# Harvey Mudd College

## CS 152
## Neural Networks
## Fall 2008

Bob Keller, Professor
keller@cs.hmc.edu
621-8483
Olin 1253

# Office Hours (1253 Olin):

- MTWTh 2:45-4:00,
- By drop-in (as available)
- By appointment:
  - email keller@cs.hmc.edu
  - AIM: MuddProf
  - phone 621-8483

# Text

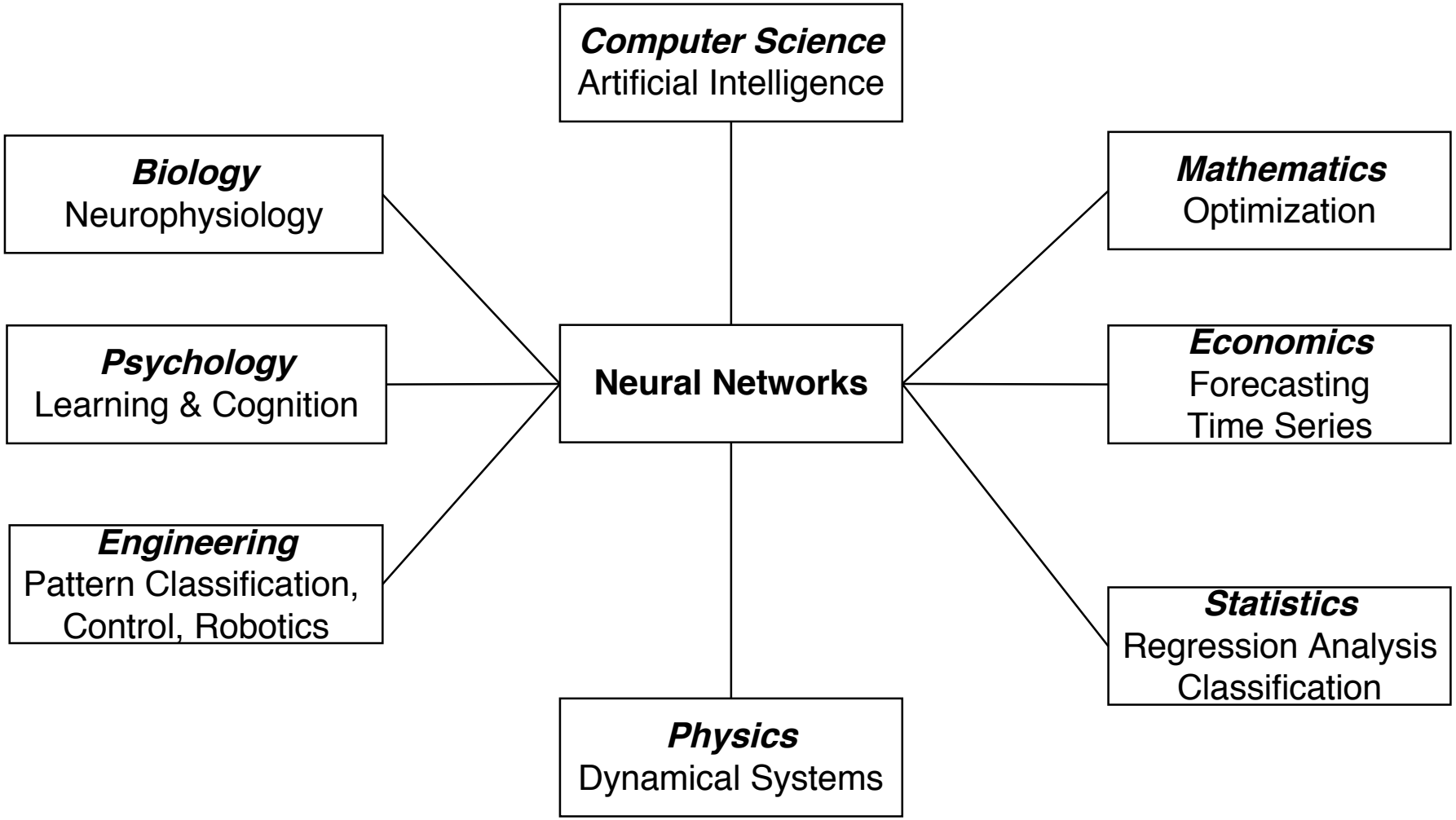
Raul Rojas, ***Neural Networks - A Systematic Introduction***, Springer-Verlag, Berlin, New-York, 1996 (502 pages,350 illustrations). This book is no longer in print. However, it may be obtained legally on the web in pdf form:
http://www.inf.fu-berlin.de/inst/ag-ki/rojas_home/documents/1996/NeuralNetworks/neuron.pdf
Other material will be announced as needed.

# Course Outline

Please Refer to Web Page for details:

`http://www.cs.hmc.edu/courses/2008/fall/cs152`

# Neural Networks: an Eclectic Discipline



# Biological Intelligence

- Intelligence, the ability to make decisions based upon input from the environment.

- Intelligence is realized by a ***network*** of ***neurons***, for example the brain and the attendant sensory and motor neurons.

# "Neurons R Us"

- Not only our intelligence, but all aspects of our behavior, are the result of neural activity:
  - emotions
  - memory
  - reflexes
  - likes and dislikes
  - habits
  - addictions

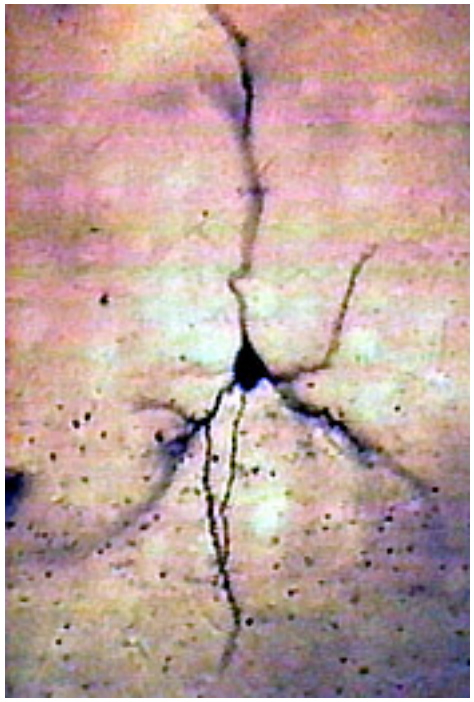
# Some Approaches to Artificial Intelligence

- Reverse Engineering of Biology
  - Understand real neurons well enough to model
  - ***Simulate*** neural behavior, including learning aspects
- Artificial Neural Networks
  - Develop a parameterized model for a class of problems
  - ***Learn*** the parameters
- Simulated Evolution
  - Provide basic evolutionary mechanism for neurons
  - ***Evolve*** the parameters

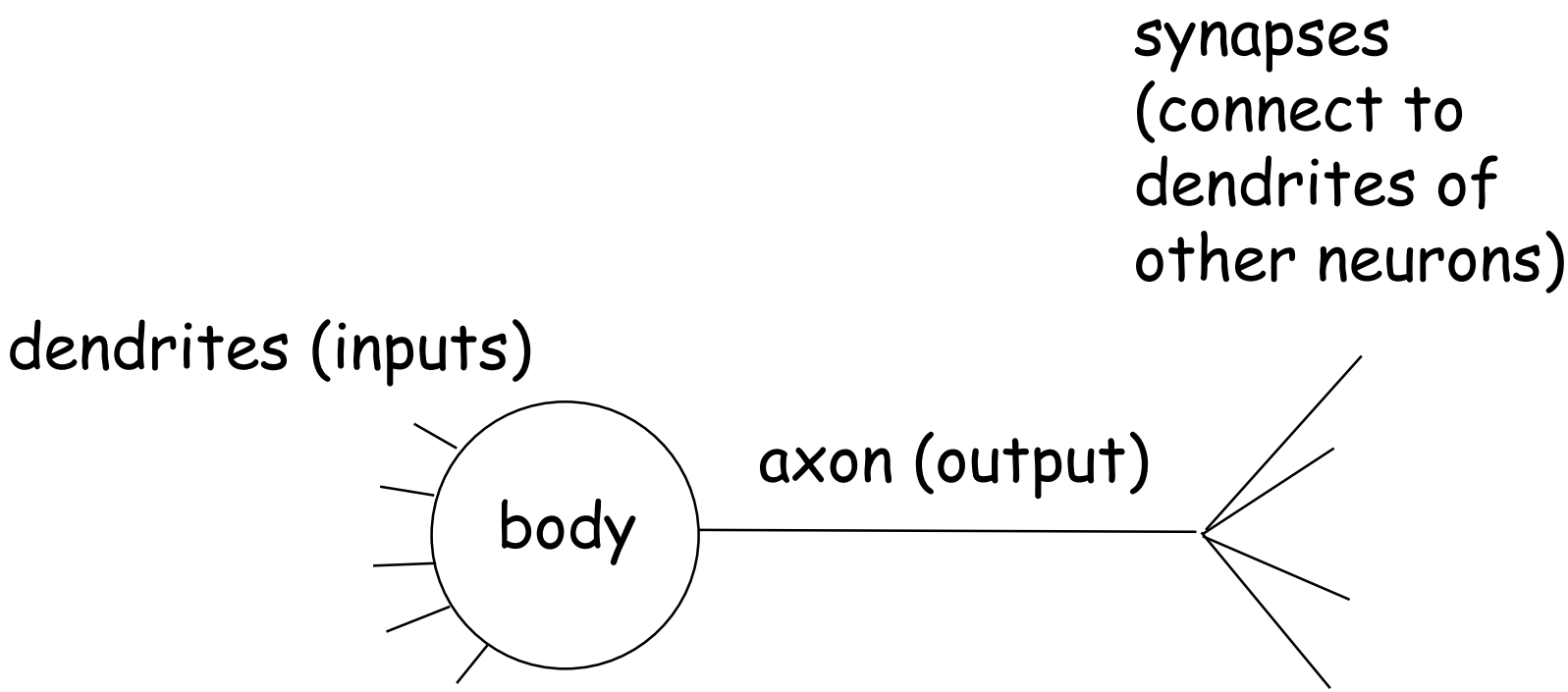
# Fundamental Problems for a Given Neural Model

- How to ***represent*** information?
- How to characterize the ***computational capability*** of the model?
- How to achieve ***learning*** in the model?

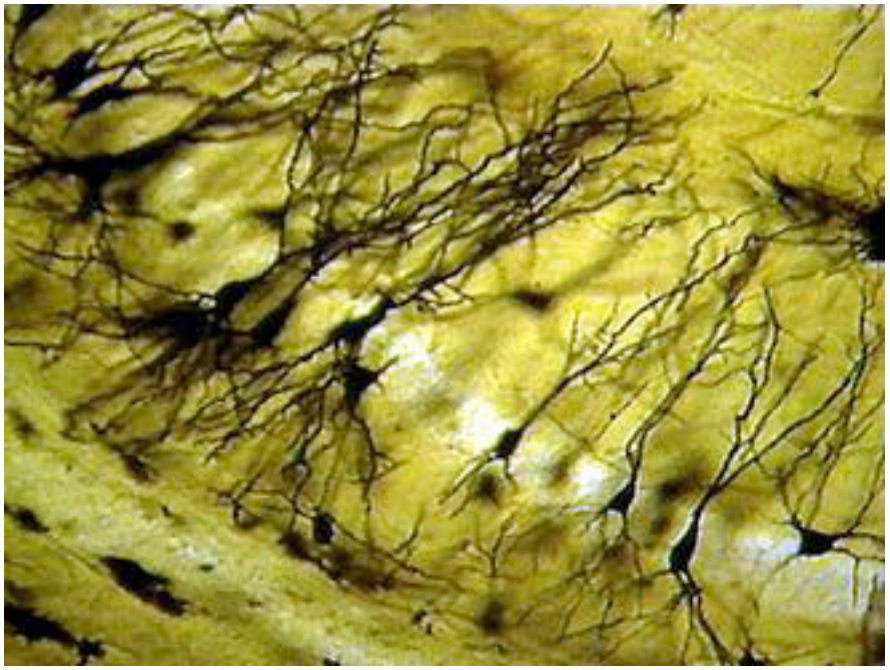
# Photomicrograph of one neural cell (from cerebral cortex)



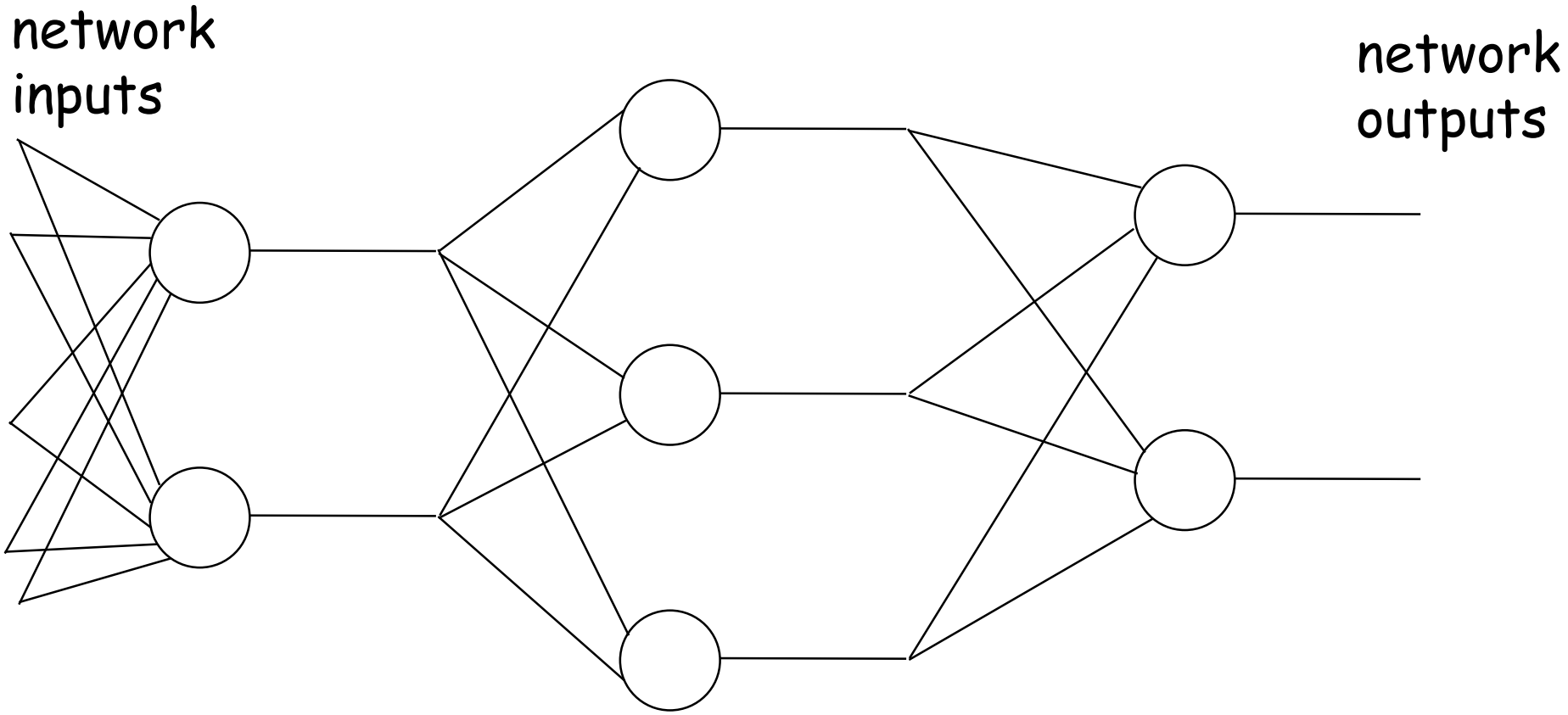
# Schematic of One Neuron



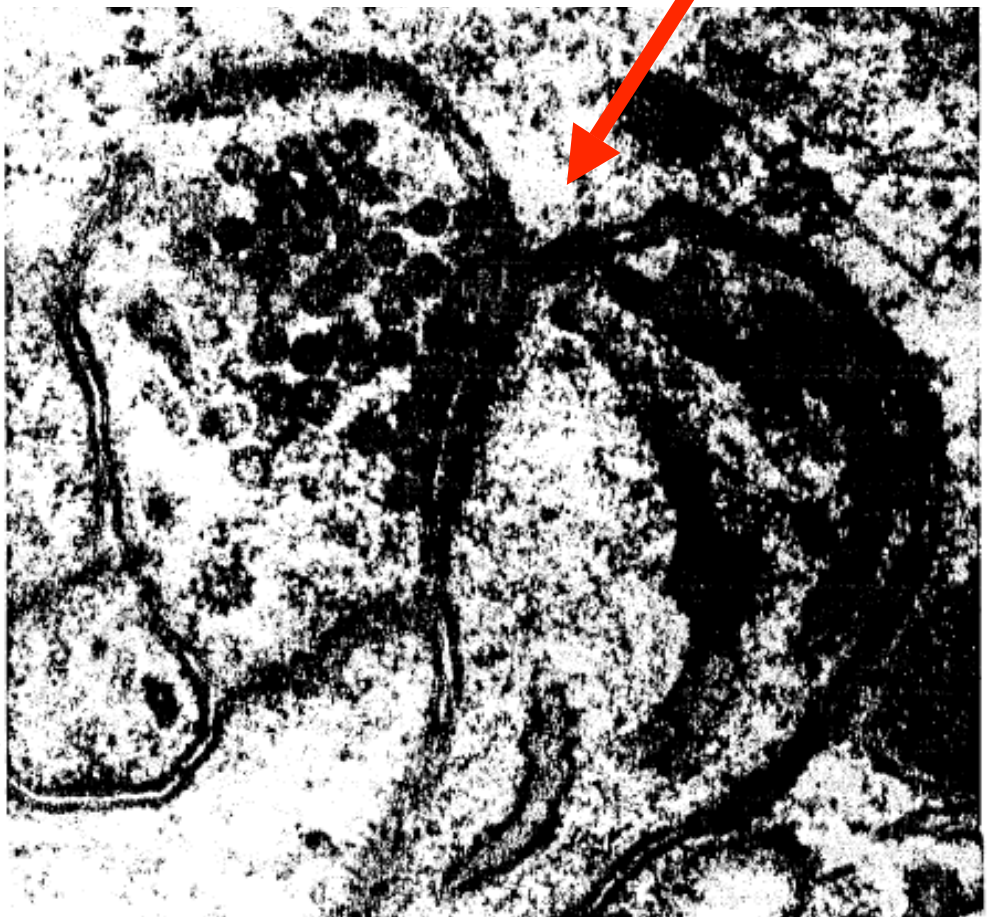
# Photomicrograph of **network** of neural cells (from the hippocampus region of the brain)



# Schematic of a Neural Network

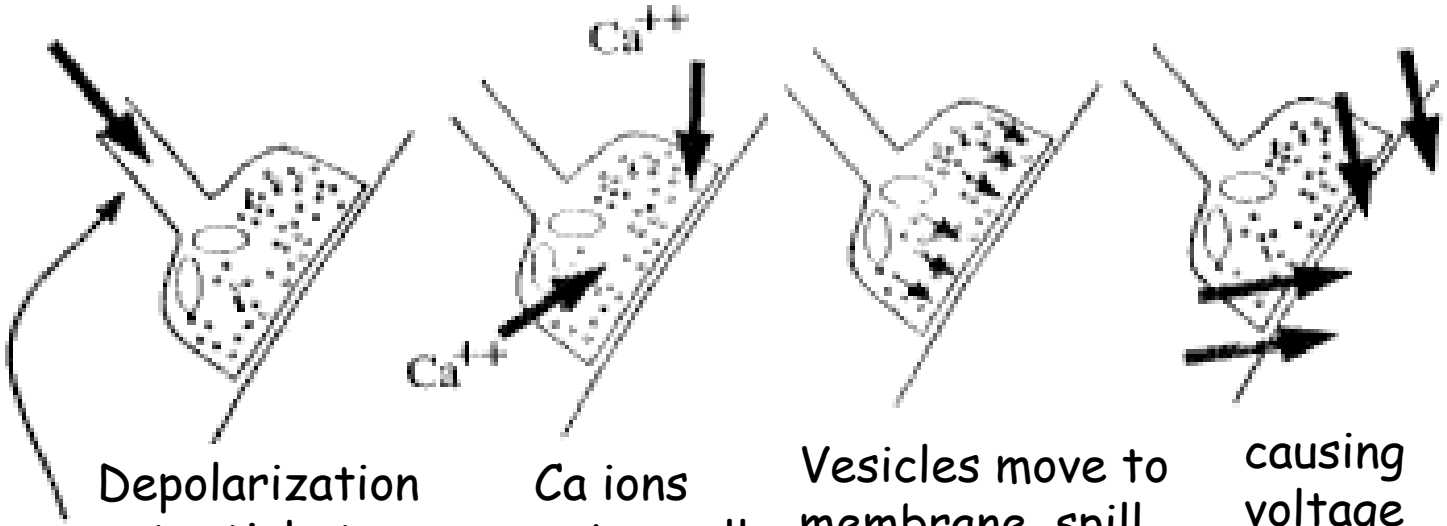


# Electronmicrograph of one synapse/dendrite connection




reference: Irwin B. Levitan and Leonard K. Kaczmarek, The Neuron, Oxford University Press, 1991.

# Ionic Neurotransmitter Reaction




Depolarization potential at synapse

Ca ions enter cell

Vesicles move to membrane, spill contents into cleft

causing voltage change on dendrite of next neuron

reference: James A. Anderson, An Introduction to Neural Networks, MIT Press, 1955.

# Biological Neuron Terminology

- **neurotransmitters**: molecules that traverse from synapse to dendrite through ion diffusion.
- **spiking**: abrupt change of output voltage
- **depolarization**: change in net input voltage toward a threshold value, at which it will "spike"
- **action potential**: the voltage change produced when the neuron spikes
- **refractory period**: period immediately after firing during which neuron is temporarily not firable

# Leaky-Integrator Behavior

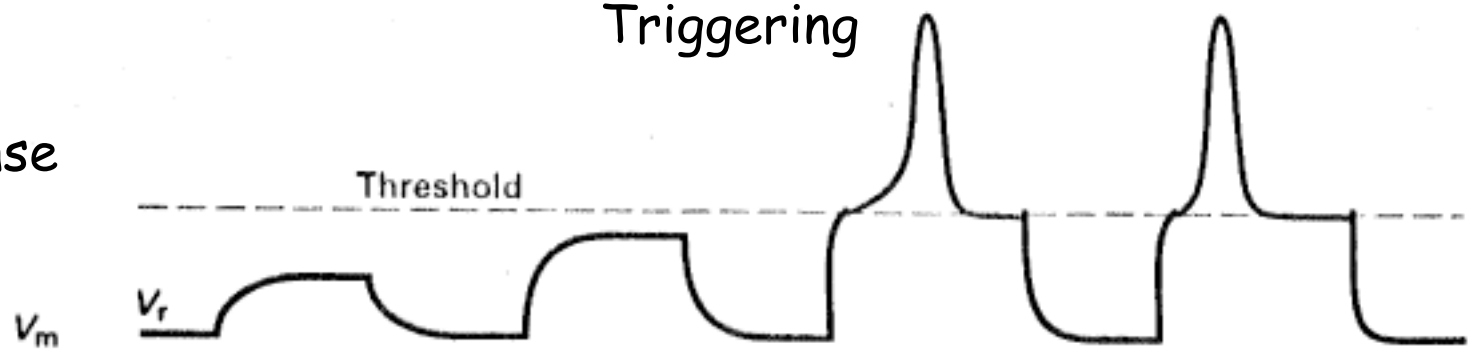
- The neuron acts like an **integrator** in accumulating the effect of input spikes.
- The accumulation cannot go on forever:
  - If the accumulation is sufficiently high in a short time period, the neuron spikes.
  - Over a longer time period without spiking, the neuron gradually **leaks** the built-up potential.

# Triggering phenomenon

Net stimulus (integrated inputs)


Triggering

Response



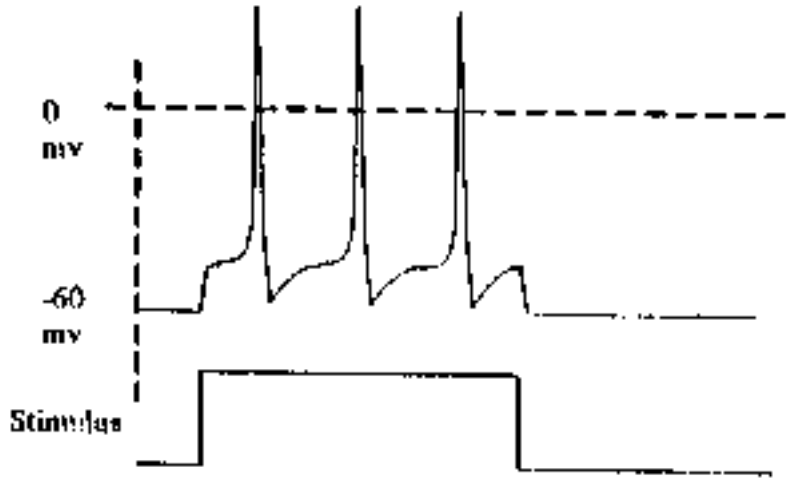

reference: Irwin B. Levitan and Leonard K. Kaczmarek, The Neuron, Oxford University Press, 1991.

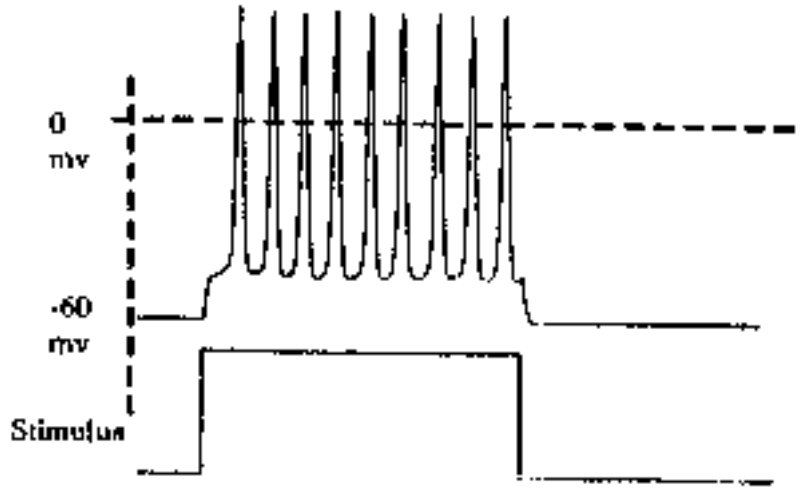
# Intensity

- Because of the triggering behavior, the neuron indicates intensity of stimulation by the ***frequency*** of spikes, not amplitude.

- Because of the necessary refractory period, there is a maximum **saturation** frequency at which the neuron can operate.

# Spiking Frequency of a Neuron as a Function of Integrated Stimulus Magnitude

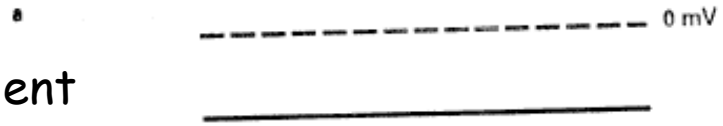



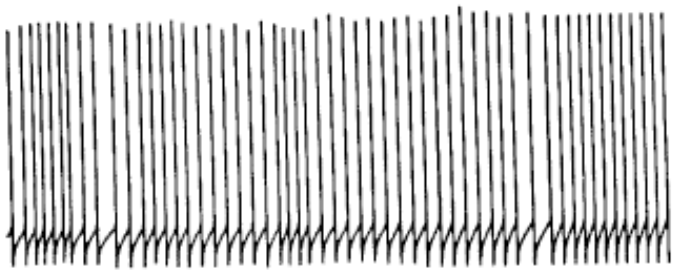

Larger stimulus implies higher frequency of output spiking

reference: James A. Anderson, An Introduction to Neural Networks, MIT Press, 1955.

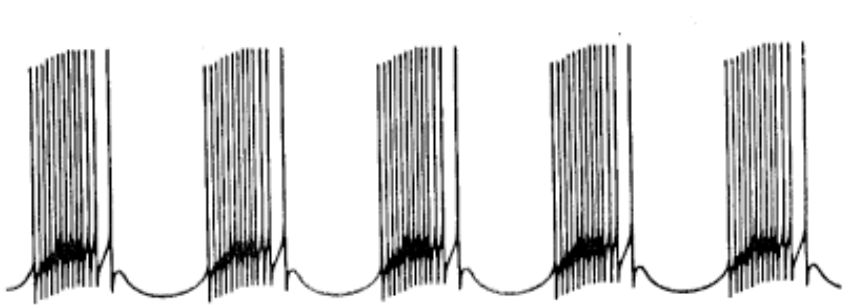
# Various Spiking Patterns




Silent




Beating pattern




Bursting pattern

reference: Irwin B. Levitan and Leonard K. Kaczmarek, The Neuron, Oxford University Press, 1991.

# Information Signal Encoding

- The flow of information from one neuron to another is typically based on the **rate of spiking**. The higher the rate, the more active the neuron.

- We abstract this rate into a **single number**, *as if* transmitted in a single instant.

# Signal Abstractions

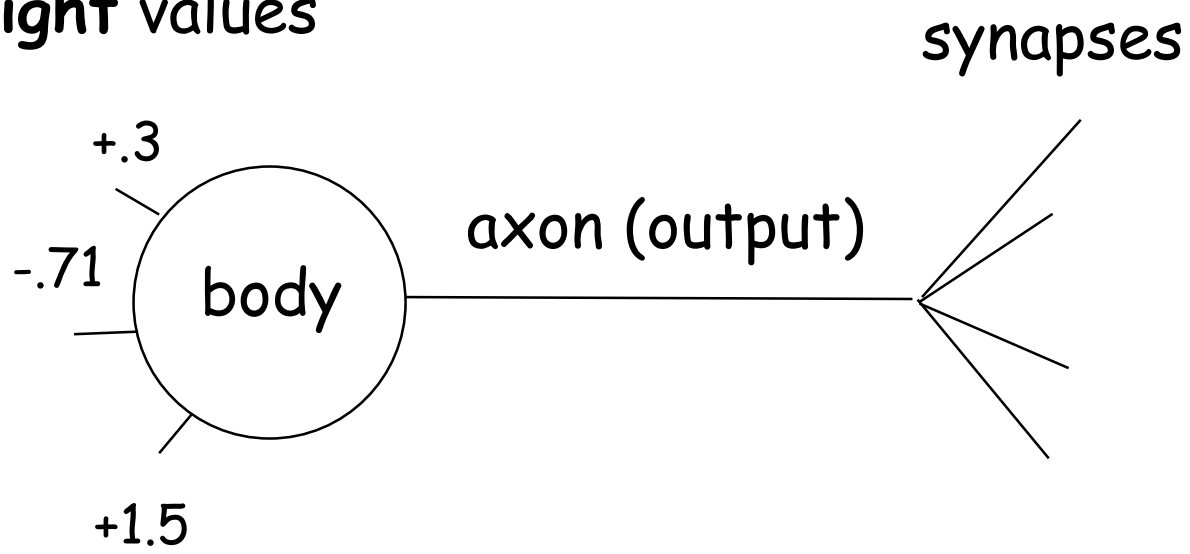
- We've already mentioned that signals can be abstracted into a single real number.
- We sometimes further abstract into a two-valued set, such as {-1, +1} or {0, 1}, depending on the intended application.

# Synaptic Strength or Weight

- The connection from one neuron to another has an associated efficiency, typically called the "synaptic strength" or simply "**weight**".
- It usually is represented a real number and can be positive or negative.
- Although weights should be ascribed to connections, it is common to associate them with the input side of the neuron.

# Neuron schematic with weights

dendrites (input)
with **weight** values

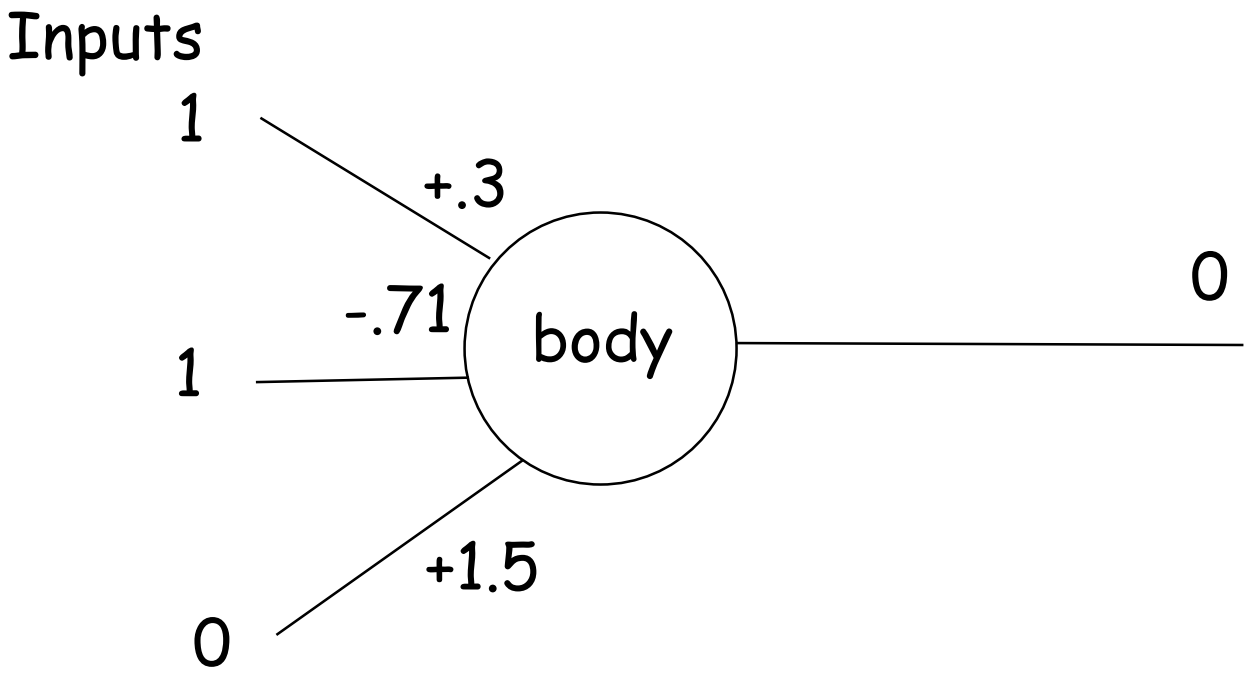


# Threshold Logic

- Early efforts at modeling neural networks used threshold logic, and are still valid for some types of applications.
  - The input values are discrete, say {0, 1}.
  - The weighted sum minus the threshold is called the "activation" or "net" value.
  - The neuron **fires** if activation value is above a threshold value associated with the neuron.

# Threshold Logic Behavior

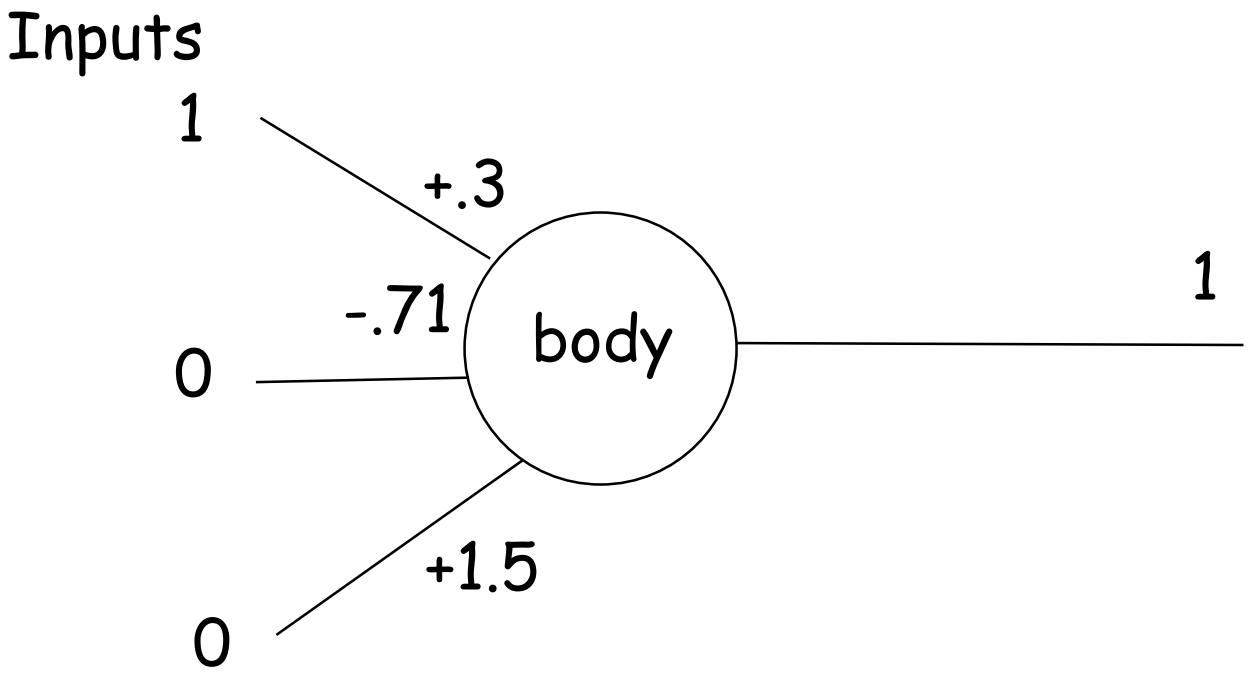
Say the threshold happens to be -0.2.




activation = 1*.3 + 1*(-.71) + 0 * 1.5 - (-0.2) = -0.21
activation < 0 so the neuron **does not** fire (0)

# Threshold Logic Behavior

Say the threshold happens to be -0.2.




activation = 1*.3 + 0*(-.71) + 0 * 1.5 - (-0.2) = 0.5
activation > 0 so the neuron **does** fire (1)

# Qualitative Weights

- Inputs with positive weights are called "excitatory".
- Inputs with negative weights are called "inhibitory".

# Early NN Chronology

- 1943: McCulloch and Pitts (U of Chicago), Linear Threshold Logic Gate models
- 1949: Hebb, proposed Learning principle (Yerkes Primate Research Center)
- 1957: Rosenblatt's Perceptron (Cornell Aeronautical)
- 1960 Widrow & Hoff's Adaline (Stanford EE)
- 1969: Minsky & Papert (MIT CS), Limitations of perceptrons

# Perceptrons

## Primitive Artificial Neurons

# Rosenblatt's Perceptron, 1957

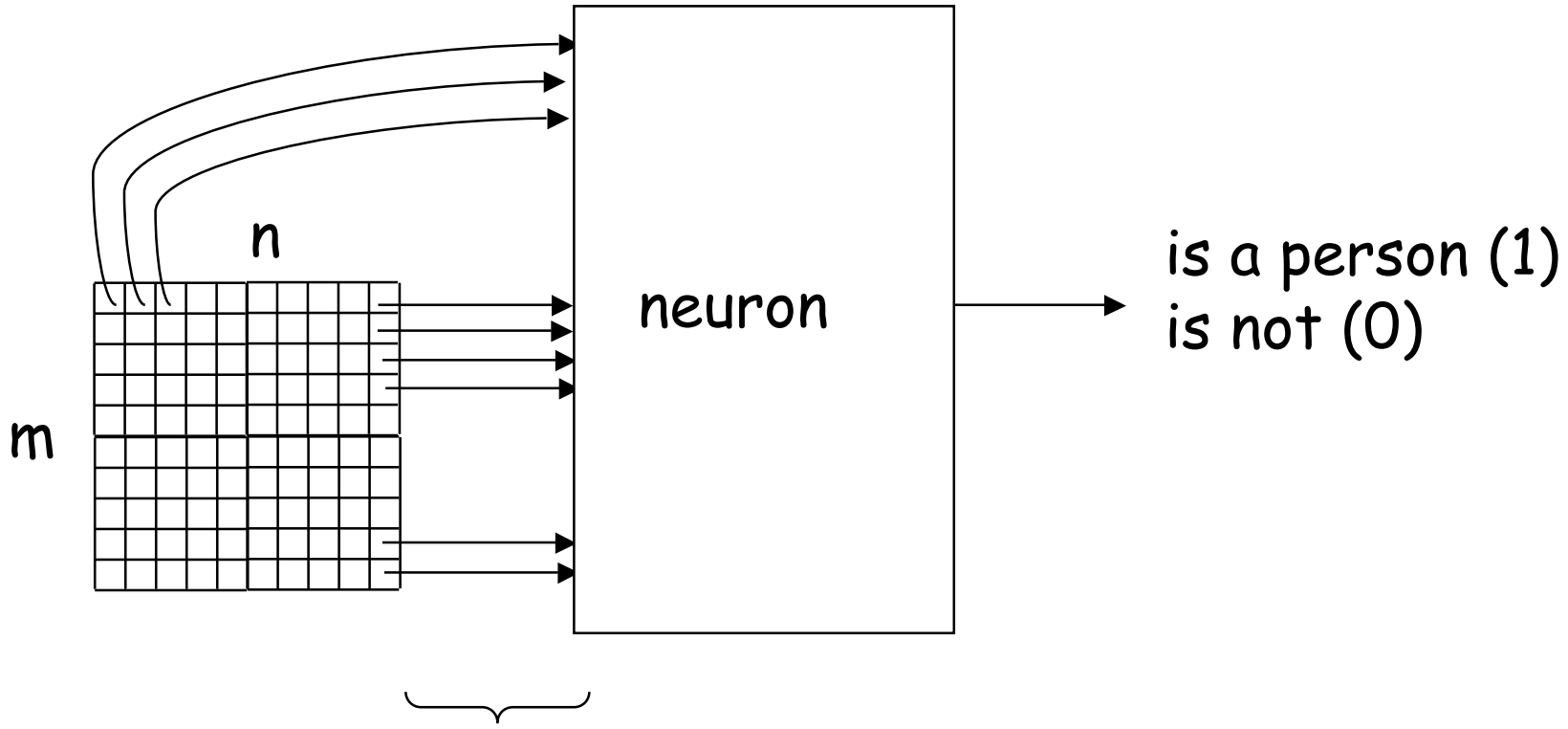
- A **perceptron** is a linear threshold gate.

- Introduced the idea of **training** for determining weights to achieve a given function

# Application for Perceptrons

- Classification problems:
  - Given a pre-specified set of input vector, each with a desired response (0 or 1), determine the weights so that a perceptron gives the desired response (and generalizes in a useful way to unseen inputs)
  - An input vector could be a retinal image, for example.

# Retinal Image Classification




vector of m x n elements in {0, 1} or even R (reals).

# Given a classification problem, try to find a perceptron to fit.

Find a vector of **weights** $\{w_i\}$ and a **threshold** $q$, such that:

$$\text{output} = \begin{cases} 1 \text{ if } \sum w_i x_i > q \\ 0 \text{ otherwise} \end{cases}$$

$$= \begin{cases} 1 \text{ if } \{x_i\} \text{ represents a vector } \textit{in} \text{ the set} \\ 0 \text{ otherwise (not } \textit{in} \text{ the set)} \end{cases}$$

# Issues

- **Existence**: Given a set of input vectors, does there **exist** a perceptron that correctly classifies the given inputs?
- **Solving**: Can the weights be found **analytically**?
- **Training**: Can the weights be found simply by presenting examples?

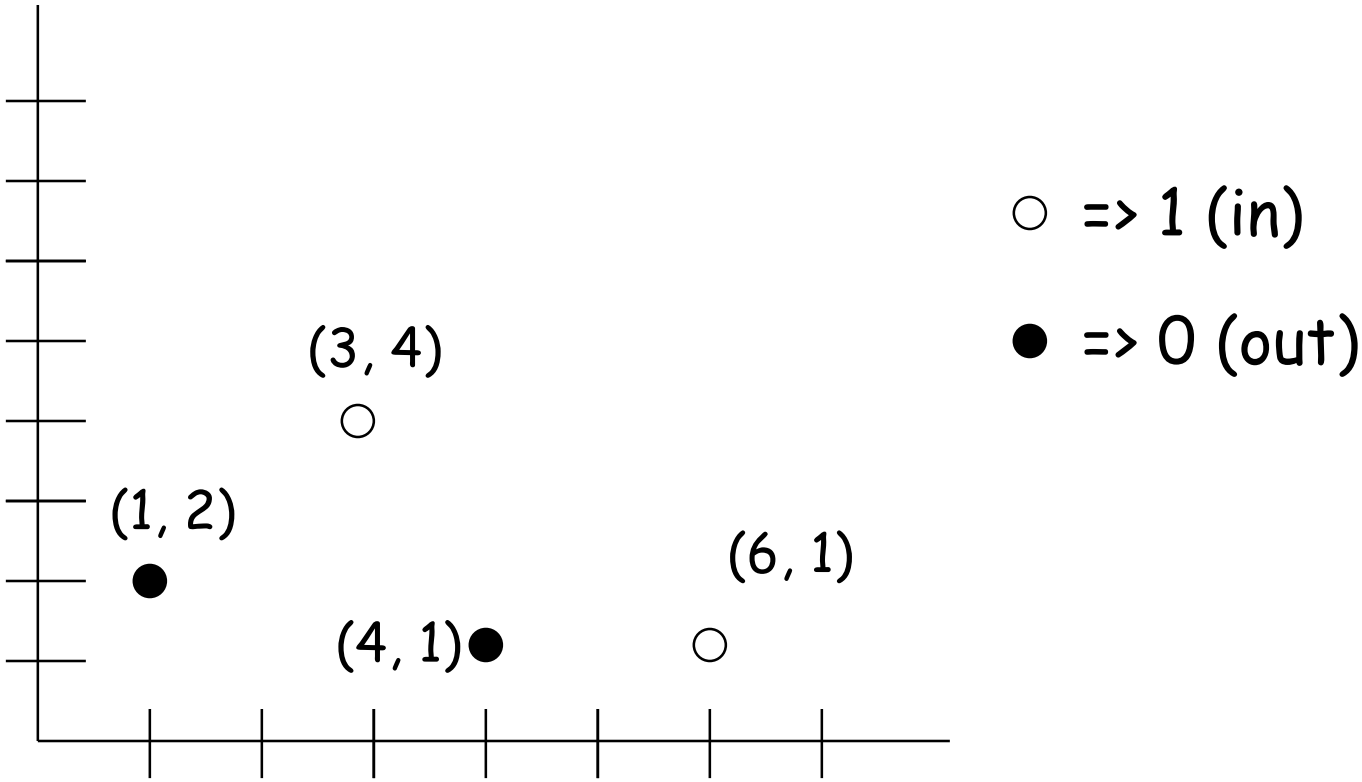
# Example

- Suppose the signals are real-valued, and the following vectors are classified as shown:

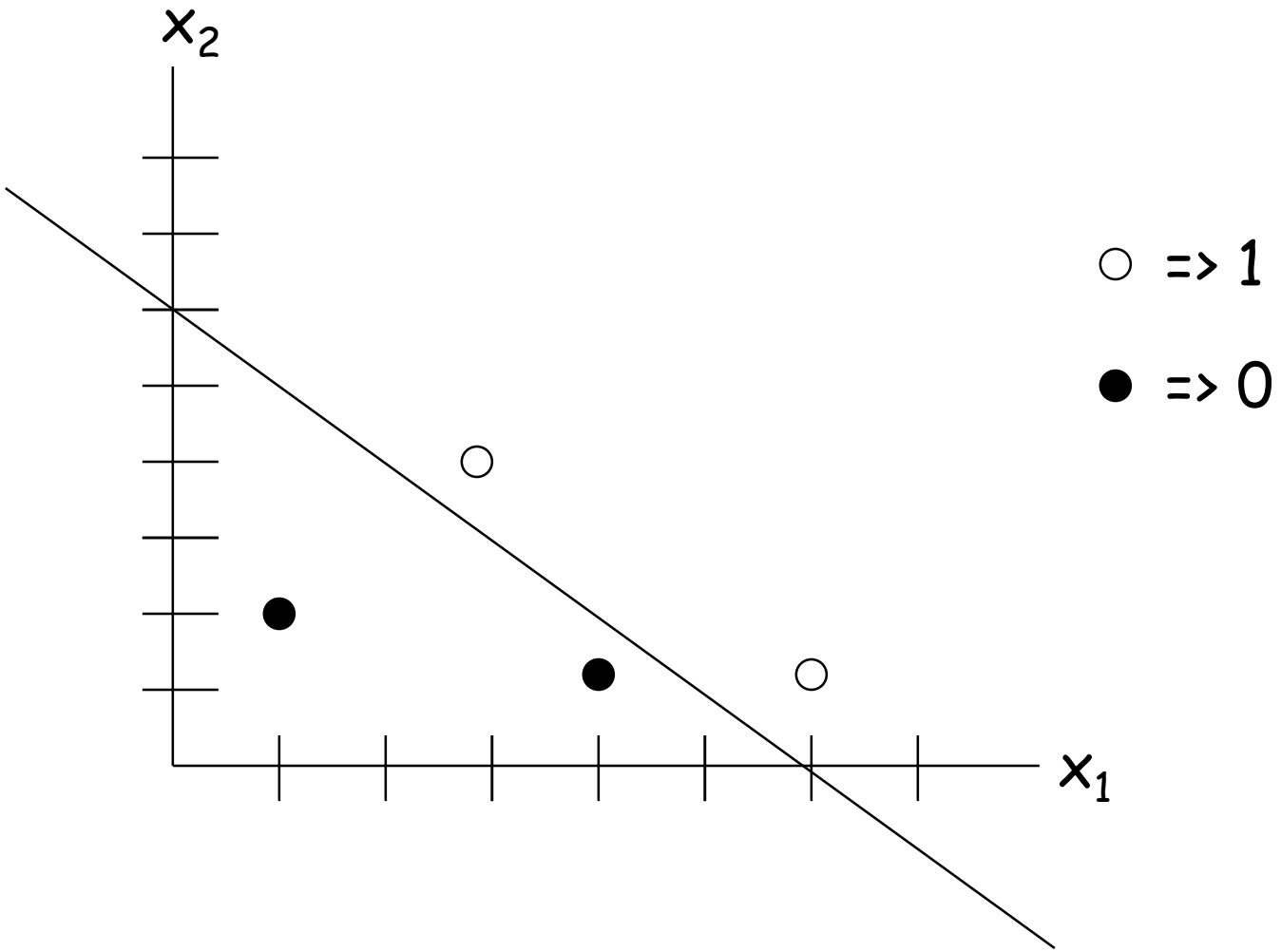
| input | output |
|---|---|
| (3, 4) | 1 |
| (6, 1) | 1 |
| (4, 1) | 0 |
| (1, 2) | 0 |

- Is there a perceptron that classifies the set as shown, and if so, what are its weights and threshold?

# Geometric Insight



# Try to Locate a Separating Line



# Separating Line Equation

- $x_1 + x_2 = 6$
- Output is
  - 1 if $x_1 + x_2 > 6$
  - 0 otherwise

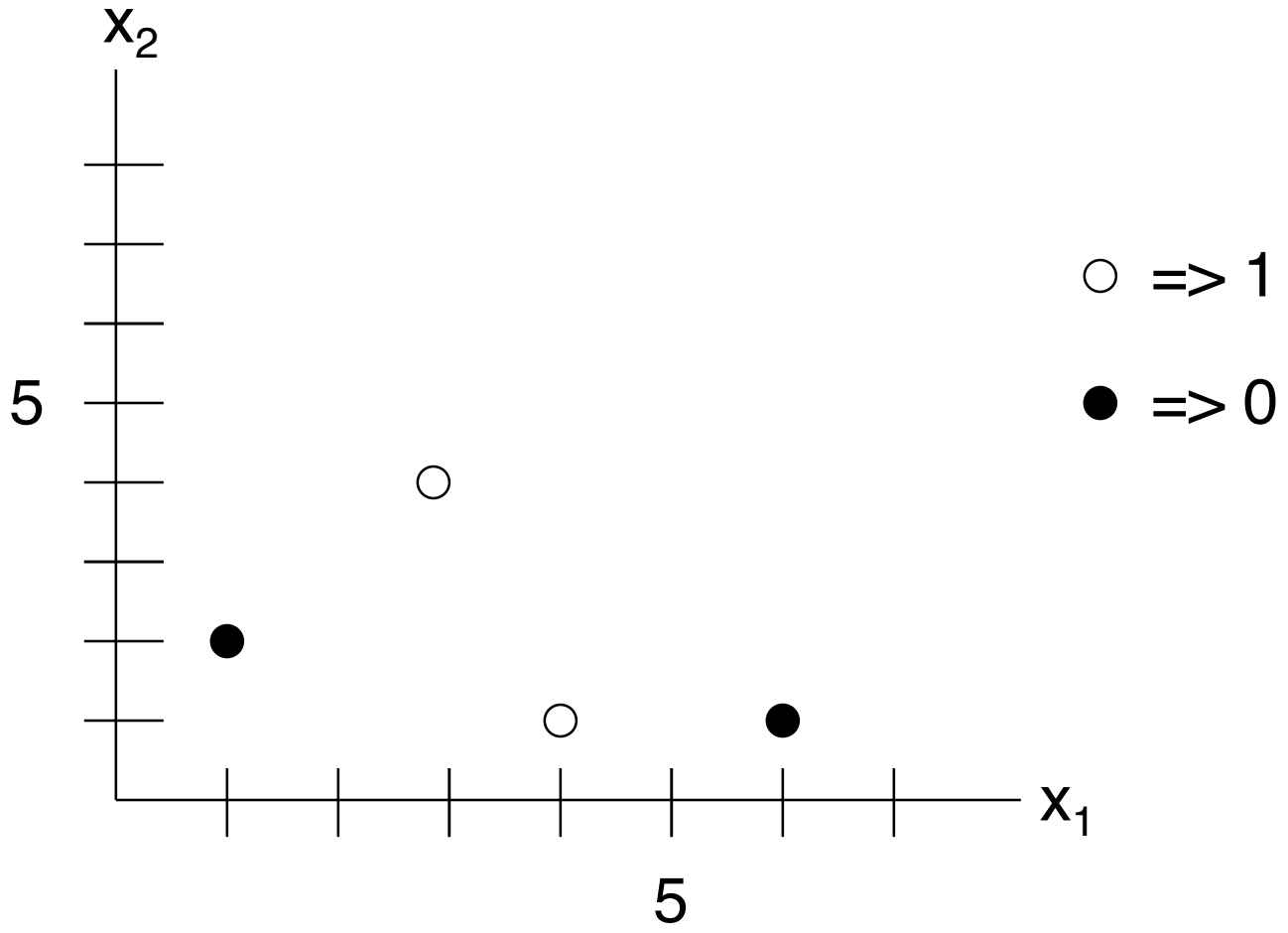
# Checking the points

- (3, 4) 1 $3 + 4 > 6$
- (6, 1) 1 $6 + 1 > 6$
- (4, 1) 0 $4 + 1 < 6$
- (1, 2) 0 $1 + 2 < 6$

# General Line Equation

- $w_1 x_1 + w_2 x_2 = \theta$
- Output is
  - 1 if $w_1 x_1 + w_2 x_2 > \theta$
  - 0 otherwise
- Will $\{w_i\}$ and $\theta$ always exist?

# Try to Locate a Separating Line



# Or prove that none exists

# Generalizing to n dimensions

- Given function $d:R^n \rightarrow \{0, 1\}$, to find $\theta, w_i \in R$ such that

  $$w_1 x_1 + w_2 x_2 + \ldots + w_n x_n = \theta$$

  separates the points:

  - $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > \theta$ when $d(x_1, x_2, \ldots, x_n) = 1$
  - $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n \leq \theta$ when $d(x_1, x_2, \ldots, x_n) = 0$

# Linear Separability

- The equation

  $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n = \theta$

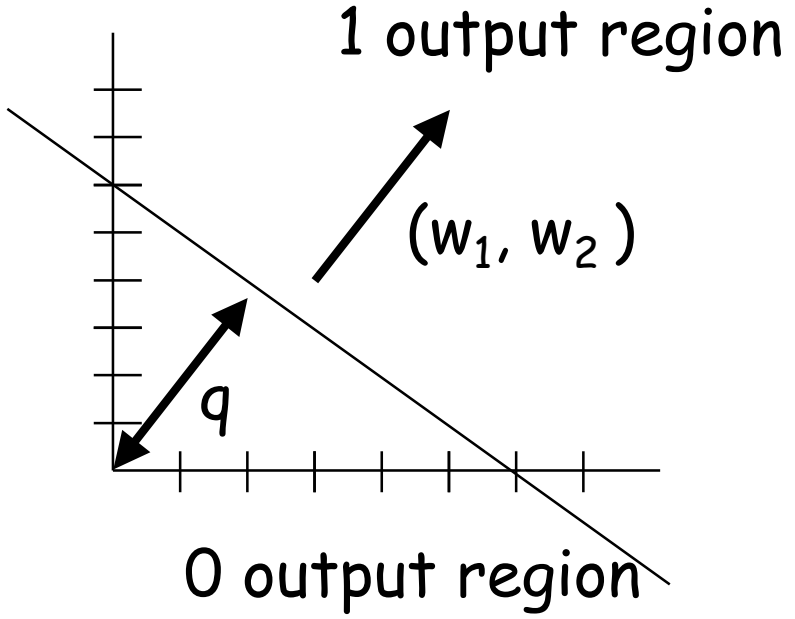
  defines a *hyperplane* in n-space

- If such a hyperplane exists for a classification problem, the problem is called **linearly-separable**.

# Geometry

- The vector of weights ($w_1$, $w_2$ , …. $w_n$) is normal (**perpendicular**) to the hyperplane.

- **Threshold** is proportional to the **distance** of the hyperplane from the origin.



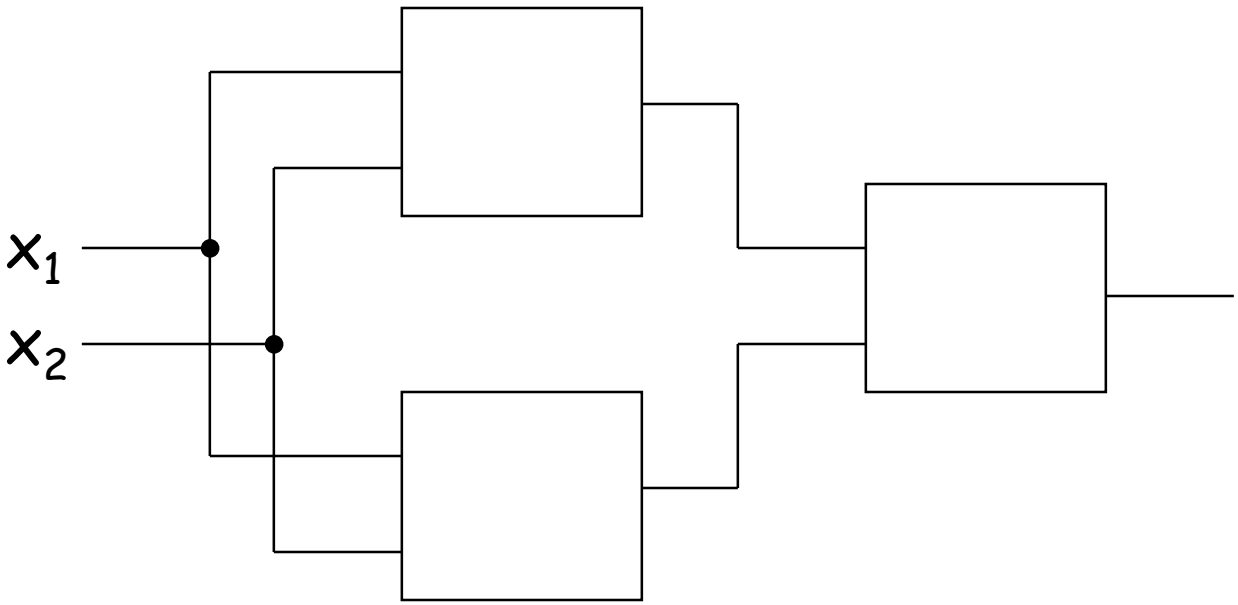
# Perceptron Summary

- A perceptron can solve a classification problem iff the problem is linearly-separable.
- There are problems a single perceptron cannot solve.
- Perhaps the simplest unsolvable one is the **XOR problem**:
  1 output: {(0, 1), (1, 0)},
  0 output: {(0, 0), (1, 1)}

# Multi-level Perceptrons

- One way to alleviate the limitations of a single perceptron is to build *networks* of perceptrons.
- Another might be to allow non-linear expressions (such as squaring).
- Can the XOR problem be solved by such models?

# What weights and thresholds for XOR?



# Perceptron Learning

# General 1-Perceptron Training

- Assuming weights exist for a problem (linear separability), how to find them?
  - Analytic? (technically not "training")
    A conventional perceptron doesn't lend well to analytic solution, due to the discontinuous nature of the function.
  - Successive approximations?

# A Somewhat General Approach

- Use a set of **training samples**:
  - Input vectors, each paired with desired output
- Choose a network structure.
- Initialize the weights and thresholds arbitrarily.
- Repeat:
  - Choose a sample
  - Test the network against the sample
  - If answer is incorrect, **adjust** weights
- until all samples test correctly.

# Omitted Details

- How to choose a sample?
- How to adjust the weights?

# Sample Choice

- Method 1: Simply cycle through all of the samples in a fixed order.
- Method 2: Choose samples randomly, but with some assurance that each will be checked before stopping.

# Weight Adjustment

- The Perceptron learning rule (Rosenblatt):
  - If the perceptron gives the correct answer, do nothing.
  - If the perceptron gives the wrong answer, nudge the weights and threshold "in the right direction", so that it eventually gives the right answer.

# When does a Perceptron give the wrong answer?

- Correct answers:

  $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > \theta$
  when $d(x_1, x_2, \ldots, x_n) = 1$;

  $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n < \theta$
  when $d(x_1, x_2, \ldots, x_n) = 0$

- For the moment, we'll ignore the $= \theta$ case.

# When does a Perceptron give the wrong answer?

- Wrong answers:

  $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > \theta$
  when $d(x_1, x_2, \ldots, x_n) = 0$;

  $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n < \theta$
  when $d(x_1, x_2, \ldots, x_n) = 1$

# Desired Output d vs Actual Output a

- Let

  $a(x_1, x_2, \ldots, x_n) =$

  1 if $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > \theta$

  0 otherwise.

- Note that a is an implied function of the weights and threshold.

# Error Value

- We can capture correct vs. incorrect answers succinctly by introducing an *error value* $\varepsilon$:
- $\varepsilon = d(x_1, x_2, \ldots, x_n) - a(x_1, x_2, \ldots, x_n)$
  $= \text{desired} - \text{actual}$
- So that
  - $\varepsilon = 0$ when the correct answer is given
  - $\varepsilon = 1$ when $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n < \theta$ but $d(x_1, x_2, \ldots, x_n) = 1$
  - $\varepsilon = -1$ when $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > \theta$ but $d(x_1, x_2, \ldots, x_n) = 0$

# Simplification: Threshold conversion

- The threshold can be treated as if one of the weights by introducing a "phantom" input of -1:

$$w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > \theta$$

iff

$$w_1 x_1 + w_2 x_2 + \ldots + w_n x_n - \theta > 0$$

iff

$$w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > 0$$

where $w_0$ is defined to be $\theta$ and $x_0 = -1$ unchangingly.

# Bias vs. Threshold

- Instead of subtracting the threshold, we could add a "bias", in which case the phantom input would be 1 rather than –1. The actual value doesn't really matter, as long as it is not 0.

# Perceptron training (continued)

- Wrong answer of the first type ($\varepsilon = 1$):
  $w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n < 0$ when $d(x_1, x_2, \ldots, x_n) = 1$

- i.e., $w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$ is **too low**.

- To correct for this, we need to make the sum **higher**.

# Perceptron training (continued)

- Wrong answer of second type ($\varepsilon$ = -1):
  $w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > 0$ when $d(x_1, x_2, \ldots, x_n) = 0$

- i.e., $w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$ is **too high**.

- To correct for this, we need to make the sum **lower**.

# Perceptron training (continued)

- Make $w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$ lower or higher by adjusting weights.
  - $\varepsilon = 1$: Make each contribution $w_i x_i$ higher
  - $\varepsilon = -1$: Make each contribution $w_i x_i$ lower
- In either case, can ***add*** some multiple $\eta$ (called the "learning rate") of $\varepsilon x_i$ to $w_i$ to get the desired effect.

# Perceptron training (continued)

- *Add* $\varepsilon\eta x_i$ to $w_i$ to get the desired effect:

$$(w_i + \varepsilon\ \eta\ x_i)\ x_i = (w_i\, x_i + \varepsilon\ \eta\ x_i^2)$$

$$\geq w_i\, x_i \qquad \text{if } \varepsilon > 0$$

$$\leq w_i\, x_i \qquad \text{if } \varepsilon < 0$$

($\varepsilon$ = desired - actual, so

in the first case need to bring actual up,

in the second bring it down)

# Learning Rate $\eta$

- $\eta$ governs the rate at which the training rule converges toward the correct solution.
- Typically $\eta \leq 1$.
- Too small an $\eta$ produces slow convergence.
- Too large of an $\eta$ can cause oscillations in the process.

# Example

- Train a perceptron to classify according to:
  - (4, 5)  1
  - (6, 1)  1
  - (4, 1)  0
  - (1, 2)  0
- There will be three weights ($w_0$, $w_1$, $w_2$) where is the $w_0$ threshold, corresponding to phantom input -1.
- Start with "random" weights, say (0, +1, -1)
- Choose $\eta = 1$.

# Perceptron Training Example
# One Pass over Data Samples

Fill out this table sequentially:

| weights | input | desired | actual | error | new weights |
|---|---|---|---|---|---|
| (0, 1, -1) | (-1, 4, 5) | 1 | | | |
| | (-1, 6, 1) | 1 | | | |
| | (-1, 4, 1) | 0 | | | |
| | (-1, 1, 2) | 0 | | | |

# Perceptron Training Example
# One Pass over Data Samples

| weights | input | desired | actual | error | new weights |
|---|---|---|---|---|---|
| (0, 1, -1) | (-1, 4, 5) | 1 | 0 | 1 | (-1, 5, 4) |
| (-1, 5, 4) | (-1, 6, 1) | 1 | 1 | 0 | no change |
| (-1, 5, 4) | (-1, 4, 1) | 0 | 1 | -1 | (0, 1, 3) |
| (0, 1, 3) | (-1, 1, 2) | 0 | 1 | -1 | (1, 0, 1) |

# Perceptron Training Example
# Second Pass over Data Samples

| weights | input | desired | actual | error | new weights |
|---|---|---|---|---|---|
| (1, 0, 1) | (-1, 4, 5) | 1 | | | |
| | (-1, 6, 1) | 1 | | | |
| | (-1, 4, 1) | 0 | | | |
| | (-1, 1, 2) | 0 | | | |

# Perceptron Training Example
# Second Pass over Data Samples

| weights | input | desired | actual | error | new weights |
|---|---|---|---|---|---|
| (1, 0, 1) | (-1, 4, 5) | 1 | 1 | 0 | no change |
| (1, 0, 1) | (-1, 6, 1) | 1 | 0 | 1 | (0, 6, 2) |
| (0, 6, 2) | (-1, 4, 1) | 0 | 1 | -1 | (1, 2, 1) |
| (1, 2, 1) | (-1, 1, 2) | 0 | 1 | -1 | (2, 1, -1) |

# Perceptron Training Example
# Third Epoch

| weights | input | desired | actual | error | new weights |
|---|---|---|---|---|---|
| (2, 1, -1) | (-1, 4, 5) | 1 | 0 | 1 | (1, 5, 4) |
| (1, 5, 4) | (-1, 6, 1) | 1 | 1 | 0 | no change |
| (1, 5, 4) | (-1, 4, 1) | 0 | 1 | -1 | (2, 1, 3) |
| (2, 1, 3) | (-1, 1, 2) | 0 | 1 | -1 | (3, 0, 1) |

# Perceptron Training Example
# Epoch 4

| weights | input | desired | actual | error | new weights |
|---|---|---|---|---|---|
| (3, 0, 1) | (-1, 4, 5) | 1 | 1 | 0 | no change |
| (3, 0, 1) | (-1, 6, 1) | 1 | 0 | 1 | (2, 6, 2) |
| (2, 6, 2) | (-1, 4, 1) | 0 | 1 | -1 | (3, 2, 1) |
| (3, 2, 1) | (-1, 1, 2) | 0 | 1 | -1 | (4, 1, -1) |

# Perceptron Training Example
# Epoch 5

| weights | input | desired | actual | error | new weights |
|---|---|---|---|---|---|
| (4, 1, -1) | (-1, 4, 5) | 1 | 0 | 1 | (3, 5, 4) |
| (3, 5, 4) | (-1, 6, 1) | 1 | 1 | 0 | no change |
| (3, 5, 4) | (-1, 4, 1) | 0 | 1 | -1 | (4, 1, 3) |
| (4, 1, 3) | (-1, 1, 2) | 0 | 1 | -1 | (5, 0, 1) |

# Perceptron Training Example
# Epoch 6

| weights | input | desired | actual | error | new weights |
|---|---|---|---|---|---|
| (5, 0, 1) | (-1, 4, 5) | 1 | 0 | 1 | (4, 4, 6) |
| (4, 4, 6) | (-1, 6, 1) | 1 | 1 | 0 | no change |
| (4, 4, 6) | (-1, 4, 1) | 0 | 1 | -1 | (5, 0, 5) |
| (5, 0, 5) | (-1, 1, 2) | 0 | 1 | -1 | (6, -1, 3) |

# Perceptron Training Example
# Epoch 7

| weights | input | desired | actual | error | new weights |
|---|---|---|---|---|---|
| (6, -1, 3) | (-1, 4, 5) | 1 | 1 | 0 | no change |
| (6, -1, 3) | (-1, 6, 1) | 1 | 0 | 1 | (5, 5, 4) |
| (5, 5, 4) | (-1, 4, 1) | 0 | 1 | -1 | (6, 1, 3) |
| (6, 1, 3) | (-1, 1, 2) | 0 | 1 | -1 | (7, 0, 1) |

# Perceptron Training Example
# Epoch 8

| weights | input | desired | actual | error | new weights |
|---|---|---|---|---|---|
| (7, 0, 1) | (-1, 4, 5) | 1 | 0 | 1 | (6, 4, 6) |
| (6, 4, 6) | (-1, 6, 1) | 1 | 1 | 0 | no change |
| (6, 4, 6) | (-1, 4, 1) | 0 | 1 | -1 | (7, 0, 5) |
| (7, 0, 5) | (-1, 1, 2) | 0 | 1 | -1 | (8, -1, 3) |

# Perceptron Training Example
# Epoch 9

| weights | input | desired | actual | error | new weights |
|---|---|---|---|---|---|
| (8, -1, 3) | (-1, 4, 5) | 1 | 1 | 0 | no change |
| (8, -1, 3) | (-1, 6, 1) | 1 | 0 | 1 | (7, 5, 2) |
| (7, 5, 2) | (-1, 4, 1) | 0 | 1 | -1 | (8, 1, 3) |
| (8, 1, 3) | (-1, 1, 2) | 0 | 0 | 0 | (8, 1, 3) |

# Perceptron Training Example
# Epoch 10

| weights | input | desired | actual | error | new weights |
|---|---|---|---|---|---|
| (8, 1, 3) | (-1, 4, 5) | 1 | 1 | 0 | no change |
| (8, 1, 3) | (-1, 6, 1) | 1 | 1 | 0 | no change |
| (8, 1, 3) | (-1, 4, 1) | 0 | 0 | 0 | no change |
| (8, 1, 3) | (-1, 1, 2) | 0 | 0 | 0 | no change |

# Perceptron Training Example Conclusion

- A perceptron with weights (8, 1, 3) correctly classifies all inputs.
- The "yes" criterion is therefore:
  $-8 + x_1 + 3\,x_2 > 0$ [i.e. $x_1 + 3\,x_2 > 8$]
- Check:

| input x1, x2 | desired | actual |
|---|---|---|
| (4, 5) | 1 | 1 |
| (6, 1) | 1 | 1 |
| (4, 1) | 0 | 0 |
| (1, 2) | 0 | 0 |

# Perceptron Training Algorithm (1)

- Inputs:
  - A list of training samples, each of the form
    - $[d(x_1, x_2, \ldots, x_n), -1, x_1, x_2, \ldots, x_n]$
    - (d is the desired output, -1 the phantom input)
  - An initial weight vector $[w_0, w_1, w_2, \ldots, w_n]$
    - ($w_0$ is the threshold)
  - A learning rate $\eta$

# Perceptron Training Algorithm (2)

- Outputs:
  - If the set of samples is linearly separable, a vector of weights $[w_0, w_1, w_2, \ldots, w_n]$ such that with these weights the perception properly separates the training samples.
  - If the set of samples is not linearly separable, then the algorithm diverges.

# Perceptron Training Algorithm (3)

- Operation:
  - Set $[w_0, w_1, w_2, \ldots, w_n]$ = initial weights;
  - **while**( there is a sample not correctly classified )
    - Let $[d(x_1, x_2, \ldots, x_n), -1, x_1, x_2, \ldots, x_n]$ be an incorrectly classified sample.
    - Let $\varepsilon = d(x_1, x_2, \ldots, x_n) - a(x_1, x_2, \ldots, x_n)$, where $a(x_1, x_2, \ldots, x_n) = 1$ if $(\Sigma\, w_i x_i > 0)$, 0 otherwise.
    - Vector-add to $[w_0, w_1, w_2, \ldots, w_n]$ the vector

      $$\Delta w = \varepsilon\, \eta\, [-1, x_1, x_2, \ldots, x_n]$$

      **Perceptron learning rule**

# Perceptron Training Algorithm Modifications for practical usage

- Put a ***limit*** on the number of iterations, so that the algorithm will terminate (without perfect classification) even if the sample set is not linearly separable.
- Include an ***error bound*** as an extra input. The algorithm can stop as soon as the portion of mis-classified samples is less than this bound (as opposed to requiring perfect classification, which would be an error bound of 0).
- Generate the initial weights ***randomly***, so that the user does not have to specify them.

# Correctness of the Perceptron Training Algorithm

# Some Simplifications

- All training vectors x (including the phantom -1) can be **normalized**, by dividing by the length ||x||, so that ||x|| = 1.

- This is because only the **sign** of wx matters in classification, and the sign of wx is the same as that of wx/||x||.

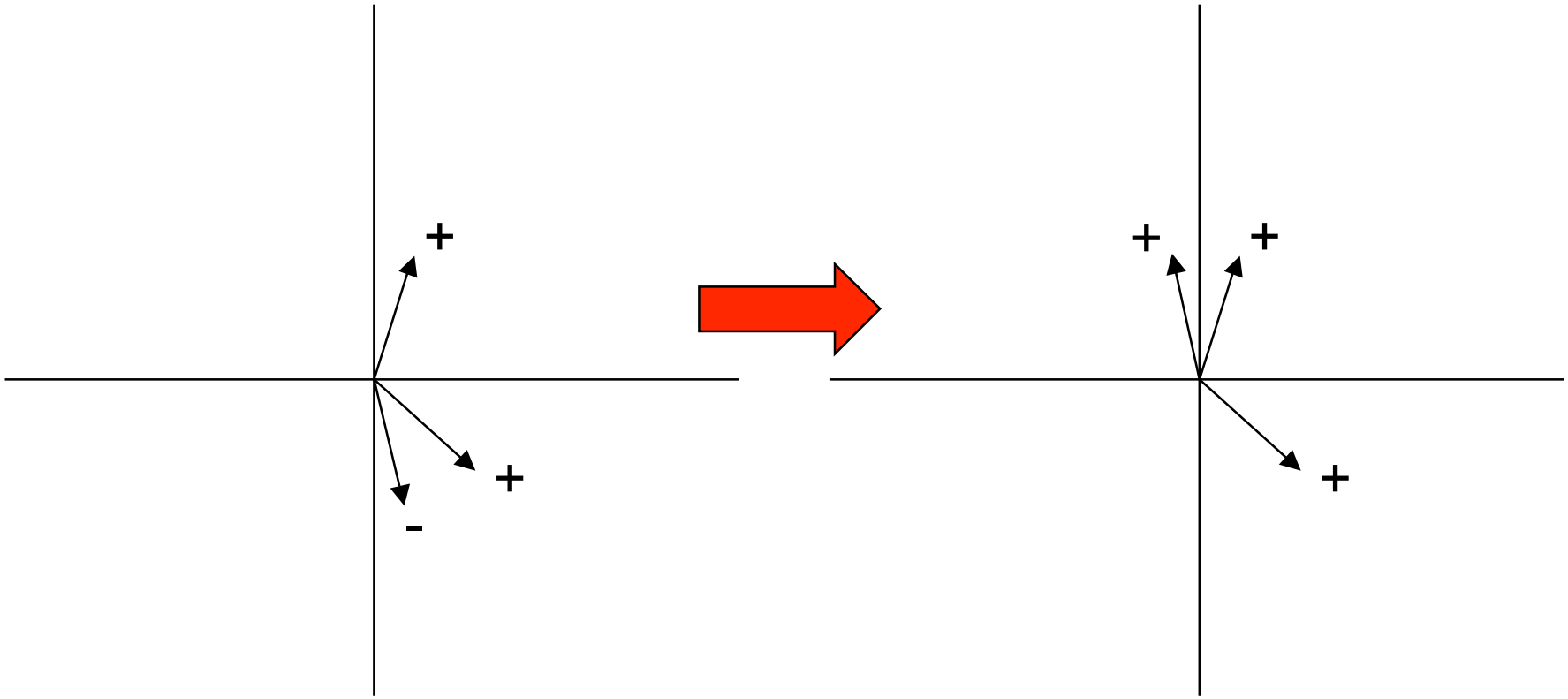
# Some Simplifications

- The weight vector w can also be normalized, so that ||w|| = 1, by the same rationale.

# Some Simplifications

- All training samples can be assumed to have positive desired value.
- If the desired value were to be negative, it can be made positive just be complementing all of the components.

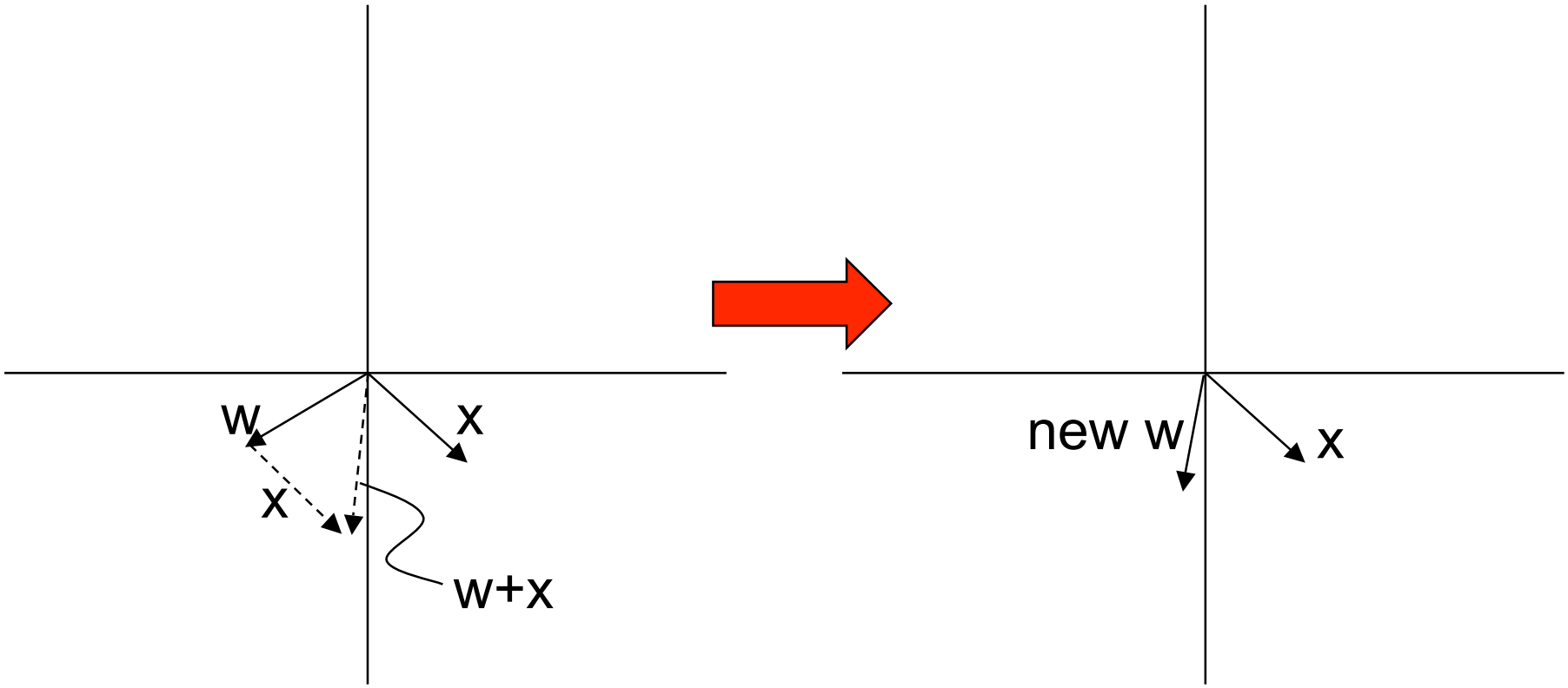
# Simplification



# Visualization

- For normalized w and x, the value wx is the **cosine** of the angle $\rho$ between w and x.
- We want to find a weight vector which has a positive value of cos $\rho$ with each x.
- A training step consists of adding vector $\Delta w$ (proprtional to x) to w, to push w away if cos $\rho < 0$.

# Learning



# Initialization Heuristic

- If all inputs are positive and normalized, average them to get the initial weight value.

# Convergence Proof

- Assume all samples are positive and normalized.
- Assume the weights are normalized.

- **Claim**: If a weight vector w* exists which correctly classifies all samples, one will be found by the perceptron training algorithm.

- WLOG, assume the learning rate $\eta$ is 1.
- WLOG, assume classification is strict, i.e. $w^*x > 0$ for all x.

# Proof (following Rojas, pp 88-89)

- Assume $w^*$ exists ($w^*$ classifies correctly).
- Let $w_t$ be the weight vector after t steps of the algorithm.
- Consider step t+1, which resulted from some vector $x_i$ being misclassified by $w_t$.
- Thus $w_{t+1} = w_t + x_i$.

# Proof continues

- Consider $\rho$ where
  $\cos \rho = w^* w_{t+1} / || w_{t+1} ||$, which must be $\leq 1$.
- In the **numerator**,
  $w^* w_{t+1} = w^*(w_t + x_i)$.
  $= w^* w_t + w^* x_i$.
  $\geq w^* w_t + \delta$,
  where $\delta = \min\{w^* x_j \mid \text{samples } x_j\} > 0$, by strict classification.
- By inductive substitution,
  $w^* w_{t+1} \geq w^* w_0 + \delta(t+1)$.

# Proof continues

- Again, $w^* \, w_{t+1} / \, || \, w_{t+1} \, || \leq 1$.
- Squaring the **denominator**,
  $|| \, w_{t+1} \, ||^2 = (w_t + x_i) \, (w_t + x_i)$
  $= ||w_t \, ||^2 + 2 w_t x_i + ||x_i \, ||^2$
- But $w_t x_i < 0$, because $x_i$ was misclassified, so
  $|| \, w_{t+1} \, ||^2 \leq ||w_t \, ||^2 + ||x_i \, ||^2$
  $\leq ||w_t \, ||^2 + 1$, since $x_i$ are normalized.
- By inductive substitution,
  $|| \, w_{t+1} \, ||^2 \leq ||w_0 \, ||^2 + (t + 1)$

# Proof continues

- From two inequalities derived on the previous two slides:

$$1 \geq \frac{(w^* w_0 + \delta(t+1))}{\text{sqrt}(||w_0||^2 + (t+1))}$$

- The RHS grows with t as sqrt(t).
- Since since $\delta > 0$, that growth must be bounded.
- Hence there is a **maximum** t for which classification is incorrect, i.e. the algorithm terminates.

# Accelerated Convergence ("Corrective Learning")

- Instead of using the learning rate to control weight increments, add "just enough" weight to make the incorrectly-classified sample be classified correctly.
- Instead of:

  $\Delta w = \varepsilon \eta [ -1, x_1, x_2, \ldots, x_n ]$

  use

  $\Delta w = \alpha [ -1, x_1, x_2, \ldots, x_n ]$

  where $\alpha$ is such that

  $(w + \Delta w) [ -1, x_1, x_2, \ldots, x_n ] > 0.$

  i.e. $(w + \alpha x) x > 0$, while $wx \leq 0$.
- Want $\alpha x x > -wx$, i.e. $\alpha > -wx / ||x||^2$.
- Pick $\alpha = -wx / ||x||^2 + \beta$ for some small $\beta$.

# Multiple Categories

- If there are more than 2 categories, the classification strategy must be extended somehow.
- Two ideas that don't work perfectly are:
  - Have a separate {yes, no} for each category.
    [Could have multiple yes answers, or none.]
  - Do pair-wise discrimination between all pairs of categories.
    [No guarantee of transitivity.]

# Winner-take-all (WTA) Strategy

- Have a separate perceptron for each category.
- Rather than using the discrete output, use the weighted sum.
- Pick the category with the highest weighted sum (argmax).
- Break ties arbitrarily.

- Even this strategy can have issues.