The Perceptron Learning Rule (a.k.a. The Perceptron Trick) Part 2

LIN 313 Language and Computers
UT Austin Fall 2025
Instructor: Gabriella Chronis

Overview 10/13

- Voting on classifiers
- Talk about the midterm
- The Perceptron
    - learning optimal weights with the Perceptron trick (aka 'the backward pass')

Pilot Annotation Results

results:

https://docs.google.com/spreadsheets/d/1c3AnPdg9rI0Q5ZDEIjUy6t0FxV4g1_L8mUPW-irkZ20/

Pilot Annotation Results

results:

https://docs.google.com/spreadsheets/d/1c3AnPdg9rI0Q5ZDEIjUy6t0FxV4g1_L8mUPW-irkZ20/

Complete the strawpoll.

Pick THREE schemes

Vote:

https://strawpoll.com/e7ZJabzWdg3

Potential Midterm Topics

Syntax

- structural ambiguity
- lexical ambiguity
- crash blossoms
- garden path sentences
- constituency

Pragmatics / social meaning

- Jakobson's elements of the speech situation / functions of language
- speech acts (informally)
- conversational maxims
  - flouting the maxims
- register/dialect/genre/persona

Machine Learning Algorithms

- Naive Bayes
- N-gram models
- The Perceptron

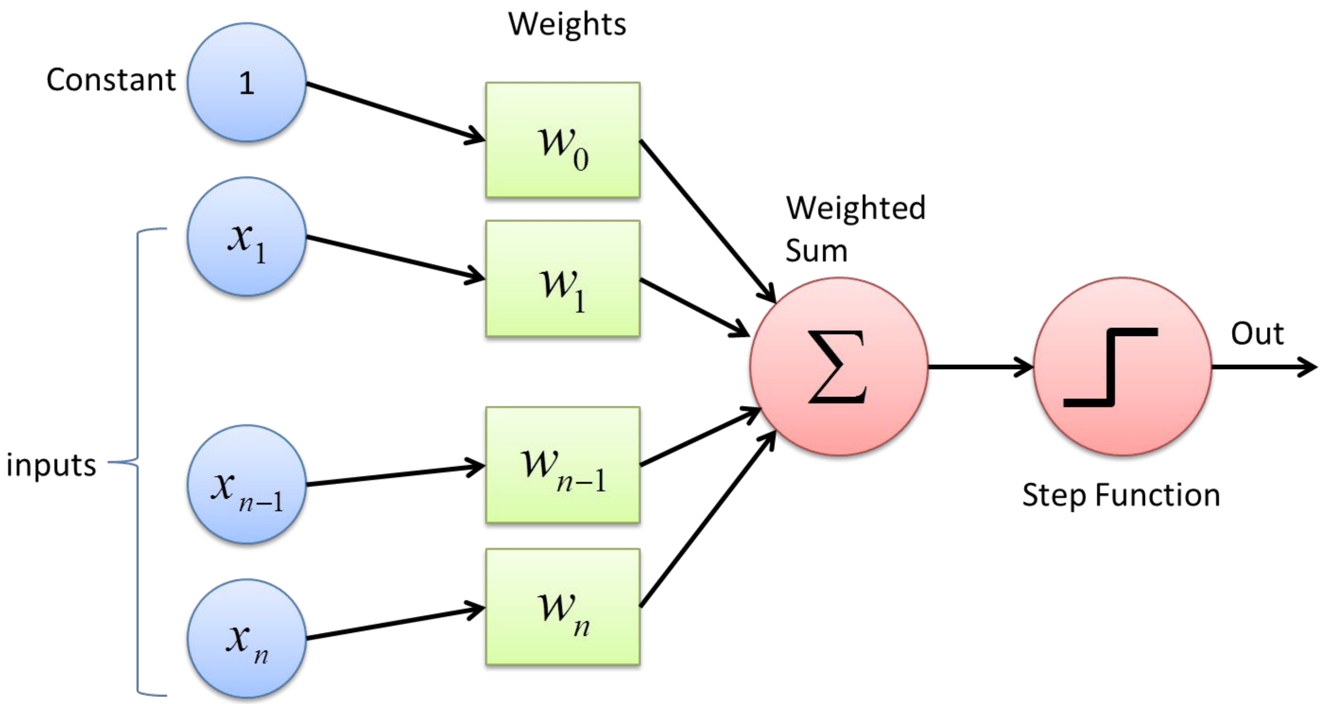
Concepts in machine learning

- symbolic AI (GOFAI) vs subsymbolic AI (connectionism)
- supervised vs unsupervised learning
- evaluation
  - accuracy, precision, recall
  - confusion matrix (TP, TN, FP, FN)

mathematical concepts (indirectly)

- basic probability
- joint probabilities
- conditional probabilities
- independence

The Perceptron Algorithm



The Perceptron Algorithm (again)

$$\boxed{\hat{y} = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)}$$

w is the weight vector ($w_1 w_2 \ldots w_i$)

x is the input vector ($x_1 x_2 \ldots x_i$)

b is the bias, still a regular number

sign( ) is the activation function

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

w · x is the dot product of **w** and **x**

the dot product is just shorthand for the weighted sum!

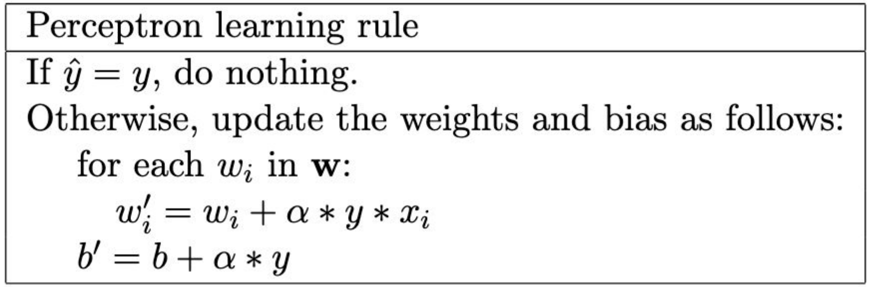
The Dot Product

Definition

$$\mathbf{a} = \langle a_1, a_2, a_3 \rangle \quad \mathbf{b} = \langle b_1, b_2, b_3 \rangle$$

$$\boxed{\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + a_3 b_3}$$

Perceptron Learning



```
Perceptron learning rule
If $\hat{y} = y$, do nothing.
Otherwise, update the weights and bias as follows:
    for each $w_i$ in $\mathbf{w}$:
        $w_i' = w_i + \alpha * y * x_i$
    $b' = b + \alpha * y$
```

w_i' :	the new weight at index i (pronounced 'w eye prime')
w_i :	the current weight at index i
α :	the learning rate (alpha) usually small
y :	the correct class label
x_i :	the input at index i

In English:

If we guessed the right answer, leave everything as is.

If we guessed the wrong answer, then move the weights a little bit towards a correct prediction for that input. Do this by updating all of the weights to look a little bit more like the input at that weight.

Visualizing the Perceptron rule with dials as weights

When the perceptron incorrectly outputs POS for a J shape, turn all of the dials for ON inputs a little bit towards NEG,and turn all of the dials for OFF inputs towards positive

Example: The forward pass

Forward Pass:

dataset:

Data point	x_1 = emojis	x_2 = period	y (correct label)
a	1	2	+1 (coworker)
b	2	1	+1 (coworker)
c	-1	-1	-1 (boss)
d	-2	1	-1 (boss)

Your task is to apply the perceptron update rule iteratively in order to train the Perceptron. You initialize the weights w = [0,0] and bias b= 0. You pick a learning rate of α = 1.

$$\hat{y} = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$$

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

Example: The forward pass

dataset:

Data point	x_1 = emojis	x_2 = period	y (correct label)
a	1	2	+1 (coworker)
b	2	1	+1 (coworker)
c	-1	-1	-1 (boss)
d	-2	1	-1 (boss)

Your task is to apply the perceptron update rule iteratively in order to train the Perceptron. You initialize the weights w = [0,0] and bias b= 0. You pick a learning rate of α = 1.

$$\hat{y} = \text{sign}(\overbrace{\mathbf{w} \cdot \mathbf{x}}^{\bullet} + b)$$

$$sign(z) = \begin{cases} 1, & \text{if } z > 0 \\ -1, & \text{if } z \le 0 \end{cases}$$

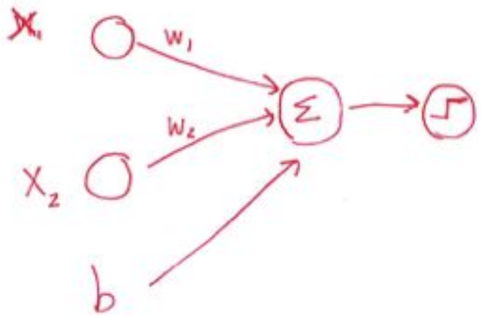
Forward Pass: A

$W = [0,0] \quad b = 0 \quad x = [1,2]$

$$\begin{aligned} \vec{W} \cdot \vec{X} &= W_1 X_1 + W_2 X_2 \\ &= 0(1) + 0(2) \\ &= 0 \end{aligned}$$

$$\begin{aligned} \hat{Y} &= \text{sign}(0 + 0) \\ &= \text{sign}(0) \\ &= -1 \end{aligned}$$

$\hat{Y} = +1$ ✗ wrong - update



Example: The backward pass

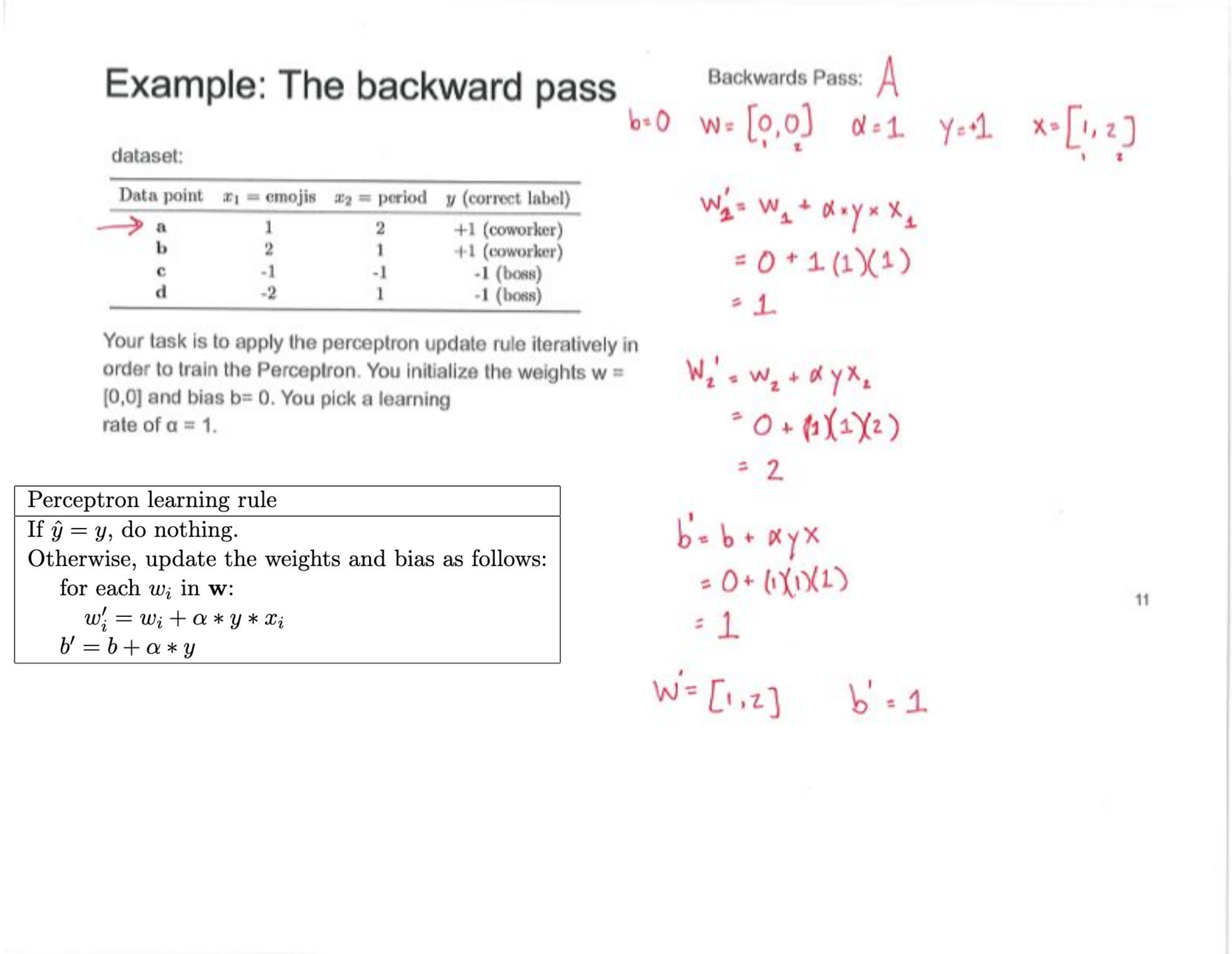
Backwards Pass:

dataset:

Data point	x_1 = emojis	x_2 = period	y (correct label)
a	1	2	+1 (coworker)
b	2	1	+1 (coworker)
c	-1	-1	-1 (boss)
d	-2	1	-1 (boss)

Your task is to apply the perceptron update rule iteratively in order to train the Perceptron. You initialize the weights w = [0,0] and bias b= 0. You pick a learning rate of α = 1.

```
Perceptron learning rule
If $\hat{y} = y$, do nothing.
Otherwise, update the weights and bias as follows:
    for each $w_i$ in $\mathbf{w}$:
        $w'_i = w_i + \alpha * y * x_i$
    $b' = b + \alpha * y$
```



Example: The backward pass

dataset:

Data point	x_1 = emojis	x_2 = period	y (correct label)
a	1	2	+1 (coworker)
b	2	1	+1 (coworker)
c	-1	-1	-1 (boss)
d	-2	1	-1 (boss)

Your task is to apply the perceptron update rule iteratively in order to train the Perceptron. You initialize the weights w = [0,0] and bias b= 0. You pick a learning rate of α = 1.

Backwards Pass: A

$b=0 \quad w=[\underset{1}{0},\underset{2}{0}] \quad \alpha=1 \quad y=+1 \quad x=[\underset{1}{1},\underset{2}{2}]$

$w_1' = w_1 + \alpha * y * x_1$
$= 0 + 1(1)(1)$
$= 1$

$w_2' = w_2 + \alpha y x_2$
$= 0 + (1)(1)(2)$
$= 2$

$b' = b + \alpha y x$
$= 0 + (1)(1)(1)$
$= 1$

$w' = [1,2] \quad b' = 1$

11

```
Perceptron learning rule
If $\hat{y} = y$, do nothing.
Otherwise, update the weights and bias as follows:
    for each $w_i$ in $\mathbf{w}$:
        $w_i' = w_i + \alpha * y * x_i$
    $b' = b + \alpha * y$
```

Training on real data

One run through training data through the update rule is called an "epoch."

We typically train neural nets with multiple epochs, i.e. the algorithm sees each training pair several times.

The model **converges** when it stops updating.

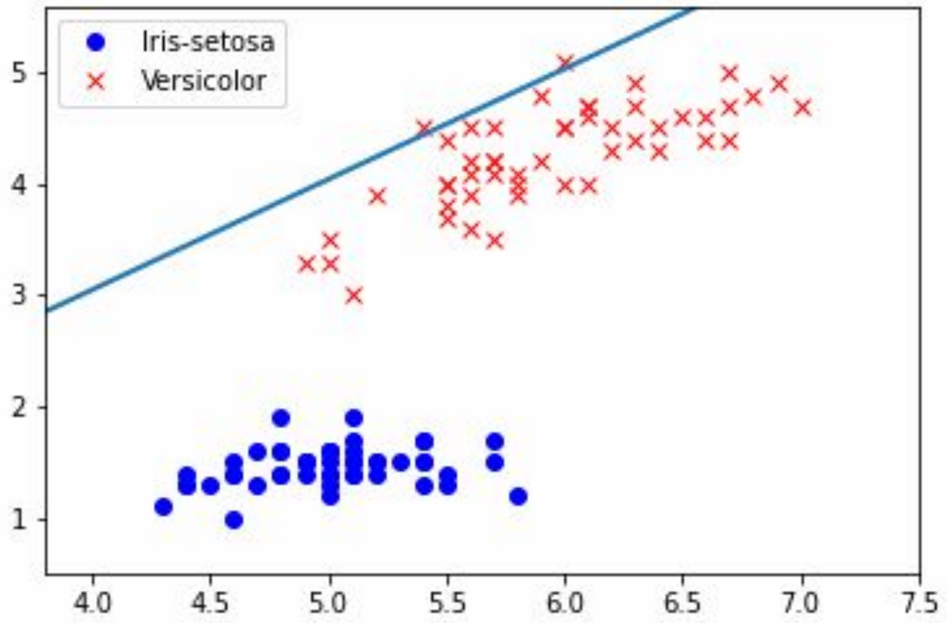
The perceptron is guaranteed to converge if the data are **linearly separable**.

Perceptron Learning

The model **converges** when it stops updating.

The perceptron learning rule is guaranteed to **converge** if the data are **linearly separable.**

linear separability: a line exists that separates the two types of points



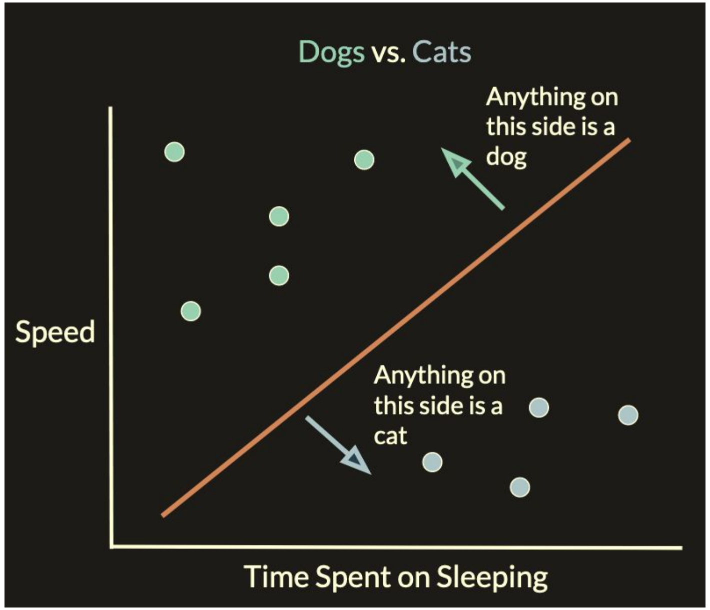
Geometric Interpretation

we can make decision function (the weighted sum) into an equation for a line:

$$w_1 x_1 + w_2 x_2 + b = 0$$

Rewrite that line in standard y=mx+b form

$$x_2 = -(w_1/w_2)x_1 - (b/w_2)$$



https://karthikvedula.com/2024/01/05/visualizing-the-perceptron-learning-algorithm/

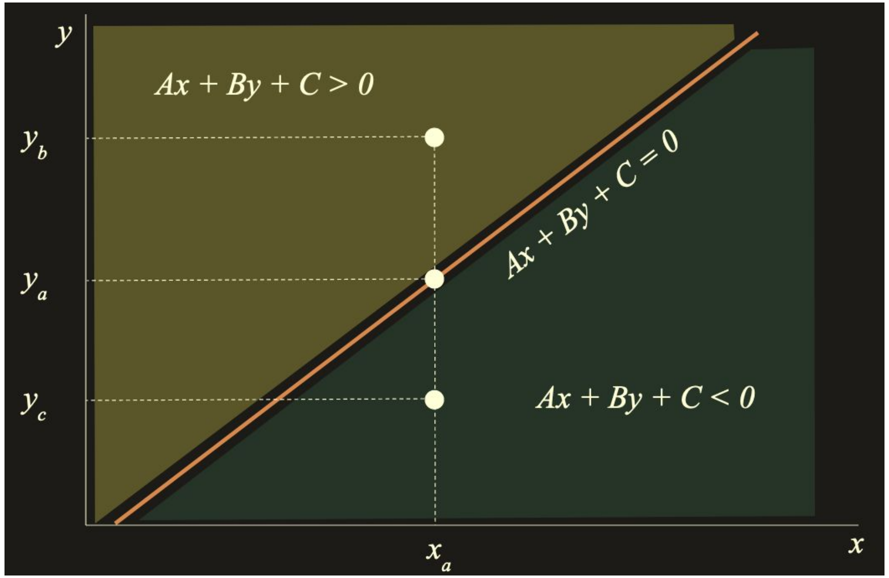
Geometric Interpretation

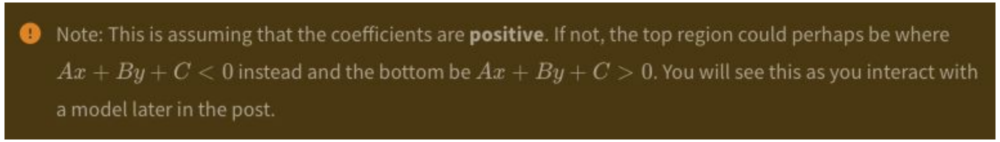
we can make decision function (the weighted sum) into an equation for a line:

$$w_1 x_1 + w_2 x_2 + b = 0$$

Rewrite that line in standard y=mx+b form

$$x_2 = -(w_1/w_2)x_1 - (b/w_2)$$



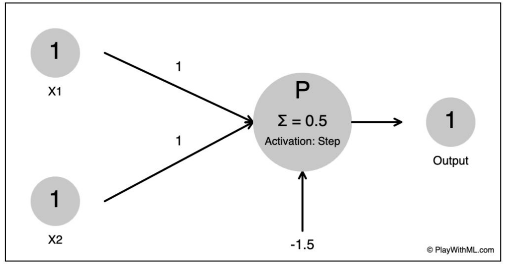

Note: This is assuming that the coefficients are **positive**. If not, the top region could perhaps be where $Ax + By + C < 0$ instead and the bottom be $Ax + By + C > 0$. You will see this as you interact with a model later in the post.

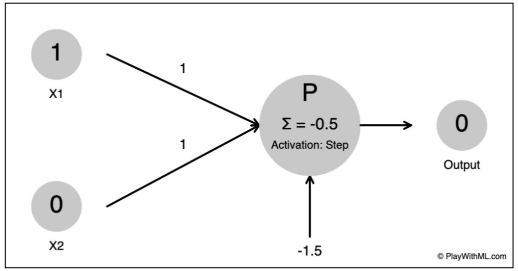
https://karthikvedula.com/2024/01/05/visualizing-the-perceptron-learning-algorithm/

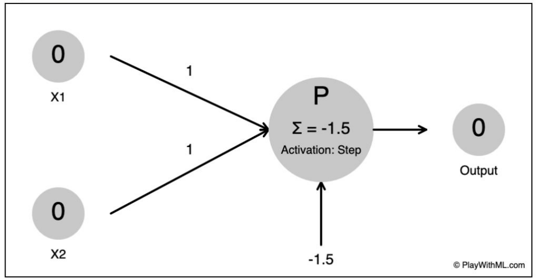
What can the perceptron do?

- which side of a piece of paper is the X on?
- logical functions

Perceptron AND gate

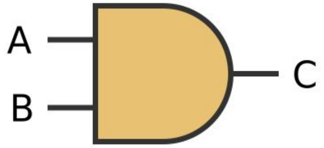






perceptron

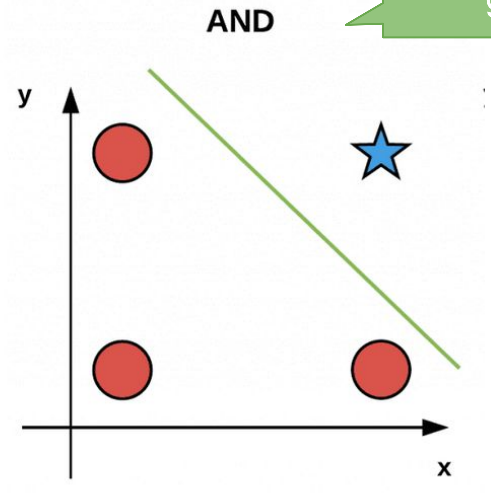
physical circuit



lookup table

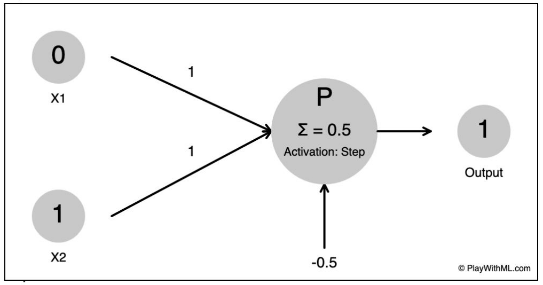
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

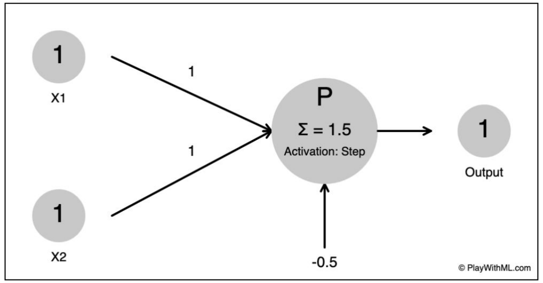
geometric

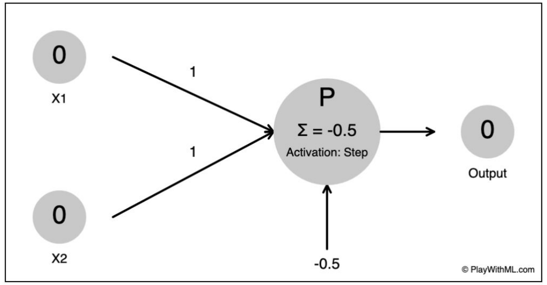


4 different representations of logical AND

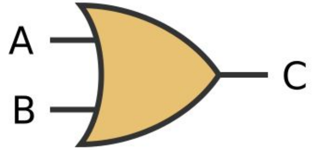
Perceptron OR gate



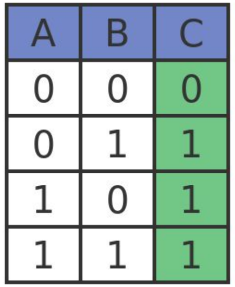




physical circuit



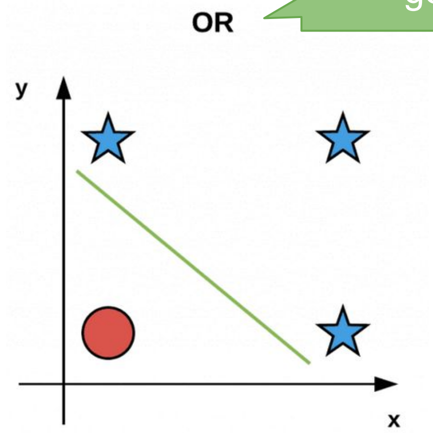
lookup table



A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

perceptron

geometric

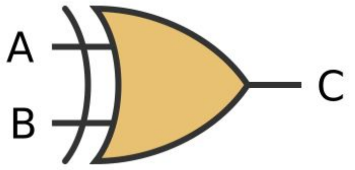


4 different representations of logical OR

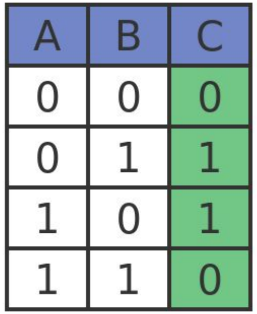
Perceptron XOR gate

- XOR is NOT linearly separable. We can't draw a line in space to solve this problem, so we can't design (or learn) a solution with the perceptron

physical circuit

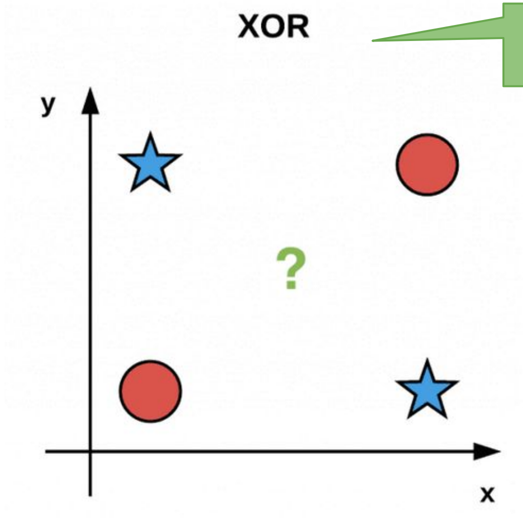


lookup table



A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

geometric

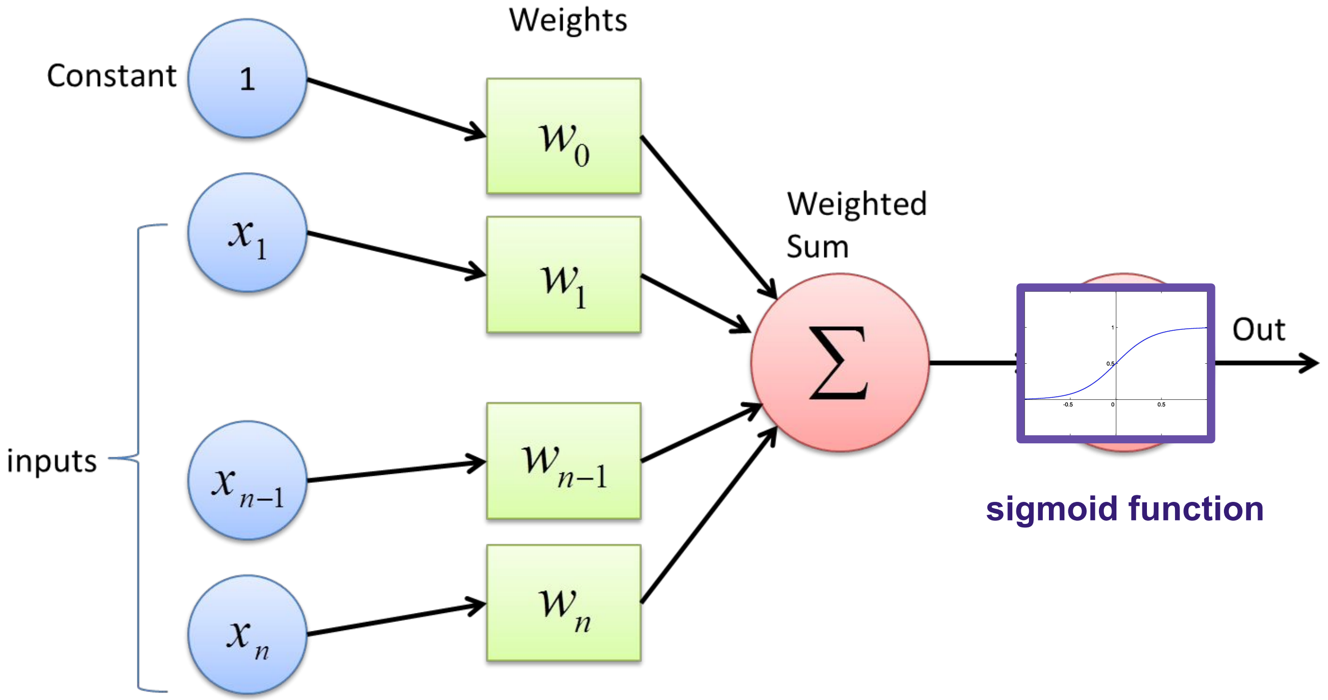


different representations of logical XOR

Linear Regression

The only difference is the activation function that we choose!

Linear regression uses the **sigmoid** activation function



sigmoid function

Interactive Demos

- perceptron: https://karthikvedula.com/2024/01/05/visualizing-the-perceptron-learning-algorithm/
- perceptron: https://perceptron.streamlit.app/
- dot product: https://maththebeautiful.com/dot-product/

The Curly Fry Conundrum (Jennifer Golbeck)

https://www.youtube.com/watch?v=hgWie9dnssU

Demographic study shows that “Liking” the page for curly fries is one of the strongest indicators of intelligence.

How can this be?

Spurious Correlations/Features

Eating curly fries don't make you intelligent. It doesn't cause intelligence.

But there is a correlation in the dataset between liking curly fries and intelligence.

When a machine learning system learns to use this feature for predicting intelligence, this mistake can lead to poor generalization and biased outcomes.