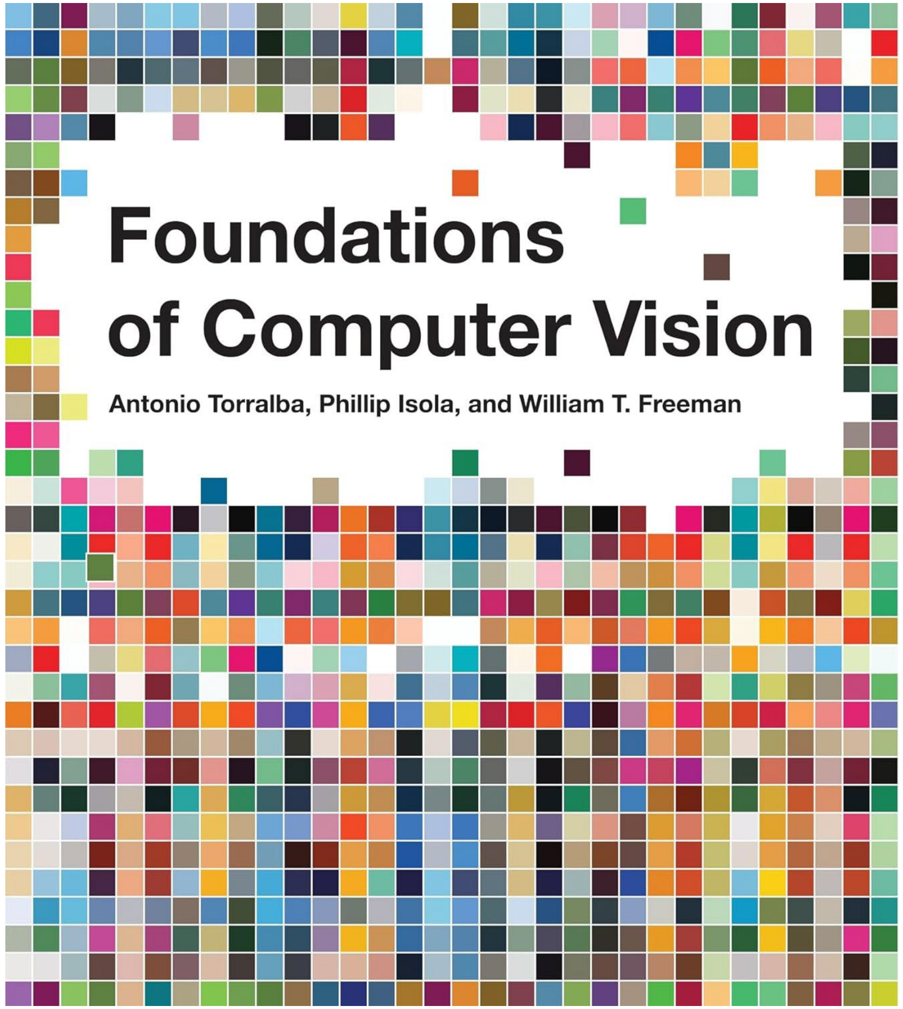
Reminders / Announcements

- **Homework 1 is due tonight!**
  - Each student gets 10 late days (total). See the syllabus for details.
  - You **DO NOT** need to email me for permission to use late days!
- **Project Proposal is due 10/03**
  - Group sizes <3 need my explicit permission!
  - Proposal needs to be turned in on Blackboard by each group member
- **Midterm Exam is on 10/20**
  - In class; closed-book; 1 hour; everything up to and including 10/15 lecture
  - More details in the next class.

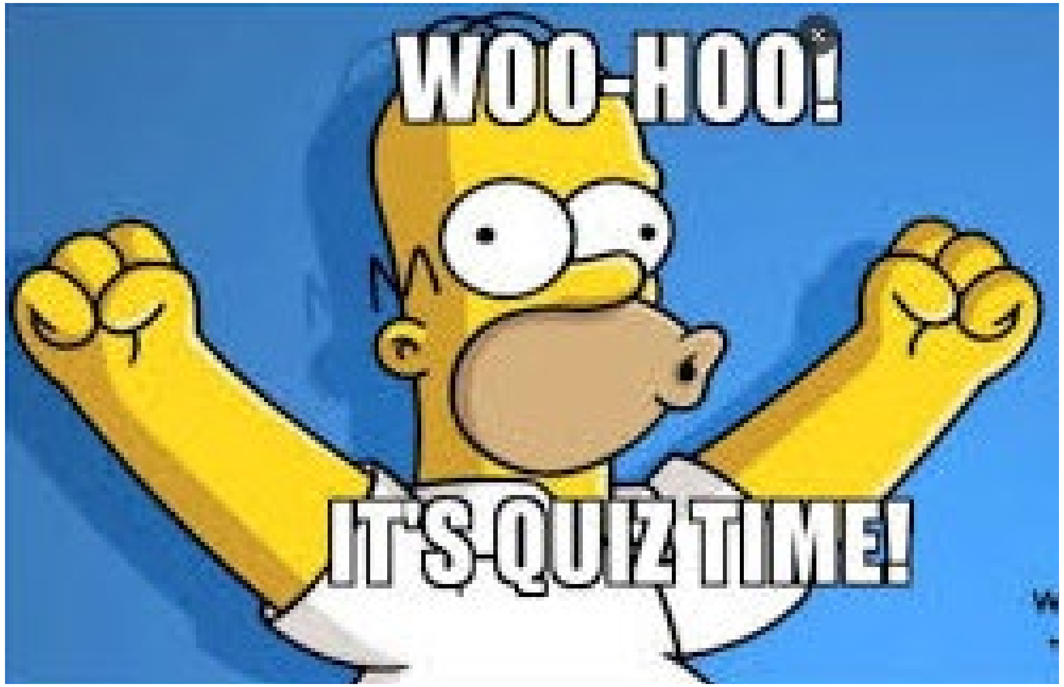
A New Useful Reference Book

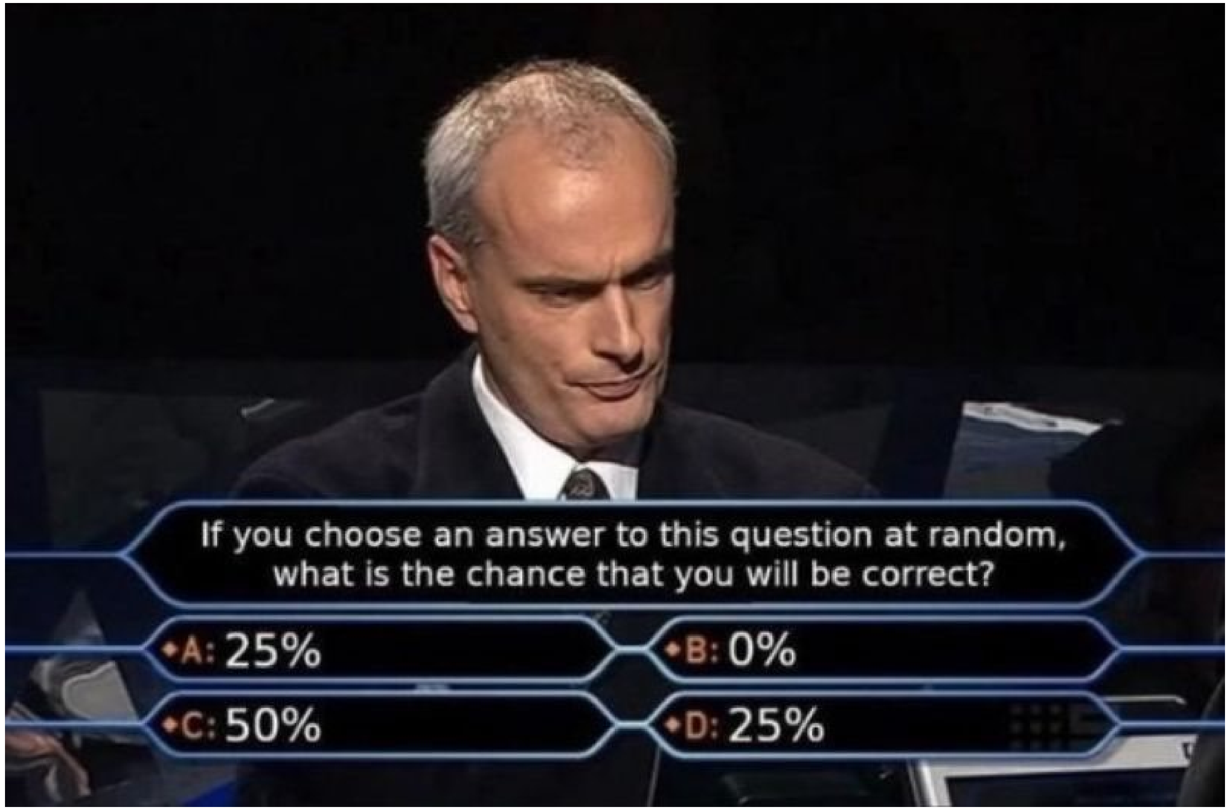
- Available for free on: https://visionbook.mit.edu/
- Published in 2024 – a modern take compared to the other reference books mentioned in the syllabus (these are also available for free)
- As a reminder: you are encouraged to read relevant chapters of reference books
  - Course website lists book chapters for each lecture
  - This is optional, but encouraged



I've requested UMBC library to buy it, but that might take some time …

Quiz 3!


WOO-HOO!
IT'S QUIZ TIME!


If you choose an answer to this question at random, what is the chance that you will be correct?
A: 25%
B: 0%
C: 50%
D: 25%

CMSC 472/672 Computer Vision

Lecture 8: Neural Networks

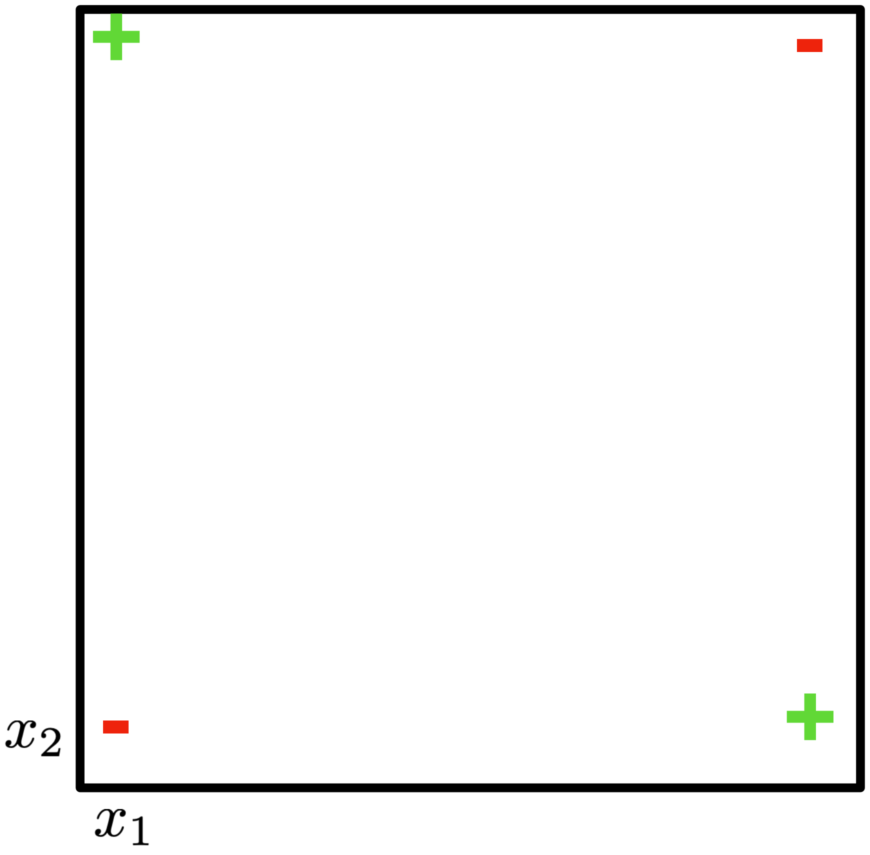
Some slides from Suren Jayasuriya (ASU), Phillip Isola (MIT)

UMBC

Artificial
Intelligence

$$\hat{y} = \boldsymbol{w}^\top \boldsymbol{x} + b$$

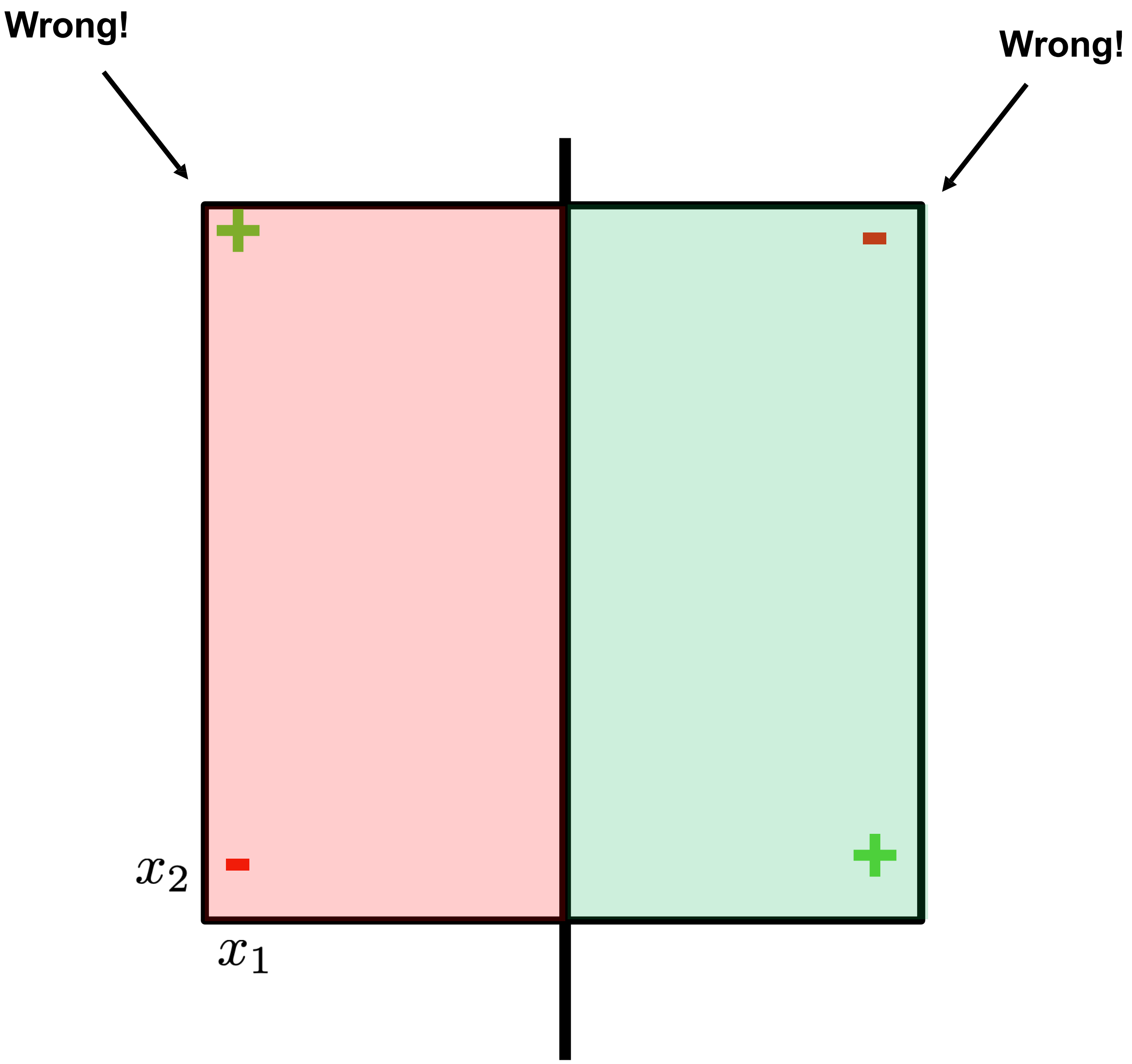
Limitations to linear classifiers



		x_2	
		0	1
x_1	0	0	1
	1	1	0

XOR

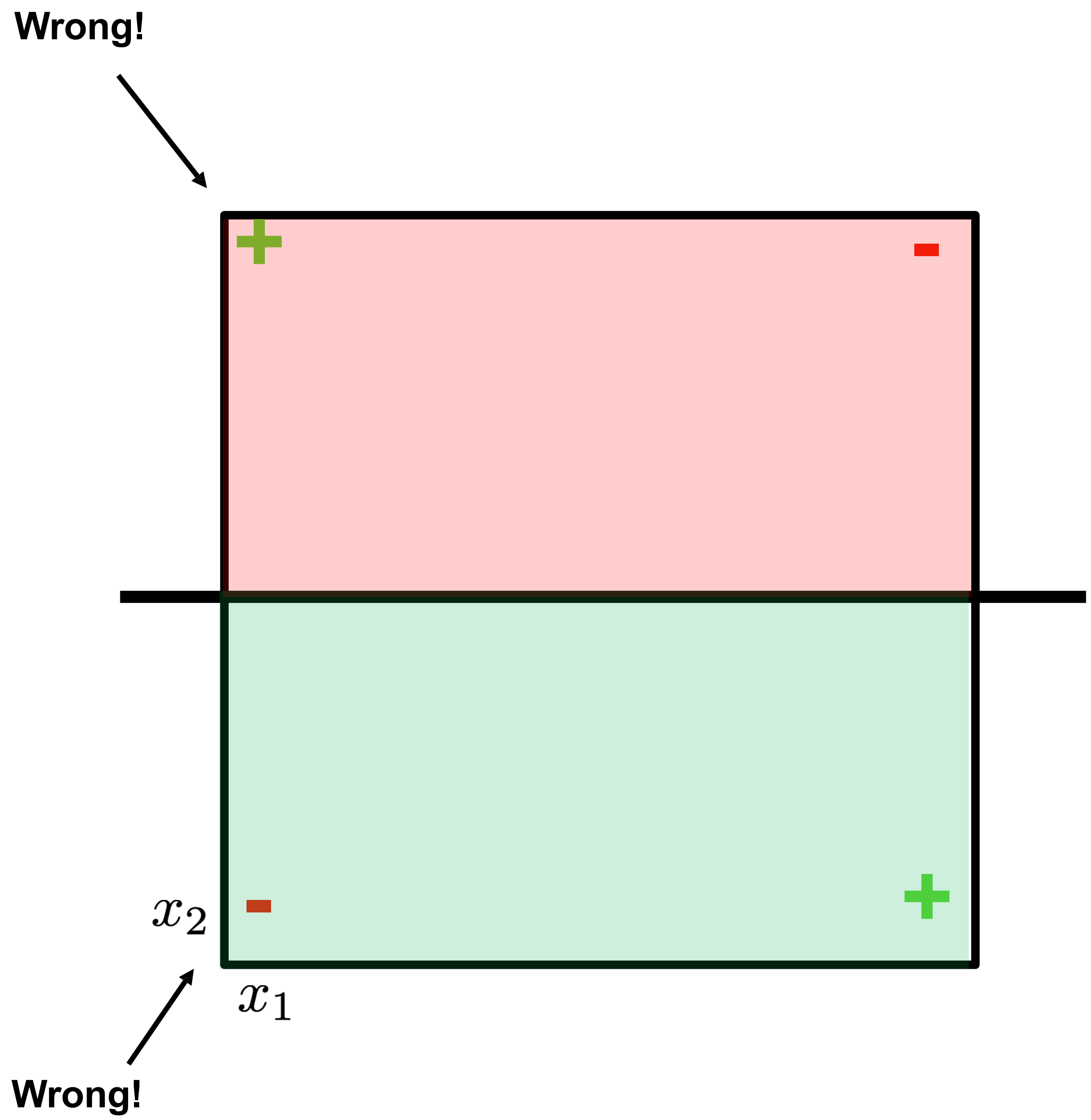
Limitations to linear classifiers



x_1 \ x_2	0	1
0	0	1
1	1	0

XOR

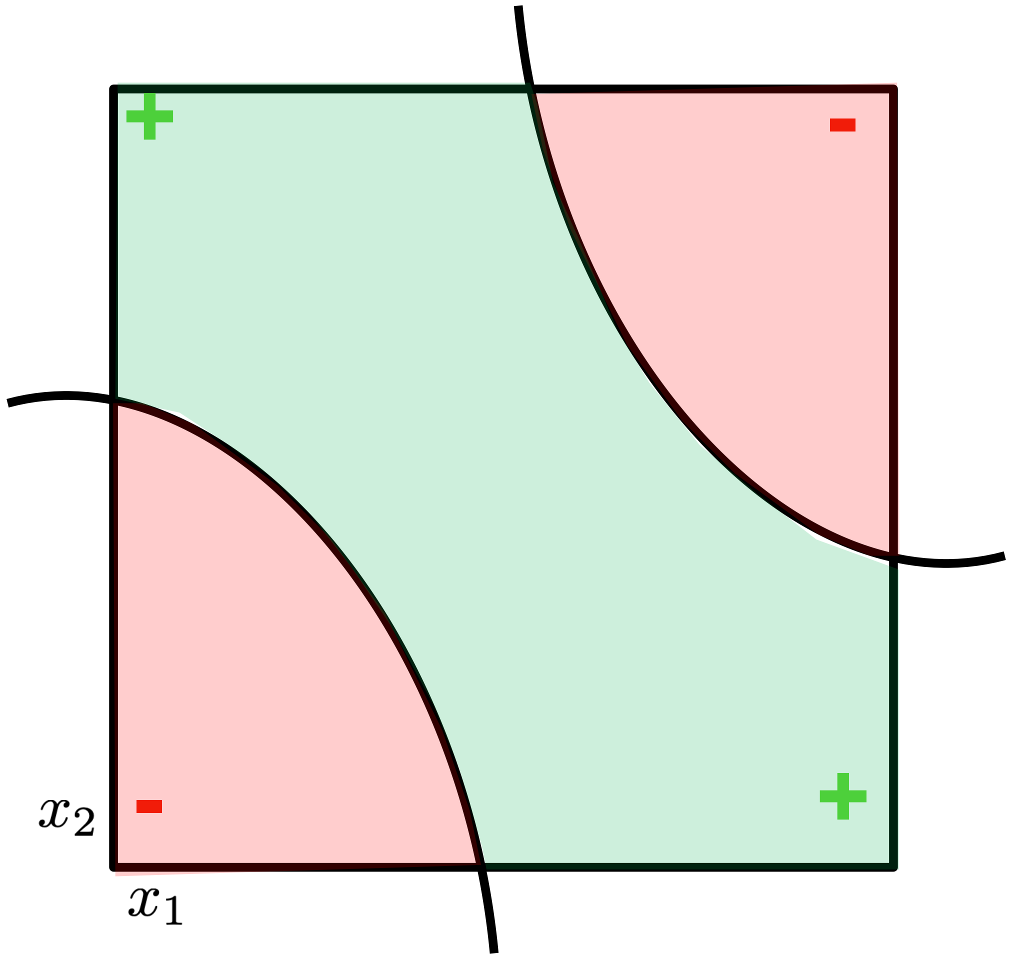
Limitations to linear classifiers



		x_2	
		0	1
x_1	0	0	1
	1	1	0

XOR

Goal: Non-linear decision boundary

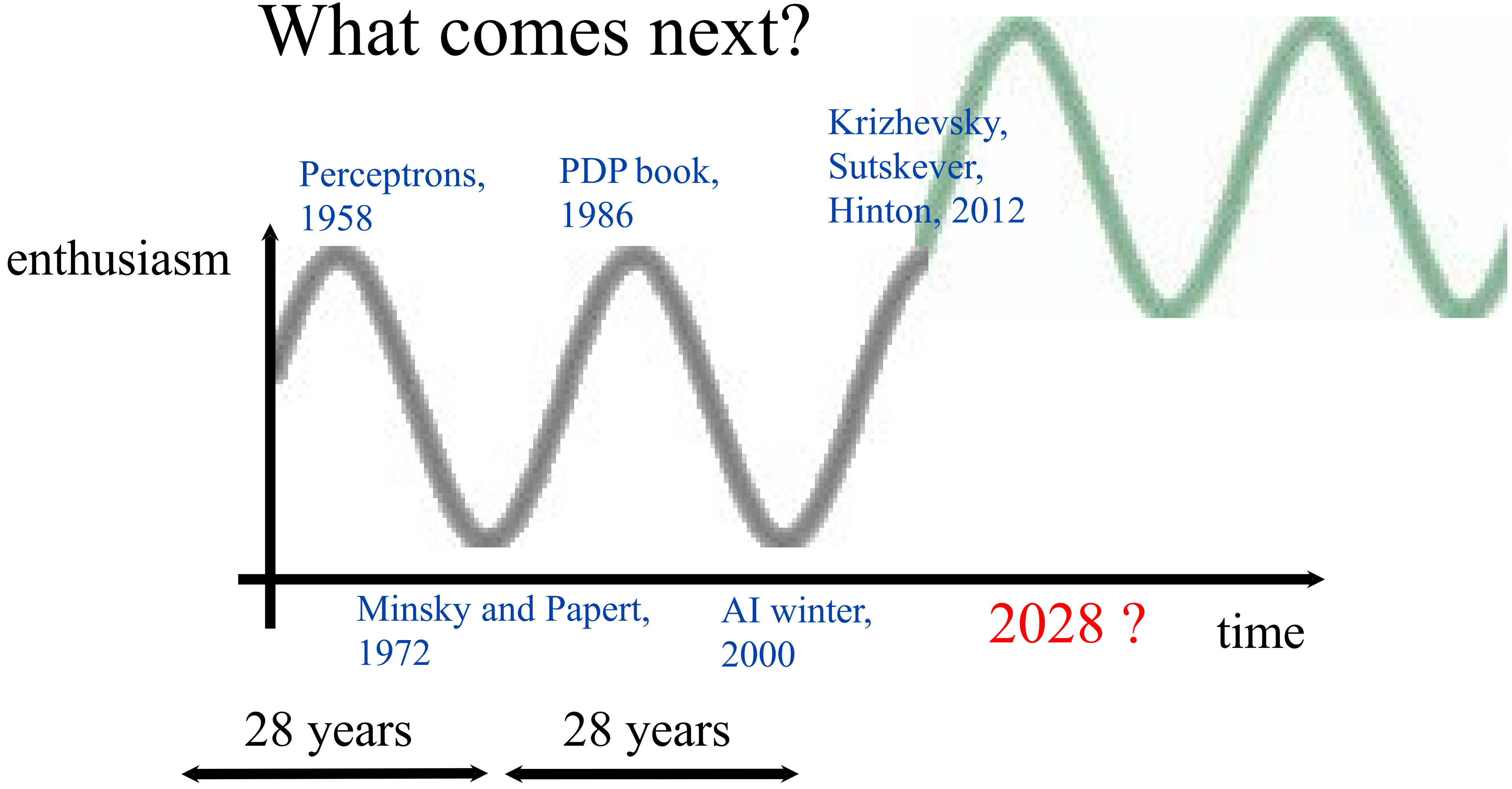


		x_2	
		0	1
x_1	0	0	1
	1	1	0

XOR

A brief history of Neural Networks

What comes next?



Parametric Approach

Image

Array of **32x32x3** numbers
(3072 numbers total)

$f(\mathbf{x}, \mathbf{W})$

10 numbers giving class scores

W

parameters or weights

Parametric Approach: Linear Classifier

$$f(x,W) = Wx$$

Image

Array of **32x32x3** numbers
(3072 numbers total)

→ f(**x**,**W**) →

10 numbers giving
class scores

↑

W

parameters
or weights

Parametric Approach: Linear Classifier

3072x1

$$\boxed{f(x,W)} = \boxed{W}\boxed{x}$$

10x1 10x3072

Image

Array of **32x32x3** numbers (3072 numbers total)

$f(\mathbf{x},\mathbf{W})$

10 numbers giving class scores

W

parameters or weights

Parametric Approach: Linear Classifier

$$\boxed{f(x,W)} = \underbrace{\boxed{W}}_{10\times3072}\overbrace{\boxed{x}}^{3072\times1} + \boxed{b}\ \ 10\times1$$

10x1

Image

Array of **32x32x3** numbers
(3072 numbers total)

f(**x**,**W**)

10 numbers giving
class scores

W

parameters
or weights

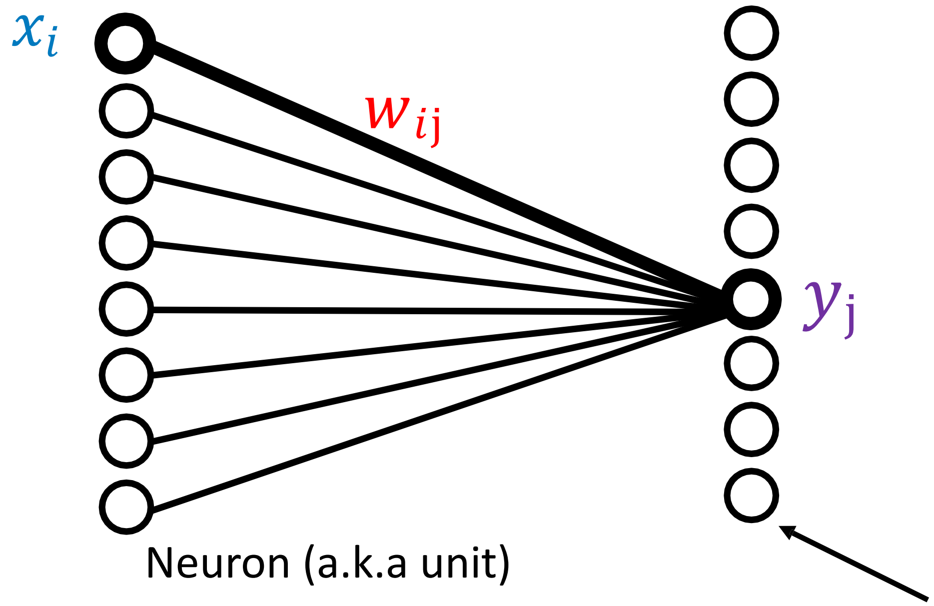
Computation in a neural net

Let's say we have some 1D input that we want to convert to some new feature space:

Linear layer

Input representation

Output representation



weights

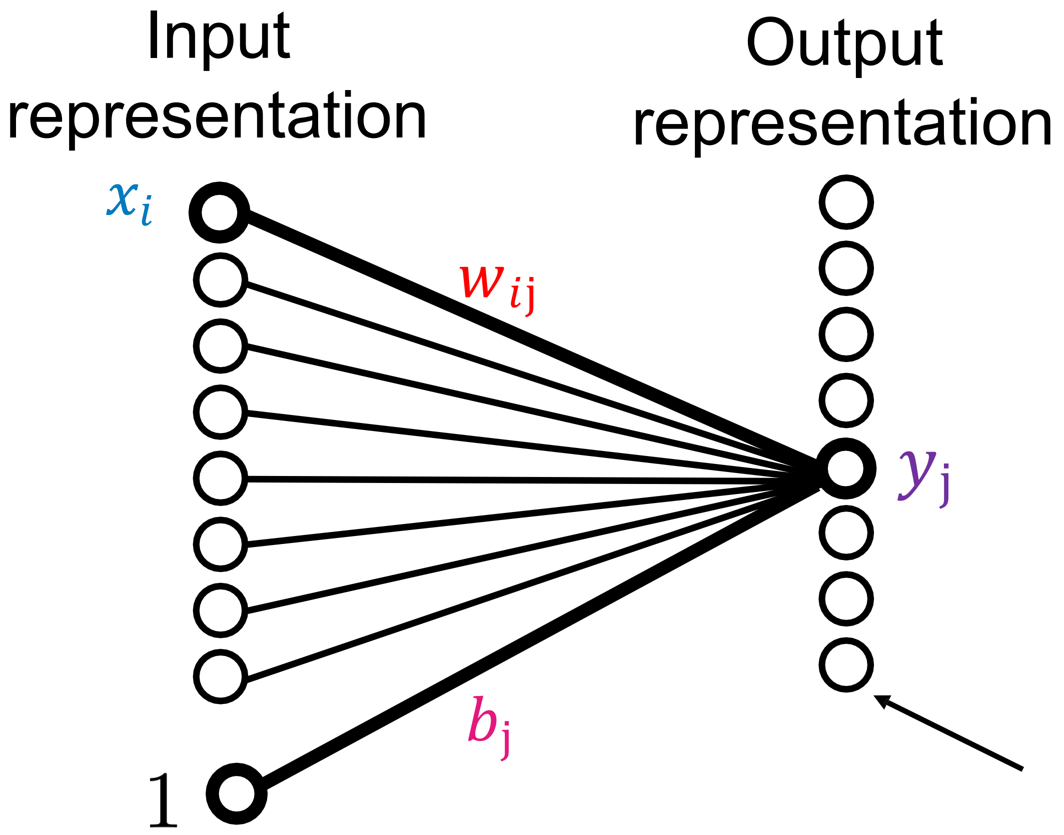
$$y_j = \sum_i w_{ij} x_i$$

Adapted from: Isola, Torralba, Freeman

Computation in a neural net

Let's say we have some 1D input that we want to convert to some new feature space

Linear layer



$$y_j = \sum_i w_{ij} x_i + b_j$$

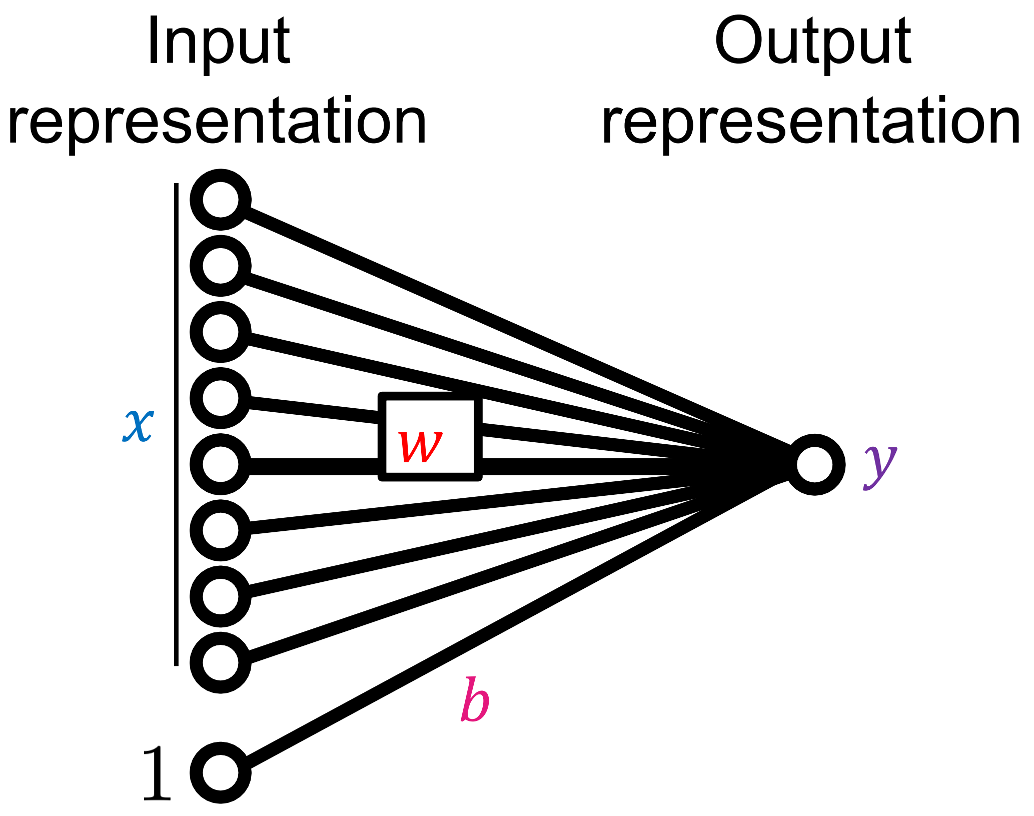
weights

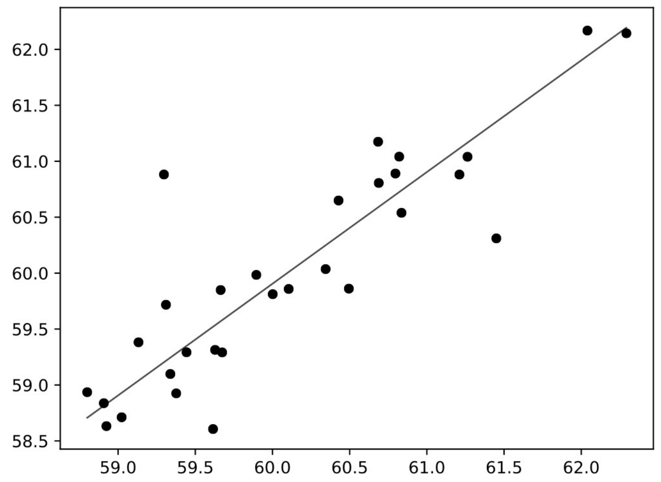
bias

Adapted from: Isola, Torralba, Freeman

Example: Linear Regression

Linear layer



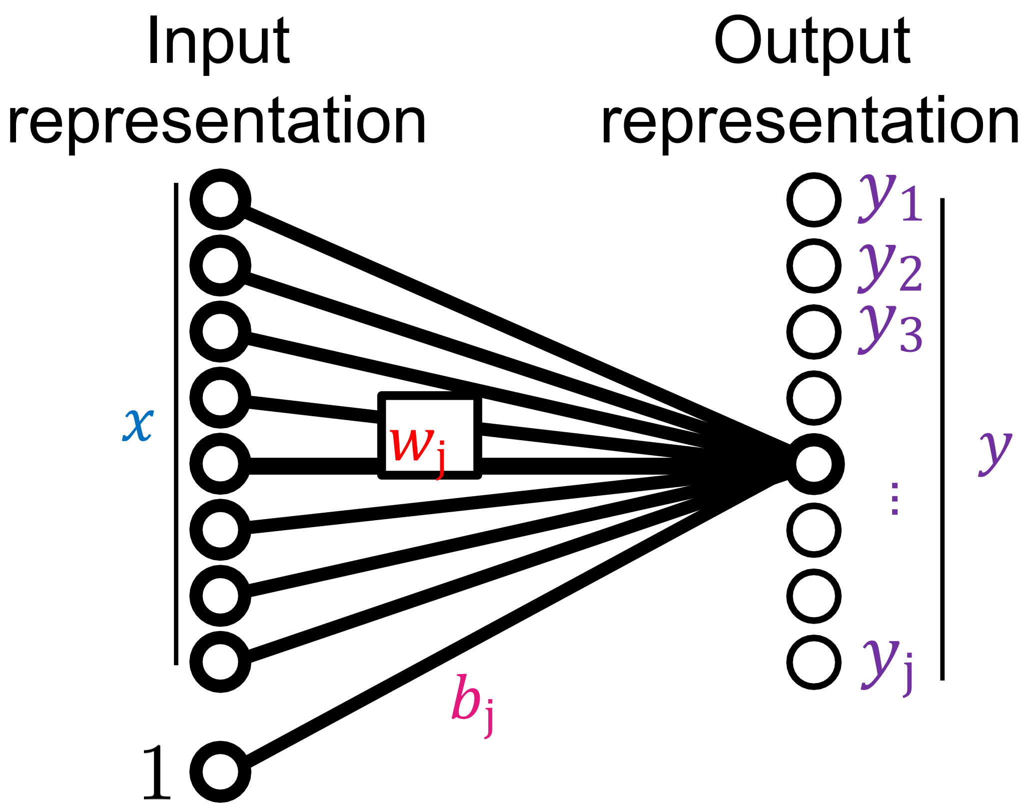


$$f_{\mathbf{w},b}(\mathbf{x}) = \mathbf{x}^T\mathbf{w} + b$$

Adapted from: Isola, Torralba, Freeman

Computation in a neural net – Full Layer

Linear layer



$$y = Wx + b$$

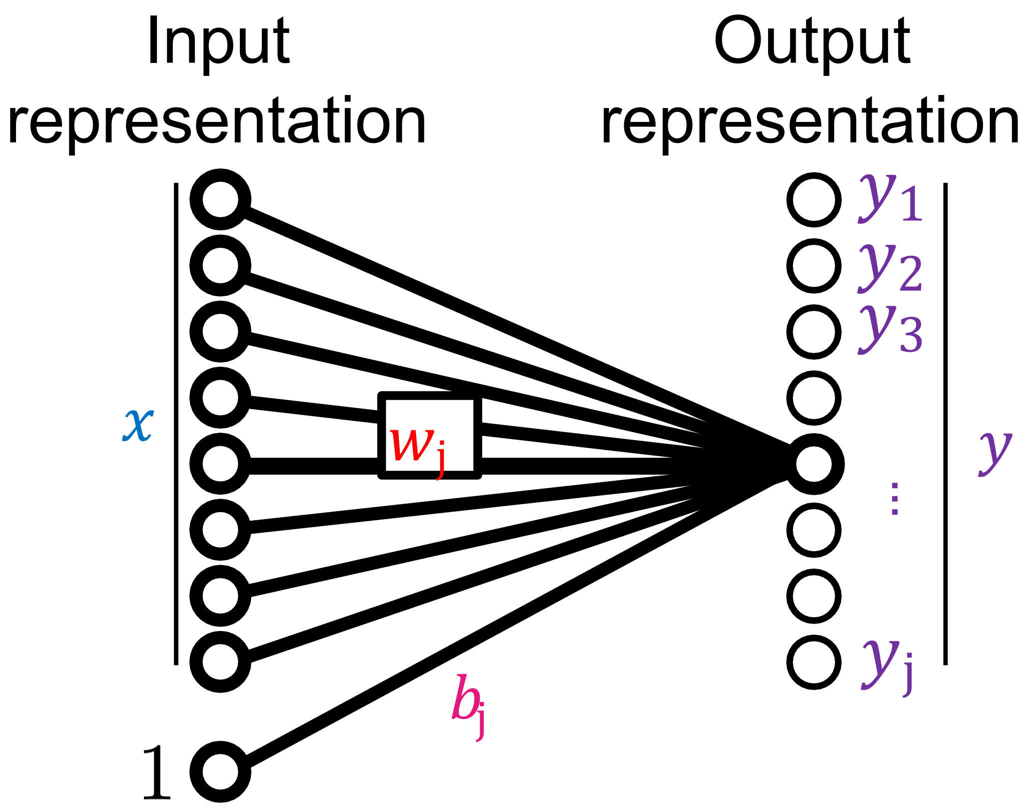
$$\begin{bmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{j1} & \cdots & w_{jn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_j \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_j \end{bmatrix}$$

parameters of the model: $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{b}\}$

Adapted from: Isola, Torralba, Freeman

Computation in a neural net – Full Layer

Linear layer



Full layer

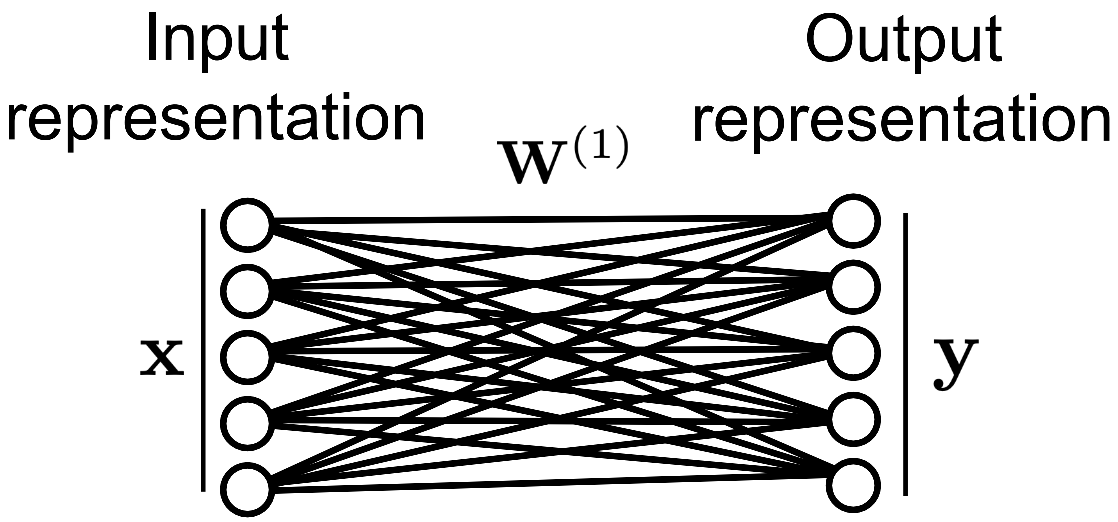
$$y = Wx + b$$

$$\begin{bmatrix} w_{11} & \cdots & w_{jn} & b_1 \\ \vdots & \ddots & \vdots & \vdots \\ w_{j1} & \cdots & w_{jn} & b_j \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \\ 1 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_j \end{bmatrix}$$

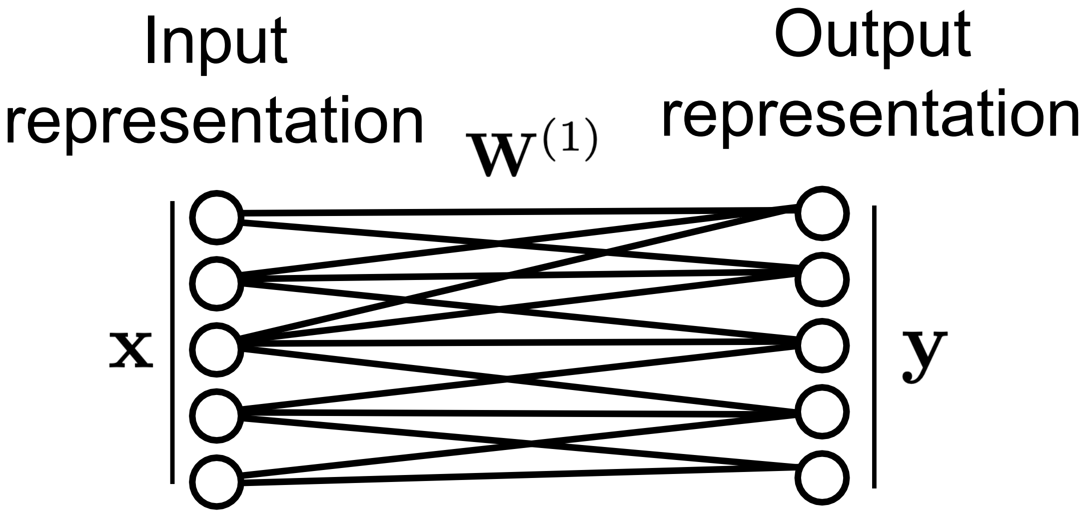
Can again simplify notation by appending a 1 to $\mathbf{x}$

Adapted from: Isola, Torralba, Freeman

Connectivity patterns



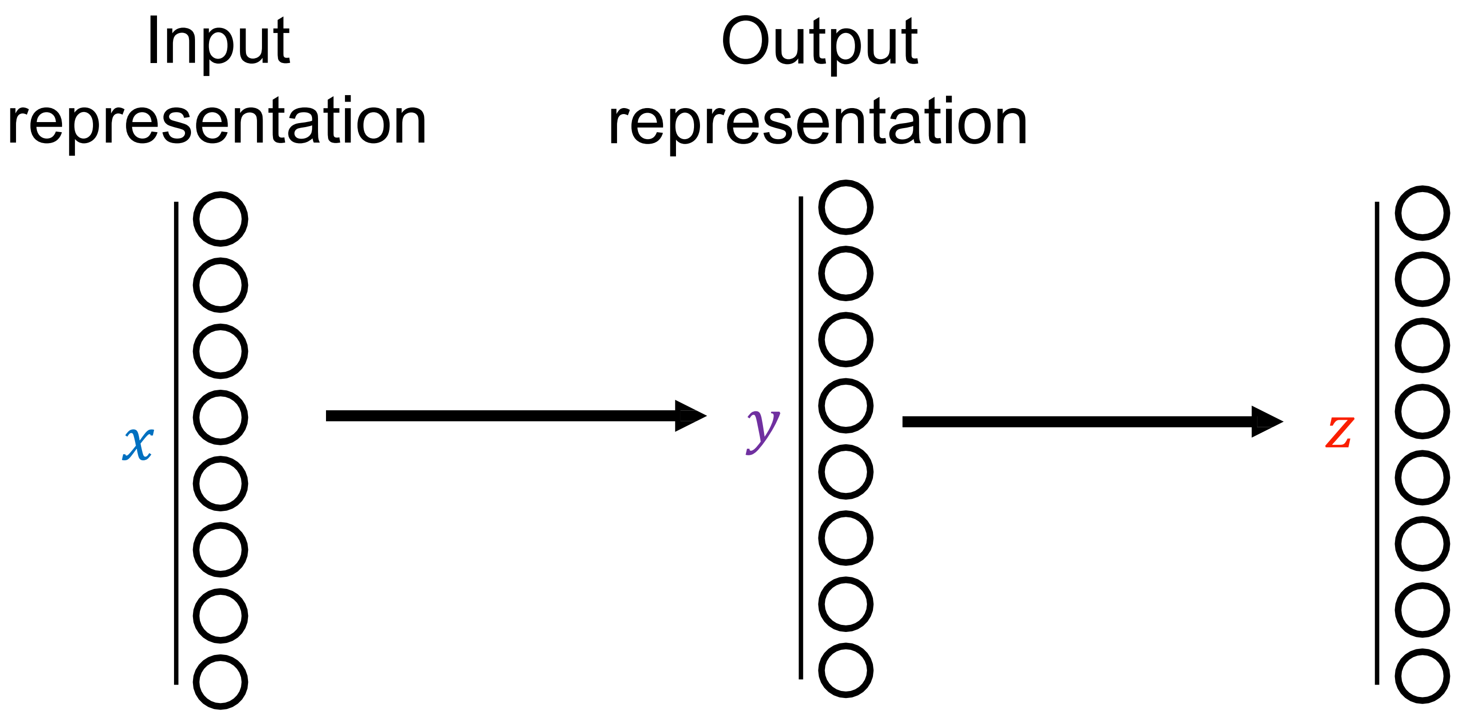
Fully connected layer



Locally connected layer
(Sparse W)

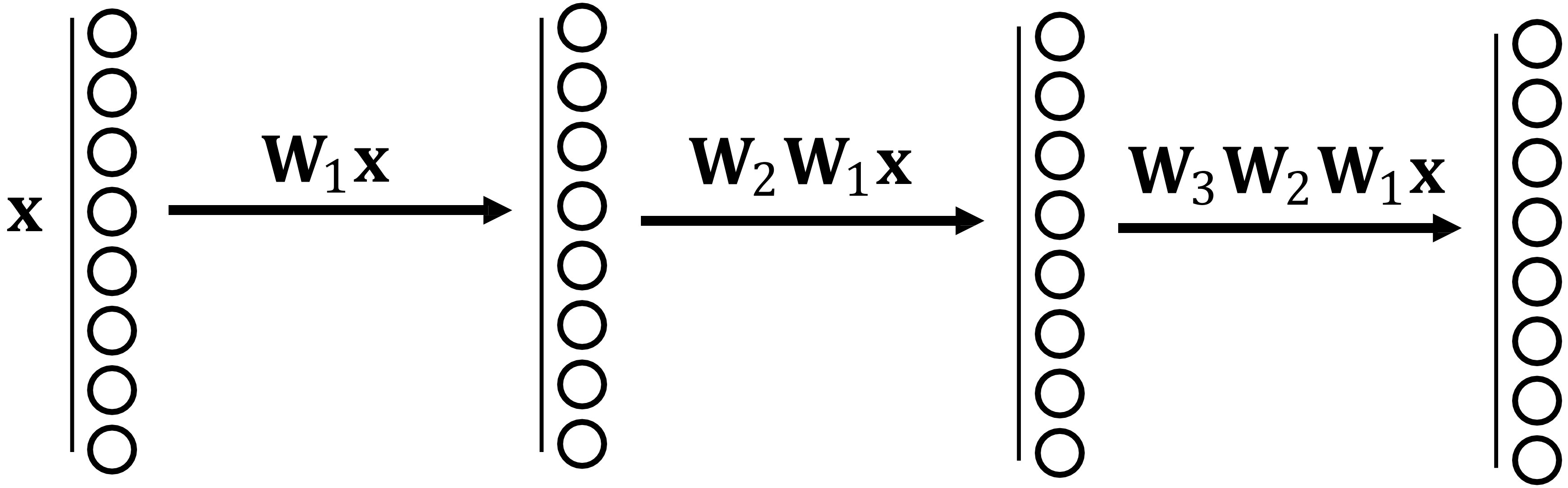
Computation in a neural network

We can now transform our input representation vector into some output representation vector using a bunch of linear combinations of the input:

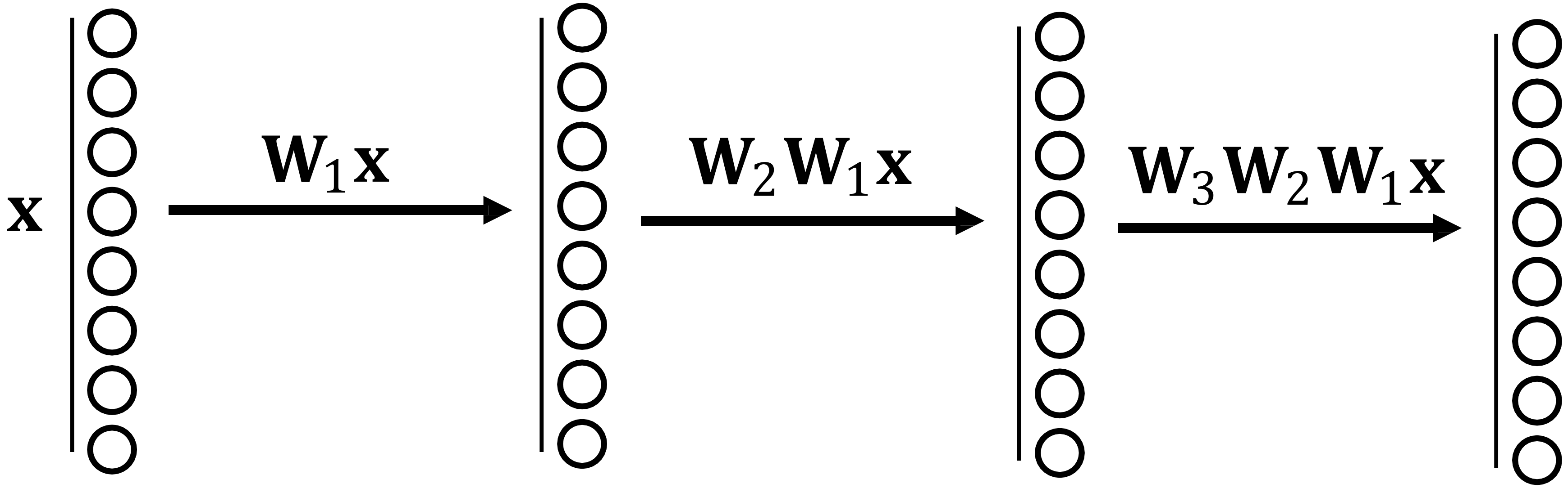


We can repeat this as many times as we want!

What is the problem with this idea?



What is the problem with this idea?

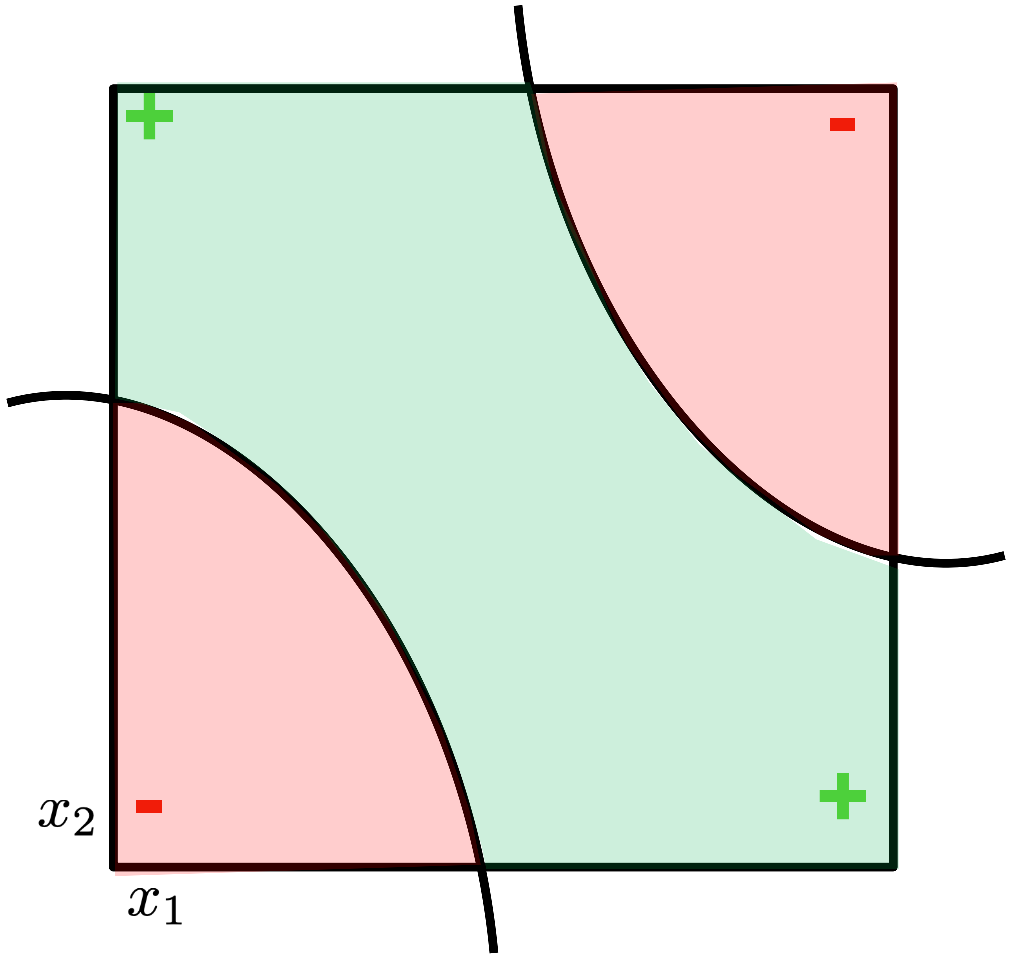


Can be expressed as single linear layer!

$$\widehat{W}x$$

Limited power: can't solve XOR ☹

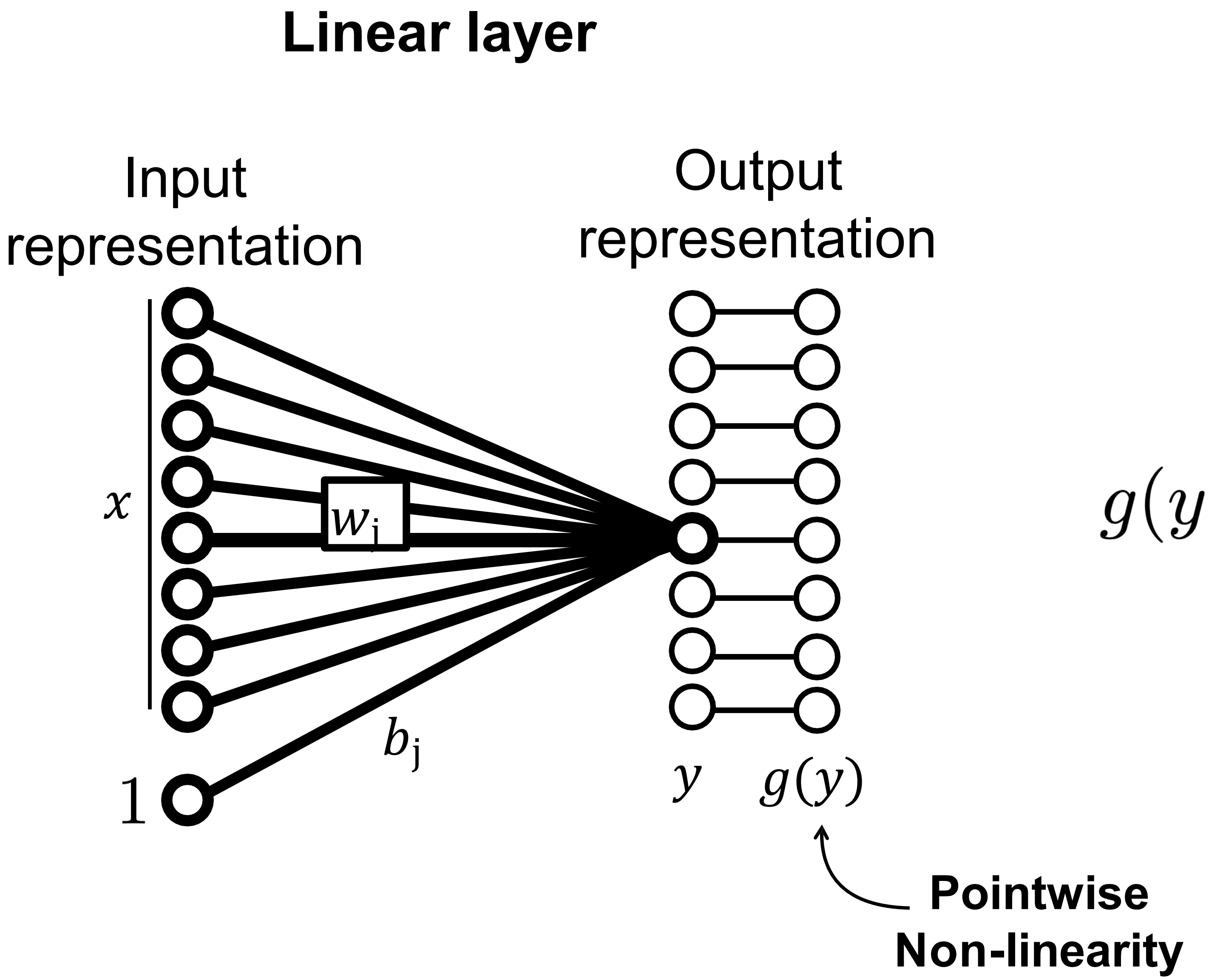
Recall
Goal: Non-linear decision boundary



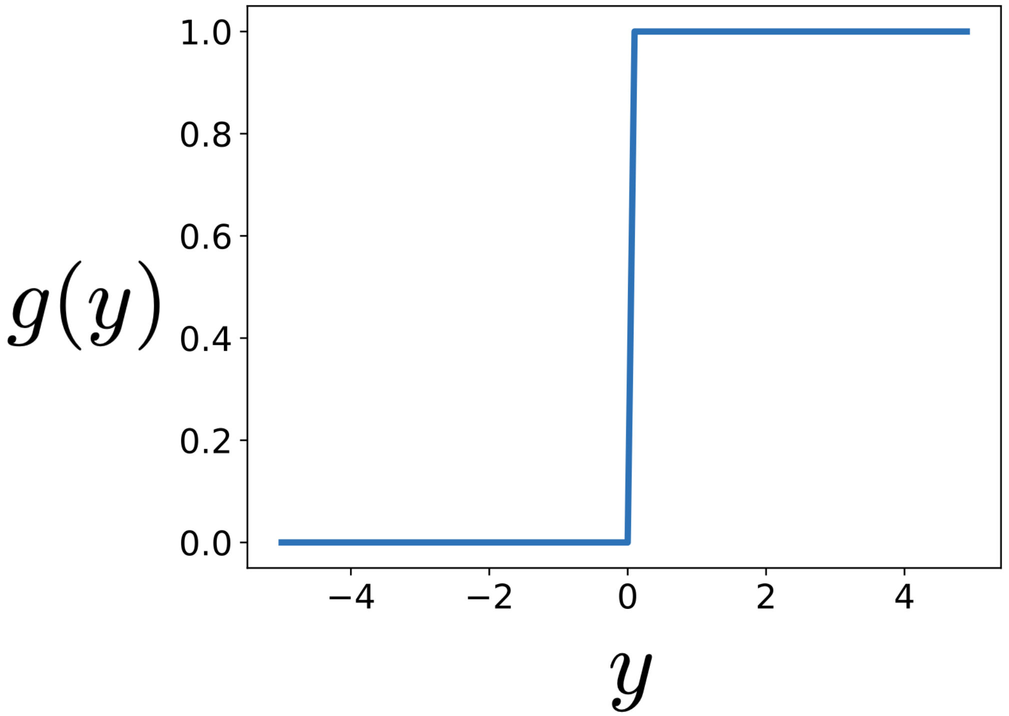
		x_2	
		0	1
x_1	0	0	1
	1	1	0

XOR

Solution: simple nonlinearity



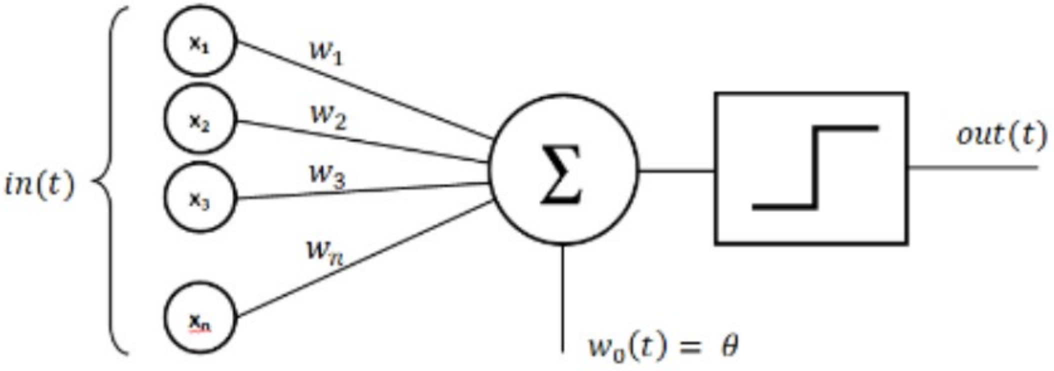
$$g(y) = \begin{cases} 1, & \text{if} \quad y > 0 \\ 0, & \text{otherwise} \end{cases}$$



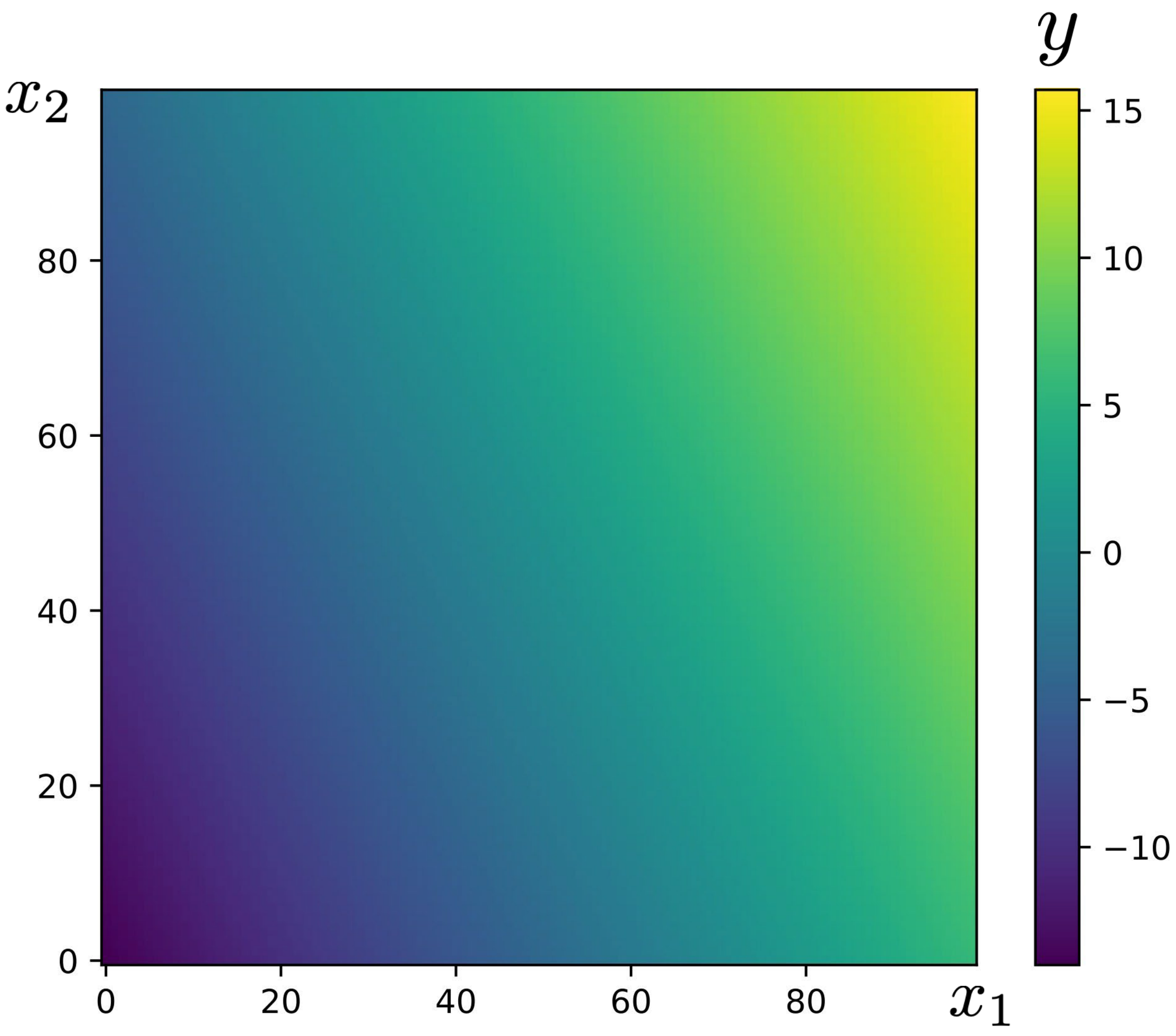
The Perceptron

WHAT PERCEPTRON SOUNDS LIKE

wHat pERceptRoNS ArE



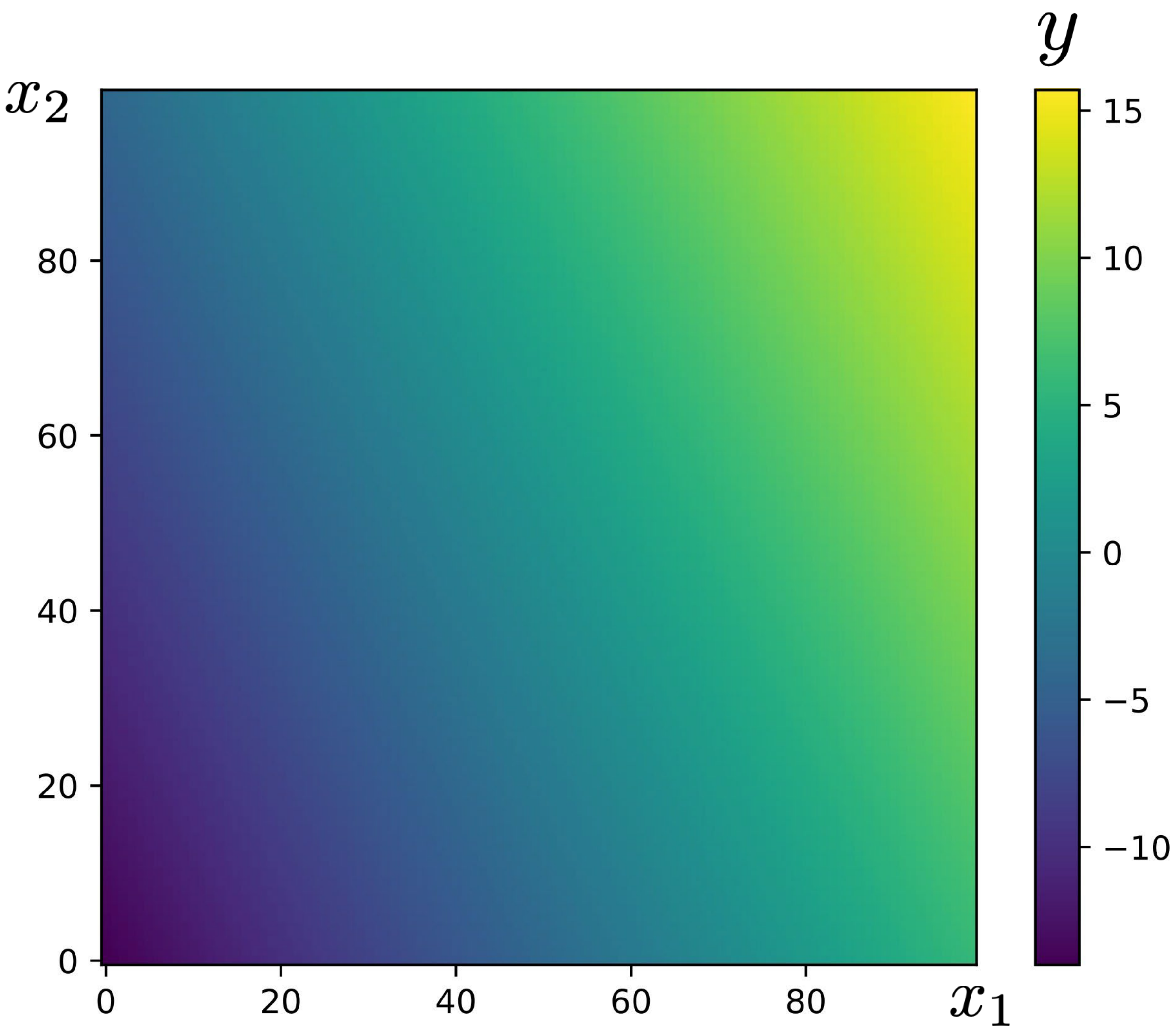
Example: linear classification with a perceptron



$$y = \mathbf{x}^T \mathbf{w} + b$$

Source: Isola, Torralba, Freeman

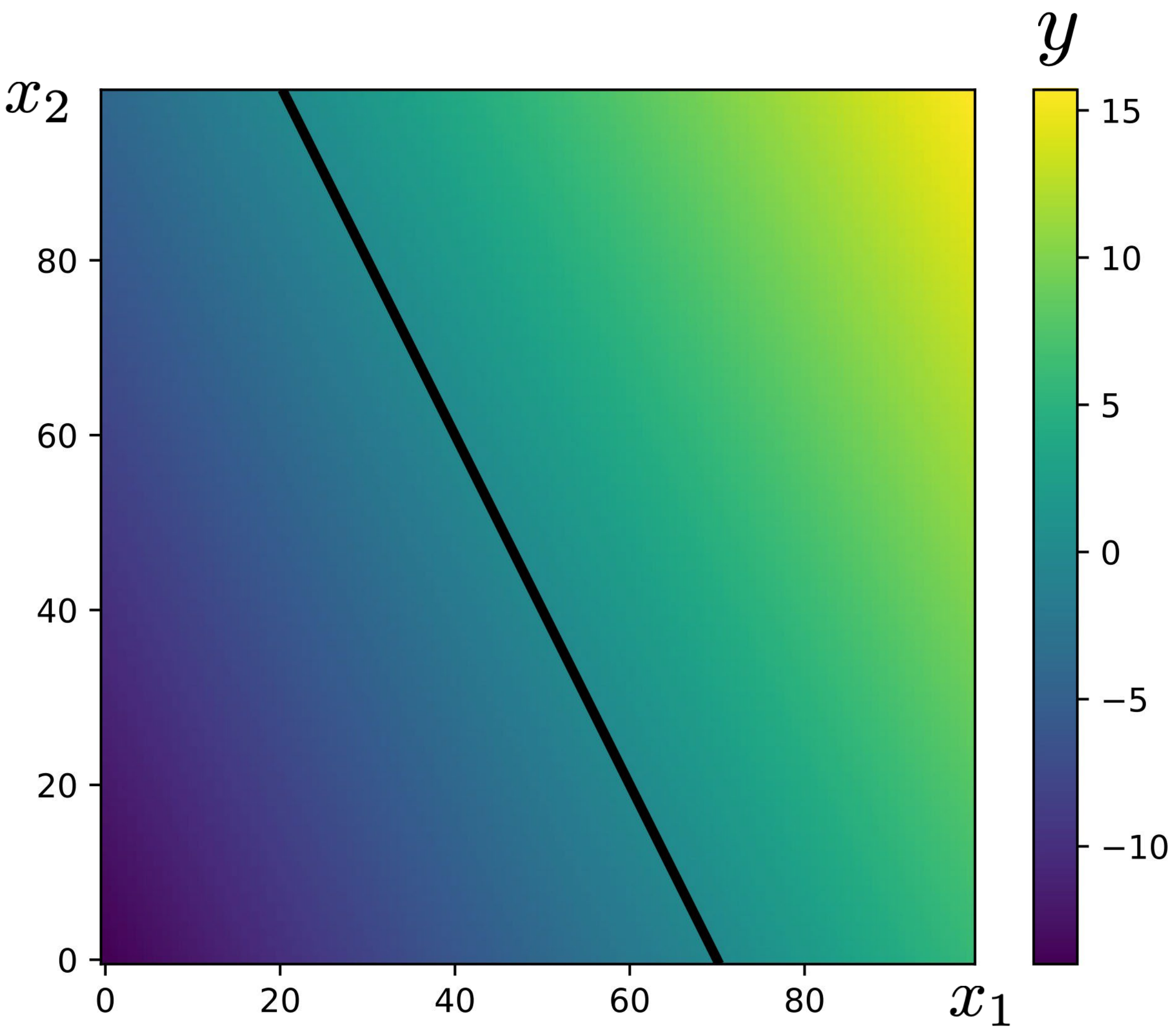
Example: linear classification with a perceptron



$$y = \mathbf{x}^T \mathbf{w} + b$$

$$g(y) = \begin{cases} 1, & \text{if} \quad y > 0 \\ 0, & \text{otherwise} \end{cases}$$

Example: linear classification with a perceptron

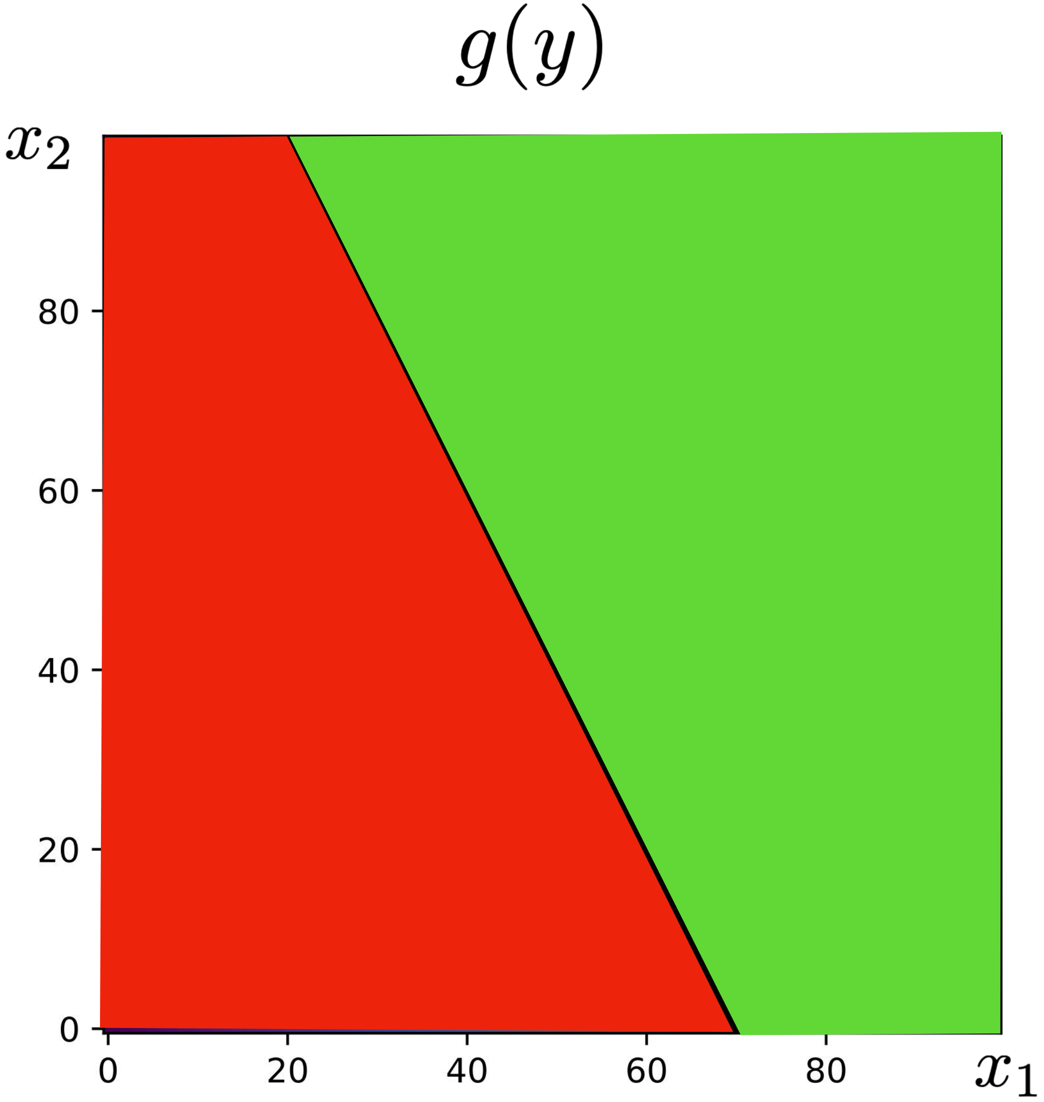


$$y = \mathbf{x}^T \mathbf{w} + b$$

$$g(y) = \begin{cases} 1, & \text{if} \quad y > 0 \\ 0, & \text{otherwise} \end{cases}$$

"when y is greater than 0, set all pixel values to 1 (green), otherwise, set all pixel values to 0 (red)"

Example: linear classification with a perceptron

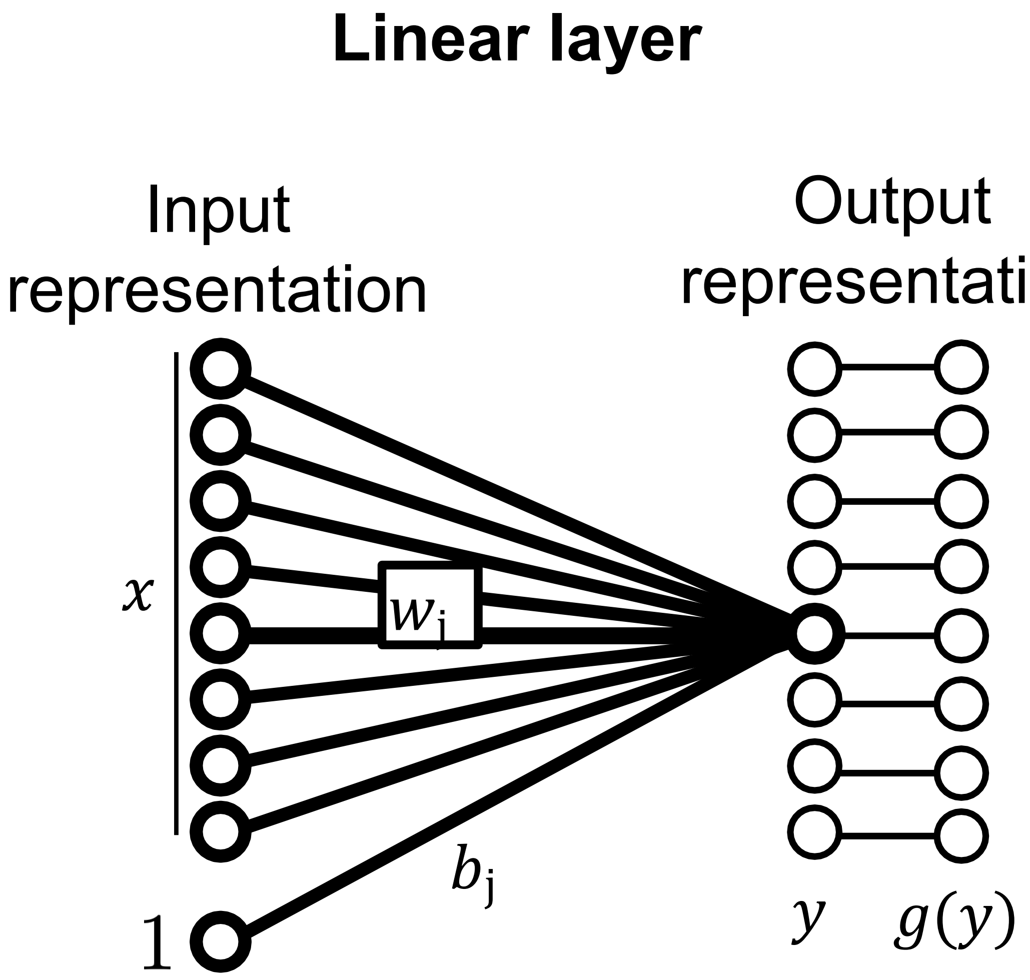


$$y = \mathbf{x}^T \mathbf{w} + b$$

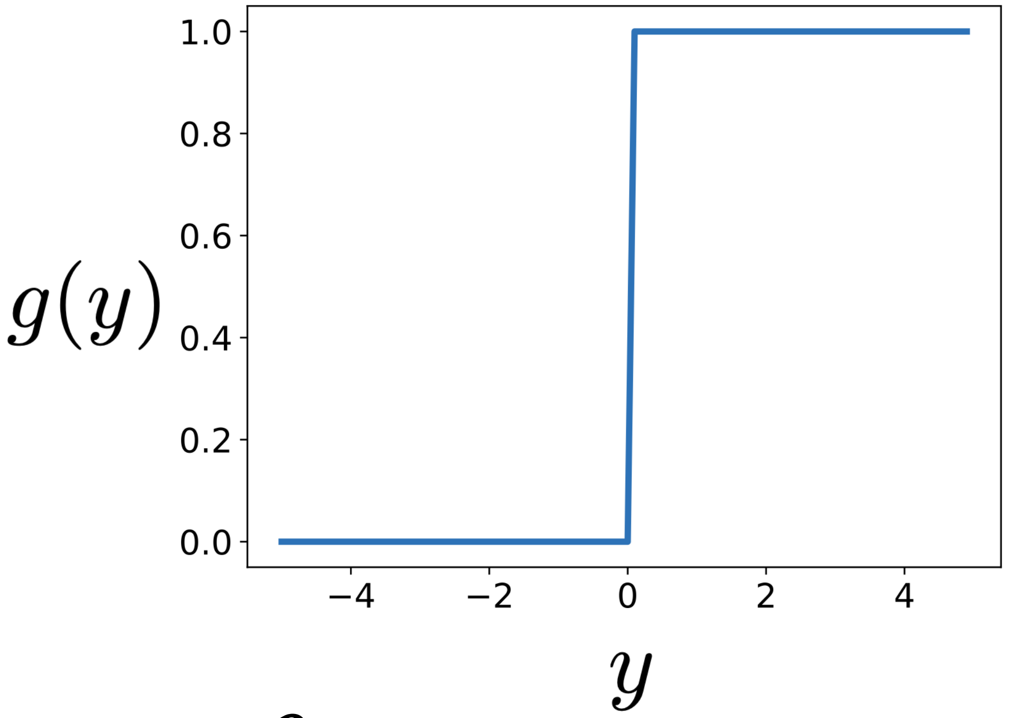
$$g(y) = \begin{cases} 1, & \text{if} \quad y > 0 \\ 0, & \text{otherwise} \end{cases}$$

"when y is greater than 0, set all pixel values to 1 (green), otherwise, set all pixel values to 0 (red)"

Computation in a neural net - nonlinearity



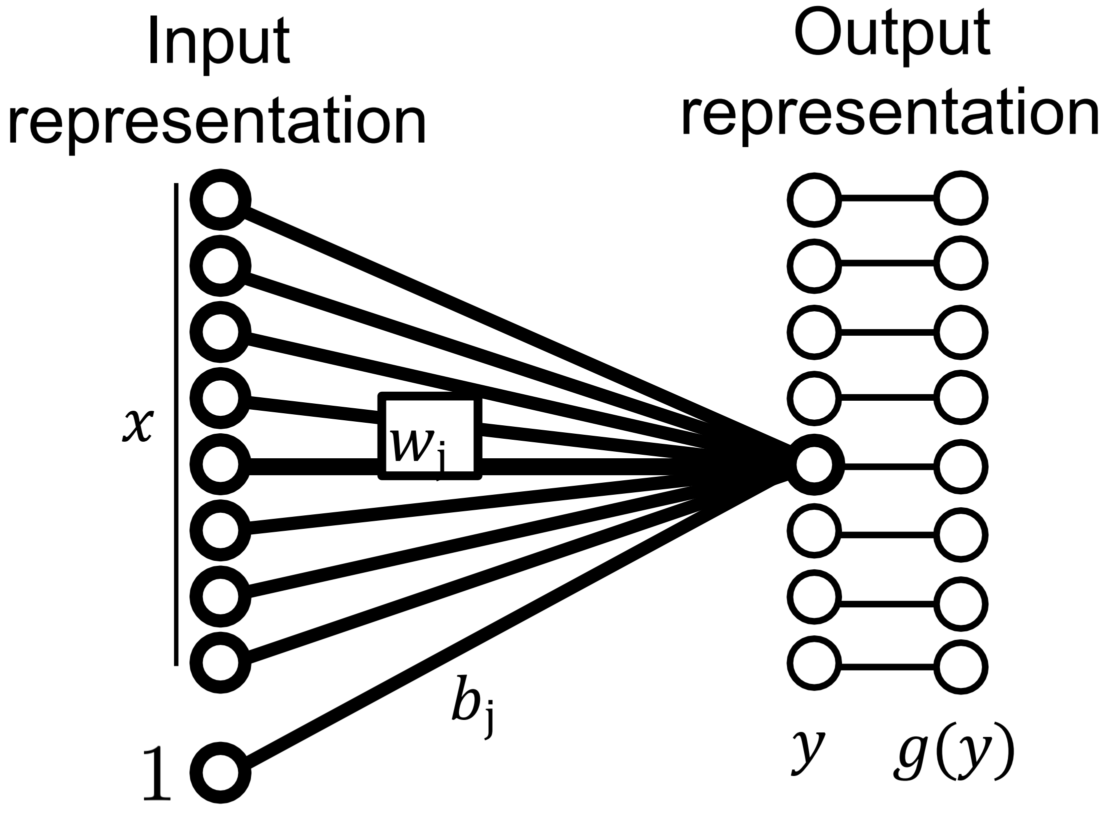
$$g(y) = \begin{cases} 1, & \text{if} \quad y > 0 \\ 0, & \text{otherwise} \end{cases}$$



Can't use gradient-based optimization, $\frac{\partial}{\partial y} g = 0$

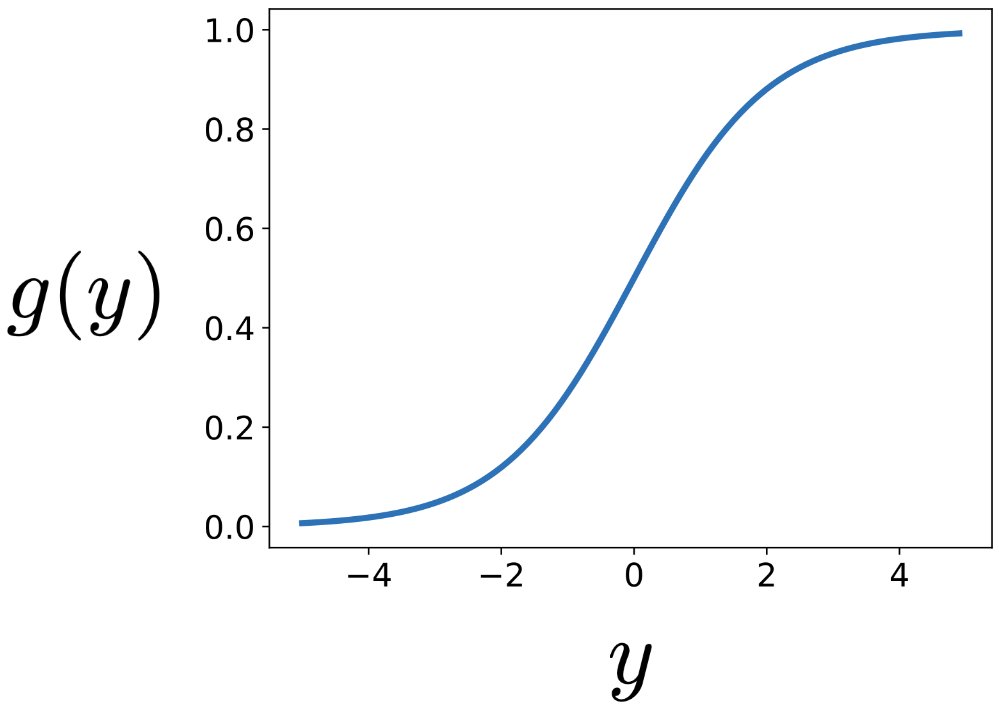
Computation in a neural net - nonlinearity

Linear layer



Sigmoid

$$g(y) = \sigma(y) = \frac{1}{1 + e^{-y}}$$

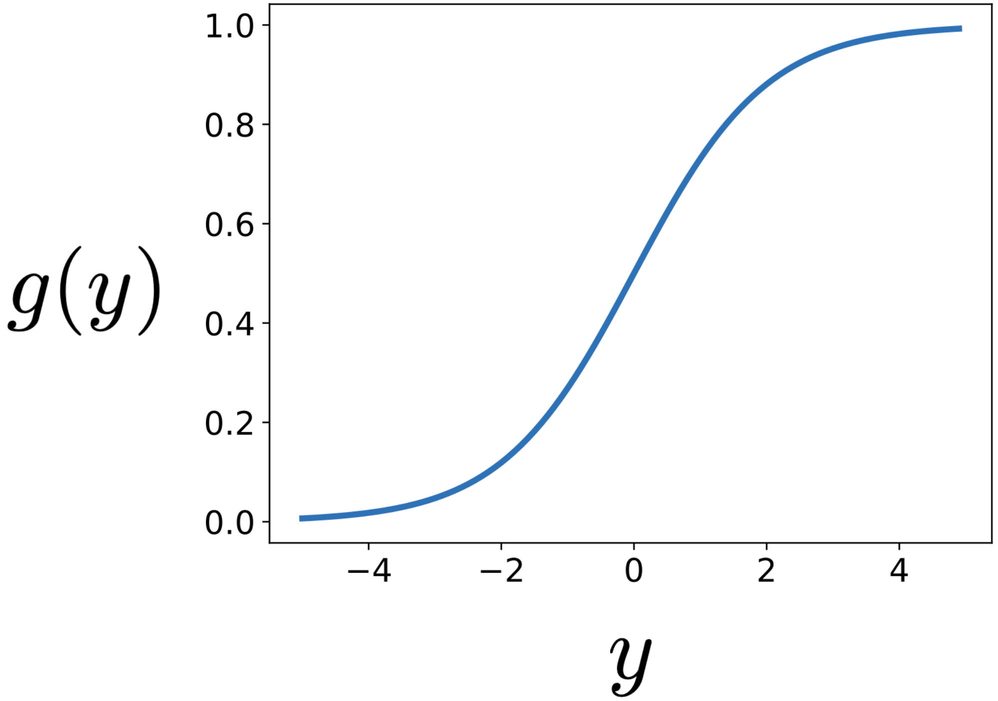


Computation in a neural net - nonlinearity

- Bounded between [0,1]
- Saturation for large +/- inputs
- Gradients go to zero

Sigmoid

$$g(y) = \sigma(y) = \frac{1}{1 + e^{-y}}$$

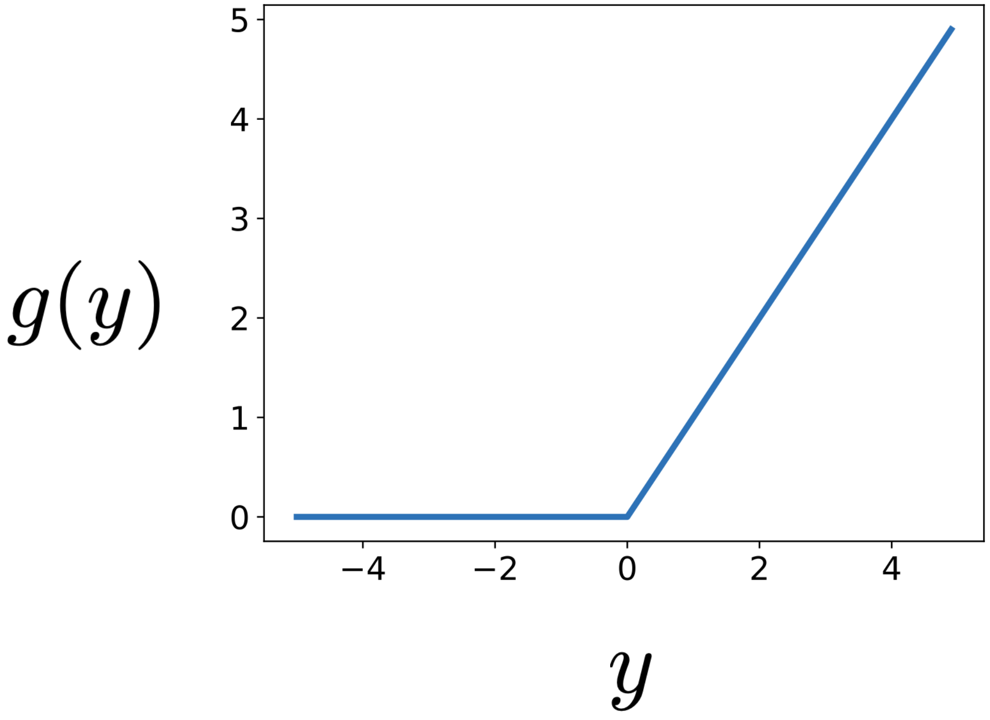


Computation in a neural net — nonlinearity

- Unbounded output (on positive side)
- Efficient to implement: $\frac{\partial g}{\partial y} = \begin{cases} 0, & \text{if} \quad y < 0 \\ 1, & \text{if} \quad y \geq 0 \end{cases}$
- Also seems to help convergence (6x speedup vs. tanh in [Krizhevsky et al. 2012])
- Drawback: if strongly in negative region, unit is dead forever (no gradient).
- Default choice: widely used in current models!

Rectified linear unit (ReLU)

$$g(y) = \max(0, y)$$



Computation in a neural net — nonlinearity

- where α is small (e.g., 0.02)

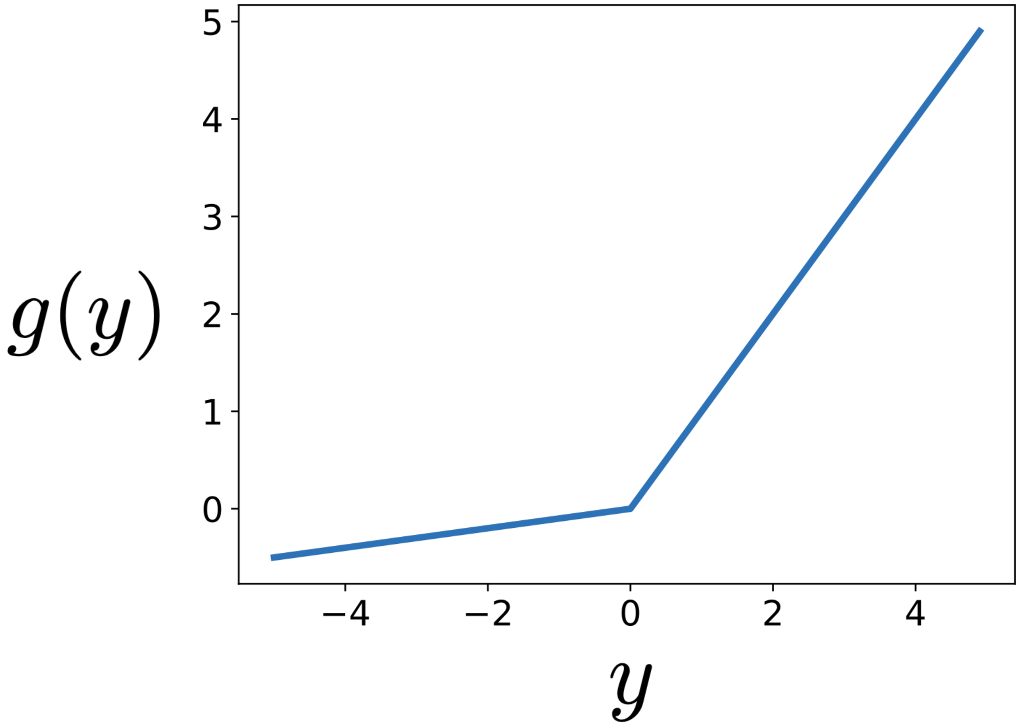
- Efficient to implement:

- Has non-zero gradients everywhere (unlike ReLU)

$$\frac{\partial g}{\partial y} = \begin{cases} -a, & \text{if} \quad y < 0 \\ 1, & \text{if} \quad y \geq 0 \end{cases}$$

Leaky ReLU

$$g(y) = \begin{cases} \max(0, y), & \text{if} \quad y \geq 0 \\ a \min(0, y), & \text{if} \quad y < 0 \end{cases}$$



Perceptron: Old Idea!

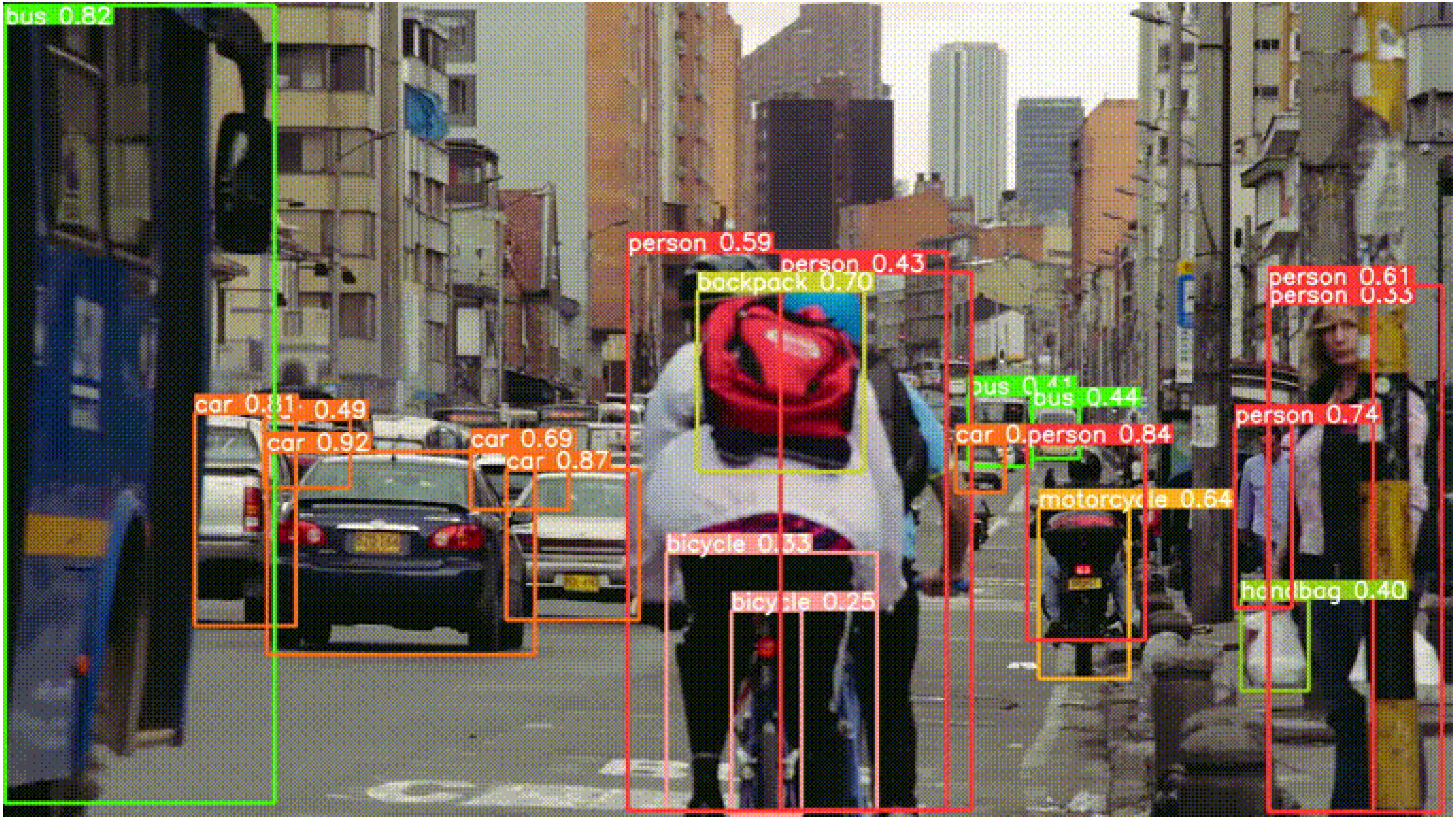
Late 1950s video on Rosenblatt’s perceptron research

“While promising, this approach to machine intelligence virtually died out …”

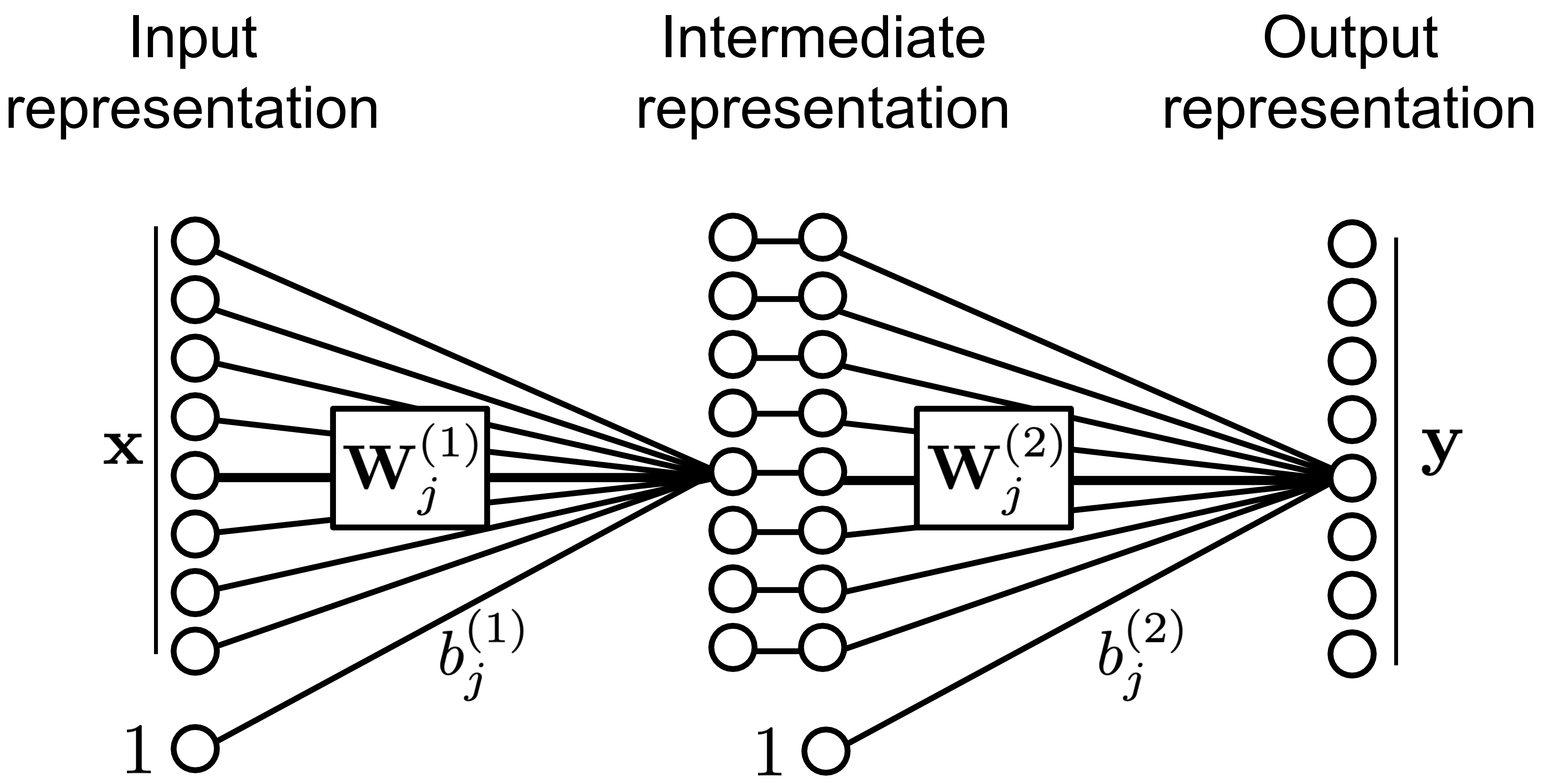
Perceptron: Old Idea!

Late 1950s video on Rosenblatt's perceptron research

"While promising, this approach to machine intelligence virtually died out …"


bus 0.82
car 0.81
0.49
car 0.92
car 0.69
car 0.87
person 0.59
person 0.43
backpack 0.70
bicycle 0.33
bicycle 0.25
bus 0.44
person 0.84
motorcycle 0.64
person 0.61
person 0.33
person 0.74
handbag 0.40

Stacking layers



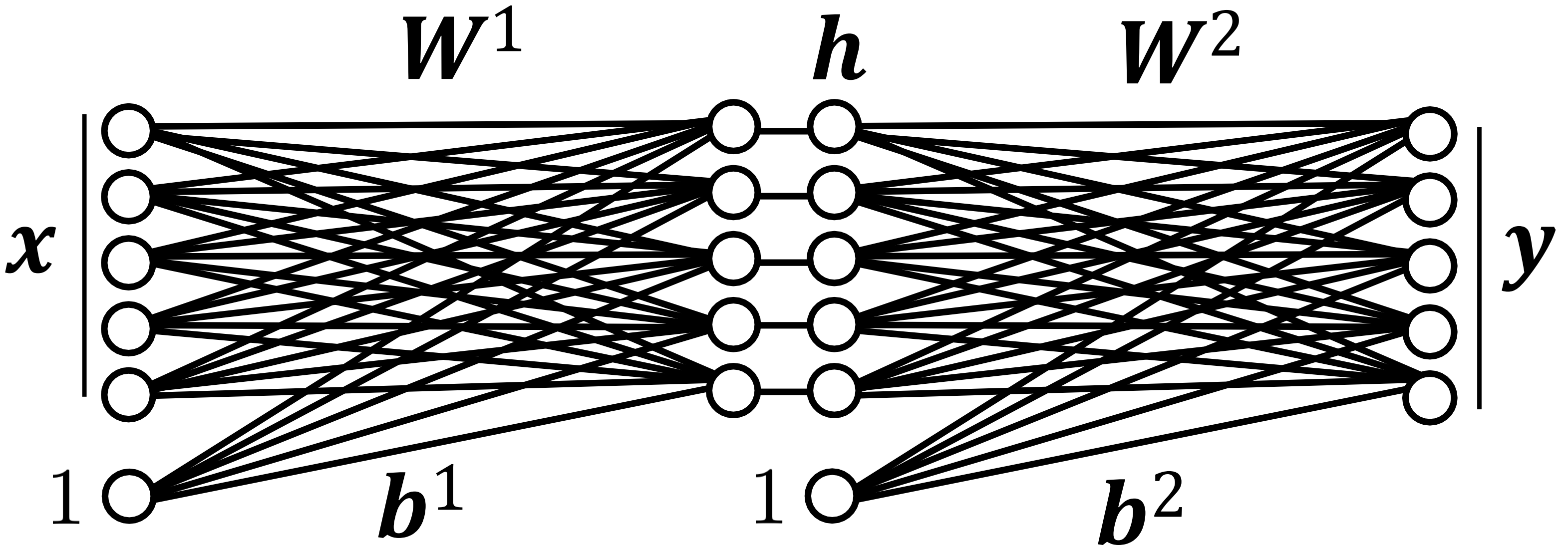
$\mathbf{h}$ = “hidden units”

Stacking layers

Input representation

Intermediate representation

Output representation



$$\boldsymbol{h} = g(\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1) \quad \boldsymbol{y} = g(\boldsymbol{W}^2\boldsymbol{h} + \boldsymbol{b}^2)$$

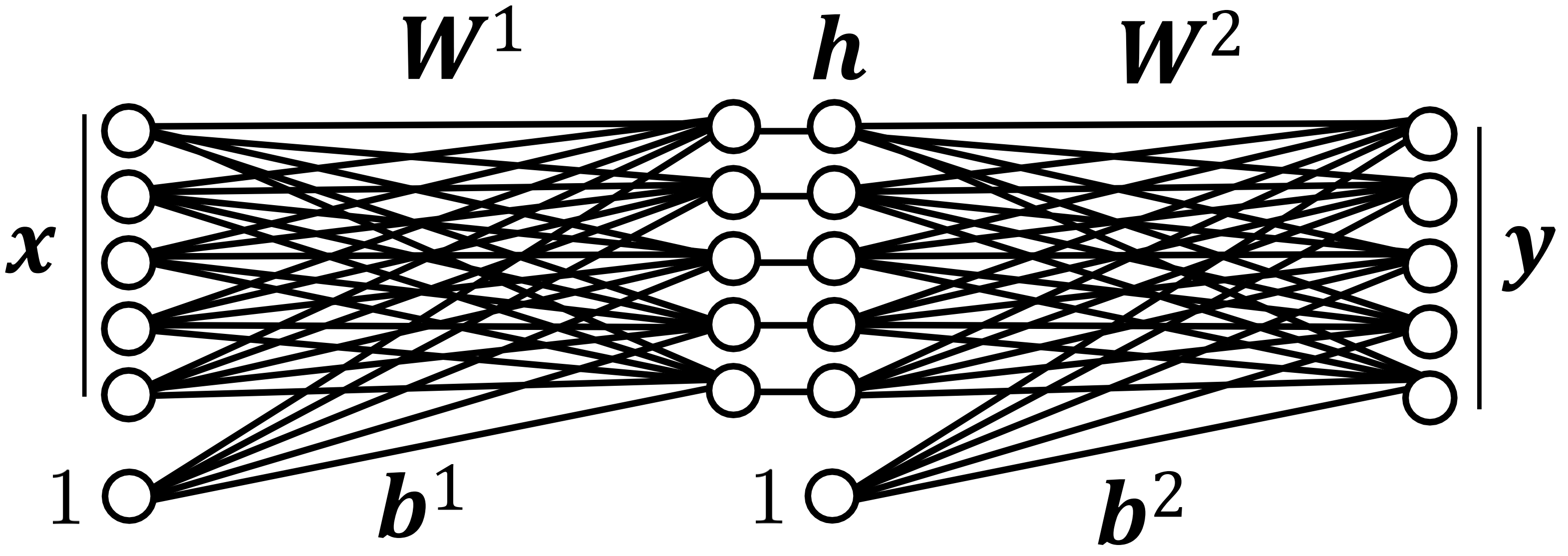
ReLU

$$\theta = \{\boldsymbol{W}^1, \ldots, \boldsymbol{W}^L, \boldsymbol{b}^1, \ldots, \boldsymbol{b}^L\}$$

Source: Isola, Torralba, Freeman

Stacking layers

Input representation | Intermediate representation | Output representation



positive

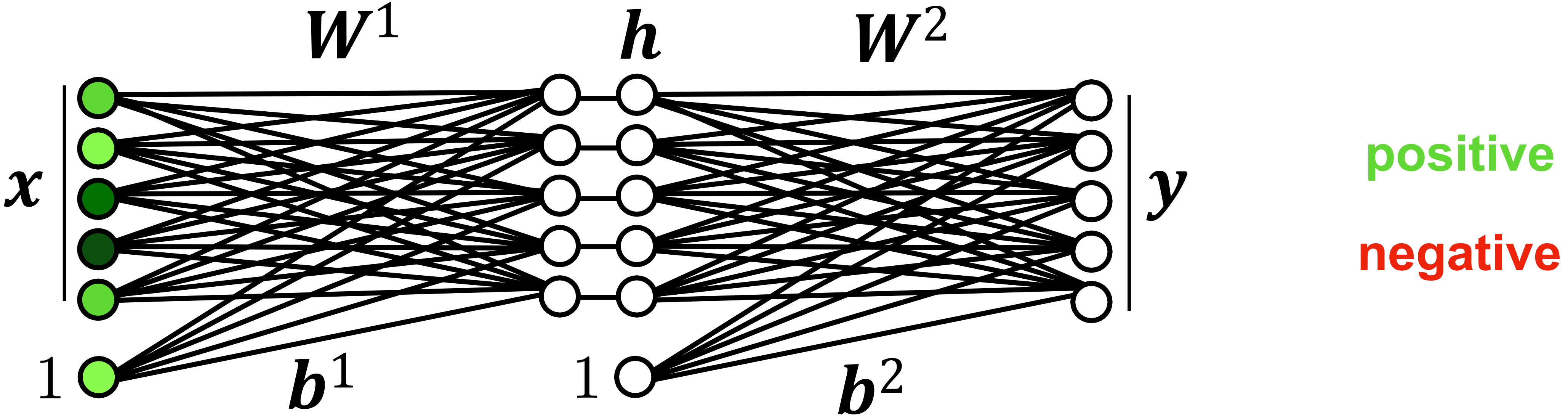
negative

$$\boldsymbol{h} = g(\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1) \qquad \boldsymbol{y} = g(\boldsymbol{W}^2\boldsymbol{h} + \boldsymbol{b}^2)$$

ReLU

$$\theta = \{\boldsymbol{W}^1, \ldots, \boldsymbol{W}^L, \boldsymbol{b}^1, \ldots, \boldsymbol{b}^L\}$$

Stacking layers



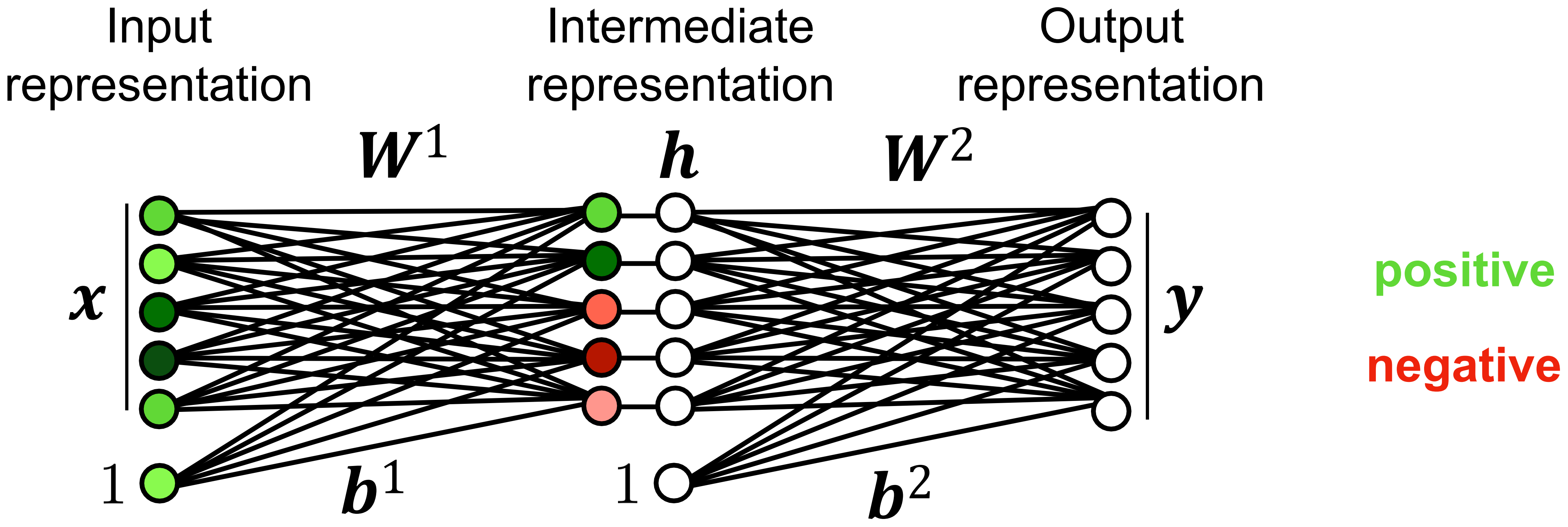
$$\boldsymbol{h} = g(\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1) \quad \boldsymbol{y} = g(\boldsymbol{W}^2\boldsymbol{h} + \boldsymbol{b}^2)$$

ReLU

$$\theta = \{\boldsymbol{W}^1, \dots, \boldsymbol{W}^L, \boldsymbol{b}^1, \dots, \boldsymbol{b}^L\}$$

Source: Isola, Torralba, Freeman

Stacking layers

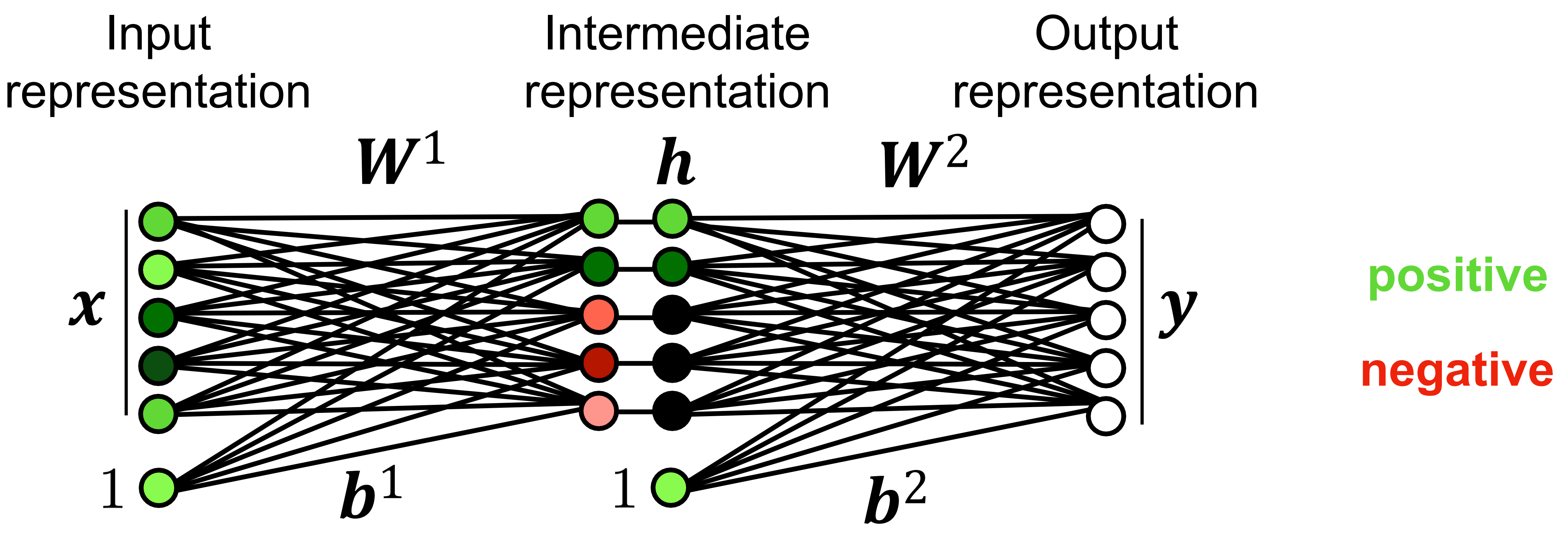


$$\boldsymbol{h} = g(\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1) \quad \boldsymbol{y} = g(\boldsymbol{W}^2\boldsymbol{h} + \boldsymbol{b}^2)$$

ReLU

$$\theta = \{\boldsymbol{W}^1, \dots, \boldsymbol{W}^L, \boldsymbol{b}^1, \dots, \boldsymbol{b}^L\}$$

Stacking layers

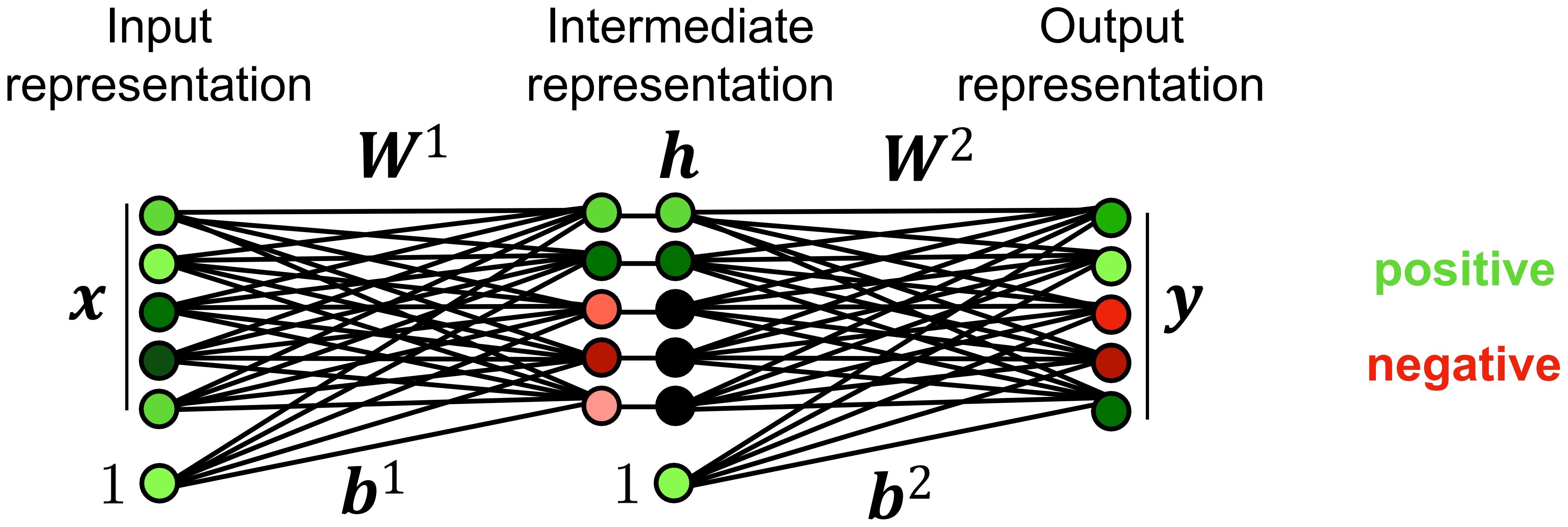


$$\boldsymbol{h} = g(\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1) \qquad \boldsymbol{y} = g(\boldsymbol{W}^2\boldsymbol{h} + \boldsymbol{b}^2)$$

ReLU

$$\theta = \{\boldsymbol{W}^1, \ldots, \boldsymbol{W}^L, \boldsymbol{b}^1, \ldots, \boldsymbol{b}^L\}$$

Stacking layers

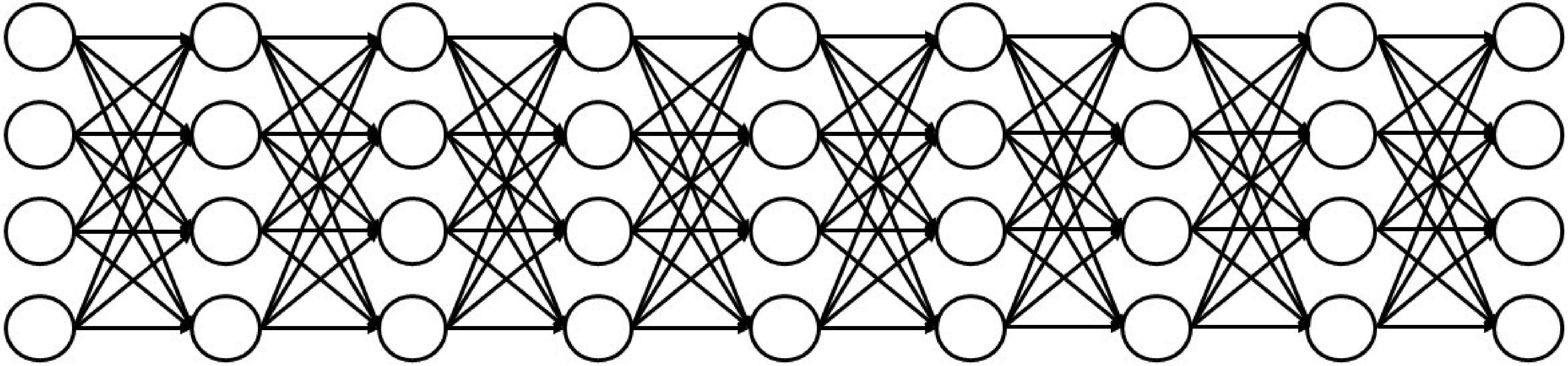


$$\boldsymbol{h} = g(\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1) \qquad \boldsymbol{y} = g(\boldsymbol{W}^2\boldsymbol{h} + \boldsymbol{b}^2)$$

ReLU

$$\theta = \{\boldsymbol{W}^1, \dots, \boldsymbol{W}^L, \boldsymbol{b}^1, \dots, \boldsymbol{b}^L\}$$

DEEP Neural Nets?

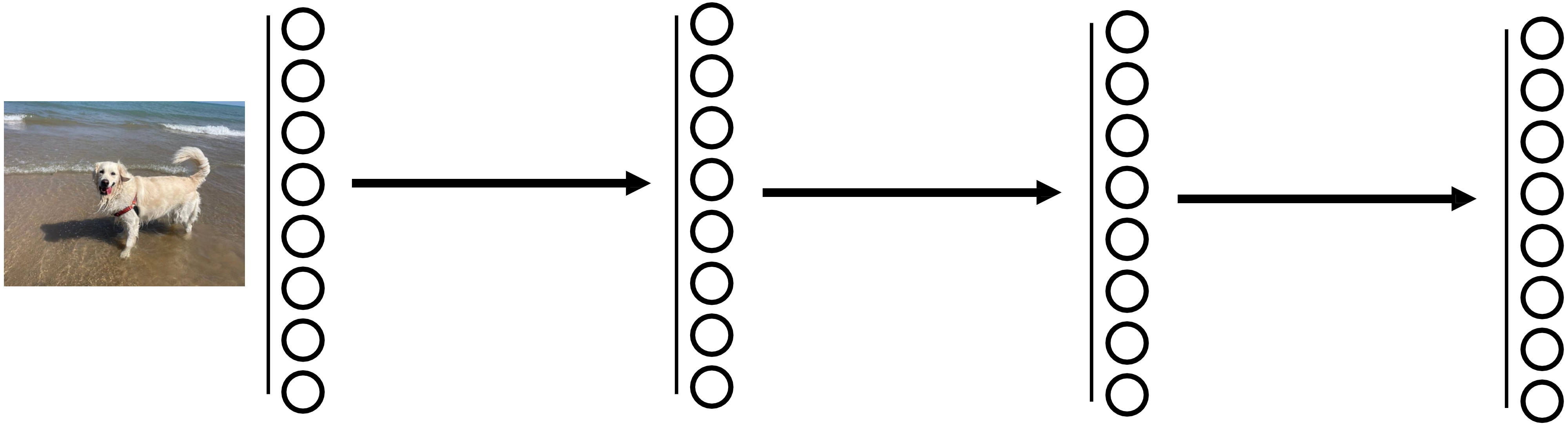


Stacking layers - What's actually happening?

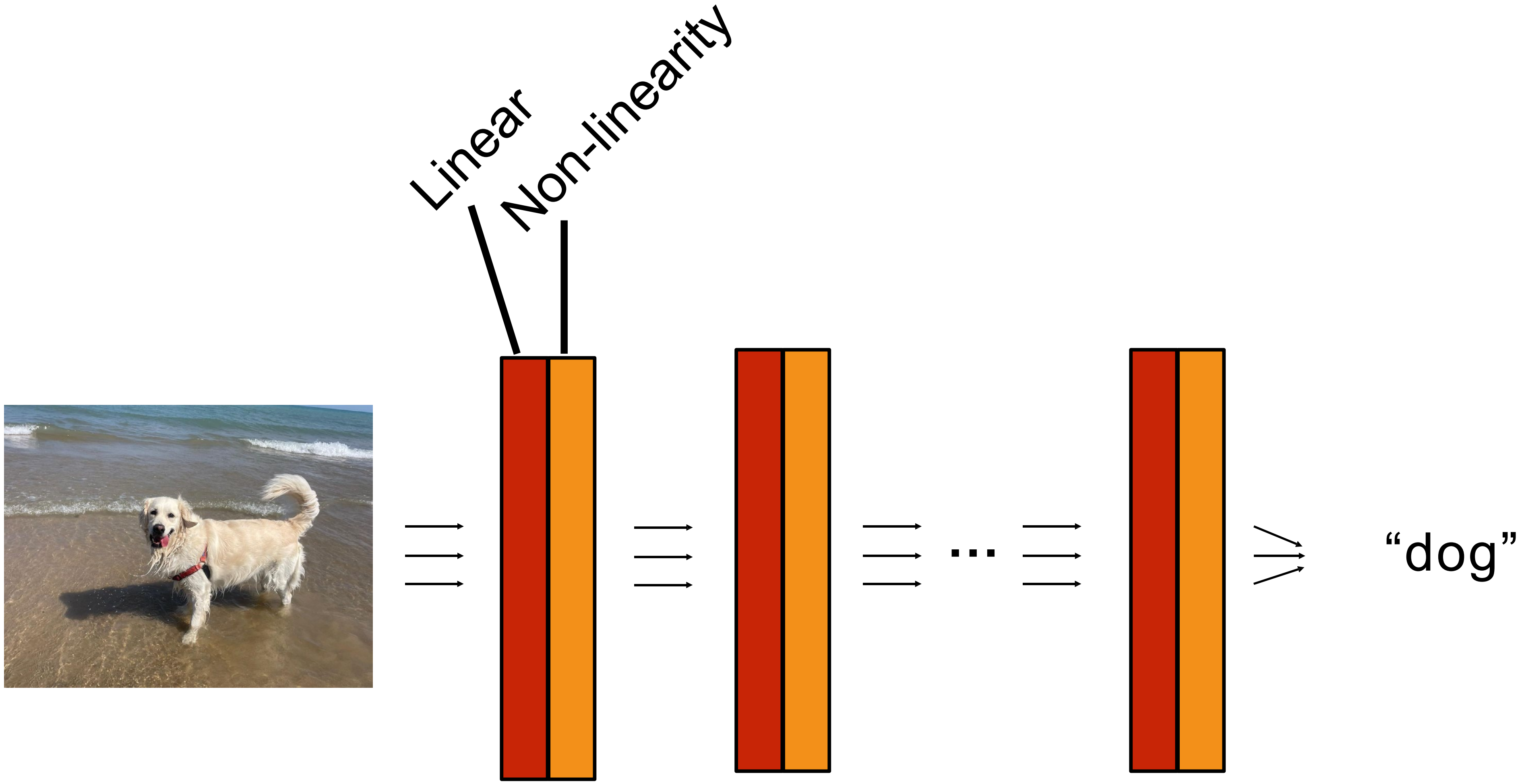
Low level features:
e.g., edge, texture, …

higher level features:
e.g., shape

even higher level features:
e.g., "paw", "fur"



Deep nets



$$f(x) = f_L(\ldots f_3(f_2(f_1(x))))$$

Source: Isola, Torralba, Freeman

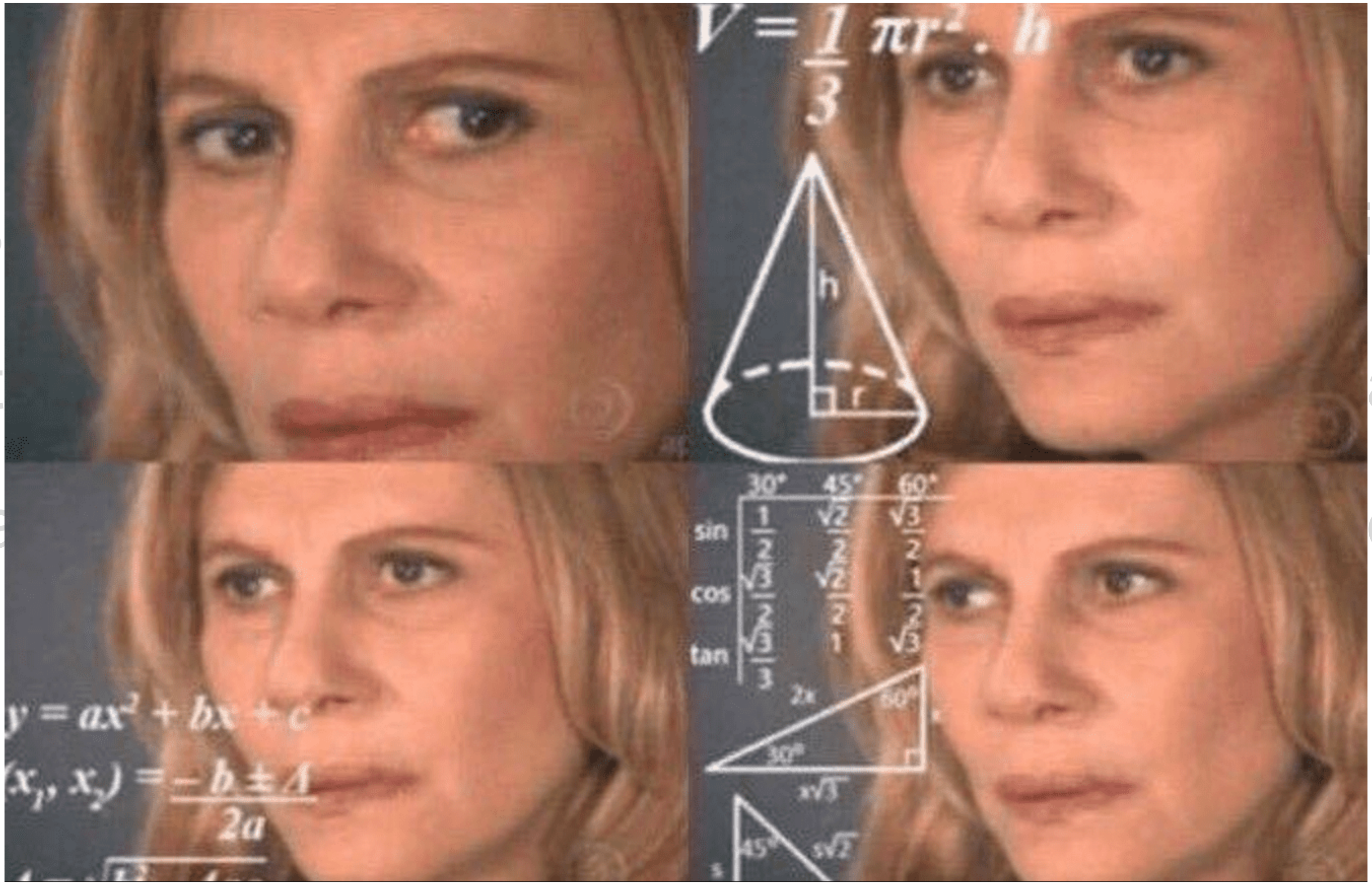
Computation has a simple form

- Composition of linear functions with nonlinearities in between
- E.g. matrix multiplications with ReLU, $max(0, \mathbf{x})$ afterwards
- Do a matrix multiplication, set all negative values to 0, repeat

But where do we get the weights from?

Computation has a simple form

- Compos… in between
- E.g. mat… afterwards
- Do a ma… o 0, repeat



But where do we get the weights from?

Where do we get the weights from ?

with nonlinearities in between

ReLU, $max(0, \mathbf{x})$ afterwards

all negative values to 0, repeat

How would we learn the parameters?

$\mathbf{y}_1$

“dog”

$\mathbf{x}_1$

$\mathcal{L}(f_\theta(\mathbf{x}_1), \mathbf{y}_1)$

predicted

ground truth

Learned

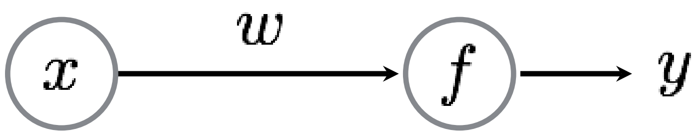
$\theta_1 \quad \theta_2 \quad \theta_3 \quad \theta_4 \quad \theta_5 \quad \theta_6$

$$\theta^* = \arg\min_\theta \sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i)$$

Training neural networks

Let’s start easy

world's smallest neural network!
(a "perceptron")



$$y = wx$$

(a.k.a. line equation, linear regression)

Training a Neural Network

Given a set of samples and a Perceptron

$$\{x_i, y_i\}$$

$$y = f_{\text{PER}}(x; w)$$

Estimate the parameter of the Perceptron

$$w$$

Given training data:

x	y
10	10.1
2	1.9
3.5	3.4
1	1.1

What do you think the weight parameter is?

$$y = wx$$

Given training data:

x	y
10	10.1
2	1.9
3.5	3.4
1	1.1

What do you think the weight parameter is?

$$y = wx$$

not so obvious as the network gets more complicated so we use …

An Incremental Learning Strategy

(gradient descent)

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

and a perceptron

$$\hat{y} = wx$$

An Incremental Learning Strategy

(gradient descent)

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

and a perceptron

$$\hat{y} = wx$$

Modify weight w such that $\hat{y}$ gets **'closer'** to y

An Incremental Learning Strategy

(gradient descent)

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

and a perceptron

$$\hat{y} = wx$$

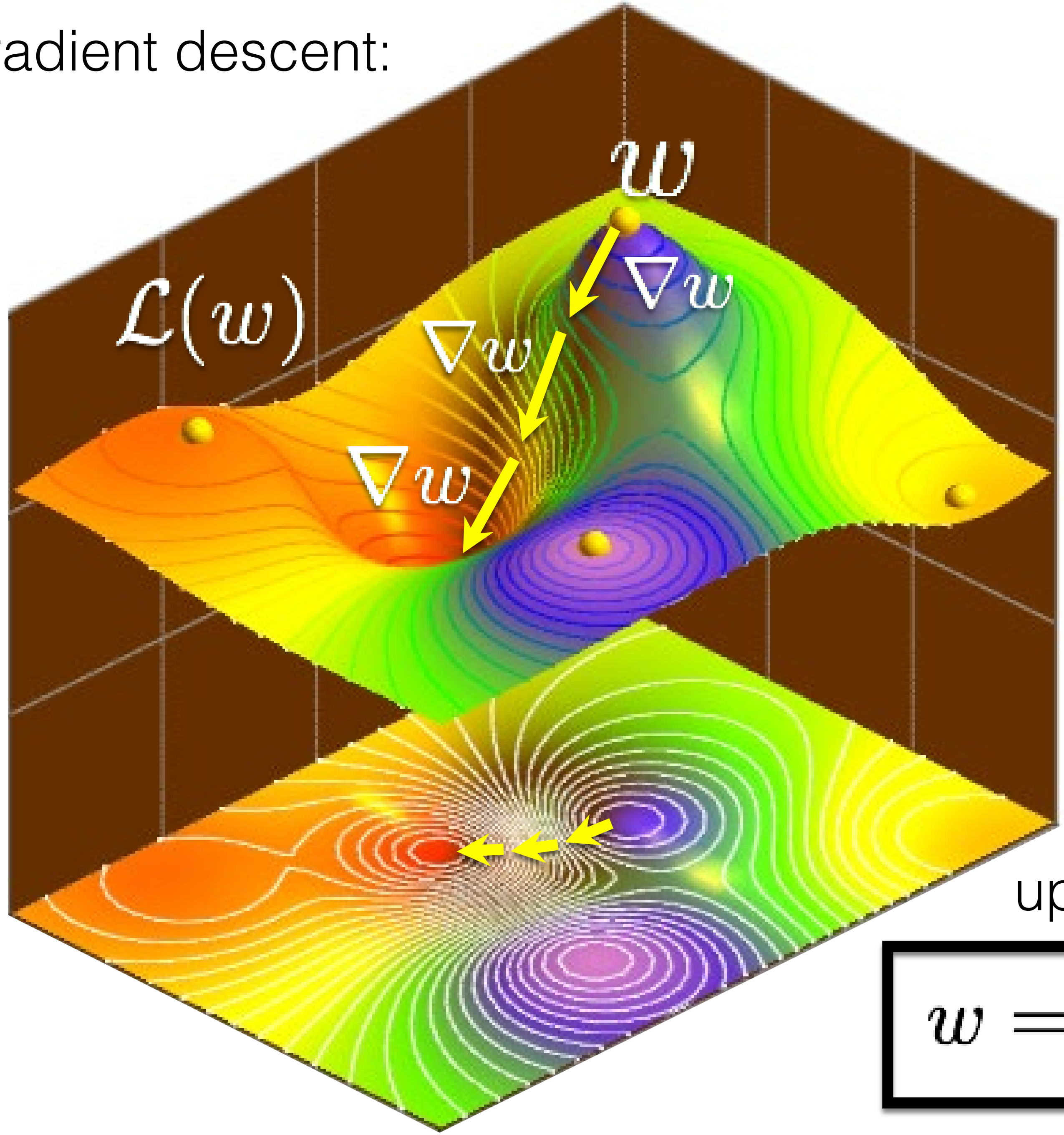
Modify weight w such that $\hat{y}$ gets **'closer'** to y

perceptron parameter

perceptron output

true label

Gradient descent:



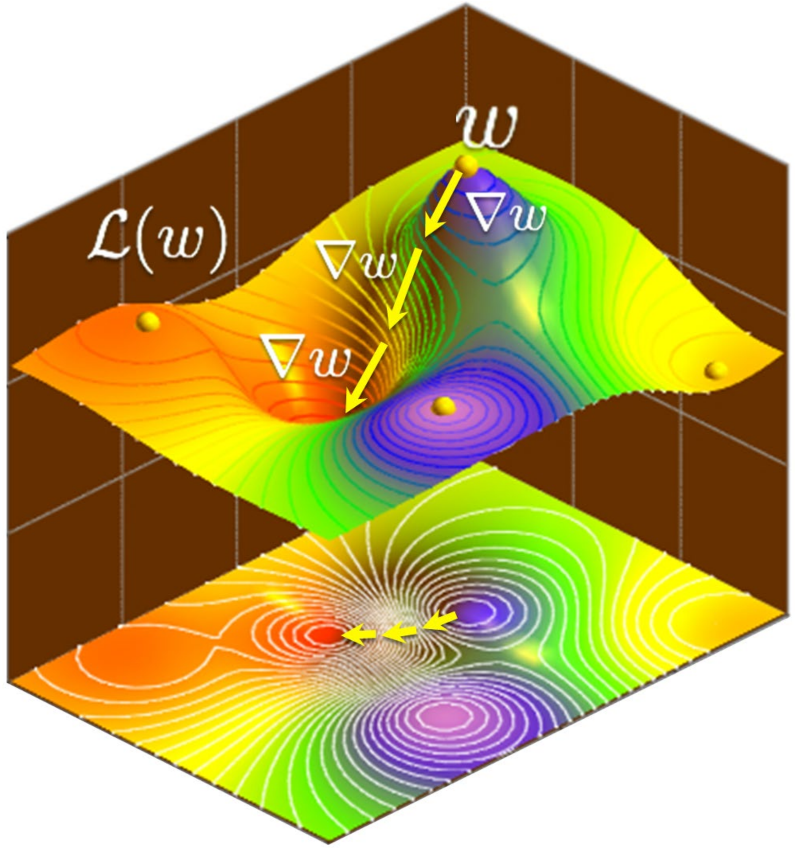
update rule:

$$\boxed{w = w - \nabla w}$$

$$\frac{d\mathcal{L}}{dw}$$ ...is the rate at which **this** will change...

$$\mathcal{L} = \frac{1}{2}(y - \hat{y})^2$$

the loss function



... per unit change of **this**

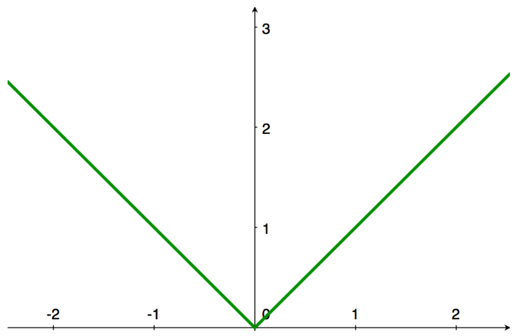
$$y = wx$$

the weight parameter

Let's compute the derivative...

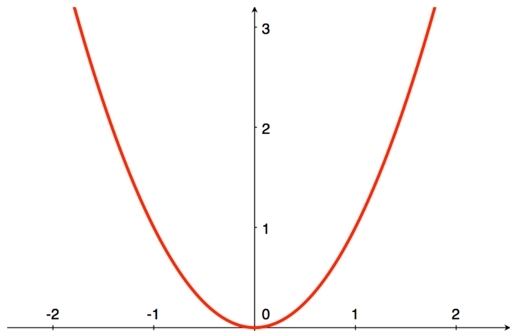
L1 Loss

$$\ell(\hat{y}, y) = |\hat{y} - y|$$



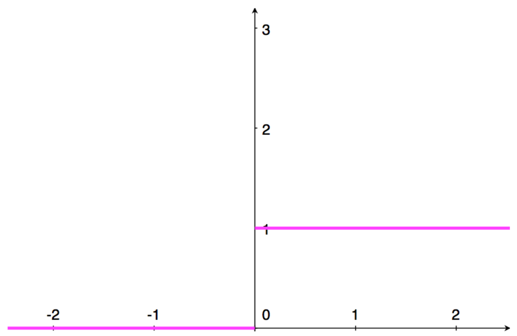
L2 Loss

$$\ell(\hat{y}, y) = (\hat{y} - y)^2$$



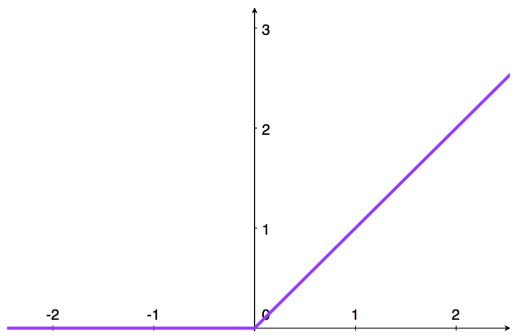
Zero-One Loss

$$\ell(\hat{y}, y) = \mathbf{1}[\hat{y} = y]$$



Hinge Loss

$$\ell(\hat{y}, y) = \max(0, 1 - y \cdot \hat{y})$$



$$\frac{d\mathcal{L}}{dw}$$ ...is the rate at which **this** will change...

$$\mathcal{L} = \frac{1}{2}(y - \hat{y})^2$$

the loss function

... per unit change of **this**

$$y = wx$$

the weight parameter

Let's compute the derivative...

Compute the derivative

$$\frac{d\mathcal{L}}{dw} = \frac{d}{dw}\left\{\frac{1}{2}(y-\hat{y})^2\right\}$$

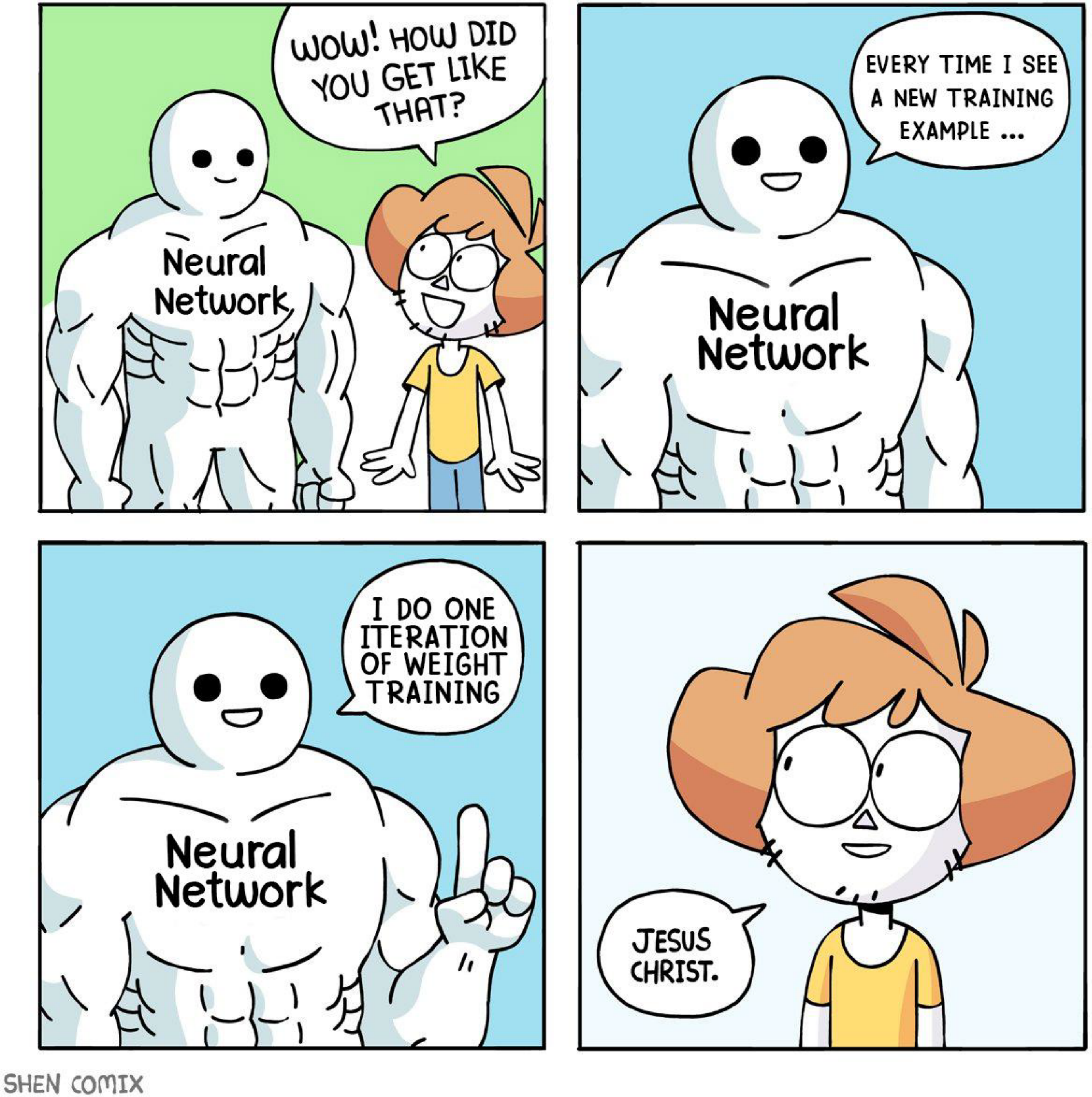
$$= -(y-\hat{y})\frac{dwx}{dw}$$

$$= -(y-\hat{y})x = \nabla w$$

That means the weight update for **gradient descent** is:

$$w = w - \nabla w$$ move in direction of negative gradient

$$= w + (y-\hat{y})x$$

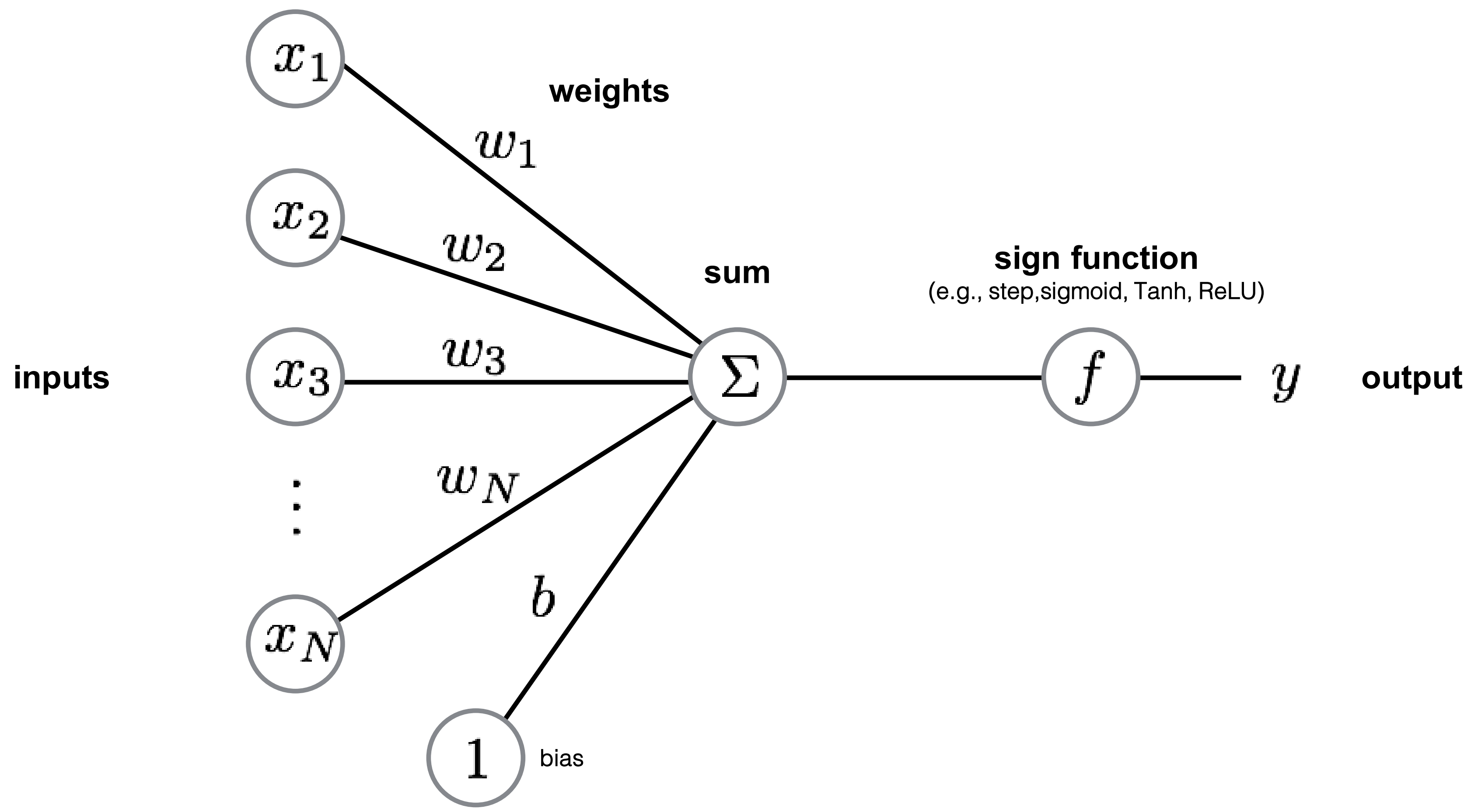

WOW! HOW DID YOU GET LIKE THAT?
Neural Network
EVERY TIME I SEE A NEW TRAINING EXAMPLE ...
Neural Network
I DO ONE ITERATION OF WEIGHT TRAINING
Neural Network
JESUS CHRIST.
SHEN COMIX

Gradient Descent (world's smallest perceptron)

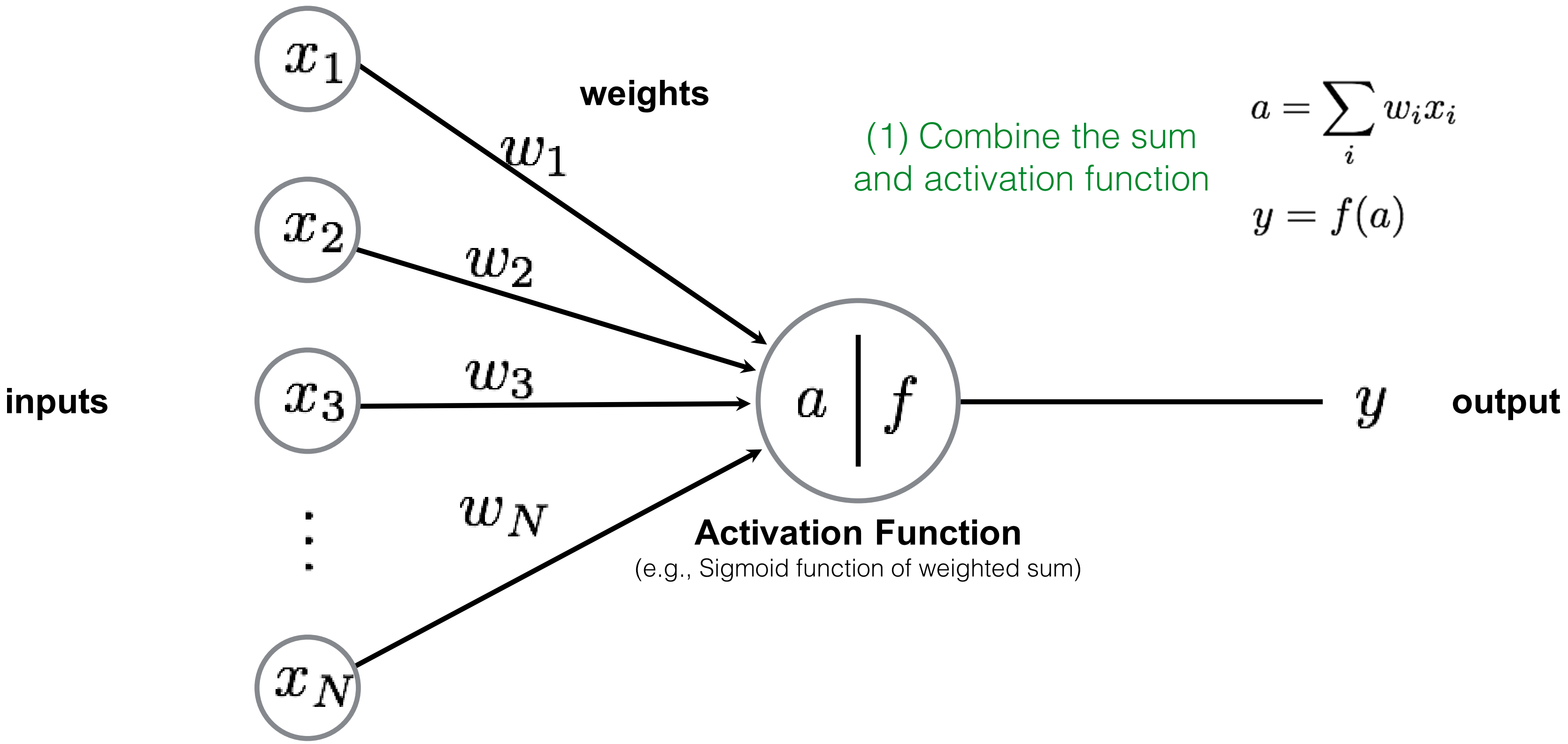
For each sample $\{x_i, y_i\}$

1. Predict
   a. Forward pass $\hat{y} = wx_i$
   b. Compute Loss $\mathcal{L}_i = \frac{1}{2}(y_i - \hat{y})^2$
2. Update
   a. Back Propagation $\frac{d\mathcal{L}_i}{dw} = -(y_i - \hat{y})x_i = \nabla w$
   b. Gradient update $w = w - \nabla w$

The Perceptron



Another way to draw it…



(1) Combine the sum and activation function

$$a = \sum_i w_i x_i$$

$$y = f(a)$$

(2) suppress the bias term (less clutter)

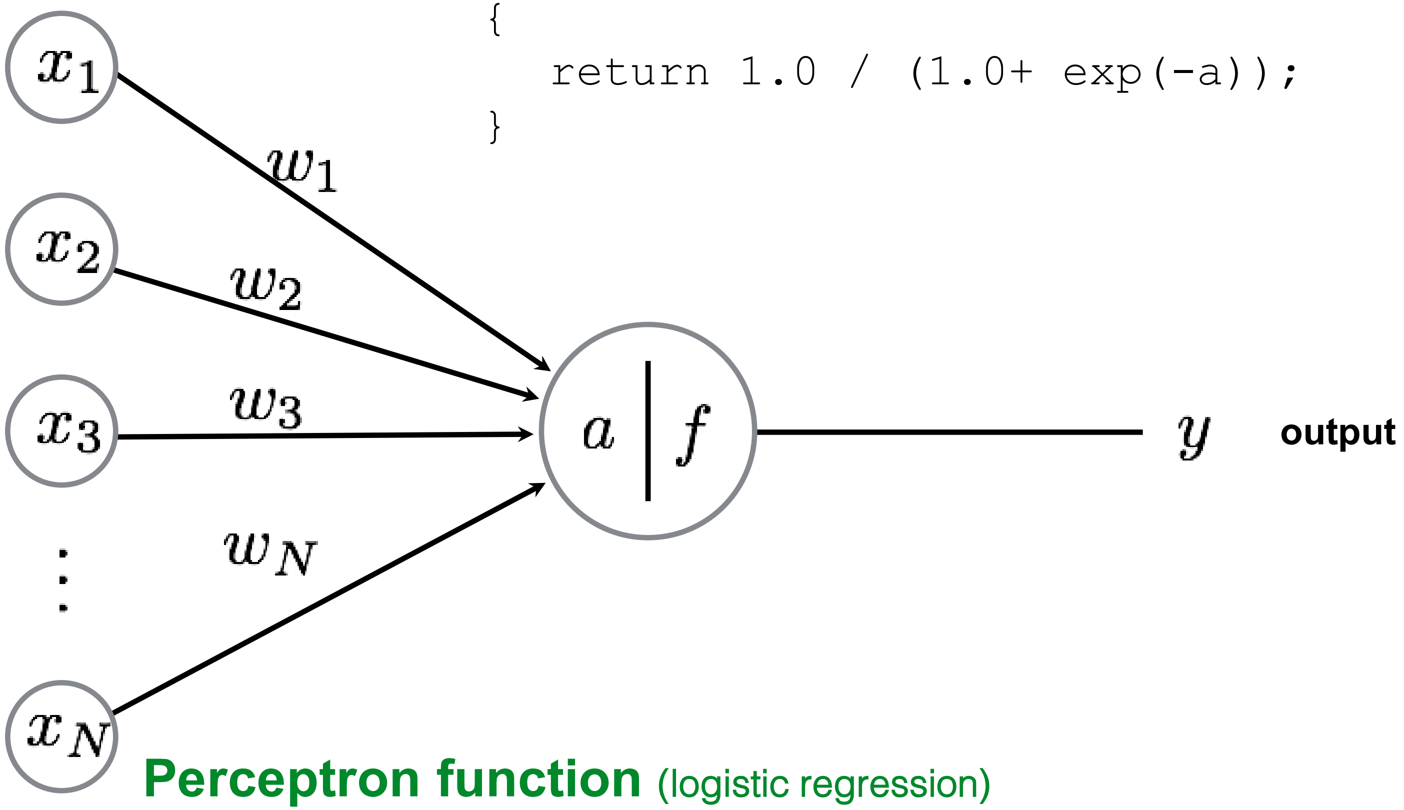
$$x_N = 1$$

$$w_N = b$$

Programming the 'forward pass'

Activation function (sigmoid, logistic function)

```
float f(float a)
{
   return 1.0 / (1.0+ exp(-a));
}
```



Perceptron function (logistic regression)

```
float perceptron(vector<float> x, vector<float> w)
{
   float a  = dot(x,w);
   return f(a);
}
```

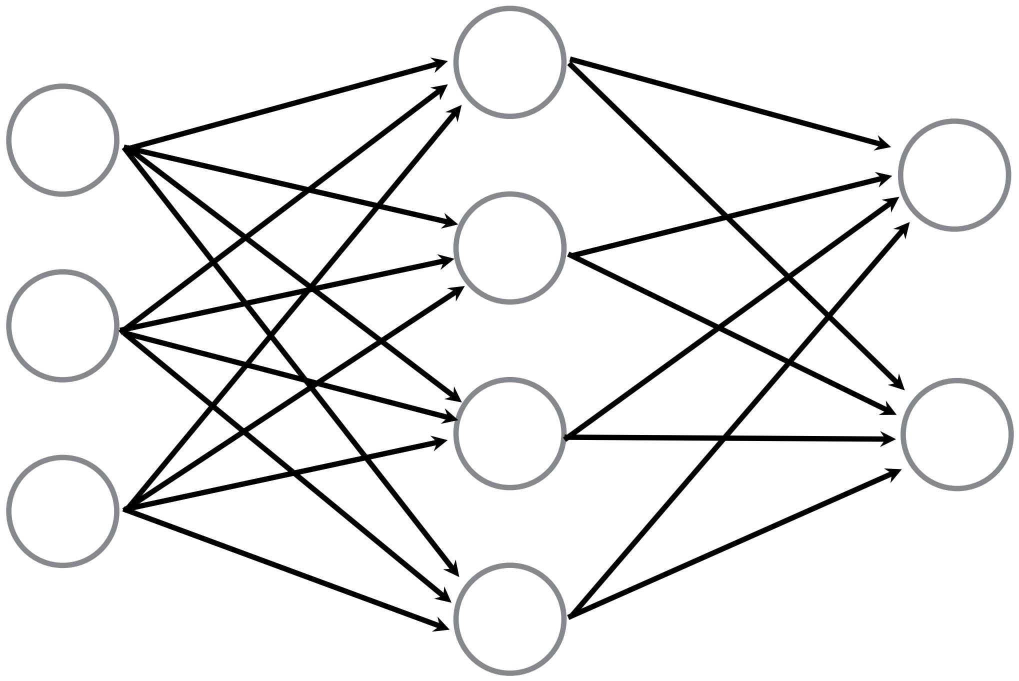
Neural networks

Connect a bunch of perceptrons together …

Connect a bunch of perceptrons together …

Neural Network

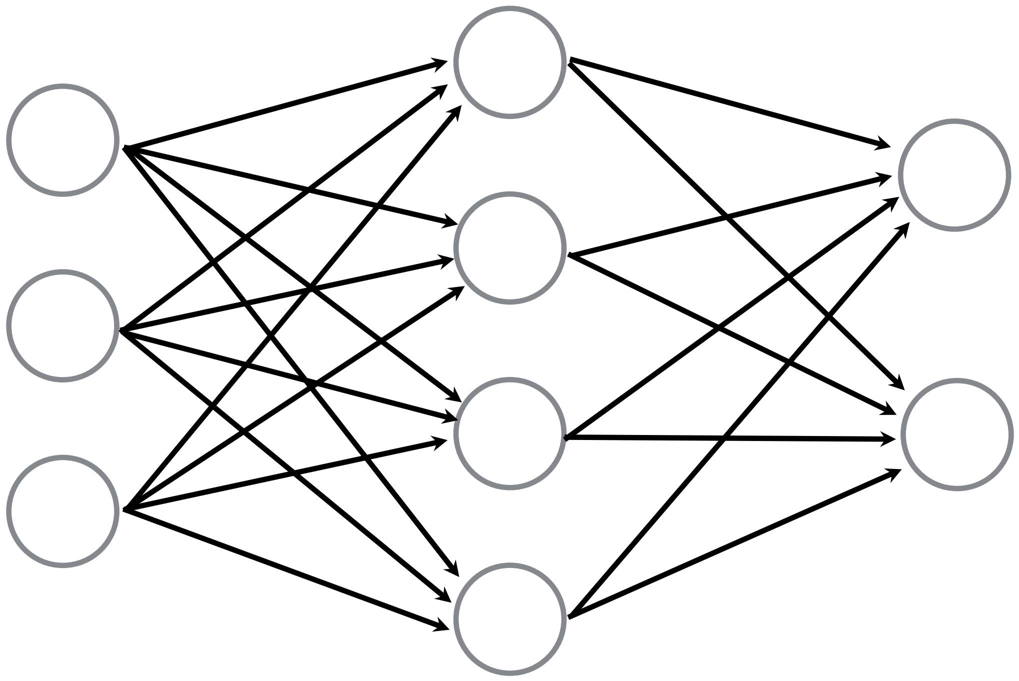
a collection of connected perceptrons



Connect a bunch of perceptrons together …

Neural Network

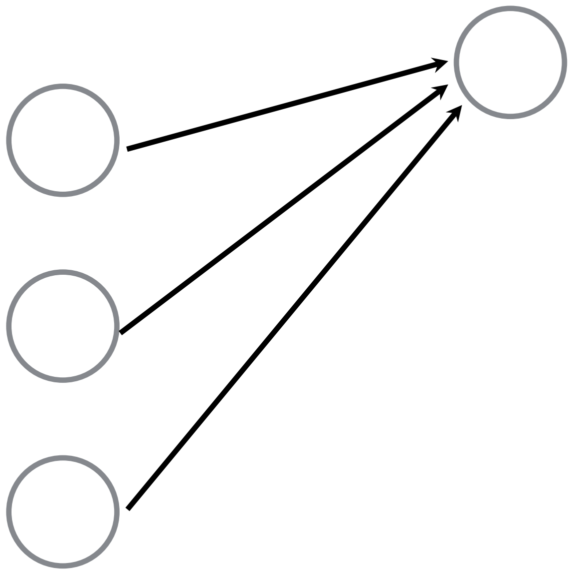
a collection of connected perceptrons



Connect a bunch of perceptrons together …

Neural Network

a collection of connected perceptrons

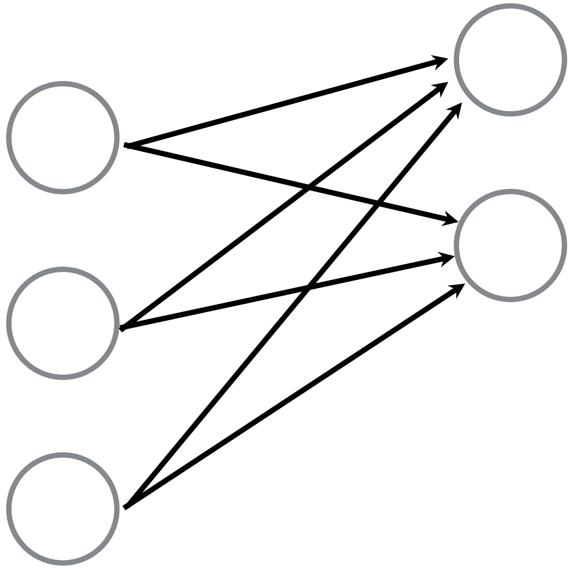


‘one perceptron’

Connect a bunch of perceptrons together …

Neural Network

a collection of connected perceptrons

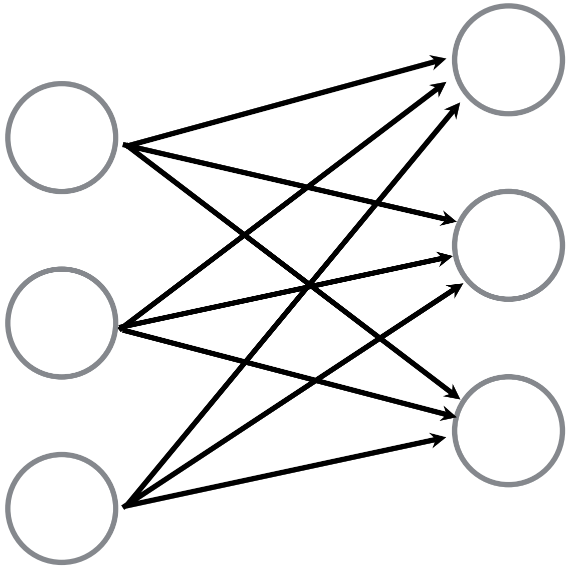


'two perceptrons'

Connect a bunch of perceptrons together …

Neural Network

a collection of connected perceptrons

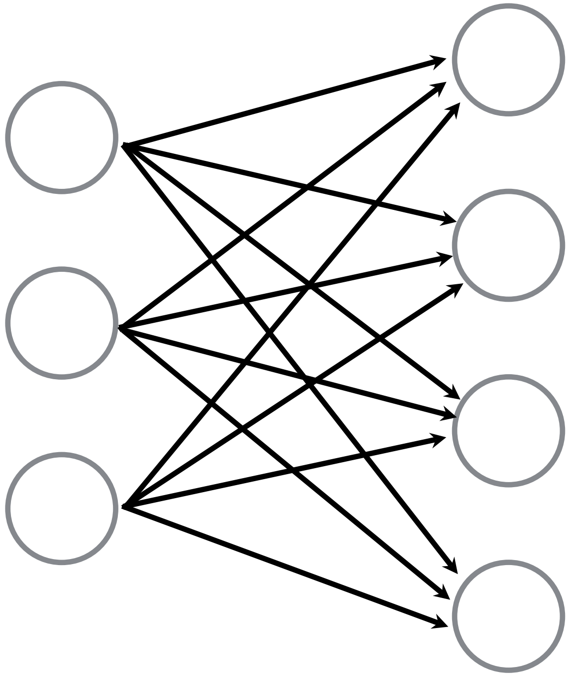


‘three perceptrons’

Connect a bunch of perceptrons together …

Neural Network

a collection of connected perceptrons

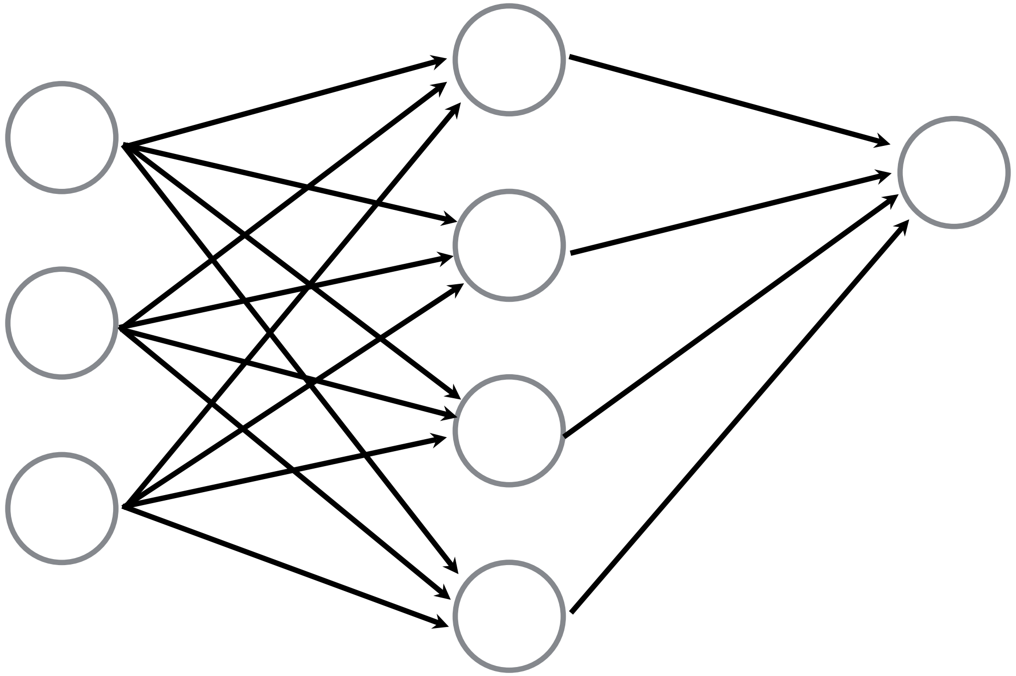


‘four perceptrons’

Connect a bunch of perceptrons together …

Neural Network

a collection of connected perceptrons

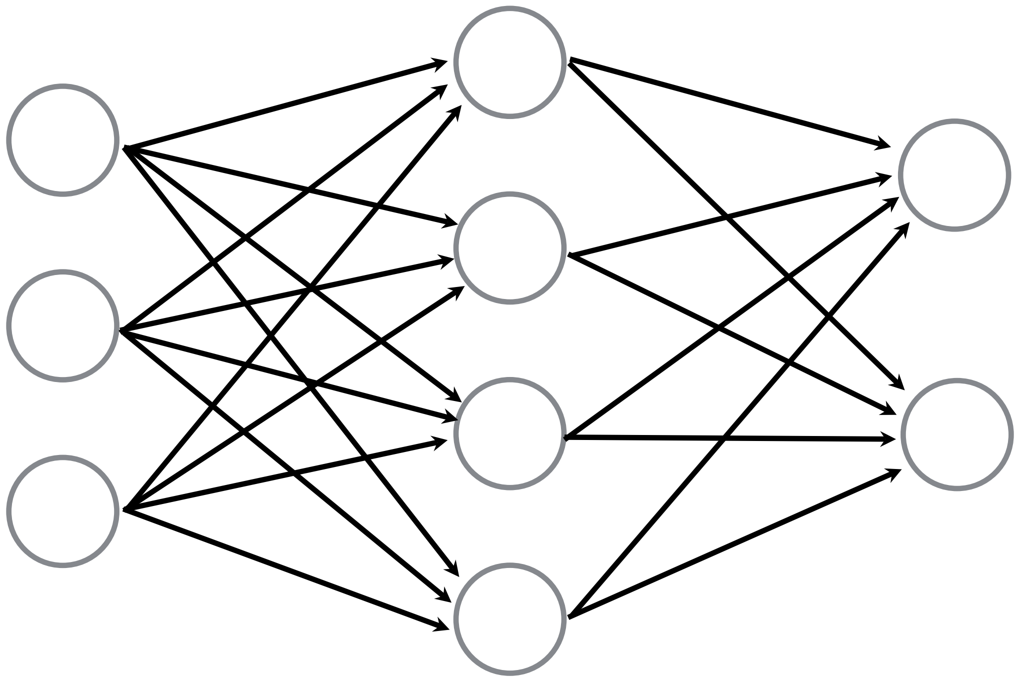


'five perceptrons'

Connect a bunch of perceptrons together …

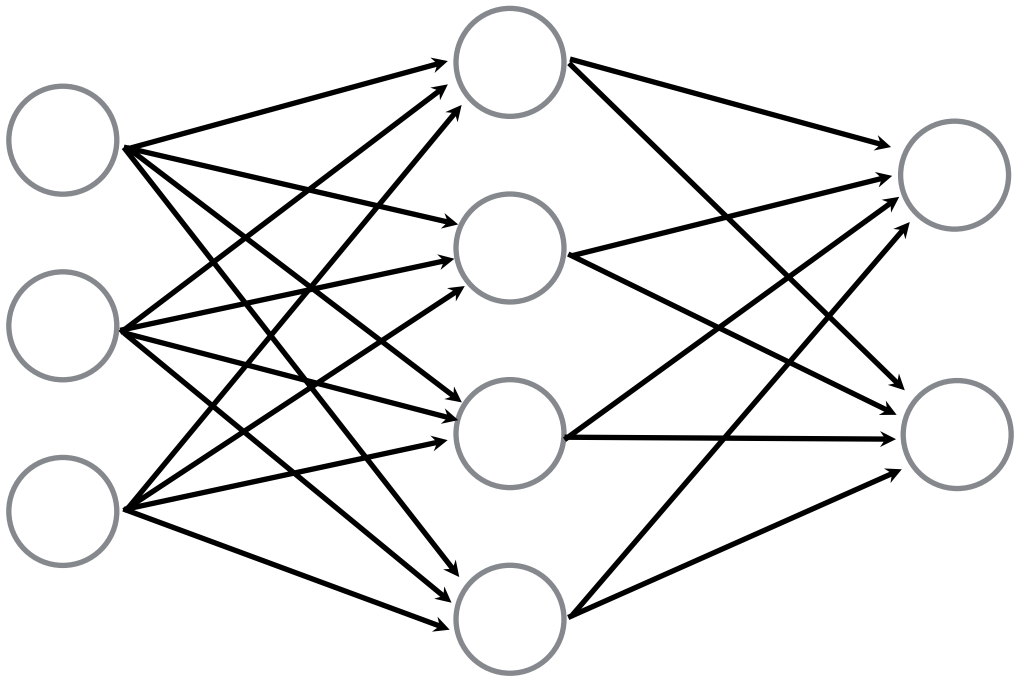
Neural Network

a collection of connected perceptrons



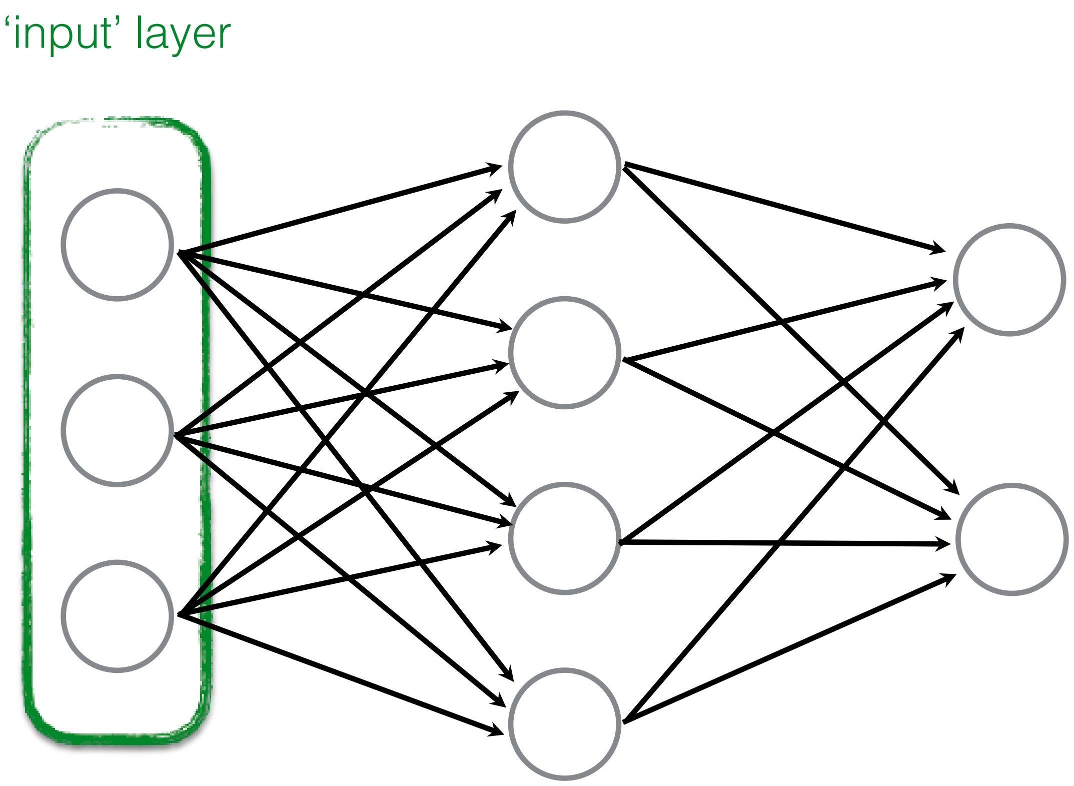
'six perceptrons'

Some terminology…



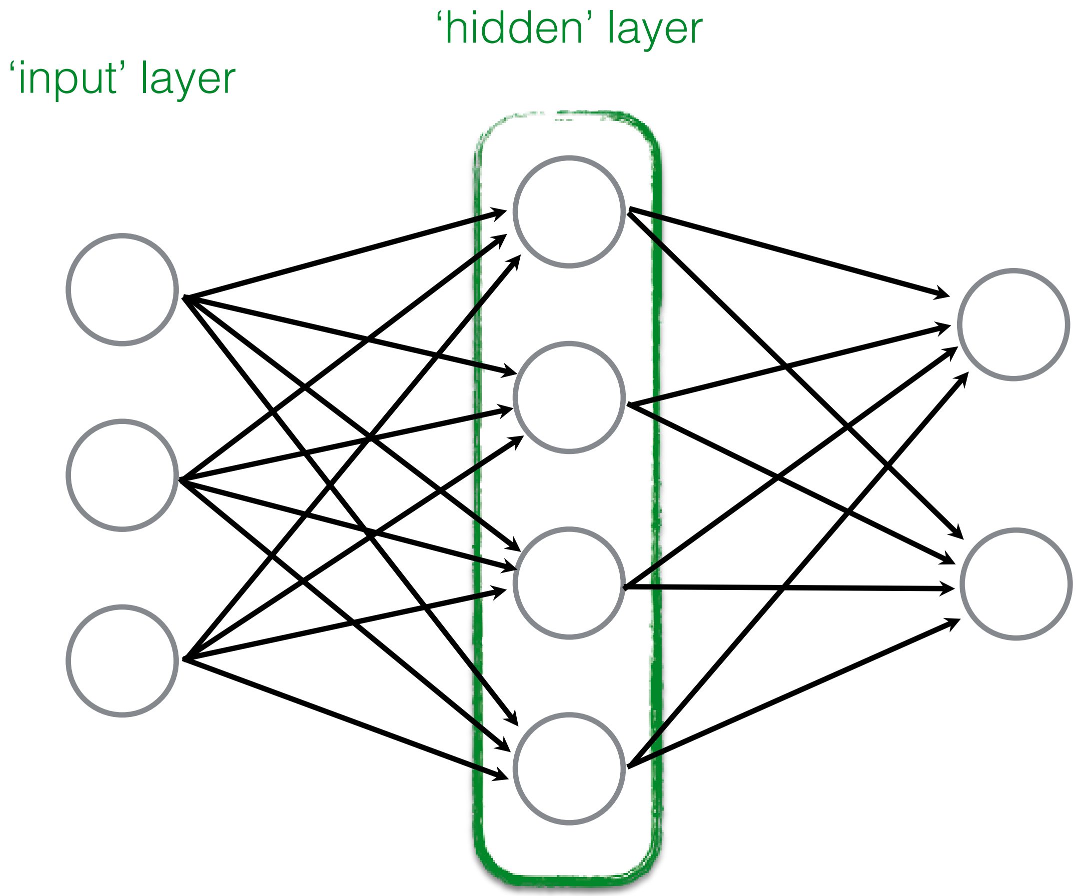
…also called a **Multi-layer Perceptron** (MLP)

Some terminology…



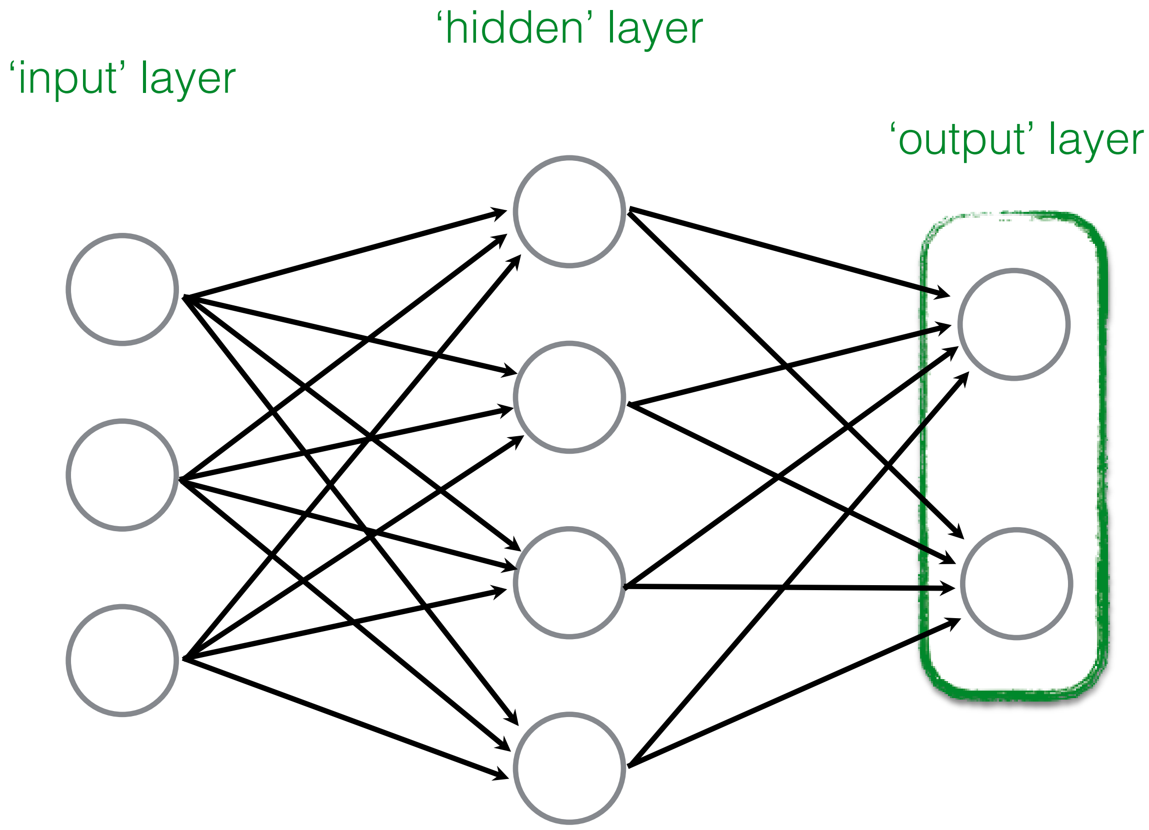
…also called a **Multi-layer Perceptron** (MLP)

Some terminology…



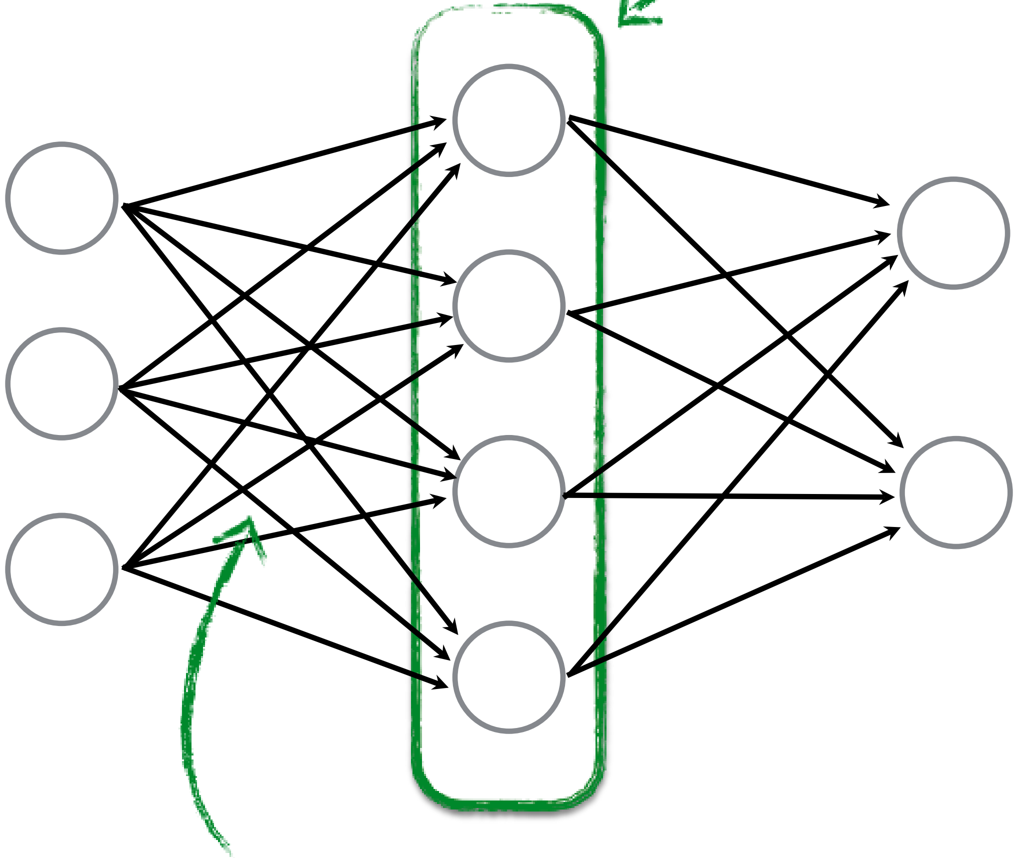
…also called a **Multi-layer Perceptron** (MLP)

Some terminology…

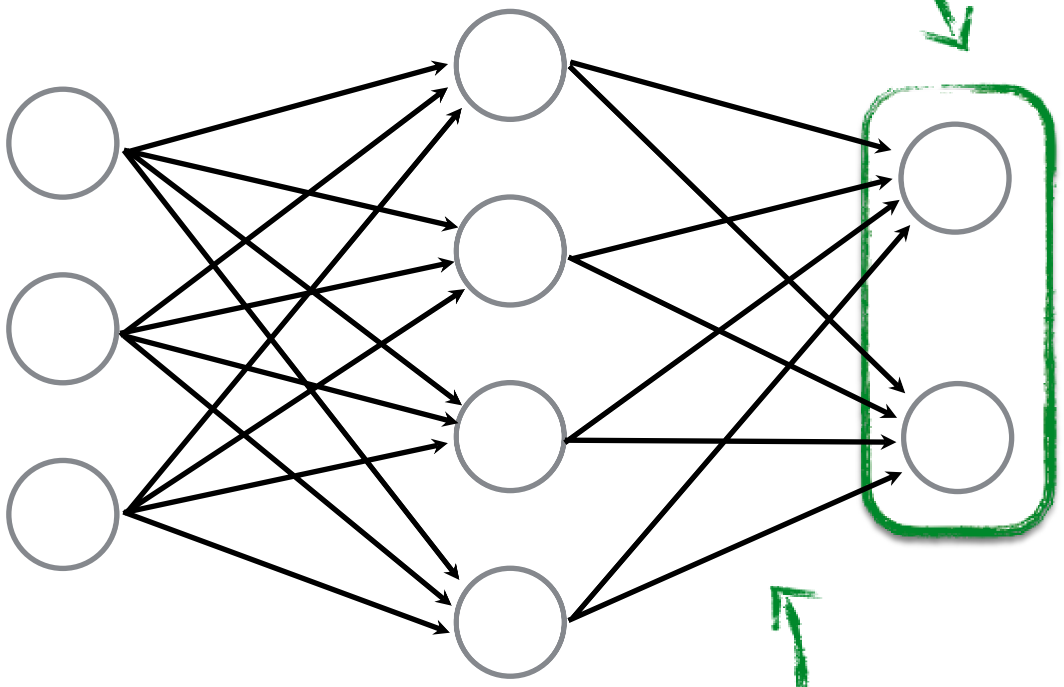


…also called a **Multi-layer Perceptron** (MLP)

this layer is a
‘fully connected layer’



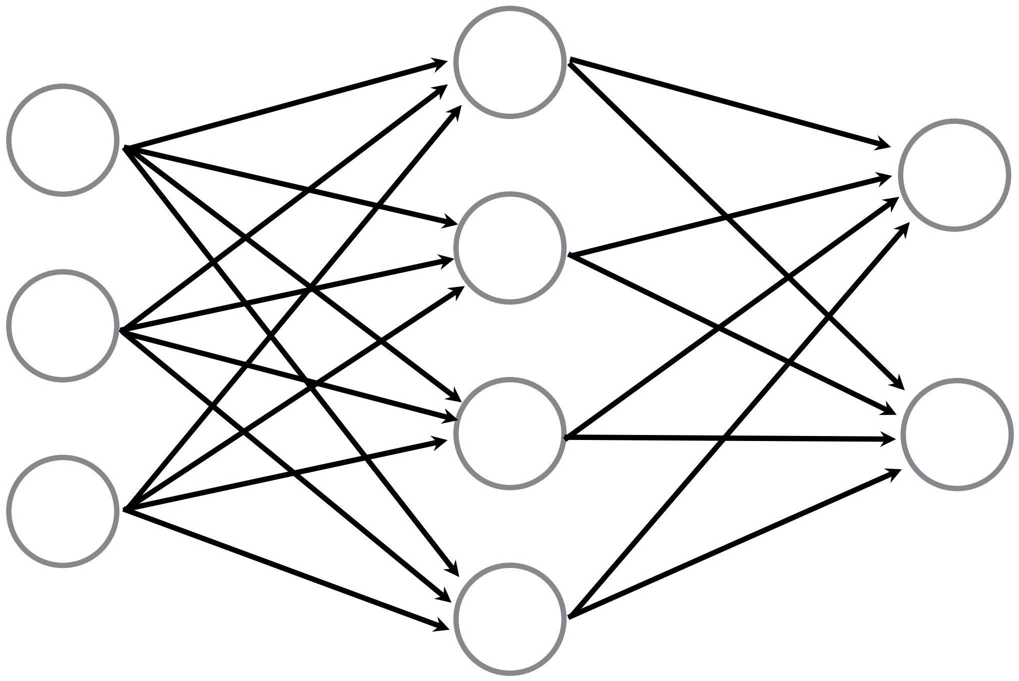
all pairwise neurons between layers are connected so is this



all pairwise neurons between layers are connected

How many neurons (perceptrons)?

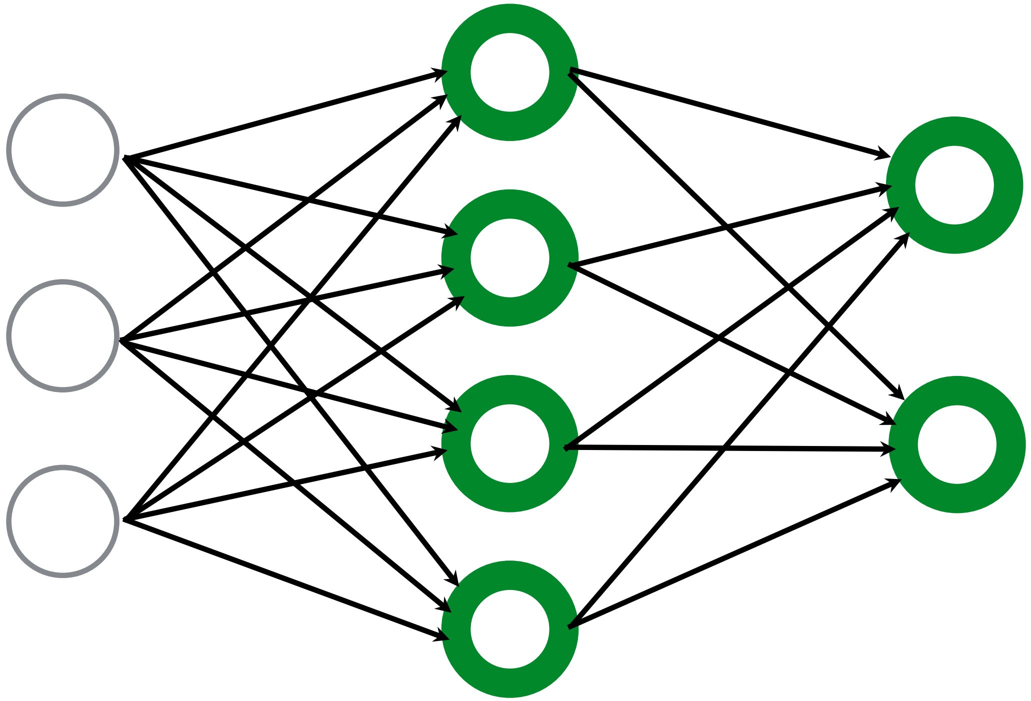
How many weights (edges)?



How many learnable parameters total?

How many neurons (perceptrons)? 4 + 2 = 6

How many weights (edges)?



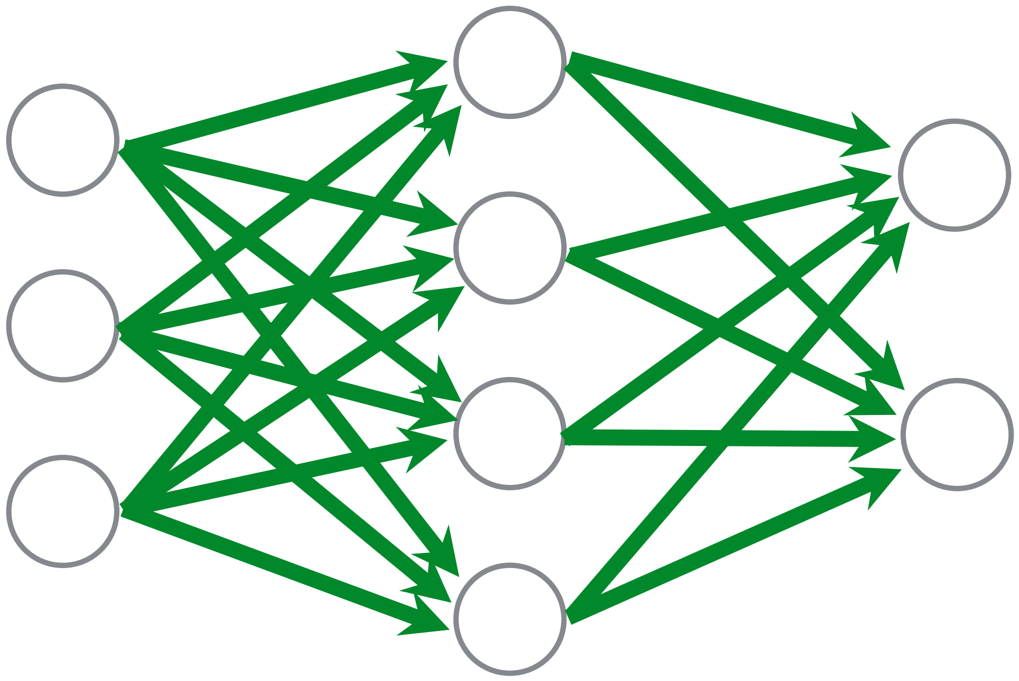
How many learnable parameters total?

How many neurons (perceptrons)?

4 + 2 = 6

How many weights (edges)?

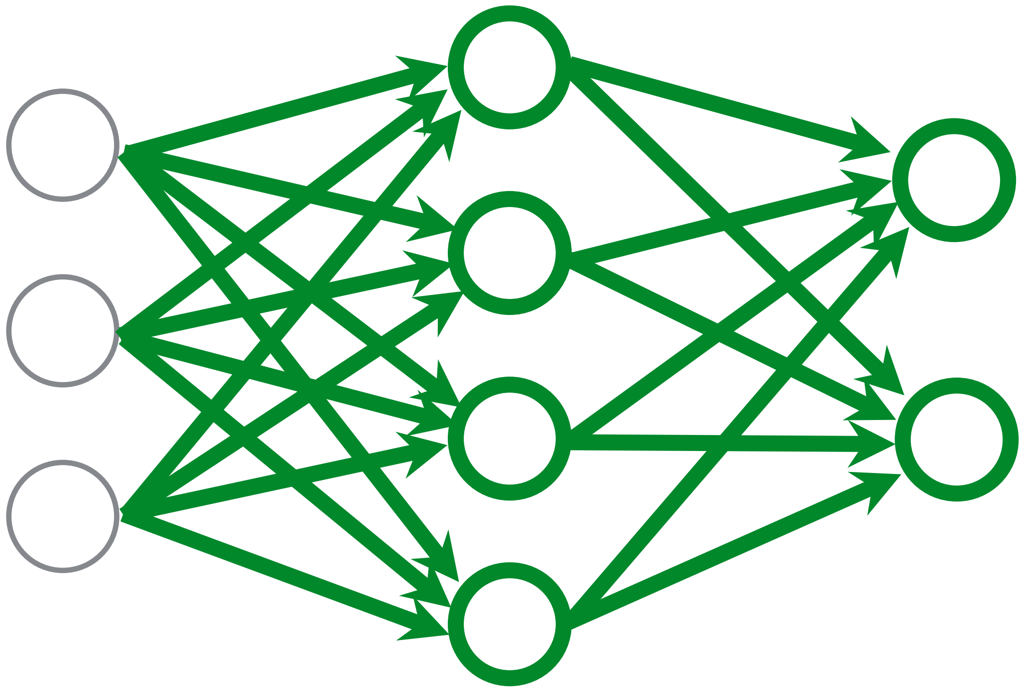
(3 x 4) + (4 x 2) = 20



How many learnable parameters total?

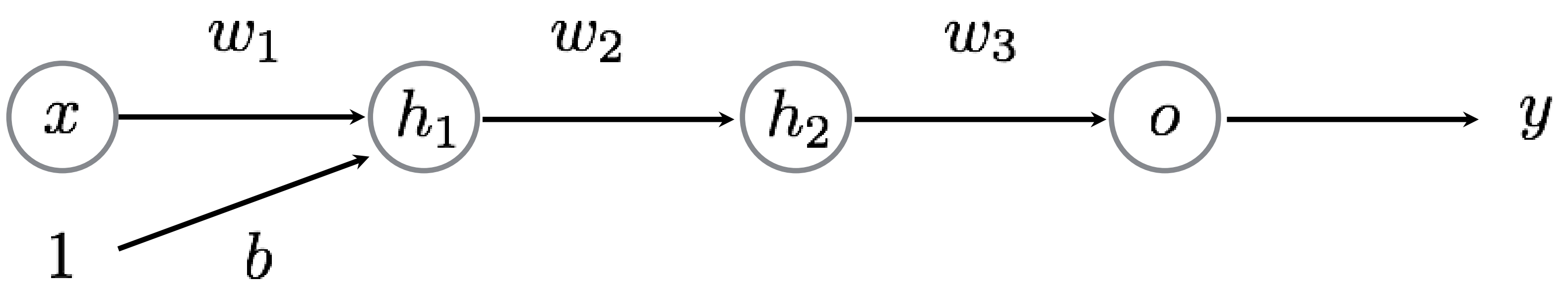
How many neurons (perceptrons)? $4 + 2 = 6$

How many weights (edges)? $(3 \times 4) + (4 \times 2) = 20$

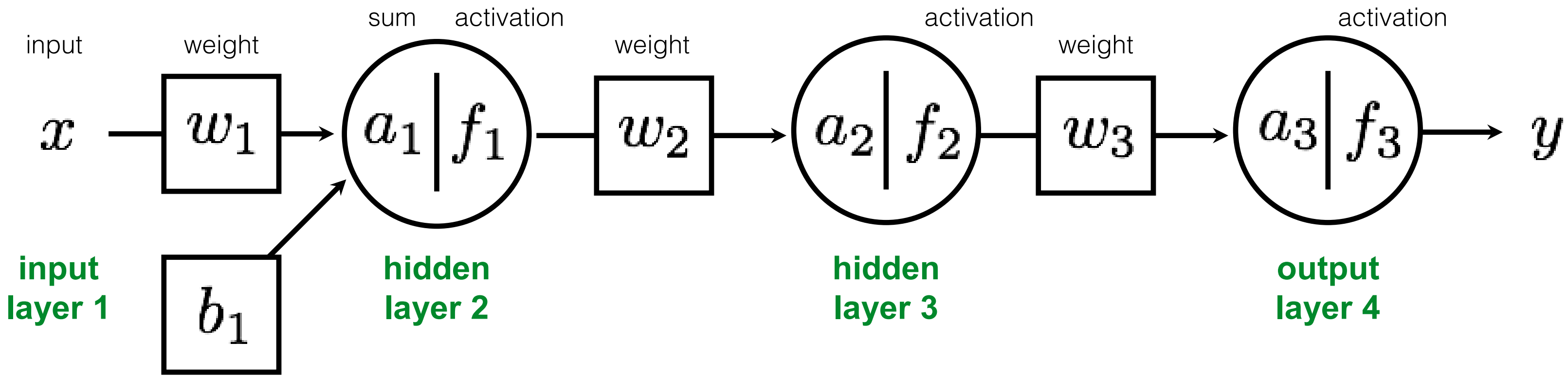


How many learnable parameters total? $20 + \underbrace{4 + 2}_{\text{bias terms}} = 26$

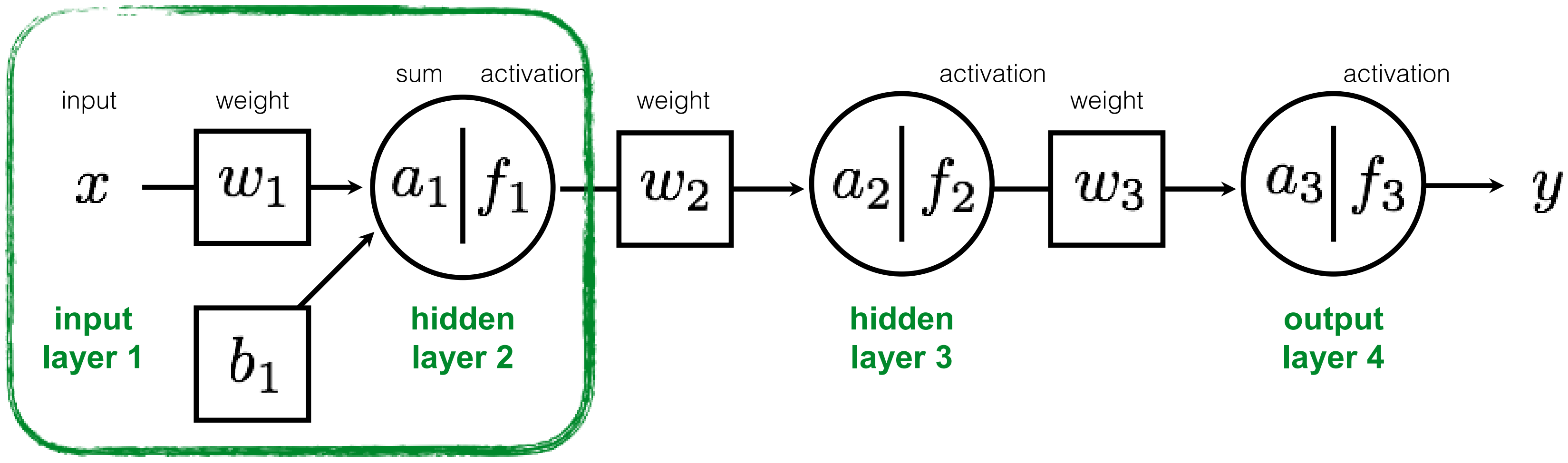
Example of a multi-layer perceptron



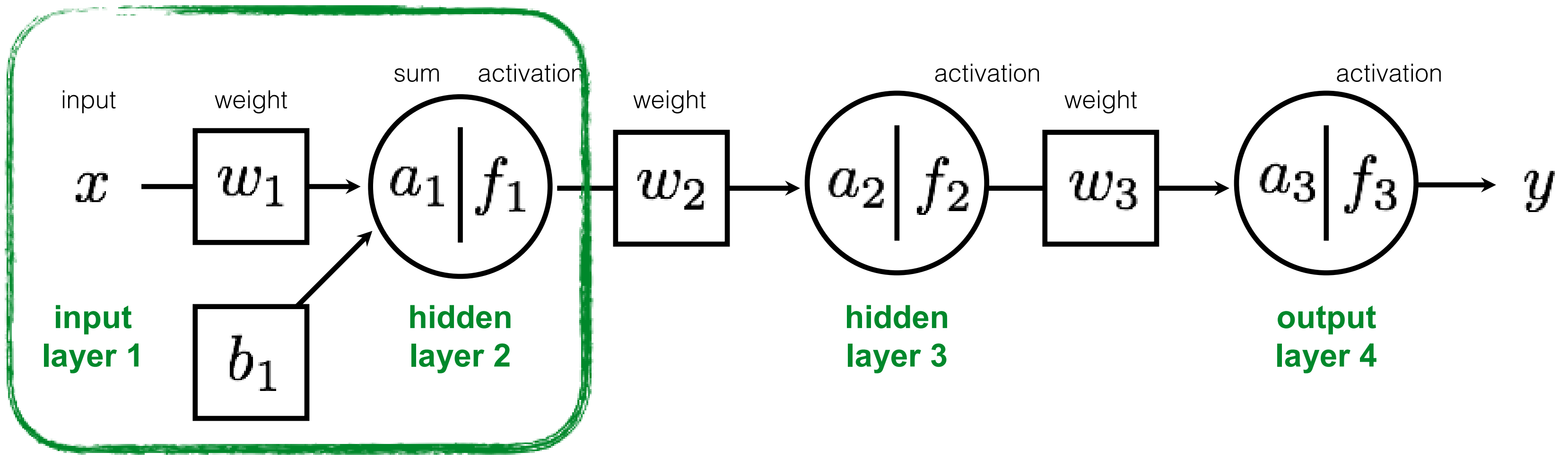
function of **FOUR** parameters and **FOUR** layers!



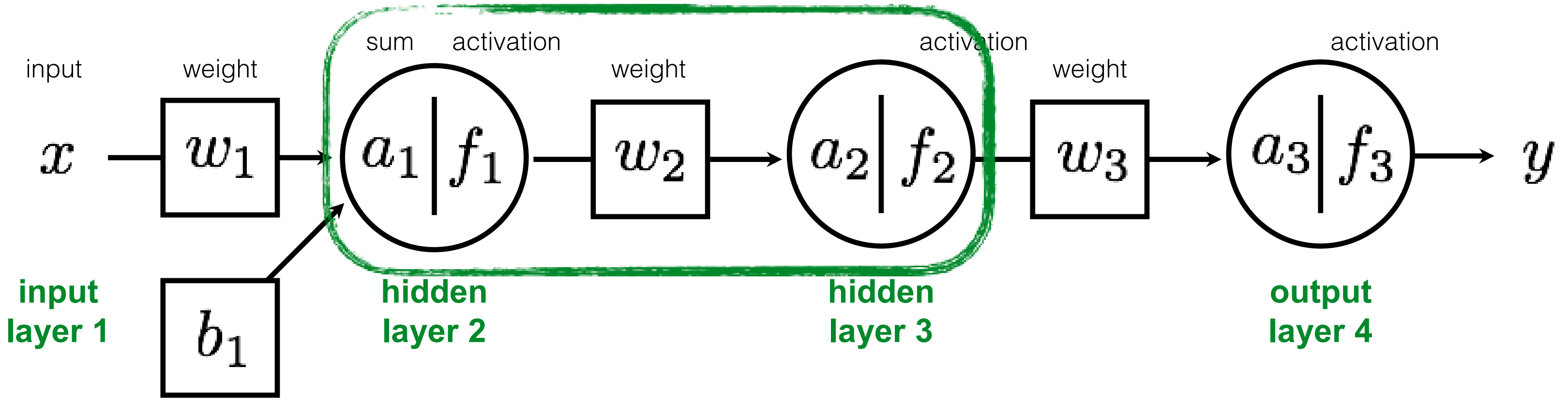
input
weight
sum
activation
weight
activation
weight
activation
x
w_1
a_1
f_1
w_2
a_2
f_2
w_3
a_3
f_3
y
b_1
input layer 1
hidden layer 2
hidden layer 3
output layer 4



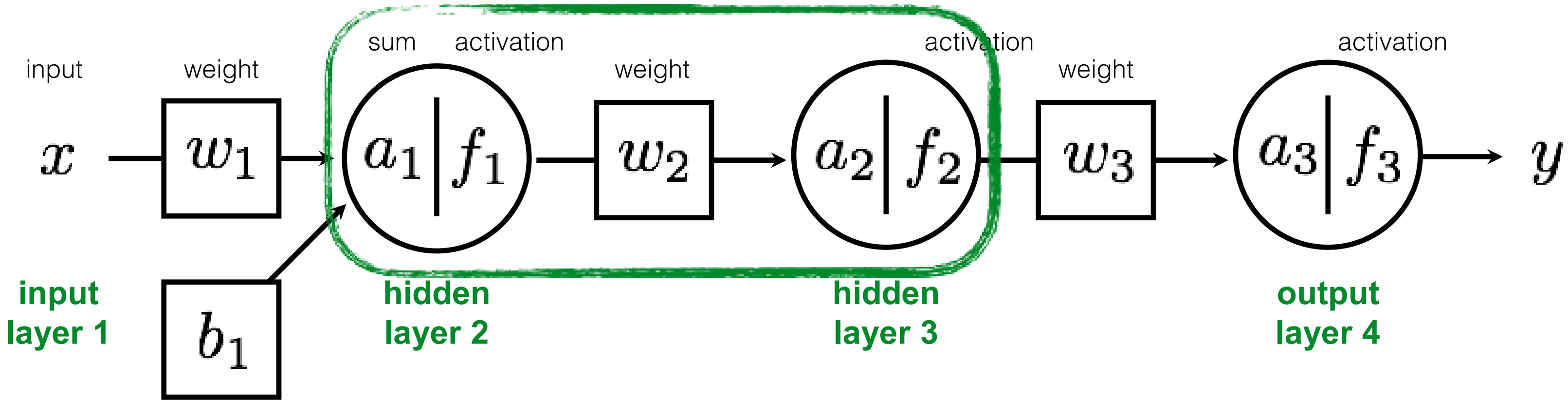
input
weight
sum
activation
weight
activation
weight
activation
x
w_1
a_1 | f_1
w_2
a_2 | f_2
w_3
a_3 | f_3
y
b_1
input
layer 1
hidden
layer 2
hidden
layer 3
output
layer 4



$$a_1 = w_1 \cdot x + b_1$$

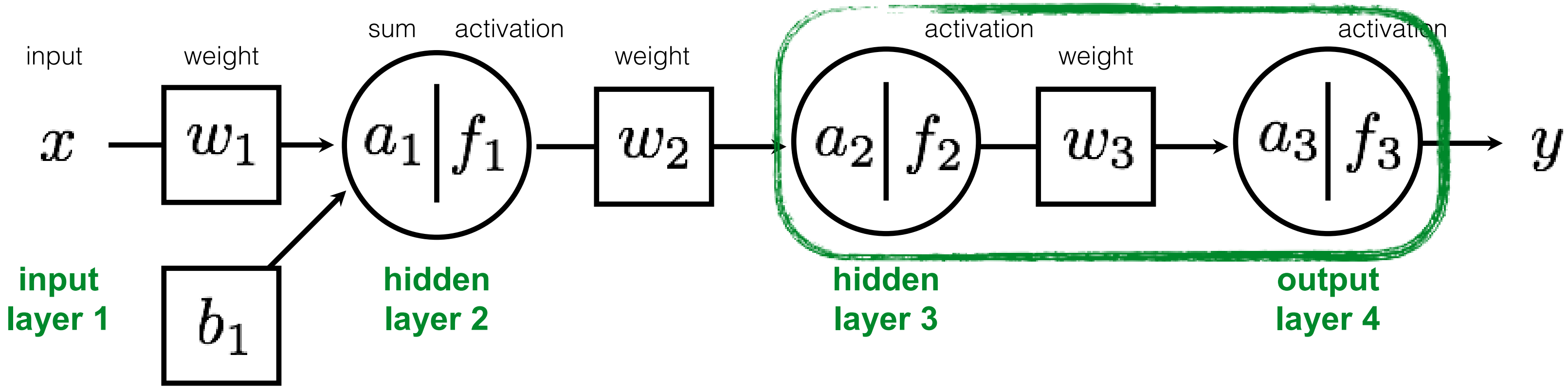


$$a_1 = w_1 \cdot x + b_1$$



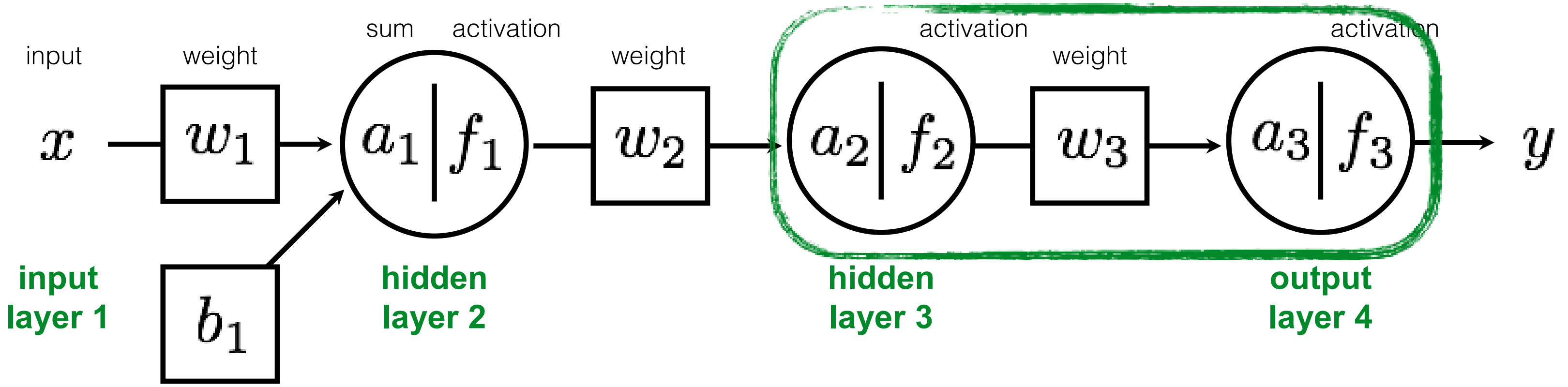
$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$



$$a_1 = w_1 \cdot x + b_1$$

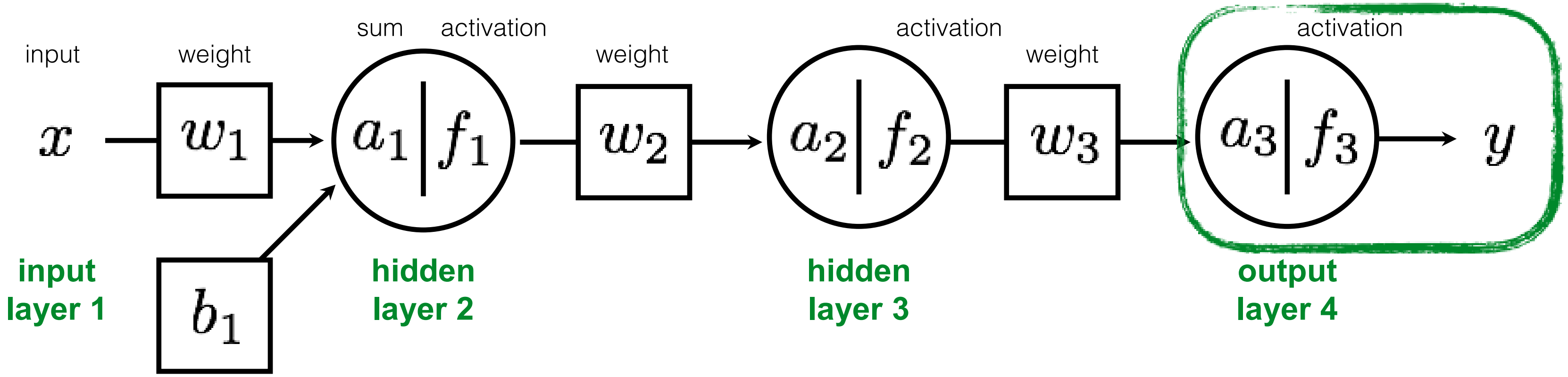
$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$



$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$

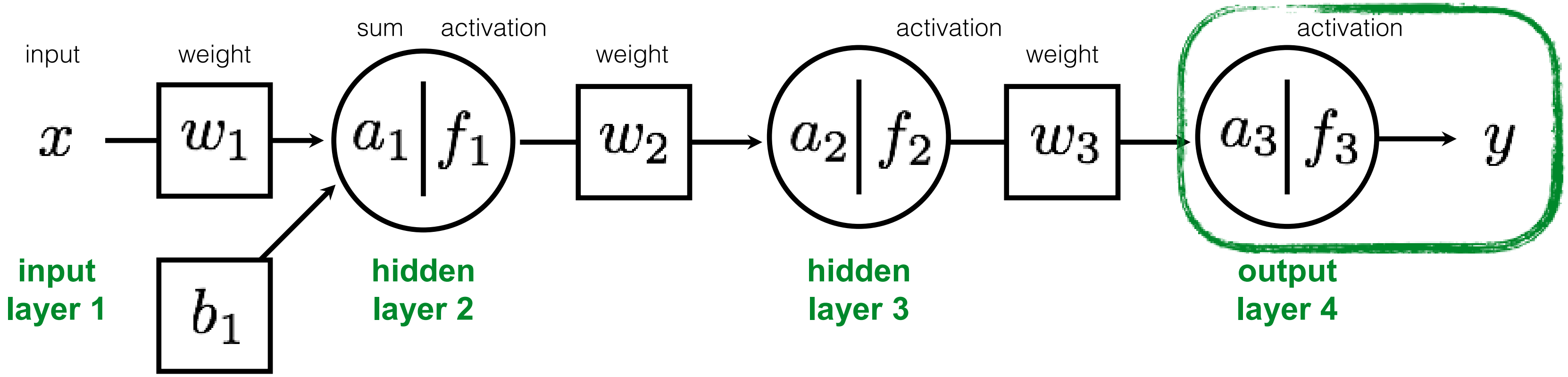
$$a_3 = w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1))$$



$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$

$$a_3 = w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1))$$



$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$

$$a_3 = w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1))$$

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

Entire network can be written out as one long equation

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

We need to train the network:

What is known? What is unknown?

Entire network can be written out as one long equation

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

known

We need to train the network:

What is known? What is unknown?

Entire network can be written out as one long equation

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

activation function
sometimes has
parameters

unknown

We need to train the network:

What is known? What is unknown?

Learning an MLP

Given a set of samples and a MLP

$$\{x_i, y_i\}$$

$$y = f_{\text{MLP}}(x; \theta)$$

Estimate the parameters of the MLP

$$\theta = \{f, w, b\}$$

Gradient Descent

For each **random** sample $\{x_i, y_i\}$

1. Predict
   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$
   b. Compute Loss
2. Update
   a. Back Propagation $\frac{\partial \mathcal{L}}{\partial \theta}$ vector of parameter partial derivatives
   b. Gradient update $\theta \leftarrow \theta - \eta \nabla \theta$

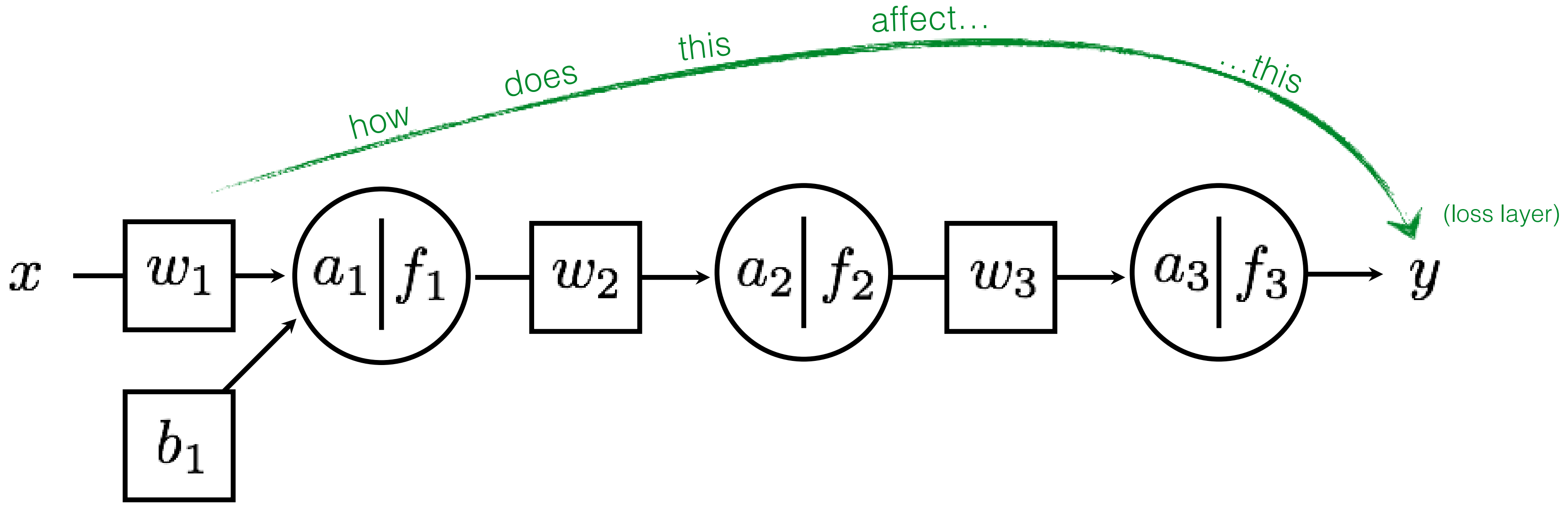
vector of parameter update equations

So we need to compute the partial derivatives

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \left[\frac{\partial \mathcal{L}}{\partial w_3} \frac{\partial \mathcal{L}}{\partial w_2} \frac{\partial \mathcal{L}}{\partial w_1} \frac{\partial \mathcal{L}}{\partial b} \right]$$

Remember,

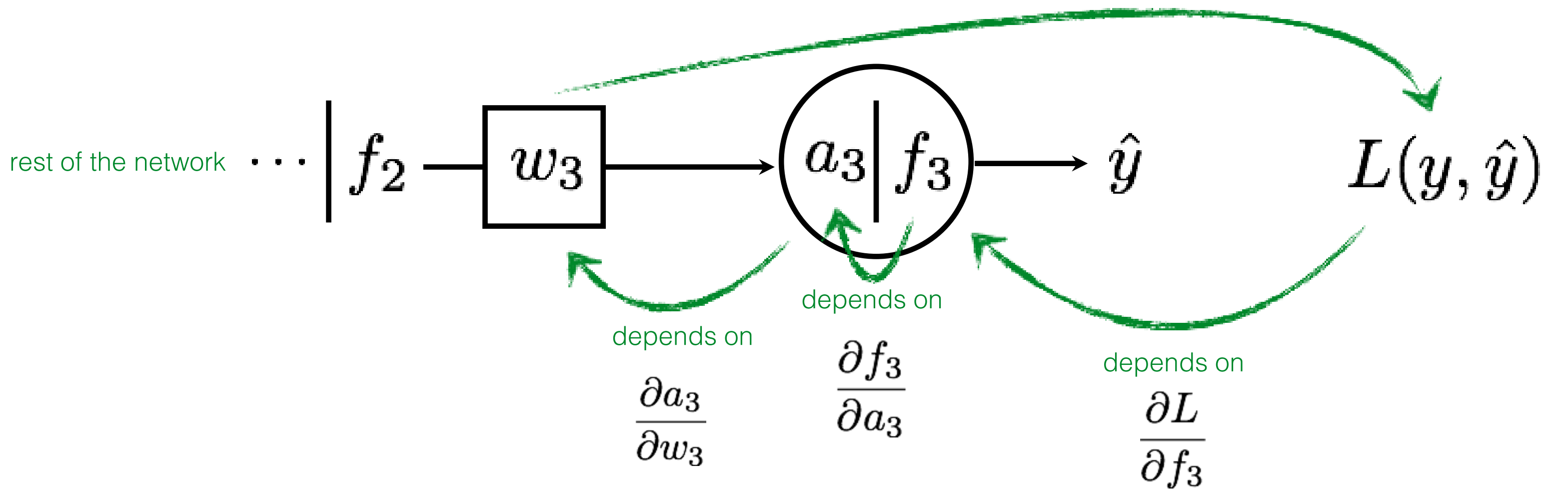
Partial derivative $\dfrac{\partial L}{\partial w_1}$ describes…

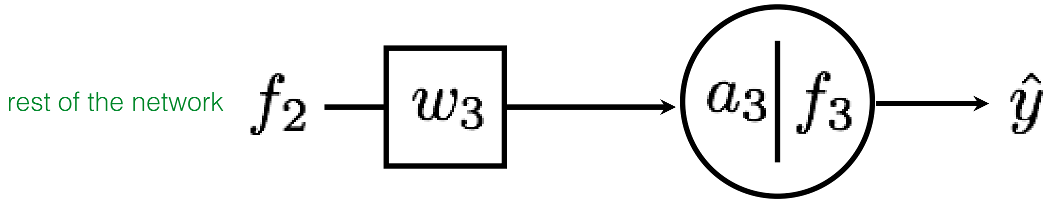


According to the chain rule…

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

Intuitively, the effect of weight on loss function : $\frac{\partial L}{\partial w_3}$





$$L(y, \hat{y})$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3}$$

Chain Rule!

Next Class:

How to use chain rule and “backpropagation” to train any neural network