Harvey Mudd College

CS 152
Neural Networks
Fall 2010

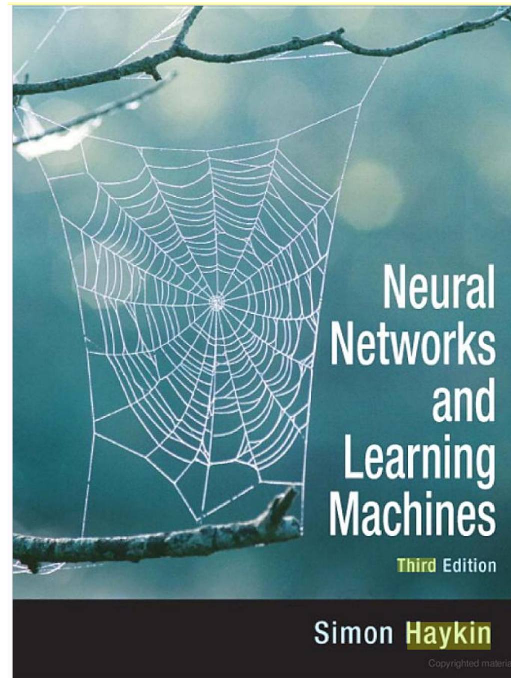
Bob Keller, Professor
keller@cs.hmc.edu
621-8483
Olin 1253

Office Hours (1253 Olin):

- MTW 4:00-5:30
- By drop-in (as available, just knock!)

  Door signals: 1 on top = in, 0 on top = out
- By appointment:
  - email keller@cs.hmc.edu
  - AIM: MuddProf
  - phone 621-8483

Text



Preview:
Visit books.google.com.
Enter: Haykin, Third edition

Another Helpful Text, Free and Useful, but Dated

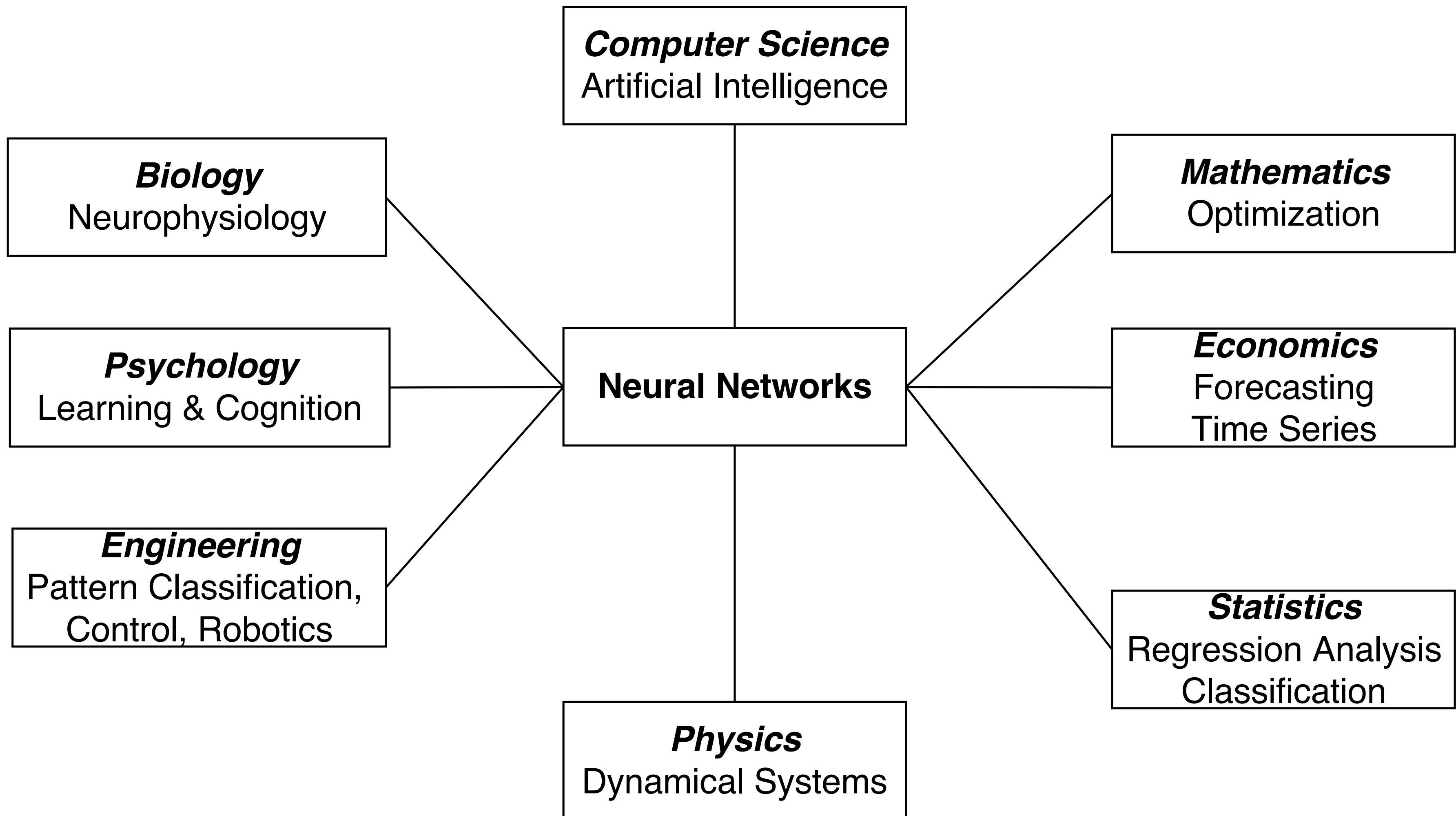
Raul Rojas, ***Neural Networks - A Systematic Introduction***, Springer-Verlag, Berlin, New-York, 1996 (502 pages,350 illustrations). This book is no longer in print. However, it may be obtained legally on the web in pdf form:
http://www.inf.fu-berlin.de/inst/ag-ki/rojas_home/documents/1996/NeuralNetworks/neuron.pdf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

Course Outline

Please Refer to Home Page for details:

http://www.cs.hmc.edu/courses/2010/fall/cs152

Neural Networks: an Eclectic Discipline



Biological Intelligence

- **Intelligence**: the ability to make decisions based upon input from the environment that aid individual survival.
- Intelligence is **realized** by ***networks*** of ***neurons***, for example the brain and the attendant sensory and motor neurons.

figure source: http://en.wikipedia.org/wiki/Intelligence

"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-Engineer Biology
  - Understand **real** neurons well enough to model accurately
  - ***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
- Combinations of the above

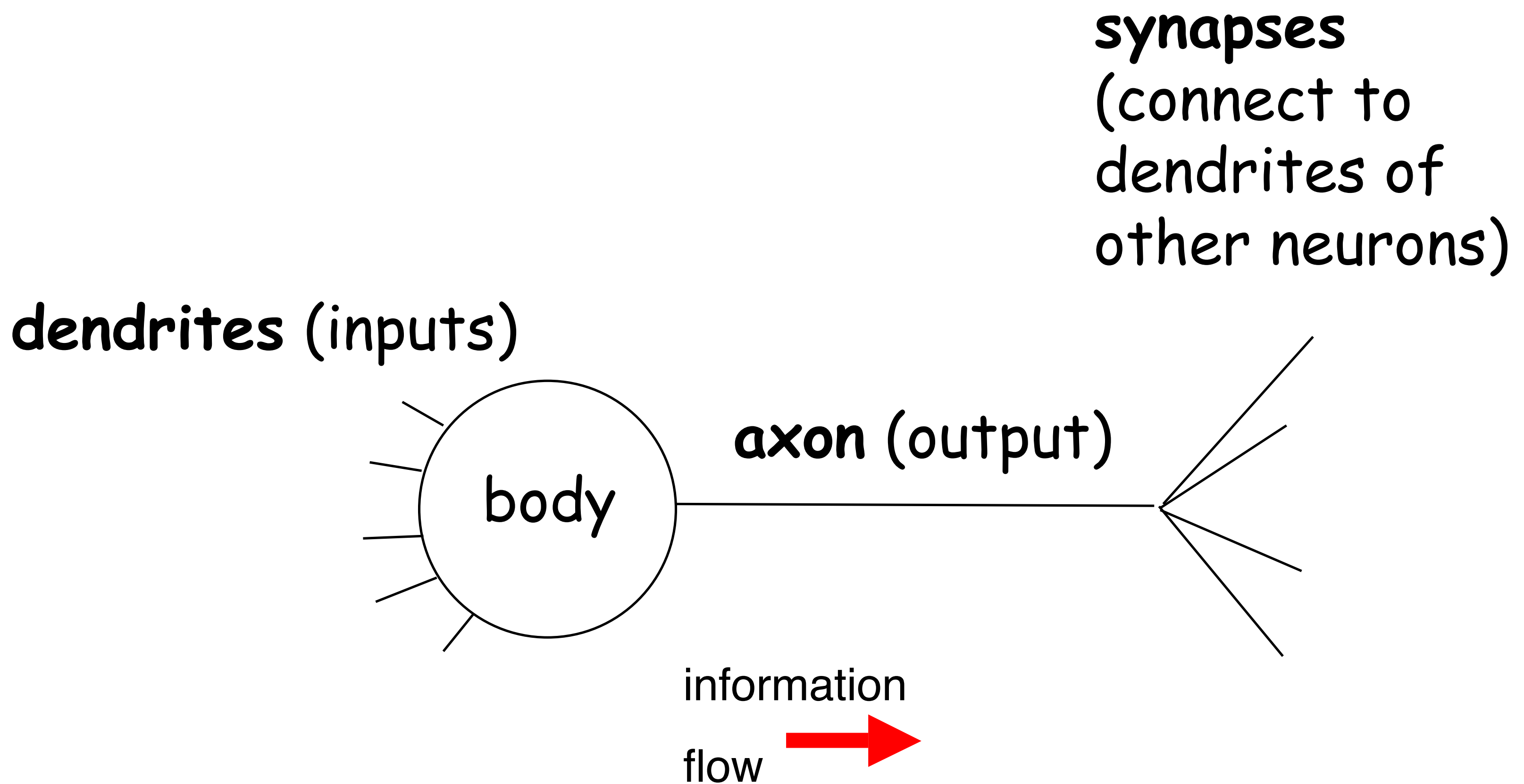
Fundamental Problems for a Given Neural Model

- 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)

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

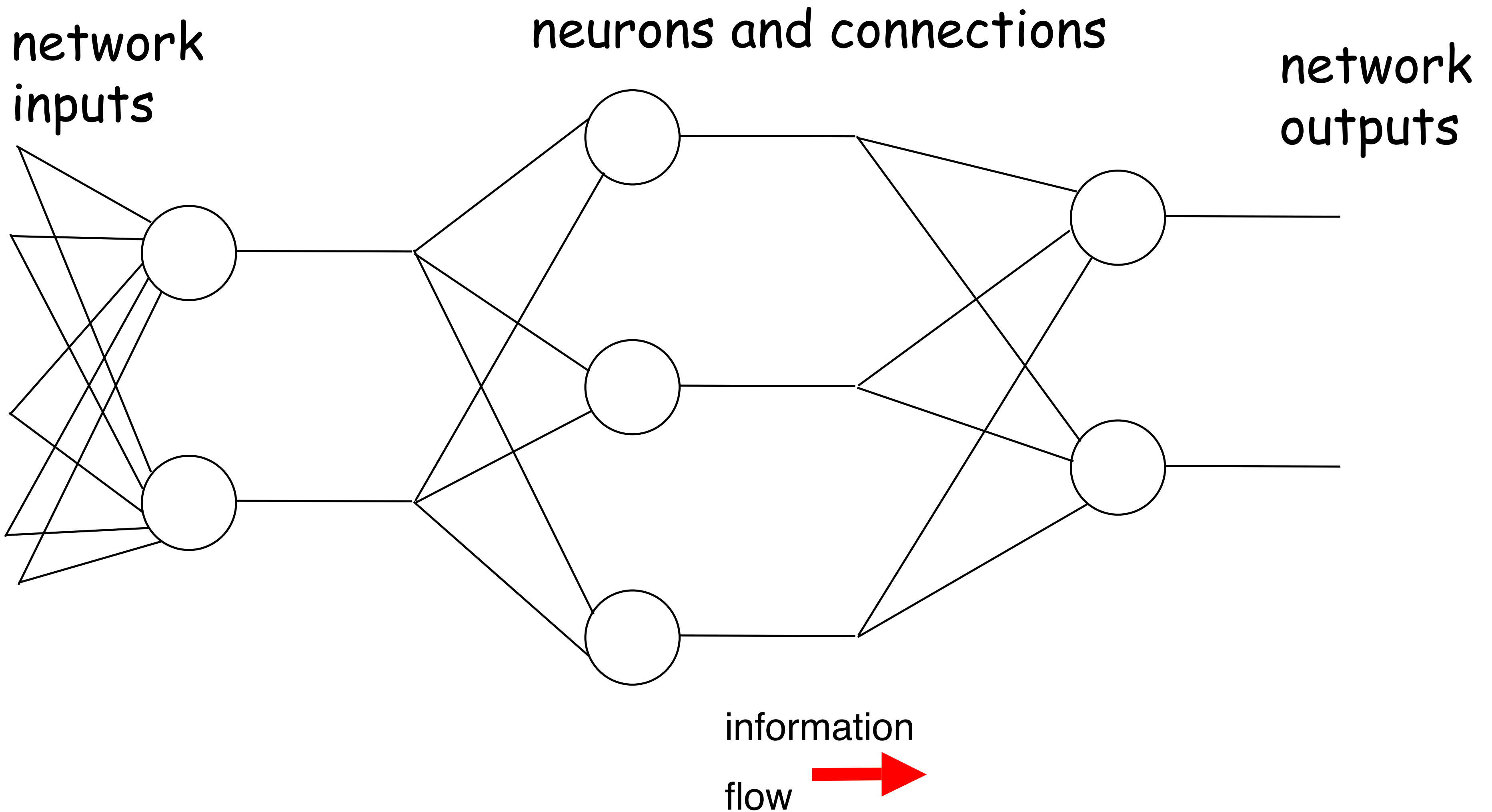
Schematic of One Neuron



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

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

Schematic of a Neural Network



Electronmicrograph of one synapse/dendrite connection

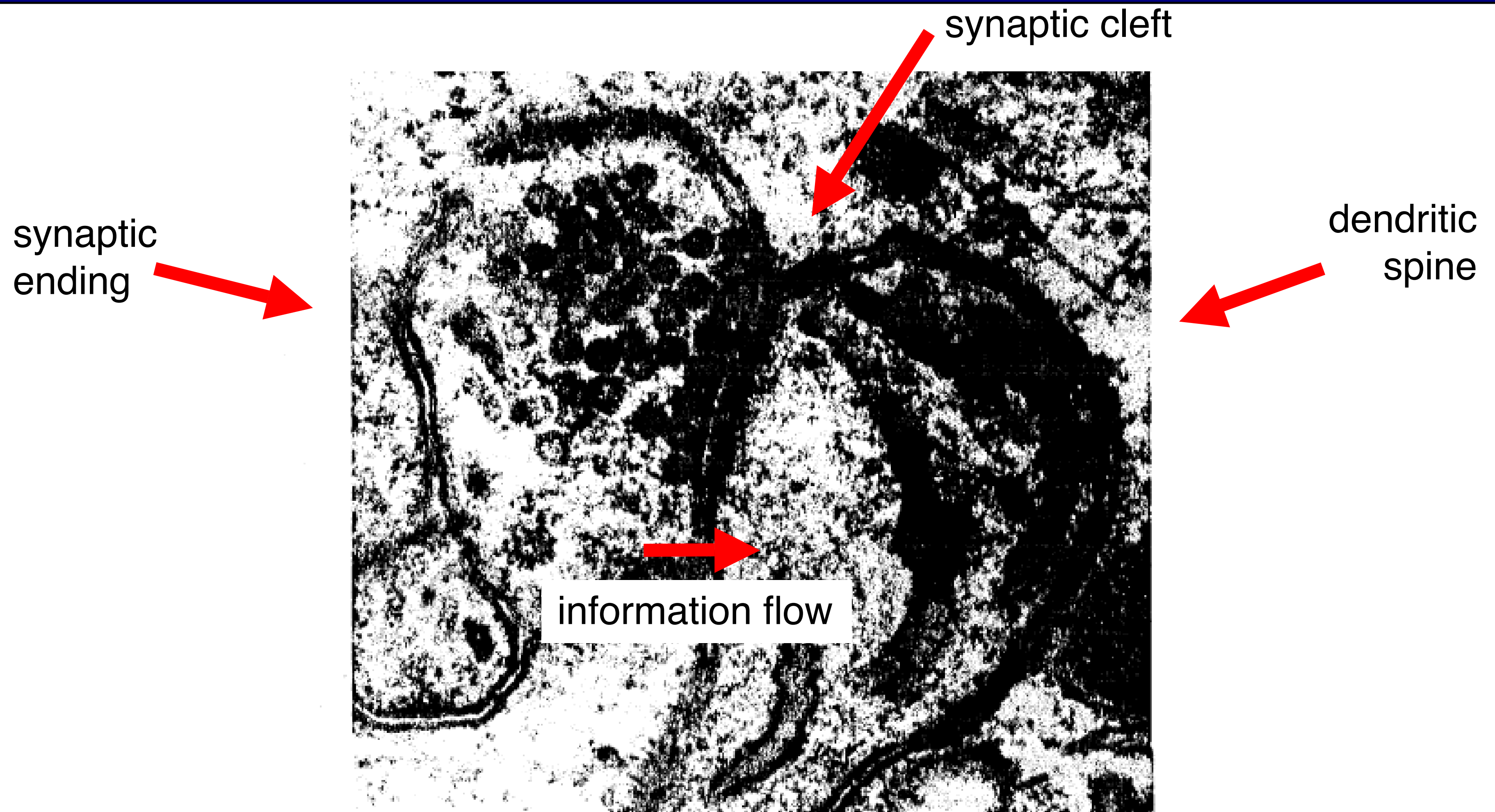
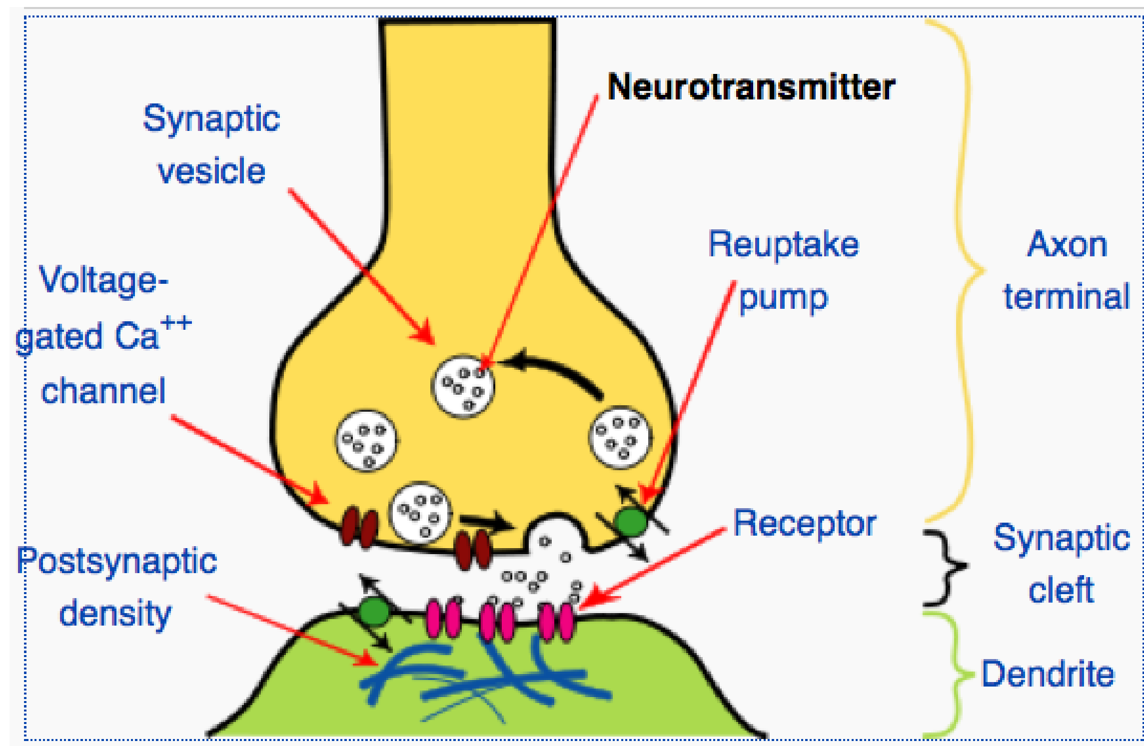


figure source: Irwin B. Levitan and Leonard K. Kaczmarek, The Neuron, Oxford University Press, 1991.

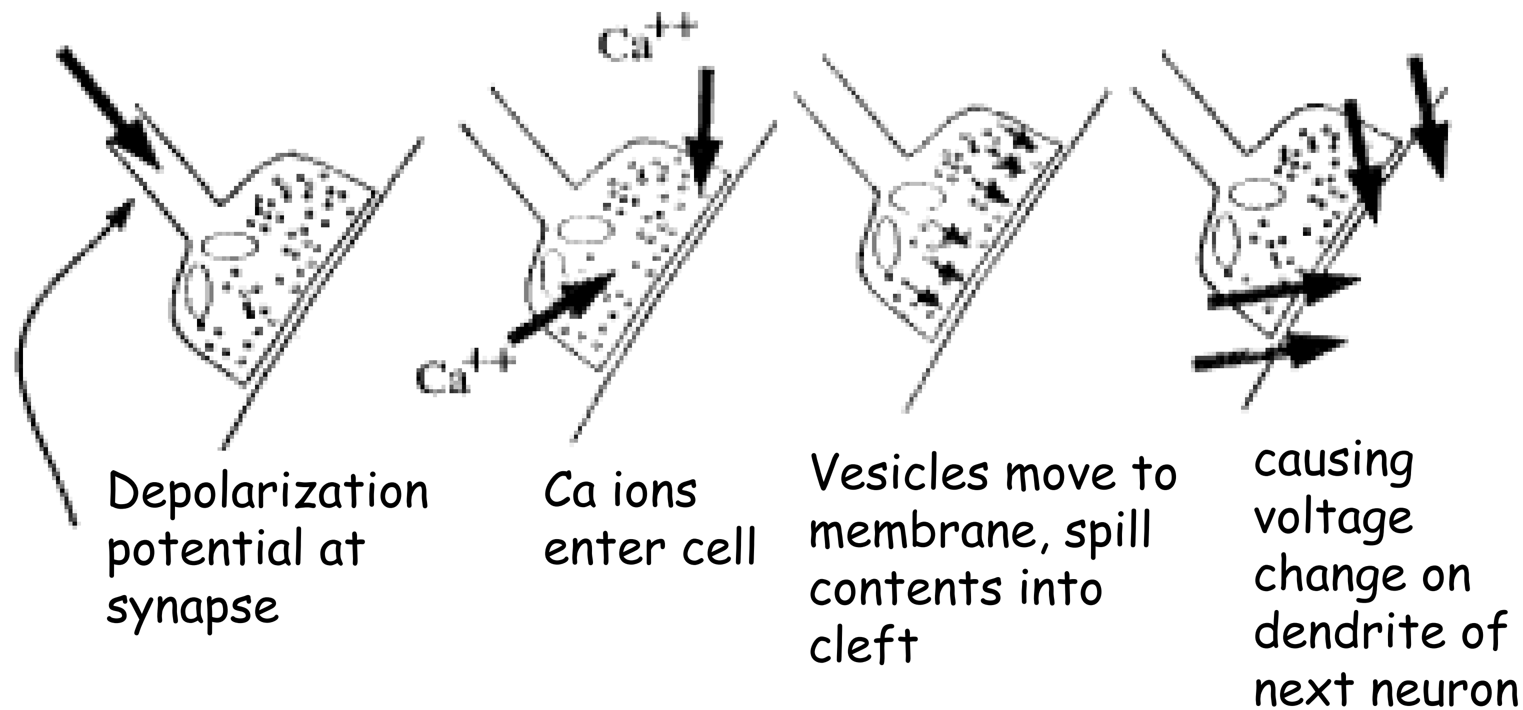
Chemical Synapse Structure



information flow

figure source: http://en.wikipedia.org/wiki/Neurotransmitter

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.

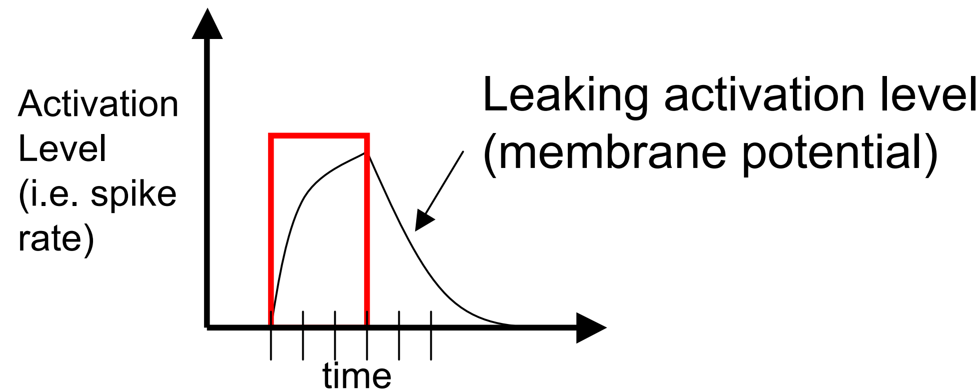
Biological Neuron Terminology

- **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 slowly **leaks** the built-up potential.

"Leaky-Integrator" Behavior



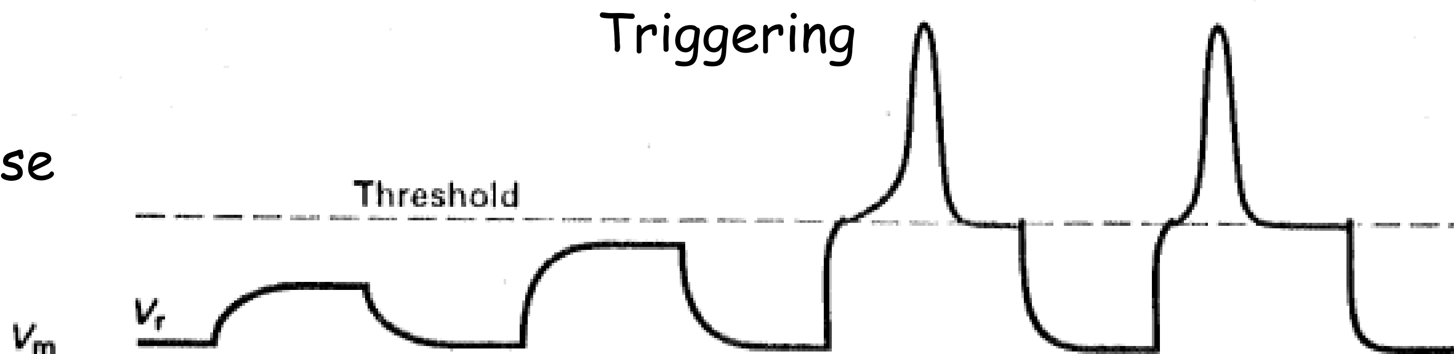
Source: Kenneth Stanley,
Leaky Integrator Neurons and CTRNNs, 2006

Triggering phenomenon

Net stimulus (integrated inputs)

Triggering

Response



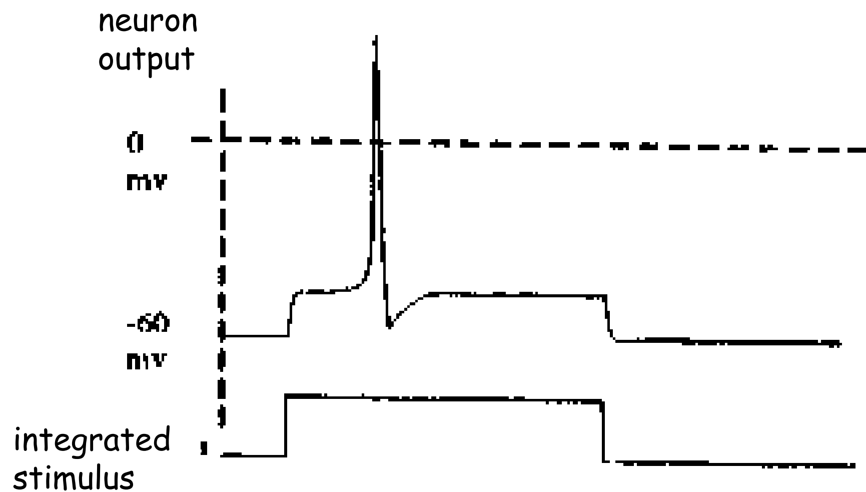
reference: Irwin B. Levitan and Leonard K. Kaczmarek, The Neuron, Oxford University Press, 1991.

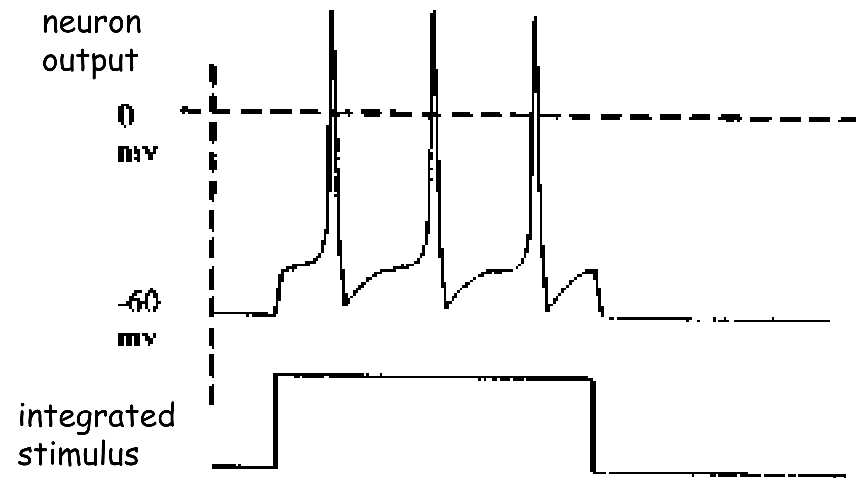
Intensity

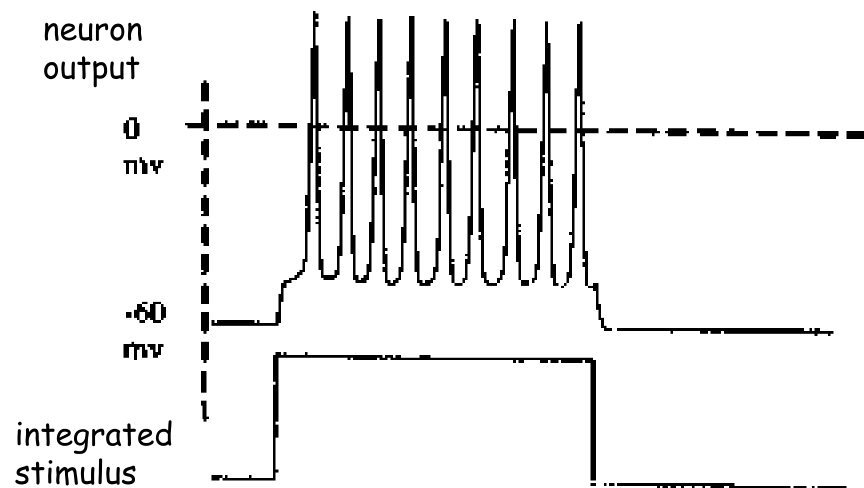
- Because of the triggering behavior, the neuron indicates intensity of stimulation by the ***frequency*** of spikes, rather than amplitude.

- 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





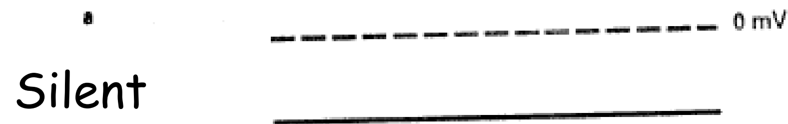


Larger integrated stimulus implies higher frequency of output spiking

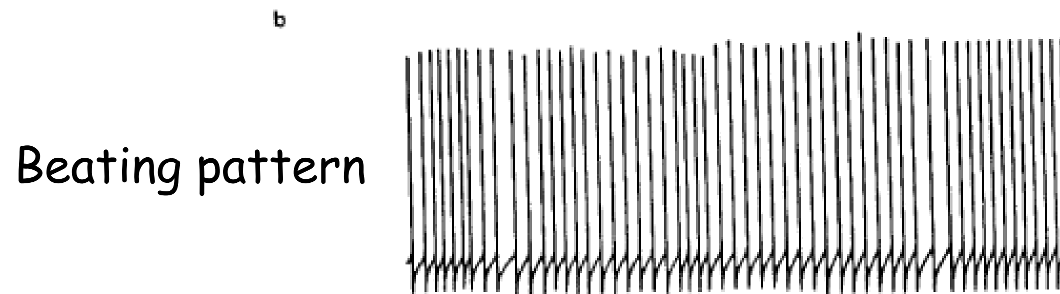
reference: James A. Anderson, An Introduction to Neural Networks, MIT Press, 1955.

Various Spiking Patterns

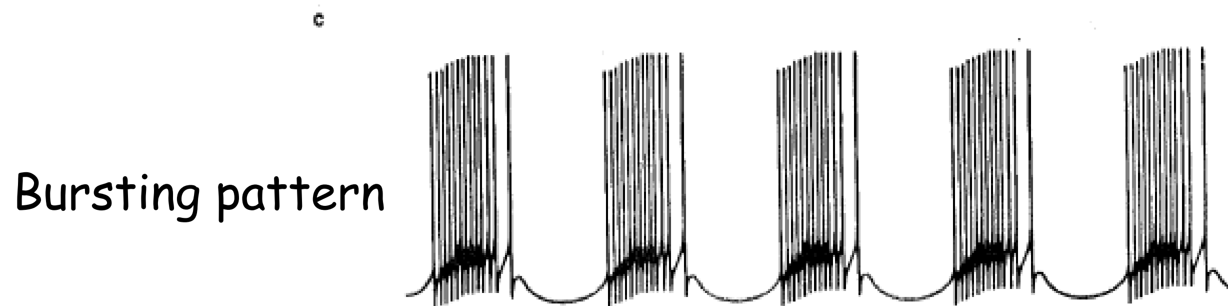
Silent



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.

- In computer science, we typically 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.

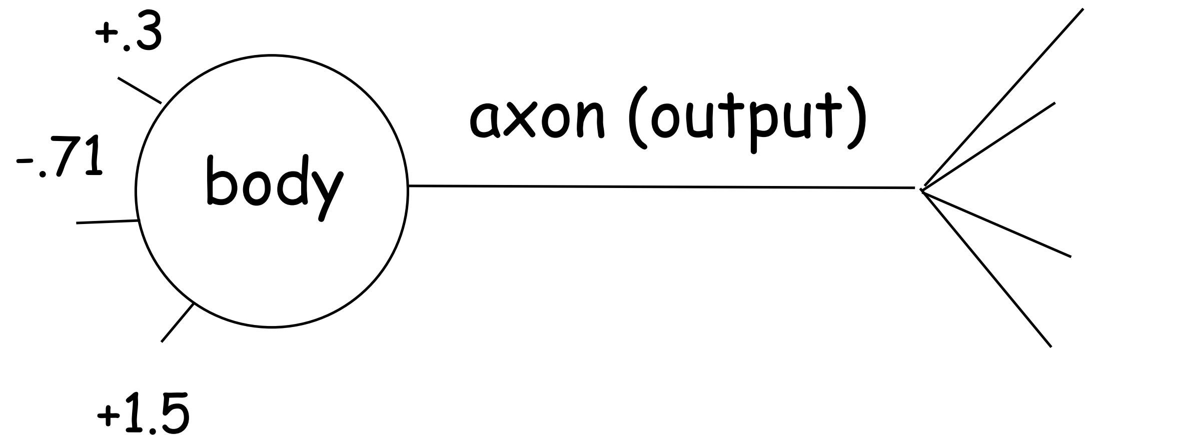
Synaptic Strength or Weight

- 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 depict them as associated with the **input** side of the neuron.

Neuron schematic with weights

dendrites (input)
with **weight** values

synapses

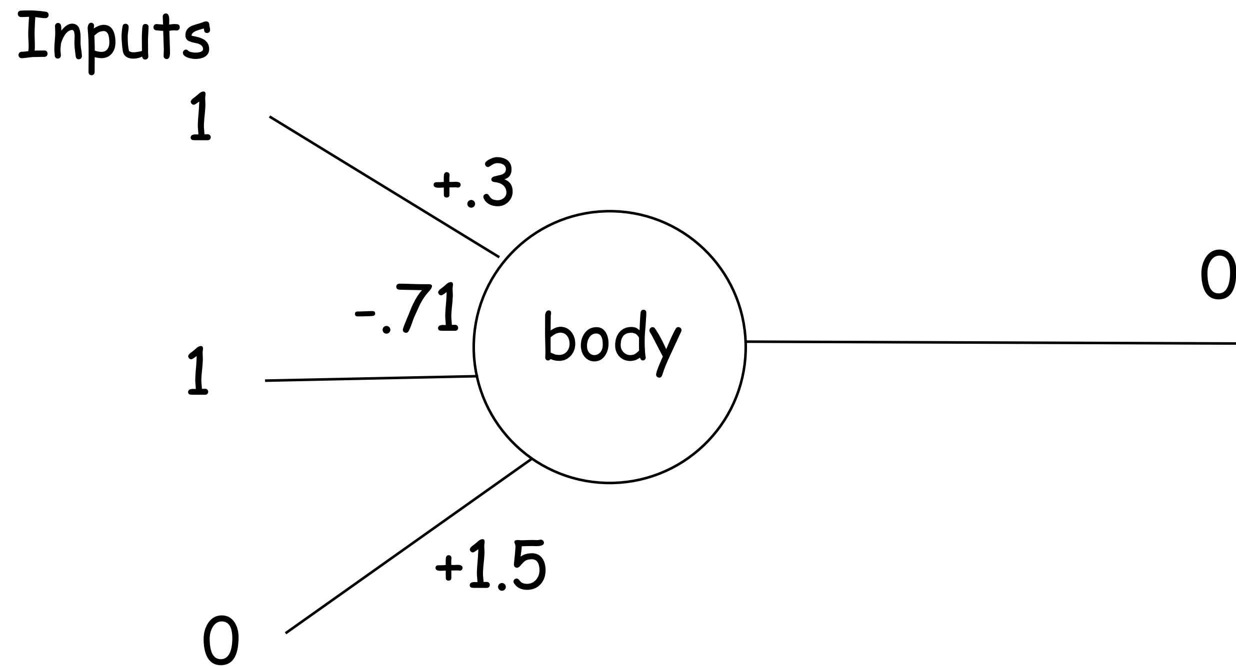


Threshold Logic

- 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.

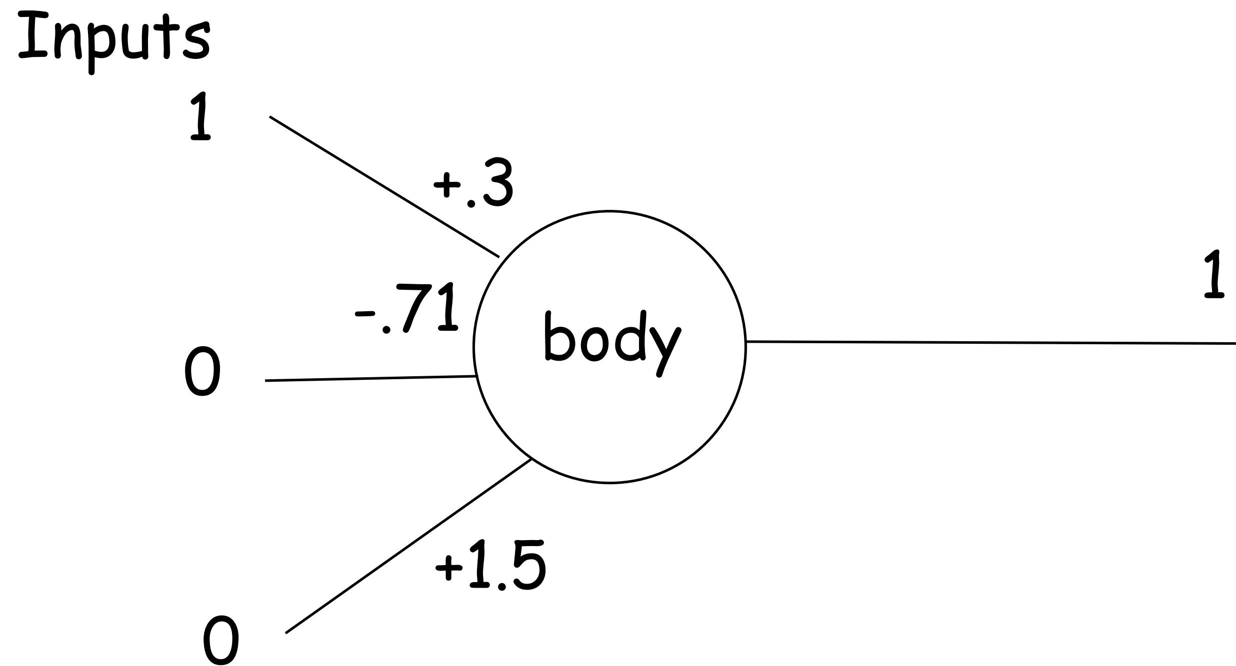


activation = 1*.3 + 1*(-.71) + 0 * 1.5 - (-0.2) = -0.21

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 Weight Description

- Inputs having positive weights are called "excitatory".
- Inputs having 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

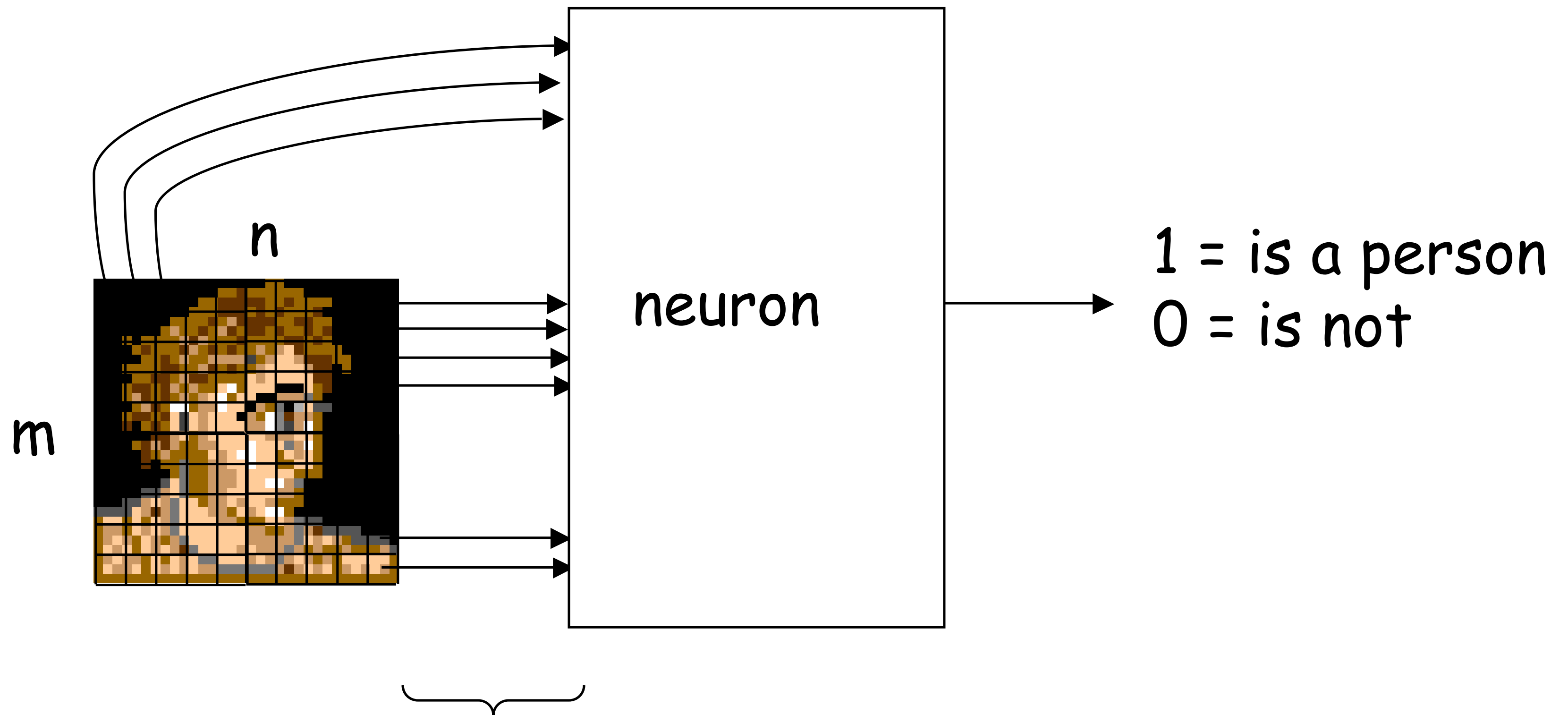
Rosenblatt's Perceptron, 1957

- A **perceptron** in simplest form is a linear threshold gate.
- More generally, it can have **reals** as inputs rather than just {0, 1}. But it still has {0, 1} as output.
- 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 vectors, each with a desired response
    (1 = in the set, 0 = not in the set),
    determine the weights so that a perceptron gives the desired response (and **generalizes** in an appropriate way to unseen inputs)
  - An input vector could be a retinal image, for example.

Retinal Image Classification



vector of m x n bits, RGB, or gray-scale levels.

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?

Existence

- In terms of switching logic, a perceptron is a "linear threshold gate" (LTG).
- As the number of inputs increases, the likelihood of realizing a function as an LTG diminishes.

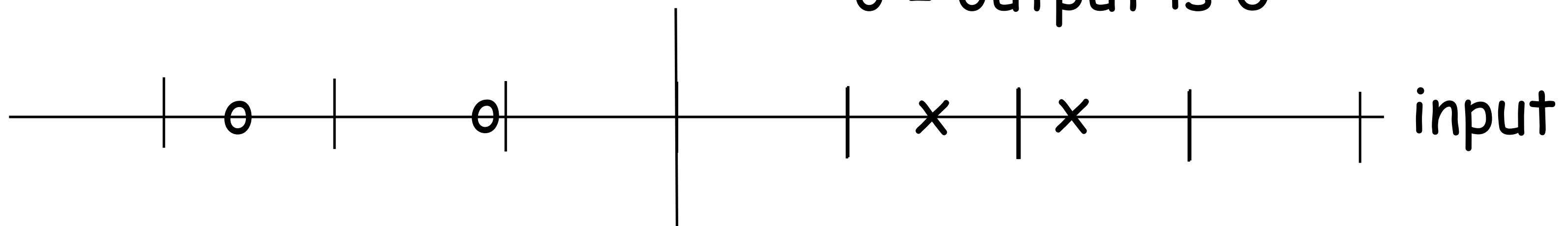
n	NUMBER OF THRESHOLD FUNCTIONS B_n	TOTAL NUMBER OF BOOLEAN FUNCTIONS $\left(2^{2^n}\right)$
1	4	4
2	14	16
3	104	256
4	1,882	65,536
5	94,572	4.3×10^{9}
6	15,028,134	1.8×10^{19}
7	8,378,070,864	3.4×10^{38}
8	17,561,539,552,946	1.16×10^{77}

1-dimensional version

- 1 input, 1 weight, 1 threshold

$$\text{output} = \begin{cases} 1 \text{ if } w\,x > q \\ 0 \text{ otherwise} \end{cases}$$

x = output is 1
o = output is 0



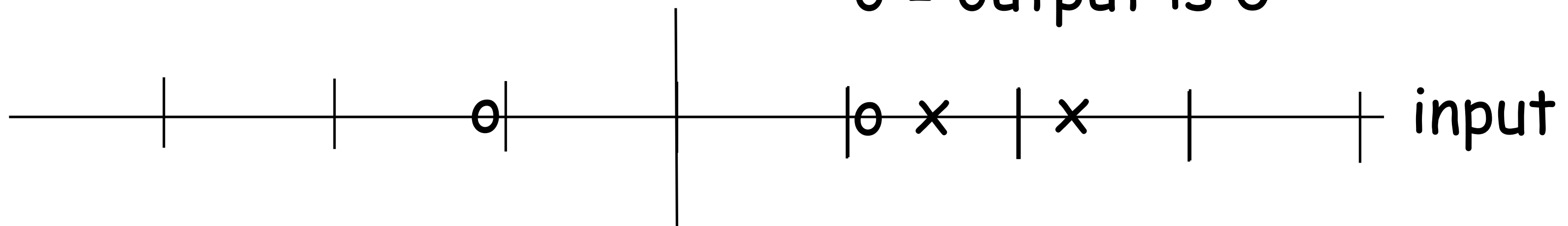
Solvable by Perceptron? What are suitable w and q?

1-dimensional version

- 1 input, 1 weight, 1 threshold

$$\text{output} = \begin{cases} 1 \text{ if } w\,x > q \\ 0 \text{ otherwise} \end{cases}$$

x = output is 1
o = output is 0



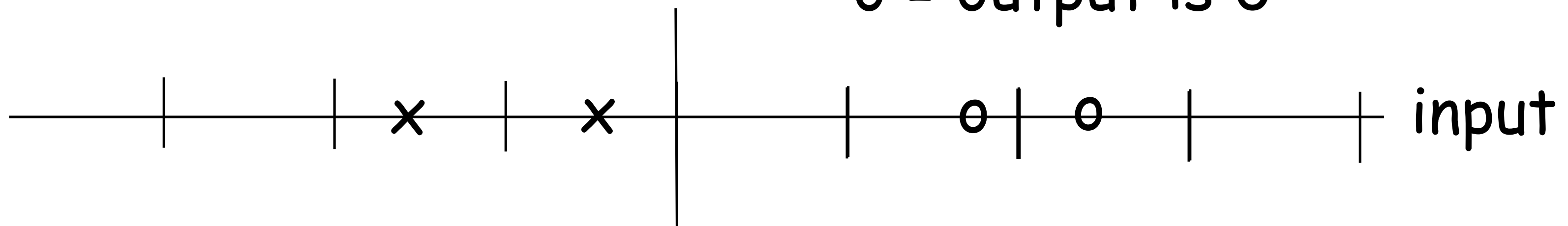
Solvable by Perceptron? What are suitable w and q?

1-dimensional version

- 1 input, 1 weight, 1 threshold

$$\text{output} = \begin{cases} 1 \text{ if } w\,x > q \\ 0 \text{ otherwise} \end{cases}$$

x = output is 1
o = output is 0



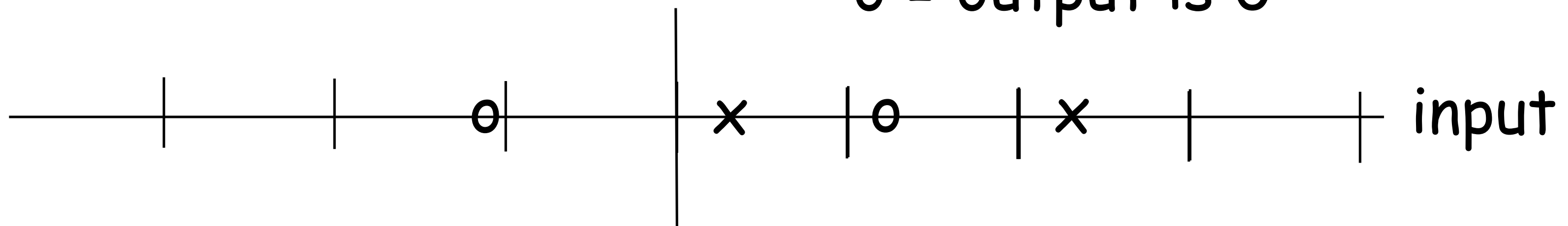
Solvable by Perceptron? What are suitable w and q?

1-dimensional version

- 1 input, 1 weight, 1 threshold

$$\text{output} = \begin{cases} 1 \text{ if } w\,x > q \\ 0 \text{ otherwise} \end{cases}$$

x = output is 1
o = output is 0



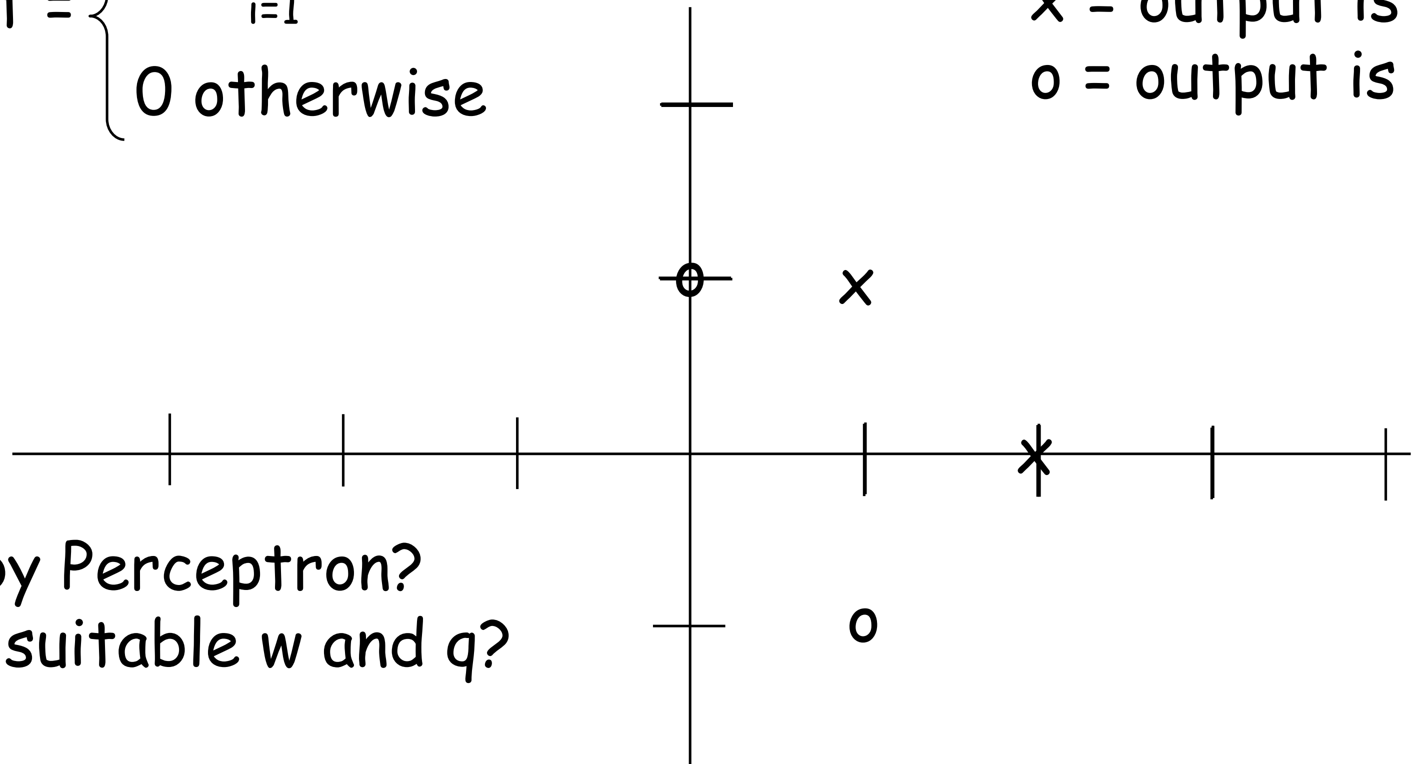
Solvable by Perceptron? What are suitable w and q?

2-dimensional version

- 2 inputs, 2 weights, 1 threshold

$$\text{output} = \begin{cases} 1 \text{ if } \sum_{i=1}^{2} w_i x_i > q \\ 0 \text{ otherwise} \end{cases}$$

x = output is 1
o = output is 0



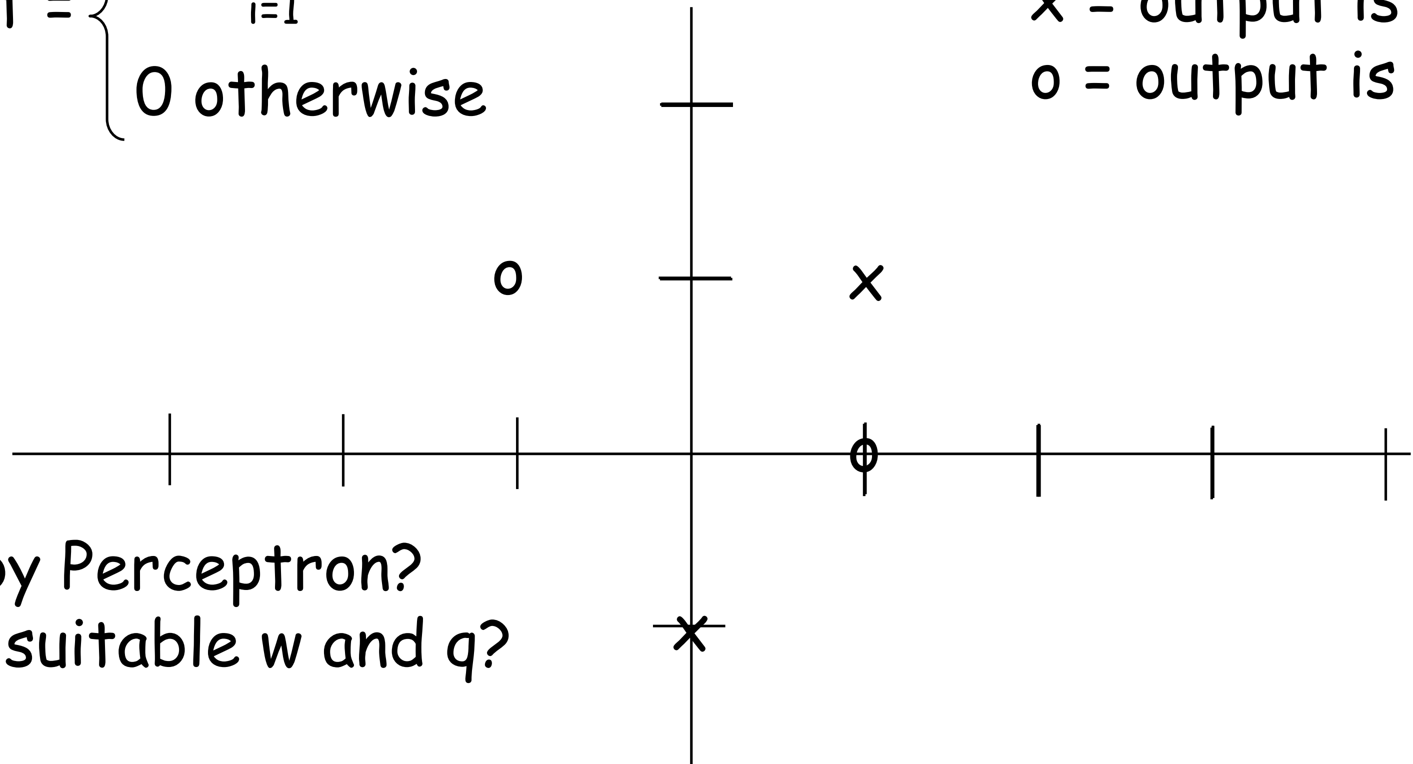
Solvable by Perceptron?
What are suitable w and q?

2-dimensional version

- 2 inputs, 2 weights, 1 threshold

$$\text{output} = \begin{cases} 1 \text{ if } \sum_{i=1}^{2} w_i x_i > q \\ 0 \text{ otherwise} \end{cases}$$

x = output is 1
o = output is 0



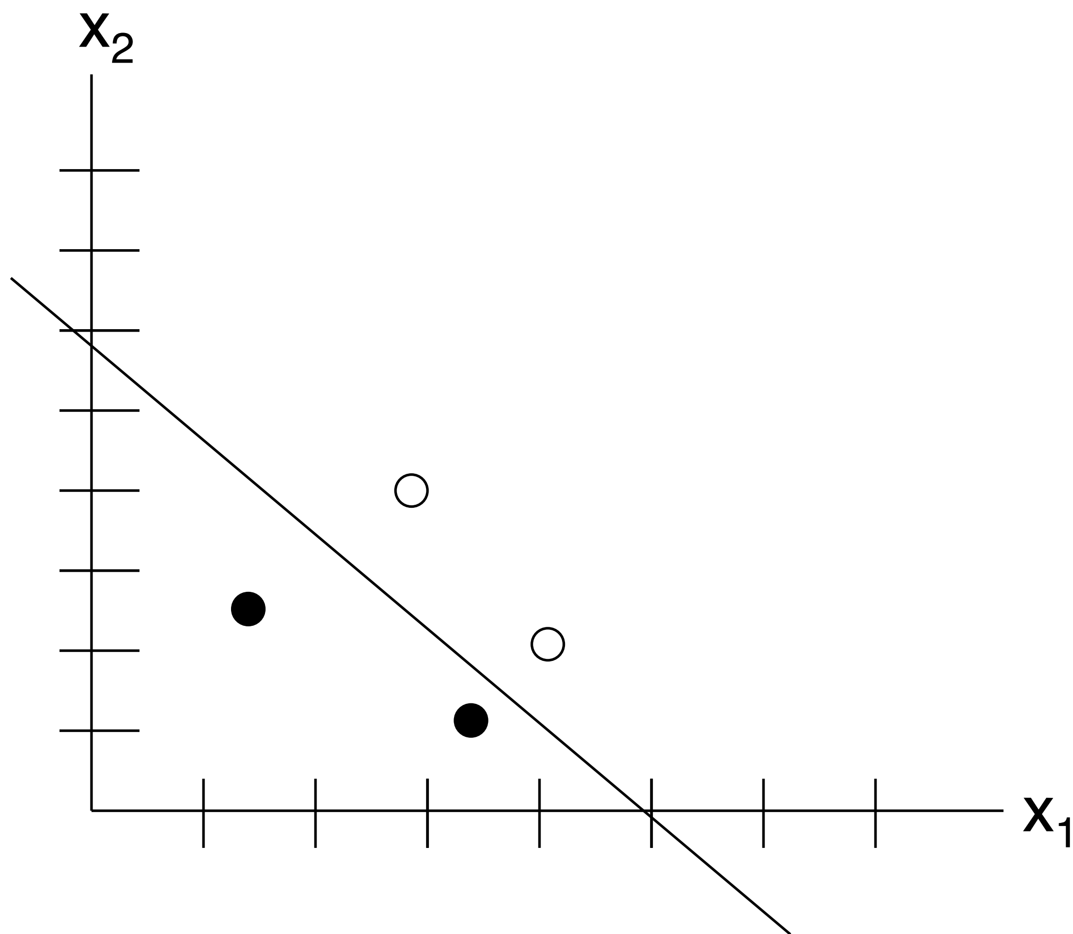
Solvable by Perceptron?
What are suitable w and q?

General Line Equation

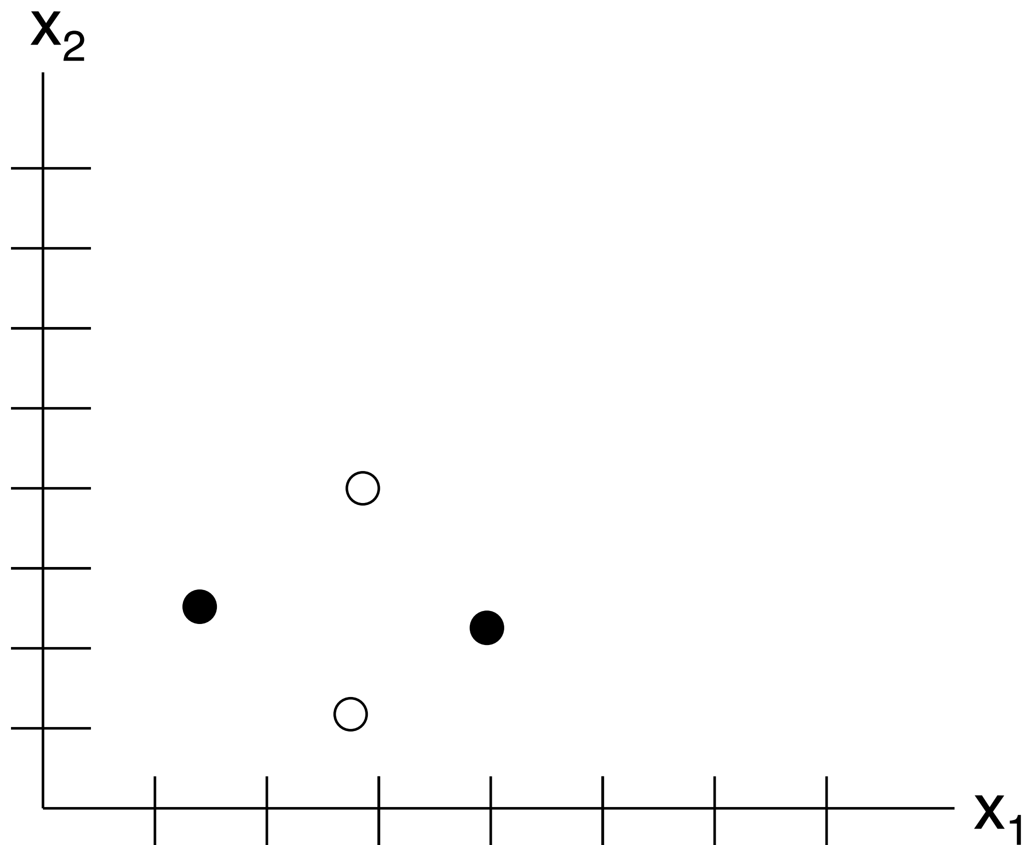
- Output is
  - 1 if $w_1 x_1 + w_2 x_2 > q$
  - 0 otherwise

- When will $\{w_i\}$ and q exist?
- Exactly when a straight line can be drawn that separates the points ("linearly separable")
- $w_1 x_1 + w_2 x_2 = q$ is the equation of such a line.

Linearly Separable



Non-Linearly Separable



Generalizing to n dimensions

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

  $$w_1 x_1 + w_2 x_2 + \dots + w_n x_n = q$$

  separates the points:

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

Linear Separability

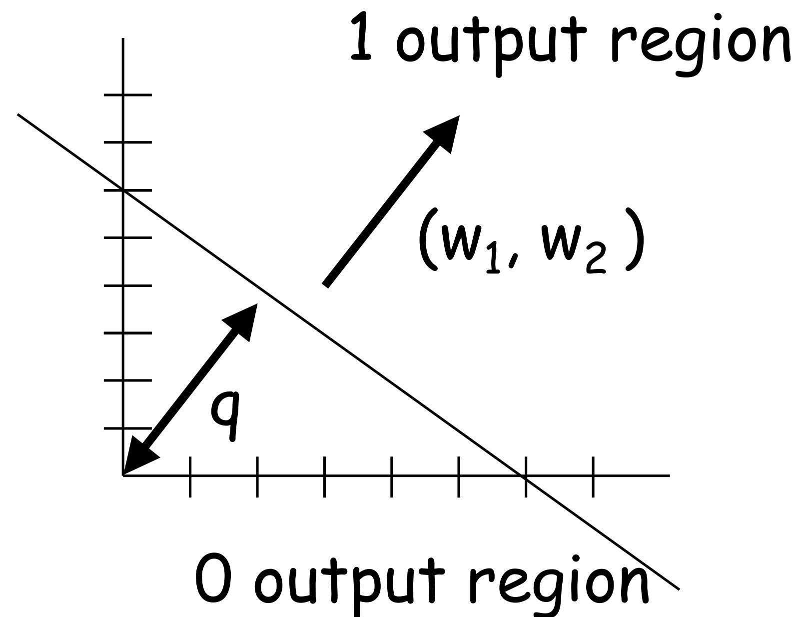
- The equation
  $w_1 x_1 + w_2 x_2 + ... + w_n x_n = q$
  defines a *hyperplane* in n-space

- 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.



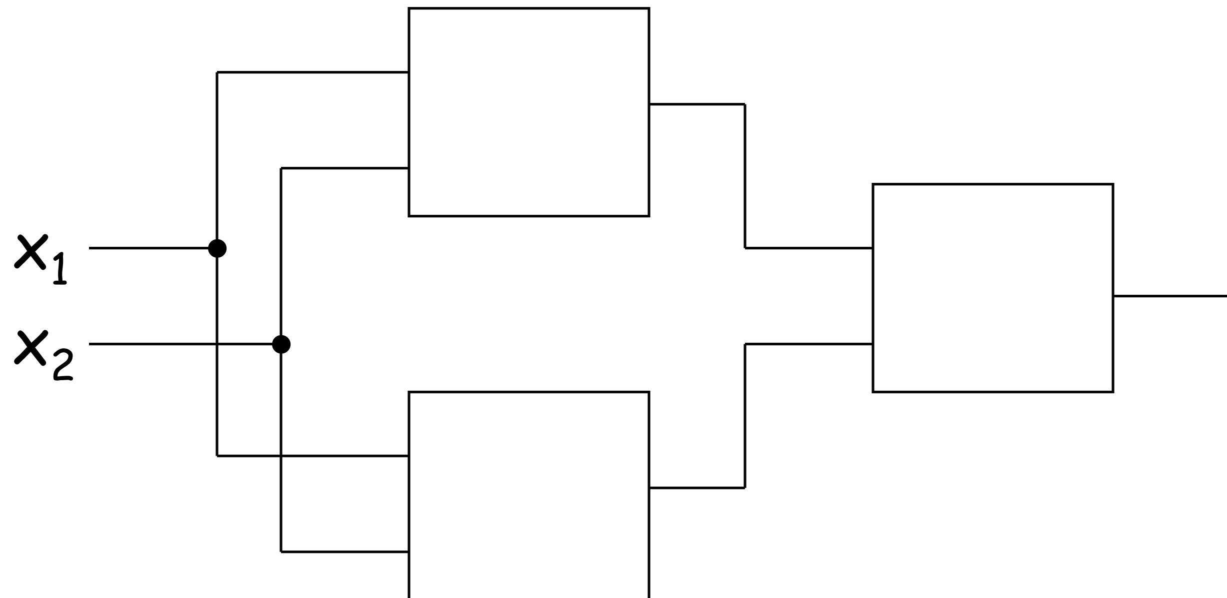
Perceptron Summary

- 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.

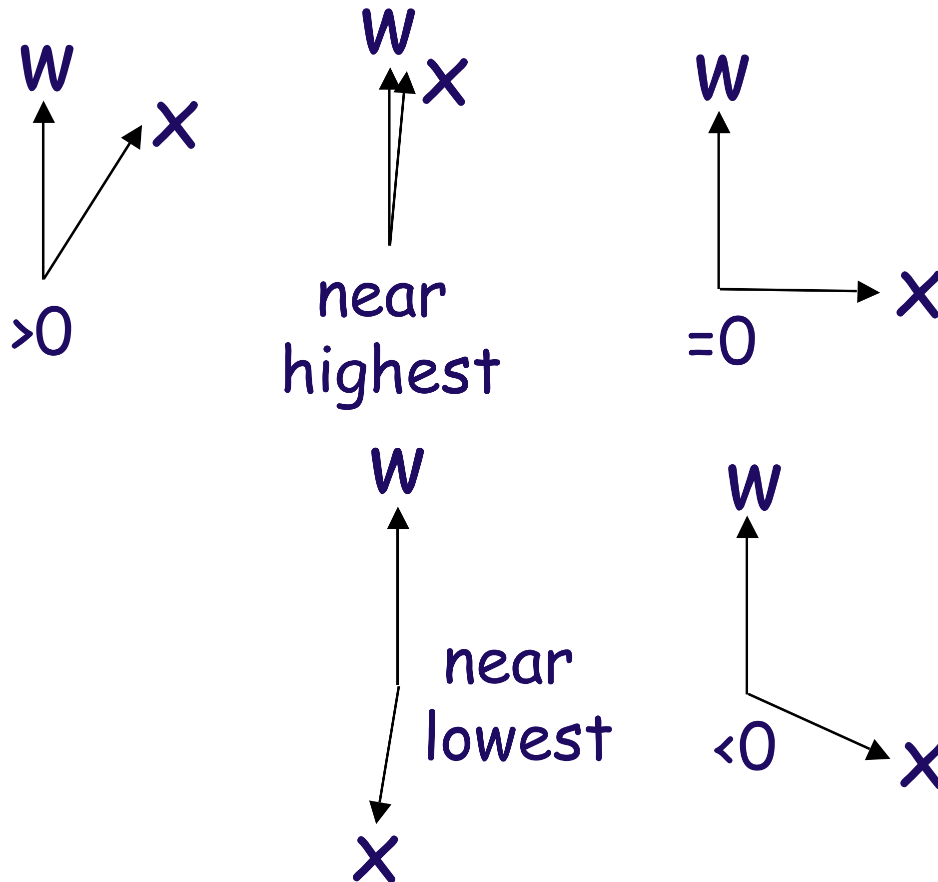
Weight Adjustment

- 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.

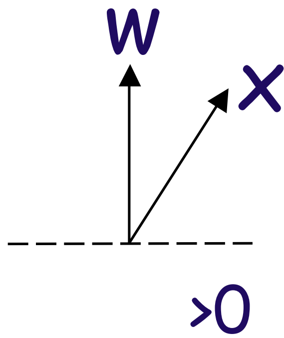
Inner-Product Interpretation

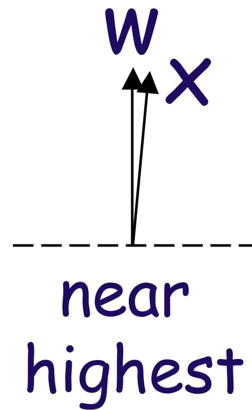
- $w_1 x_1 + w_2 x_2 + ... + w_n x_n$ is the **inner product wx** of two **vectors**, **w** and **x**.
- The more closely-aligned the directions of these vectors are, the higher the value.
- In fact, $\mathbf{wx} = |\mathbf{w}|\ |\mathbf{x}| \cos\theta$ where is the **angle** between w and x and $|\mathbf{w}|$ is the **length** of w.

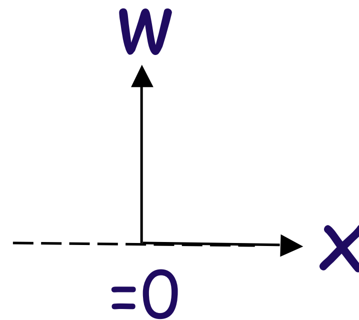
Values of wx

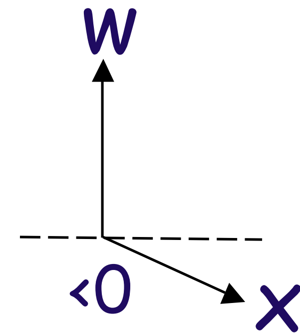


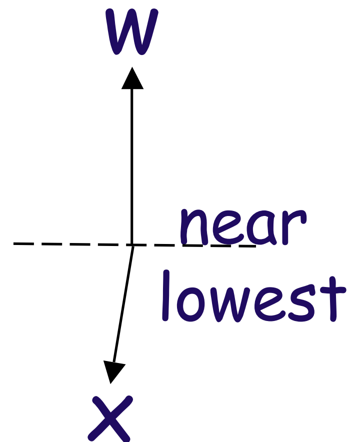
w as normal to hyperplane





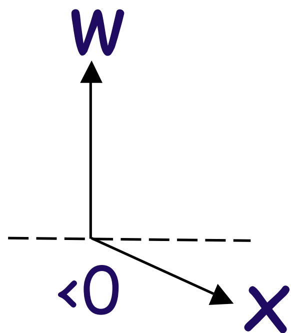


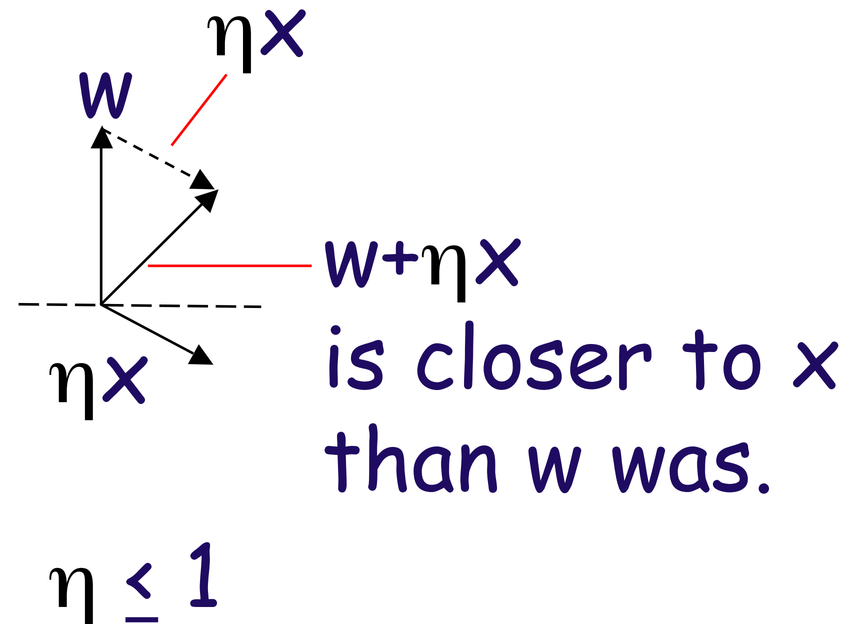




Adjusting w by vector addition

If an *increase* of wx is desired:





Adjusting w by vector addition

If a *decrease* of wx is desired:

w-ηx
is farther
from x
than w was.

$\eta < 1$

Perceptron Learning Rule

- If wx is negative, but it should be positive, add αx to w.
- If wx is positive, but it should be negative, subtract αx from w.
- If wx is as desired, do nothing.

Perceptron Training Semi-Algorithm

- Keep applying the learning rule, until all examples are correctly classified.
- Effectively this rotates the weight vector, and thus the hyperplane to which it is a normal, into an orientation where the examples are correctly classified.
- **This process only terminates if the points are linearly separable.** Therefore, it is best to impose an artificial limit on the number of steps just in case.

Training for Threshold

- In addition to rotating the hyperplane, we also need to adjust its distance from the origin.

- We can handle this along with adjusting other weights by making the dimension 1 higher and treating the threshold as another weight, against a constant input of -1.

- We'll provide details in a moment.

Desired Output d vs Actual Output a

- Define *actual* output
  $a(x_1, x_2, \ldots, x_n) =$
  1 if $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > q$
  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* ε:
- $\varepsilon = d(x_1, x_2, \ldots, x_n) - a(x_1, x_2, \ldots, x_n)$
  $=$ desired output - actual output
- So that
  - $\varepsilon = 0$ when the correct answer is given
  - $\varepsilon = 1$ when $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n < q$
    but $d(x_1, x_2, \ldots, x_n) = 1$
  - $\varepsilon = -1$ when $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > q$
    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:

$$w_1 x_1 + w_2 x_2 + ... + w_n x_n > q$$

iff

$$w_1 x_1 + w_2 x_2 + ... + w_n x_n - q > 0$$

iff

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

where w_0 is defined to be q 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 (1, -1, …) doesn't really matter, as long as it is constant and not 0.

Perceptron training (continued)

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

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

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

Perceptron training (continued)

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

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

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

Perceptron training (continued)

- Make $w_0 x_0 + w_1 x_1 + w_2 x_2 + ... + w_n x_n$ 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 η (called the "learning rate") of εx_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:

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

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

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

(ε = desired - actual, so

in the first case need to bring actual up,

in the second bring it down)

Learning Rate η

- η governs the rate at which the training rule converges toward the correct solution.
- Typically $\eta \leq 1$.
- Too small an η produces slow convergence.
- Too large of an η 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 η = 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
    - $[d(x_1, x_2, ..., x_n), -1, x_1, x_2, ..., x_n]$
    - (d is the desired output, -1 the phantom input)
  - An initial weight vector $[w_0, w_1, w_2, ..., w_n]$
    - (w_0 is the threshold)
  - A learning rate η

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. Or just start with all 0's (but this won't work in more advanced models).

Correctness of the Perceptron Training Algorithm assuming linear separability

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

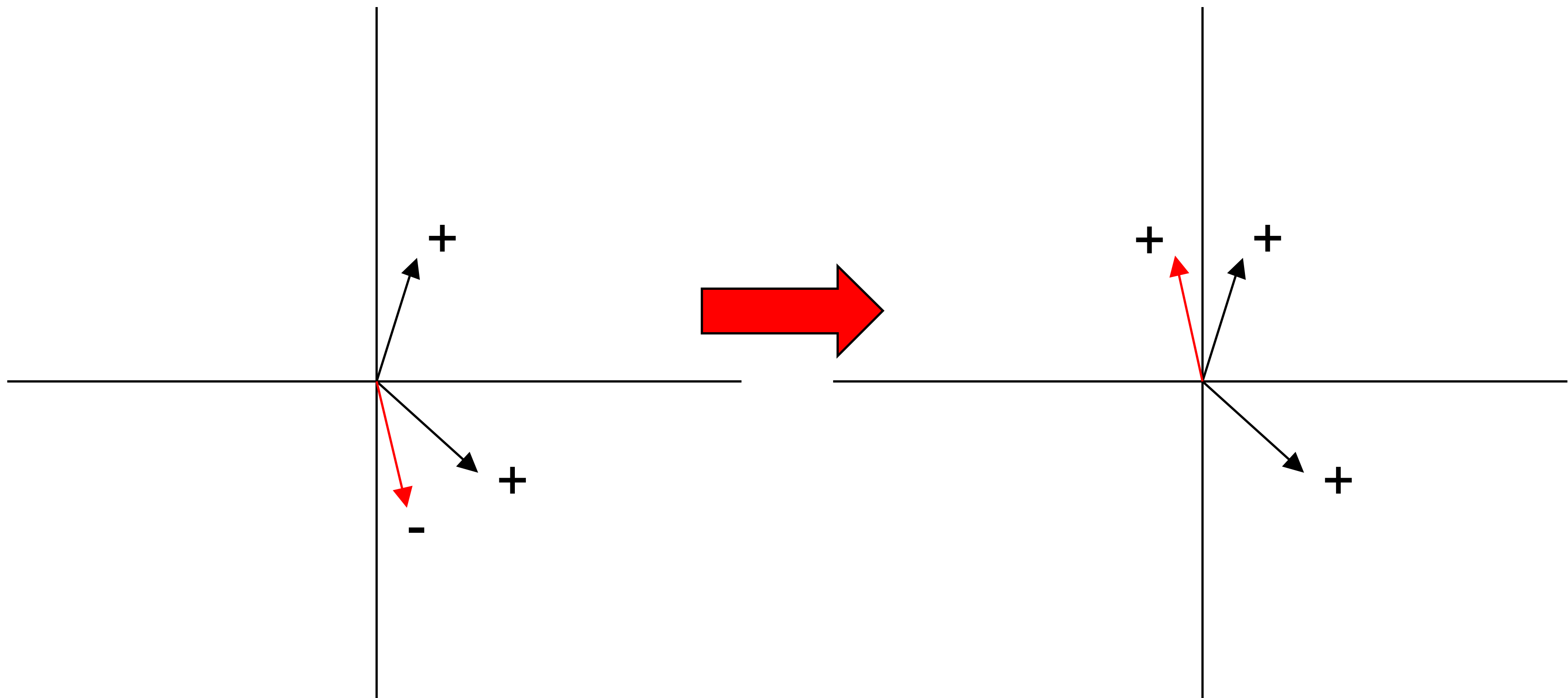
- The weight vector w can also be normalized, so that |w| = 1, by the same rationale.

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 of the sample.

Simplification

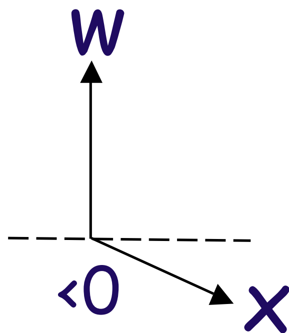


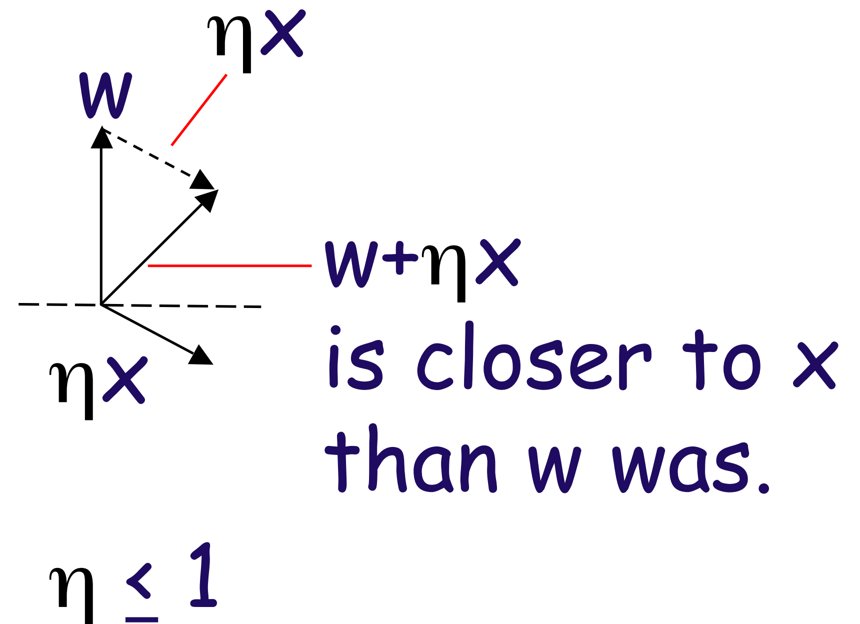
Visualization

- For normalized w and x, the value wx is the **cosine** of the angle θ between w and x.
- We want to find a weight vector which has a positive value of cos θ with each x.
- A training step consists of adding vector Δw (proportional to x) to w, to push w away if cos $\theta < 0$.
- We can assume the learning rate is $\eta = 1$ without loss of generality.
- We can assume the weights are all initially 0, without loss of generality.

Learning Based on Mis-Classified Samples

(from before) An *increase* of wx is desired:





Convergence Proof

- Assume all samples are positive and normalized.
- Assume the weights are normalized.
- **Claim**: If a weight vector w^* exists that correctly classifies all samples, one will be found by the perceptron training algorithm.
- WLOG, assume the learning rate η 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 all samples correctly).
- Let w_t be the weight vector after t steps of the algorithm.
- Consider step t+1, which must have resulted from some vector x_i being *misclassified* by w_t.
- Thus $w_{t+1} = w_t + x_i$ (because the error is 1 and $\eta = 1$).

Proof continues

- Consider ρ where
  $\cos \rho = w^* (w_{t+1} / |w_{t+1}|)$, which must be ≤ 1.

- Factoring 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, for all t
  $w^* w_{t+1} \geq w^*w_0 + \delta(t+1)$, where w_0 is the initial weight vector.

- So we have linear lower bound on $w^* w_{t+1}$, which translates to a **quadratic** lower bound on $|w_{t+1}|^2$.

Proof continues

- Again, $w^* w_{t+1} / | w_{t+1} | \leq 1$.
- Squaring the **denominator**,
  $| w_{t+1} |^2 = (w_t + x_i)(w_t + x_i)$
  $= |w_t|^2 + 2w_t x_i + |x_i|^2$
- 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,
  $| w_{t+1} |^2 \leq |w_0|^2 + (t + 1)$
- So we have a *linear* upper bound on $|w_t|^2$

Proof concludes

- From two inequalities derived on the previous two slides:

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

- The RHS as a function of t is bounded above.
- Hence there is a **maximum** t for which classification can be incorrect.
- Thus the algorithm terminates, because there are no incorrect classifications beyond that t.