Homework 6

- We will turn our recipe regression classifier into a neural network classifier.
- Also: we'll work on trying to understand:
  - *How does the model make its predictions?*
  - *How well do they generalize to novel domains (medieval recipes!)?*

Recap: Regression

How to do classification

For each feature x_i, weight w_i tells us importance of x_i

- (Plus we'll have a bias b)

We'll sum up all the weighted features and the bias

$$z = \left(\sum_{i=1}^{n} w_i x_i\right) + b$$

$$z = w \cdot x + b$$

If this sum is high, we say y=1
If low, then y=0

$$P(y=1) = \sigma(w \cdot x + b)$$

$$P(y=0) = 1 - \sigma(w \cdot x + b)$$

Classifying sentiment for input x

Weights: [2.5, -2.5, -3, 4, 0.1]

Bias: 0.2

Features for radish sandwich review: [2, 0, 1, 0, 3.33]

$$P(y=1) = \sigma(w \cdot x + b)$$
$$= 0.926$$

$$P(y=0) = 1 - \sigma(w \cdot x + b)$$
$$= 0.074$$

Really wanted to love this sandwich. Of course, I love the butter and the baguette (and the salt), but the radishes just don't do anything for me.

The two phases of logistic regression

Test: Given a test example x, we compute $p(y \mid x)$ using learned weights w and b, and return whichever label ($y = 1$ or $y = 0$) is higher probability

Training: we learn weights w and b using **stochastic gradient descent** and **cross-entropy loss**.

What About More Than Two Classes?

If >2 classes we use **multinomial logistic regression:**

The probability of everything must still sum to 1

```
P(positive|doc) + P(negative|doc) +
P(neutral|doc) = 1
```

Need a generalization of the sigmoid called **softmax**

- Takes a vector $z = [z1, z2, ..., zk]$ of k arbitrary values
- Outputs a probability distribution

Features in binary versus multinomial logistic regression

Binary: positive weight

$$x_5 = \begin{cases} 1 & \text{if } \text{“!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases} \qquad w_5 = 3.0$$

Multinominal: separate weights for each class:

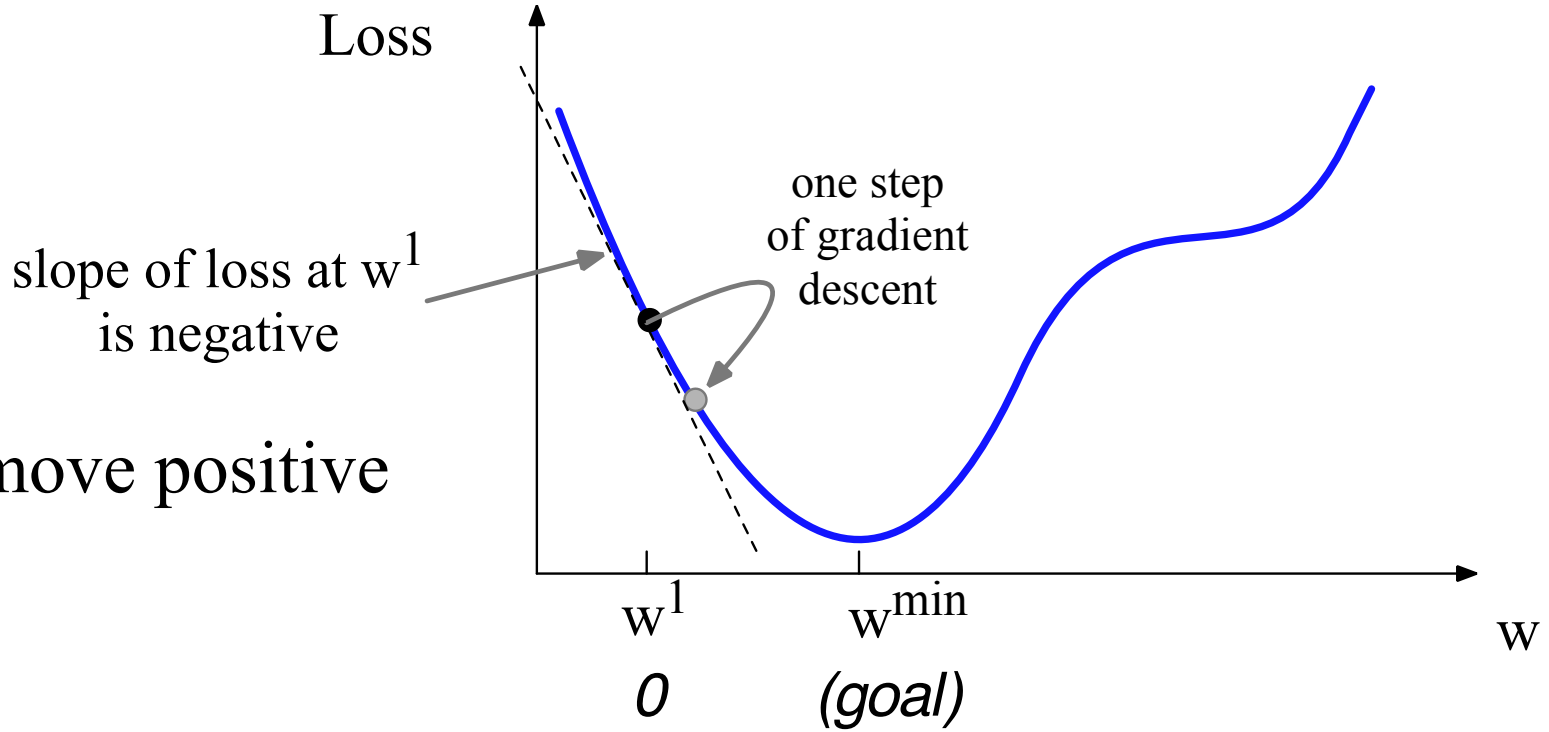
Feature	Definition	$w_{5,+}$	$w_{5,-}$	$w_{5,0}$
$f_5(x)$	$\begin{cases} 1 & \text{if } \text{“!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	3.5	3.1	−5.3

Recap: Gradient Descent

Our goal: minimize the loss

Q: Given current w, should we make it bigger or smaller?

A: Move *w* in the reverse direction from the slope of the function



So we'll move positive

Game

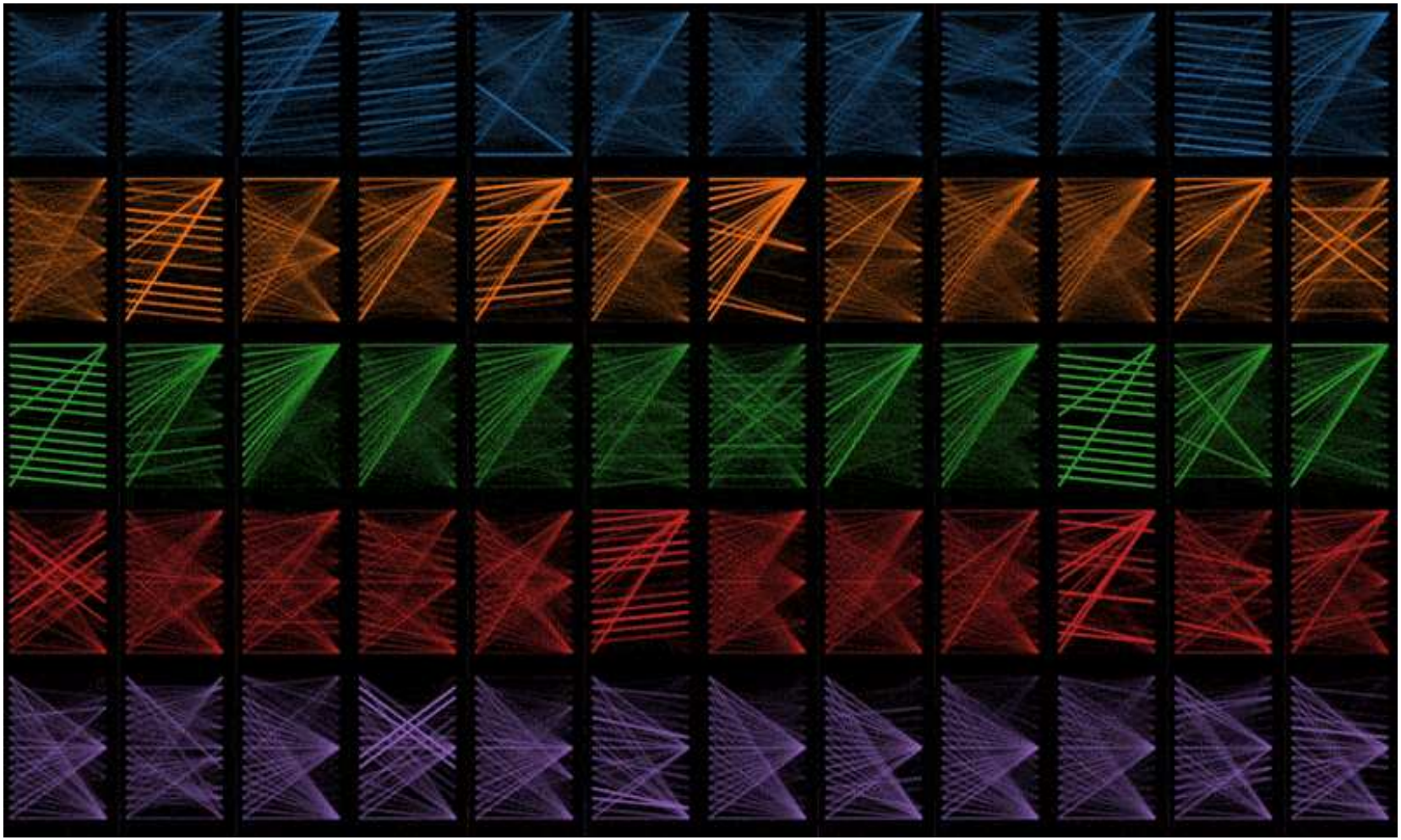
Neural Networks

This is a brain



estimated to have 100 trillion synapses connecting 100 billion neurons

This is not your brain

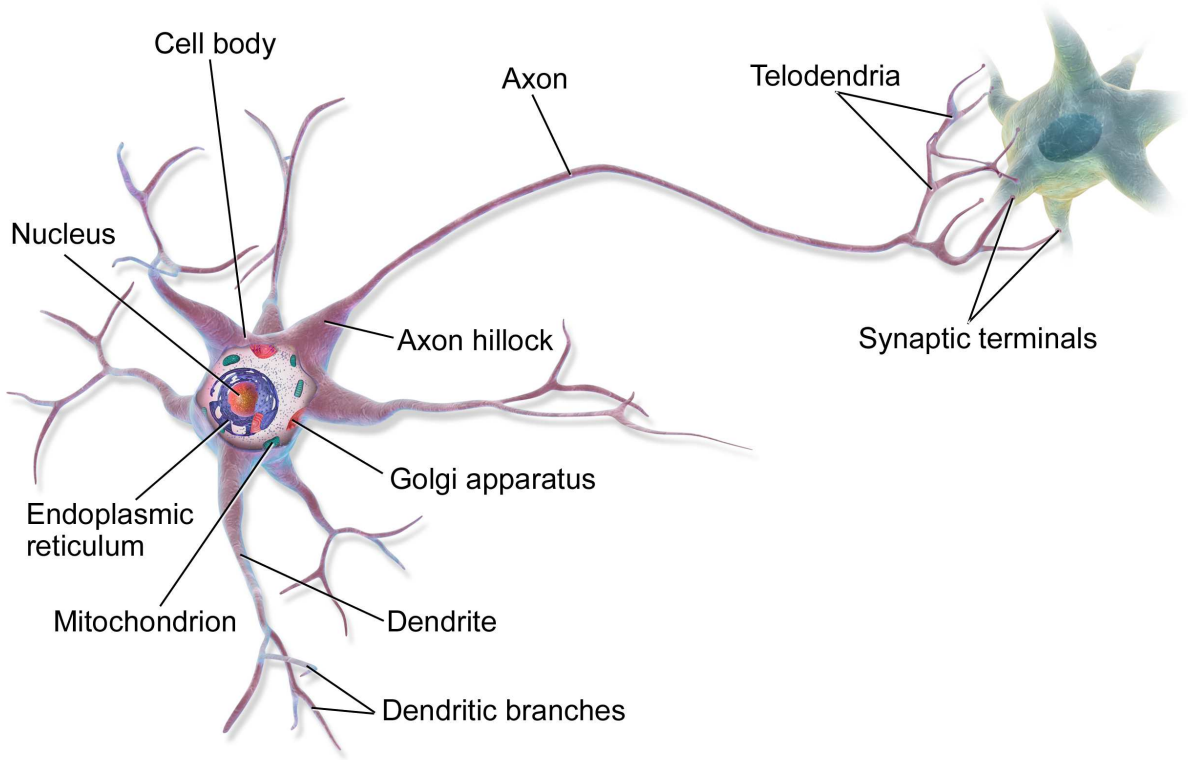


https://github.com/jessevig/bertviz

It's a large language model

(state-of-the-art models have around 10 trillion parameters)

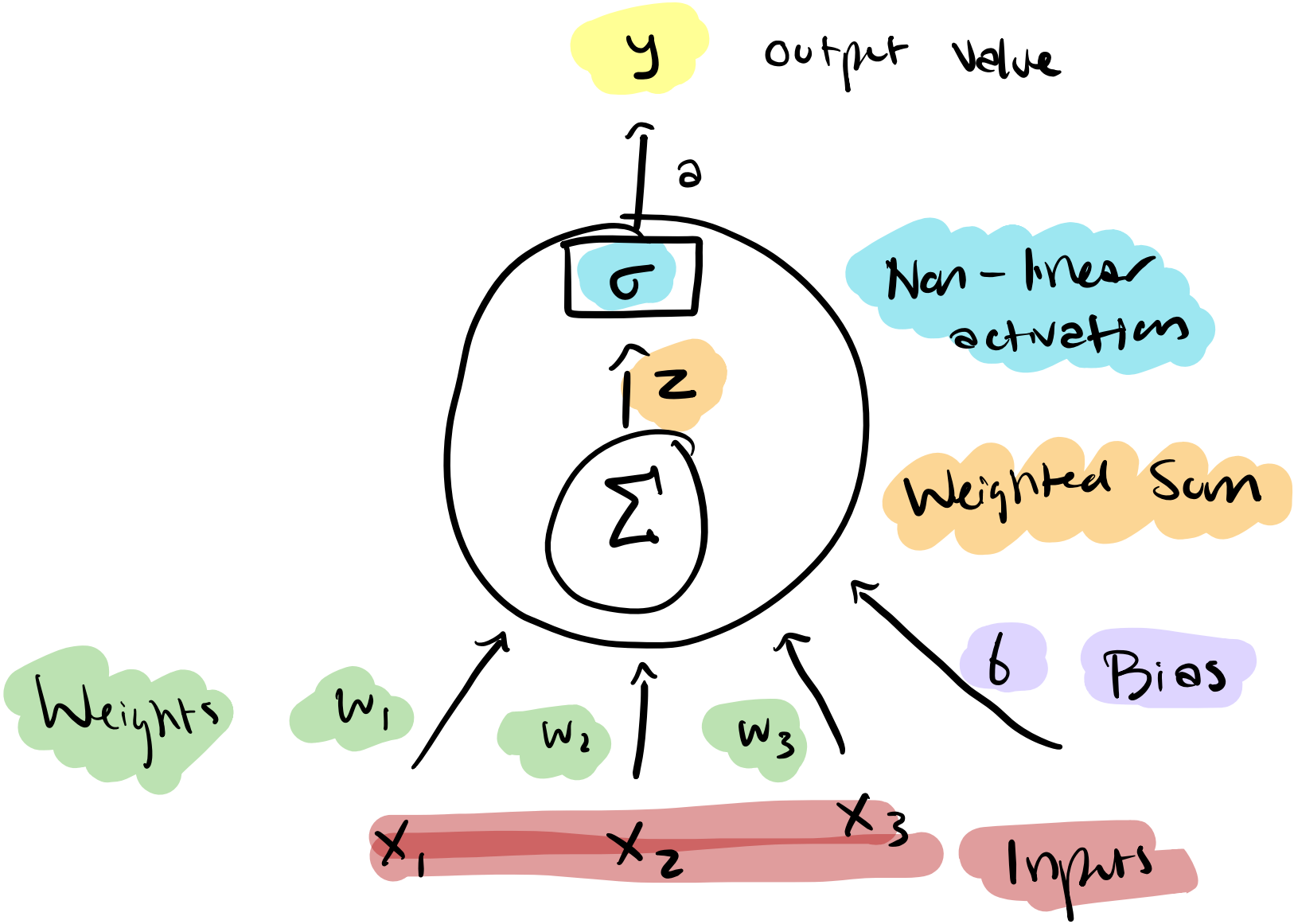
This is a brain



https://commons.wikimedia.org/w/index.php?curid=28761830

Neural Network Unit

This is not in your brain



Units in Neural Networks

Neural unit

Take weighted sum of inputs & a bias

$$z = b + \sum_i w_i x_i$$

$$z = wx + b$$

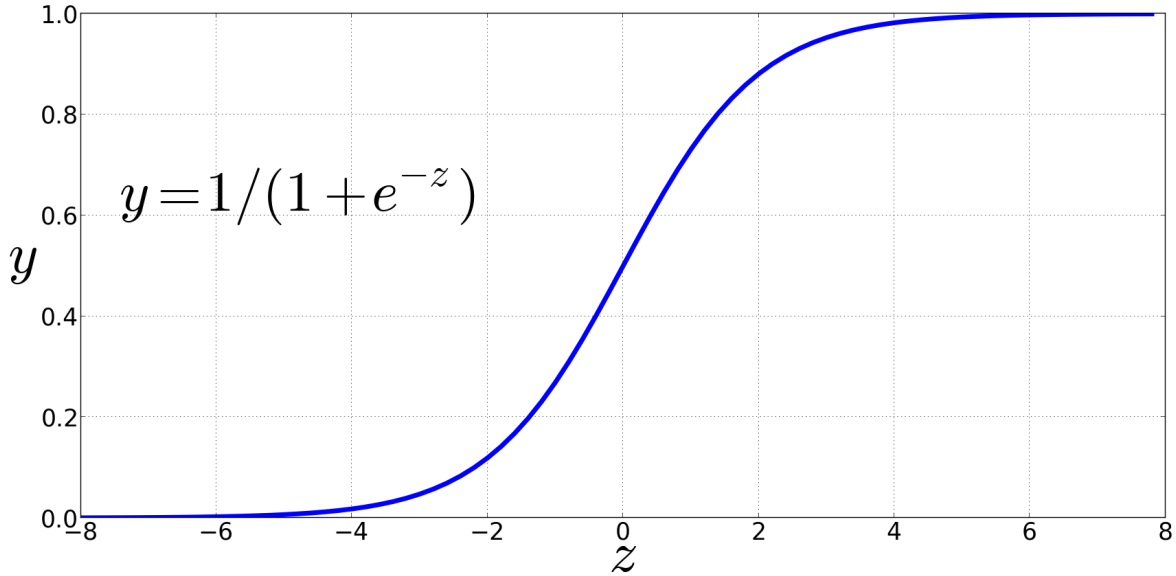
Instead of using z, we apply a non-linear activation function f:

$$y = f(z)$$

Non-Linear Activation Functions

We're already seen the sigmoid for logistic regression:

Sigmoid



Final function computed by a single unit

$$y = \sigma(wx + b)$$

Spot the differences

Neural Network Unit

$$z = b + \sum_i w_i x_i$$

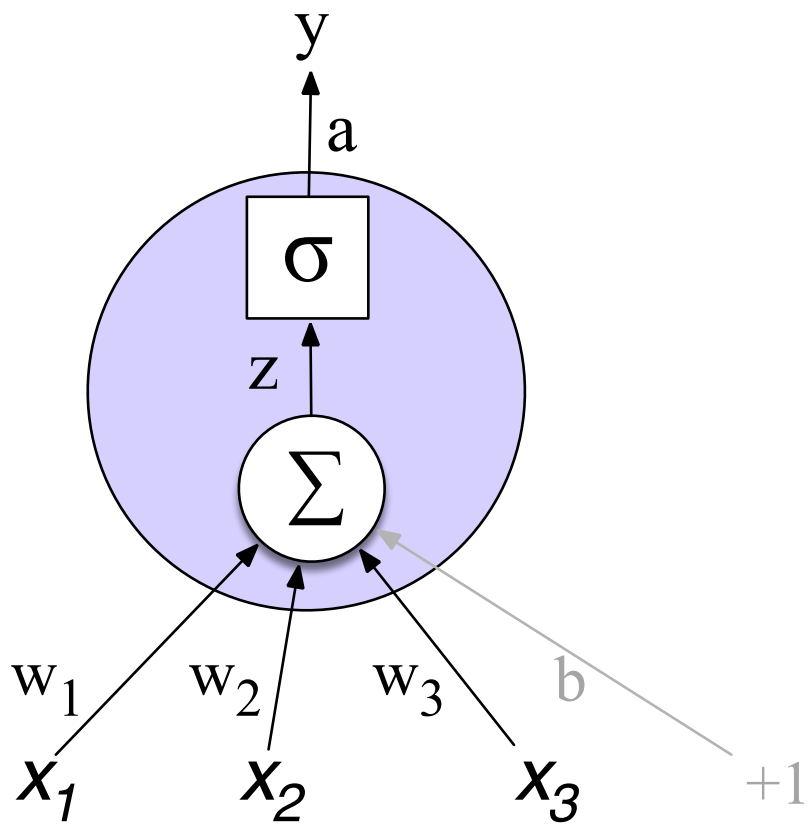
$$y = \sigma(wx + b)$$

Logistic Regression

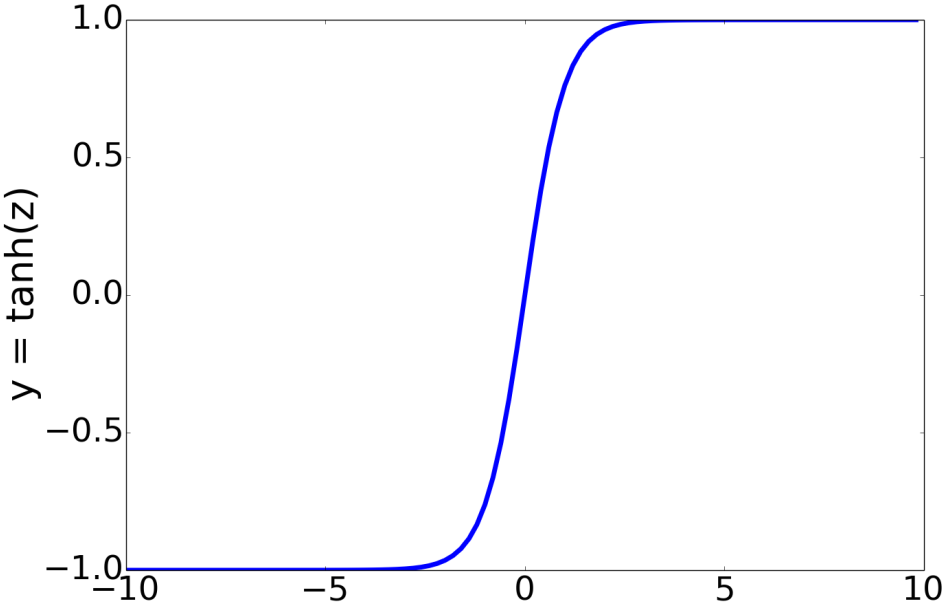
$$z = \left(\sum_{i=1}^{n} w_i x_i \right) + b$$

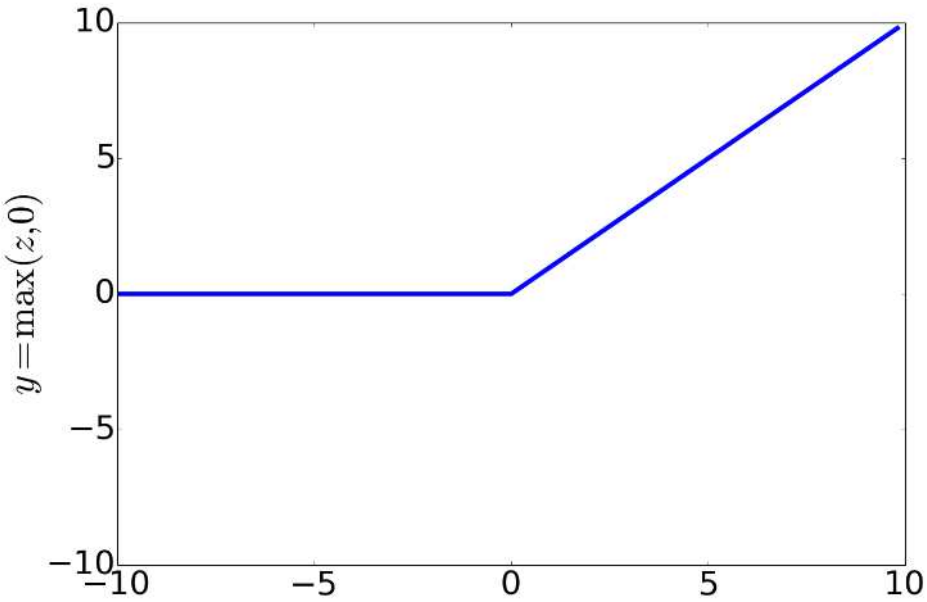
$$p(y=1) = \sigma(wx + b)$$

Final unit again



Non-Linear Activation Functions besides sigmoid





tanh

$$y = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

ReLU

Rectified Linear Unit

$$y = max(z, 0)$$

Example: XOR

The XOR problem

Minsky and Papert (1969)

Can neural units compute simple functions of input?

✓ AND

x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

✓ OR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1

XOR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

1 = True

Perceptrons: a very simple unit

A very simple neural unit

- Binary output (0 or 1)
- A simple thresholding output function in place of sigmoid:

$$y = \begin{cases} 0, & \text{if } \mathbf{w} \cdot \mathbf{x} + b \leq 0 \\ 1, & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \end{cases}$$

Solving AND

Deriving AND

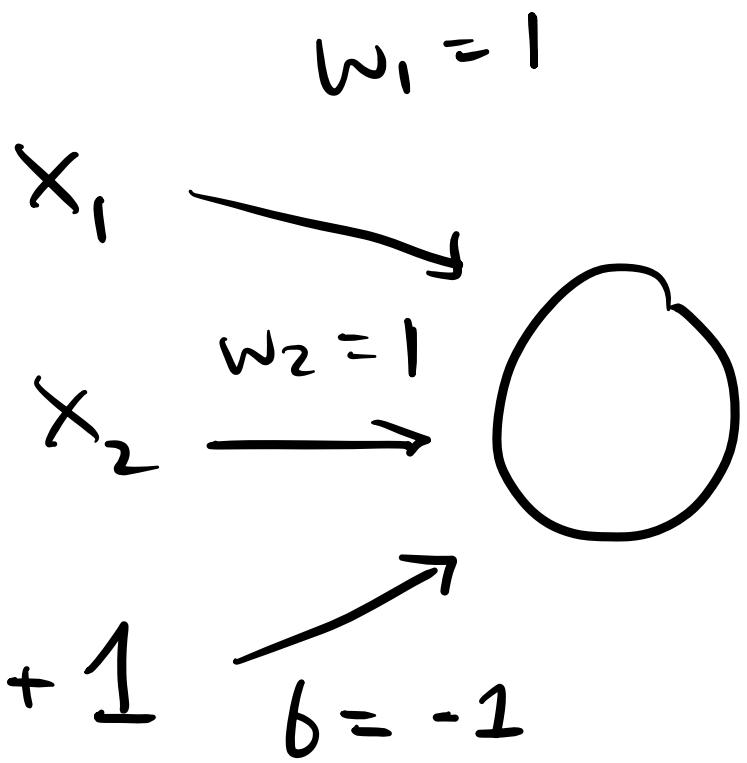
$$y = \begin{cases} 0, & \text{if } \mathbf{w} \cdot \mathbf{x} + b \leq 0 \\ 1, & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \end{cases}$$

AND		
x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

Goal: return 1 if x1 and x2 are 1

Deriving AND

Goal: return 1 if x1 and x2 are 1



AND

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

$w_1 = 1 \quad w_2 = 1$

$b = -1$

$z = w_1 x_1 + w_2 x_2 + b$

$1 \cdot 1 \quad 1 \cdot 1 \quad > -2$

$0 \quad 1 \cdot 1 \quad <= -1$

Exercise: solving OR

OR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1

Deriving OR

$w_1 = 1$

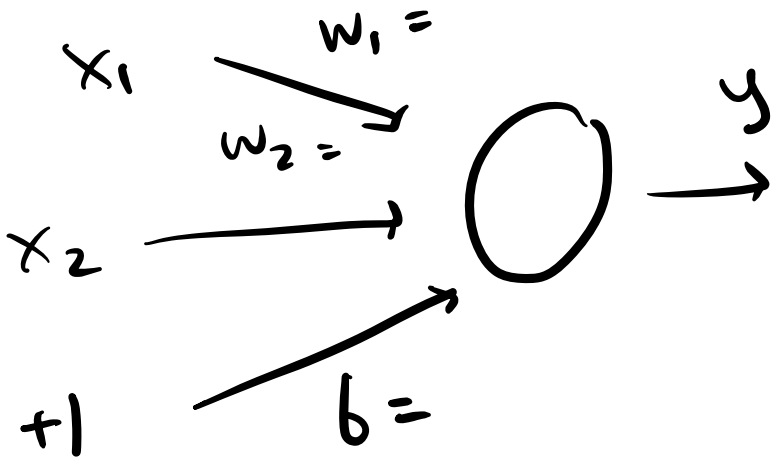
$w_2 = 1$

$b = 0$

$$y = \begin{cases} 0, & \text{if } \mathbf{w} \cdot \mathbf{x} + b \leq 0 \\ 1, & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \end{cases}$$

OR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1

Goal: return 1 if either input is 1



Deriving OR

Goal: return 1 if either input is 1

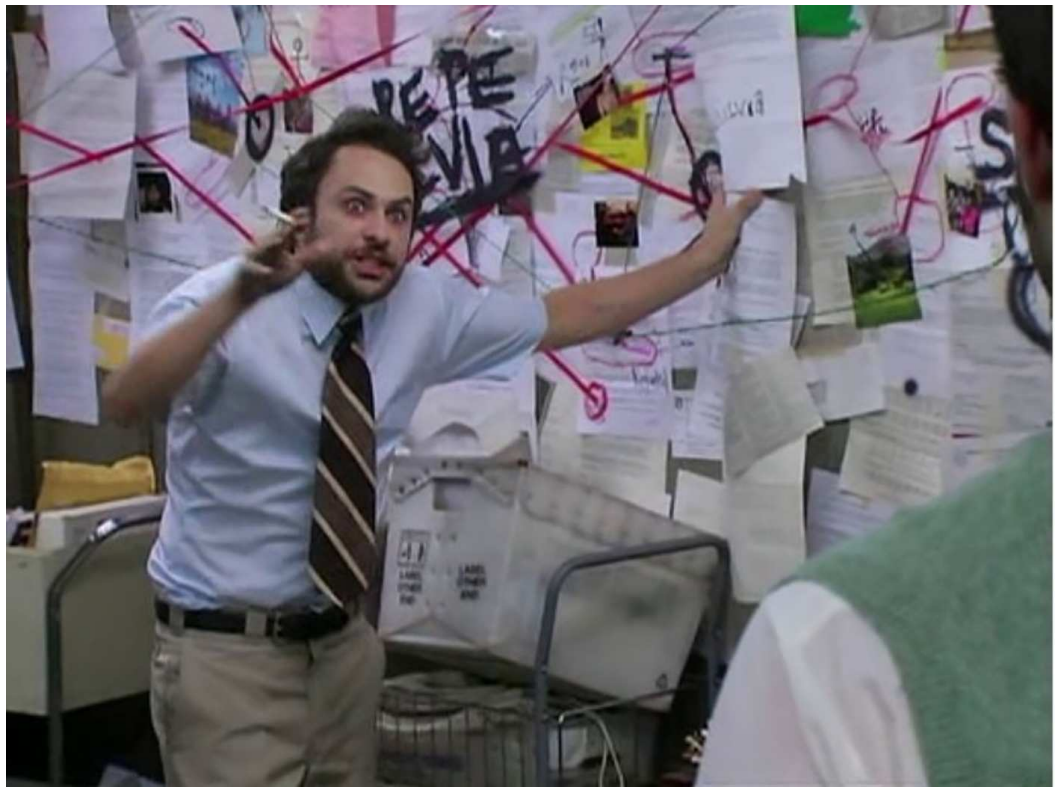
solving XOR

XOR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

Trick question!

It's not possible to capture XOR with perceptrons



Why? Perceptrons are linear classifiers

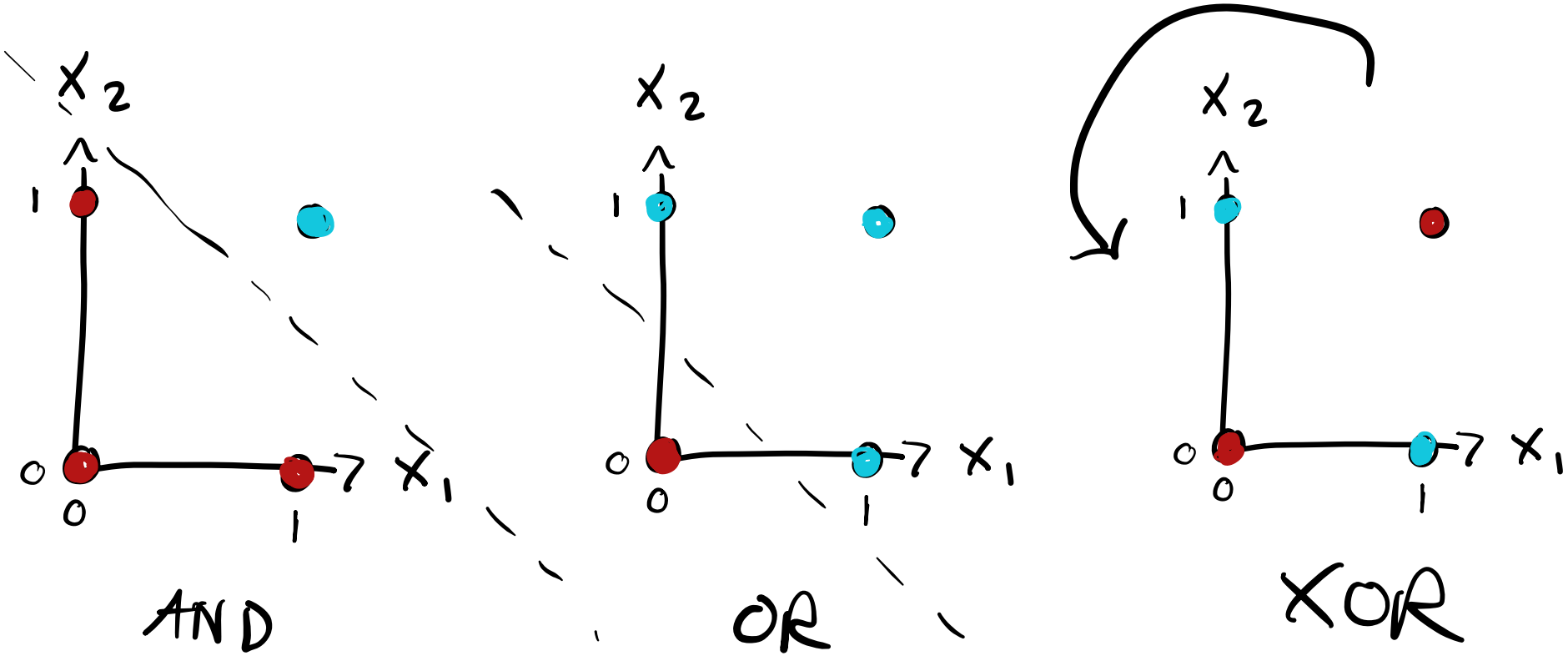
Perceptron equation is the equation of a line

$w_1x_1 + w_2x_2 + b = 0$

This line acts as a **decision boundary**

- 0 if input is on one side of the line
- 1 if on the other side of the line

Decision boundaries



= $y=0$

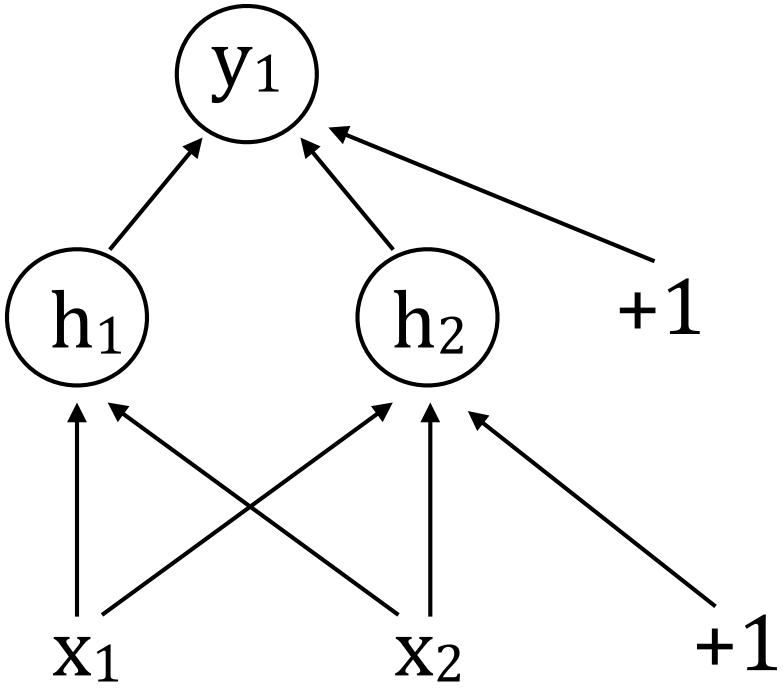
= $y=1$

Solution to the XOR problem

XOR can't be calculated by a single perceptron

XOR can be calculated by a layered network of units.

XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0



$$h_2 = \text{ReLu}(w_{21} x_1 + w_{22} x_2 + 1)$$