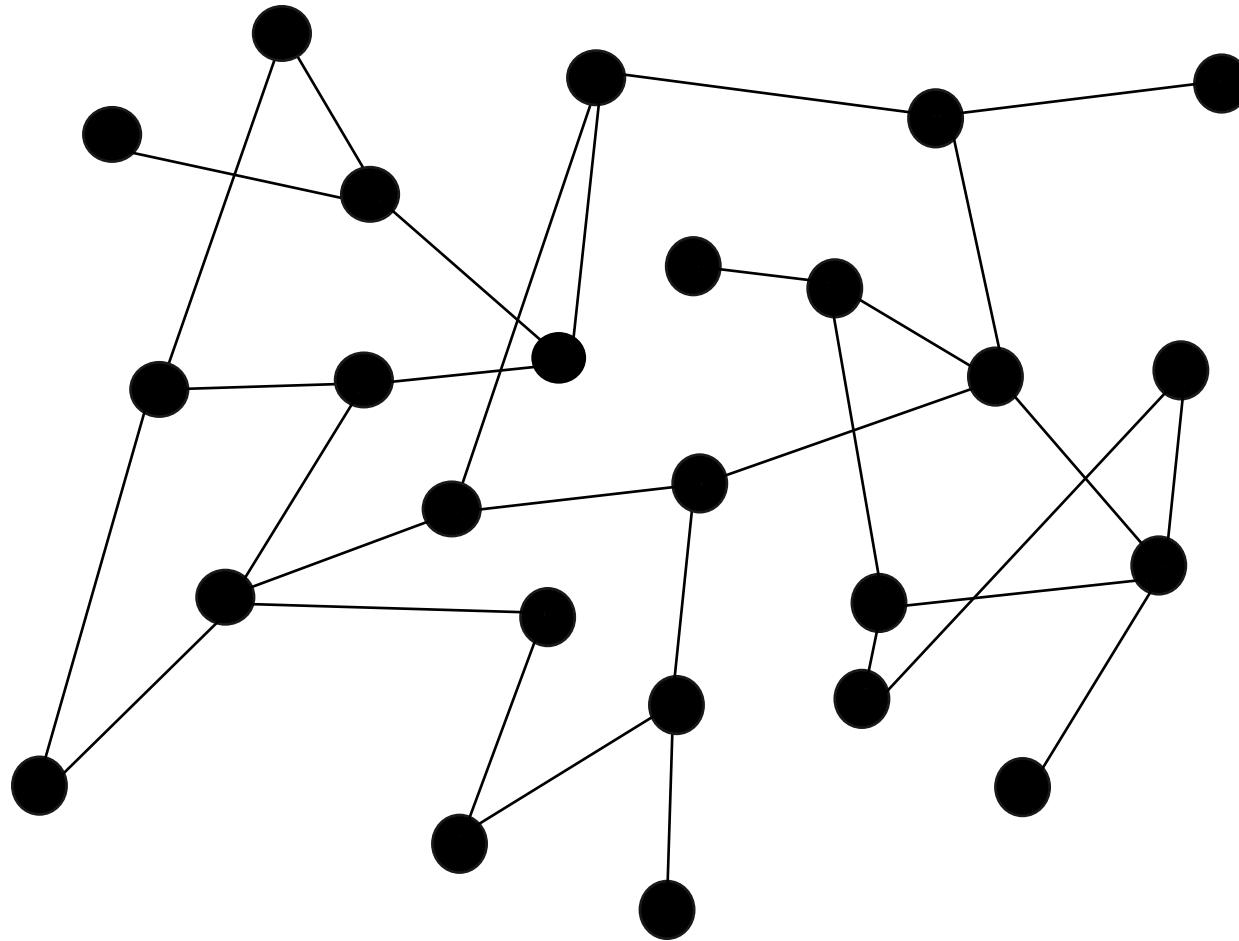
Minimum Vertex Cover



Maximum Independent Set

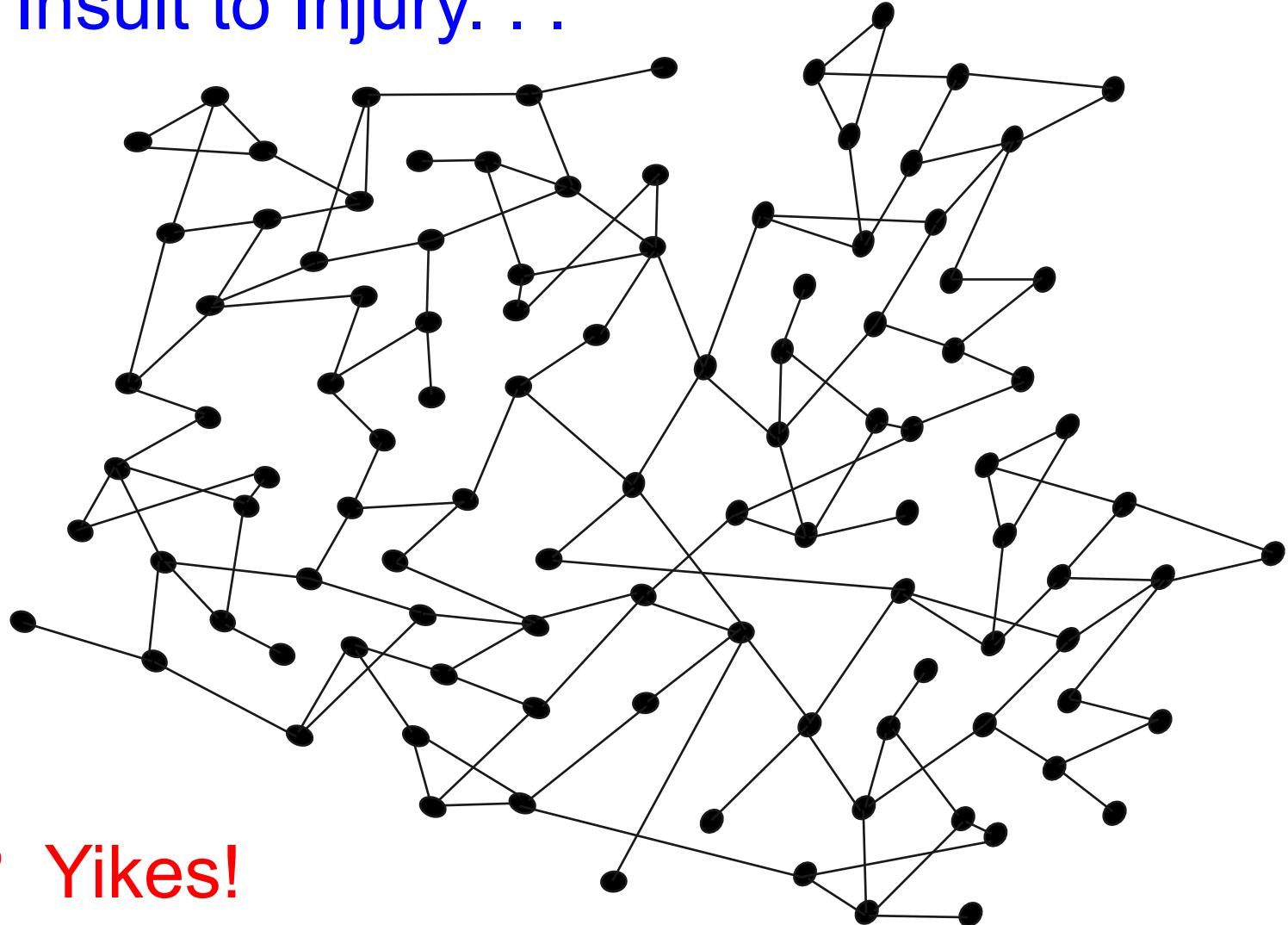


Not Always So Obvious . . .





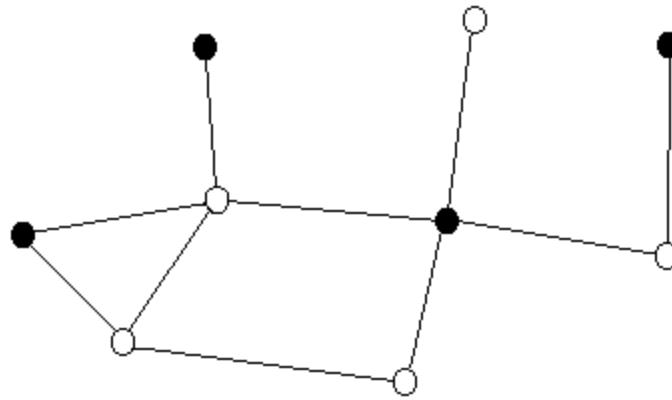
Adding Insult to Injury. . .



21,000,000? Yikes!



Implementation Issues



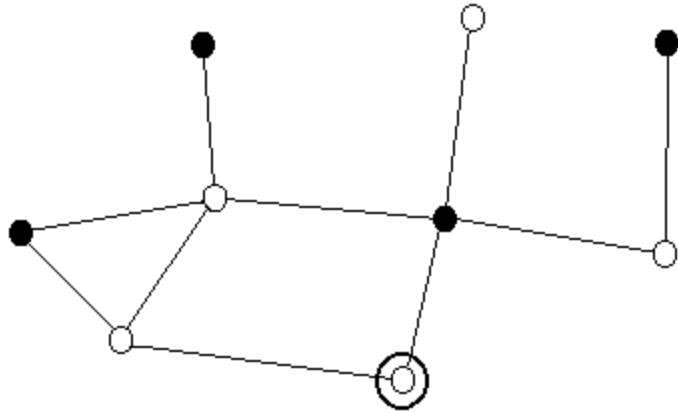
A good objective function is simply the number of nodes in the current set.

Current Solution = 4
Maximum Set Size = 5

How do we get to better solutions without doing a lot of work?



Select a node, any node...



But this violates independent set constraints!

Use some penalty function.

Penalty function should decrease
objective function value

and

be based on those elements which violate constraints.



Objective Functions

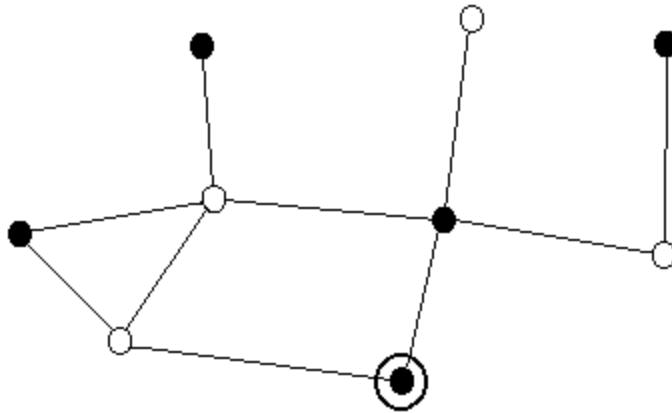
What are the elements which violate constraints?

edges

$$f(G) = \text{number of nodes} - g(\text{edges})$$



New Combination of Nodes, New Objective Function Value



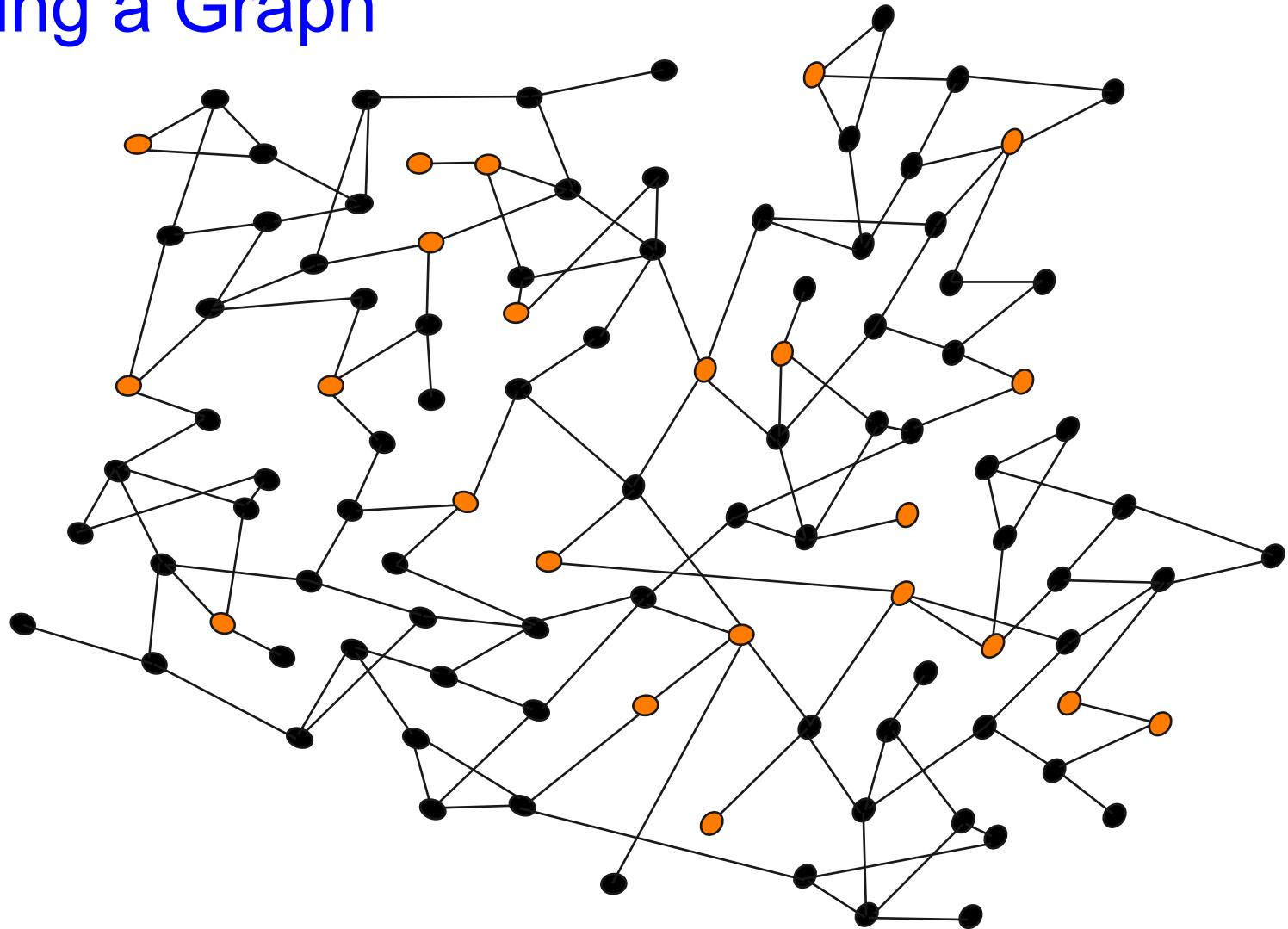
Current Objective Function Value:

From $f_{\text{indep set}}(V', G) = v' - \lambda E_1(V') = 4 - 0$

to $f_{\text{indep set}}(V', G) = v' - \lambda E_1(V') = 5 - \lambda 1$



Annealing a Graph





E.g. the Stationary Probability

$$\pi_i(t) = \frac{e^{-f_i/t}}{\sum_{i=1}^s e^{-f_i/t}}.$$

Boltzmann Distribution Function

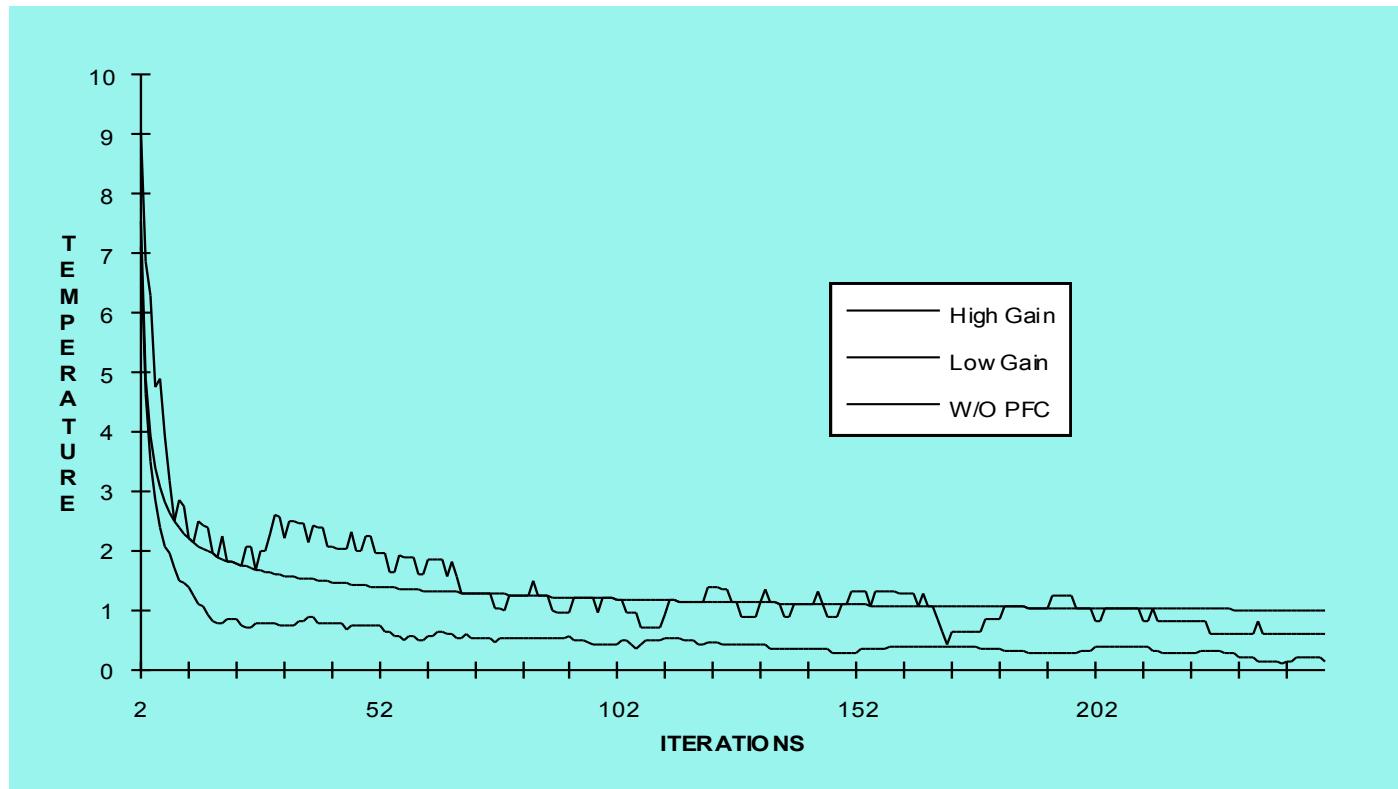


Joint Distribution

$$\begin{aligned}
 \pi_{i_1 \dots i_p}(t) &= \prod_{m=1}^p \pi_{i_m}(t) \\
 &= \prod_{m=1}^p \frac{e^{-f_{i_m}/t}}{\sum_{i_m=1}^s e^{-f_{i_m}/t}} \\
 &= \frac{e^{-\left(\sum_{m=1}^p f_{i_m}\right)/t}}{\prod_{m=1}^p \sum_{i_m=1}^s e^{-f_{i_m}/t}} \\
 &= \frac{e^{-f_{i_1 \dots i_p}/t}}{\sum_{i_1 \dots i_p}^{s^p} e^{-f_{i_1 \dots i_p}/t}}
 \end{aligned}$$



Non-Monotonic Cooling



Journal of Heuristics Vol. 1 No. 2 p.245



Experimental Results and Observations

Number of Processors	Gain Setting	Objective Function Values				z-value
		\bar{F}_{\min}	s_F^2	\bar{G}_{\min}	s_G^2	
5	1	0.683	0.303	0.541	0.057	1.290
5	5	0.212	0.057	0.536	0.536	-2.300
5	10	0.232	0.090	0.541	0.057	-4.419
10	1	0.488	0.560	0.388	0.040	0.709
10	5	0.130	0.027	0.324	0.047	-3.899
10	10	0.110	0.020	0.438	0.062	-6.261

Table 5.1 : Efficacy of PFC by Candidate Generation



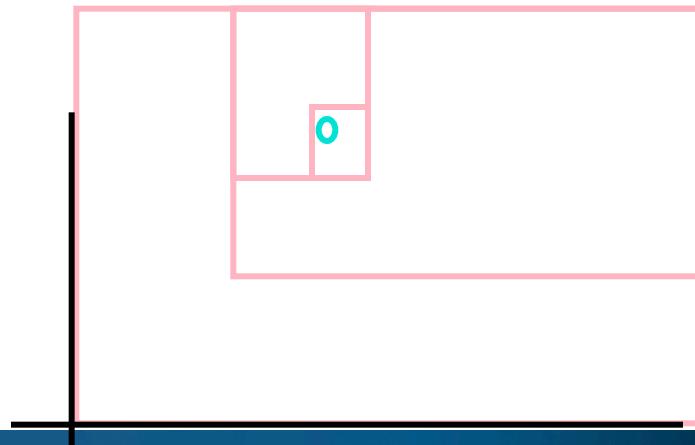
What About Continuous Variable Problems?

- Picking candidate solutions is main issue.
- SA doesn't work well on continuous variable problems.



Recursive Intensification

- Accelerates convergence to optima, experimentally.
- Increases probability of never converging to the optima.





Introduction to Neural Networks

**Johns Hopkins University
Engineering for Professionals Program**

605-447/625-438

Dr. Mark Fleischer

Copyright 2014 by Mark Fleischer

Module 6.3: Genetic Algorithms



This Sub-Module ...

- Briefly describes some of the history behind the development of genetic algorithms (GAs).
- Describes the basic ideas behind genetic algorithms.
- Implementation generally involves a few types of ‘operators’ inspired by biological systems



Genetic Algorithms

- Another metaphor for random search.
- Based on biological evolution.
- Analogies to biological mechanisms of sexual selection.
 - Chromosomes/Genes/DNA: solution schema
 - Combining DNA: crossover operation
 - Mutation: mutation operator
 - Natural Selection: fitness function

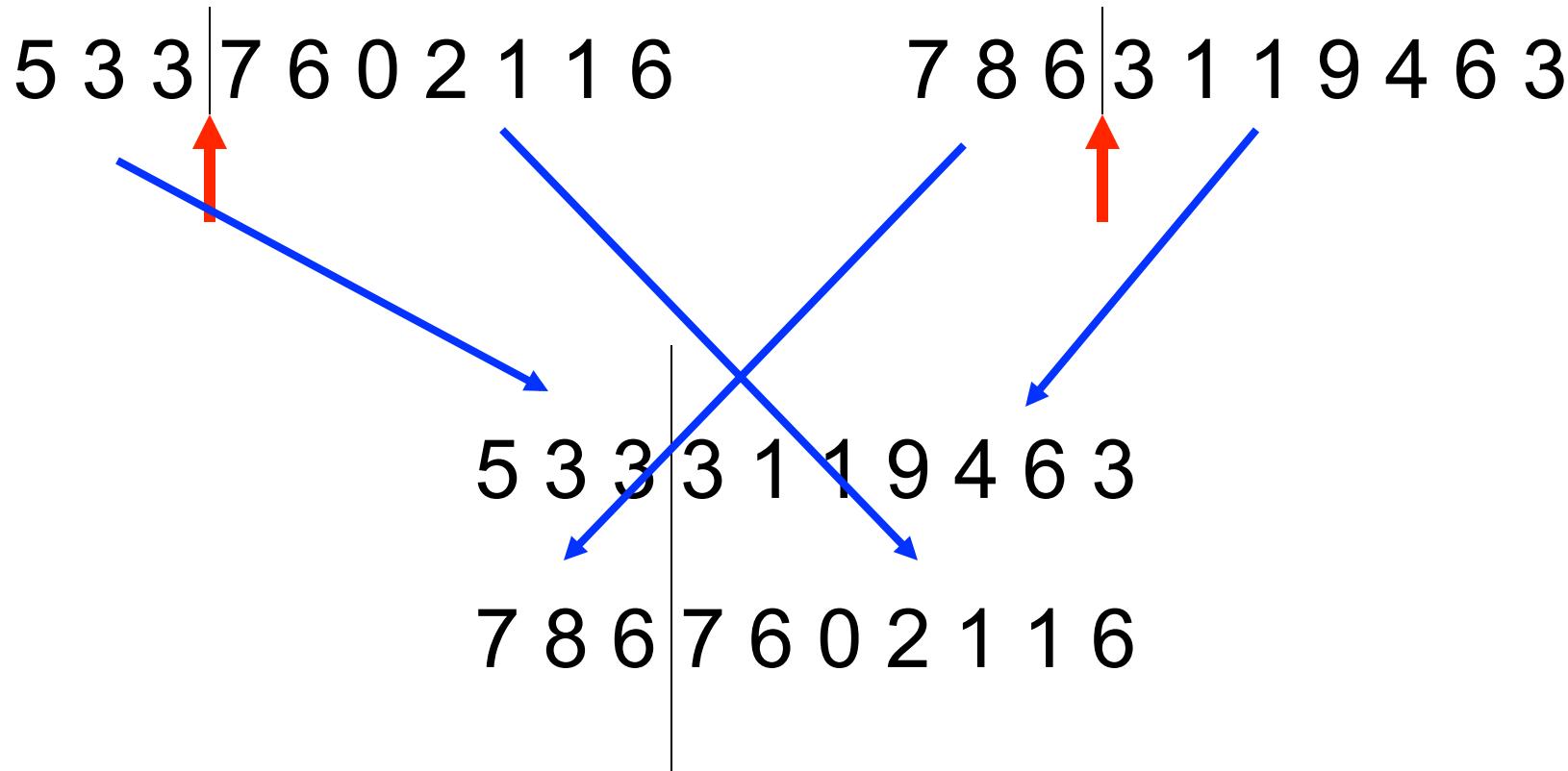


GA -- History

- Invented by Holland (1960s).
- Schema theorem developed in 1975.
- Many implementations, possibilities, and applications.
- One of the most useful global optimization technique.



Crossover Operator



Two new “children”(solutions).



Mutation Operator

5 3 3 3 1 1 9 4 6 3



5 3 3 3 1 1 5 4 6 3



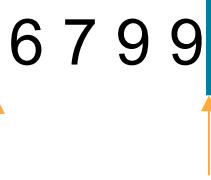
Natural Selection via Fitness Operators

- Once new children are created, a *fitness operator* is applied to all population members.
- Those members that rank high enough in fitness value, are kept for another generation of crossover and mutation operations.
- Those members that do not rank sufficiently high are “**deleted**”.





Implementation Issues

- Crossover operators
 - Many different operators possible
 - *E.g.*, 3 5 1 1 | 6 7 9 9 | 0 5 5 3 2
- Mutation operators
 - Different probability values
 - Different functional dependencies
- Fitness operators
 - Different scaling functions



Conclusion

- We've looked at two meta-heuristics inspired by nature.
 - Simulated Annealing: Thermodynamics and Statistical Mechanics
 - Genetic Algorithms: Evolution and Natural Selection
- More metaphors exist
- Many many research papers about on these topics.
- They can be applied to neural networks.



Introduction to Neural Networks

**Johns Hopkins University
Engineering for Professionals Program**

605-447/625-438

Dr. Mark Fleischer

Copyright 2013 by Mark Fleischer

Module 7.1: The Training Process

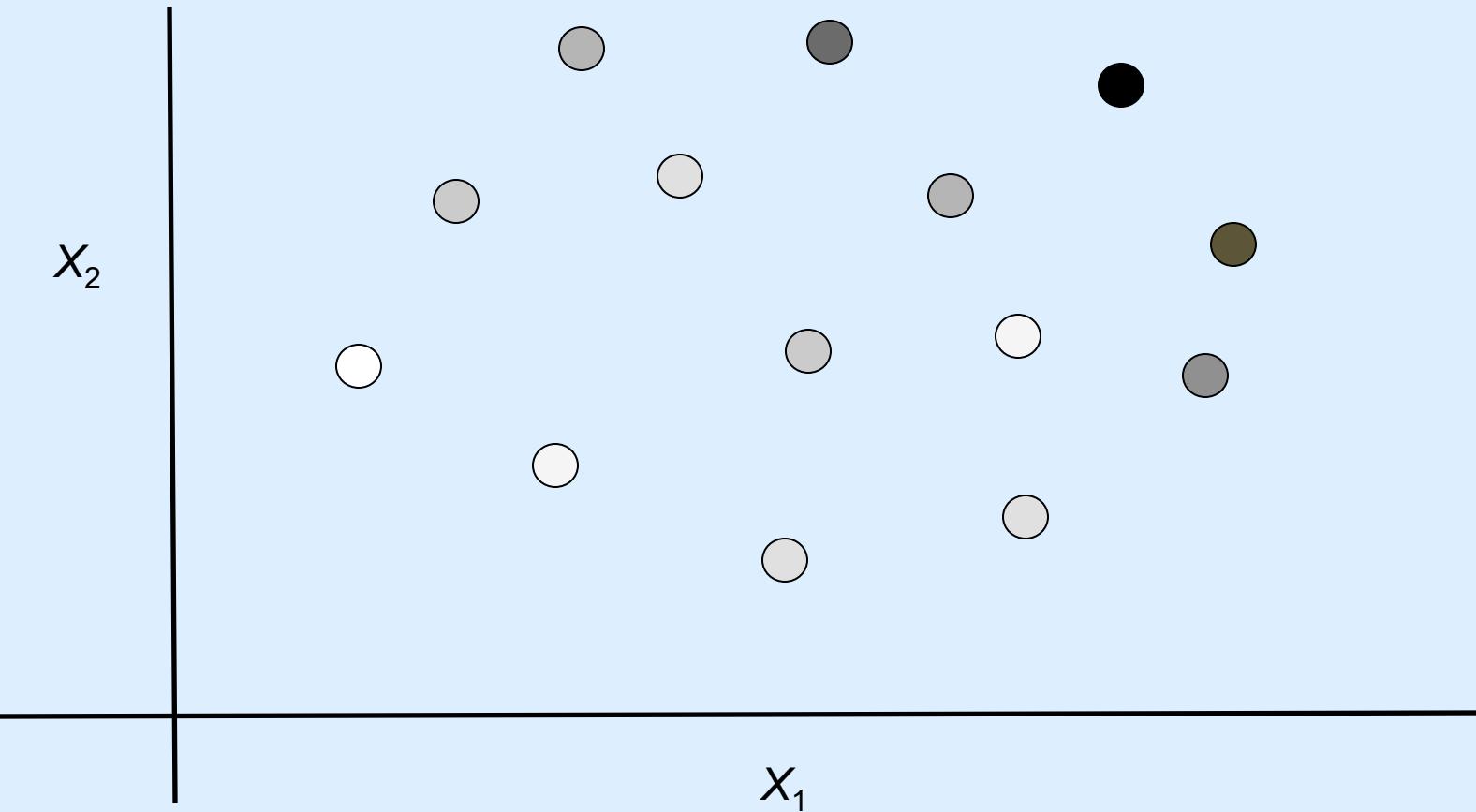


This Sub-Module Covers ...

- What a typical training scenario looks like.
- Issues that arise during the training process.
 - Overtraining and efficiency
 - Lay the groundwork for describing procedures to handle multiple input/output data pairs and related issues
 - Motivate important performance measures covered in subsequent sub-modules



A Classification Problem





Can A Neural Network Model a Polynomial?

- From the manner of training a Neural Network, they can
 - Interpolate data given the training data.
 - Can they model a polynomial?
- How can we mathematically define a polynomial given specific data?



Collocation Method of Polynomials

To define an n -degree polynomial function $P_n(x)$ that goes “through” a specified set of points (x,y) , define

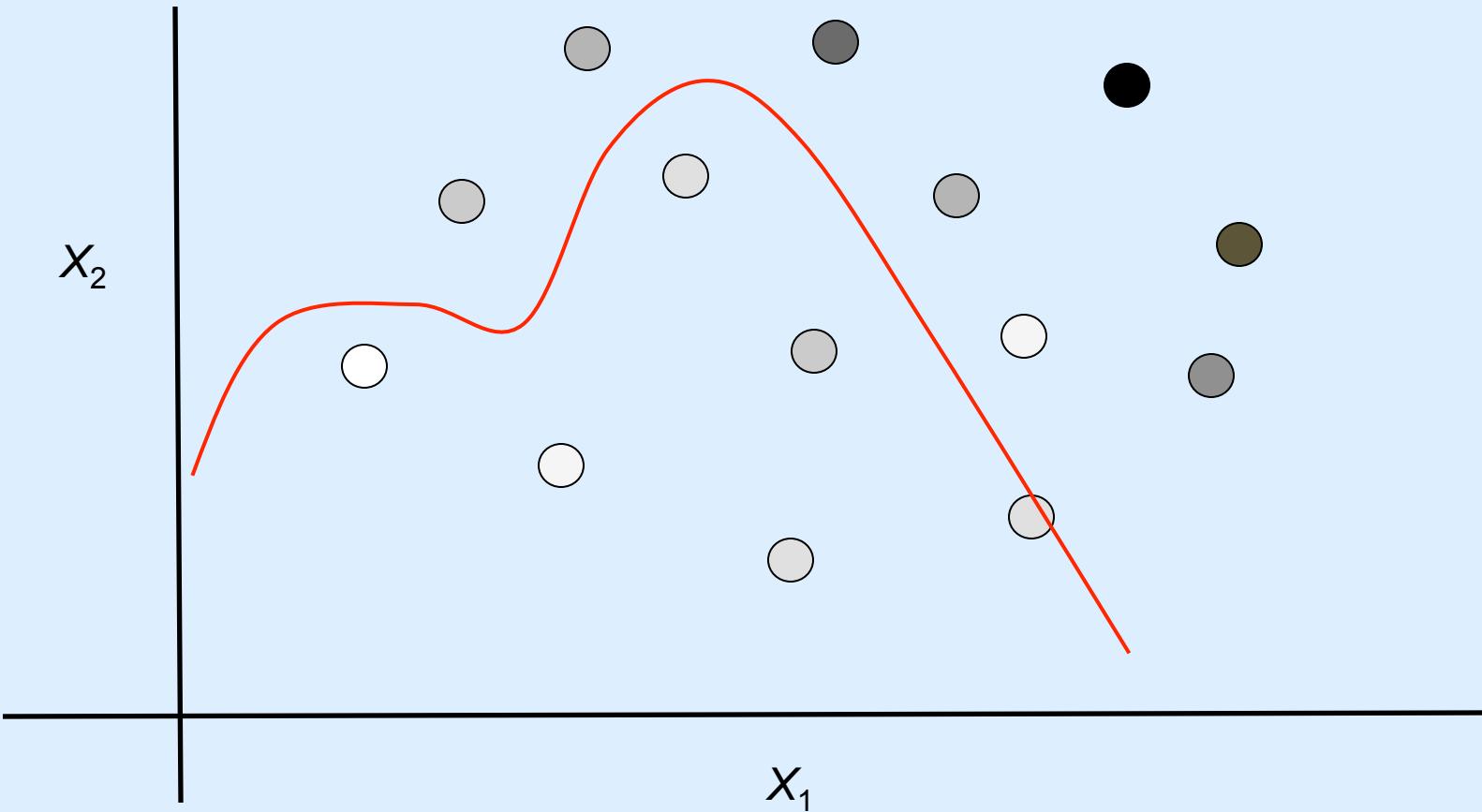
$$P_n(x) = \sum_{i=0}^n f_i L_i(x)$$

where

$$L_i(x) = \prod_{\substack{k=0 \\ k \neq i}}^n \frac{(x - x_k)}{(x_i - x_k)}$$

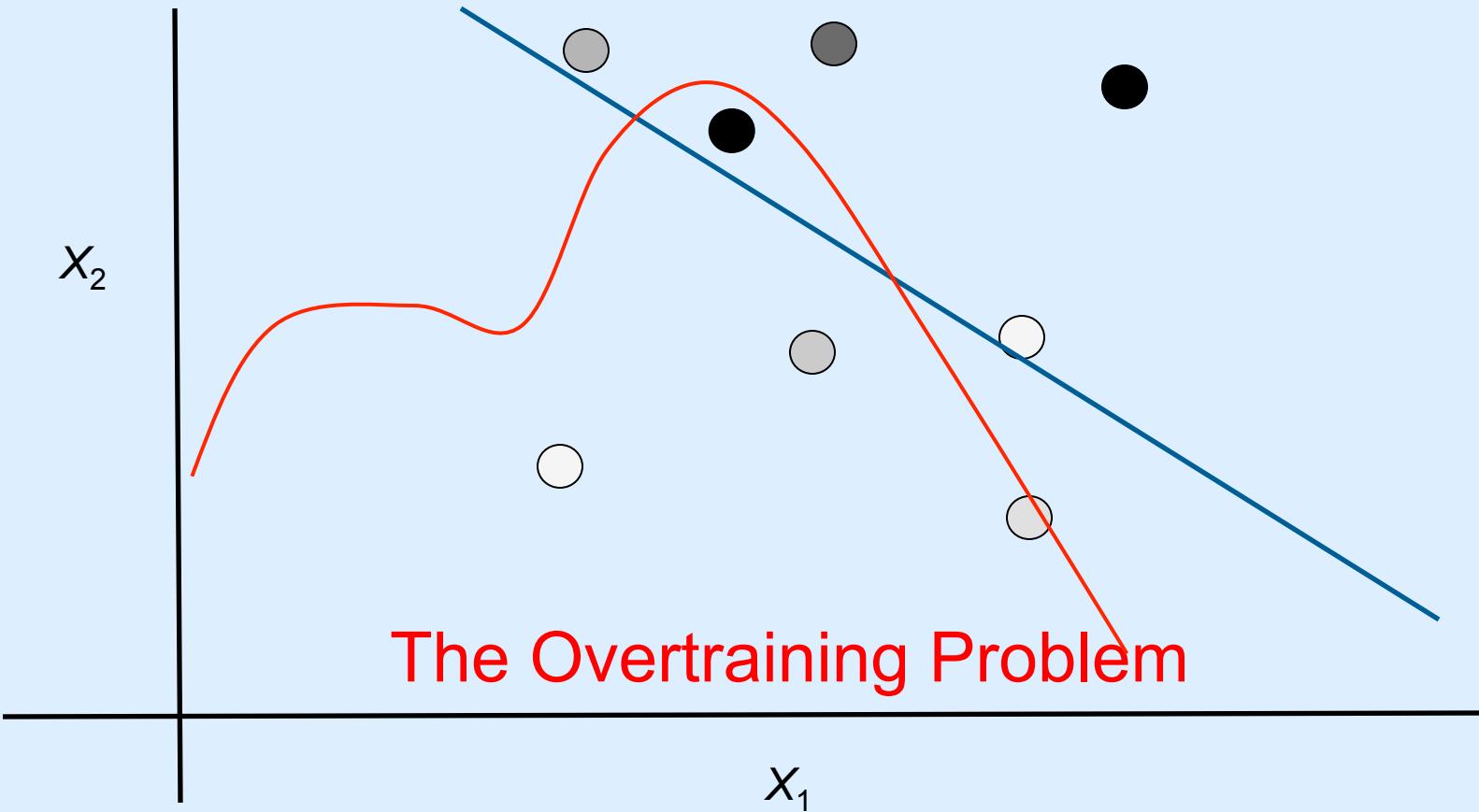


A Classification Problem





A Classification Problem





Training: Supervised Learning

- Want to use a neural network to handle data in some **future** scenario.
- Need to train the network with data we already have to do this.
- But we also need to assess how well the network works.
- Obviously, we can't use the same data we trained the network with to assess its performance. Why?



Training: Supervised Learning

- A common approach is to minimize errors in a FFBP setup.
- Obtain raw data:
 - parts corresponding to inputs,
 - parts corresponding to associated outputs.
- Divide the data into two parts: Training Data and Testing data.
- Take the training data and **train and train and train**, (but not too much) until errors are as small as possible.
- Then test the trained network using the testing data set to see how well it works using various performance criteria.
- Then make adjustments as necessary.



Partitioning Data Sets and Statistical Analysis

- Many ways to partition data
 - Just once as in the following example, or
 - Many times using different ‘partitions’ and then average the results (the weights) from each partition.
- Many statistical methods do this type of partitioning. Popular methods include:
 - Bootstrap Method
 - Jackknife Method
- Basically idea is to use the same data but create many training/testing sets. Randomly select out the training set and the testing set. E.g. for 20 I/O pairs, using 10 I/O pairs for training, 10 I/O pairs for testing. But

$$\binom{20}{10} = \frac{20!}{10!10!} = 184,756$$



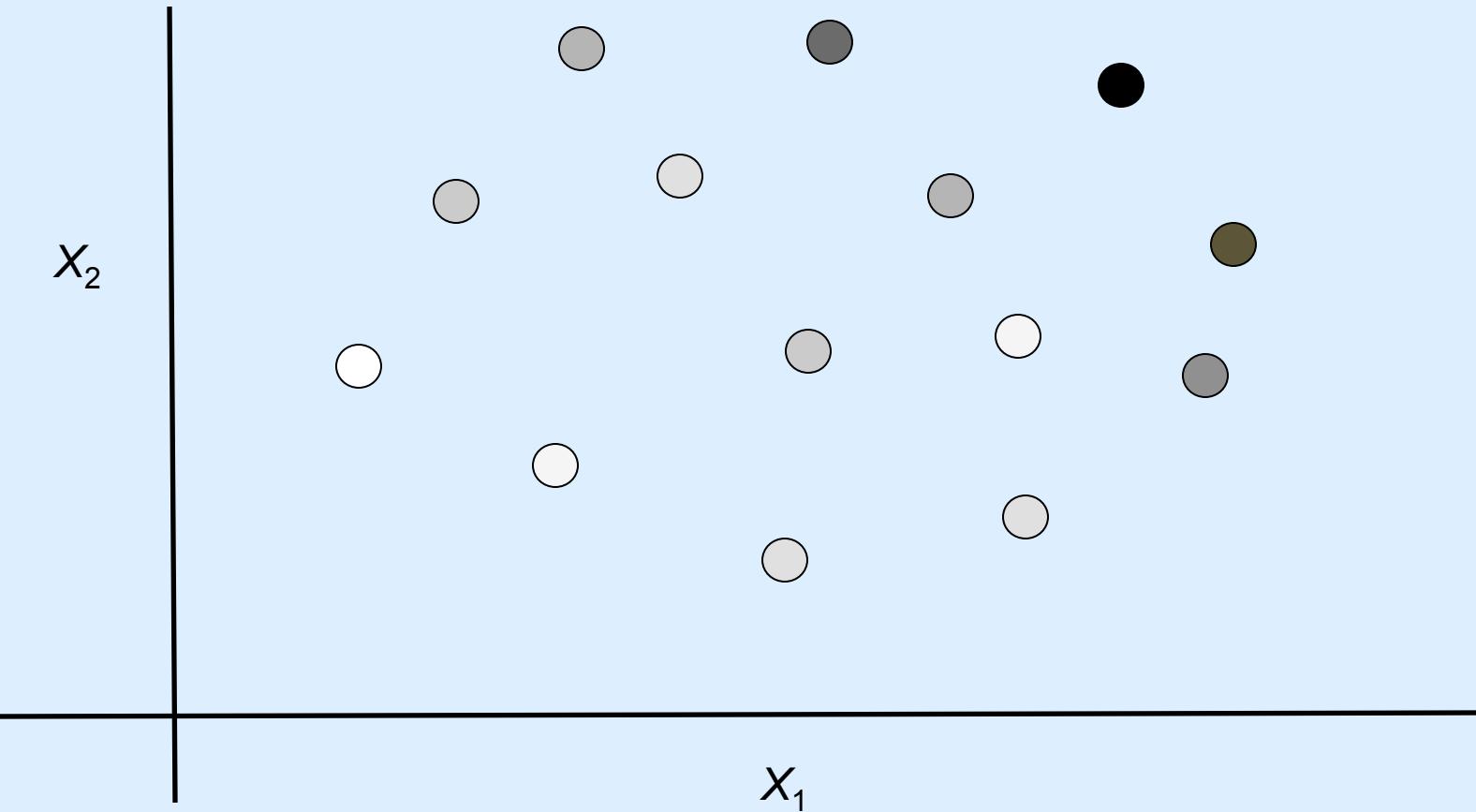
This Scenario Requires

- A means for evaluating network performance (obviously).
 - Performance refers to how well the neural network performs the desired function for which it is being trained.
 - We will, for now, **assume some method of measuring performance** and get into the details in later sub-modules.
- Also means being able to use the neural network and associated data effectively and efficiently!
 - Computational efficiency is important for training AND assessing performance.

Efficiency

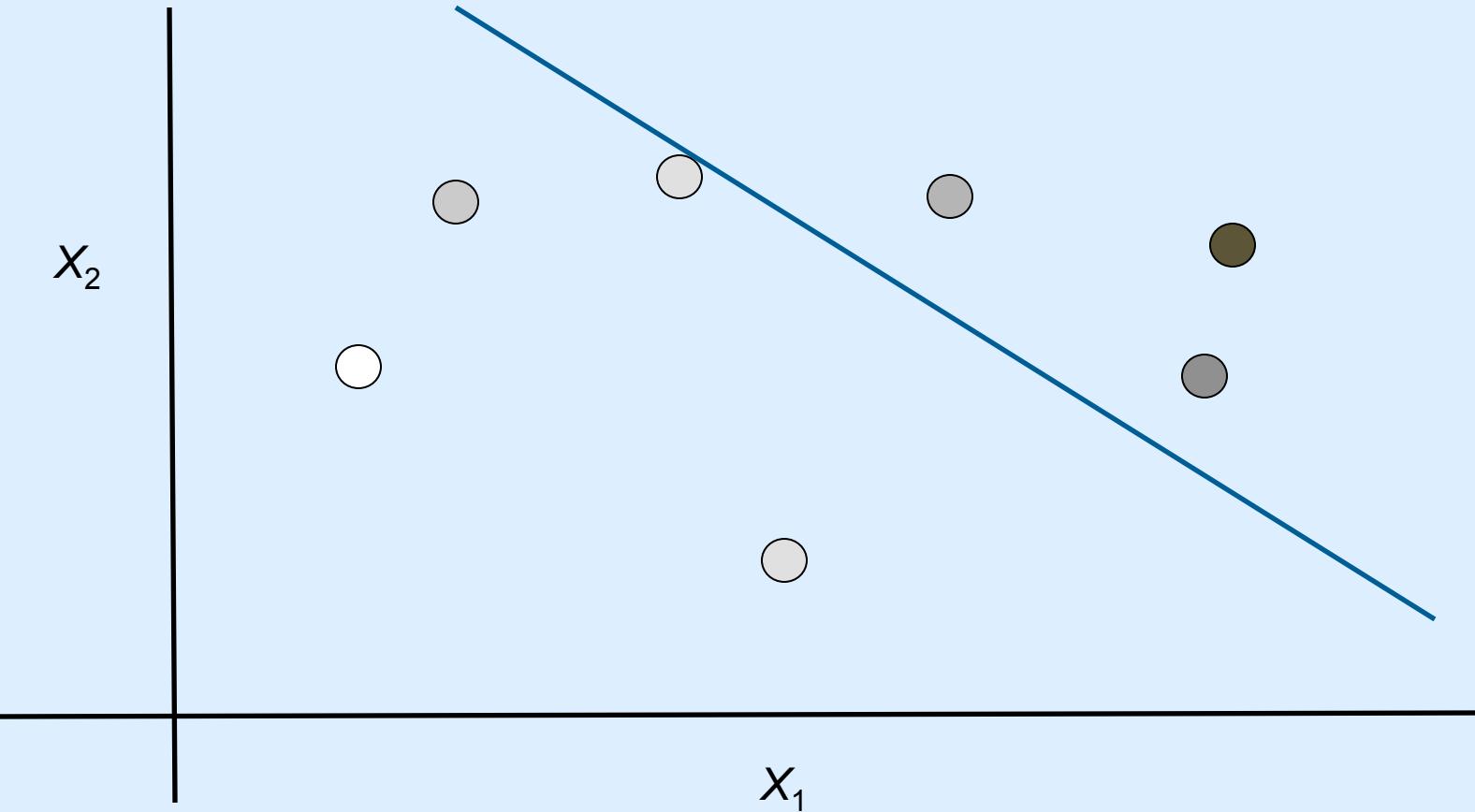


A Classification Problem



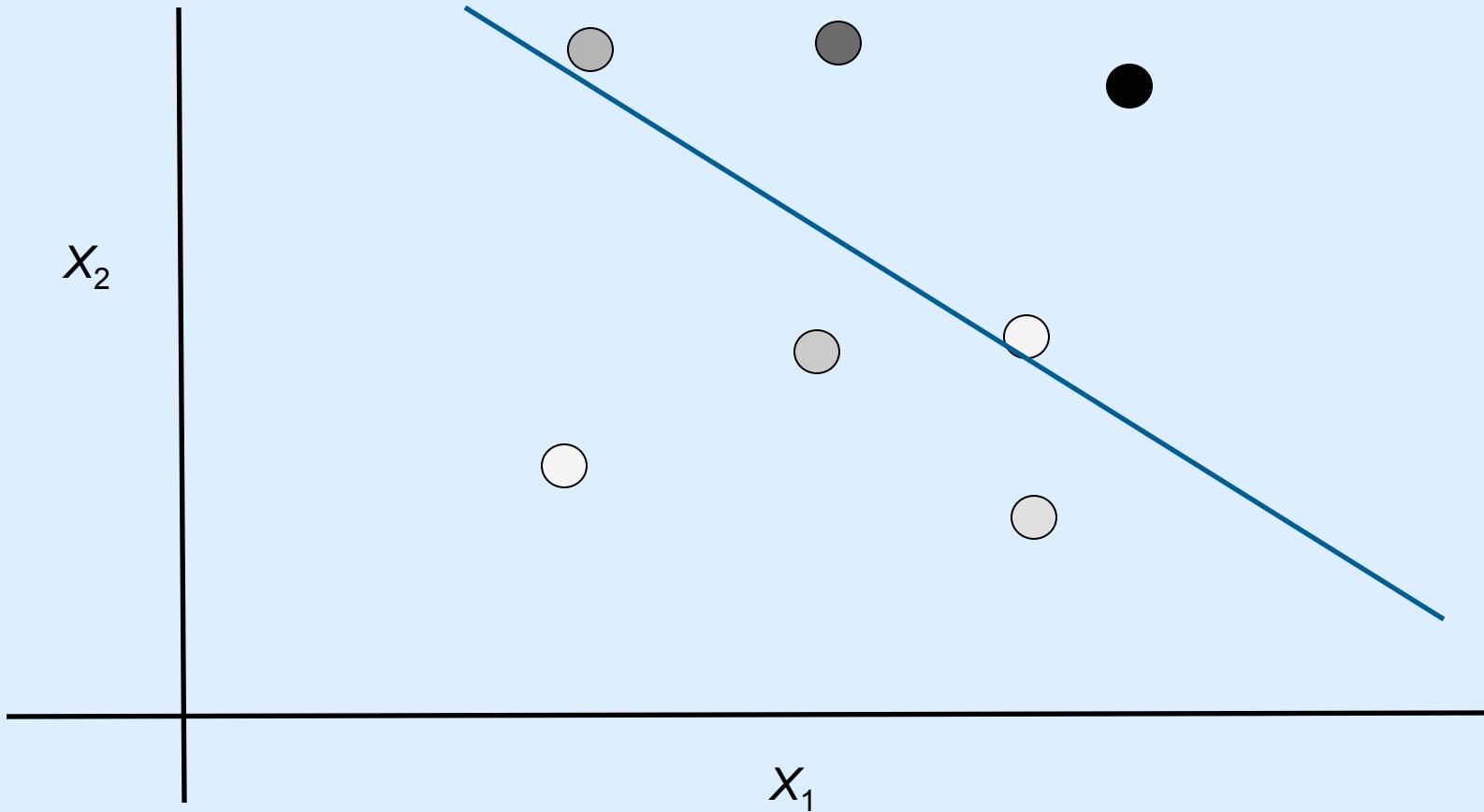


A Classification Problem





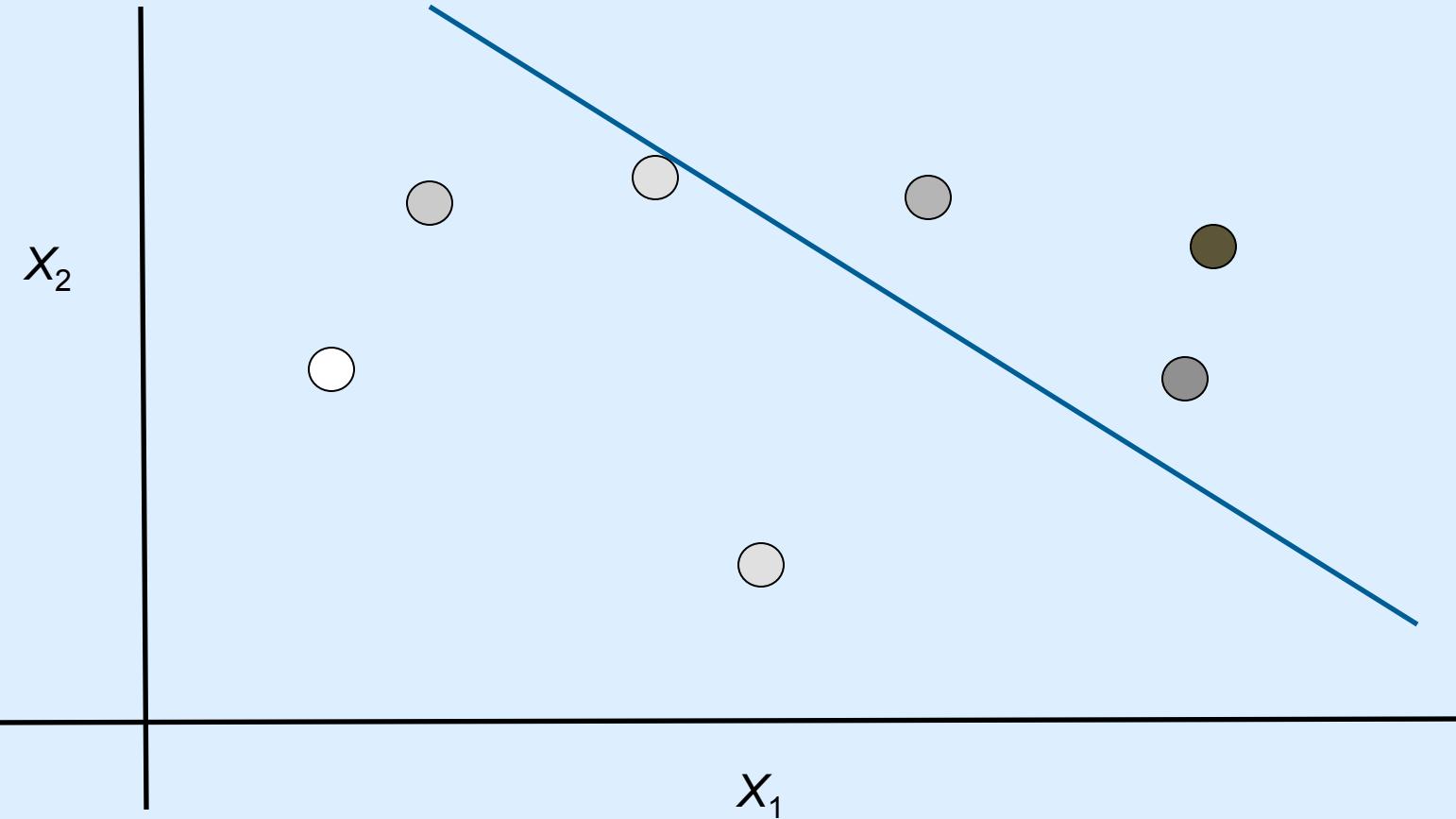
A Classification Problem





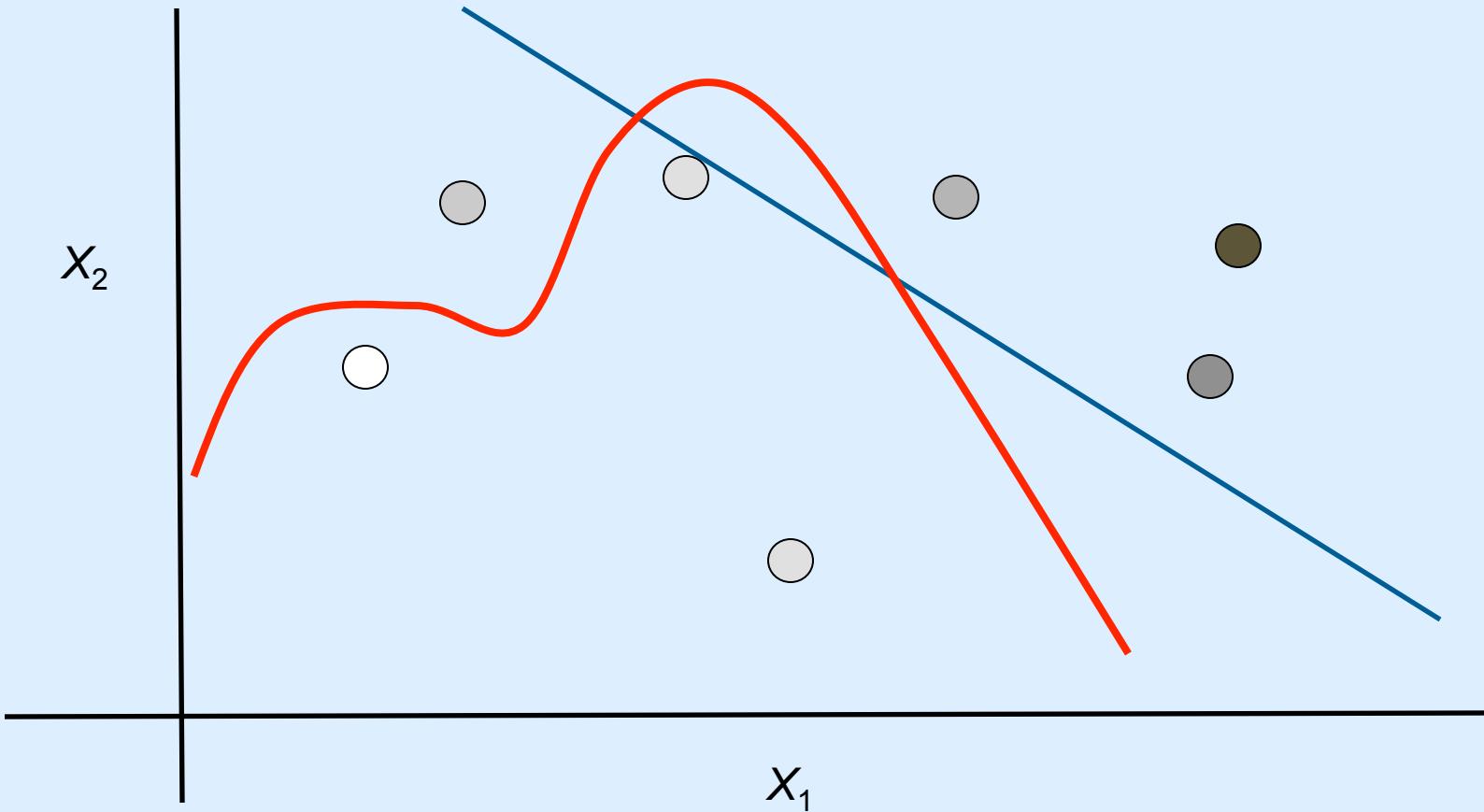
A Classification Problem

Back to Training Set



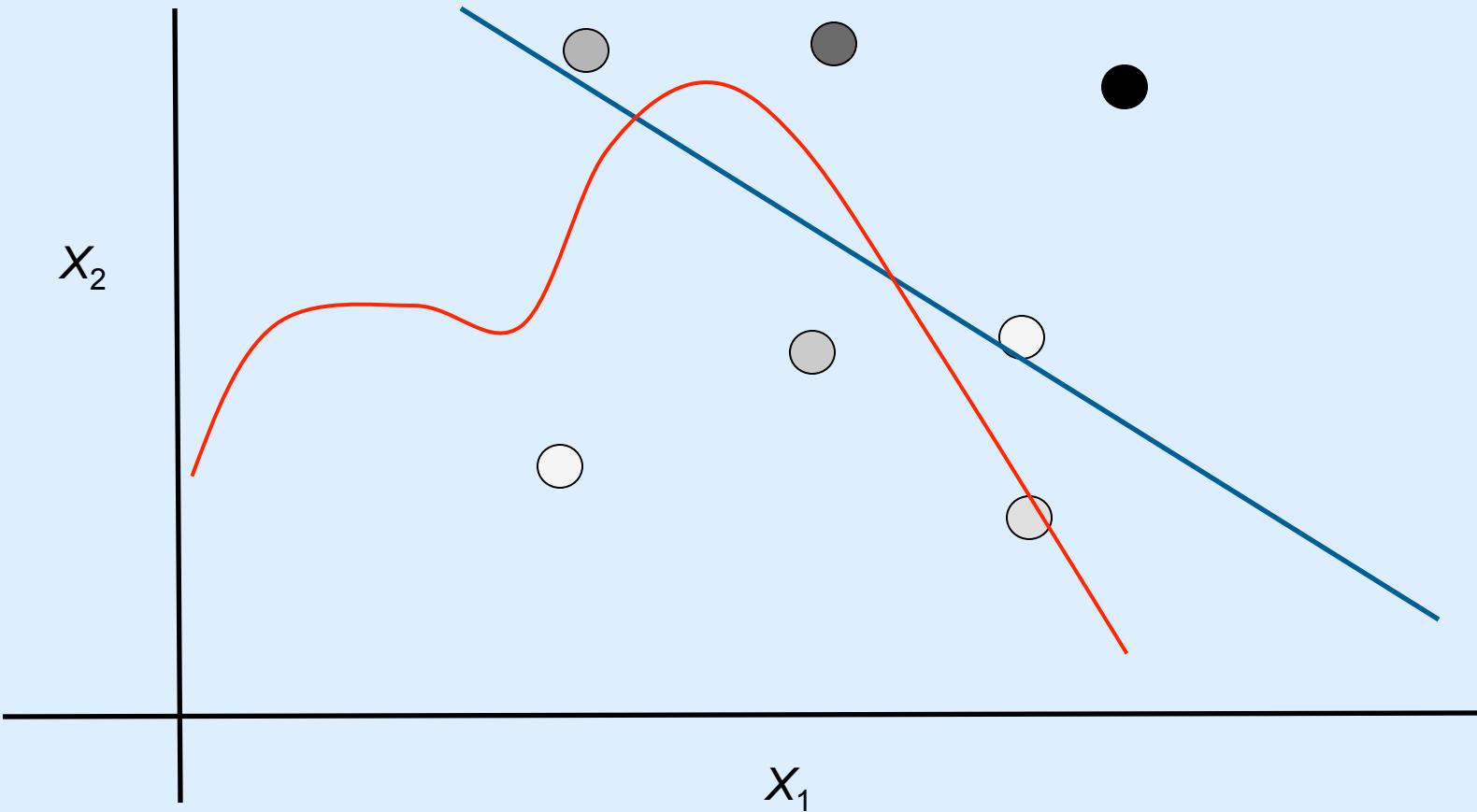


A Classification Problem





The Overtraining Problem





The Overtraining Problem

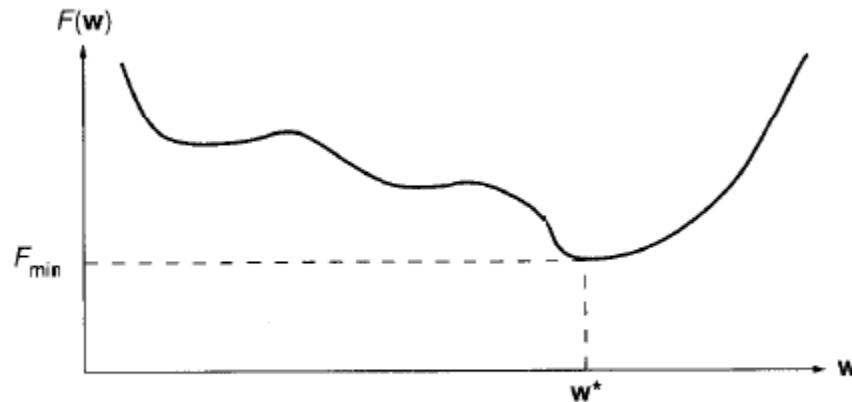


Fig. 5.1. • A typical error surface. A goal of neural network training is to find a network weight vector w^* that minimize the error F .

From *Neurocomputing*
by Hecht-Nielsen



The Overtraining Problem

- Random variations may be part of some underlying process.
- A Regression Line estimates the deterministic part of the process.
- Training attempts to capture this process.
- Training too much may end up including the effects of random variation in the deterministic part of the modeling of a process.

We end up training the network on the randomness instead of the underlying deterministic process!

Training on the errors.



Training Reprise

- During training, we use the error to compute the gradient in steepest descent methods, to guide us towards minimizing the error.
- During testing/evaluation, we use the error to minimize the testing error.
- Must do a lot of experimentation: training, testing, training, testing...
- A lot of computational effort is involved to get it right!



Training Efficiency

- How can we maximize use of our data?
- How do we decide how many times to train and test? How many different partitions to use?
- Efficient and effective training and testing methods must be employed.



Introduction to Neural Networks

**Johns Hopkins University
Engineering for Professionals Program**

605-447/625-438

Dr. Mark Fleischer

Copyright 2014 by Mark Fleischer

Module 7.2: Training Efficiency



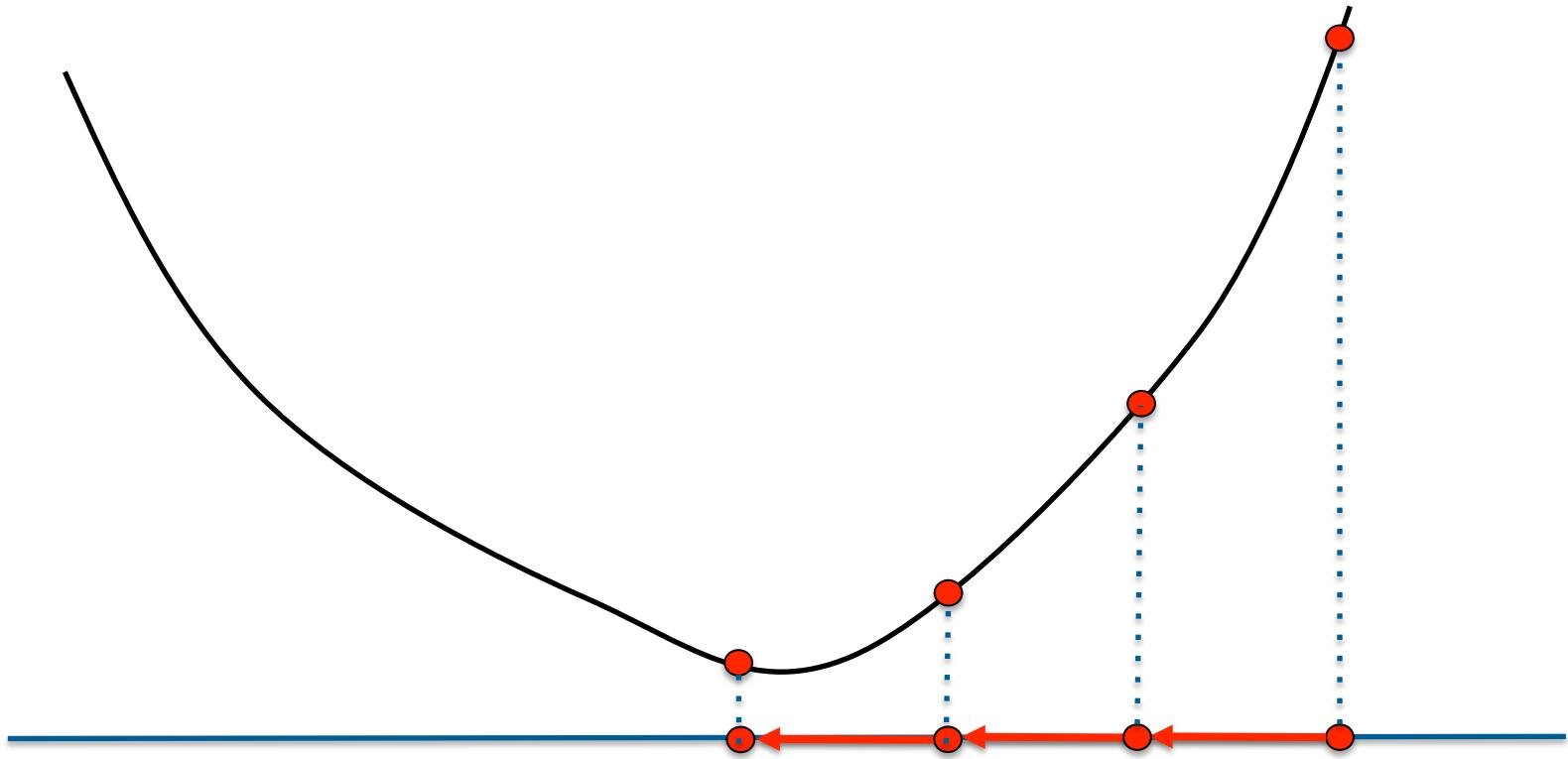
This Sub-Module Covers ...

- We know that in training and testing, there can be a high computational burden.
- Want to make training more efficient.
 - Explore ways for speeding up the FFBP algorithm with a momentum term;
 - Explore some gardening techniques (pruning techniques) to decrease the network size.



Training: Method of Steepest Descent

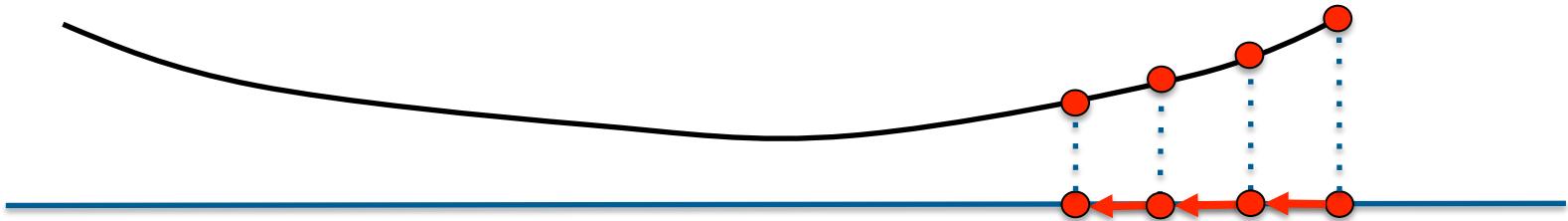
The problem with ‘fixed’ step sizes: $w_{k+1} = w_k - \eta \nabla f(w_k)$





Training: Method of Steepest Descent

The problem with ‘fixed’ step sizes: $w_{k+1} = w_k - \eta \nabla f(w_k)$





Issues with Steepest Descent Methods

- Can take a long time to find a local minimum when the step size η is too small.
- Can miss or overstep where the local minimum is when the step size η is too large and even lead to oscillations or meandering around the minima.



Getting to Minima Faster

- Some approaches can be proven to converge to the minima:
 - Simulated annealing, but there are issues in continuous variable problems.
 - Stochastic approximation methods.
 - Nelder-Mead ...
- The No-Free Lunch Theorem applies!



The Momentum Term

- Strikes a good balance between ease of implementation and computational overhead and speeding up the process.
- Based on using historical information in the step size.

$$\Delta w_i(k) = -\eta \frac{\partial E}{\partial w_i} + \alpha \Delta w_i(k-1)$$



The Momentum Term

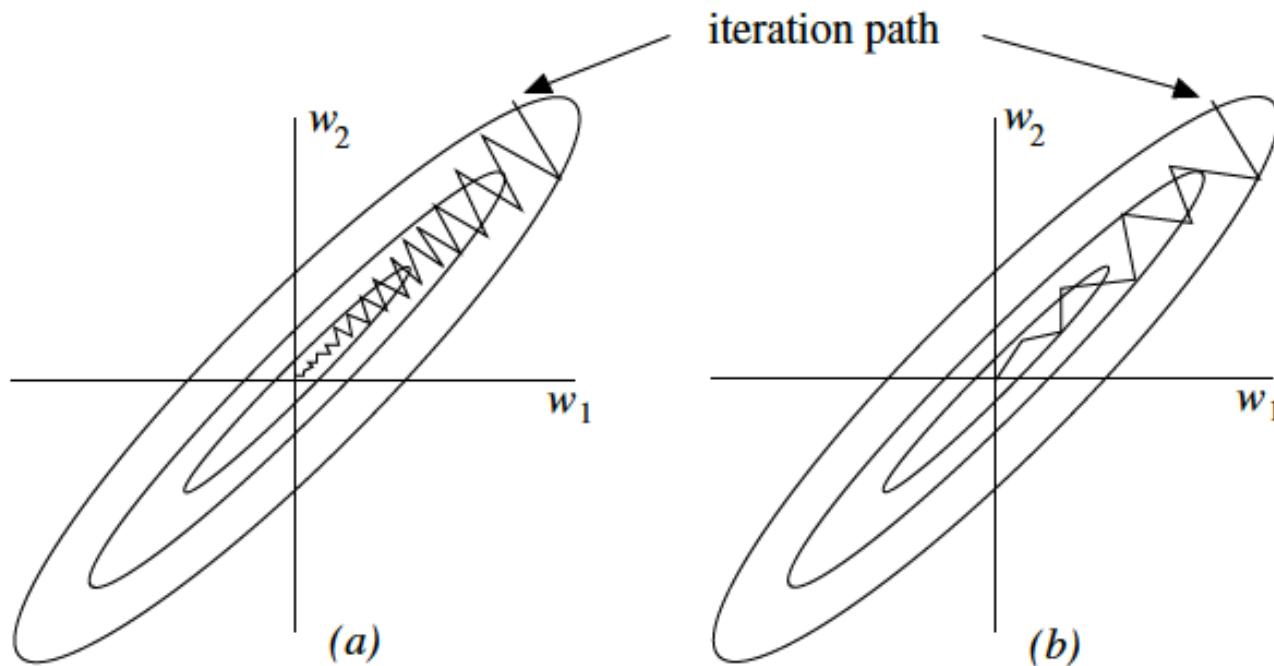


Fig. 8.1. Backpropagation without (a) or with (b) momentum term

From Rojas, p.186



Other Improvement Approaches

- Training and evaluating a network can take a very long time to get it right (we'll see what 'right' means later).
- We have seen how to improve the training algorithm with momentum terms.
- Let's improve training by modifying the network.



Can A Neural Network Model a Polynomial?

- From the manner of training a Neural Network, they can
 - Interpolate data given the training data.
 - Can they model a polynomial?
- How can we mathematically define a polynomial given specific data?
- What does this have to do with training efficiency?



Collocation Method of Polynomials

To define an n -degree polynomial function $P_n(x)$ that goes “through” a specified set of points (x,y) , define

$$P_n(x) = \sum_{i=0}^n f_i L_i(x)$$

where

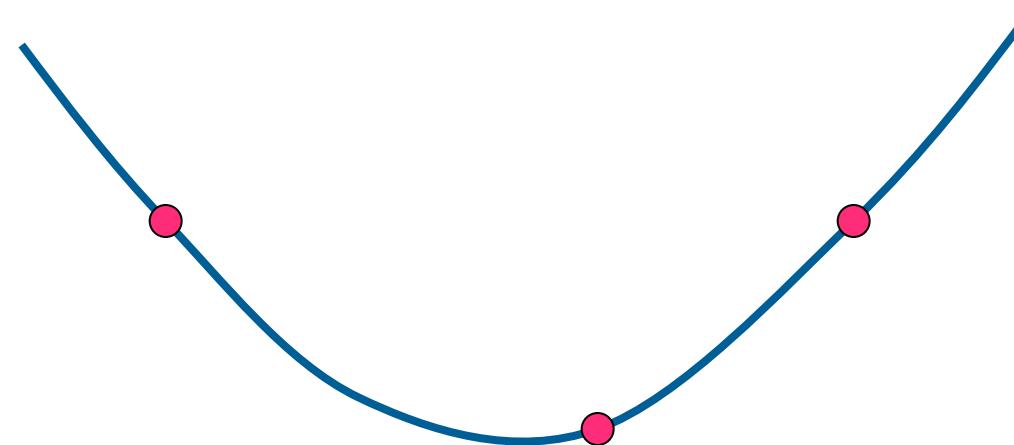
$$L_i(x) = \prod_{\substack{k=0 \\ k \neq i}}^n \frac{(x - x_k)}{(x_i - x_k)}$$



Example

Say we want a polynomial to go through specific points in some space. For three points . . .

$$(x_0, f(x_0)), (x_1, f(x_1)), (x_2, f(x_2))$$





Remembering our function definitions:

$$P_n(x) = \sum_{i=0}^n f_i L_i(x)$$

$$L_i(x) = \prod_{\substack{k=0 \\ k \neq i}}^n \frac{(x - x_k)}{(x_i - x_k)}$$

then

$$P_n(x) = f_0 \left[\frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} \right] + f_1 \left[\frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} \right] + f_2 \left[\frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} \right]$$

$$(x_0, f(x_0)), (x_1, f(x_1)), (x_2, f(x_2))$$



Remembering our function definitions:

$$P_n(x) = \sum_{i=0}^n f_i L_i(x)$$

$$L_i(x) = \prod_{\substack{k=0 \\ k \neq i}}^n \frac{(x - x_k)}{(x_i - x_k)}$$

then

$$P_n(x) = f_0 \left[\frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} \right] + f_1 \left[\frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} \right] + f_2 \left[\frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} \right]$$

1

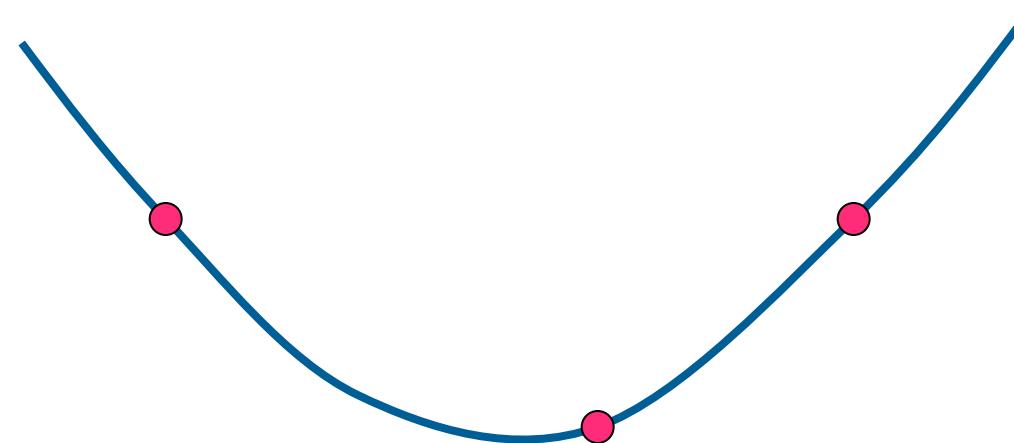
$$(x_0, f(x_0)), (x_1, f(x_1)), (x_2, f(x_2))$$



Example

Say we want a polynomial to go through specific points in some space. For three points . . .

$$(x_0, f(x_0)), (x_1, f(x_1)), (x_2, f(x_2))$$





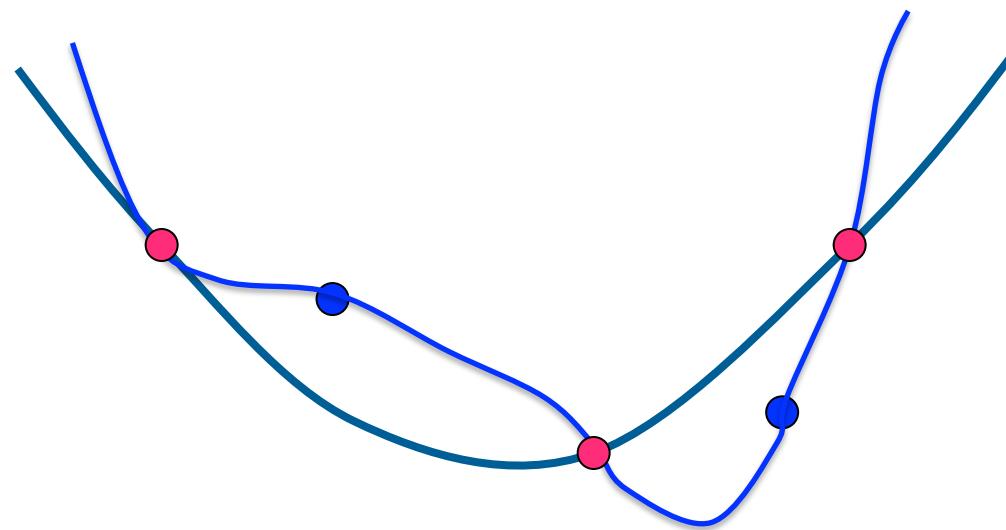
JOHNS HOPKINS

WHITING SCHOOL
of ENGINEERING



Example

Now let's add some "dummy" points.





Remembering our function definitions:

$$P_n(x) = \sum_{i=0}^n f_i L_i(x)$$

$$L_i(x) = \prod_{\substack{k=0 \\ k \neq i}}^n \frac{(x - x_k)}{(x_i - x_k)}$$

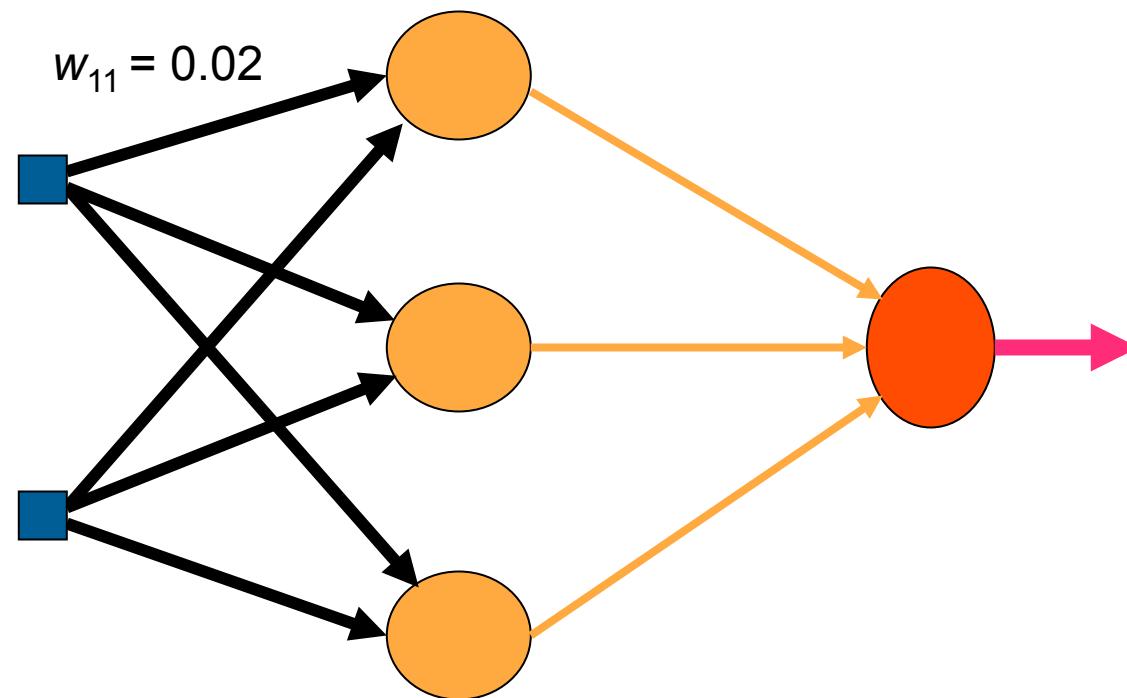
then

$$P_4(x) = f_0 \left[\frac{(x - x_1)(x - x_2)(x - x_3)(x - x_4)}{(x_0 - x_1)(x_0 - x_2)(x_0 - x_3)(x_0 - x_4)} \right] + f_1 \left[\frac{(x - x_0)(x - x_2)(x - x_3)(x - x_4)}{(x_1 - x_0)(x_1 - x_2)(x_1 - x_3)(x_1 - x_4)} \right] + f_2 \left[\frac{(x - x_0)(x - x_1)(x - x_3)(x - x_4)}{(x_2 - x_0)(x_2 - x_1)(x_2 - x_3)(x_2 - x_4)} \right] + f_3 \left[\frac{(x - x_0)(x - x_1)(x - x_2)(x - x_4)}{(x_3 - x_0)(x_3 - x_1)(x_3 - x_2)(x_3 - x_4)} \right] + f_4 \left[\frac{(x - x_0)(x - x_1)(x - x_2)(x - x_3)}{(x_4 - x_0)(x_4 - x_1)(x_4 - x_2)(x_4 - x_3)} \right]$$

Well, you get the idea... complications, complexity, etc.

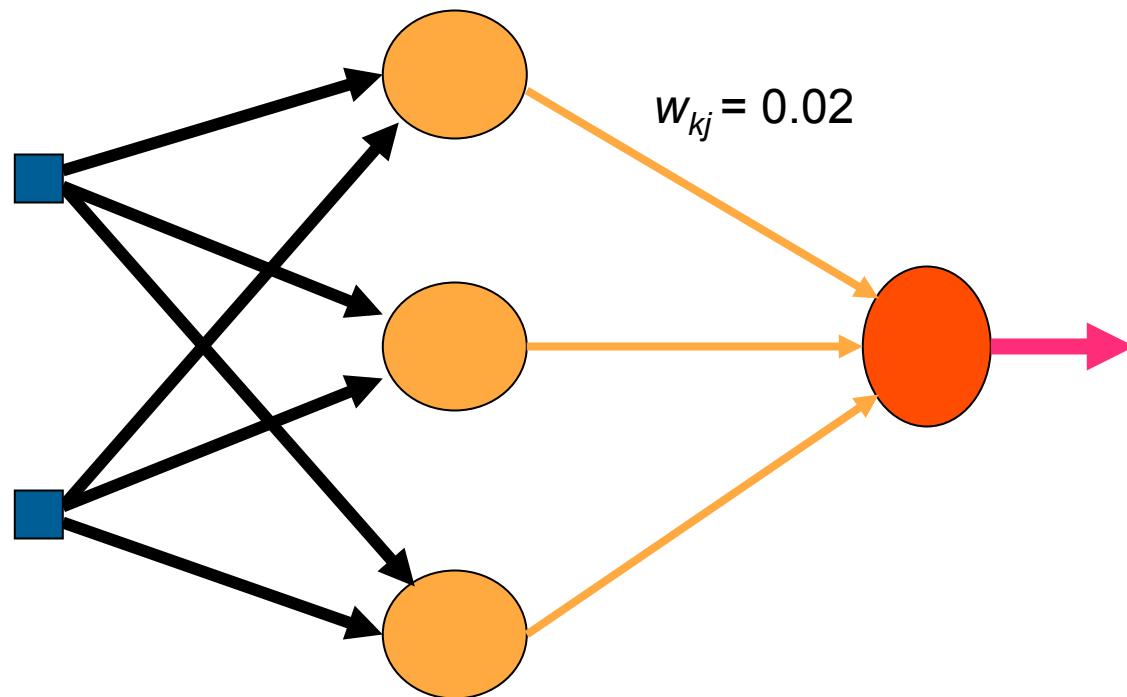


Pruning The Network





Pruning The Network





Summary – 7.2

- Described using the momentum term to speed-up gradient descent optimization (i.e., FFBP).
- Collocation Polynomial and how to define a polynomial function that passes through a specified set of points.
- Pruning the network to eliminate ‘weak’ links and/or nodes that do not carry much weight.



Introduction to Neural Networks

**Johns Hopkins University
Engineering for Professionals Program**

605-447/625-438

Dr. Mark Fleischer

Copyright 2013 by Mark Fleischer

Module 7.3: Training with Multiple I/O Pairs



This Sub-Module Covers ...

- What a typical training scenario looks like when there are multiple input/output pairs.
 - “online” or “incremental” training
 - “batch” training
- Implications and why they are different.

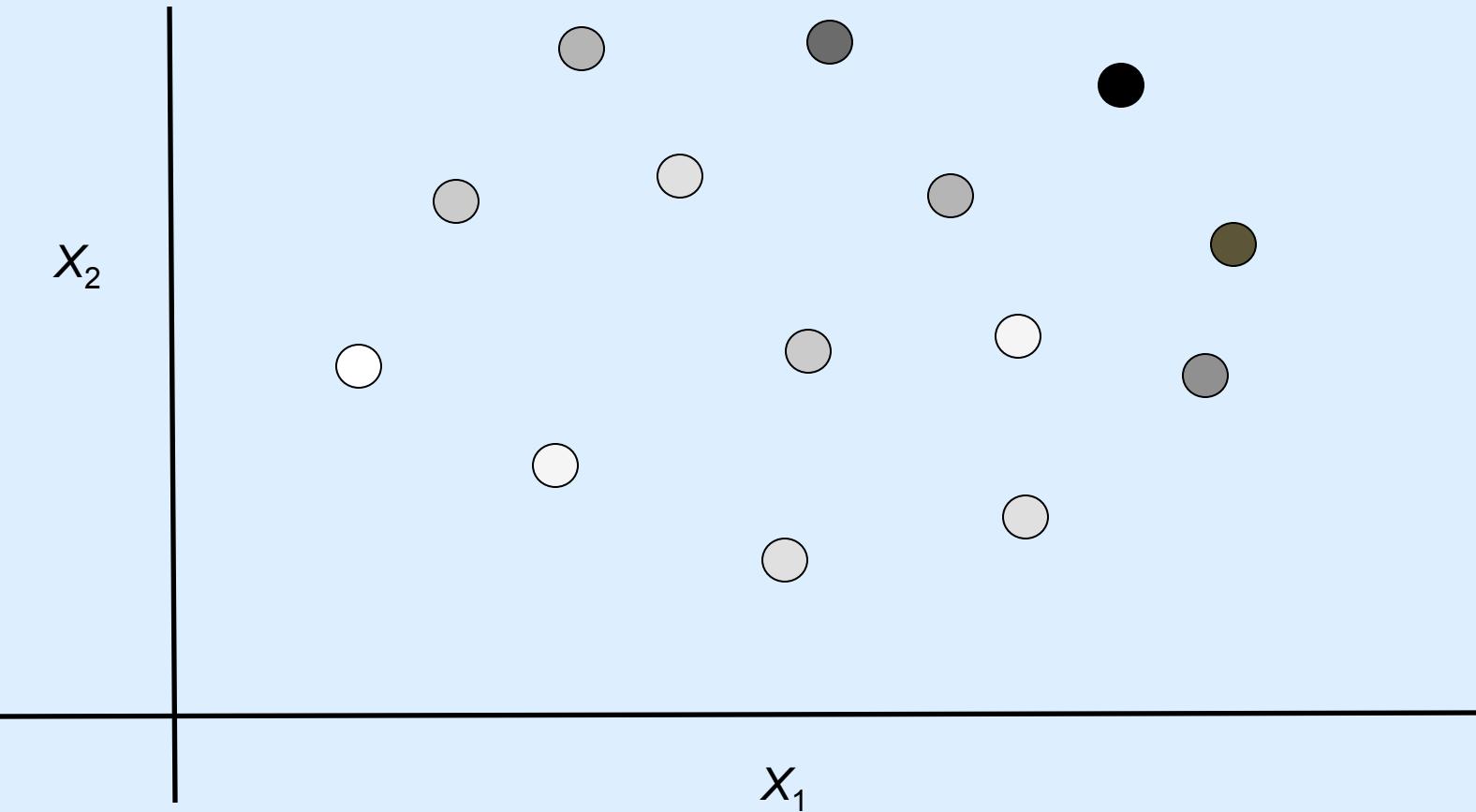


Training with Multiple Input/Output Pairs

- We've seen we can decrease errors with a single I/O pair with the Perceptron.
- Can do the same with multilayer networks.
- Still, with FFBP algorithm, training can get stuck in local optima---with one I/O pair we can get error as low as desired.
- Let's examine a type of problem with many I/O pairs.

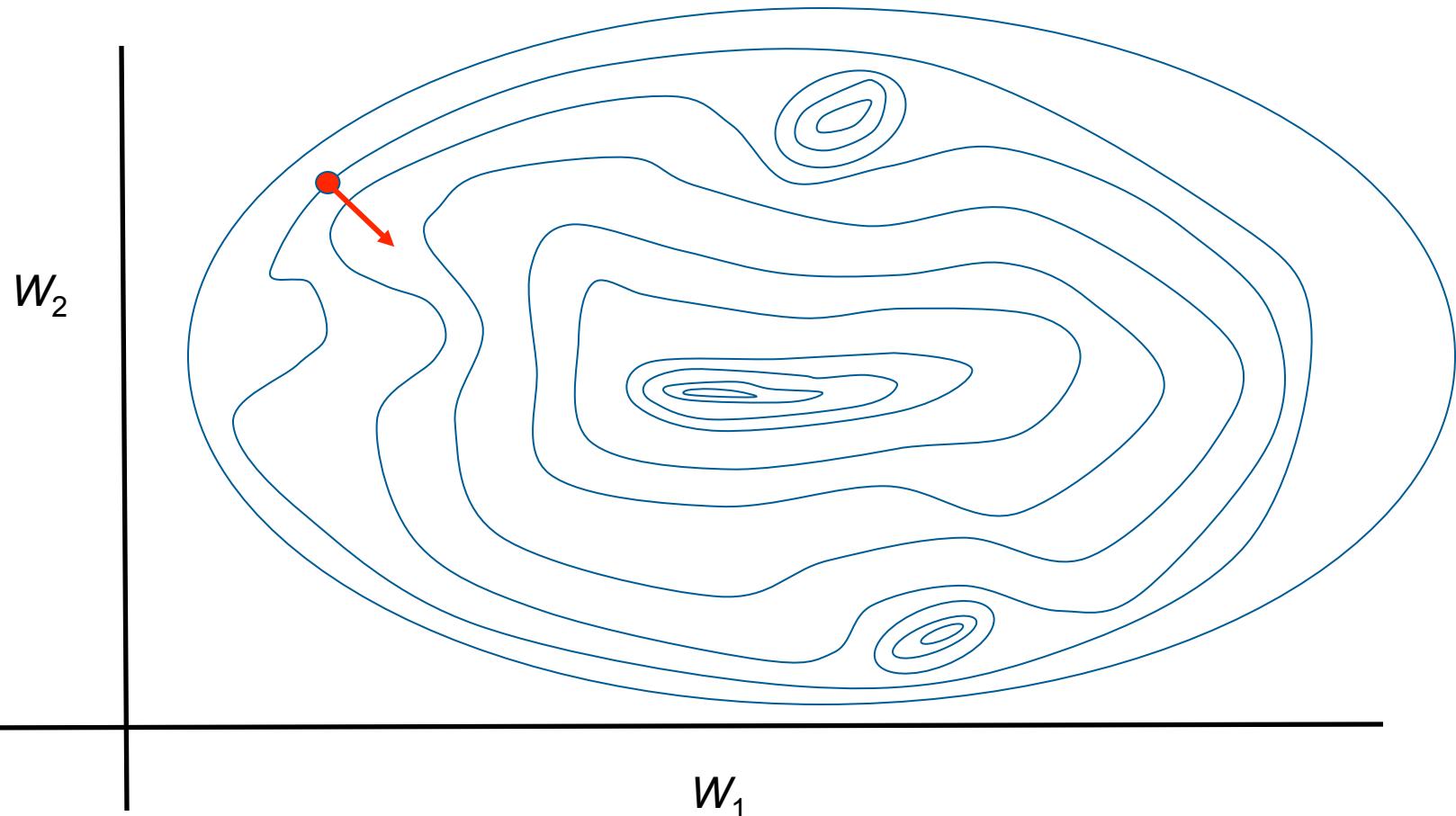


A Classification Problem



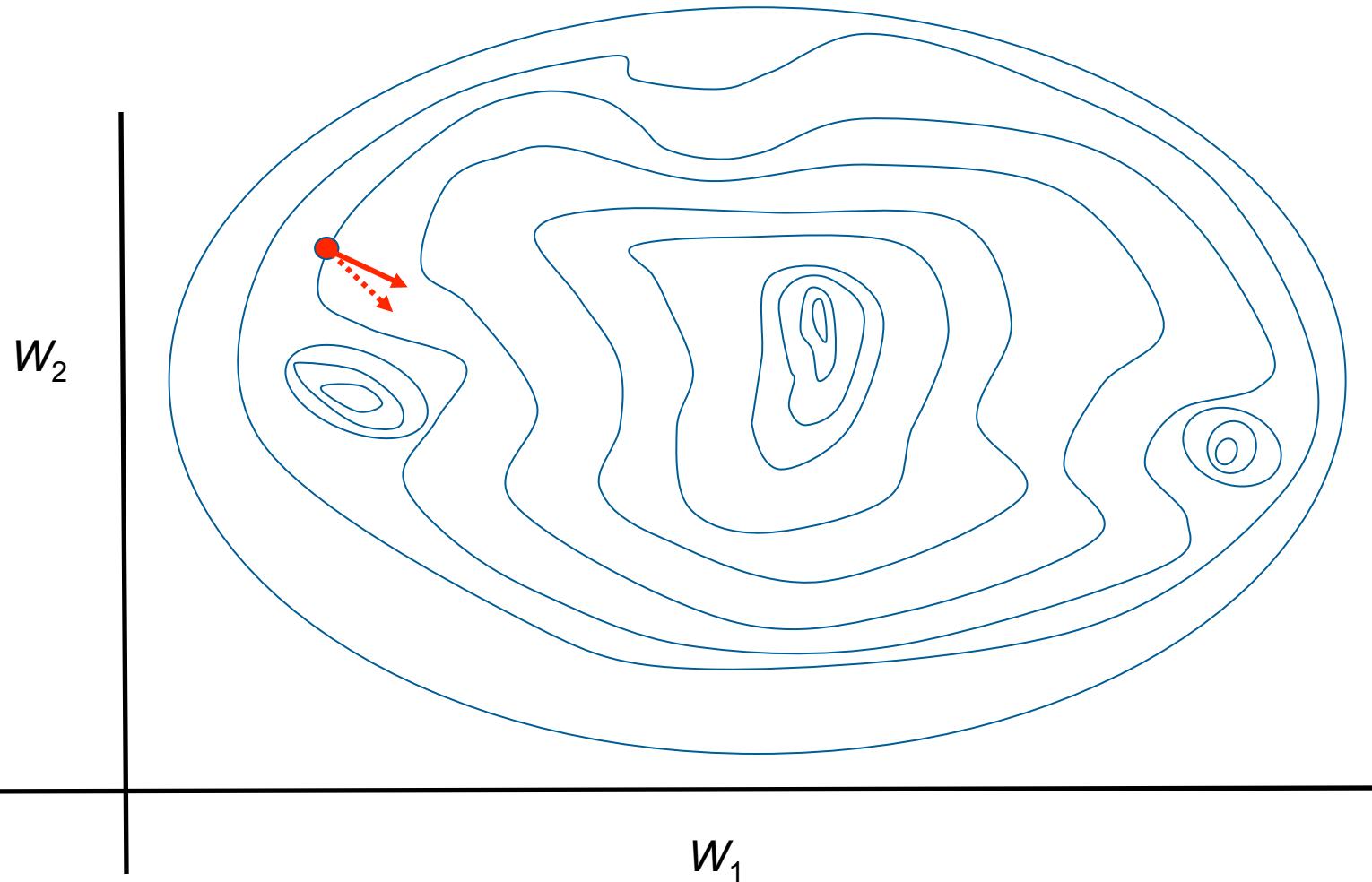


Present 1st Input/Output Pair



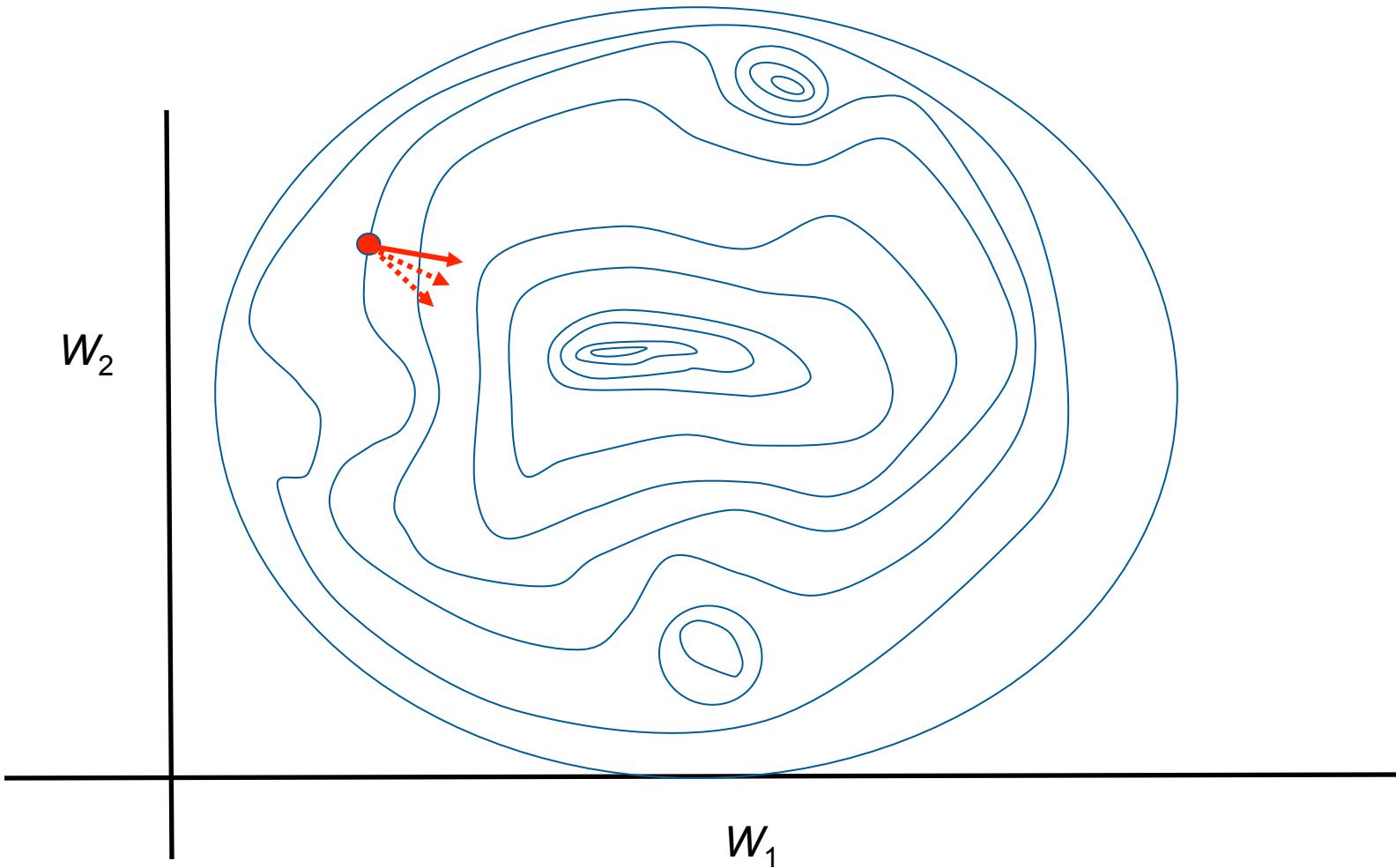


Present 2nd Input/Output Pair



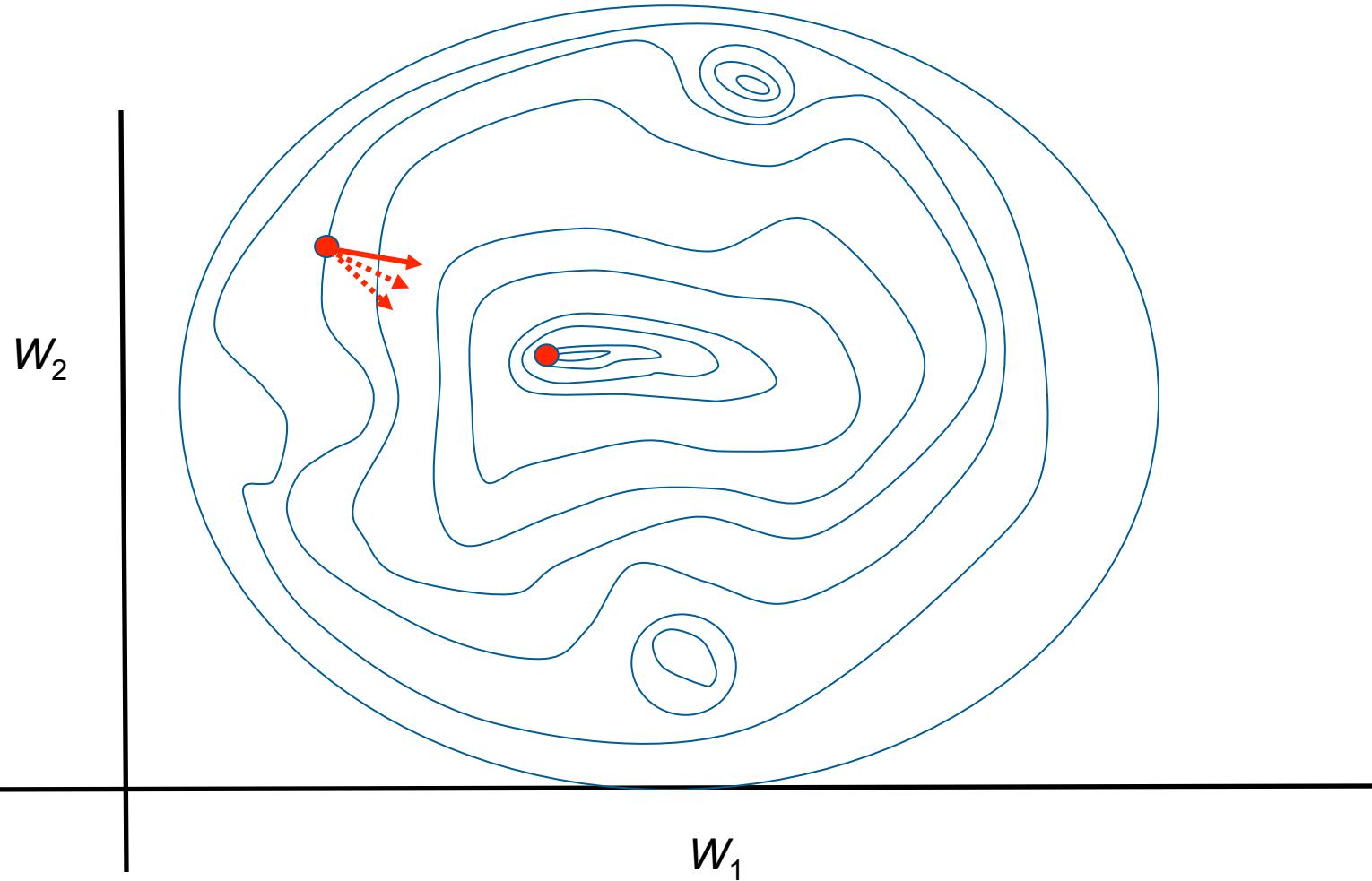


Present 3rd Input/Output Pair





Updating Weights





Batch Training

Training cycle ($c = 1$ to C) {

$\Delta w == 0$ // Initialize weight update

 Present I/O pair ($p = 1$ to P) {

$\Delta w +=$ Calculate Gradient estimate();

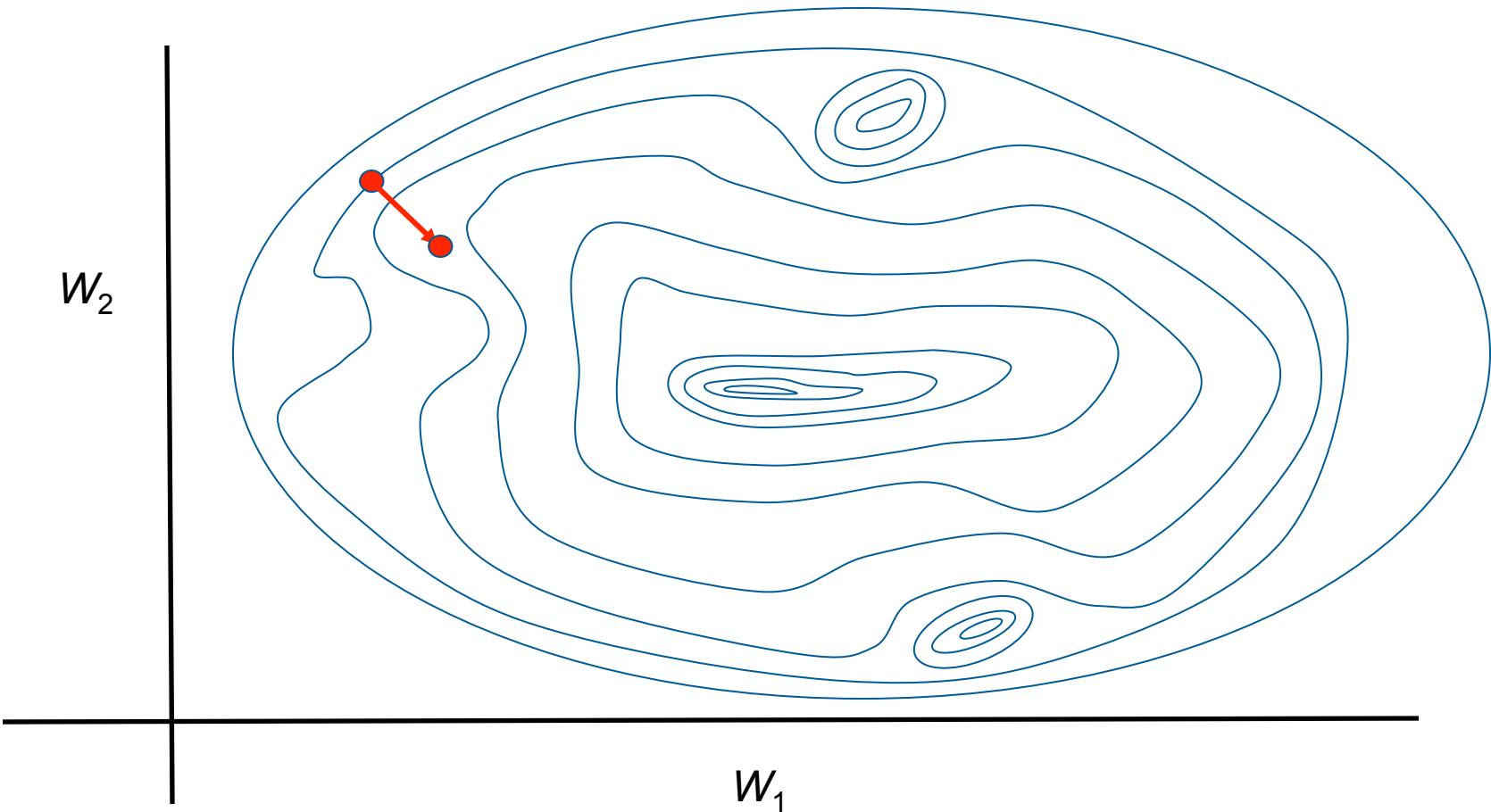
 } // Present next I/O pair

 Update weight $w = w + \Delta w$

} // Next training cycle

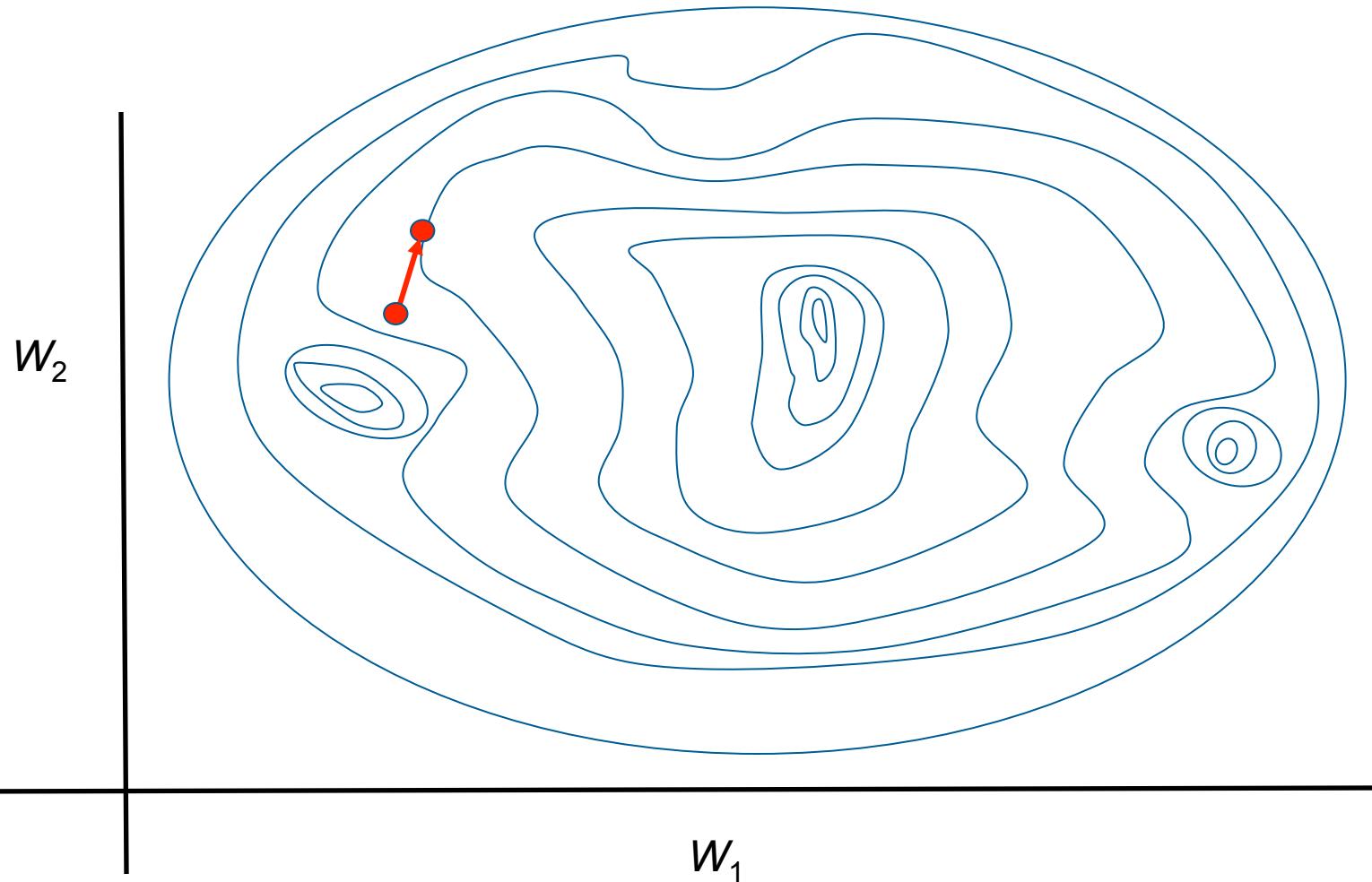


Present 1st Input/Output Pair



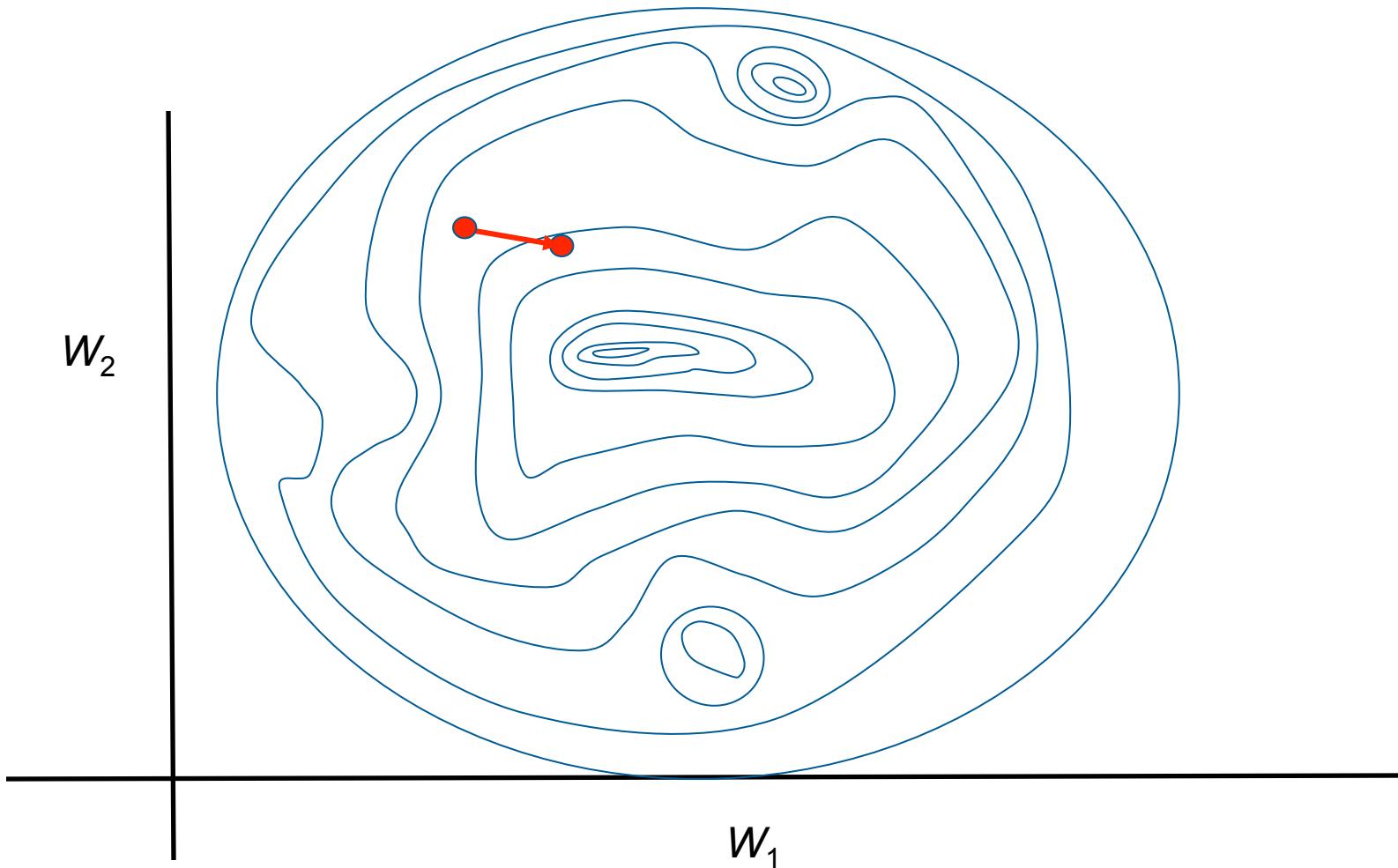


Present 2nd Input/Output Pair



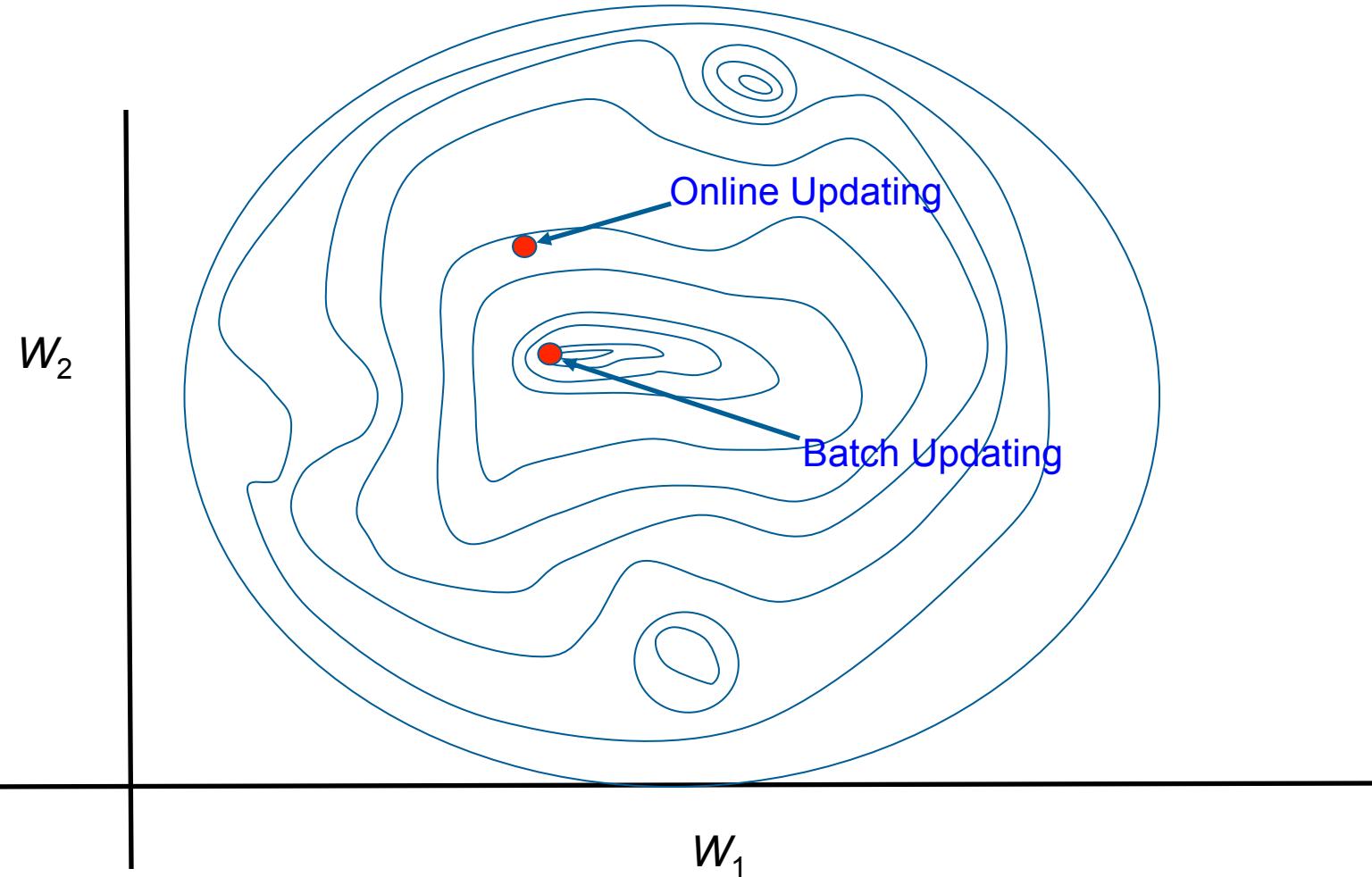


Present 3rd Input/Output Pair





Comparing Updated Weights





Online (Incremental) Training

```
Training cycle (c = 1 to C){  
    Present I/O pair (p = 1 to P) {  
        Δw = Calculate Gradient estimate();  
        Update weight w = w + Δw  
    } // Present next I/O pair  
} // Next training cycle
```



Batch vs. Online Training

Batch

Pros:

- Closer match to theoretical (true) gradient.
- Lower residual error.
- Permits other, useful modifications of FFBP to be used.

Online

Pros:

- Faster, earlier updates (but remember, each update changes the error function for subsequent I/O pairs).
- Useful if no ‘fixed’ training set exists or when the data are ‘nonstationary’.
- Creates effective/useful ‘noise’ that can overcome local minima.



Batch vs. Online Training

Batch

Cons:

- Requires some memory overhead.
- Can require more computation effort to get to comparable weights than online (requires all I/O pairs before an update occurs)

Online

Cons:

- Each update alters the error function landscape --- not the true gradient descent vector for all I/O pairs—perturbs the learning process.
- Sometimes noise isn't a good thing.



The Best of Both Worlds

- Use ‘online’ training for ‘initial training’ with a first part of the training data
 - Get a good, fast start with some updated weights.
- Then as we get closer to minimal error, resort to batch training
 - More carefully and accurate zero-in to true minimal error.



Introduction to Neural Networks

**Johns Hopkins University
Engineering for Professionals Program**

605-447/625-438

Dr. Mark Fleischer

Copyright 2014 by Mark Fleischer

Module 7.4: Receiver Operating Characteristics



This Sub-Module Covers ...

- Performance evaluation.
- Proper training does not merely involve minimizing the error function using the training data.
- Must have some appropriate way of gauging how well the network performs using data it hasn't seen during the training phase.

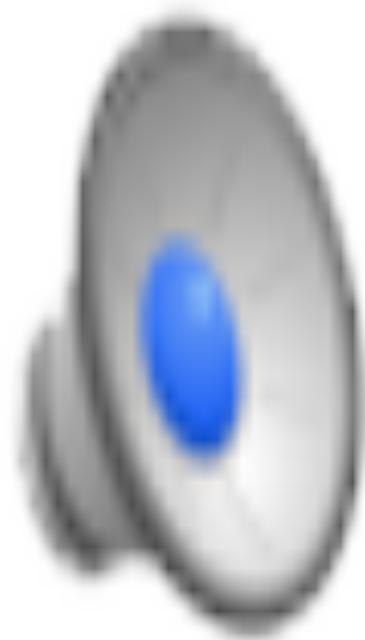


Let's Examine Classification Problems

- How well does the network correctly classify an input it has not previously seen?
- Simplest approach is to consider two classes
 - Corresponds nicely to mathematical logic,
 - Corresponds nicely to computer logic,
 - Some action is good/bad, or true/false
- How do we come up with reasonable, mathematically sound ways of evaluating performance in such a context?
- Consider the early days of radar development in WWII.
- Receiver Operating Characteristics.
- Basis is that everyone has their own particular vexations that set them ‘off’!



Mandrake, those missiles are coming. Red Alert!



or are they?.....



Our friend Gen. Ripper ...

- has an apparent ‘sensitivity’ to communist infiltration, indoctrination, subversion and conspiracies.
 - This sensitivity led him to trigger and order
-



Receiver Operating Characteristics

- Partition a set into two mutually exclusive subsets:
 - one subset contains entities with a characteristic C and the other subset contains entities that do not have characteristic C.
 - Sets could be a “real target” and “not a target”; or “signal” and “noise”; “Wow” and “no extraterrestrial signal”; “heart disease” and “absence of heart disease”, etc.
...
- A detector to determine the “detected set” and to indicate which set a detected “signal” belongs to—define set D .
 - The detector is not perfect---won’t give the correct information all the time.
 - i.e., sometimes it’ll ‘detect’ things that you don’t want it to ‘detect’.
 - Other times, it’ll detect something (send a signal) when it isn’t truly warranted.
- Partition D into two mutually exclusive subsets: Entities that are detected and entities that are not detected.



Assessing the Most Basic Classification

- Either an input produces the desired response, or it does not.

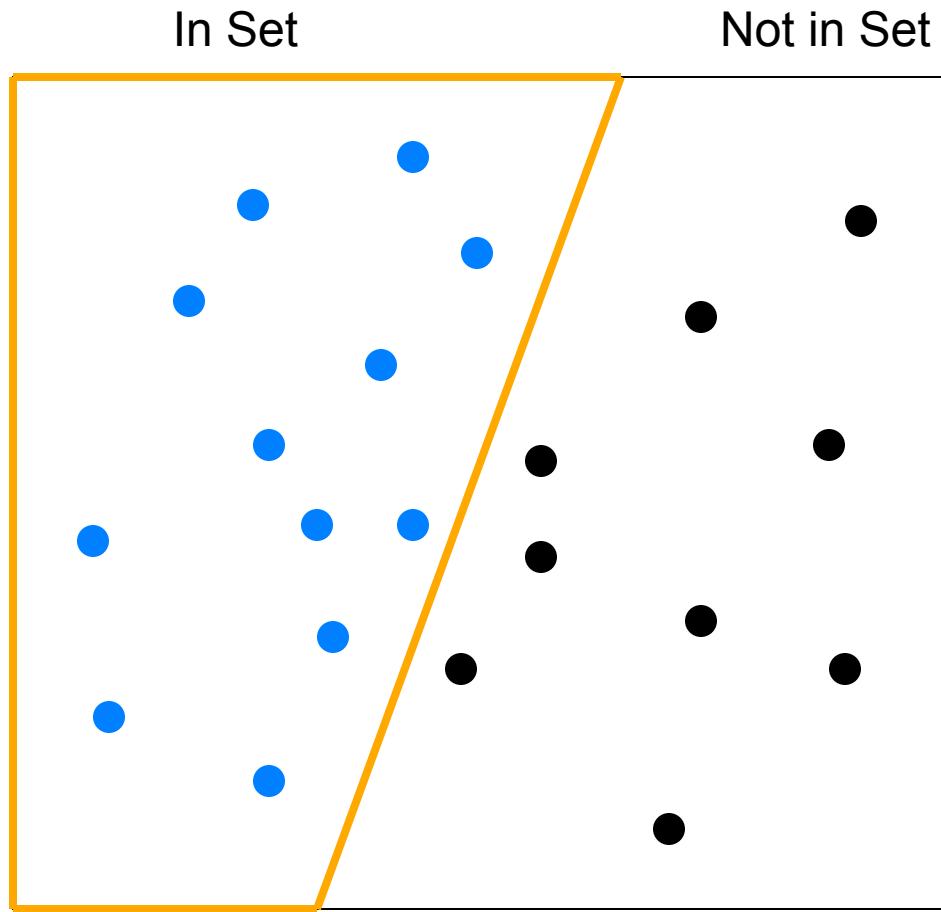
Duh!

Say we want to “detect” only certain types of ‘signals’.

Don’t want to detect signals that are not of this ‘type’.



Receiver Operating Characteristics





The Purpose of Detection

- We want to design our detector system (or medical test or neural network) to function such that a ‘detected’ signal means that which caused it to be detected also has the ‘characteristic’ .
- We want detection to be strongly associated with a specified characteristic.



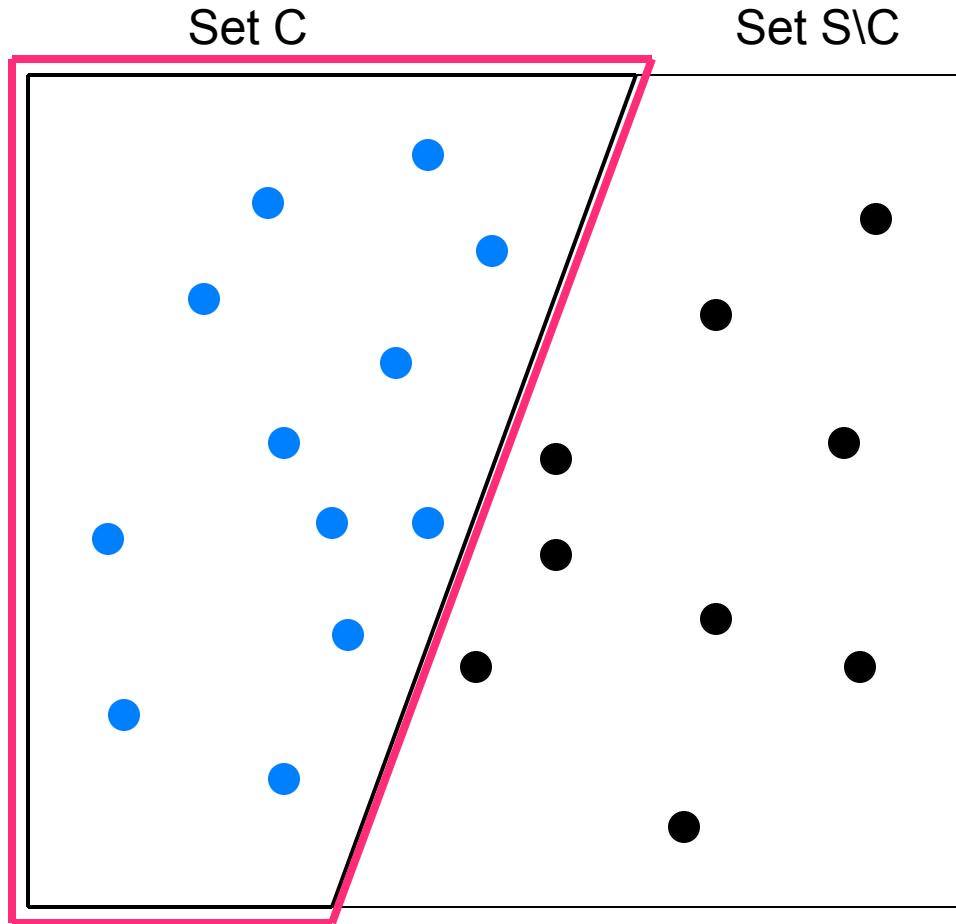
Set Definitions

We want to detect signals from set C, i.e., want to detect whether an entity has the characteristic.

- Define the following sets.
 - $S_1 = \{x_i \mid x \in D \wedge x \in C\}$
 - $S_2 = \{x_i \mid x \in C\}$
 - $S_3 = \{x_i \mid x \notin D \wedge x \notin C\}$
 - $S_4 = \{x_i \mid x \notin C\}$

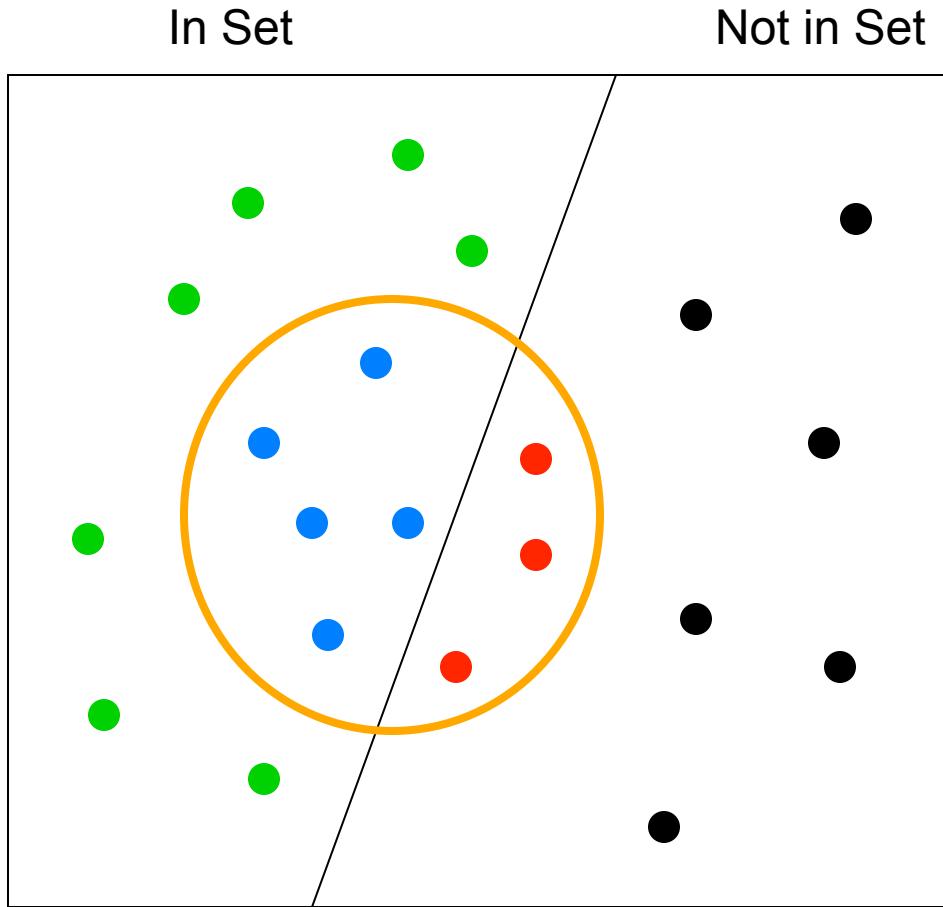


Ideally, we want this





In Reality, we usually have this



Everything in the yellow circle is
'detected'; i.e.,
produces a **positive**
test result



Receiver Operating Characteristics

- Define the following sets.

- $S_{TP} = \{x_i \mid x \in D \wedge x \in C\}$



- $S_{TP+FN} = \{x_i \mid x \in C\}$



- $S_{TN} = \{x_i \mid x \notin D \wedge x \notin C\}$



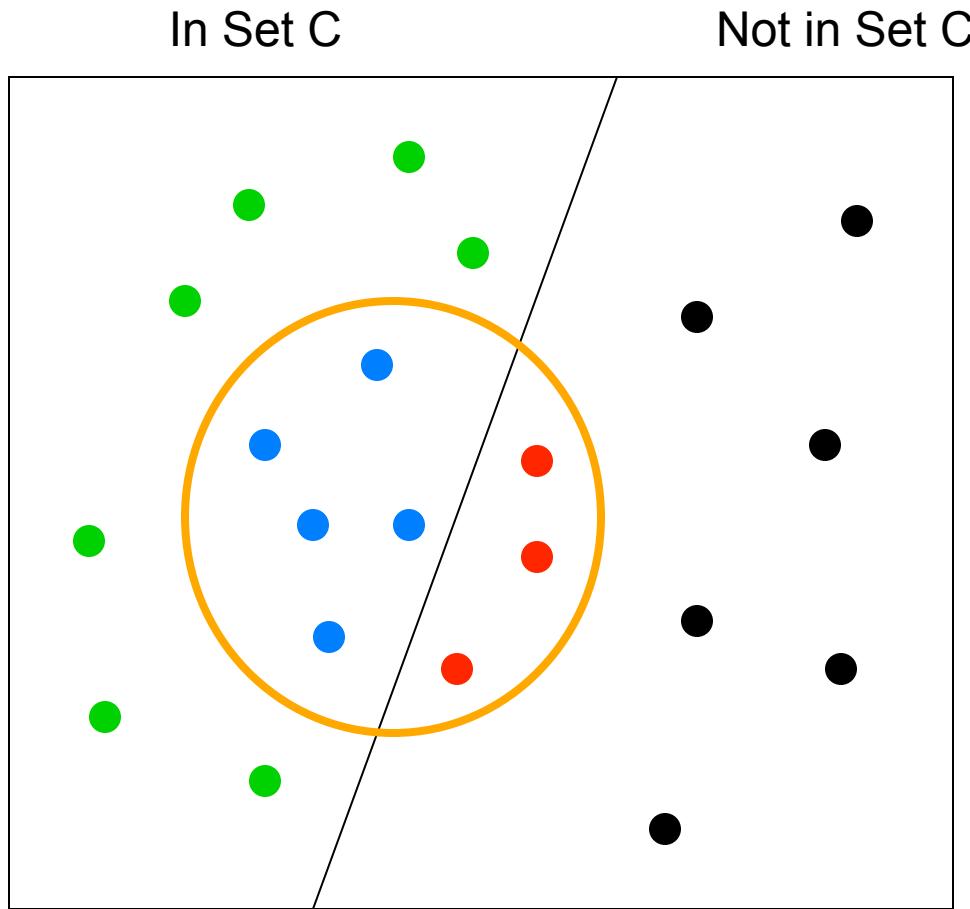
- $S_{TN+FP} = \{x_i \mid x \notin C\}$



Note, any element $x_i \notin D$ is an element which tests negative.
Any element $x_i \in D$ is an element which tests positive.



Receiver Operating Characteristics



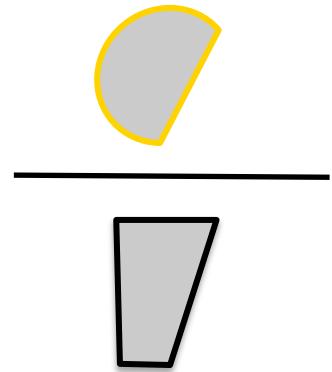
Blue is TP and Red is FP. Green is FN. Black is TN



Some Quantities we are interested in:

$$\Pr\{x \in D | x \in C\} = \frac{\Pr\{x \in D \cap C\}}{\Pr\{x \in C\}}$$

$$\Pr\{D | C\} = \frac{\Pr\{D \cap C\}}{\Pr\{C\}}$$

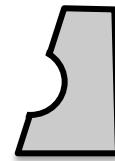


Sensitivity



Some Quantities we are interested in:

$$\Pr\{x \notin D | x \notin C\} = \Pr\{\bar{D} | \bar{C}\} = \frac{\Pr\{\bar{D} \cap \bar{C}\}}{\Pr\{\bar{C}\}}$$





Specificity



Receiver Operating Characteristics

Recall the following sets.

- $S_{TP} = \{x_i \mid x \in D \wedge x \in C\}$
- $S_{TP+FN} = \{x_i \mid x \in C\}$
- $S_{TN} = \{x_i \mid x \notin D \wedge x \notin C\}$
- $S_{TN+FP} = \{x_i \mid x \notin C\}$

$$\text{Sensitivity} = \frac{TP}{TP + FN}$$

$$\text{Specificity} = \frac{TN}{TN + FP}$$



The Terms

Sensitivity: The higher this is, the greater the likelihood that if an object is in the characteristic set, it will result in a positive test result---*i.e.*, it will be detected hence the term.



The Terms

Specificity: A measure of the probability that if an object is *not* in the characteristic set then it will *not* be detected. Thus, the detection is specific to the object having the characteristic.



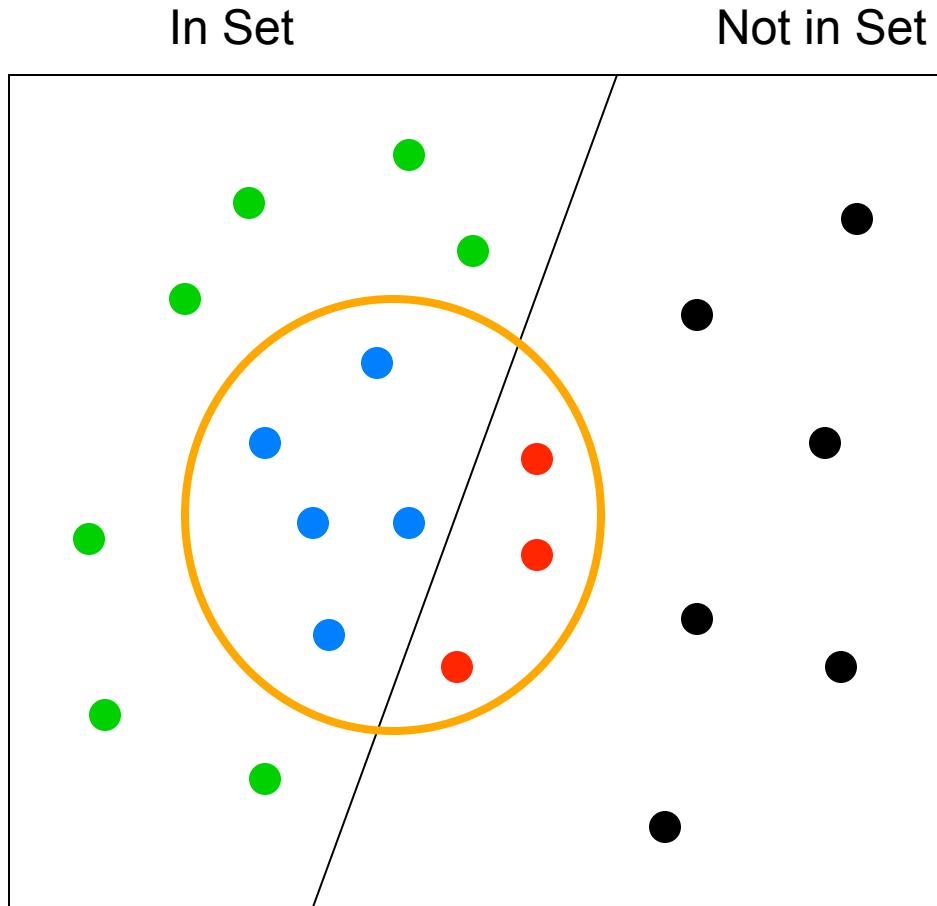
Matrix Form

	In Set	Not in Set
Test Positive	##	##
Test Negative	##	##

Confusion Matrix



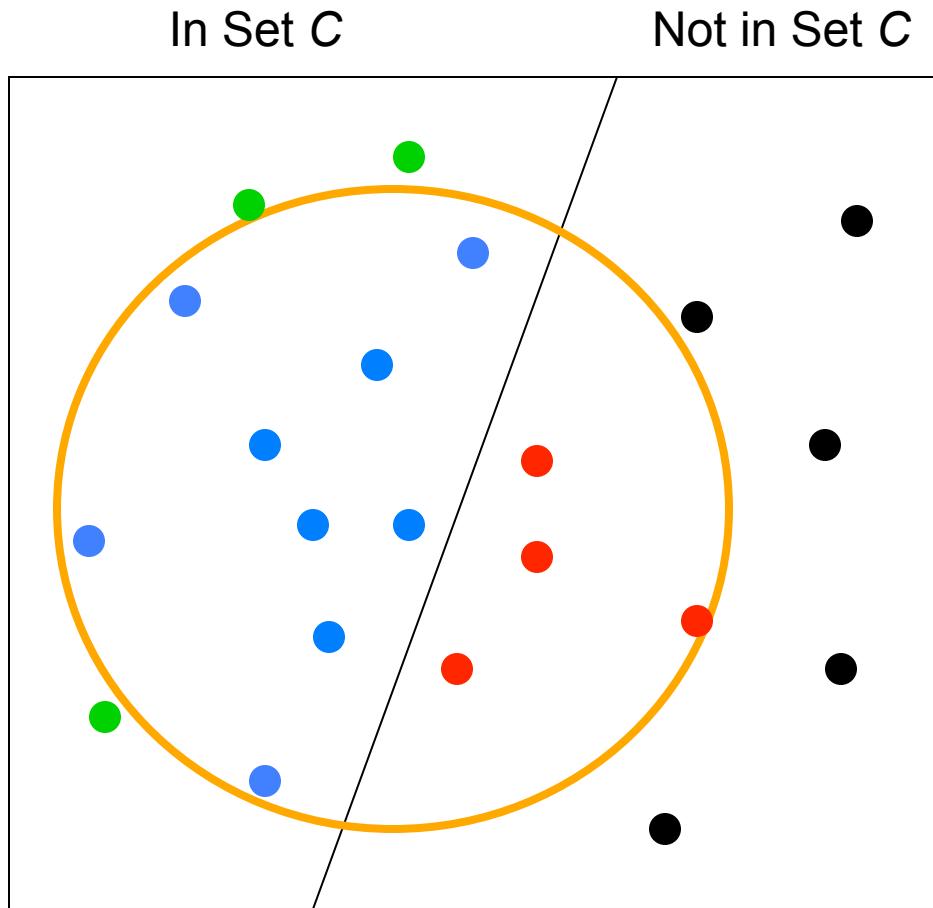
Receiver Operating Characteristics



Blue is TP and Red is FP. Green is FN. Black is TN



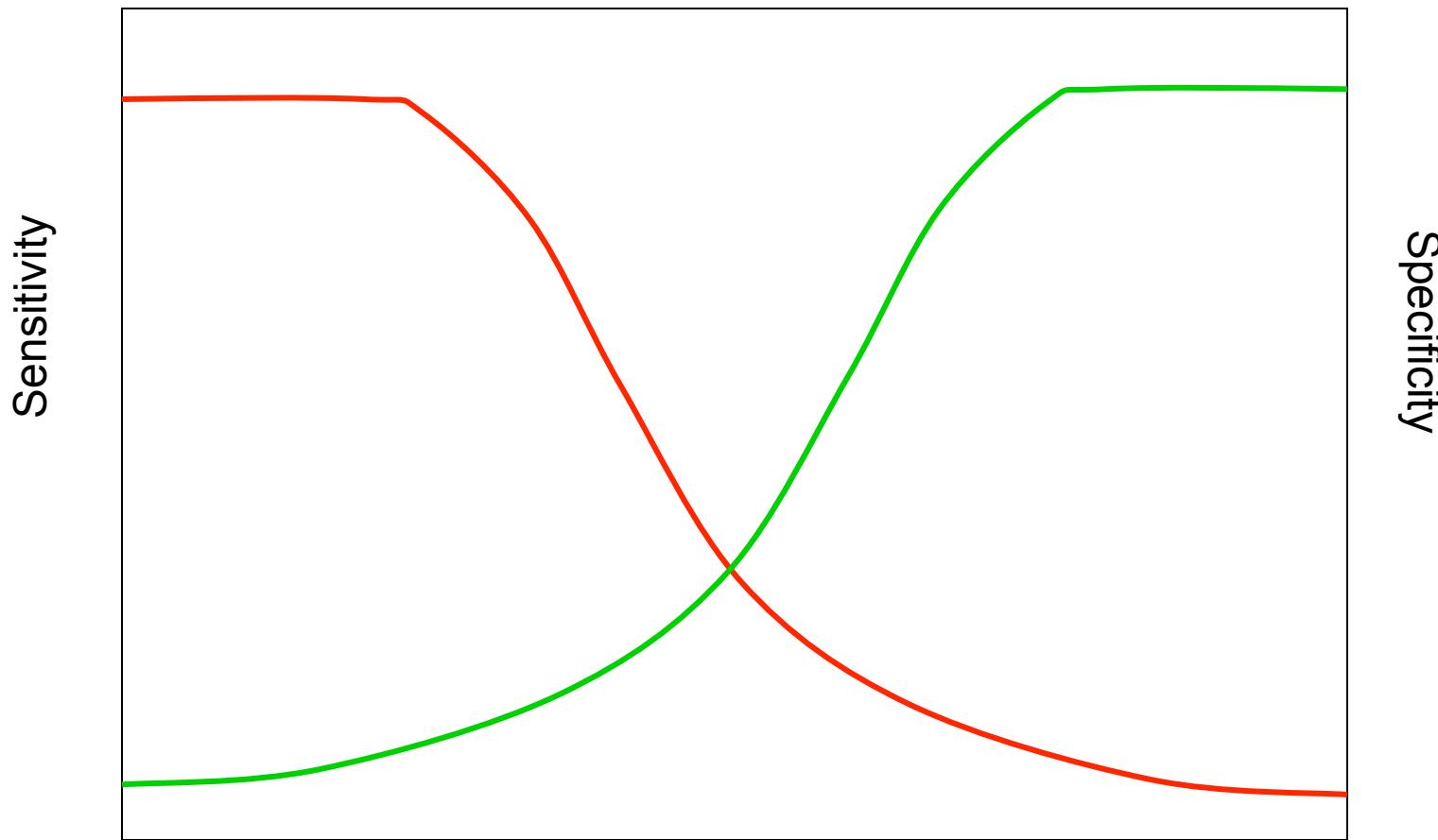
Receiver Operating Characteristics



Blue is TP and Red is FP. Green is FN. Black is TN



Characteristic Curves





Other Useful Quantities

$$\Pr\{x \in C | x \in D\} = \Pr\{C | D\} = \frac{\Pr\{C \cap D\}}{\Pr\{D\}}$$

Positive Predictive Value---PPV

$$\Pr\{x \notin C | x \notin D\} = \Pr\{\bar{C} | \bar{D}\} = \frac{\Pr\{\bar{C} \cap \bar{D}\}}{\Pr\{\bar{D}\}}$$

Negative Predictive Value---NPV



Summary

- ROCs are useful ways of assessing performance of neural networks especially in the context of the accuracy of classification-type applications.
- Can be used with classic MSEs to assess neural network training and performance.



Introduction to Neural Networks

**Johns Hopkins University
Engineering for Professionals Program**

605-447/625-438

Dr. Mark Fleischer

Copyright 2014 by Mark Fleischer

Module 8.1: Hebbian Learning



This Sub-Module Covers ...

- The concept of Hebbian Learning.
- How to implement Hebbian learning paradigms with recurrent neural networks.
- How to model recurrent networks using matrix methods.



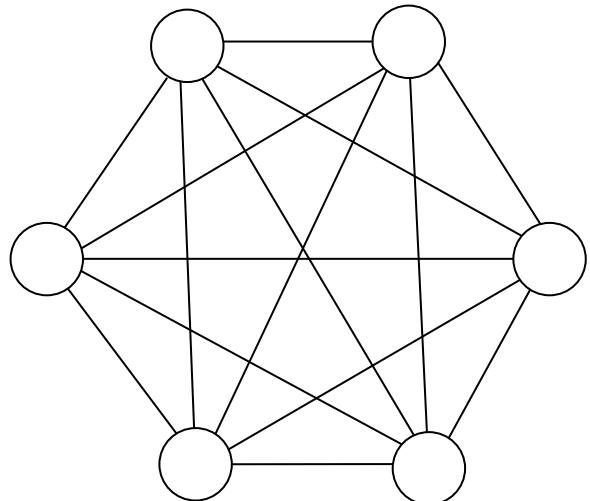
Dynamical Systems

- Neural Networks can be fashioned into dynamical systems.
- Feeding outputs back as inputs and cycling them ala fixed point exercise.
- How does such a system behave?
- Let's examine the most basic formulation.



Associative Neural Networks

- Examine the Hebbian Learning paradigm.
- Examine how and why the Hopfield Network works.



For n nodes, there are $\frac{n(n - 1)}{2}$ connections which can be associated as the weights between each node.

With the addition of edges from each node to itself, and the symmetry of the nodes, we get n^2 weights.



Hebbian Learning

- In this type of net, and the FFBP algorithm, once the weights are established they are treated as fixed for the given input/output pairs that are used to train the net.
- Experiments with physiological neurons however reveal that the post-synaptic potential varies with time (as I mentioned during our first class).
- Moreover, the “weights” are also affected temporally by other efficacies for the same node. Some excite, others inhibit and if inhibitory potentials (negative weights) have “fired” in close temporal proximity to another excitatory potential, both may then have modified efficacies closer to their average (zero for example).
- Long-term potentiation of synapses, long-term depression of synapses occur if the potentials remain increased or decreased for extended periods of time.
- Thus, repeated temporal firings over time can lead to a more permanent modification of weights while short-term modifications do not last as long.



Hebbian Learning

- “Synaptic changes that are driven by correlated activity of pre- and post-synaptic neurons. This class of learning rule can be motivated by Hebb's principle and is therefore often called ‘Hebbian learning’.”
- This is often described in terms of *synaptic plasticity*.

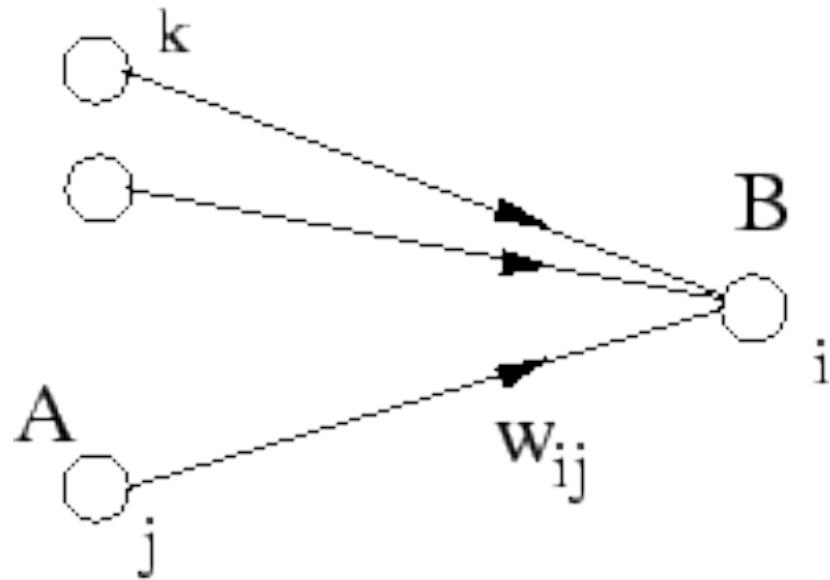


Hebbian Learning

- When an axon of cell *A* is near enough to excite cell *B* or repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that *A*'s synaptic efficacy, as one of the cells firing *B*, is increased.



Hebbian Learning



The change at synapse w_{ij} depends on the state of the presynaptic neuron j and the postsynaptic neuron i and the present efficacy w_{ij} .



Hebbian Learning

- Correlation-based learning is now generally called *Hebbian learning*. Correlations with firings on either side of the synapse are the basis of this type of learning which helps to stabilize the behavior of the neuron. Leads to the “cascade of associations” I mentioned in the first class.
- Long-lasting changes of synaptic efficacies were found experimentally by many researchers.
- How can we model this type of behavior?
- There are two aspects in Hebb's postulate that are particularly important, viz. **locality** and **cooperativity**.
 - Locality means that the change of the synaptic efficacy can only depend on local variables, i.e., on information that is available at the site of the synapse, such as pre- and postsynaptic firing rate, and the actual value of the synaptic efficacy, but not on the activity of other neurons. Based on the locality of Hebbian plasticity we can make a rather general ansatz for the change of the synaptic efficacy,



Hebbian Idea

$$\frac{dw_{ij}}{dt} = f(w_{ij}, v_i, v_j)$$

The main idea of Hebbian Learning is that the behavior of other nodes can affect the synaptic weights of a given node in a manner that is either **reinforcing** or **inhibiting**.



Network Structure

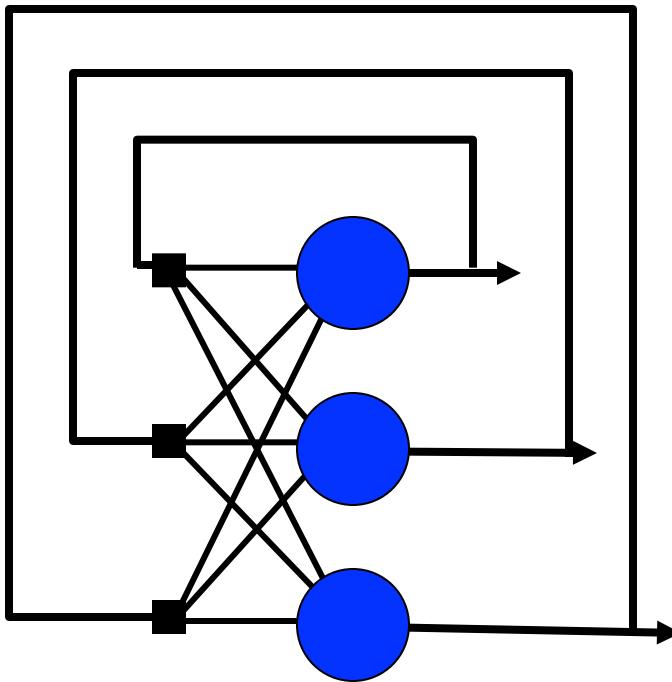
Let's start with simple ideas:

- One node for every input element of an exemplar.
- Two arcs between any two nodes (symmetry).
- No arc from a node to itself.
- Should map an exemplar to itself.



The Recurrent Network Topology

Let's view the network a bit differently.



Exemplar:

(1, -1, -1)

What is the weight from
Node 1 to Node 2?



Network to Matrix

Each output x_i for a node i in a network of N perceptrons can be written as:

$$x_i = \sum_{j=1}^N w_{ij} x_j$$

But this is just the definition of the i^{th} element of a vector after a matrix/vector multiplication!

$$\mathbf{x}_{k+1} = \mathbf{W}\mathbf{x}_k$$

where index k corresponds to the iterate number.



Network to Matrix

$$\mathbf{X}_{k+1} = \mathbf{W}\mathbf{X}_k$$

But how should we define \mathbf{W} ?

Sometimes in order to answer a question relating to a problem,
we must change the problem!

$$\mathbf{p} = \mathbf{W}\mathbf{p}$$

Instead of deciphering a general dynamical system,
let's examine a special case --- a fixed point!



Introduction to Neural Networks

**Johns Hopkins University
Engineering for Professionals Program**

605-447/625-438

Dr. Mark Fleischer

Copyright 2014 by Mark Fleischer

Module 8.2: Hebbian Learning Continued



This Sub-Module Covers ...

- Performance evaluation.
- Proper training does not merely involve minimizing the error function using the training data.
- Must have some appropriate way of gauging how well the network performs using data it hasn't seen during the training phase.



Network to Matrix

Each output x_i for a node i with N perceptrons can be written as:

$$x_i = \sum_{j=1}^N w_{ij} x_j$$

But this is just the definition of the i^{th} element of a vector after a matrix/vector multiplication!

$$\mathbf{x}_{k+1} = \mathbf{W}\mathbf{x}_k$$

where index k corresponds to the iterate number.



Network to Matrix

$$\mathbf{X}_{k+1} = \mathbf{W}\mathbf{X}_k$$

But how should we define \mathbf{W} ?

Sometimes in order to answer a question relating to a problem,
we must change the problem!

$$\mathbf{p} = \mathbf{W}\mathbf{p}$$

Instead of deciphering a general dynamical system,
let's examine a special case --- a fixed point!



A Test Exemplar

Given some input pattern (or exemplar)

$$p = \begin{Bmatrix} \oplus & \cdot & \cdot \\ \cdot & \oplus & \cdot \\ \cdot & \cdot & \oplus \end{Bmatrix},$$

where \oplus = On and \cdot = Off

We want to design a network that will converge to a set of exemplars given a partial or noisy representation of the exemplars.

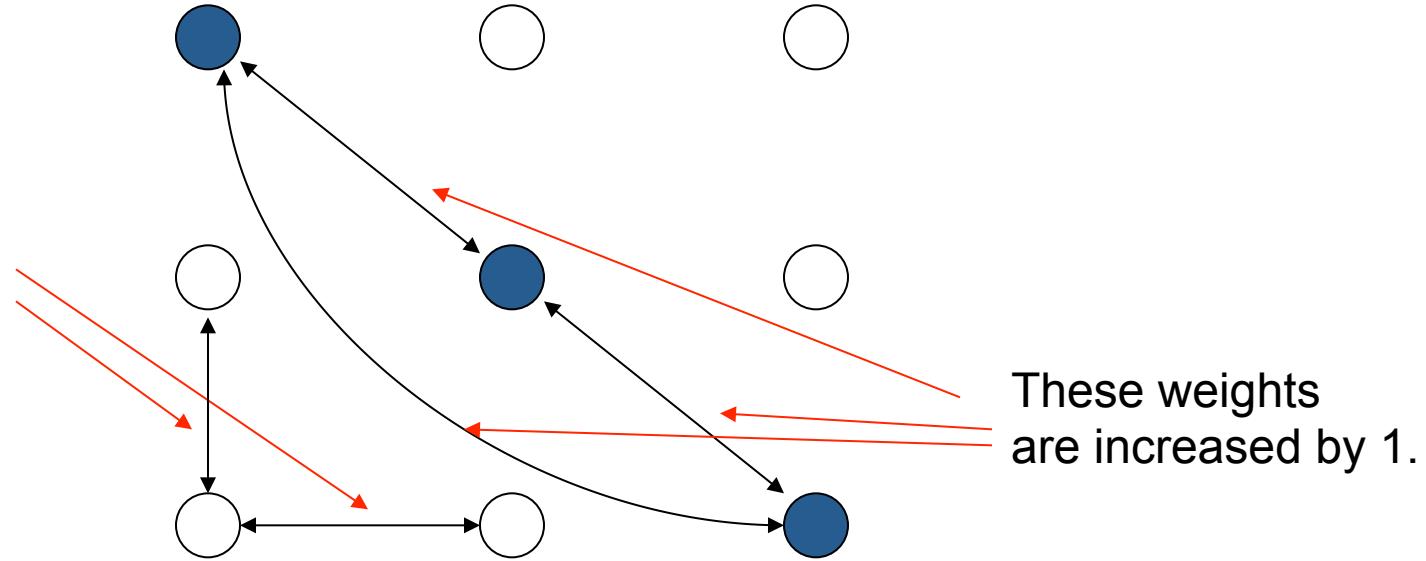


Mutual Reinforcement ala Hebbian Learning

The  represent a 1 input, the  represent a 0 input.

Initially, all weights are 0.

These weights
stay 0.



Note: This doesn't show all the edges (weights).

How can we model this mathematically?



Change the input pattern into a vector of 1's and 0's.

$$\begin{aligned} \mathbf{p}^T &= \left\{ \oplus \cdots | \cdot \oplus \cdot | \cdots \oplus \right\} \\ &= (100 \quad 010 \quad 001) \end{aligned}$$

Now we compute a matrix \mathbf{W} such that $\mathbf{Wp} = \mathbf{p}$.

How should such a matrix be defined?



W = ?

~~**W = I?**~~

Trivial and generic!

Hint: Must be somehow linked to the vector \mathbf{p} to facilitate the calculation
 $\mathbf{Wp} = \mathbf{p}$.

Can we create a matrix from the vector \mathbf{p} ?



$$\mathbf{p} \mathbf{p}^T = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} (100010001) = \left(\begin{array}{cccc|ccc|cc} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{array} \right)$$

We will need to get rid of the 1's on the diagonal. Can't have a node reinforce itself. Why?



We will create/introduce a function Z which operates on matrices and sets diagonal elements to 0.

So let $\Delta\mathbf{W} = Z(\mathbf{p}\mathbf{p}^T)$ and

$$\mathbf{W}_{\text{new}} = \mathbf{W}_{\text{old}} + Z(\mathbf{p}\mathbf{p}^T)$$

We will also use a hard-limiting activation function: $F_h(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases}$

So does $\mathbf{W}_{\text{new}}\mathbf{p} = \mathbf{p}$?



What is the Z function?

- For any 1, 0 vector p producing pp^T we want to subtract a modified identity matrix to get rid of entries on the diagonal with a 1.

E.g., $p = (1 \ 0 \ 1)$ then $pp^T =$

$$\begin{array}{r} 1 \ 0 \ 1 \\ 0 \ 0 \ 0 \\ 1 \ 0 \ 1 \end{array} \xrightarrow{\hspace{2cm}} \begin{array}{r} 0 \ 0 \ 1 \\ 0 \ 0 \ 0 \\ 1 \ 0 \ 0 \end{array}$$

Thus we must subtract the $I^* =$

$$\begin{array}{r} 1 \ 0 \ 0 \\ 0 \ 0 \ 0 \\ 0 \ 0 \ 1 \end{array}$$

Note the diagonal is the same as the vector p and that $I^*p = p$ for any p .



Another example

$$p^T = (1 \ 0 \ 1 \ 0 \ 1) \text{ then } pp^T = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix} \quad I^* = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{and } I^*p = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$



Define $\Delta\tilde{\mathbf{W}} = \mathbf{p}\mathbf{p}^T$; hence $\Delta\mathbf{W} = Z(\mathbf{p}\mathbf{p}^T)$

$$\begin{aligned}\mathbf{W}_{\text{new}} &= \mathbf{W}_{\text{old}} + Z(\mathbf{p}\mathbf{p}^T) \\ &= \Delta\mathbf{W} + \mathbf{W}(0) \\ &= \Delta\mathbf{W}\end{aligned}$$

So does $F_h(\mathbf{W}_{\text{new}}\mathbf{p}) = \mathbf{p}$?

$$\begin{aligned}F_h(\mathbf{W}_{\text{new}}\mathbf{p}^T) &= F_h[\Delta\mathbf{W}\mathbf{p}] \\ &= F_h[(\Delta\tilde{\mathbf{W}} - \mathbf{I}_n^*)\mathbf{p}] \\ &= F_h[\Delta\tilde{\mathbf{W}}\mathbf{p} - \mathbf{p}] \\ &= F_h[\mathbf{p}\mathbf{p}^T\mathbf{p} - \mathbf{p}]\end{aligned}$$



$$\begin{aligned} F_h(\mathbf{W}_{\text{new}} \mathbf{p}) &= F_h[\mathbf{p} \mathbf{p}^T \mathbf{p} - \mathbf{p}] \\ &= F_h[d\mathbf{p} - \mathbf{p}] \\ &= F_h[(d-1)\mathbf{p}] \end{aligned}$$

where $\mathbf{p}^T \mathbf{p} = d$, a scalar equal to the number of 1's in the vector (do you see why?)

Note that $d \geq 1$ in non-trivial cases. Therefore, the above

$$= \mathbf{p}$$



What About Noisy Inputs?

$$I_3 = \sum_{j=1}^9 w_{3j} x_j \text{ with } w_{33} = 0. \text{ In general, } I_i = \sum_{j=1}^N w_{ij} x_j$$

for N = number of neurons = number of elements in a pattern.

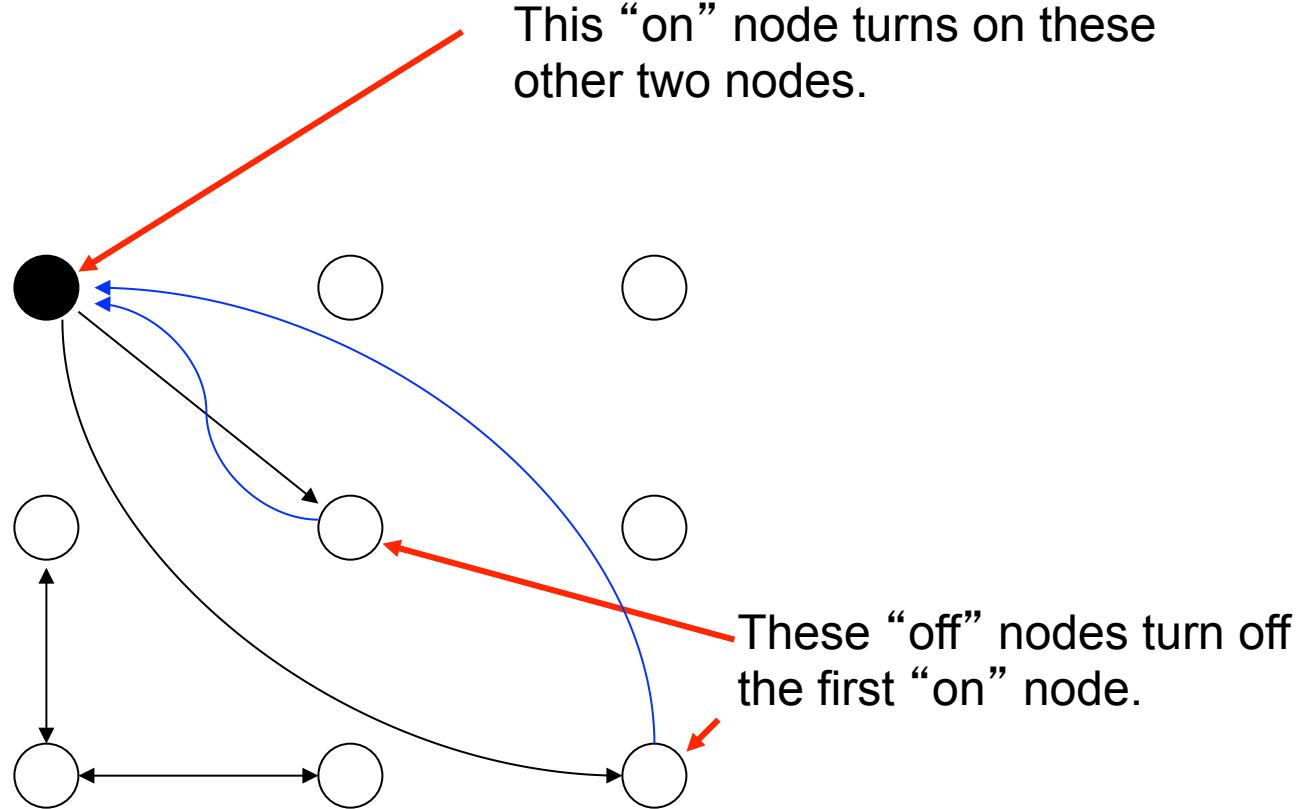
Now make a noisy version of the pattern p :

$$\tilde{p} = \left\{ \begin{array}{cccc} \oplus & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{array} \right\} = (100 | 000 | 000)^T$$



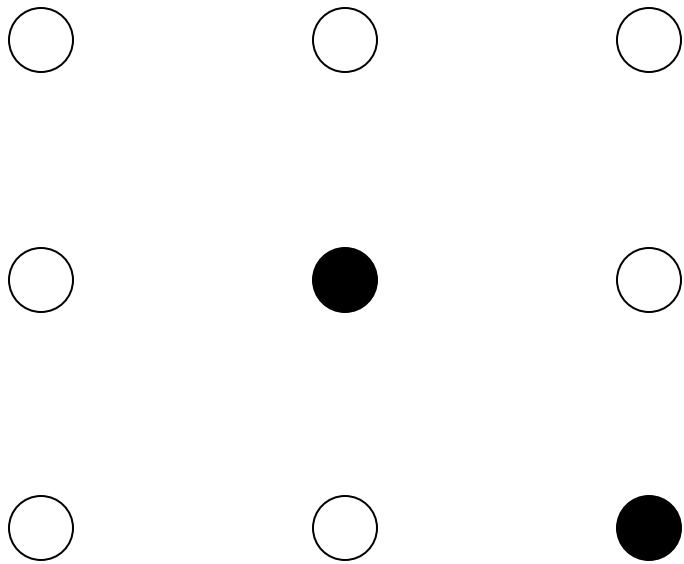
$$W_{\text{new}} \tilde{p} = \left(\begin{array}{ccc|ccc|ccc} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{array} \right) \left(\begin{array}{c} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \right) = \left(\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{array} \right)$$

Putting this into F_h produces no change.





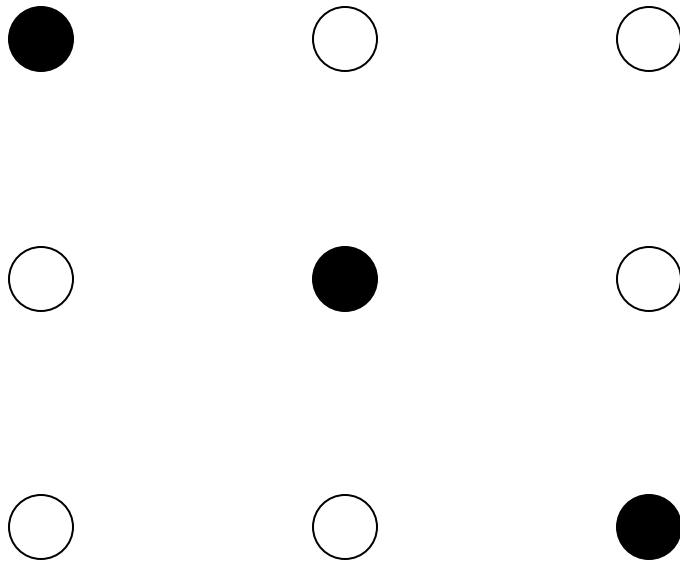
The resulting pattern:



Now using this as the input in a second iteration, and using the same logic, we get



Our original pattern.





But hold on. Not all input patterns result in our “trained” pattern.

$$\mathbf{W} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

and in fact \mathbf{W}

$$\begin{pmatrix} 0 \\ \times \\ \times \\ \times \\ 0 \\ \times \\ \times \\ \times \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

This is because the 1 in the first vector was not associated with any 1 in the exemplar, hence, the other 0s turned this 1 to off (0).



What About Another Exemplar?

$$\begin{array}{ccc} \circ & \circ & \circ \\ \bullet & \bullet & \bullet \\ \circ & \circ & \circ \end{array} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$



$$\Delta \mathbf{W} = Z(\mathbf{q}\mathbf{q}^T)$$

$$\mathbf{q}\mathbf{q}^T = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} (000111000) = \left(\begin{array}{cccc|ccc|c} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right)$$



$$Z(\mathbf{q} \mathbf{q}^T) = \begin{pmatrix} 0 & 0 & 0 & | & 0 & 0 & 0 & | & 0 & 0 & 0 \\ 0 & 0 & 0 & | & 0 & 0 & 0 & | & 0 & 0 & 0 \\ 0 & 0 & 0 & | & 0 & 0 & 0 & | & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & | & 0 & 1 & 1 & | & 0 & 0 & 0 \\ 0 & 0 & 0 & | & 1 & 0 & 1 & | & 0 & 0 & 0 \\ 0 & 0 & 0 & | & 1 & 1 & 0 & | & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & | & 0 & 0 & 0 & | & 0 & 0 & 0 \\ 0 & 0 & 0 & | & 0 & 0 & 0 & | & 0 & 0 & 0 \\ 0 & 0 & 0 & | & 0 & 0 & 0 & | & 0 & 0 & 0 \end{pmatrix}$$



$$\mathbf{W}_{\text{new}} = \mathbf{W}_{\text{old}} + Z(\mathbf{q}\mathbf{q}^T) = \left(\begin{array}{ccc|ccc|ccc} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{array} \right)$$

This system “knows” (is trained for) the two exemplars.



$$F_h(\mathbf{W}_{\text{new}} \tilde{\mathbf{p}}) = F_h \left(\begin{array}{c|ccccc|ccc} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{array} \right) = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$



$$F_h(\mathbf{W}_{\text{new}} \tilde{\mathbf{p}}) = F_h \left(\begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \right) = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

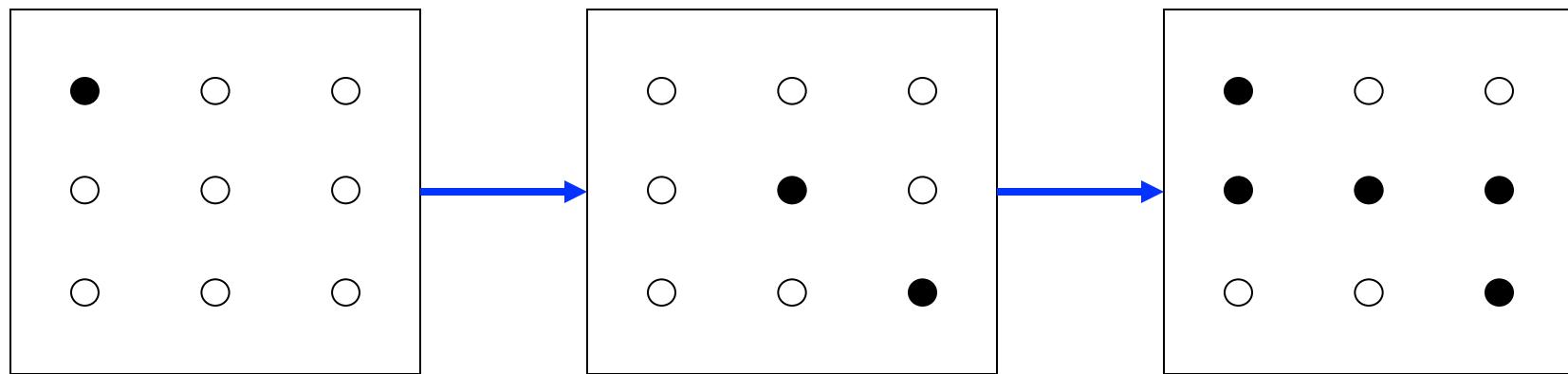
The matrix is partitioned into four quadrants by red lines: top-left (3x3), top-right (3x3), bottom-left (3x3), and bottom-right (3x3). A red arrow labeled '1' points to the first element of the rightmost column of the matrix. Another red arrow labeled '2' points to the second element of the rightmost column of the matrix.



$$F_h(\mathbf{W}_{\text{new}} \tilde{\mathbf{p}}_1) = F_h \left(\begin{pmatrix} 2 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \right) = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \tilde{\mathbf{p}}_2$$



$$F_h(\mathbf{W}_{\text{new}} \tilde{\mathbf{p}}_2) = F_h \left(\begin{array}{c|ccc|cc|ccc} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{array} \right) = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$



All exemplar patterns “on”. This is an example of “heat death”.



Introduction to Neural Networks

**Johns Hopkins University
Engineering for Professionals Program**

605-447/625-438

Dr. Mark Fleischer

Copyright 2014 by Mark Fleischer

Module 8.3: Hopfield Networks



This Sub-Module Covers ...

- We've examined Hebbian Learning and the notion of 'heat death'.
- Now we examine ways that can provide inhibition and not just excitation to recurrent networks using bipolar values.



Beyond The Hebbian Paradigm

- Node states should **reinforce** each other if their pattern elements are the same.
- Node states should also **inhibit** each other if their pattern elements are different.

Here, ‘reinforce’ means increasing the magnitude of their connecting weights. Inhibition means increasing the magnitude of their connecting weights, **but in a negative or opposite direction.**

How can this be effected?



The Hopfield Algorithm

$$x_i \in \{-1, 1\}, \quad f(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$

Initialize with M Exemplars and $1 \leq i, j \leq M$:

$$w_{ij} = \begin{cases} \sum_{m=1}^M x_i^{[m]} x_j^{[m]} & i \neq j \\ 0 & i = j \end{cases}$$

$x_i(0) = x_i^*$ for all inputs i ,

$$x_j(t+1) = f \left[\sum_{i=1}^M w_{ij} x_i(t) \right]$$



Instead of using binary values, let's use “bi-polar” values.

$$\begin{aligned}
 \mathbf{p} &= \left\{ \oplus \cdots | \cdot \oplus \cdot | \cdots \oplus \right\} \\
 &= \begin{pmatrix} 1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & 1 & 1 \end{pmatrix} \\
 \mathbf{q} &= \left\{ \cdots | \oplus \oplus \oplus | \cdots \right\} \\
 &= \begin{pmatrix} -1 & -1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 \end{pmatrix}
 \end{aligned}$$

$$F_h(x) = \begin{cases} 1, & \text{if } x > 0 \\ -1, & \text{if } x \leq 0 \end{cases}$$



Benefits of Bipolar Values

- They magically provide inhibition *and* reinforcement:
- Consider: $I/O_A \times \text{link weight} = I/O_B$
 - If we want to *reinforce negative values*:
 - $-1 \times 1 = -1$
 - If we want to *reinforce positive values*:
 - $1 \times 1 = 1$
 - If we want to inhibit negative values:
 - $-1 \times -1 = 1$
 - If we want to inhibit positive values:
 - $1 \times -1 = -1$



$$\Delta \tilde{\mathbf{W}} = \mathbf{p} \mathbf{p}^T = \begin{pmatrix} 1 \\ -1 \\ -1 \\ -1 \\ 1 \\ -1 \\ -1 \\ -1 \\ 1 \end{pmatrix} (1 - 1 - 1 - 1 1 - 1 - 1 - 1) = \begin{pmatrix} 1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 & 1 \\ -1 & 1 & 1 & 1 & -1 & 1 & 1 & 1 & 1 & -1 \\ -1 & 1 & 1 & 1 & -1 & 1 & 1 & 1 & 1 & -1 \\ -1 & 1 & 1 & 1 & -1 & 1 & 1 & 1 & 1 & -1 \\ 1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 & 1 \\ -1 & 1 & 1 & 1 & -1 & 1 & 1 & 1 & 1 & -1 \\ -1 & 1 & 1 & 1 & -1 & 1 & 1 & 1 & 1 & -1 \\ -1 & 1 & 1 & 1 & -1 & 1 & 1 & 1 & 1 & -1 \\ 1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 & 1 \end{pmatrix}$$

Each row is either \mathbf{p} or $-\mathbf{p}$.



$$\Delta \mathbf{W} = \Delta \widetilde{\mathbf{W}} - \mathbf{I}_9 = \begin{pmatrix} (1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 & 1) \\ 0 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 & 1 \\ -1 & 0 & 1 & 1 & -1 & 1 & 1 & 1 & 1 & -1 \\ -1 & 1 & 0 & 1 & -1 & 1 & 1 & 1 & 1 & -1 \\ -1 & 1 & 1 & 0 & -1 & 1 & 1 & 1 & 1 & -1 \\ 1 & -1 & -1 & -1 & 0 & -1 & -1 & -1 & -1 & 1 \\ -1 & 1 & 1 & 1 & -1 & 0 & 1 & 1 & 1 & -1 \\ -1 & 1 & 1 & 1 & -1 & 1 & 0 & 1 & 1 & -1 \\ -1 & 1 & 1 & 1 & -1 & 1 & 1 & 1 & 0 & -1 \\ 1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 & 0 \end{pmatrix}$$

$$\mathbf{W}_{\text{new}} = \Delta \mathbf{W} + \mathbf{W}_{\text{old}}$$

and so $F_h(\mathbf{W}_{\text{new}} \mathbf{p}) = \mathbf{p}$



Whereas before we had:

Remember that

$$\begin{aligned} F_h(\mathbf{W}_{\text{new}} \mathbf{p}) &= F_h[\mathbf{p} \mathbf{p}^T \mathbf{p} - \mathbf{p}] \\ &= F_h[d\mathbf{p} - \mathbf{p}] \\ &= F_h[(d-1)\mathbf{p}] \end{aligned}$$

$$\mathbf{W} = \mathbf{p} \mathbf{p}^T - \mathbf{I}^*$$

Now, we have this:

$$\begin{aligned} F_h(\mathbf{W}_{\text{new}} \mathbf{p}) &= F_h[\mathbf{p} \mathbf{p}^T \mathbf{p} - \mathbf{p}] \\ &= F_h[n\mathbf{p} - \mathbf{p}] \\ &= F_h[(n-1)\mathbf{p}] \end{aligned}$$

where n is the size (length) of the vector \mathbf{p} .

Do you see where the difference comes from?



And for the second exemplar...

$$\mathbf{q}\mathbf{q}^T = \begin{pmatrix} -1 \\ -1 \\ -1 \\ 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \end{pmatrix} \begin{pmatrix} -1 & -1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \end{pmatrix}$$

$\Delta \tilde{\mathbf{W}}$



And for the second exemplar...

$$Z(\mathbf{q}\mathbf{q}^T) = \left(\begin{array}{ccc|ccc|c} 0 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \\ 1 & 0 & 1 & -1 & -1 & -1 & 1 & 1 & 1 \\ 1 & 1 & 0 & -1 & -1 & -1 & 1 & 1 & 1 \\ \hline -1 & -1 & -1 & 0 & 1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 0 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 1 & 0 & -1 & -1 & -1 \\ \hline 1 & 1 & 1 & -1 & -1 & -1 & 0 & 1 & 1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & 0 & 1 \\ 1 & 1 & 1 & -1 & -1 & -1 & 1 & 1 & 0 \end{array} \right)$$

$\Delta\mathbf{W}$

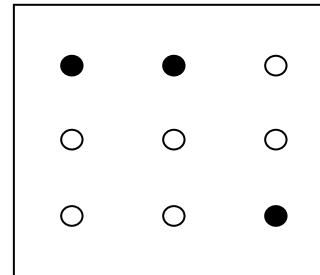


$$\mathbf{W}_{\text{new}} = \mathbf{W}_{\text{old}} + \Delta \mathbf{W}$$

$$= \begin{pmatrix} 0 & 0 & 0 & -2 & 0 & -2 & 0 & 0 & 2 \\ 0 & 0 & 2 & 0 & -2 & 0 & 2 & 2 & 0 \\ 0 & 2 & 0 & 0 & -2 & 0 & 2 & 2 & 0 \\ -2 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & -2 \\ 0 & -2 & -2 & 0 & 0 & 0 & -2 & -2 & 0 \\ -2 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & -2 \\ 0 & 2 & 2 & 0 & -2 & 0 & 0 & 2 & 0 \\ 0 & 2 & 2 & 0 & -2 & 0 & 2 & 0 & 0 \\ 2 & 0 & 0 & -2 & 0 & -2 & 0 & 0 & 0 \end{pmatrix}$$



Now examine a noisy representation of \mathbf{p} .



$$\tilde{\mathbf{p}}^T = (1 \ 1 -1 \ -1 \ -1 -1 \ -1 -1 \ 1)$$

$$F_h(\mathbf{W}\tilde{\mathbf{p}}) = \mathbf{p}'_1$$

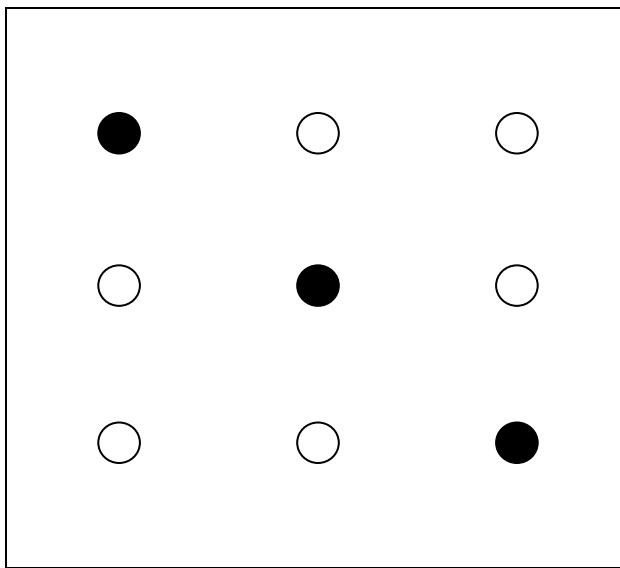
$$F_h(\mathbf{W}\mathbf{p}'_1) = \mathbf{p}'_2$$

$$\vdots$$

$$F_h(\mathbf{W}\mathbf{p}'_{n-1}) = \mathbf{p}'_n$$



This will converge to exemplar 1 after only two iterations and does not reflect “heat death”.





The Hopfield Algorithm

$$x_i \in \{-1, 1\}, \quad f(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$

Initialize with M Exemplars and $1 \leq i, j \leq M$:

$$w_{ij} = \begin{cases} \sum_{m=1}^M x_i^{[m]} x_j^{[m]} & i \neq j \\ 0 & i = j \end{cases}$$

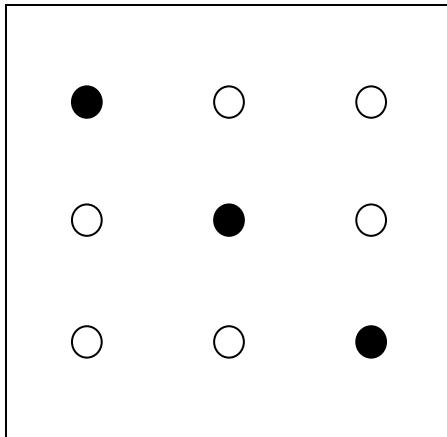
$x_i(0) = x_i^*$ for all inputs i ,

$$x_j(t+1) = f \left[\sum_{i=1}^M w_{ij} x_i(t) \right]$$

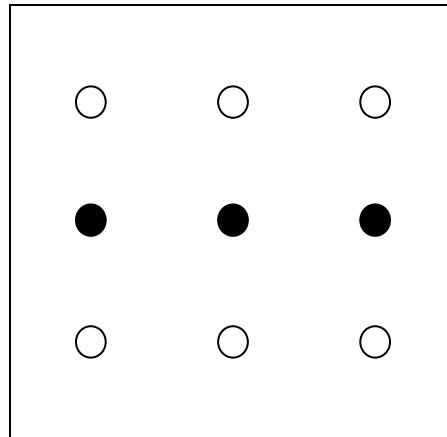


Suppose we have the following three exemplars.

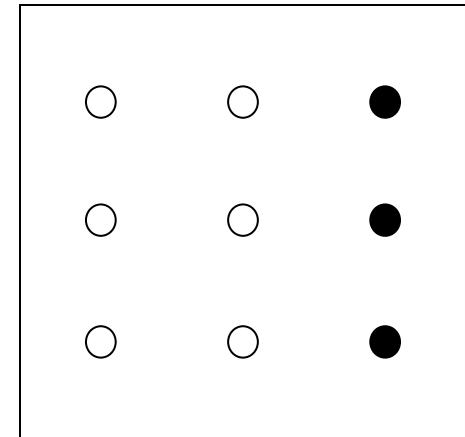
p



q



r



How do you think a Hopfield network trained with these three exemplars will evolve given some part of one of them?



The weight matrix for the first two exemplars was ...

$$\mathbf{W}_{\text{new}} = \mathbf{W}_{\text{old}} + \Delta \mathbf{W}$$

$$= \begin{pmatrix} 0 & 0 & 0 & -2 & 0 & -2 & 0 & 0 & 2 \\ 0 & 0 & 2 & 0 & -2 & 0 & 2 & 2 & 0 \\ 0 & 2 & 0 & 0 & -2 & 0 & 2 & 2 & 0 \\ -2 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & -2 \\ 0 & -2 & -2 & 0 & 0 & 0 & -2 & -2 & 0 \\ -2 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & -2 \\ 0 & 2 & 2 & 0 & -2 & 0 & 0 & 2 & 0 \\ 0 & 2 & 2 & 0 & -2 & 0 & 2 & 0 & 0 \\ 2 & 0 & 0 & -2 & 0 & -2 & 0 & 0 & 0 \end{pmatrix}$$



And for the third exemplar...the weight matrix is...

$$\mathbf{r}\mathbf{r}^T = \begin{pmatrix} -1 \\ -1 \\ 1 \\ -1 \\ -1 \\ 1 \\ -1 \\ -1 \\ 1 \end{pmatrix} \begin{pmatrix} -1 & -1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 \\ -1 & -1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 \\ 1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 \\ -1 & -1 & 1 & -1 & -1 & 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 \\ -1 & -1 & 1 & -1 & -1 & 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 \\ -1 & -1 & 1 & -1 & -1 & 1 & -1 & -1 & 1 \end{pmatrix}$$



Zeroing out the diagonal ...

$$Z(\mathbf{r}\mathbf{r}^T) = \left(\begin{array}{cccc|ccc|ccc} 0 & 1 & -1 & 1 & 1 & -1 & 1 & 1 & -1 \\ 1 & 0 & -1 & 1 & 1 & -1 & 1 & 1 & -1 \\ -1 & -1 & 0 & -1 & -1 & 1 & -1 & -1 & 1 \\ \hline 1 & 1 & -1 & 0 & 1 & -1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 & 0 & -1 & 1 & 1 & -1 \\ -1 & -1 & 1 & -1 & -1 & 0 & -1 & -1 & 1 \\ \hline 1 & 1 & -1 & 1 & 1 & -1 & 0 & 1 & -1 \\ 1 & 1 & -1 & 1 & 1 & -1 & 1 & 0 & -1 \\ -1 & -1 & 1 & -1 & -1 & 1 & -1 & -1 & 0 \end{array} \right)$$

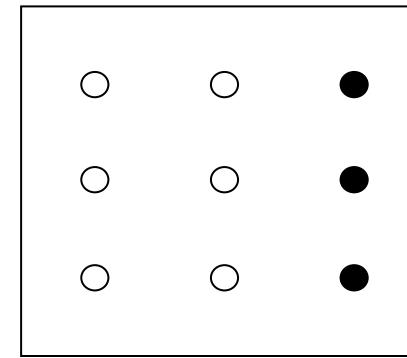
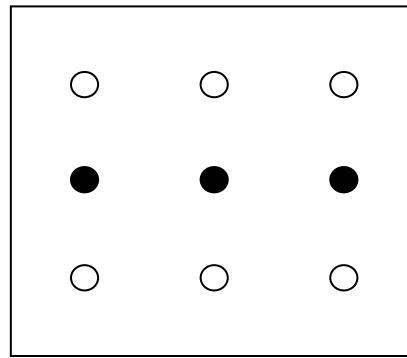
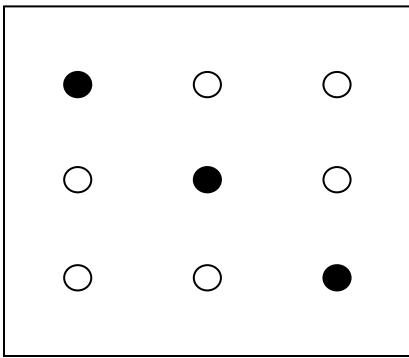


and adding it to the weight matrix for the first two exemplars, we get...

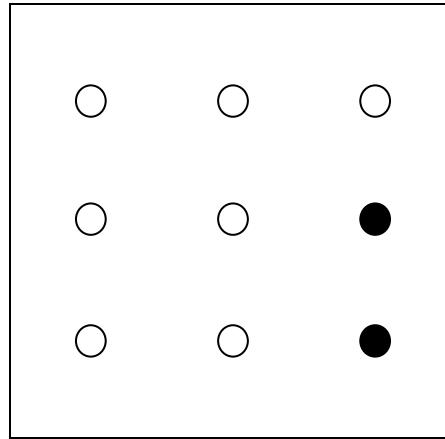
$$\left(\begin{array}{ccccccccc} 0 & 1 & -1 & -1 & 1 & -3 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & -1 & -1 & 3 & 3 & -1 \\ -1 & 1 & 0 & -1 & -3 & 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 0 & 1 & 1 & 1 & 1 & -3 \\ 1 & -1 & -3 & 1 & 0 & -1 & -1 & -1 & -1 \\ -3 & -1 & 1 & 1 & -1 & 0 & -1 & -1 & -1 \\ 1 & 3 & 1 & 1 & -1 & -1 & 0 & 3 & -1 \\ 1 & 3 & 1 & 1 & -1 & -1 & 3 & 0 & -1 \\ 1 & -1 & 1 & -3 & -1 & -1 & -1 & -1 & 0 \end{array} \right)$$



The three exemplars...



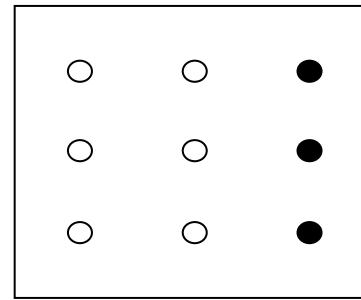
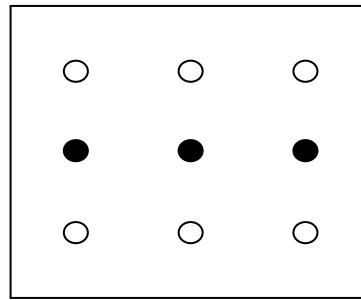
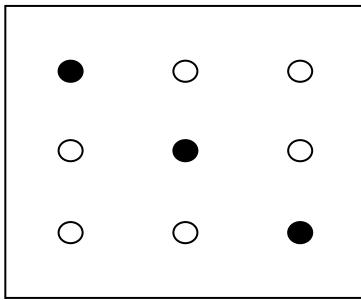
Suppose we present the following pattern to the net.



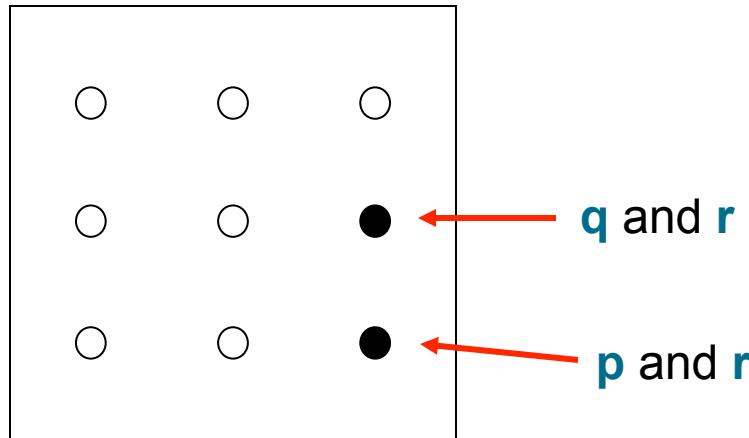
What can we expect the net to converge to? What are some intelligent guess?



The three exemplars...



Suppose we present the following pattern to the net.

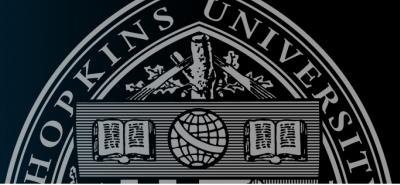


What can we expect the net to converge to? What are some intelligent guess?

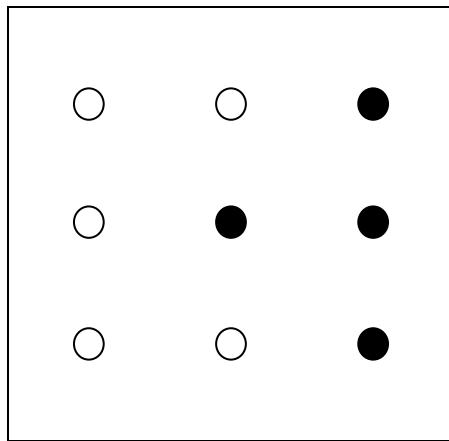


and adding it to the weight matrix for the first two exemplars, we get...

$$\left(\begin{array}{ccccccccc}
 0 & 1 & -1 & -1 & 1 & -3 & 1 & 1 & 1 \\
 1 & 0 & 1 & 1 & -1 & -1 & 3 & 3 & -1 \\
 -1 & 1 & 0 & -1 & -3 & 1 & 1 & 1 & 1 \\
 -1 & 1 & -1 & 0 & 1 & 1 & 1 & 1 & -3 \\
 1 & -1 & -3 & 1 & 0 & -1 & -1 & -1 & -1 \\
 -3 & -1 & 1 & 1 & -1 & 0 & -1 & -1 & -1 \\
 1 & 3 & 1 & 1 & -1 & -1 & 0 & 3 & -1 \\
 1 & 3 & 1 & 1 & -1 & -1 & 3 & 0 & -1 \\
 1 & -1 & 1 & -3 & -1 & -1 & -1 & -1 & 0
 \end{array} \right) \left[\begin{array}{c} -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ 1 \\ -1 \\ -1 \\ 1 \end{array} \right] = \left[\begin{array}{c} -4 \\ -10 \\ 4 \\ -4 \\ 2 \\ 4 \\ -10 \\ -10 \\ 4 \end{array} \right] \rightarrow \left[\begin{array}{c} -1 \\ -1 \\ 1 \\ -1 \\ 1 \\ 1 \\ -1 \\ -1 \\ 1 \end{array} \right]$$



and that vector corresponds to this pattern...



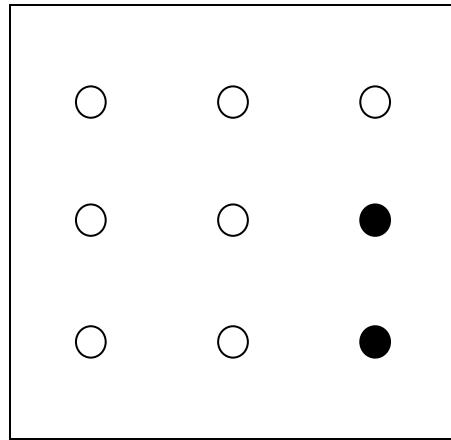


doing another iteration with the last output vector...

$$\left(\begin{array}{ccccccccc} 0 & 1 & -1 & -1 & 1 & -3 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & -1 & -1 & 3 & 3 & -1 \\ -1 & 1 & 0 & -1 & -3 & 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 0 & 1 & 1 & 1 & 1 & -3 \\ 1 & -1 & -3 & 1 & 0 & -1 & -1 & -1 & -1 \\ -3 & -1 & 1 & 1 & -1 & 0 & -1 & -1 & -1 \\ 1 & 3 & 1 & 1 & -1 & -1 & 0 & 3 & -1 \\ 1 & 3 & 1 & 1 & -1 & -1 & 3 & 0 & -1 \\ 1 & -1 & 1 & -3 & -1 & -1 & -1 & -1 & 0 \end{array} \right) \begin{pmatrix} -1 \\ -1 \\ 1 \\ -1 \\ 1 \\ 1 \\ -1 \\ -1 \\ 1 \end{pmatrix} = \begin{pmatrix} -4 \\ -10 \\ -2 \\ -4 \\ -4 \\ 4 \\ -10 \\ -10 \\ 4 \end{pmatrix} \rightarrow \begin{pmatrix} -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ 1 \\ -1 \\ -1 \\ 1 \end{pmatrix}$$



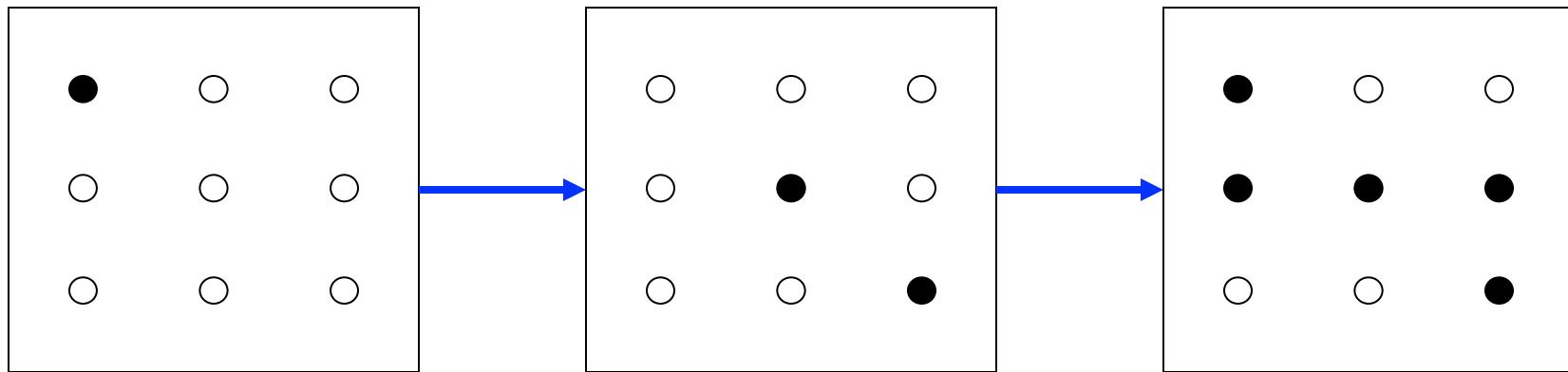
and that vector corresponds to this pattern...



Hmmmm.... what's going on here?



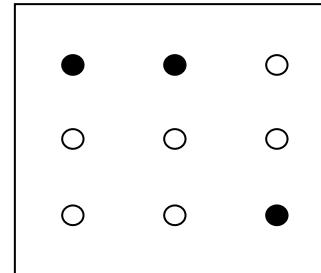
Recall from using the binary values, the net exhibited “heat death”



All exemplar patterns “on”.



Recall a noisy representation of \mathbf{p} .



$$\tilde{\mathbf{p}}^T = (1 \ 1 -1 \ -1 \ -1 -1 \ -1 -1 \ 1)$$

$$F_h(\mathbf{W}\tilde{\mathbf{p}}) = \mathbf{p}'_1$$

$$F_h(\mathbf{W}\mathbf{p}'_1) = \mathbf{p}'_2$$

$$\vdots$$

$$F_h(\mathbf{W}\mathbf{p}'_{n-1}) = \mathbf{p}'_n$$



Convergence of Hopfield Nets

- Obviously, there is a cycle.
- Why are there chapters concerning convergence of Hopfield Nets?
- If there is a cycle, there is no ‘universal’ convergence.
- Can we consider other modalities?



Introduction to Neural Networks

**Johns Hopkins University
Engineering for Professionals Program**

605-447/625-438

Dr. Mark Fleischer

Copyright 2014 by Mark Fleischer

Module 8.4: Hopfield Convergence



What We've Covered So Far...

- Learned about the Hopfield Network
 - Hebbian Learning
 - Recurrent neural networks
 - Matrix/vector representation of a recurrent network
 - Heat death
 - Excitation and Inhibition in Hopfield Networks via bi-polar values
 - Possibility of cycling in Hopfield networks in part because of inhibition and excitation.
- In this sub-module
 - Another modality to the dynamical system.
 - Prove convergence of the Hopfield network using this modality.



A Peek at the Hopfield Convergence Theorem

- In a Hopfield network, with **asynchronous updating**, the Hopfield net will always converge.
- This is a **sufficient condition**. It is not necessary—there may be other ways to converge even without the Hopfield conditions (no self connections $w_{ii} = 0$ for all i , symmetric connections $w_{ji} = w_{ij}$).



Convergence

- Want the simplest approach.
- If we simply go down hill, that's easy ...
- If there exists some lower-bound to some value --- a bottom of a hill, then we'll eventually reach it.
- In other words, if every possible state has a ranking or metric associated with it (we'll call it **energy**), then showing that we always reduce that metric is equivalent to showing that we will never get into a cycle.



So What is Asynchronous Updating?

$$\mathbf{x}_{k+1} = F_h(\mathbf{W}\mathbf{x}_k)$$

$$F_h \left(\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & & \ddots & \\ w_{n1} & w_{n2} & \cdots & w_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \right) = \begin{bmatrix} x_1^* \\ x_2^* \\ \vdots \\ x_n^* \end{bmatrix}$$

$$F_h \left(\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & & \ddots & \\ w_{n1} & w_{n2} & \cdots & w_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \right) = \begin{bmatrix} x_1^* \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$



So What is Asynchronous Updating?

$$\mathbf{x}_{k+1} = F_h(\mathbf{W}\mathbf{x}_k)$$

$$F_h \left(\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & & \ddots & \\ w_{n1} & w_{n2} & \cdots & w_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \right) = \begin{bmatrix} x_1^* \\ x_2^* \\ \vdots \\ x_n^* \end{bmatrix}$$

$$F_h \left(\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & & \ddots & \\ w_{n1} & w_{n2} & \cdots & w_{nn} \end{bmatrix} \begin{bmatrix} x_1^* \\ x_2^* \\ \vdots \\ x_n^* \end{bmatrix} \right) = \begin{bmatrix} x_1^* \\ x_2^* \\ \vdots \\ x_n^* \end{bmatrix}$$



Define the Hecht-Nielsen Function:

$$H(\mathbf{x}) = - \sum_{i=1}^N \sum_{j=1}^N w_{ij} x_i x_j + \sum_{i=1}^N \theta_i x_i$$

Want to prove that

$$\Delta H(\mathbf{x}) \leq 0$$



Define an Update Rule

So, apply the update rule to a given node k (the one being updated---remember, asynchronously). x^* refers to the current iteration. Now,

$$x_i^* = \begin{cases} x_k^* & i = k \\ x_i & i \neq k \end{cases}$$

where k refers to the one we're updating.



The Update Expression

$$\Delta H = H^* - H = - \sum_{i=1}^N \sum_{j=1}^N w_{ij} \left(x_i^* x_j^* - x_i x_j \right)$$

↑ ↑
Current Previous Iteration

Remember, the * refers to an updated value.



The Update Expression

$$= - \underbrace{\sum_{i=1}^N \sum_{\substack{j \neq k \\ j=1}}^N w_{ij} (x_i^* x_j^* - x_i x_j)}_A - \underbrace{\sum_{i=1}^N w_{ik} (x_i^* x_k^* - x_i x_k)}_B$$

Simplifying Part *B* first, we get:

$$B = - \sum_{\substack{i=1 \\ i \neq k}}^N w_{ik} (x_i x_k^* - x_i x_k) = - \sum_{\substack{i=1 \\ i \neq k}}^N w_{ik} (x_k^* - x_k) x_i$$

Why?



The Update Expression

Remember,

$$x_i^* = \begin{cases} x_k^* & i = k \\ x_i & i \neq k \end{cases}$$

$$B = - \sum_{\substack{i=1 \\ i \neq k}}^N w_{ik} (x_i x_k^* - x_i x_k) = - \sum_{\substack{i=1 \\ i \neq k}}^N w_{ik} (x_k^* - x_k) x_i$$

First, $w_{kk} = 0$ (no self-connections) and $x_i^* = x_i$ for $i \neq k$.

Which reduces to:

$$= - \sum_{\substack{i=1 \\ i \neq k}}^N w_{ik} x_i \Delta x_k \quad \text{for Part B.}$$



The Update Expression

$$= - \underbrace{\sum_{i=1}^N \sum_{\substack{j \neq k \\ j=1}} w_{ij} (x_i^* x_j^* - x_i x_j)}_A - \underbrace{\sum_{i=1}^N w_{ik} (x_i^* x_k^* - x_i x_k)}_B$$

Now Part A, recall that for i or $j \neq k$, $x_i^* = x_i$ so the sum

$$(x_i^* x_j^* - x_i x_j) = 0$$

so $i = k$ is the only term left, so

$$A = - \sum_{\substack{j=1 \\ j \neq k}}^N w_{kj} (x_k^* x_j - x_k x_j) = - \sum_{\substack{j=1 \\ j \neq k}}^N w_{kj} x_j \Delta x_k$$



Let's Sum the deltas...

$$\begin{aligned}
 \Delta H = A + B &= - \sum_{\substack{j=1 \\ j \neq k}}^N w_{kj} x_j \Delta x_k - \sum_{\substack{i=1 \\ i \neq k}}^N w_{ik} x_i \Delta x_k \\
 &= - \sum_{\substack{j=1 \\ j \neq k}}^N w_{kj} x_j \Delta x_k - \sum_{\substack{j=1 \\ j \neq k}}^N w_{jk} x_j \Delta x_k \\
 &= - \sum_{\substack{j=1 \\ j \neq k}}^N (w_{kj} + w_{jk}) x_j \Delta x_k = -2 \underbrace{\sum_{\substack{j=1 \\ j \neq k}}^N w_{kj} x_j \Delta x_k}_{Activity \ a_k}
 \end{aligned}$$



Implications for cases

$$\Delta H = -2 \sum_{\substack{j=1 \\ j \neq k}}^N w_{kj} x_j \Delta x_k$$

Activity a_k

Suppose

$$a_k > 0, \quad \text{then} \quad x_k^* = f_h(a_k) = 1 > 0$$

then

$$\Delta x_k = x_k^* - x_k \geq 0.$$



Implications for cases

$$\Delta H = -2 \sum_{\substack{j=1 \\ j \neq k}}^N w_{kj} x_j \Delta x_k$$

Activity a_k

Activity $a_k > 0$

↓

$$\Delta x_k = x_k^* - x_k \geq 0.$$

Why? Well, if $x^* = 1$, then x must have been either 1 (unchanged) or -1 (changed) hence $1-1=0$ or $1 - -1 = 2 > 0$. Thus, $-2a_k\Delta x_k \leq 0$.



Implications for cases

$$\Delta H = -2 \sum_{\substack{j=1 \\ j \neq k}}^N w_{kj} x_j \Delta x_k$$

Activity a_k

Suppose $a_k < 0$, then $x_k^* = f_h(a_k) = -1 < 0$

Activity $a_k < 0$



$$\Delta x_k = x_k^* - x_k \leq 0.$$



Implications for cases

$$\Delta H = -2 \sum_{\substack{j=1 \\ j \neq k}}^N w_{kj} x_j \Delta x_k$$

Activity a_k

Activity $a_k < 0$



$$\Delta x_k = x_k^* - x_k \leq 0.$$

Why? Well, if $x^* = -1$, then x must have been either -1 (unchanged) or 1 (changed) hence $-1 - -1 = 0$ or $-1 - 1 = -2 < 0$. So again, $-2a_k\Delta x_k \leq 0$.



H-N Function Never Increases

- This means, the H value can only decrease and since there is a lower bound, it will eventually converge such that

$$f_h \left(\sum_j w_{ij} x_j \right) = \mathbf{x}$$

Cannot oscillate between solutions. Why?



Now $\Delta H = -2a_k \Delta x_k = \begin{cases} 0 \\ -4|a_k| \end{cases}$

For $a_k > 0$:

If $\Delta x_k = 0$, then no change in state, hence no oscillation.

If $\Delta x_k \neq 0$ then $\Delta x_k = 2$ and $\Delta H < 0$ as we showed above. Can't increase H .

For $a_k \leq 0$:

In this case, then $\Delta H = 0$ if $a_k = 0$ or $\Delta x_k = 0$.

The latter means no change. If $a_k = 0$ then $\Delta H = 0$ and
Again no change in H .



Introduction to Neural Networks

**Johns Hopkins University
Engineering for Professionals Program**

605-447/625-438

Dr. Mark Fleischer

Copyright 2014 by Mark Fleischer

Module 9.1: Hopfield Memory Capacity



What We've Covered So Far...

- Learned about the Hopfield Network
 - Hebbian Learning
 - Recurrent neural networks
 - Matrix/vector representation of a recurrent network
 - Heat death
 - Excitation and Inhibition in Hopfield Networks via bi-polar values
 - Possibility of cycling in Hopfield networks in part because of inhibition and excitation.
- In this sub-module
 - We examine how many exemplars a Hopfield network can ‘reasonably’ store.



H-N Function Never Increases

- This means, the H value can only decrease and since there is a lower bound, it will eventually converge such that

$$f_h \left(\sum_j w_{ij} x_j \right) = \mathbf{x}$$

Cannot oscillate between solutions.



The Hopfield Outerproduct

Recall

$$w_{ij} = \begin{cases} \sum_{s=1}^P x_i^s x_j^s & i \neq j \\ 0 & i = j \end{cases}$$

$$\text{Let } \mathbf{x}^r = (x_1^r, x_2^r, \dots, x_n^r)$$

This corresponds to the r^{th} exemplar.



Memory Recall/Completion

Activity of the i^{th} neuron
at the end of the first
iteration.

Substituting in the
expression for w_{ij}

$$a_i(1) = \sum_{\substack{j=1 \\ j \neq i}}^n w_{ij} x_j^r$$

$$a_i(1) = \sum_{\substack{j=1 \\ j \neq i}}^n \sum_{s=1}^P x_i^s x_j^s x_j^r$$



Memory Recall/Completion

From previous slide:

$$a_i(1) = \sum_{j=1}^n \sum_{\substack{s=1 \\ j \neq i}}^P x_i^s x_j^s x_j^r$$

Let's do some mathematical tricks and break up the double sum:

$$a_i(1) = \sum_{\substack{j=1 \\ j \neq i}}^n \left[\underbrace{x_i^r x_j^r x_j^r}_{\substack{s=r^{\text{th}} \\ \text{term}}} + \sum_{\substack{s=1 \\ s \neq r}}^P x_i^s x_j^s x_j^r \right]$$



Memory Recall/Completion

Some further manipulations:

$$\begin{aligned}
 a_i(1) &= \sum_{\substack{j=1 \\ j \neq i}}^n \left[\underbrace{x_i^r x_j^r x_j^r}_{\substack{s=r^{\text{th}} \\ \text{term}}} + \sum_{\substack{s=1 \\ s \neq r}}^P x_i^s x_j^s x_j^r \right] \\
 &= \sum_{\substack{j=1 \\ j \neq i}}^n x_i^r x_j^r x_j^r + \sum_{\substack{j=1 \\ j \neq i}}^n \sum_{\substack{s=1 \\ s \neq r}}^P x_i^s x_j^s x_j^r
 \end{aligned}$$



Memory Recall/Completion

Some further manipulations:

$$\begin{aligned}
 a_i(1) &= \sum_{\substack{j=1 \\ j \neq i}}^n x_i^r x_j^r x_j^r + \sum_{\substack{j=1 \\ j \neq i}}^n \sum_{\substack{s=1 \\ s \neq r}}^P x_i^s x_j^s x_j^r \\
 &= x_i^r \sum_{\substack{j=1 \\ j \neq i}}^n x_j^r x_j^r + \sum_{\substack{j=1 \\ j \neq i}}^n \sum_{\substack{s=1 \\ s \neq r}}^P x_i^s x_j^s x_j^r
 \end{aligned}$$

So ...

$$a_i(1) = x_i^r (n - 1) + N_i$$

Call this S_i



Memory Recall/Completion

$$\begin{aligned}
 a_i(1) &= x_i^r \sum_{\substack{j=1 \\ j \neq i}}^n x_j^r x_j^r + \sum_{\substack{j=1 \\ j \neq i}}^n \sum_{s=1}^P x_i^s x_j^s x_j^r \\
 &= x_i^r (n - 1) + N_i
 \end{aligned}$$

If the noise term is 0 and we put this into the hard-limiting function

$$\begin{aligned}
 f_h(a_i(1)) &= f_h(x_i^r (n - 1) + N_i) \\
 &= f_h(x_i^r (n - 1)) \\
 &= x_i^r
 \end{aligned}$$



When is the Noise term likely to be 0?

When two different exemplars are *statistically independent*.

What does that mean?

No significant correlations between two exemplars regarding corresponding vector elements.

Roughly, each set of corresponding vector elements has approximately a 50% chance of having the same value.

We can simply say therefore that the sum of the products of the corresponding elements have an expectation value of 0!

$$\begin{aligned} A &= (1, 1, -1, 1, -1, 1) \\ B &= (-1, 1, 1, -1, -1, 1) \end{aligned}$$

$$-1 + 1 - 1 - 1 + 1 + 1 = 0$$



Memory Recall/Completion

What if N_i is not zero? Let's make some quick, hand-waving arguments:

Let's assume that the exemplars are “statistically independent”.
Therefore:

$$N_i \sim N(0, \sigma^2) \text{ that is } \mu(N_i) = 0$$

$$\text{and } \sigma^2(N_i) = (n - 1)(P - 1)$$

Recall that for a normally distributed random variable, 99.5% of all events lie within 3σ .



Memory Recall/Completion

We want to ensure that

$$|S_i| > |N_i|$$

$$|S_i| = n - 1 \geq 3\sigma = 3\sqrt{(n - 1)(P - 1)}$$

$n - 1 \geq 3\sqrt{(n - 1)(P - 1)}$ and squaring both sides...

$$(n - 1)^2 \geq 9(n - 1)(P - 1)$$

$$n - 1 \geq 9(P - 1)$$

Or roughly $P \approx n/9$.



Introduction to Neural Networks

**Johns Hopkins University
Engineering for Professionals Program**

605-447/625-438

Dr. Mark Fleischer

Copyright 2014 by Mark Fleischer

Module 9.2: Binary Associative Memories



In this sub-module...

- We will learn about Binary Associative Memories (BAMs)
 - Based on *Adaptive Resonance Theory*
 - Another example of *unsupervised learning*
 - Restricted form of a Hopfield network
- We will also learn about
 - Concepts such as *Feature Detectors*
 - Matrix/vector analysis of BAMs.

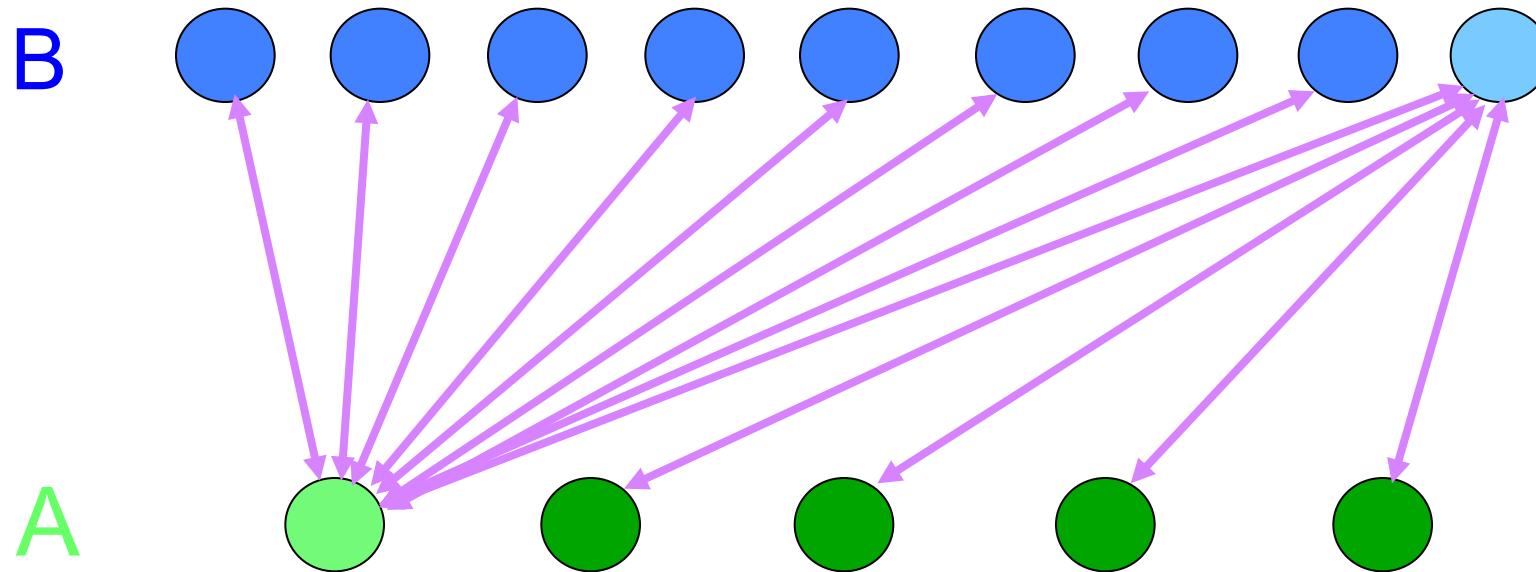


Binary Associative Memories

- Based on a bi-partite graph
- All nodes from one layer connect to all nodes in a second layer.
- No nodes in the same layer connect to each other.
- Connections are bi-directional.
- Same weights in each direction (more general examples don't have this.)

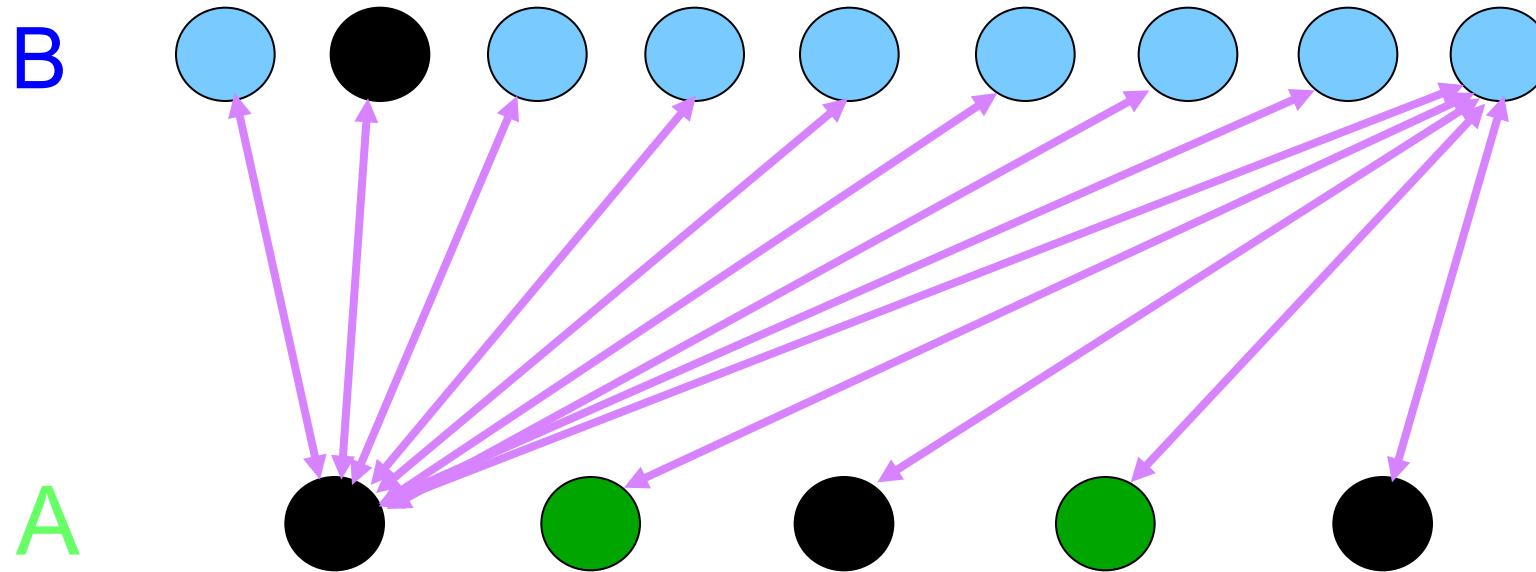


Binary Associative Memories



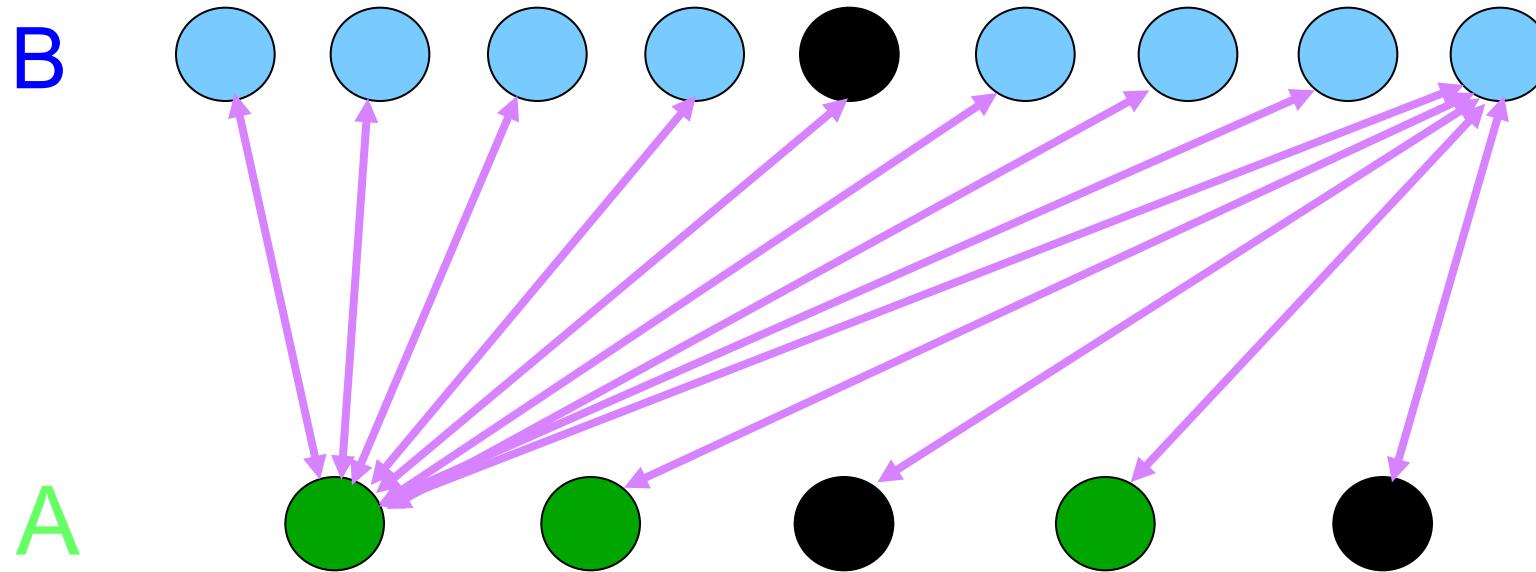


Binary Associative Memories as Feature Detectors



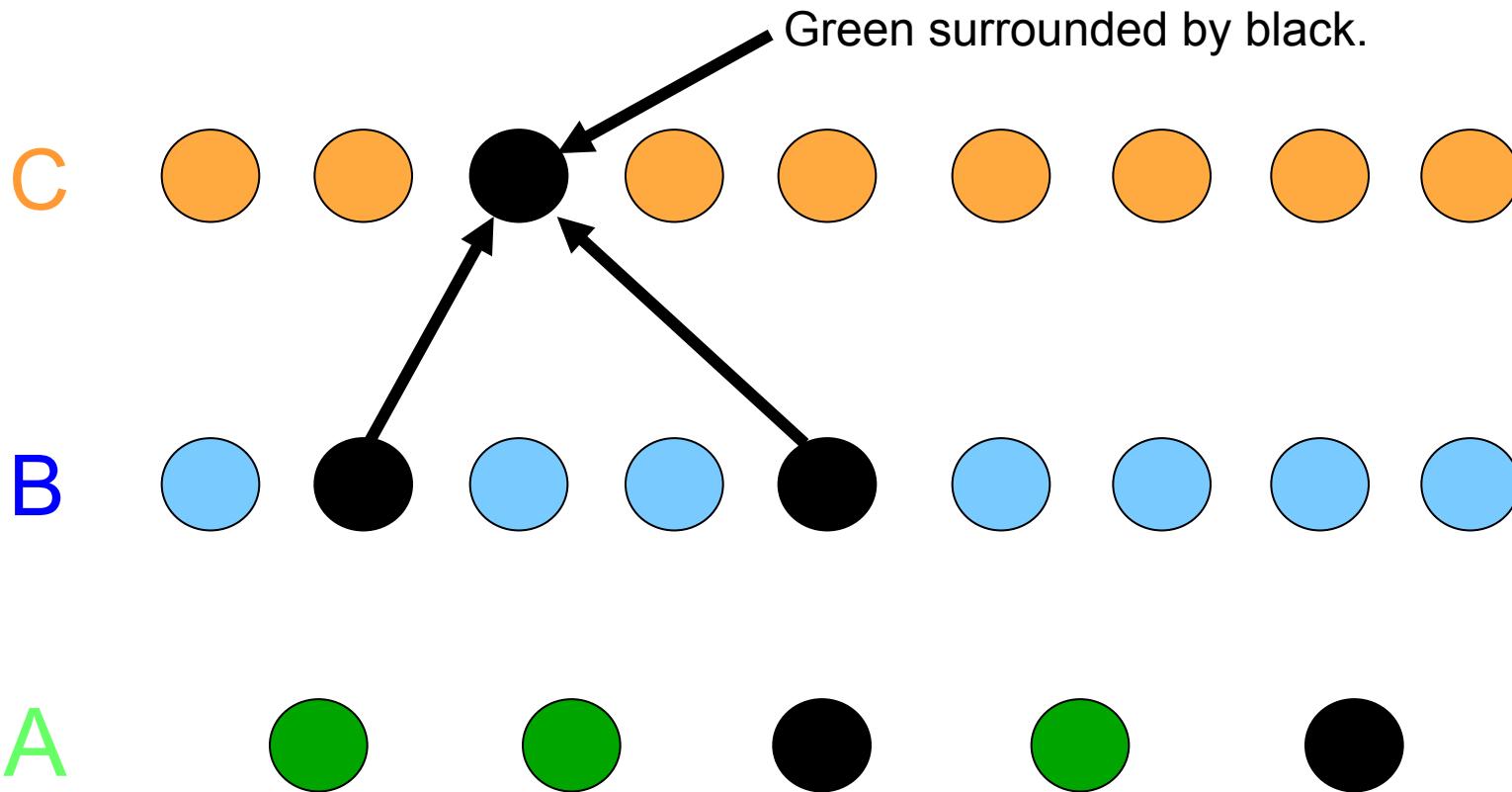


Binary Associative Memories as Feature Detectors





Binary Associative Memories Features of Features



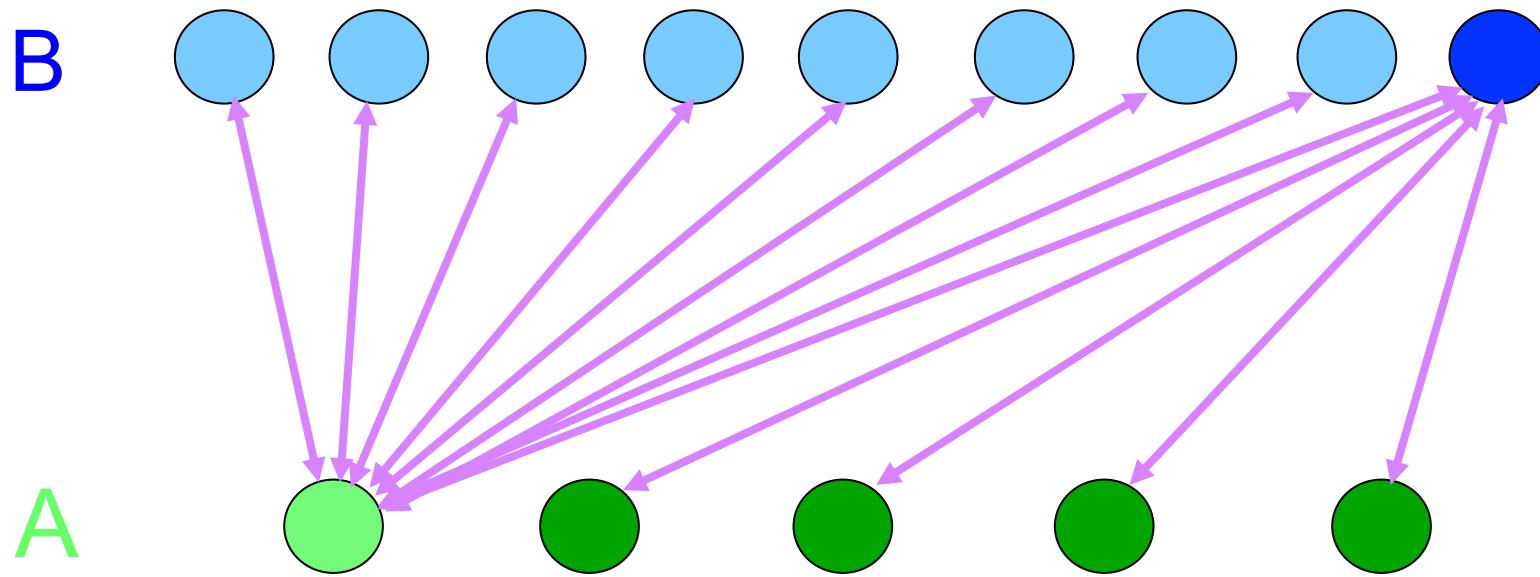


Binary Associative Memories

- More layers, more feature generalizations, more abstractions.
- More layers, more versatility, more weights to train.
- For now we'll only consider two layers.



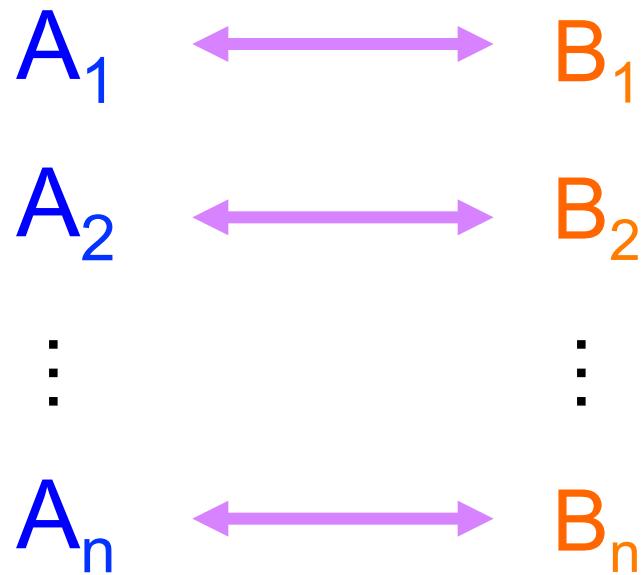
Binary Associative Memories



We can use this topology to train the network with two sets of **associated** exemplars.



Binary Associative Memories



Goal: Noisy A_1 produces a correct B_1 which then produces a correct A_1 .

$$\tilde{A}_1 \rightarrow B_1 \rightarrow A_1$$

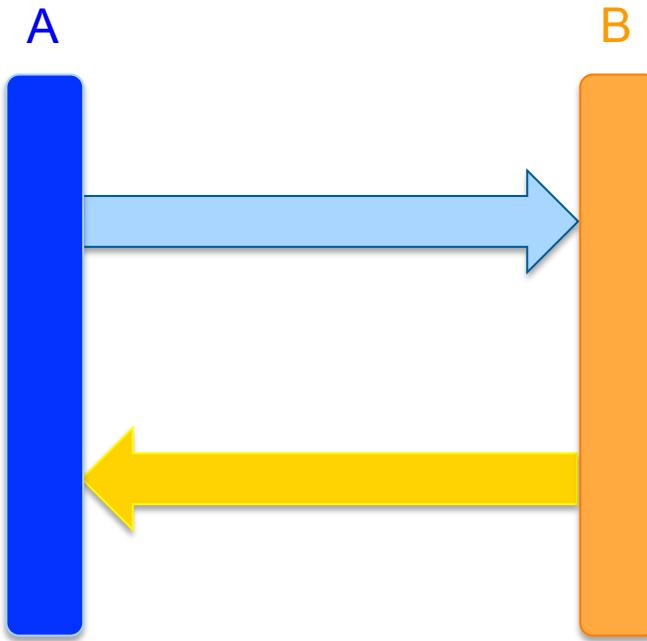


Binary Associative Memories

- Present a noisy A as input to the A nodes.
- The A nodes produce outputs and are presented to the B nodes.
- The B nodes produce outputs and are presented back to the A nodes.



Binary Associative Memories





Binary Associative Memories

An example:

$$\mathbf{A}_1^T = \begin{pmatrix} 1, & -1, & 1, & -1, & 1, & 1 \end{pmatrix}$$

$$\mathbf{A}_2^T = \begin{pmatrix} 1, & 1, & 1, & -1, & -1, & -1 \end{pmatrix}$$

$$\mathbf{B}_1^T = \begin{pmatrix} 1, & 1, & -1, & 1 \end{pmatrix}$$

$$\mathbf{B}_2^T = \begin{pmatrix} 1, & -1, & 1, & 1 \end{pmatrix}$$

Want to associate $\mathbf{A}_1 \Leftrightarrow \mathbf{B}_1$ and $\mathbf{A}_2 \Leftrightarrow \mathbf{B}_2$



Binary Associative Memories

Create a weight matrix ala Hopfield thusly:

$$\mathbf{W}_{6 \times 4} = \mathbf{A}_1 \mathbf{B}_1^T + \mathbf{A}_2 \mathbf{B}_2^T$$

$$= \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & -1 & 1 \end{pmatrix}^T + \begin{pmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \end{pmatrix} \begin{pmatrix} 1 & -1 & 1 & 1 \end{pmatrix}^T$$



Binary Associative Memories

$$\mathbf{W}_{6 \times 4} = \begin{bmatrix} 2 & 0 & 0 & 2 \\ 0 & -2 & 2 & 0 \\ 2 & 0 & 0 & 2 \\ -2 & 0 & 0 & -2 \\ 0 & 2 & -2 & 0 \\ 2 & 0 & 0 & 2 \end{bmatrix}$$

$$\mathbf{W}_{6 \times 4} = \mathbf{A}_1 \mathbf{B}_1^T + \mathbf{A}_2 \mathbf{B}_2^T$$

$$[1 \times 6] \times [6 \times 4] = [1 \times 4]$$



Binary Associative Memories

$$\hat{\mathbf{A}}_1^T \mathbf{W} = \begin{pmatrix} -1, & -1, & 1, & -1, & 1, & 1 \end{pmatrix} \begin{bmatrix} 2 & 0 & 0 & 2 \\ 0 & -2 & 2 & 0 \\ 2 & 0 & 0 & 2 \\ -2 & 0 & 0 & -2 \\ 0 & 2 & -2 & 0 \\ 2 & 0 & 0 & 2 \end{bmatrix}$$

$$\begin{pmatrix} 4, & 4, & -4, & 4 \end{pmatrix} = \hat{\mathbf{b}}^T$$

$$f_h(\hat{\mathbf{b}}^T) = \begin{pmatrix} 1, & 1, & -1, & 1 \end{pmatrix} = \mathbf{B}_1^T$$



Binary Associative Memories

Just some integer value!

$$\hat{A}_1^T W = \hat{A}_1^T A_1 B_1^T + \hat{A}_1^T A_2 B_2^T$$

How do we analyze this multiplication?
Why does this work the way it does?

So what do these integers evaluate to?



Binary Associative Memories

When $\hat{\mathbf{A}}_1^T$ is not too different from \mathbf{A}_1^T , then most of the vector elements will be the same and

$\hat{\mathbf{A}}_1^T \mathbf{A}_1$ will be a positive number while $\hat{\mathbf{A}}_1^T \mathbf{A}_2$ will tend to be ... ?

$$(1) \quad f_h(x) = \begin{cases} 1 & x \geq 1 \\ 0 & -1 < x < 1 \\ -1 & x \leq -1 \end{cases}$$

where n is the vector length

$$(2) \quad f_h(x) = \begin{cases} 1 & x \geq n/2 \\ 0 & -n/2 < x < n/2 \\ -1 & x \leq -n/2 \end{cases}$$



Introduction to Neural Networks

**Johns Hopkins University
Engineering for Professionals Program**

605-447/625-438

Dr. Mark Fleischer

Copyright 2014 by Mark Fleischer

Module 10.1: The Boltzmann Machine



What We've Covered So Far that is Relevant to Boltzmann Machines

Simulated Annealing

- The thermodynamic basis of SA
- The stationary and transition probabilities associated with SA
- Examples of combinatorial optimization problems

Hopfield Recurrent Neural Networks

- How to define the weight matrix.
- Examined their capability to recall/remember exemplar patterns.
- Examined their memory capacity.



In This Module We Will Cover

The Boltzmann Machine

- A stochastic version of the Hopfield Network
- Consensus/Energy function
- Using the sigmoid function as the activation function
- Briefly discussed the training formulae
- Look at calculating the respective stationary probabilities of various configurations (sets of states)
- Look at how to modify the weights so as to obtain desired stationary probabilities
- Examples



Recall the Hopfield Network

- Memory capacity about 11% of the length of exemplars. E.g., an exemplar vector with 100 elements could store approximately 11 exemplars with very high accuracy.
- Accuracy based on statistical independence of the exemplars, and a 3σ standard deviation.
 - This means a near certainty ($\geq 99\%$) for accurately determining the most likely exemplar that an input vector represents.
 - Remember, the input vector is an exemplar perturbed by some noise.
 - Variance of the ‘noise’ associated with inputs is approximately $(n - 1)(P - 1)$



A Tradeoff!

- Can we sacrifice some certainty associated with memory recall/completion for **greater memory capacity?**
- Can't be based on issue of 'noise' --- once an input is provided, it is known/certain.
- Don't want it to be based on statistical independence --- relationship among exemplars is not the issue insofar as 'certainty' is concerned.



The ‘Uncertainty’

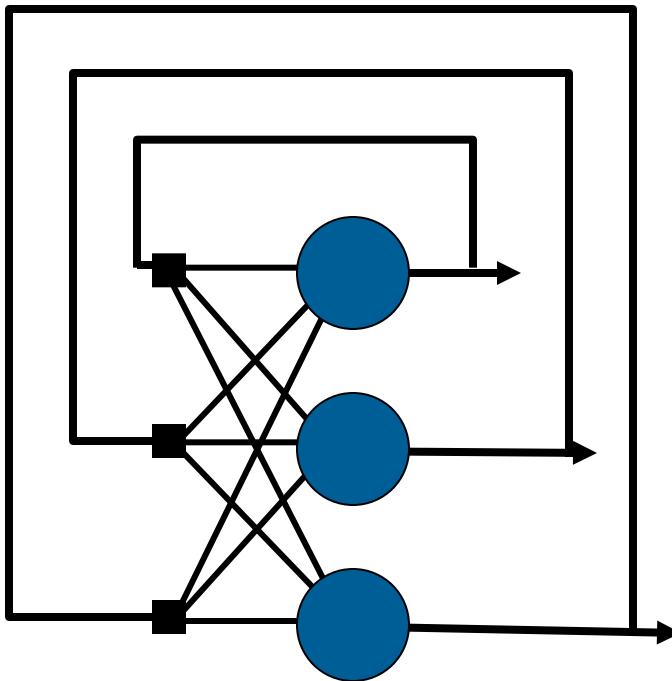
- Want to let the network ‘run’ and hopefully arrive at the correct exemplar.
- Could allow some probability the network will ‘arrive’ at the wrong exemplar.
- Allow the network to make some mistakes.
- How?

Let the nodes take on random states!



Recurrent Network Topology Reprise

Let's view the network a bit differently.



Exemplar:

(1, -1, -1)

What is the weight from
Node 1 to Node 2?



Boltzmann/Hopfield Comparisons

- Very similar in architecture.
- Boltzmann uses stochastic methods for updating node states.
- Hopfield uses bipolar state values.
- Boltzmann typically uses binary state values.



Activity Functions and Energy

- Asynchronous update of neuron activations.
- Cell activity S_i is computed (here without a bias term):

$$S_i = \sum_j w_{ij} x_j$$



The Hecht-Nielsen Function

Define the energy function E

$$E = - \sum_{i < j} w_{ij} x_i x_j + \sum_i \theta_i x_i$$

$$E = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{ij} x_i x_j + \sum_i \theta_i x_i$$



Energy → Consensus

Minimize Energy

$$E = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{ij} x_i x_j + \sum_i \theta_i x_i$$

Maximize Consensus

$$C = \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{ij} x_i x_j - \sum_i \theta_i x_i$$

Energy and Consensus values are
additive inverses of one another!



An Optimization Problem ala Simulated Annealing

- Minimize energy or Maximize consensus.
- Change the state of a node to modify a candidate energy/consensus function value.
 - Means changing it from $0 \rightarrow 1$ or $1 \rightarrow 0$.
- Accept new configuration probabilistically as in SA.



Calculating ΔE

Recall:

$$E = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{ij} x_i x_j + \sum_i \theta_i x_i$$

WLG:

$$E = -\frac{1}{2} \sum_{\substack{i=1 \\ i \neq k}}^n \sum_{\substack{j=1 \\ j \neq k}}^n w_{ij} x_i x_j + \sum_{\substack{i=1 \\ i \neq k}}^n \theta_i x_i - \frac{1}{2} \sum_{j=1}^n w_{kj} x_k x_j - \frac{1}{2} \sum_{i=1}^n w_{ik} x_i x_k + \theta_k x_k$$



Calculating ΔE

$$E_{\text{cand}} = -\frac{1}{2} \sum_{\substack{i=1 \\ i \neq k}}^n \sum_{\substack{j=1 \\ j \neq k}}^n w_{ij} x_i x_j + \sum_{\substack{i=1 \\ i \neq k}}^n \theta_i x_i - \frac{1}{2} \sum_{j=1}^n w_{kj} x'_k x_j - \frac{1}{2} \sum_{i=1}^n w_{ik} x_i x'_k + \theta_k x'_k$$

$$E_{\text{cur}} = -\frac{1}{2} \sum_{\substack{i=1 \\ i \neq k}}^n \sum_{\substack{j=1 \\ j \neq k}}^n w_{ij} x_i x_j + \sum_{\substack{i=1 \\ i \neq k}}^n \theta_i x_i - \frac{1}{2} \sum_{j=1}^n w_{kj} x_k x_j - \frac{1}{2} \sum_{i=1}^n w_{ik} x_i x_k + \theta_k x_k$$

$$E_{\text{cand}} = -\frac{1}{2} \sum_{\substack{i=1 \\ i \neq k}}^n \sum_{\substack{j=1 \\ j \neq k}}^n w_{ij} x_i x_j + \sum_{\substack{i=1 \\ i \neq k}}^n \theta_i x_i - x'_k \sum_{i=1}^n w_{ik} x_i + \theta_k x'_k$$

$$E_{\text{cur}} = -\frac{1}{2} \sum_{\substack{i=1 \\ i \neq k}}^n \sum_{\substack{j=1 \\ j \neq k}}^n w_{ij} x_i x_j + \sum_{\substack{i=1 \\ i \neq k}}^n \theta_i x_i - x_k \sum_{i=1}^n w_{ik} x_i + \theta_k x_k$$



Calculating ΔE

$$E_{\text{cand}} = -\frac{1}{2} \sum_{\substack{i=1 \\ i \neq k}}^n \sum_{\substack{j=1 \\ j \neq k}}^n w_{ij} x_i x_j + \sum_{\substack{i=1 \\ i \neq k}}^n \theta_i x_i - x'_k \sum_{i=1}^n w_{ik} x_i + \theta_k x'_k$$

$$E_{\text{cur}} = -\frac{1}{2} \sum_{\substack{i=1 \\ i \neq k}}^n \sum_{\substack{j=1 \\ j \neq k}}^n w_{ij} x_i x_j + \sum_{\substack{i=1 \\ i \neq k}}^n \theta_i x_i - x_k \sum_{i=1}^n w_{ik} x_i + \theta_k x_k$$

$$\Delta E = E_{\text{cand}} - E_{\text{cur}}$$



Calculating ΔE

$$E_{\text{cand}} - E_{\text{cur}} = -x'_k \sum_{i=1}^n w_{ik} x_i + \theta_k x'_k - -x_k \sum_{i=1}^n w_{ik} x_i - \theta_k x_k$$

$$\Delta E = -x'_k \sum_{i=1}^n w_{ik} x_i + \theta_k x'_k + x_k \sum_{i=1}^n w_{ik} x_i - \theta_k x_k$$

$$= x'_k \left[- \sum_{i=1}^n w_{ik} x_i + \theta_k \right] + x_k \left[\sum_{i=1}^n w_{ik} x_i - \theta_k \right]$$

Change in state
of Node k

$$= x'_k \left[- \sum_{i=1}^n w_{ik} x_i + \theta_k \right] - x_k \left[- \sum_{i=1}^n w_{ik} x_i + \theta_k \right]$$

$$= (x'_k - x_k) \left[- \sum_{i=1}^n w_{ik} x_i + \theta_k \right]$$



Calculating ΔC

Similarly for the consensus value. Thus,

$$\begin{aligned}
 C_{\text{cand}} - C_{\text{cur}} &= x'_k \sum_{i=1}^n w_{ik} x_i - \theta_k x'_k - x_k \sum_{i=1}^n w_{ik} x_i + \theta_k x_k \\
 \Delta C &= x'_k \sum_{i=1}^n w_{ik} x_i + \theta_k x'_k - x_k \sum_{i=1}^n w_{ik} x_i + \theta_k x_k \\
 &= x'_k \left[\sum_{i=1}^n w_{ik} x_i + \theta_k \right] - x_k \left[\sum_{i=1}^n w_{ik} x_i + \theta_k \right] \\
 &= x'_k \left[\sum_{i=1}^n w_{ik} x_i + \theta_k \right] - x_k \left[\sum_{i=1}^n w_{ik} x_i + \theta_k \right] \\
 &= (x'_k - x_k) \left[\sum_{i=1}^n w_{ik} x_i + \theta_k \right]
 \end{aligned}$$



Relating the Change of State to Probability

$$\pi_i(t) = \frac{e^{-E_i/t}}{\sum_i e^{-E_i/t}}$$

$$\begin{aligned} \frac{\pi_i(t)}{\pi_{i'}(t)} &= \frac{e^{-E_i/t}}{\sum_i e^{-E_i/t}} \\ &= \frac{e^{-E_i/t}}{e^{-E_{i'}/t}} = e^{(E_{i'} - E_i)/t} = e^{\Delta E/t} \end{aligned}$$

Now, taking the logarithm of both sides we get ...



Relating the Change of State to Probability

$$\ln\left(\frac{\pi_i(t)}{\pi_{i'}(t)}\right) = \ln e^{\Delta E/t} = \frac{\Delta E}{t}$$

$$\ln \pi_i(t) - \ln \pi_{i'}(t) = \frac{\Delta E}{t}$$

$$\ln(\Pr\{x_k = 1\}) - \ln(\Pr\{x_k = 0\}) = \frac{\Delta E}{t}$$

$$\ln(\Pr\{x_k = 1\}) - \ln(1 - \Pr\{x_k = 1\}) = \frac{\Delta E}{t}$$



Relating the Change of State to Probability

$$\ln(\Pr\{x_k = 1\}) - \ln(1 - \Pr\{x_k = 1\}) = \frac{\Delta E}{t}$$

$$\ln\left(\frac{\Pr\{x_k = 1\}}{1 - \Pr\{x_k = 1\}}\right) = \frac{\Delta E}{t}$$

$$\ln\left(\frac{1 - \Pr\{x_k = 1\}}{\Pr\{x_k = 1\}}\right) = \frac{-\Delta E}{t}$$



Relating the Change of State to Probability

$$\ln\left(\frac{1 - \Pr\{x_k = 1\}}{\Pr\{x_k = 1\}}\right) = \frac{-\Delta E}{t}$$

$$\frac{1 - \Pr\{x_k = 1\}}{\Pr\{x_k = 1\}} = e^{-\Delta E/t}$$

$$\frac{1}{\Pr\{x_k = 1\}} - 1 = e^{-\Delta E/t}$$

$$\frac{1}{\Pr\{x_k = 1\}} = 1 + e^{-\Delta E/t}$$

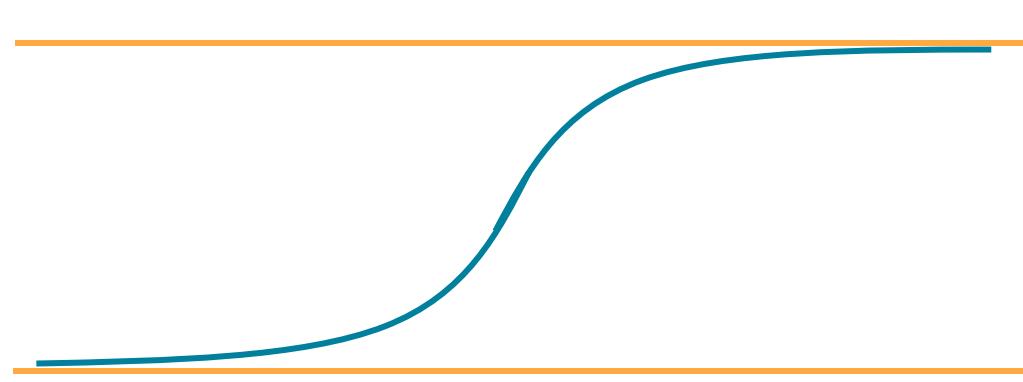
$$\Pr\{x_k = 1\} = \frac{1}{1 + e^{-\Delta E/t}}$$



Dynamics

Recall the Sigmoid activation function

$$\frac{1}{1 + e^{-S_i/T}}$$



We're dealing with stochastic neurons.
What does this curve remind you of?



Dynamics

- Yes, a probability distribution function (monotonically increasing to 1).
- Set cell activation (state) according to:

$$x_i = \begin{cases} 1 & \text{w/prob } p_i = \frac{1}{1 + e^{-S_i/T}} \\ 0 & \text{w/prob } 1 - p_i \end{cases}$$

At high temperatures, what is the probability of $x_i = 1$?



Summary

- Each node is update asynchronously and probabilistically.
- The temperature is lowered to minimize the energy value of the network or maximize the consensus value of the network as the case may be.
- Since the node states are probabilistic, **all information is encoded in the weights!**



Introduction to Neural Networks

**Johns Hopkins University
Engineering for Professionals Program**

605-447/625-438

Dr. Mark Fleischer

Copyright 2014 by Mark Fleischer

Module 10.2: Training The Boltzmann Machine



What We've Covered So Far

We derived a formula for $\Delta E/\Delta C$ and showed that it is equivalent to the activity function.

$$\Delta E_i = - \sum_j w_{ij} x_j$$

We derived the probability value of a node's state to be equal to the sigmoid function.

$$\Pr\{X_i = 1\} = \frac{1}{1 + e^{-\Delta E_i/t}} \quad \longrightarrow \quad \Pr\{X_i = 1\} = \frac{1}{1 + e^{-S_i/t}}$$



Modalities for Applications

- BM can learn distributions ala supervised learning.
 - by updating weights
- BM can solve constrained optimization problems
 - some of the weights are established by the nature of the problem.
 - using simulated annealing approaches.



Behavior

1. Set T to an initially high temperature value.
2. Randomly select a free-running node i .
3. Compute activation function S_i .
4. Set node i to 1 or 0 according to acceptance function.
5. Reduce T .
6. Goto step 2 if not at equilibrium.

This looks like an annealing process. And it is!

So what are we minimizing?



Dynamics

Goal is to minimize E or maximize C .

Modify the network and in effect run a simulated annealing algorithm on it.

This approach is appropriate when the weights have been determined by the nature of the problem. *E.g., COPs.*

Other problems require you to **train the weights** so that the probability distribution for the configurations match a given distribution.



Training Boltzmann Machines

- The Boltzmann distribution is the probability distribution for different configurations.
- The BM ‘learns’ the distributions (not specific targets)!



Training

- If we want to ‘train’ the network, what does this mean?
- We want the relative frequency of configurations to match those of a training set.
- The relative frequency of a configuration is affected by the weight connections.



Training

- Thus, we must find a way to determine how to update weights based on differences between the actual frequency of configurations and ‘desired’ frequency of configurations.



Probability of Configurations

For any two system configurations

$$\alpha = [00110100]$$
$$\beta = [10011010]$$

$$\frac{P_\alpha}{P_\beta} = e^{- (E_\alpha - E_\beta) / T}$$



Training

- Slow, and arduous.
- Intimately connected with entropy.

Define

p_α = The probability of state α based on training examples.

p'_α = The probability of state α at equilibrium.

$$G = \sum_{\alpha} p_{\alpha} \ln\left(\frac{p_{\alpha}}{p'_{\alpha}}\right)$$

Ackley et al. 1986



Value of G

If there is a mismatch in probabilities, some values of

$$\left(\frac{p_a}{p'_a} \right)$$

in the terms in the summation of G will
of necessity be greater than AND less than 1.
Why?



Training

Use function G for gradient descent formulation:

$$\Delta w_{ij} = -\eta \frac{\partial G}{\partial w_{ij}} \text{ where}$$

$$\frac{\partial G}{\partial w_{ij}} = -\frac{1}{T} (p_{ij} - p'_{ij})$$

Where p_{ij} is the probability that x_i and $x_j = 1$ when system is clamped (according to input distribution or training set). Similarly for free-running.

Estimating these values requires a great deal of computation.



Training

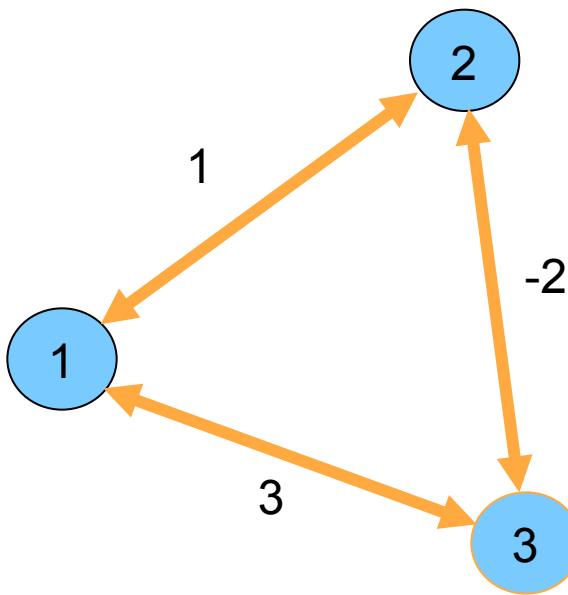
Sometimes you'll see this written

$$\Delta w_{ij} = -\eta \frac{\partial G}{\partial w_{ij}} \text{ where}$$

$$\frac{\partial G}{\partial w_{ij}} = -\frac{1}{T} \left(p_{ij} - p'_{ij} \right) = -\frac{1}{T} \left(\langle s_i s_j \rangle - \langle s'_i s'_j \rangle \right)$$



A Simple Example



What are the energy values associated with each ‘configuration’ ?



A Simple Example

- We have 8 possible configurations,
- Each has an energy value,
- Each has a probability based on the Boltzmann Distribution at a given temperature
- For a small problem, we can cheat in order to highlight our general approach



A Simple Example

- For each configuration, we calculate:
 - The energy/consensus function.
 - The Boltzmann Distribution
 - Calculate the numerators, sum them up to determine the normalizing Partition Function value, then divide into the numerators to determine the steady-state probability
 - In the real-world, we would run the algorithm at a fixed temperature to estimate the steady-state probabilities



A Simple Example

	Temperature = 10			Energy	Boltzmann Numerator
Weights	x1	x2	x3		
Config	1	0	0	0	0
2	0	0	1	0	0
3	0	1	0	0	0
4	0	1	1	2	1
5	1	0	0	0	1
6	1	0	1	1	1
7	1	1	0	2	1
8	1	1	1	3	1



$$E = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{ij} x_i x_j + \sum_i \theta_i x_i$$



A Simple Example

- Now we can examine the actual, steady-state probabilities, versus the “training set” or the “desired probabilities”.
- Let’s look at a spread sheet where we calculate the values for G and the gradient of G . First, recall



Learning Equations

Akin to an error function:

$$G = \sum_{\alpha} p_{\alpha} \ln\left(\frac{p_{\alpha}}{p'_{\alpha}}\right)$$

Method of Steepest Descent:

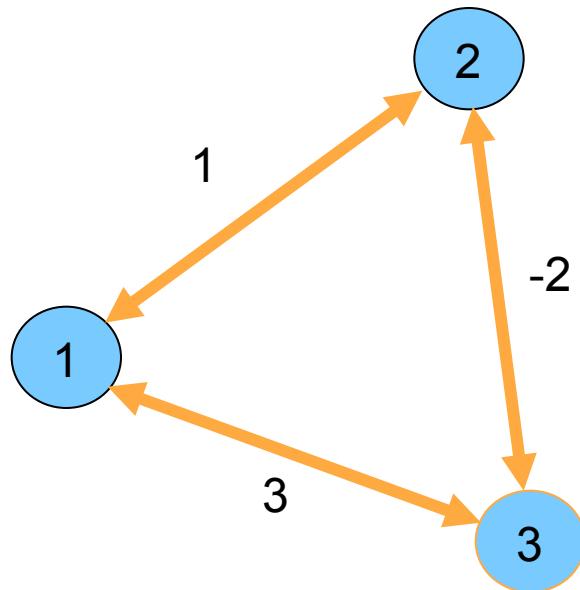
$$\Delta w_{ij} = -\eta \frac{\partial G}{\partial w_{ij}} \text{ where}$$

$$\frac{\partial G}{\partial w_{ij}} = -\frac{1}{T} (p_{ij} - p'_{ij})$$



An Example

Let's say we have the following network with the indicated weights and using Temperature = 1:





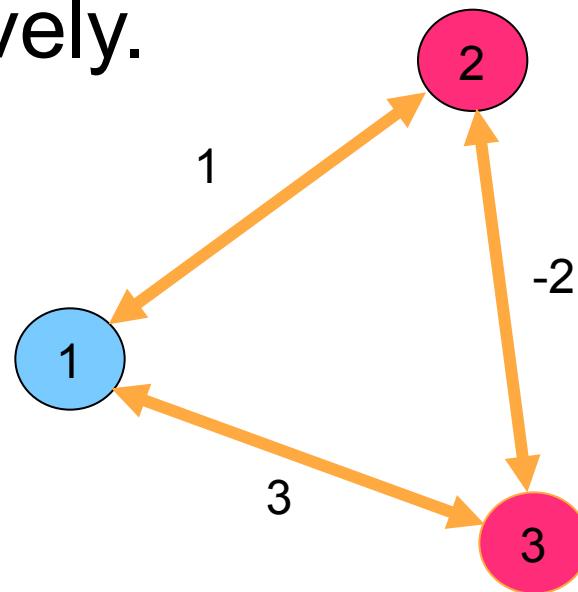
Step 1:

- Assign random initial states:
- Use a 0-1 pseudo-random number generator.
- If number $0 \leq r < 0.5$, assign that node a 0 state.
- If number $0.5 \leq r < 1$, assign that node a 1 state.



Step 1, cont.

- We have 3 random numbers:
- 0.233691, .622338, 901122
- Therefore we assign values of 0, 1 and 1 to nodes 1, 2 and 3 respectively.





Step 2:

- Select at random, a node to update.
- We'll do this by using a random number generator:
- Since there are 3 nodes, we do the following:
- Obtain a uniform, 0-1 pseudo-random number



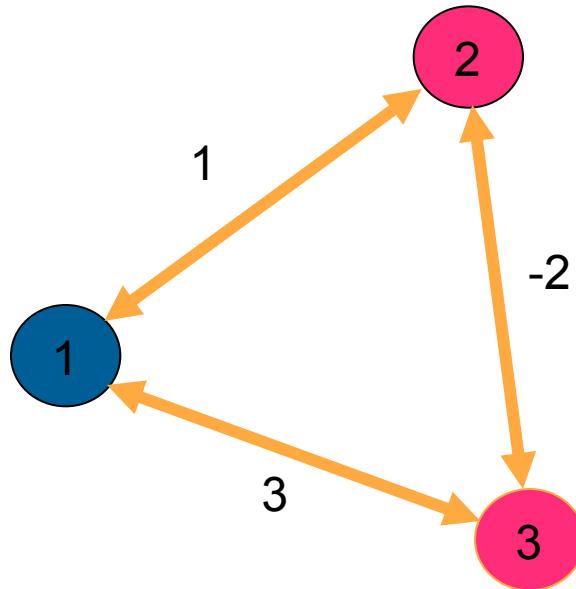
Select a Node

- Get a pseudo-random 0-1 number:
0.4468921
- If number $0 \leq r < 0.33333$, select Node 1
- If number $0.33333 \leq r < 0.66666$, select Node 2.
- If number $0.66666 \leq r < 1$, select Node 3



Step 2: Calculate the Activity

- We selected node 2.
- $S_2 = 0 \cdot 1 + -2 \cdot 1 = -2$





Step 3: Assign new state

Given the Activity value, use the sigmoid function to determine a probability of assigning a 0 or 1.

$$x_i = \begin{cases} 1 & \text{w/prob } p_i = \frac{1}{1 + e^{-S_i/T}} \\ 0 & \text{w/prob } 1 - p_i \end{cases}$$

Using this function,

$$\Pr\{x_2 = 1\} = 1/(1 + \exp\{2\}) = 0.11920.$$

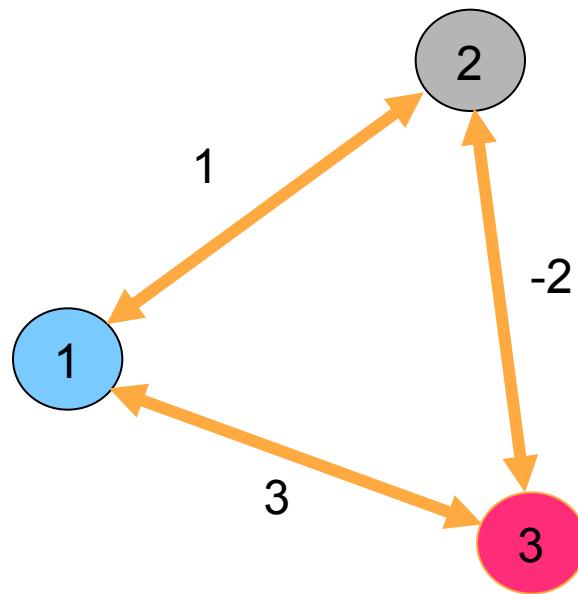


Step 3, cont.

- Using a 0-1 pseudo-random number generator,
- assigning a 1 with prob. = 0.11920
- Means if $r \leq 0.11920$, assign 1, otherwise, assign 0.
- $r = 0.314721$, therefore assign a 0.



Our New Network Configuration and we start the process over.





Embellishments

- We could start with a high initial temperature and random initial configuration
- Do many iterations at that temperature.
- Then, lower the temperature and again, do many iterations,
- Then, lower the temperature and again, do many iterations,...
- Eventually, stop.



Or, we could leave the temperature fixed

- In this case, we can compute that relative frequency of each configuration.
- Requires a great deal of computation... many iterations at a given temperature.



Summary

- Showed how the activity function affects the probabilistic state of a node.
- Showed how to ‘anneal’ a network
 - Randomly assigning initial states
 - Stochastically updating states
- Showed how to induce these steady-state probabilities to match a given distribution.