Lecture 03

The Perceptron

And introduction to single-layer neural networks

STAT 453: Deep Learning, Spring 2020

Sebastian Raschka

http://stat.wisc.edu/~sraschka/teaching/stat453-ss2020/

Announcements

- Project groups (by next Tue), 3 members per group -- TA will set up a document where you can add your team member preferences

- Project topics (brainstorm with group members)

- HW1 (related to the Perceptron; more about that later)

- Piazza for questions, encouraged to help each other (but don't share your HW solutions)

From the News

GREGORY BARBER SCIENCE 01.22.2020 11:00 AM

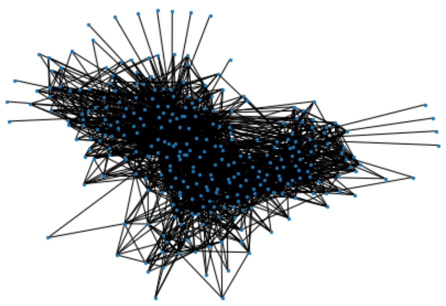
The Most Complete Brain Map Ever Is Here: A Fly's 'Connectome'

It took 12 years and at least $40 million to chart a region about 250 micrometers across—about the thickness of two strands of hair.

https://www.wired.com/story/most-complete-brain-map-ever-is-here-a-flys-connectome/

After this lecture, you will be able to implement your first neuron model for making predictions!*

*Sorry, we are not going to implement a fruit fly brain, but if you are interested in a worm brain: "MNIST classification using the neuronal network of Caenorhabditis elegans"
https://github.com/vinayprabhu/Network_Science_Meets_Deep_Learning/blob/master/1_MNIST_C_Elegans.ipynb



Overview

1/5 -- Brains and neuron models

2/5 -- The perceptron learning rule

3/5 -- Optional: The perceptron convergence theorem

4/5 -- Geometric intuition

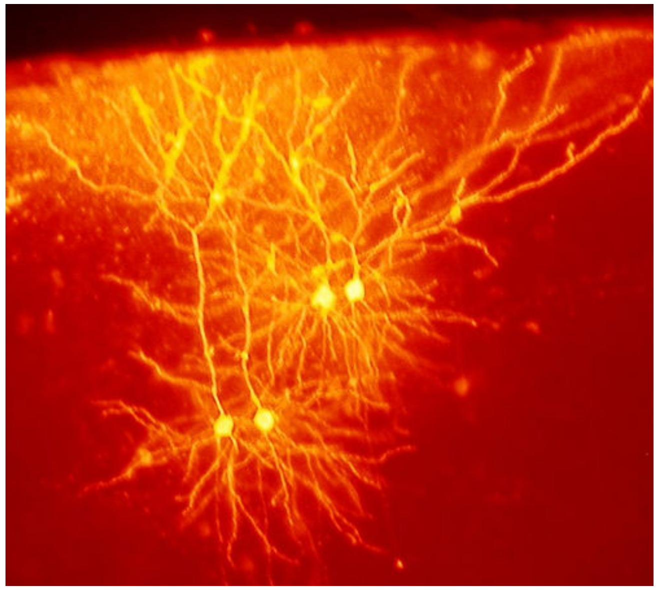
5/5 -- HW1

Do our brains use deep learning?

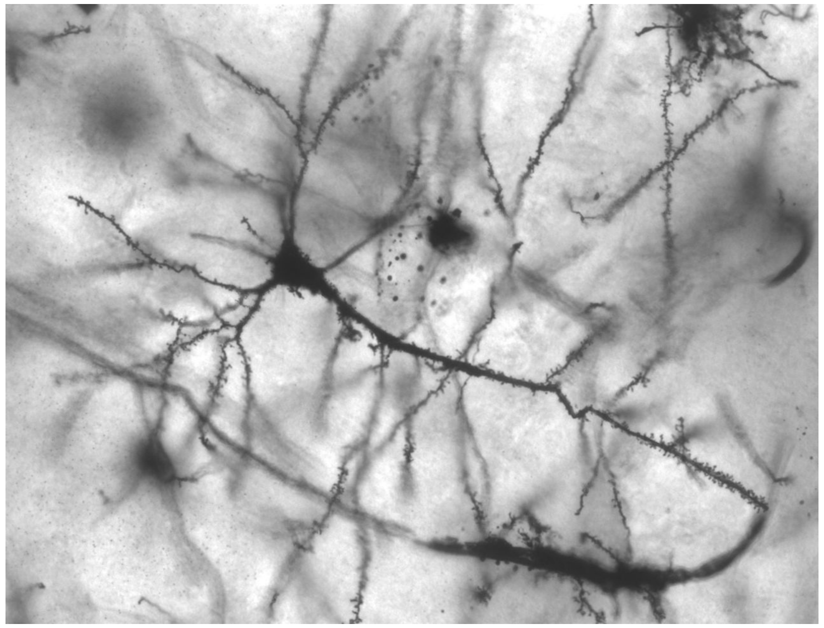
1/5 -- Brains and neuron models
2/5 -- The perceptron learning rule
3/5 -- Optional: The perceptron convergence theorem
4/5 -- Geometric intuition
5/5 -- HW1

Inspired by Biological Brains and Neurons

https://publicdomainpictures.net/en/view-image.php?image=130359&picture=human-brain



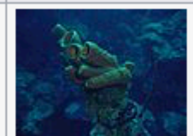
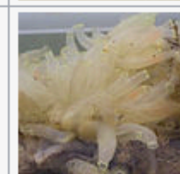
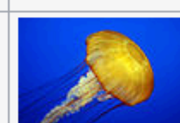
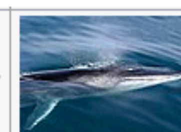
https://commons.wikimedia.org/wiki/Neuron#/media/File:Mouse_cingulate_cortex_neurons.jpg



https://commons.wikimedia.org/wiki/Neuron#/media/File:Pyramidal_hippocampal_neuron_40x.jpg

Source: https://media.mnn.com/assets/images/2016/10/plane-birds.jpg.1000x0_q80_crop-smart.jpg

Number of neurons in brains ...

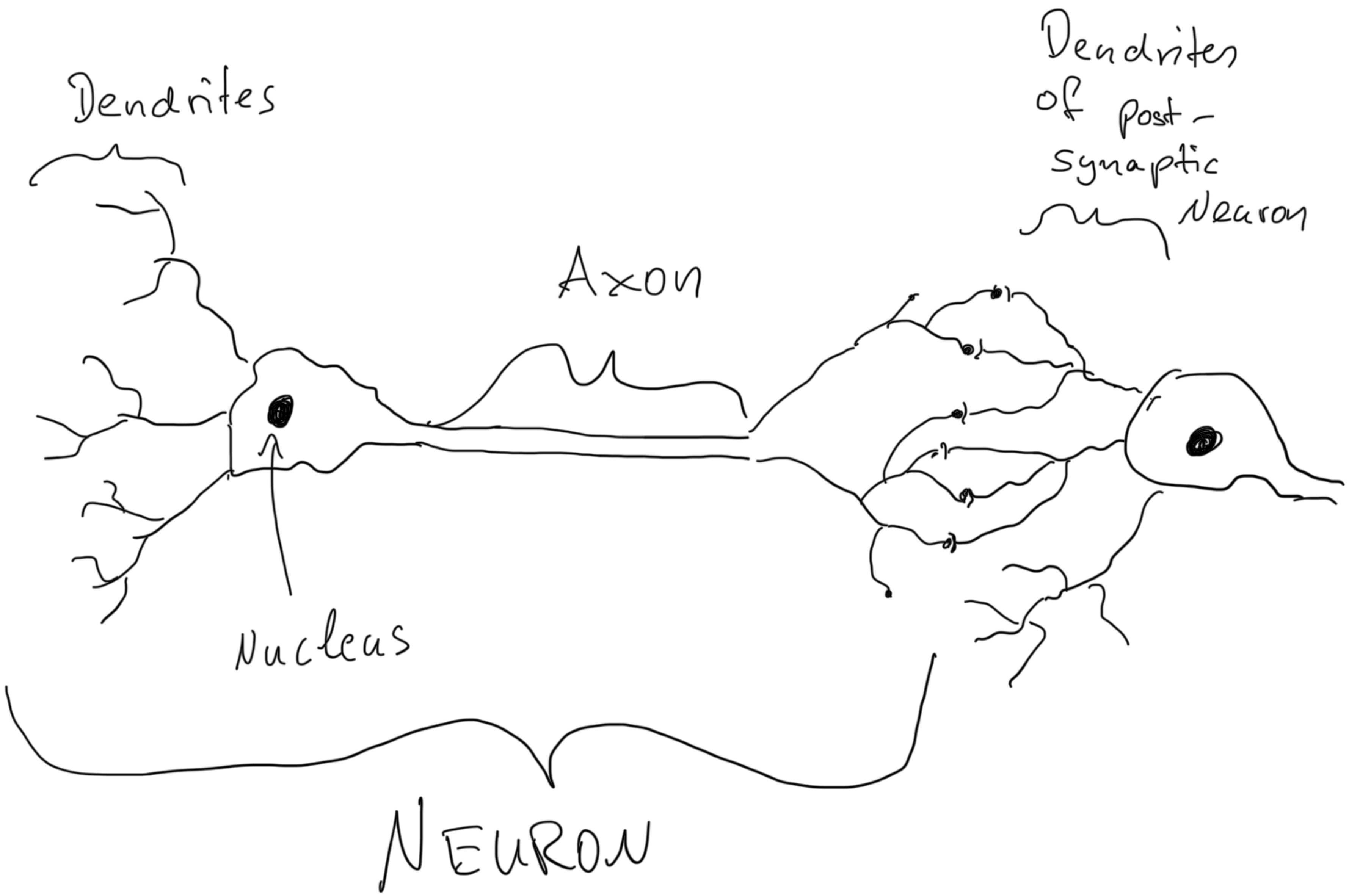
Name	Neurons in the brain/whole nervous system	Synapses	Details	Image	Source
Sponge	0				[3]
Trichoplax	0				[4]
Ciona intestinalis larva (sea squirt)	231	8617 (central nervous system only)			[5] [6]
Asplanchna brightwellii (rotifer)	about 200		Brain only		[7]
Caenorhabditis elegans (roundworm)	302	~7,500			[8]
Jellyfish	5,600		*Hydra vulgaris* (*H. attenuate*)		[9]
Fin whale	15,000,000,000		*Balaenoptera physalus*		[55]
Human	16,000,000,000		*Homo sapiens*: (For average adult) "The human cerebral cortex, with an average 1233 g and 16 billion neurons, is slightly below expectations for a primate brain of 1.5 kg, while the human cerebellum, at 154 g and 69 billion neurons, matches or even slightly exceeds the expected"		[43][2][56]
Long-finned pilot whale	37,200,000,000		*Globicephala melas*: "For the first time, we show that a species of dolphin has more neocortical neurons than any mammal studied to date including humans."		[57]

Source: https://en.wikipedia.org/wiki/List_of_animals_by_number_of_neurons

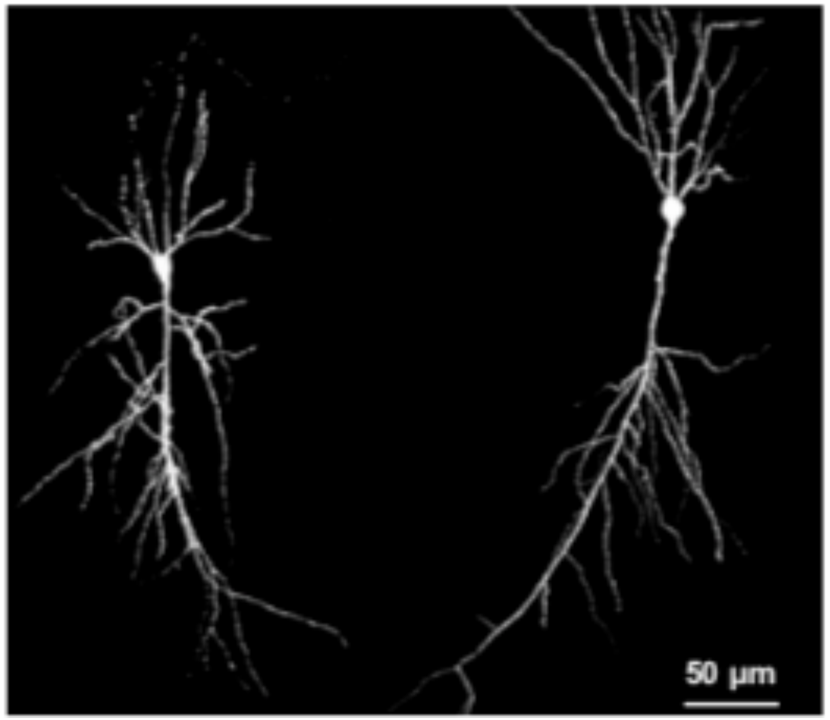
On a sidenote:

Name	Short scale (US, Eastern Europe, English Canadian, Australian, and modern British)	Long scale (Western, Central Europe, older British, and French Canadian)
Million	10^6	10^6
Milliard		10^9
Billion	10^9	10^{12}
Billiard		10^{15}
Trillion	10^{12}	10^{18}

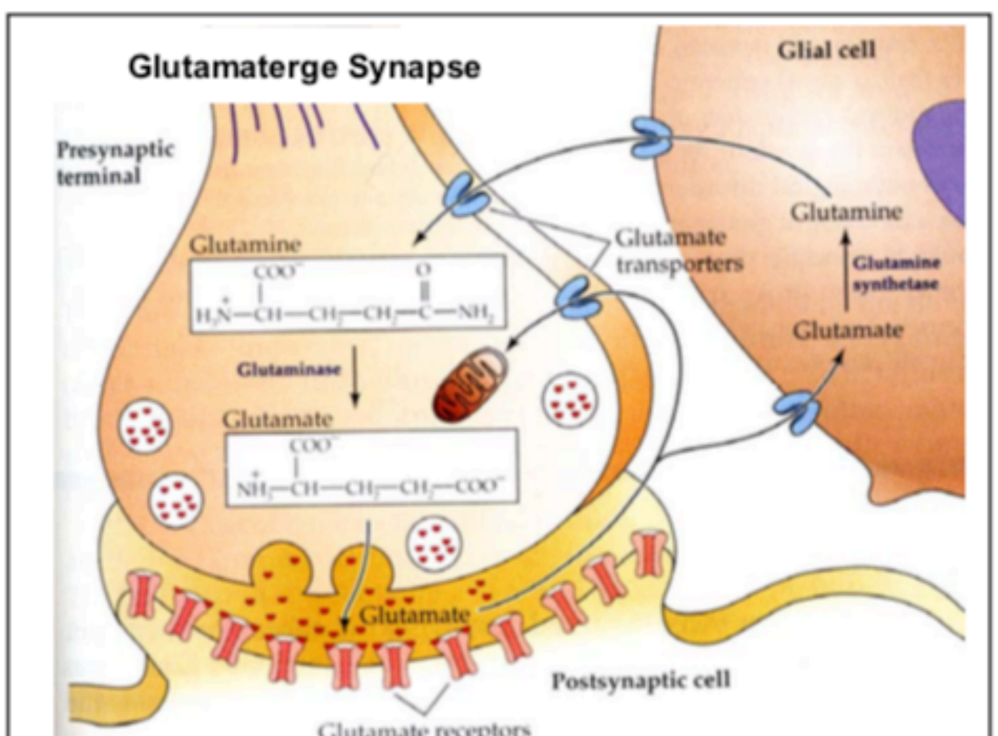
A Biological Neuron



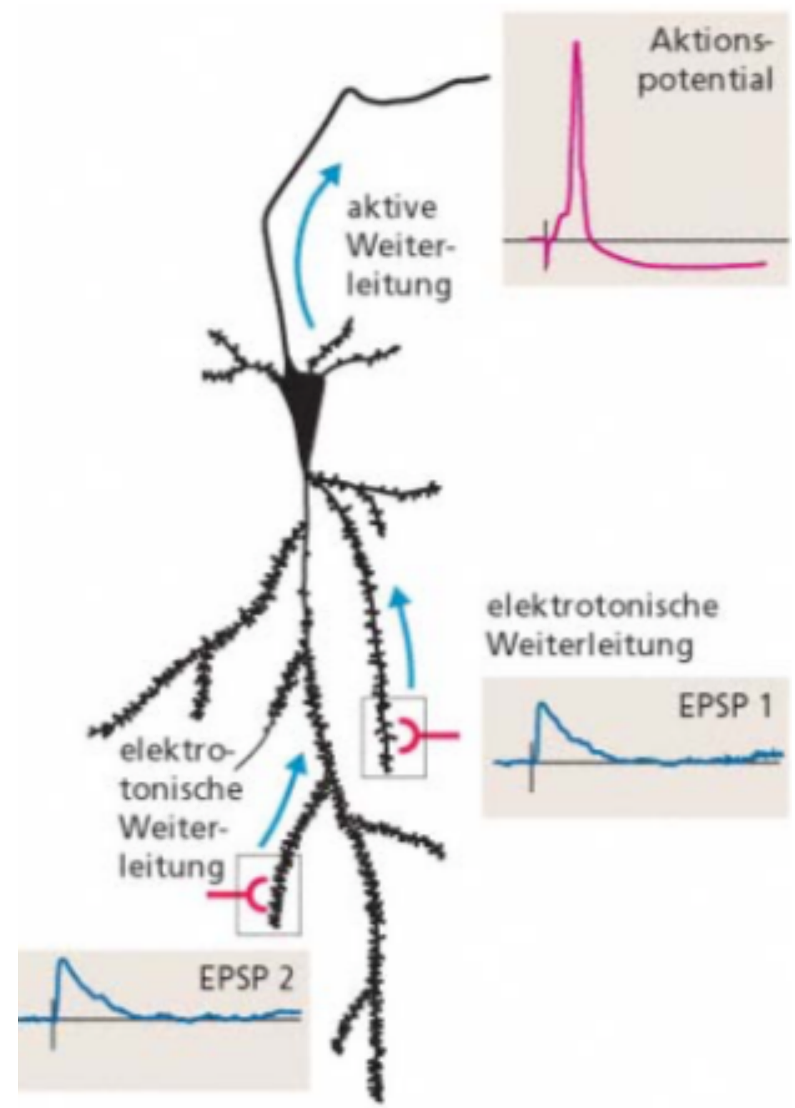
Biological Neurons



Pyramidal neuron cells in mouse cortex



Synaptic connection is chemical

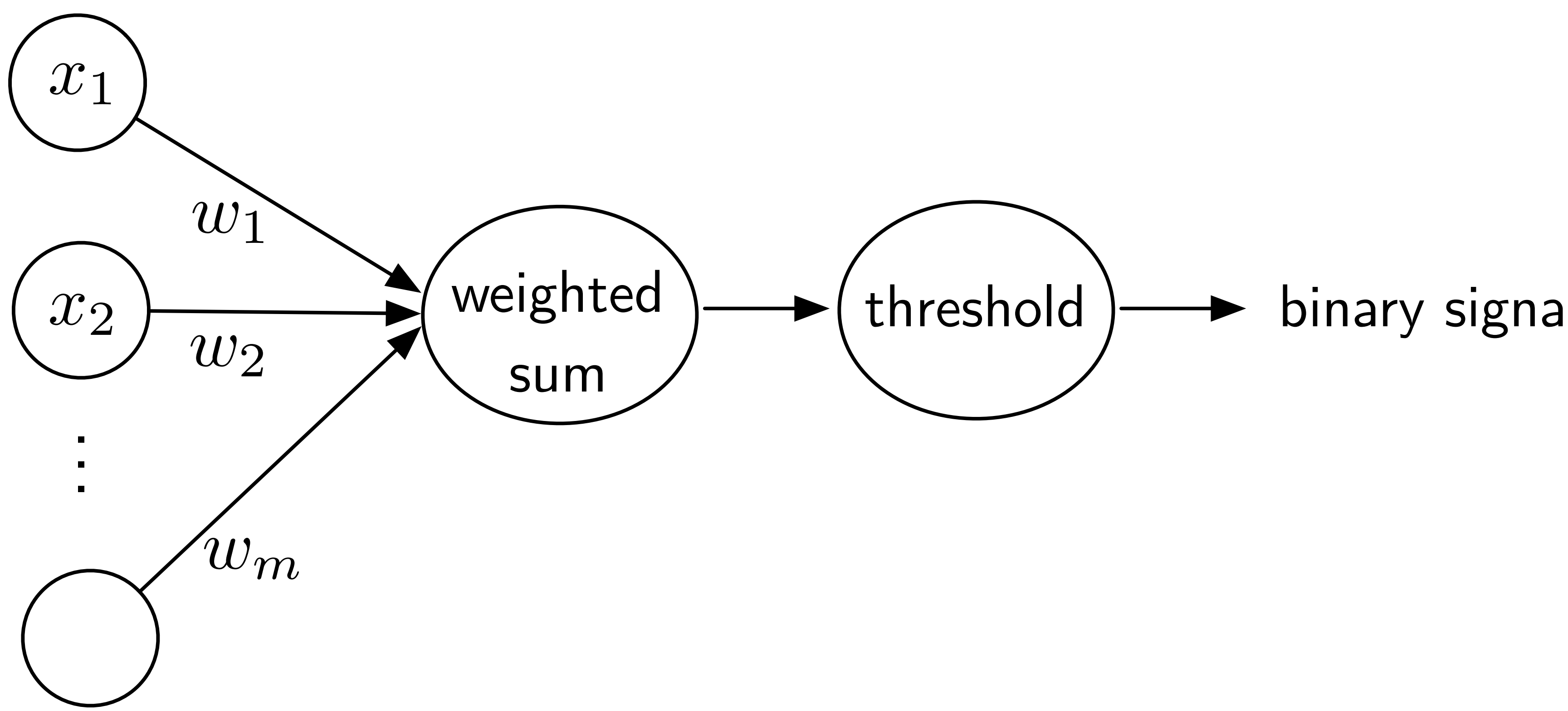


electrical postsynaptic potential accumulates; when it reaches a threshold => action potential signal

McCulloch & Pitts Neuron Model

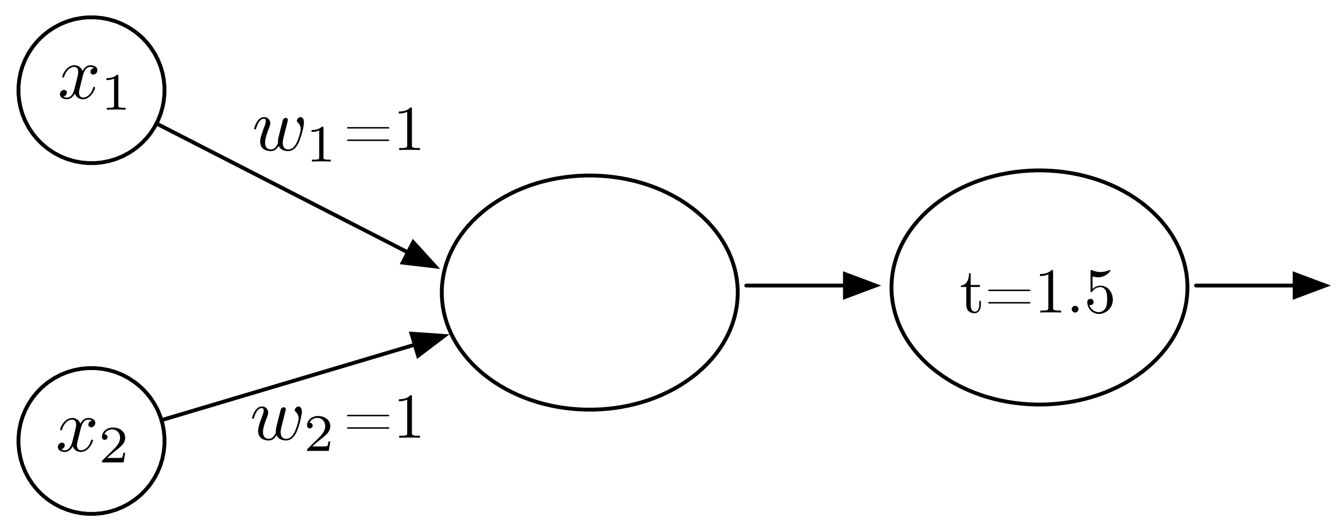
A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY

WARREN S. MCCULLOCH and WALTER H. PITTS 1943



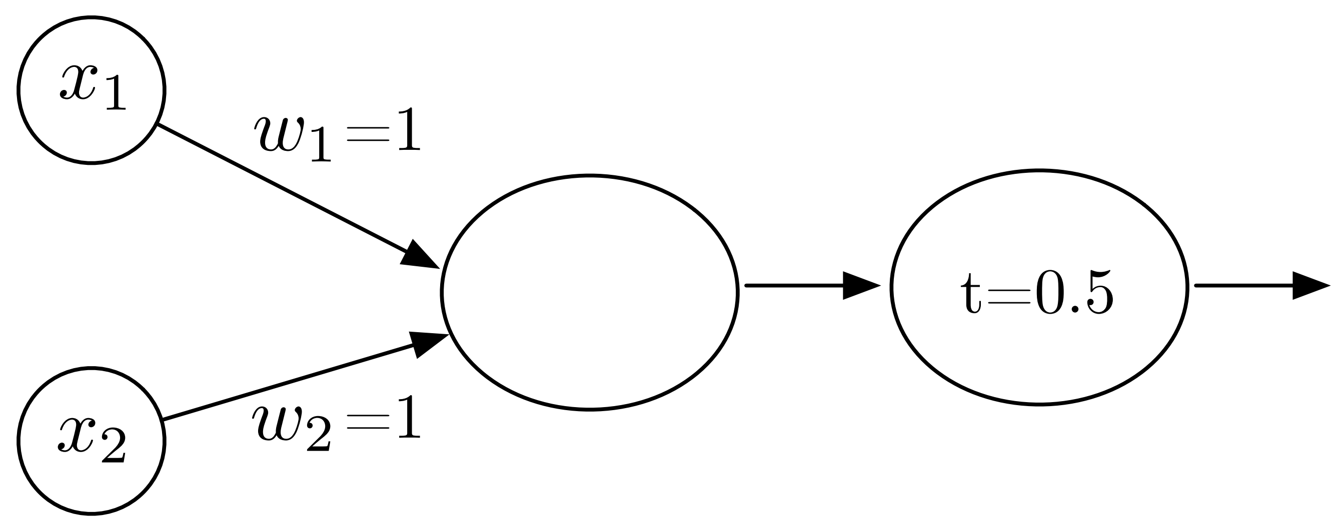
Logical AND Gate

x_1	x_2	Out
0	0	0
0	1	0
1	0	0
1	1	1



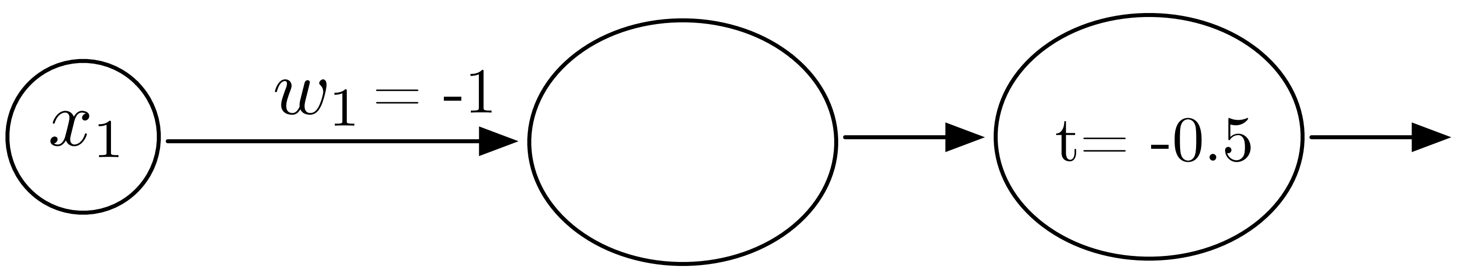
Logical OR Gate

x_1	x_2	Out
0	0	0
0	1	1
1	0	1
1	1	1



Logical NOT Gate

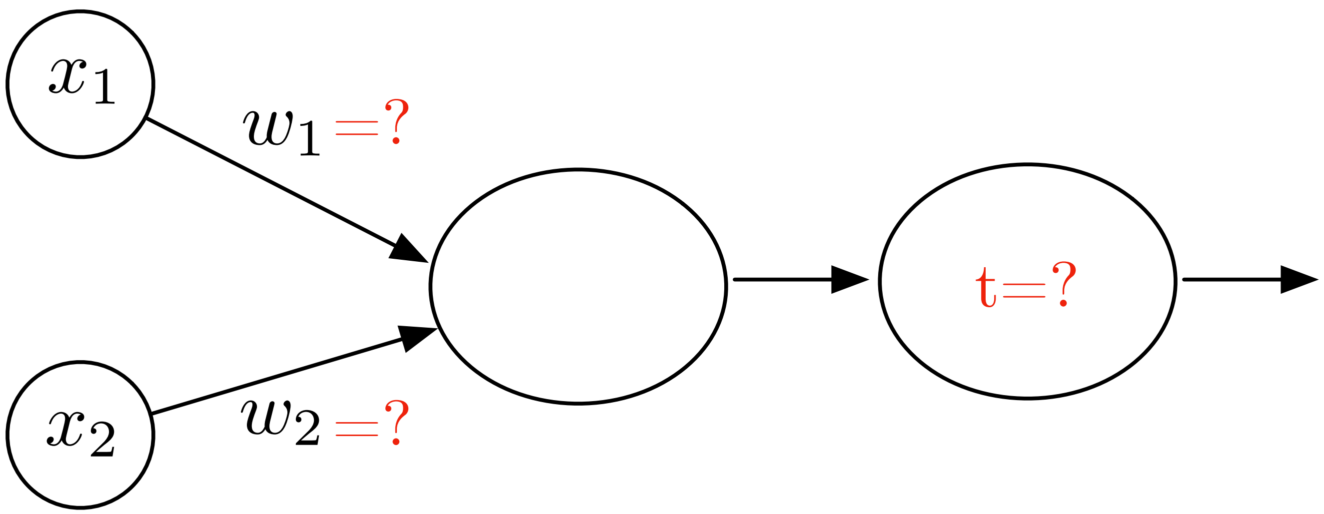
x_1	Out
0	1
1	0



Logical XOR Gate

(Take-home exercise)

x_1	x_2	Out
0	0	0
0	1	1
1	0	1
1	1	0



Training Single-Layer Neural Networks

Rosenblatt's Perceptron

A learning rule for the computational/mathematical neuron model

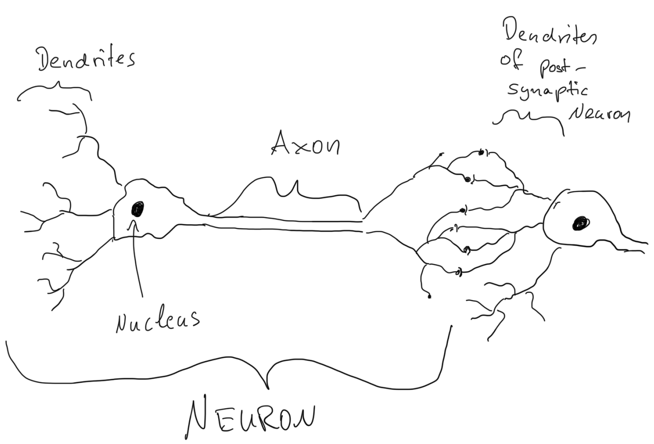
Rosenblatt, F. (1957). *The perceptron, a perceiving and recognizing automaton. Project Para*. Cornell Aeronautical Laboratory.

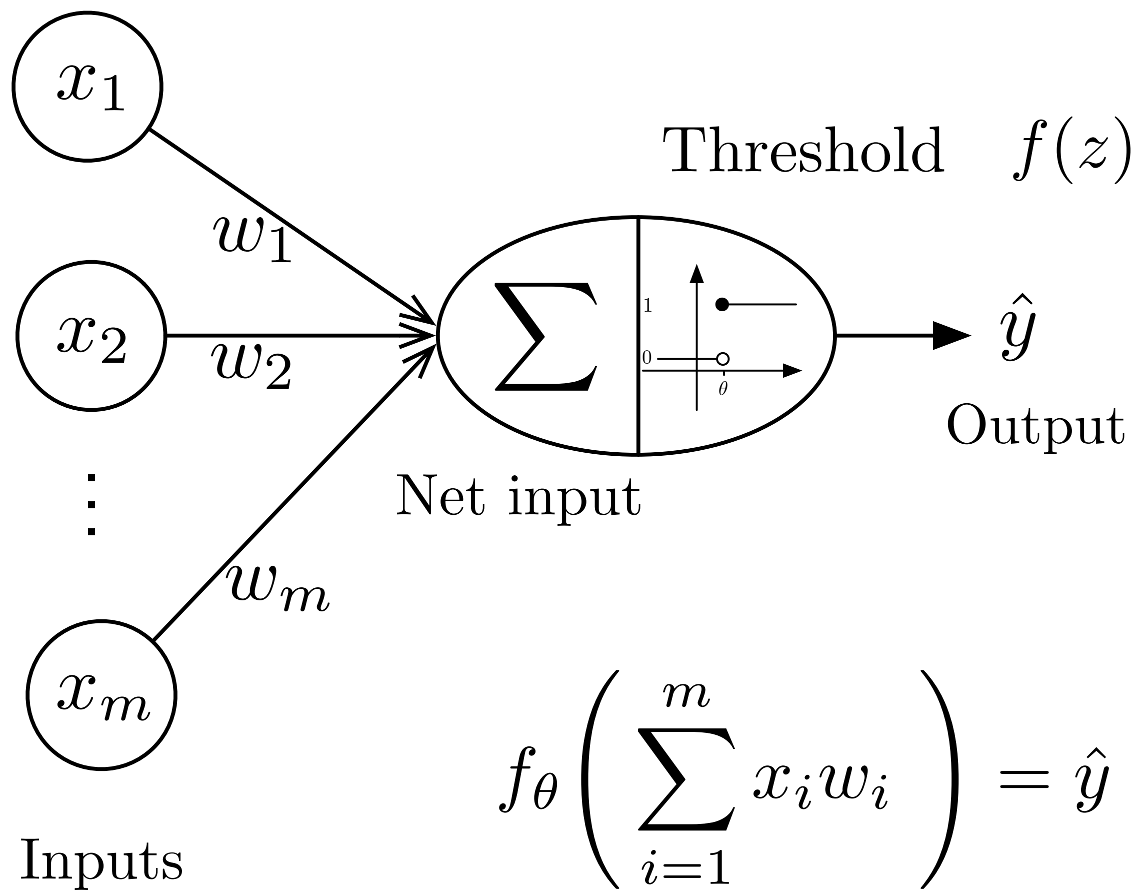
Source: http://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/Members/wilex4/Rosen-2.jpg

Perceptron Variants

Note that Rosenblatt (and later others) proposed many variants of the Perceptron model and learning rule.
We discuss a "basic" version;
let's say,
"Perceptron" = "a classic Rosenblatt Perceptron"

A Computational Model of a Biological Neuron





$$f(z) = \begin{cases} 0, & z \leq \theta \\ 1, & z > \theta \end{cases}$$

$$f_\theta\left(\sum_{i=1}^{m} x_i w_i\right) = \hat{y}$$

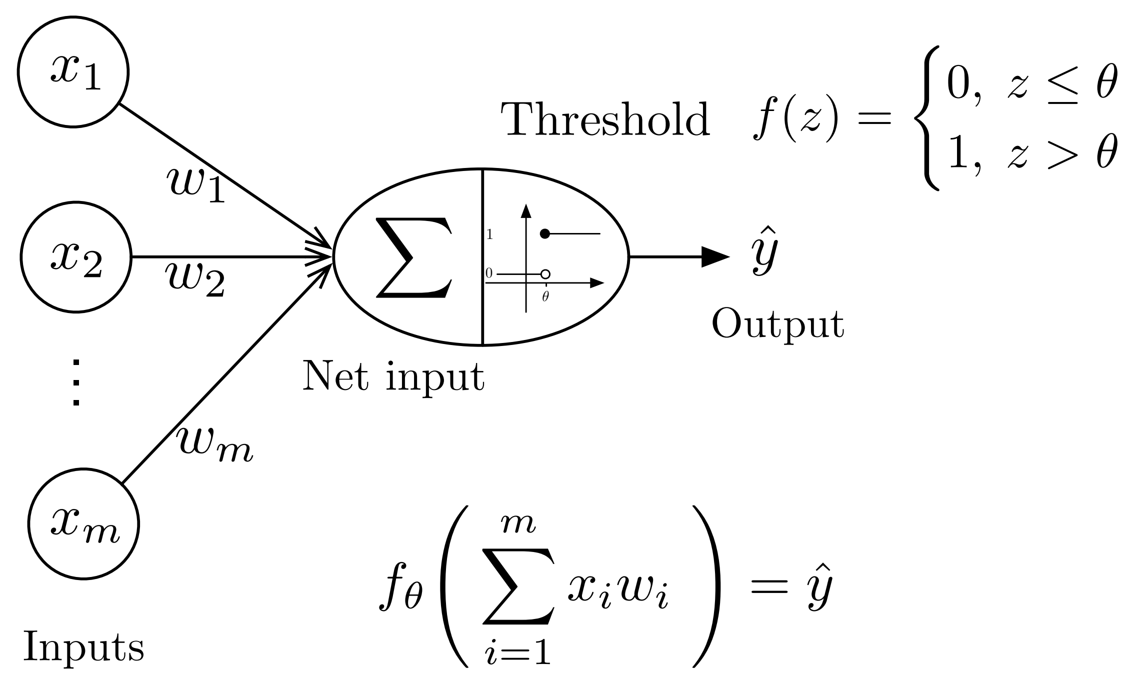
Terminology

General (logistic regression, multilayer nets, ...):

- Net input = weighted inputs, z
- Activations = activation function(net input); $a = \sigma(z)$
- Label output = threshold(activations of last layer); $\hat{y} = f(a)$

Special cases:

- In perceptron: activation function = threshold function
- In linear regression: activation = net input = output



Perceptron Output

$$\hat{y} = \begin{cases} 0, & z \leq \theta \\ 1, & z > \theta \end{cases}$$

More convenient to re-arrange:

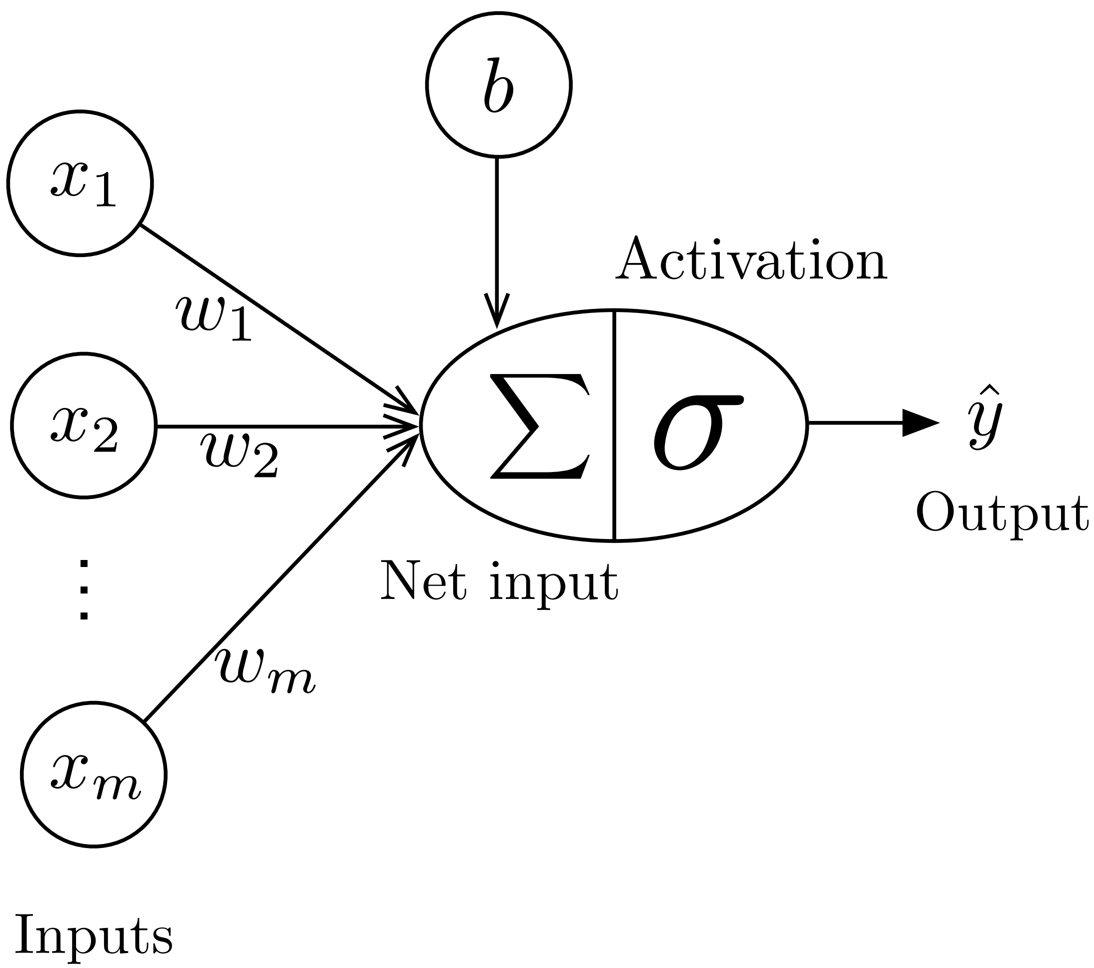
$$\hat{y} = \begin{cases} 0, & z - \theta \leq 0 \\ 1, & z - \theta > 0 \end{cases}$$

negative threshold

$\boxed{-\theta = \text{"bias"}}$

General Notation for Single-Layer Neural Networks

- Common notation (in most modern texts): define the bias unit separately
- However, often inconvenient for mathematical notation



"separate" bias unit

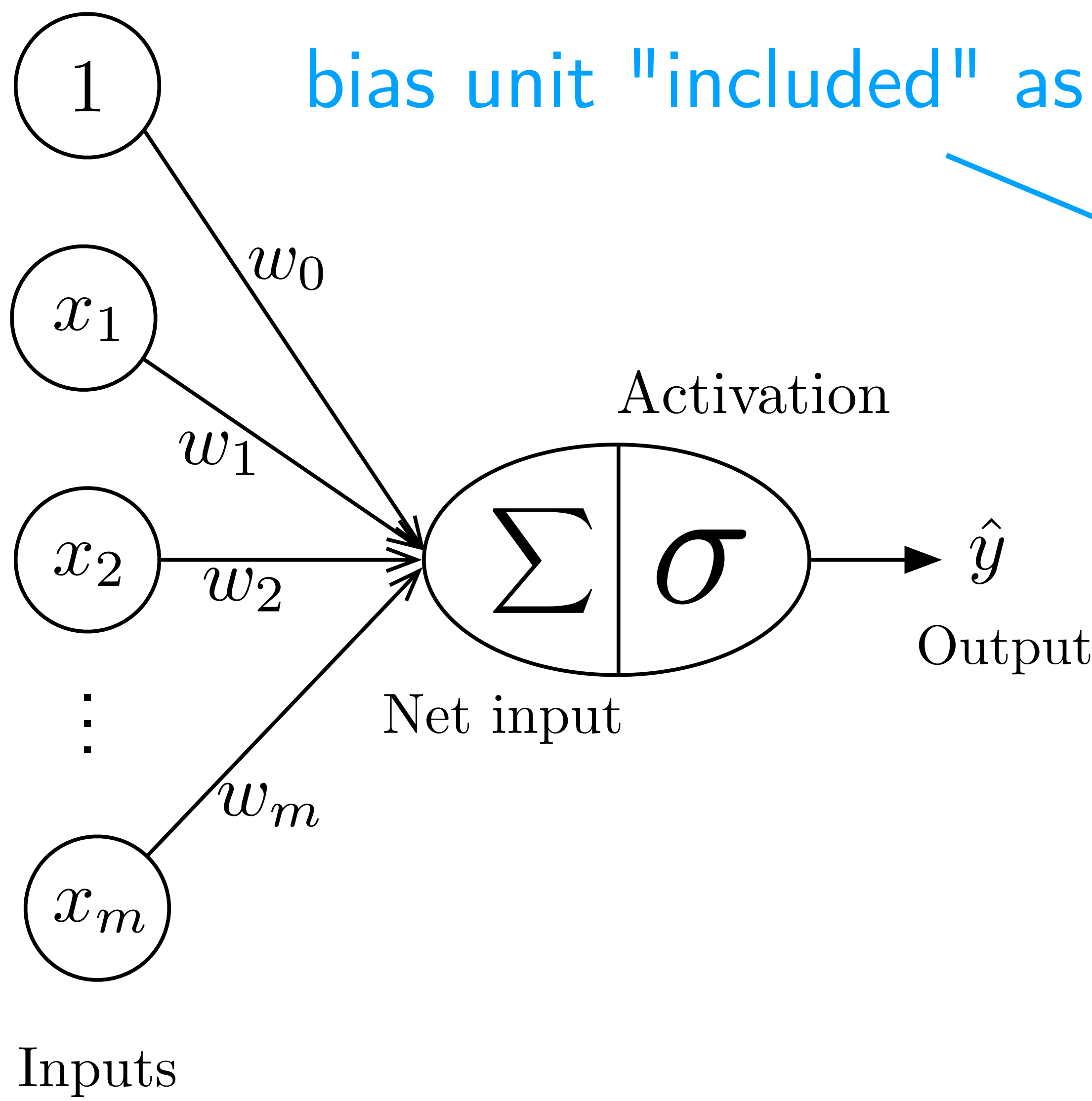
$$\sigma\left(\sum_{i=1}^{m} x_i w_i + b\right) = \sigma\left(\mathbf{x}^T \mathbf{w} + b\right) = \hat{y}$$

$$\sigma(z) = \begin{cases} 0, & z \leq 0 \\ 1, & z > 0 \end{cases}$$

$$b = -\theta$$

General Notation for Single-Layer Neural Networks

- Often more convenient notation: define bias unit as w_0 and prepend a 1 to each input vector as an additional "feature" value
- Modifying input vectors is more inconvenient/inefficient coding-wise, though



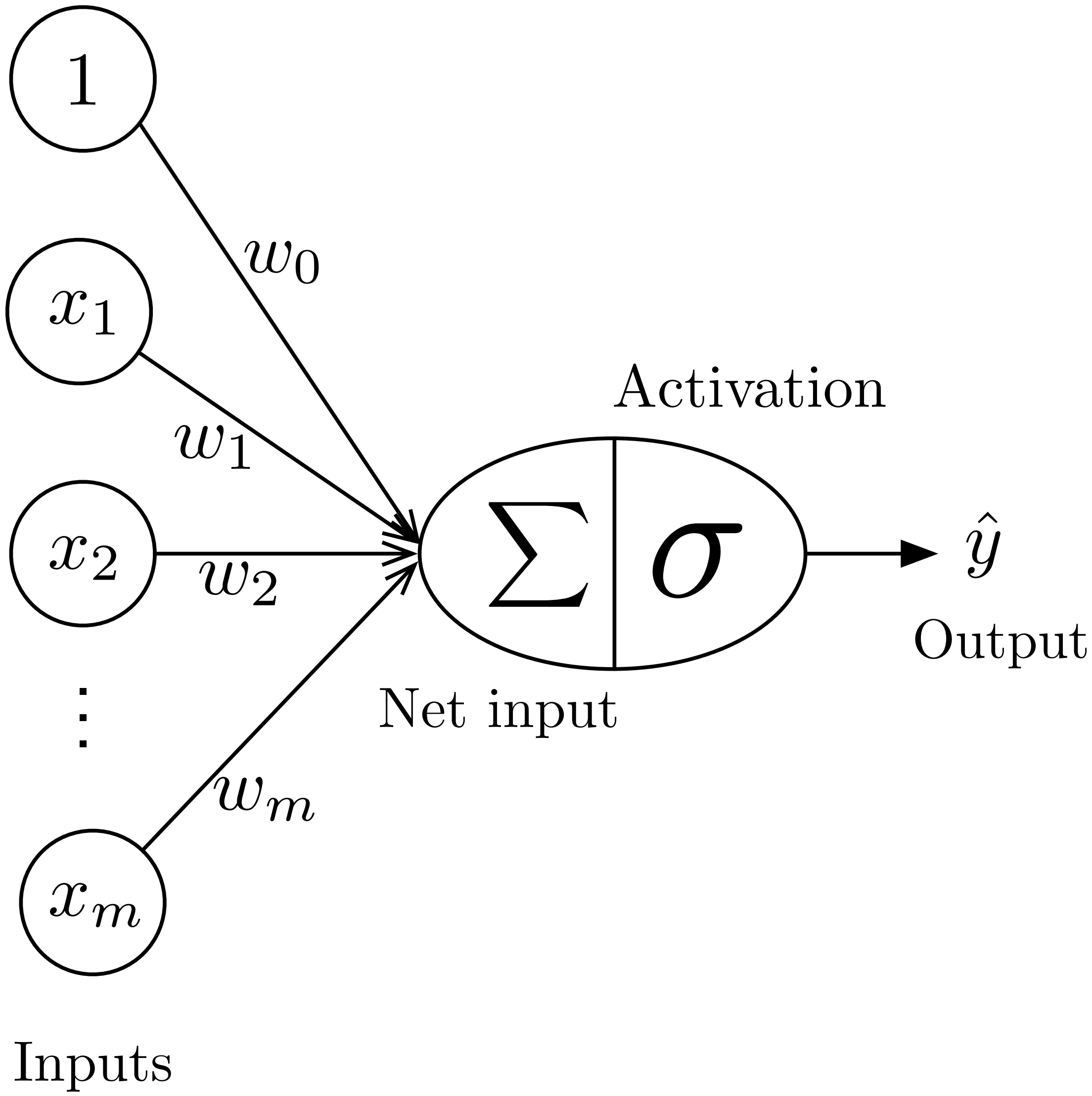
bias unit "included" as w_0

$$\sigma\left(\sum_{i=0}^{m} x_i w_i\right) = \sigma\left(\mathbf{x}^\top \mathbf{w}\right) = \hat{y}$$

$$\sigma(z) = \begin{cases} 0, & z \leq 0 \\ 1, & z > 0 \end{cases}$$

$$w_0 = -\theta$$

General Notation for Single-Layer Neural Networks



Vector dot product

$$\sigma\left(\sum_{i=0}^{m} x_i w_i\right) = \sigma\boxed{\left(\mathbf{x}^T\mathbf{w}\right)} = \hat{y}$$

$$\sigma(z) = \begin{cases} 0, & z - \theta \leq 0 \\ 1, & z - \theta > 0 \end{cases}$$

$$w_0 = -\theta$$

Interlude: "Vectorization" in Python

Question for you: What are we computing here?

In [1]:

```
x0, x1, x2 = 1., 2., 3.
bias, w1, w2 = 0.1, 0.3, 0.5

x = [x0, x1, x2]
w = [bias, w1, w2]
```

A simple for-loop:

In [2]:

```
z = 0.
for i in range(len(x)):
    z += x[i] * w[i]

print(z)
```

```
2.2
```

Interlude: "Vectorization" in Python

A simple for-loop:

In [2]:

```
z = 0.
for i in range(len(x)):
    z += x[i] * w[i]

print(z)
```

```
2.2
```

A little bit better, list comprehensions:

In [3]:

```
z = sum(x_i*w_i for x_i, w_i in zip(x, w))
print(z)
```

```
2.2
```

Interlude: "Vectorization" in Python

list comprehensions (still sequential):

In [3]:

```
z = sum(x_i*w_i for x_i, w_i in zip(x, w))
print(z)
```

```
2.2
```

A vectorized implementation using **NumPy**:

In [4]:

```
import numpy as np

x_vec, w_vec = np.array(x), np.array(w)

z = (x_vec.transpose()).dot(w_vec)
print(z)

z = x_vec.dot(w_vec)
print(z)
```

```
2.2
2.2
```

Interlude: "Vectorization" in Python

a)

```
def forloop(x, w):
    z = 0.
    for i in range(len(x)):
        z += x[i] * w[i]
    return z
```

b)

```
def listcomprehension(x, w):
    return sum(x_i*w_i for x_i, w_i in zip(x, w))
```

c)

```
def vectorized(x, w):
    return x_vec.dot(w_vec)
```

```
x, w = np.random.rand(100000), np.random.rand(100000)
```

Questions for you:

Which one is the fastest?

How much faster is the fastest one compared to the slowest one?

Interlude: "Vectorization" in Python

In [6]:
```
%timeit -r 100 -n 10 forloop(x, w)
```

```
38.9 ms ± 1.32 ms per loop (mean ± std. dev. of 100 r
uns, 10 loops each)
```

In [7]:
```
%timeit -r 100 -n 10 listcomprehension(x, w)
```

```
29.7 ms ± 842 µs per loop (mean ± std. dev. of 100 ru
ns, 10 loops each)
```

In [8]:
```
%timeit -r 100 -n 10 vectorized(x_vec, w_vec)
```

```
46.8 µs ± 8.07 µs per loop (mean ± std. dev. of 100 r
uns, 10 loops each)
```

Interlude: Connections and Parallel Computation

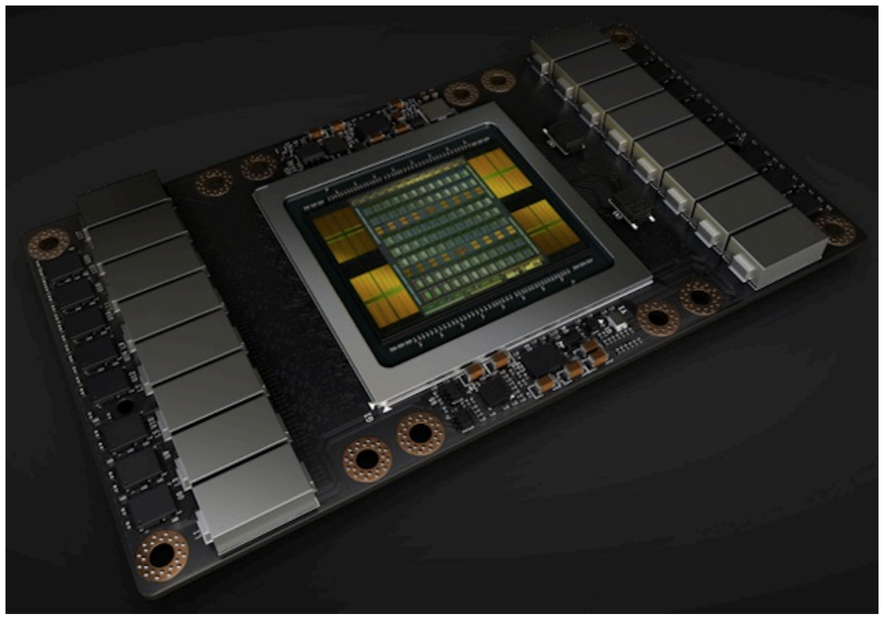


Image Source: https://fossbytes.com/wp-content/uploads/2017/05/nvidia-volta-v100-gpu.jpg

NVIDIA Volta with approx. 2.1×10^{10} transistors

approx. only 10 connections per transistor

Image Source: https://timedotcom.files.wordpress.com/2014/05/brain.jpg?w=1100&quality=85

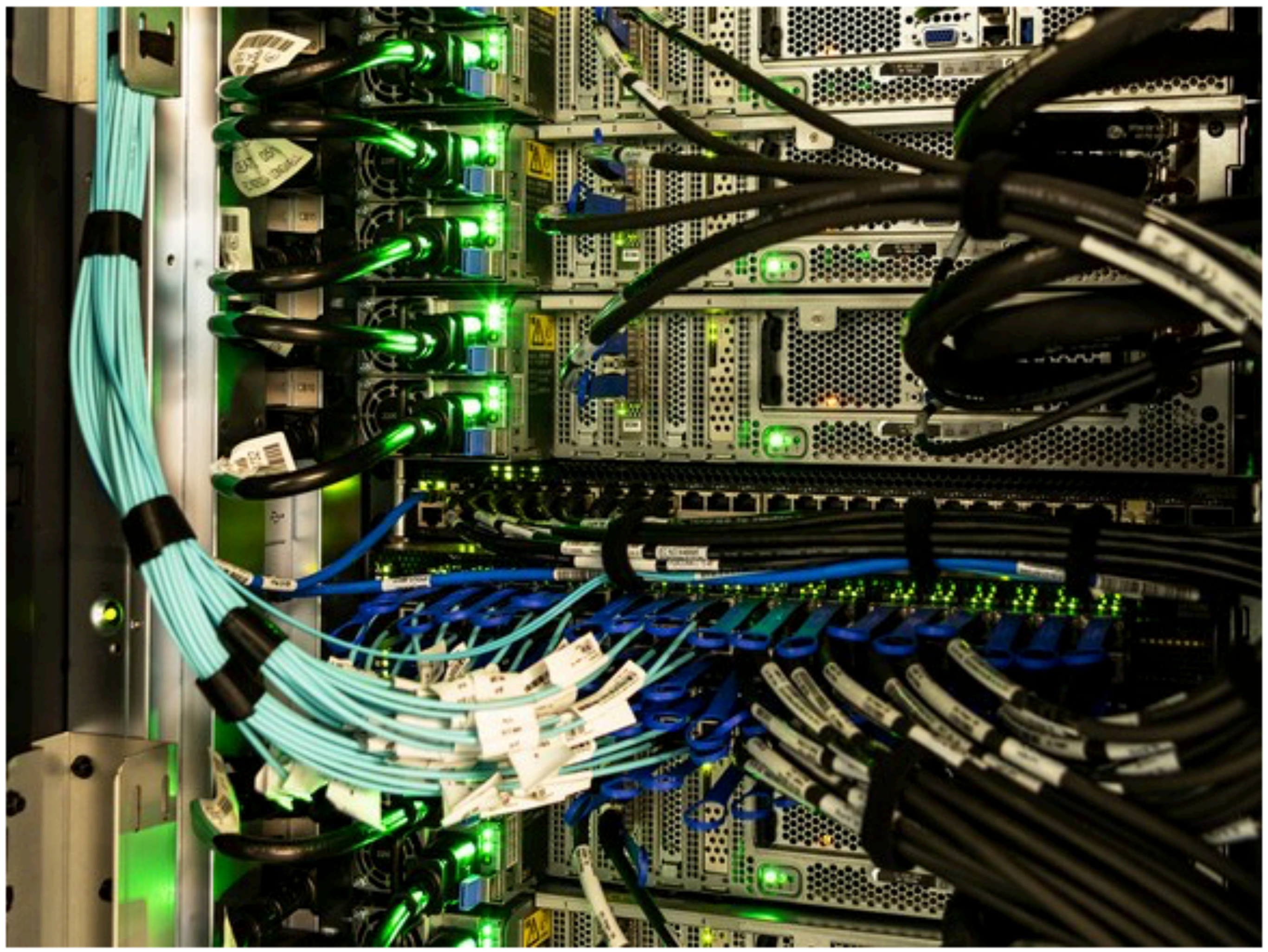
Brain with 1.6×10^{10} neurons

10^4 - 10^5 connections per neuron

approx. 10^{15} connections in total

TOM SIMONITE BUSINESS 01.31.19 05:58 PM

THE WORLD'S FASTEST SUPERCOMPUTER BREAKS AN AI RECORD



Oak Ridge National Lab's Summit supercomputer became the world's most powerful in 2018, reclaiming that title from China for the first time in five years.

CARLOS JONES/OAK RIDGE NATIONAL LAB

https://www.wired.com/story/worlds-fastest-supercomputer-breaks-ai-record/

> 27,000 GPUs
"billion billion" operations per second (exaflop)

"Deep learning has never been scaled to such levels of performance before," says Prabhat (research group leader at Berkeley National Lab)

Application: Weather patterns (three-hour forecasts)

WIRED BUSINESS CULTURE GEAR IDEAS SCIENCE SECURITY TRANSPORTATION SIGN IN SUBSCRIBE

WILL KNIGHT BUSINESS 01.21.2020 07:00 AM

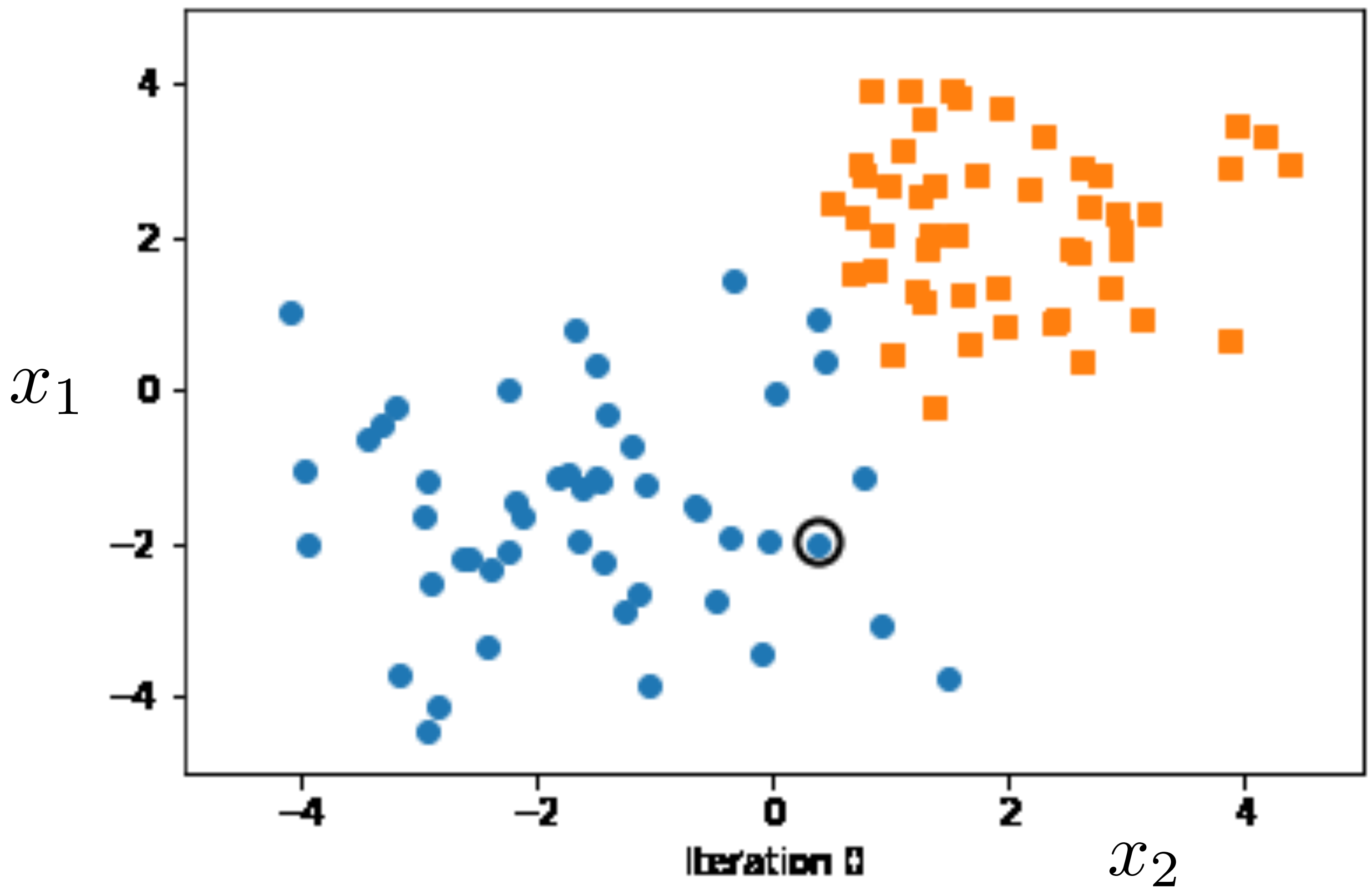
AI Can Do Great Things—if It Doesn't Burn the Planet

The computing power required for AI landmarks, such as recognizing images and defeating humans at Go, increased 300,000-fold from 2012 to 2018.

https://www.wired.com/story/ai-great-things-burn-planet/

Perceptron Learning Rule

Assume binary classification task, Perceptron finds decision boundary if classes are separable

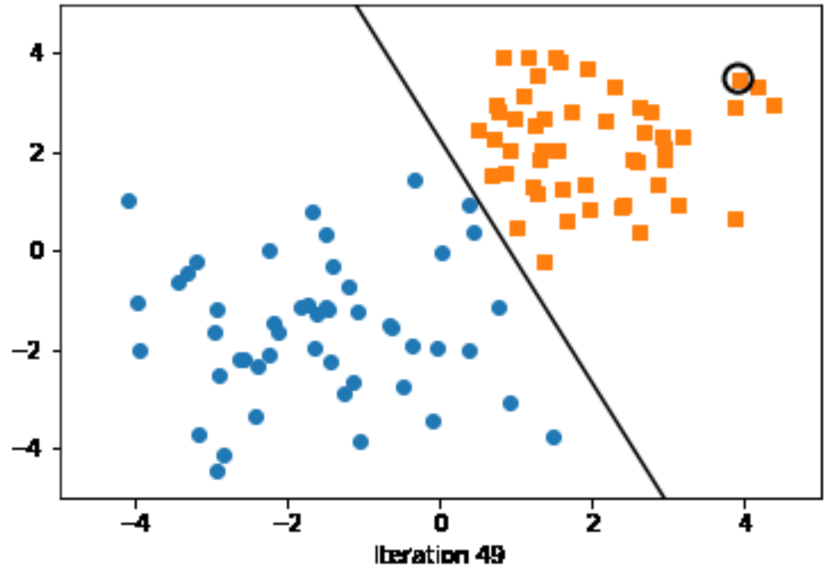


[animated GIF]

Code at https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L03-perceptron/code/perceptron-animation.ipynb if you want to play with it.

The Perceptron Learning Algorithm

- If correct: Do nothing if the prediction if output is equal to the target
- If incorrect, scenario **a)**:
  If output is 0 and target is 1, add input vector to weight vector
- If incorrect, scenario **b)**:
  If output is 1 and target is 0, subtract input vector from weight vector



Guaranteed to converge if a solution exists
(more about that later...)

The Perceptron Learning Algorithm

Let

$$\mathcal{D} = (\langle \mathbf{x}^{[1]}, y^{[1]} \rangle, \langle \mathbf{x}^{[2]}, y^{[2]} \rangle, ..., \langle \mathbf{x}^{[n]}, y^{[n]} \rangle) \in (\mathbb{R}^m \times \{0,1\})^n$$

1. Initialize $\mathbf{w} := 0^m$ (assume notation where weight incl. bias)
2. For every training epoch:
   A. For every $\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D}$:
      (a) $\hat{y}^{[i]} := \sigma\big(\mathbf{x}^{[i]\top}\mathbf{w}\big)$
      (b) $\mathrm{err} := (y^{[i]} - \hat{y}^{[i]})$
      (c) $\mathbf{w} := \mathbf{w} + err \times \mathbf{x}^{[i]}$

Perceptron Coding Example

https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L03-perceptron/code/perceptron-numpy.ipynb

https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L03-perceptron/code/perceptron-pytorch.ipynb

Optional: Perceptron Convergence Theorem

Perceptron Convergence Theorem

Let

$$\mathcal{D} = (\langle \mathbf{x}^{[1]}, y^{[1]} \rangle, \langle \mathbf{x}^{[2]}, y^{[2]} \rangle, ..., \langle \mathbf{x}^{[n]}, y^{[n]} \rangle) \in (\mathbb{R}^m \times \{0, 1\})^n$$

$$\forall y^{[i]} \in \mathcal{D}_1 : y^{[i]} = 1$$
$$\forall y^{[i]} \in \mathcal{D}_2 : y^{[i]} = 0$$

and $\mathcal{D}_1 \cup \mathcal{D}_2 = \mathcal{D}$

Assume the input vectors come from two linearly separable classes such that a feasible weight vector $\mathbf{w}*$ exists.

The perceptron learning algorithm is guaranteed to converge to a weight vector in the feasible region in a finite number of iterations such that

$$\forall \mathbf{x}^{[i]} \in \mathcal{D}_1 : \mathbf{w}^\top \mathbf{x}^{[i]} > 0$$

$$\forall \mathbf{x}^{[i]} \in \mathcal{D}_2 : \mathbf{w}^\top \mathbf{x}^{[i]} \leq 0$$

Perceptron Convergence Theorem -- Proof

Let us slightly rewrite the update rule (upon misclassification) for convenience when we construct the proof:

$$\mathbf{w}^{[i+1]} = \mathbf{w}^{[i]} + \mathbf{x}^{[i]} \text{ if } (\mathbf{w}^{[i]})^\top \mathbf{x}^{[i]} \leq 0 \, , \mathbf{x}^{[i]} \in \mathcal{D}_1$$

$$\mathbf{w}^{[i+1]} = \mathbf{w}^{[i]} - \mathbf{x}^{[i]} \text{ if } (\mathbf{w}^{[i]})^\top \mathbf{x}^{[i]} > 0 \, , \mathbf{x}^{[i]} \in \mathcal{D}_2$$

Here $[i+1]$ refers to the weight vector of the next training example (that is, the weight after updating)

Perceptron Convergence Theorem -- Proof

From the previous slide:

$$\mathbf{w}^{[i+1]} = \mathbf{w}^{[i]} + \mathbf{x}^{[i]} \;\text{ if }\; (\mathbf{w}^{[i]})^\top \mathbf{x}^{[i]} \leq 0 \;, \mathbf{x}^{[i]} \in \mathcal{D}_1$$

We can rewrite this as follows:

$$\mathbf{w}^{[i+1]} = \mathbf{w}^{[0]} + \mathbf{x}^{[1]} + ... + \mathbf{x}^{[i]}$$

Also, we can drop this term if we initialize the weight vector as 0^m

$$\mathbf{w}^{[i+1]} = \mathbf{x}^{[1]} + ... + \mathbf{x}^{[i]}$$

Perceptron Convergence Theorem -- Proof

From the previous slide, the update rule:

$$\mathbf{w}^{[i+1]} = \mathbf{x}^{[1]} + ... + \mathbf{x}^{[i]}$$

Let's multiply both sides by $\mathbf{w}^*$:

$$(\mathbf{w}^*)^T \mathbf{w}^{[i+1]} = (\mathbf{w}^*)^T \mathbf{x}^{[1]} + ... + (\mathbf{w}^*)^T \mathbf{x}^{[i]}$$

All these terms are > 0, because remember that we have

$$\mathbf{w}^{[i+1]} = \mathbf{w}^{[i]} + \mathbf{x}^{[i]} \text{ if } (\mathbf{w}^{[i]})^T \mathbf{x}^{[i]} \leq 0 \text{ , } \mathbf{x}^{[i]} \in \mathcal{D}_1$$

so the updates are all to make the net inputs more positive

Now, let $\alpha = \min_{x^{[j]}} (\mathbf{w}^*)^T \mathbf{x}^{[j]}, \ j = 1, ..., i$

then $(\mathbf{w}^*)^T \mathbf{w}^{[i+1]} \geq \alpha i$

Perceptron Convergence Theorem -- Proof

From the previous slide, we had the inequality:

$$\underbrace{(\mathbf{w}^*)^T \mathbf{w}}\,^{[i+1]} \geq \alpha i$$

Using the Cauchy-Schwarz inequality, we can then say

$$||\mathbf{w}^*||^2 \cdot ||\mathbf{w}^{[i+1]}||^2 \geq \left((\mathbf{w}^*)^T \mathbf{w}^{[i+1]}\right)^2$$

as well as

$$||\mathbf{w}^*||^2 \cdot ||\mathbf{w}^{[i+1]}||^2 \geq (\alpha i)^2$$

So, we can finally define the lower bound of the size of the weights

$$||\mathbf{w}^{[i+1]}||^2 \geq \frac{\alpha^2 i^2}{||\mathbf{w}^*||^2}$$

Perceptron Convergence Theorem -- Proof

Now that we defined the lower bound of the size of the weights, let us get the upper bound.

For that, let's go back to the update rule

$$\mathbf{w}^{[i+1]} = \mathbf{w}^{[i]} + \mathbf{x}^{[i]} \;\; \text{if} \;\; (\mathbf{w}^{[i]})^T \mathbf{x}^{[i]} \leq 0 \,, \mathbf{x}^{[i]} \in \mathcal{D}_1$$

and apply the squared L2 norm on both sides

$$\begin{aligned} ||\mathbf{w}^{[i+1]}||^2 &= ||\mathbf{w}^{[i]} + \mathbf{x}^{[i]}||^2 \\ &= ||\mathbf{w}^{[i]}||^2 + 2(\mathbf{x}^{[i]})^T \mathbf{w}^{[i]} + ||\mathbf{x}^{[i]}||^2 \end{aligned}$$

Perceptron Convergence Theorem -- Proof

Now that we defined the lower bound of the size of the weights, let us get the upper bound.

For that, let's go back to the update rule

$$\mathbf{w}^{[i+1]} = \mathbf{w}^{[i]} + \mathbf{x}^{[i]} \ \text{ if } \ (\mathbf{w}^{[i]})^T \mathbf{x}^{[i]} \leq 0 \ , \mathbf{x}^{[i]} \in \mathcal{D}_1$$

and apply the squared L2 norm on both sides

$$\begin{aligned} ||\mathbf{w}^{[i+1]}||^2 &= ||\mathbf{w}^{[i]} + \mathbf{x}^{[i]}||^2 \\ &= ||\mathbf{w}^{[i]}||^2 + 2(\mathbf{x}^{[i]})^T \mathbf{w}^{[i]} + ||\mathbf{x}^{[i]}||^2 \end{aligned}$$

Leads to

$$||\mathbf{w}^{[i+1]}||^2 \leq ||\mathbf{w}^{[i]}||^2 + ||\mathbf{x}^{[i]}||^2$$

Perceptron Convergence Theorem -- Proof

Now that we defined the lower bound of the size of the weights, let us get the upper bound.

For that, let's go back to the update rule

$$\mathbf{w}^{[i+1]} = \mathbf{w}^{[i]} + \mathbf{x}^{[i]} \ \text{ if } \ \underbrace{(\mathbf{w}^{[i]})^T \mathbf{x}^{[i]} \leq 0}_{\text{implies}} \ , \mathbf{x}^{[i]} \in \mathcal{D}_1$$

and apply the squared L2 norm on both sides

$$||\mathbf{w}^{[i+1]}||^2 = ||\mathbf{w}^{[i]} + \mathbf{x}^{[i]}||^2$$

$$= ||\mathbf{w}^{[i]}||^2 + \underbrace{2(\mathbf{x}^{[i]})^T \mathbf{w}^{[i]}}_{\leq 0} + ||\mathbf{x}^{[i]}||^2$$

Thus

$$||\mathbf{w}^{[i+1]}||^2 \leq ||\mathbf{w}^{[i]}||^2 + ||\mathbf{x}^{[i]}||^2$$

Perceptron Convergence Theorem -- Proof

Now, we simply expand:

$$||\mathbf{w}^{[i+1]}||^2 \leq \boxed{||\mathbf{w}^{[i]}||^2} + ||\mathbf{x}^{[i]}||^2$$

$$||\mathbf{w}^{[i+1]}||^2 \leq \boxed{||\mathbf{w}^{[i-1]}||^2 + ||\mathbf{x}^{[i-1]}||^2} + ||\mathbf{x}^{[i]}||^2$$

$$||\mathbf{w}^{[i+1]}||^2 \leq \boxed{||\mathbf{w}^{[i-2]}||^2 + ||\mathbf{x}^{[i-2]}||^2 + ||\mathbf{x}^{[i-1]}||^2} + ||\mathbf{x}^{[i]}||^2$$

$$\dots$$

$$||\mathbf{w}^{[i+1]}||^2 \leq ||\mathbf{w}^{[1]}||^2 + \sum_{j=1}^{i} ||\mathbf{x}^{[j]}||^2$$

$$||\mathbf{w}^{[i+1]}||^2 \leq \sum_{j=1}^{i} ||\mathbf{x}^{[j]}||^2$$

Perceptron Convergence Theorem -- Proof

From $||\mathbf{w}^{[i+1]}||^2 \leq \sum_{j=1}^{i} ||\mathbf{x}^{[j]}||^2$ we can finally get the upper bound.

Let $\beta = \max ||\mathbf{x}^{[j]}||^2$

then $||\mathbf{w}^{[i+1]}||^2 \leq \beta i$

Perceptron Convergence Theorem -- Proof

lower bound

$$||\mathbf{w}^{[i+1]}||^2 \geq \frac{\alpha^2 i^2}{||\mathbf{w}^*||^2}$$

upper bound

$$||\mathbf{w}^{[i+1]}||^2 \leq \beta i$$

combined

$$\beta_i \geq ||\mathbf{w}^{[i+1]}||^2 \geq \frac{\alpha^2 i^2}{||\mathbf{w}^*||^2}$$

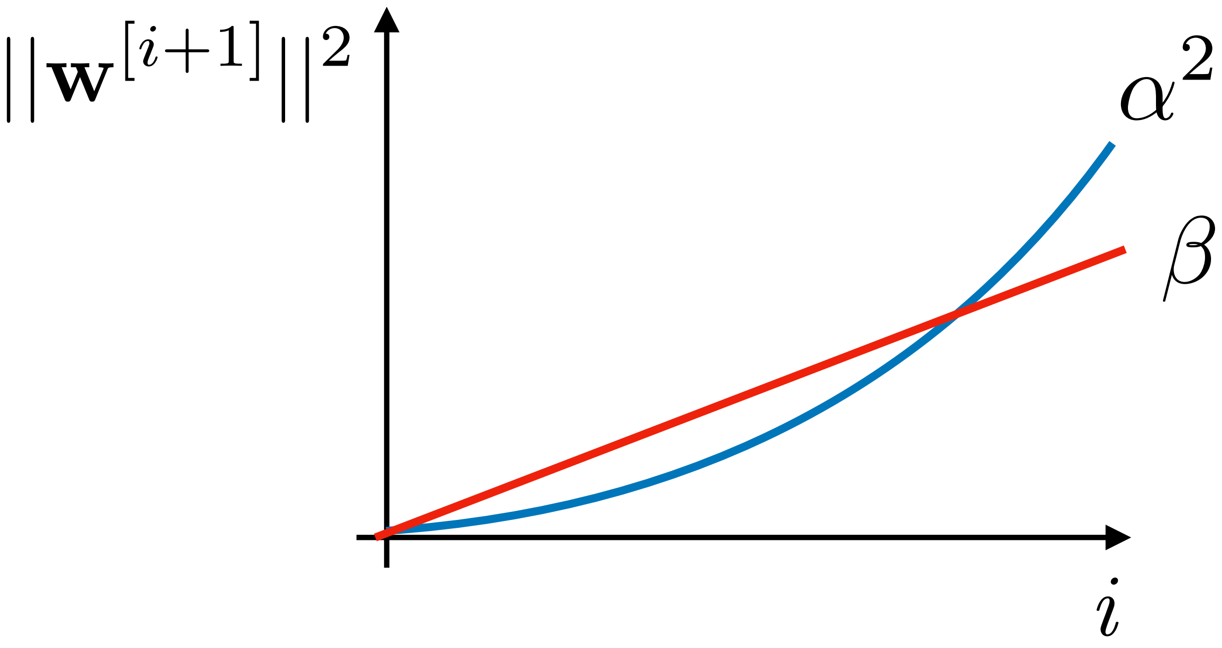
$$i \leq \frac{\beta ||\mathbf{w}^*||^2}{\alpha^2}$$

Since the number of iterations *i* has an upper bound,
we can conclude that the weights only change a finite number of times and will converge if the classes are linearly separable.

$\square$

Perceptron Convergence Theorem -- Proof

$$\beta i \geq ||\mathbf{w}^{[i+1]}||^2 \geq \alpha^2 i^2$$



Perceptron Convergence Theorem -- Proof

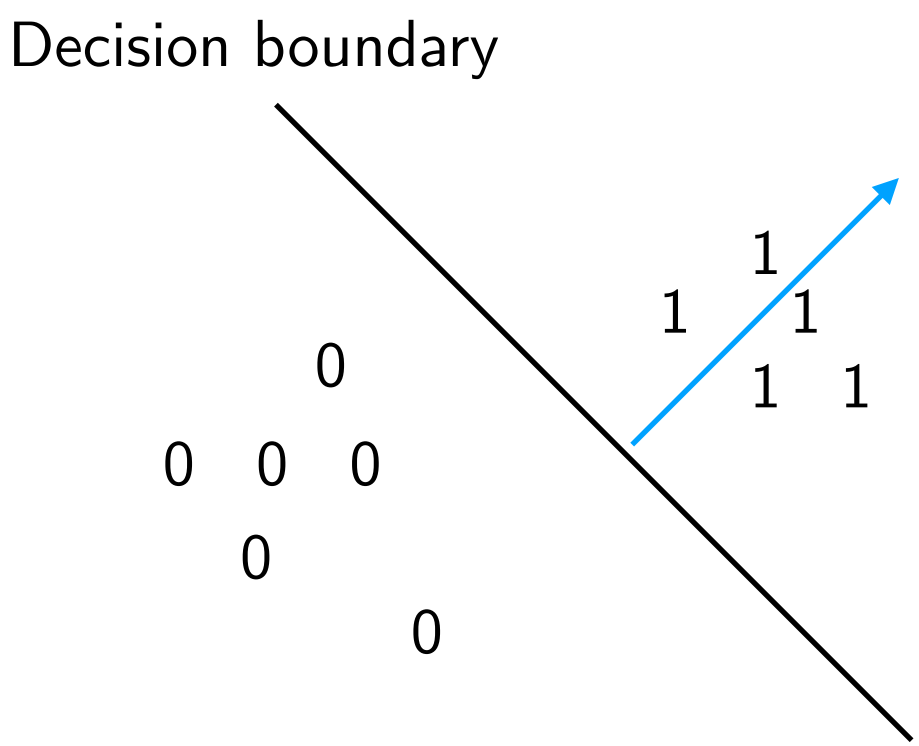
In the convergence theorem, we can assume that $||\mathbf{w}^*|| = 1$
(so you may remove it from all equations)

$$\beta_i \geq ||\mathbf{w}^{[i+1]}||^2 \geq \frac{\alpha^2 i^2}{||\mathbf{w}^*||^2} \qquad \Leftrightarrow \qquad \beta_i \geq ||\mathbf{w}^{[i+1]}||^2 \geq \alpha^2 i^2$$

$$i \leq \frac{\beta ||\mathbf{w}^*||^2}{\alpha^2} \qquad \Leftrightarrow \qquad i \leq \frac{\beta}{\alpha^2}$$

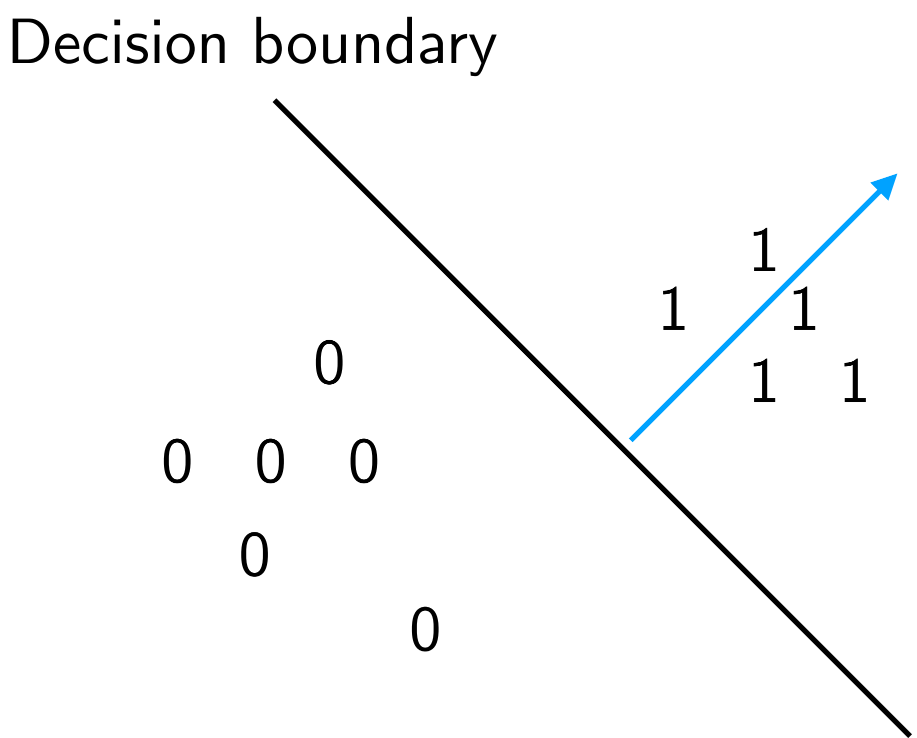
Geometric Intuition Behind the Perceptron

Geometric Intuition



Weight vector is perpendicular to the boundary. Why?

Geometric Intuition



Weight vector is perpendicular to the boundary. Why?

Remember,

$$\hat{y} = \begin{cases} 0, & \mathbf{w}^T\mathbf{x} \leq 0 \\ 1, & \mathbf{w}^T\mathbf{x} > 0 \end{cases}$$

$$\mathbf{w}^T\mathbf{x} = ||\mathbf{w}|| \cdot ||\mathbf{x}|| \cdot \underbrace{\cos(\theta)}$$

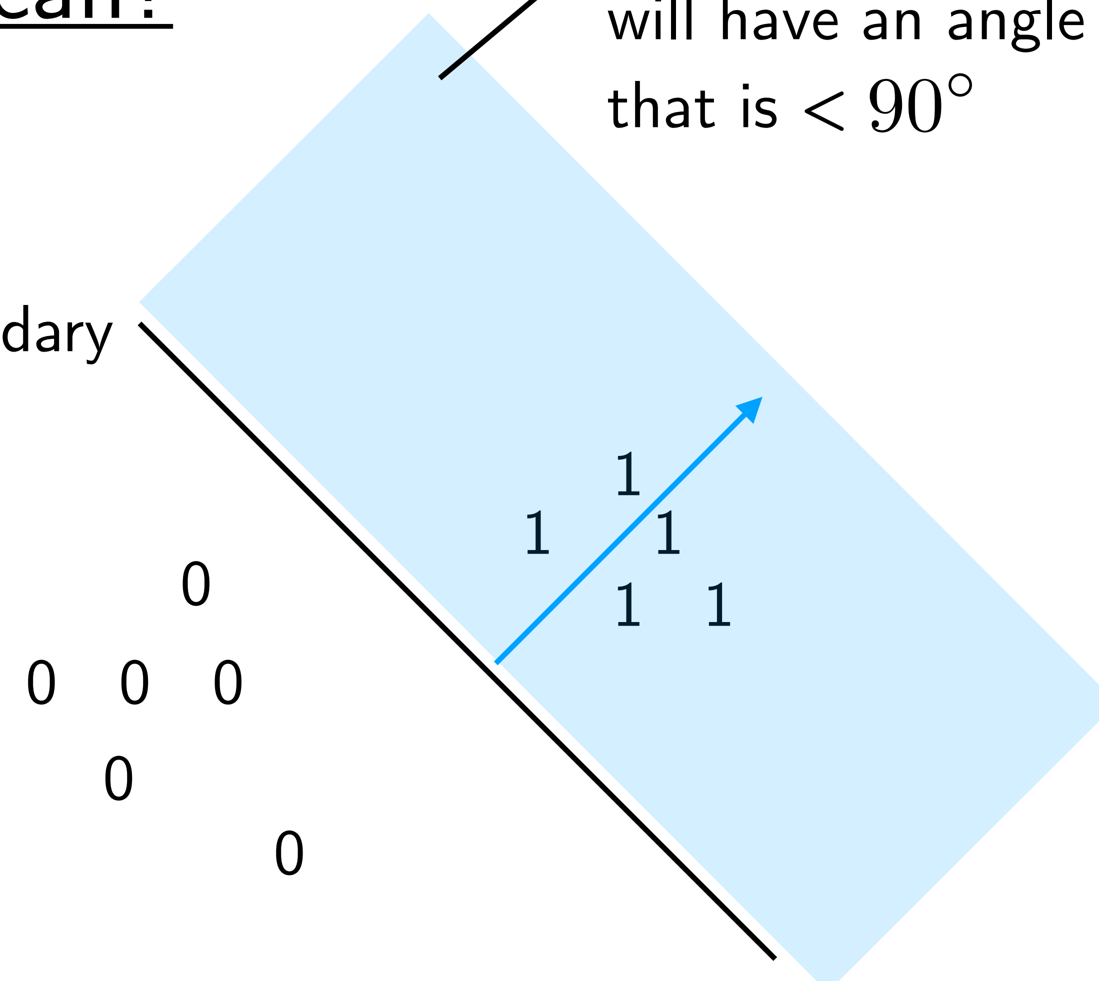
So this needs to be 0 at the boundary, and it is zero at 90°

Geometric Intuition

<u>What else does this mean?</u>

Every input vector on this side
will have an angle with the weight vector
that is $< 90^\circ$

Decision boundary

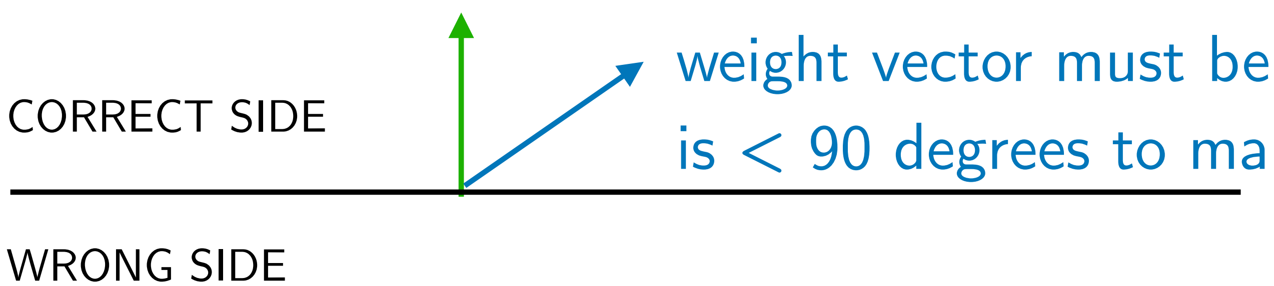


Assume origin (0, 0)
and no bias

So, we could scale the weights and/or inputs by an arbitrary factor and still get the same classification results
(but large inputs will take much longer to converge if you check the bounds we defined previously ...)

Geometric Intuition

input vector for an example with label 1



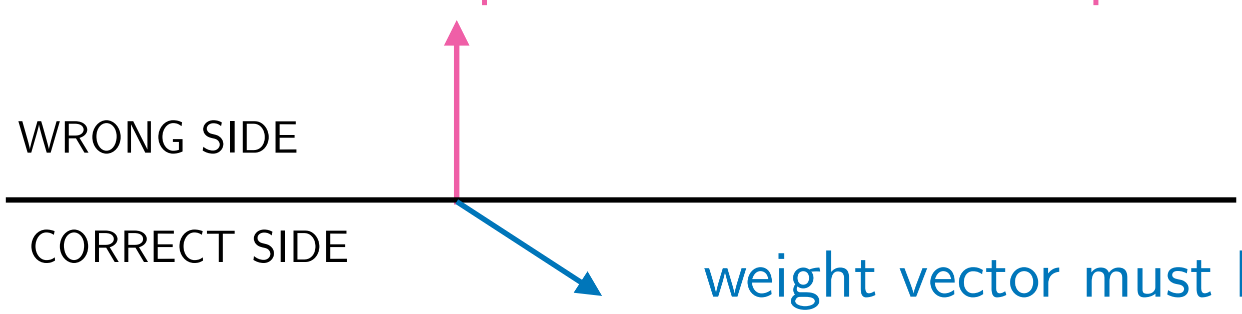
weight vector must be somewhere such that the angle is $<$ 90 degrees to make a correct prediction

The dot product will then be positive, i.e., $>$ 0, since

$$\mathbf{w}^T\mathbf{x} = ||\mathbf{w}|| \cdot ||\mathbf{x}|| \cdot \cos(\theta)$$

Geometric Intuition

input vector for an example with label 0



weight vector must be somewhere such that the angle is $\geq$ 90 degrees to make a correct prediction

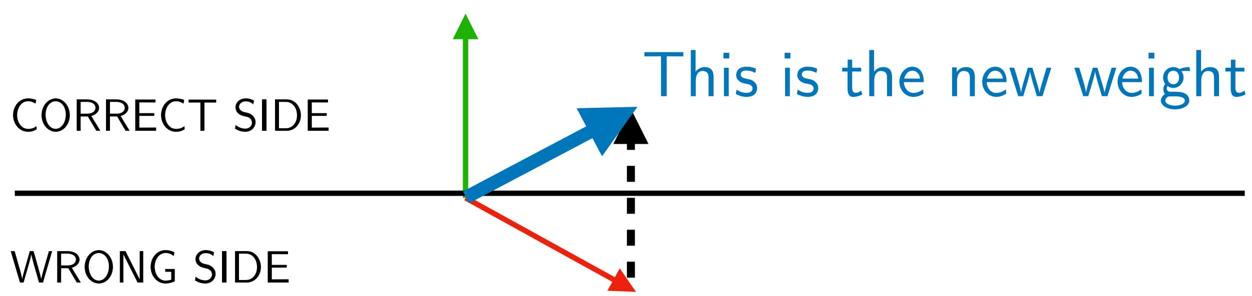
The dot product will then $\leq$ 0, since

$$\mathbf{w}^T\mathbf{x} = ||\mathbf{w}|| \cdot ||\mathbf{x}|| \cdot \cos(\theta)$$

Geometric Intuition

input vector for an example with label 1

This is the new weight vector



For this weight vector, we make a wrong prediction; hence, we update

Perceptron Conclusions

The (classic) Perceptron has many problems
(as discussed in the previous lecture)

- Linear classifier, no non-linear boundaries possible
- Binary classifier, cannot solve XOR problems, for example
- Does not converge if classes are not linearly separable
- Many "optimal" solutions in terms of 0/1 loss on the training data, most will not be optimal in terms of generalization performance

https://qph.fs.quoracdn.net/main-qimg-305eb8136c4a20f348bb7ab465bc2e10

http://theconversation.com/want-to-beat-climate-change-protect-our-natural-forests-121491

Perceptron Fun Fact

[...] Where a perceptron had been trained to distinguish between - this was for military purposes - it was looking at a scene of a forest in which there were camouflaged tanks in one picture and no camouflaged tanks in the other. And the perceptron - after a little training - made a 100% correct distinction between these two different sets of photographs. Then they were embarrassed a few hours later to discover that the two rolls of film had been developed differently. And so these pictures were just a little darker than all of these pictures and the perceptron was just measuring the total amount of light in the scene. But it was very clever of the perceptron to find some way of making the distinction.

-- Marvin Minsky, Famous AI researcher, Author of the famous "Perceptrons" book

Source: https://www.webofstories.com/play/marvin.minsky/122

An Analogy to Future Lectures ...

We can say the perceptron optimizes a loss function analogous to the squared error in least-squares regression, except that we have targets and outputs:

$$\mathcal{L}(\mathbf{w}, b) = \sum_i \frac{1}{2}(\hat{y}^{[i]} - y^{[i]})^2$$

where $$\hat{y} = \sigma(\mathbf{w}^T\mathbf{x} + b) = \begin{cases} 0, & \mathbf{w}^T\mathbf{x} + b \leq 0 \\ 1, & \mathbf{w}^T\mathbf{x} + b > 0 \end{cases}$$

An Analogy to Future Lectures ...

We can say the perceptron optimizes a loss function analogous to the squared error in least-squares regression, except that we have targets and outputs:

$$\mathcal{L}(\mathbf{w}, b) = \sum_i \frac{1}{2}(\hat{y}^{[i]} - y^{[i]})^2$$

where $\hat{y} = \sigma(\mathbf{w}^T\mathbf{x} + b) = \begin{cases} 0, & \mathbf{w}^T\mathbf{x} + b \leq 0 \\ 1, & \mathbf{w}^T\mathbf{x} + b > 0 \end{cases}$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_j} &= \frac{\partial}{\partial w_j} \sum_i (\hat{y}^{[i]} - y^{[i]})^2 \\
&= \frac{\partial}{\partial w_j} \sum_i (\sigma(\mathbf{w}^T\mathbf{x}^{[i]}) - y^{[i]})^2 \\
&= \sum_i 2(\sigma(\mathbf{w}^T\mathbf{x}^{[i]}) - y^{[i]}) \frac{\partial}{\partial w_j} (\sigma(\mathbf{w}^T\mathbf{x}^{[i]}) - y^{[i]}) \\
&= \sum_i 2(\sigma(\mathbf{w}^T\mathbf{x}^{[i]}) - y^{[i]}) \sigma'(\mathbf{w}^T\mathbf{x}^{[i]}) \frac{\partial}{\partial w_j} \mathbf{w}^T\mathbf{x}^{[i]} \\
&= \sum_i 2(\sigma(\mathbf{w}^T\mathbf{x}^{[i]}) - y^{[i]}) \sigma'(\mathbf{w}^T\mathbf{x}^{[i]}) x_j^{[i]}
\end{aligned}$$

An Analogy to Future Lectures ...

We can say the perceptron optimizes a loss function analogous to the squared error in least-squares regression, except that we have targets and outputs:

$$\mathcal{L}(\mathbf{w}, b) = \sum_i \frac{1}{2}(\hat{y}^{[i]} - y^{[i]})^2$$

where $\hat{y} = \sigma(\mathbf{w}^T\mathbf{x} + b) = \begin{cases} 0, & \mathbf{w}^T\mathbf{x} + b \leq 0 \\ 1, & \mathbf{w}^T\mathbf{x} + b > 0 \end{cases}$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_j} &= \frac{\partial}{\partial w_j} \sum_i (\hat{y}^{[i]} - y^{[i]})^2 \\
&= \frac{\partial}{\partial w_j} \sum_i (\sigma(\mathbf{w}^T\mathbf{x}^{[i]}) - y^{[i]})^2 \\
&= \sum_i 2(\sigma(\mathbf{w}^T\mathbf{x}^{[i]}) - y^{[i]}) \frac{\partial}{\partial w_j} (\sigma(\mathbf{w}^T\mathbf{x}^{[i]}) - y^{[i]}) \\
&= \sum_i 2(\sigma(\mathbf{w}^T\mathbf{x}^{[i]}) - y^{[i]}) \sigma'(\mathbf{w}^T\mathbf{x}^{[i]}) \frac{\partial}{\partial w_j} \mathbf{w}^T\mathbf{x}^{[i]} \\
&= \sum_i 2(\sigma(\mathbf{w}^T\mathbf{x}^{[i]}) - y^{[i]}) \boxed{\sigma'(\mathbf{w}^T\mathbf{x}^{[i]})} x_j^{[i]}
\end{aligned}$$

not differentiable!

An Analogy to Future Lectures ...

We can say the perceptron optimizes a loss function analogous to the squared error in least-squares regression, except that we have targets and outputs:

$$\mathcal{L}(\mathbf{w}, b) = \sum_i \frac{1}{2}(\hat{y}^{[i]} - y^{[i]})^2$$

where $\hat{y} = \sigma(\mathbf{w}^T\mathbf{x} + b) = \begin{cases} 0, & \mathbf{w}^T\mathbf{x} + b \leq 0 \\ 1, & \mathbf{w}^T\mathbf{x} + b > 0 \end{cases}$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_j} &= \frac{\partial}{\partial w_j} \sum_i (\hat{y}^{[i]} - y^{[i]})^2 \\
&= \frac{\partial}{\partial w_j} \sum_i (\sigma(\mathbf{w}^T\mathbf{x}^{[i]}) - y^{[i]})^2 \\
&= \sum_i 2(\sigma(\mathbf{w}^T\mathbf{x}^{[i]}) - y^{[i]}) \frac{\partial}{\partial w_j} (\sigma(\mathbf{w}^T\mathbf{x}^{[i]}) - y^{[i]}) \\
&= \sum_i 2(\sigma(\mathbf{w}^T\mathbf{x}^{[i]}) - y^{[i]}) \sigma'(\mathbf{w}^T\mathbf{x}^{[i]}) \frac{\partial}{\partial w_j} \mathbf{w}^T\mathbf{x}^{[i]} \\
&= \sum_i 2(\sigma(\mathbf{w}^T\mathbf{x}^{[i]}) - y^{[i]}) \boxed{\sigma'(\mathbf{w}^T\mathbf{x}^{[i]})} x_j^{[i]}
\end{aligned}$$

not differentiable!

However, perceptron does something very similar to stochastic gradient descent:

(not a real derivative)

$$\frac{\partial \mathcal{L}}{\partial w_j} = (y^{[i]} - \hat{y}^{[i]}) x_j$$

$$w_j := w_j + \frac{\partial \mathcal{L}}{\partial w_j}$$

On Deep Learning vs How the Brain Works

MARTIN FORD: You gave an interview toward the end of 2017 where you said that you were suspicious of the backpropagation algorithm and that it needed to be thrown out and we needed to start from scratch.[1] That created a lot of disturbance, so I wanted to ask what you meant by that?

GEOFFREY HINTON: The problem was that the context of the conversation wasn't properly reported. I was talking about trying to understand the brain, and I was raising the issue that backpropagation may not be the right way to understand the brain. We don't know for sure, but there are some reasons now for believing that the brain might not use backpropagation. I said that if the brain doesn't use backpropagation, then whatever the brain is using would be an interesting candidate for artificial systems. I didn't at all mean that we should throw out backpropagation. Backpropagation is the mainstay of all the deep learning that works, and I don't think we should get rid of it.

1 See: https://www.axios.com/artificial-intelligence-pioneer-says-we-need-to-start-over-1513305524-f619efbd-9db0-4947-a9b2-7a4c310a28fe.html

82

(Excerpt from "Architects of Intelligence")

1/5 -- Brains and neuron models
2/5 -- The perceptron learning rule
3/5 -- Optional: The perceptron convergence theorem
4/5 -- Geometric intuition
5/5 -- HW1

Your First Homework!

https://finmaxbo.com/uploads/posts/2019-07/1562831275_excitement.jpg

Homework Assignment 1

Your task:

Based on the NumPy example,
implement Perceptron code into pure Python to become more familiar of how PyTorch/NumPy (and the Perceptron) works
(=> no NumPy, no PyTorch, etc.)

1) Download code from:

https://github.com/rasbt/stat453-deep-learning-ss20/tree/master/hw01

2) Submit solution via Canvas for grading (submission deadline will be announced)