Machine Learning for Large-Scale Data Analysis and Decision Making (MATH80629A)
Winter 2023

Week #5 - **Summary**

Sources:

http://www.cs.cmu.edu/~16385/

http://cs231n.stanford.edu/syllabus.html

https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

https://www.cs.ubc.ca/labs/lci/mlrg/slides/rnn.pdf

Quiz 2

Login to your Gradescope account

Announcement

- **Homework 1** due **February 17**

- 2nd *optional* Python lab session on **PyTorch** will take place on Thursday **February 8, 1-2:30 pm**, you can join virtually. The recording will be added to Piazza.

- **Quiz 3** will take place **next week**. Topic: **CNNs and RNNs**

Today

- **Third Quiz** on Gradescope!
- Summary of Neural networks and deep learning
- Q&A
- Hands-on session (from the practical session)
- Hands-on session (NNs)

HEC MONTRÉAL

History

1950s Age of the Perceptron

1957 The Perceptron (Rosenblatt)

1969 Perceptrons (Minsky, Papert)

1980s Age of the Neural Network

1986 Back propagation (Hinton)

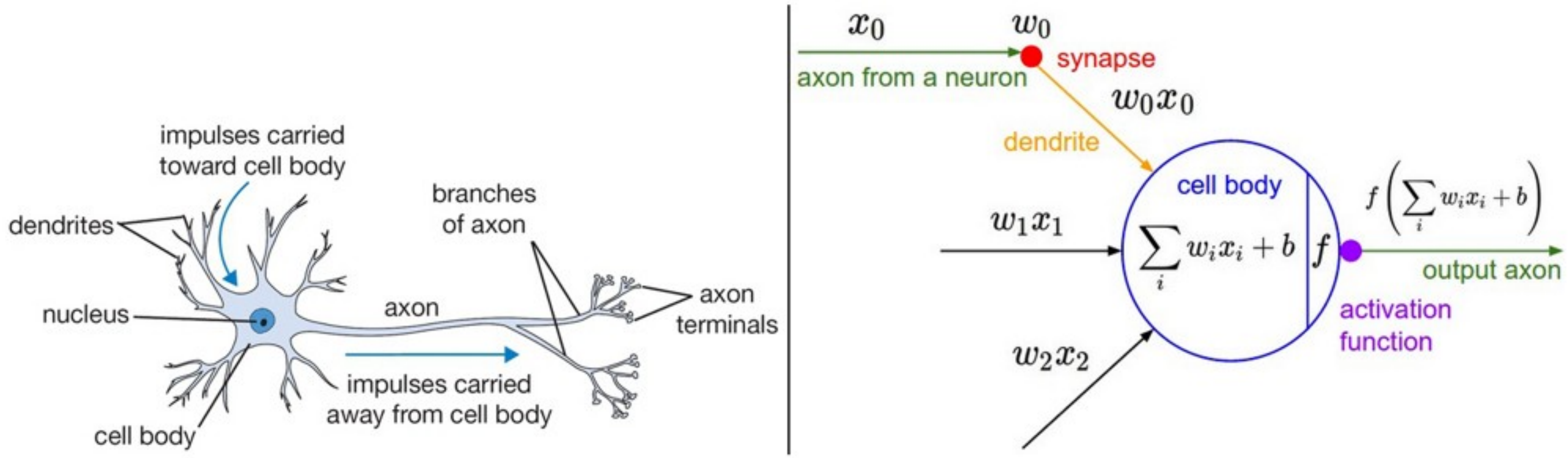
1990s Age of the Graphical Model

2000s Age of the Support Vector Machine

2010s Age of the Deep Network

deep learning = known algorithms + computing power + big data

Inspiration from Biology

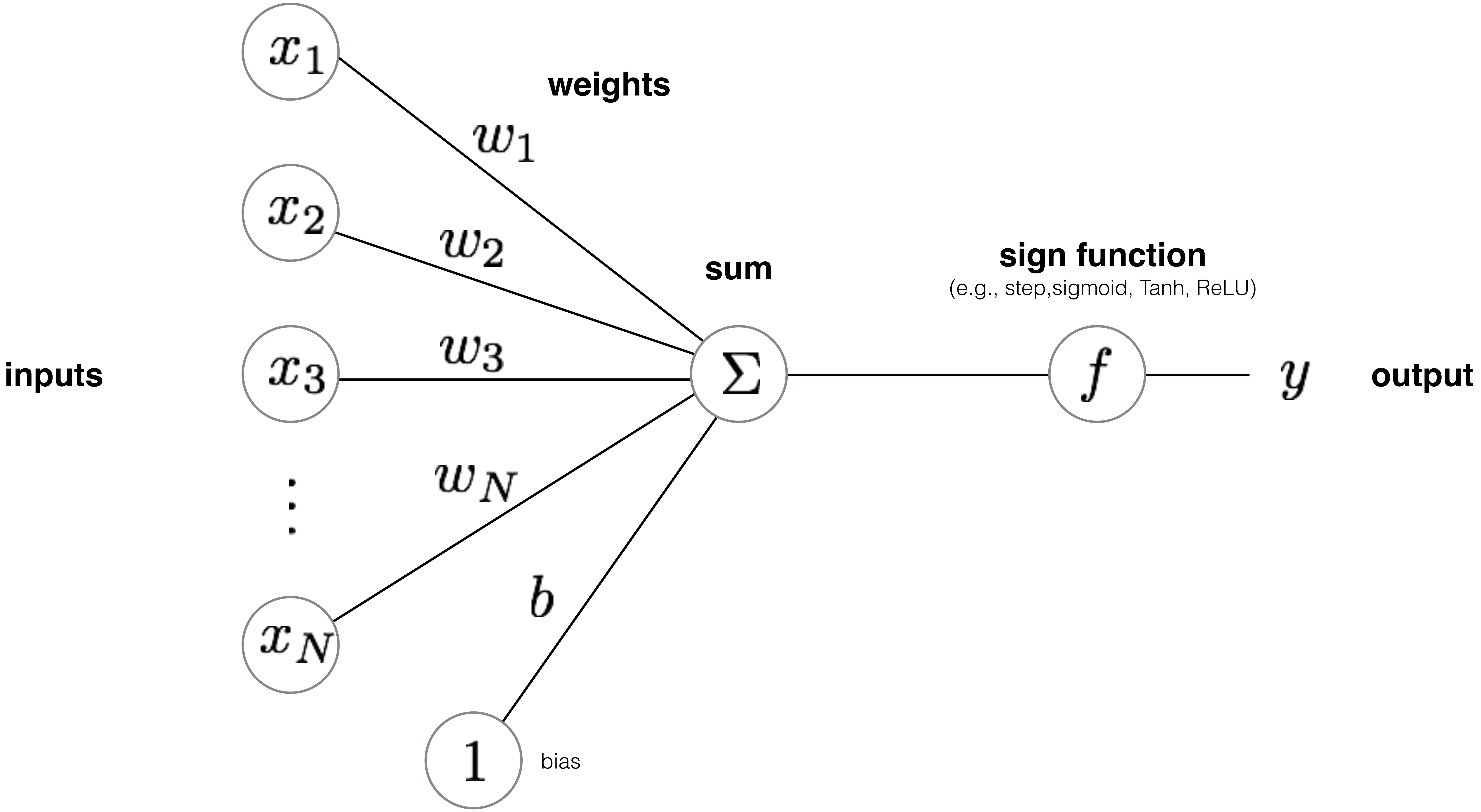


A cartoon drawing of a biological neuron (left) and its mathematical model (right).

Neural nets/perceptrons are **loosely** inspired by biology.

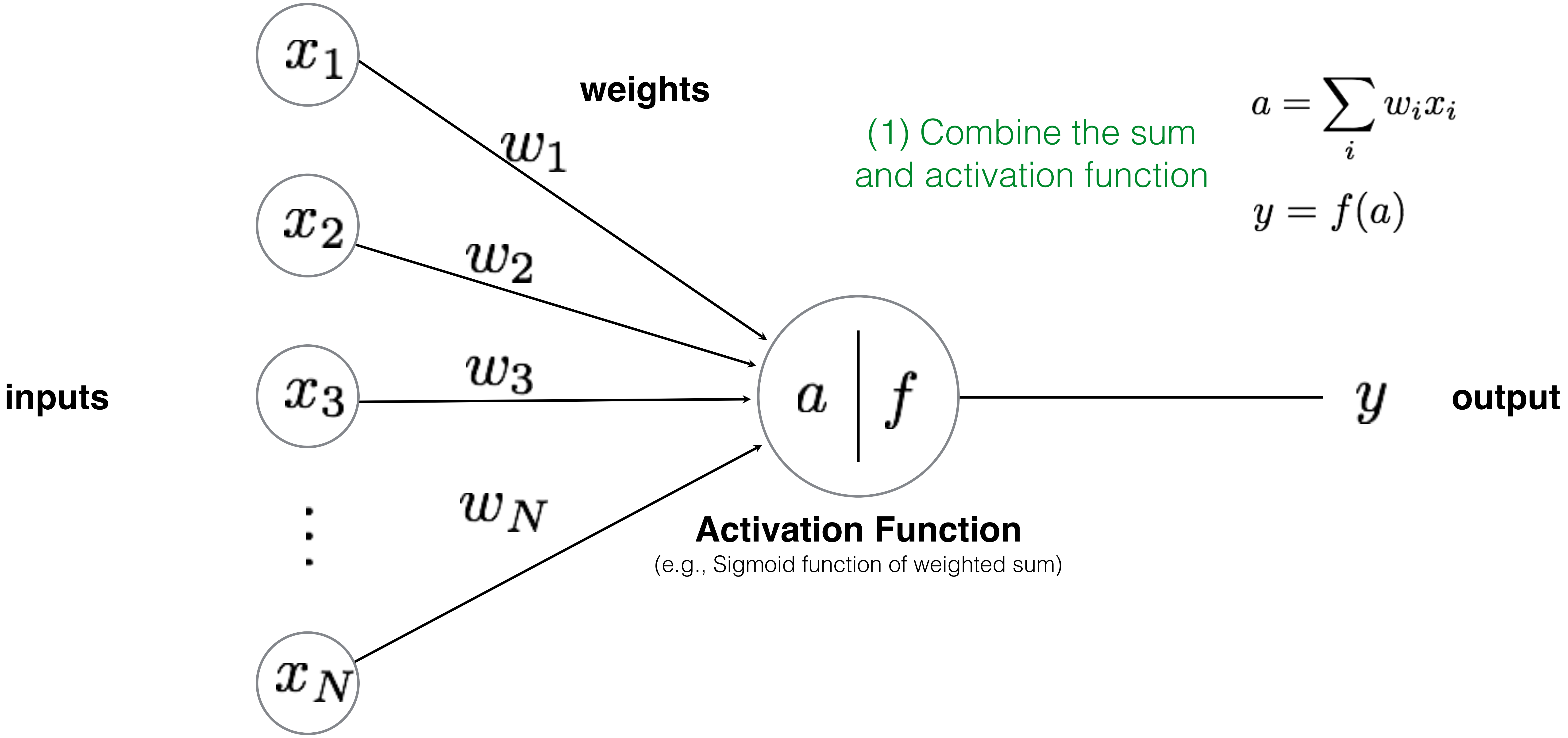
But they certainly are **not** a model of how the brain works, or even how neurons work.

Perceptron



inputs
x_1
x_2
x_3
x_N
weights
w_1
w_2
w_3
w_N
b
1
bias
sum
Σ
sign function
(e.g., step,sigmoid, Tanh, ReLU)
f
y
output

Perceptron



(1) Combine the sum and activation function

$$a = \sum_i w_i x_i$$

$$y = f(a)$$

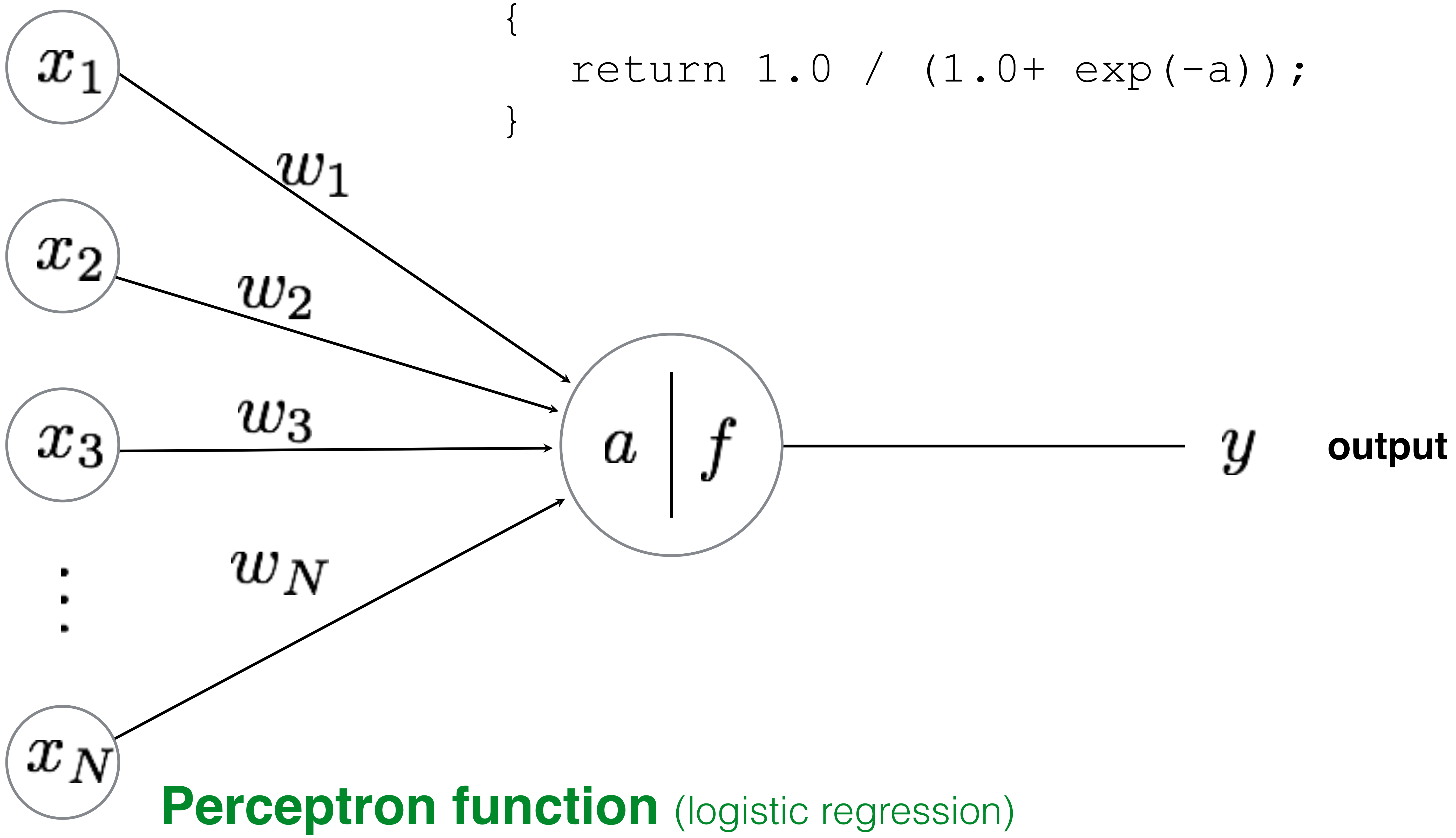
(2) suppress the bias term (less clutter)

$$x_N = 1$$
$$w_N = b$$

Programming the 'forward pass'

Activation function (sigmoid, logistic function)

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



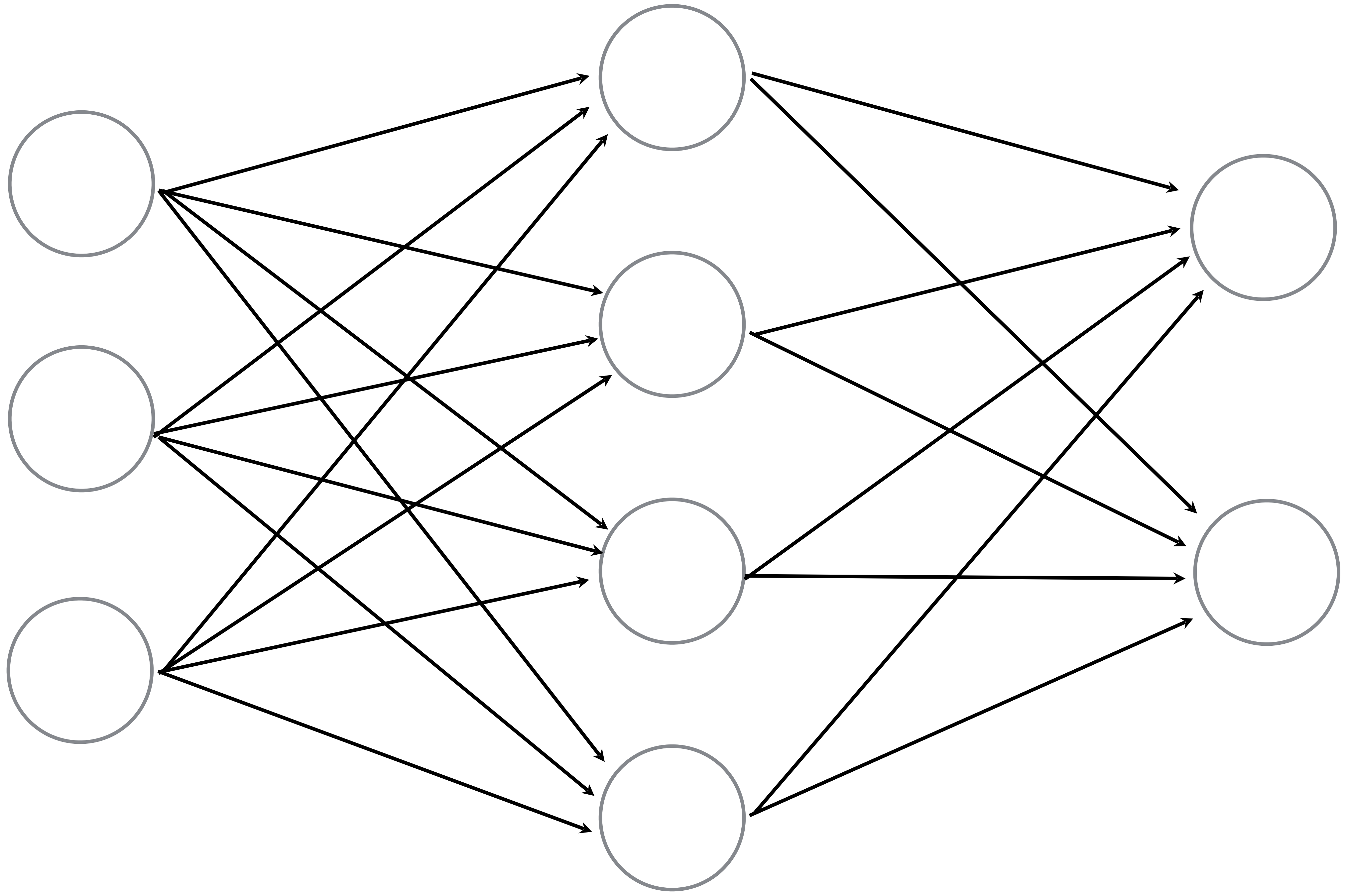
Perceptron function (logistic regression)

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

Neural Network

Connect a bunch of perceptrons together …

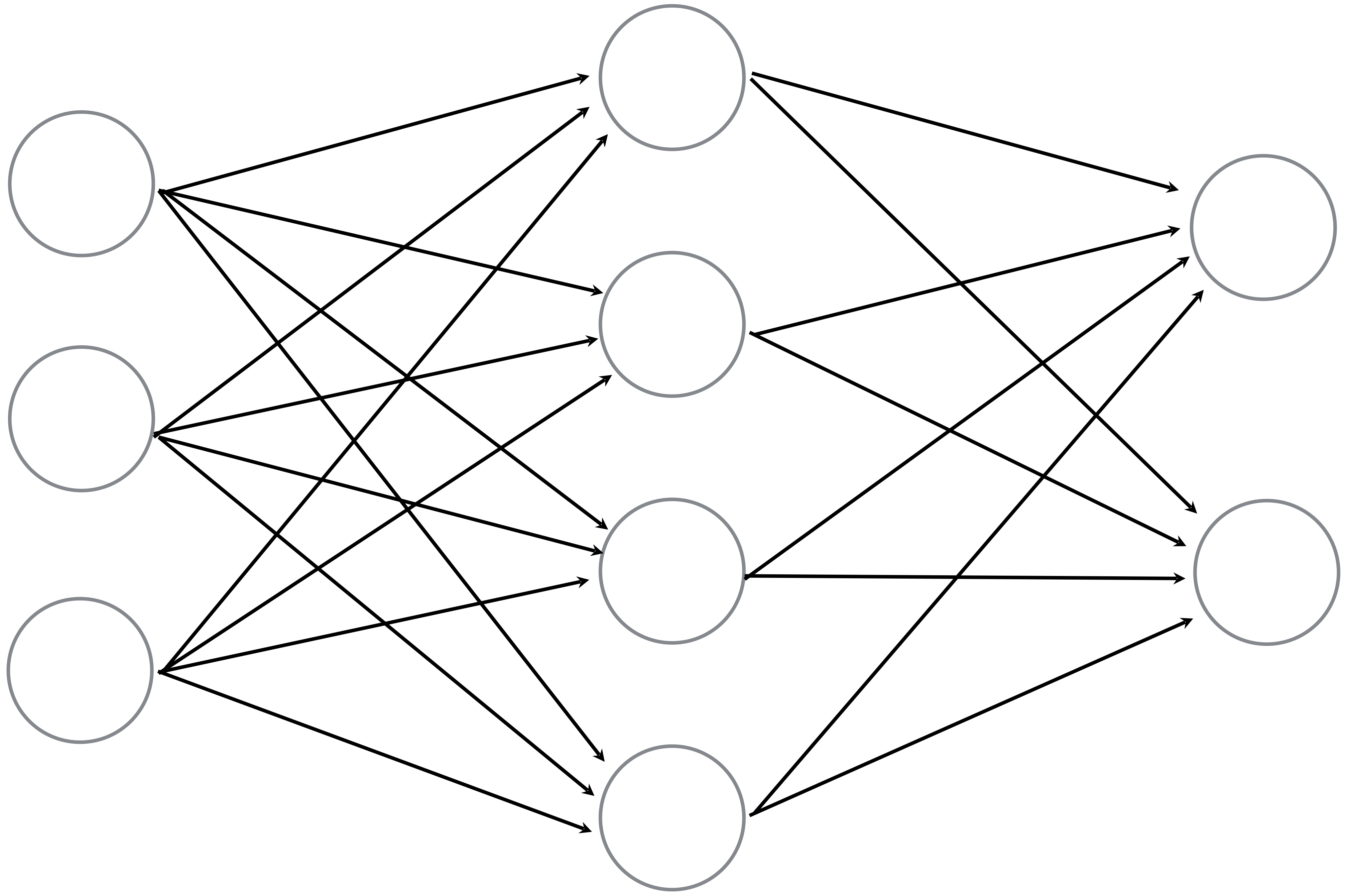
a collection of connected perceptrons



Neural Network

Connect a bunch of perceptrons together …

a collection of connected perceptrons

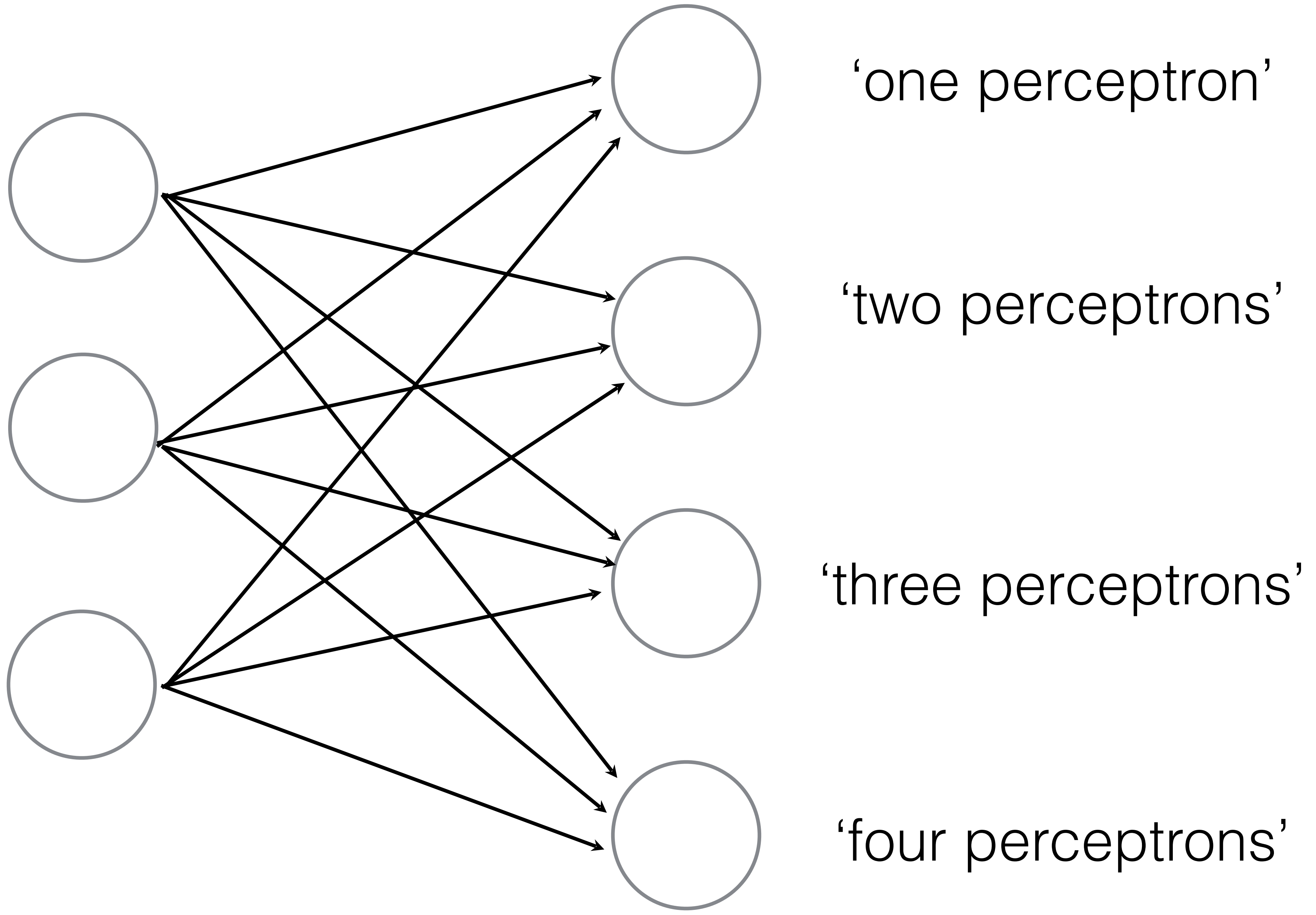


How many perceptrons in this neural network?

Neural Network

Connect a bunch of perceptrons together …

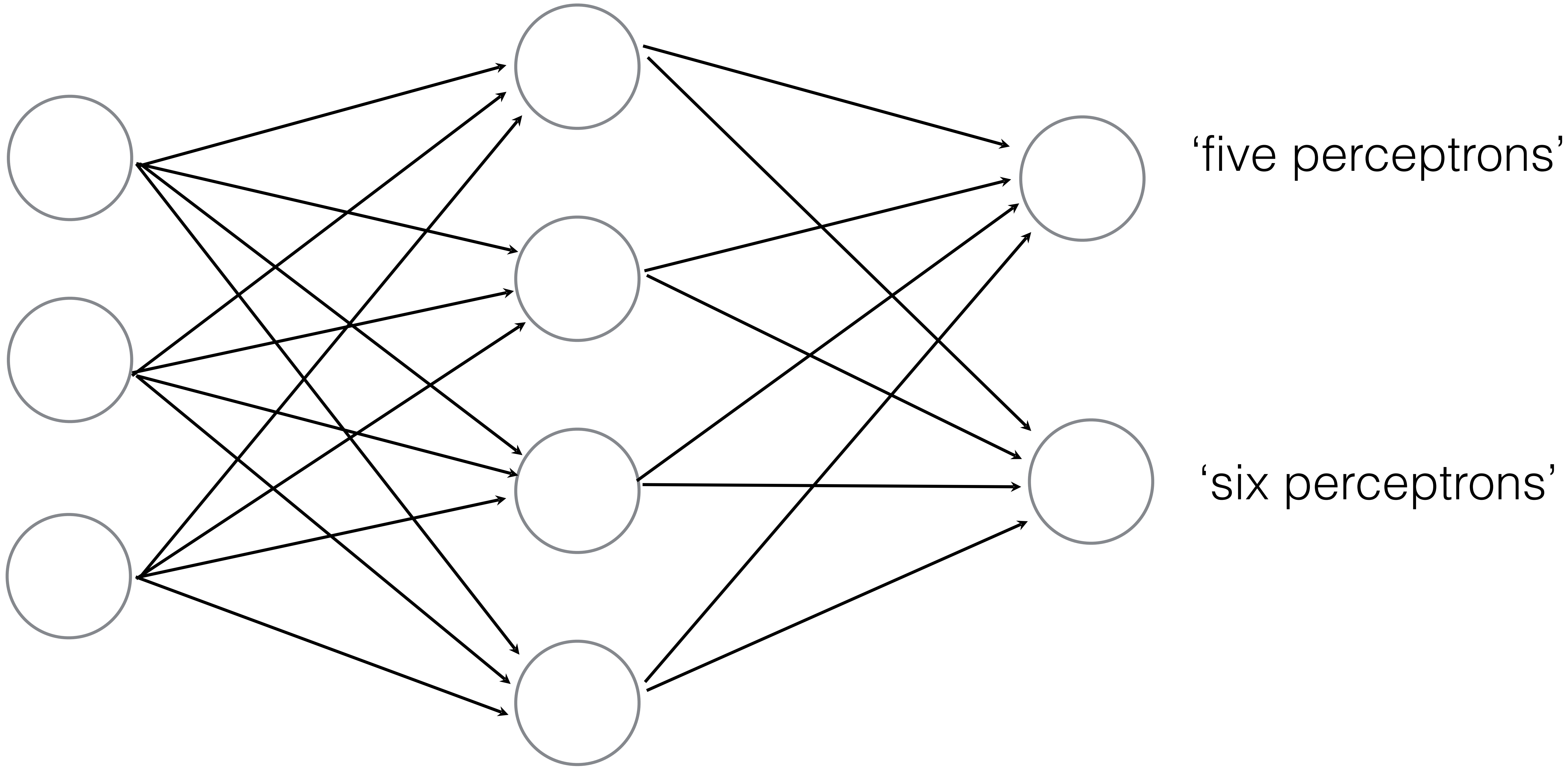
a collection of connected perceptrons



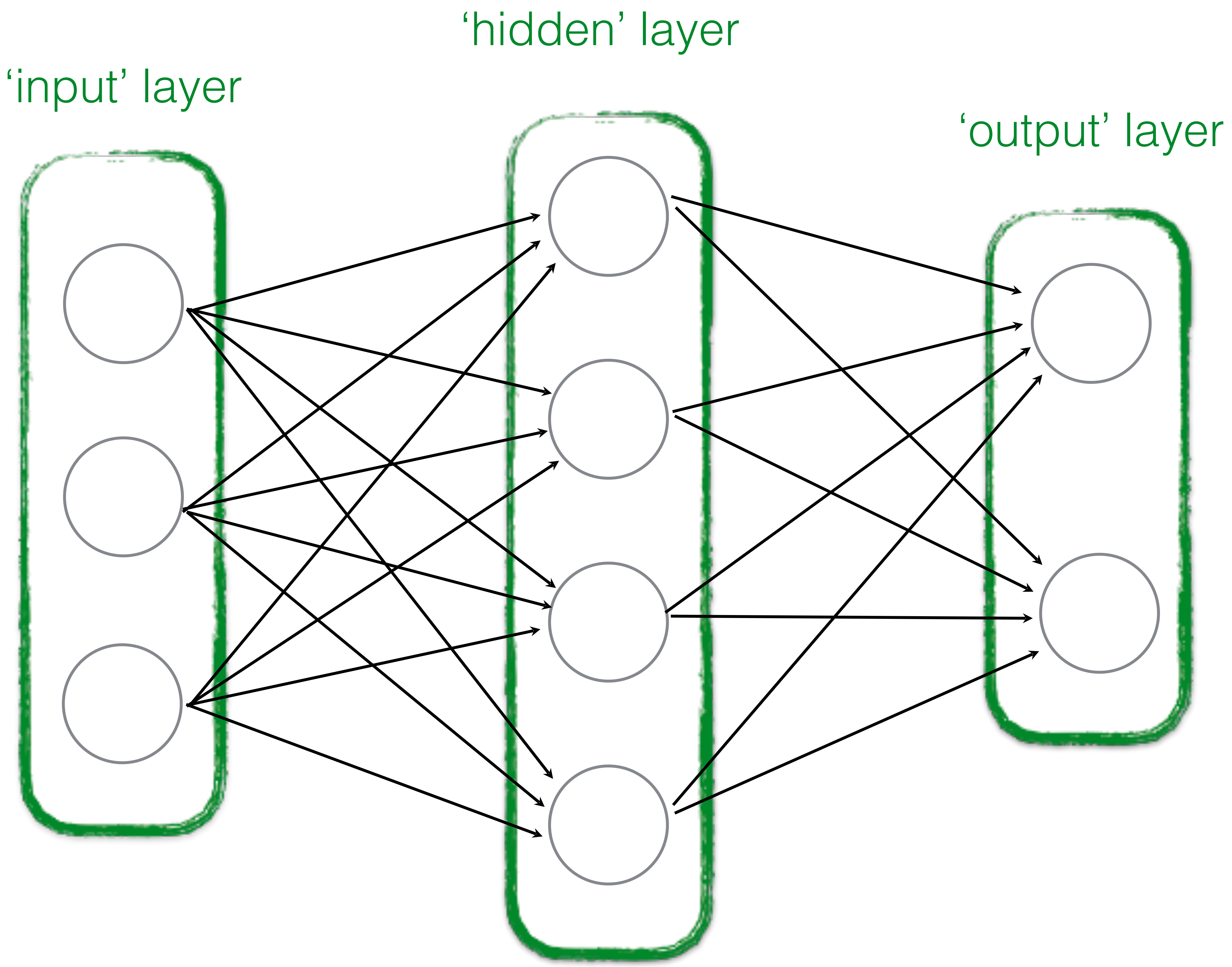
Neural Network

Connect a bunch of perceptrons together …

a collection of connected perceptrons

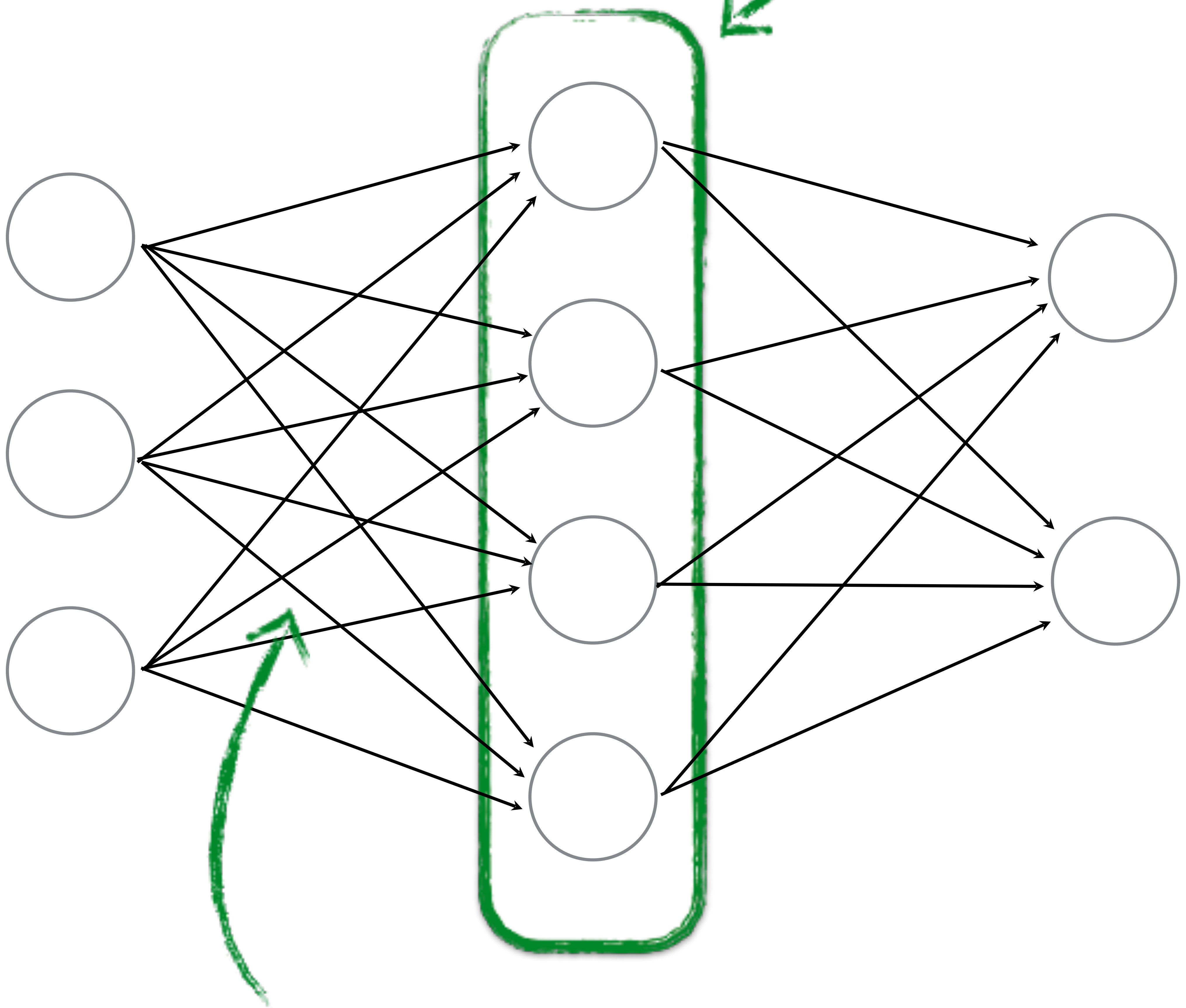


Neural Network

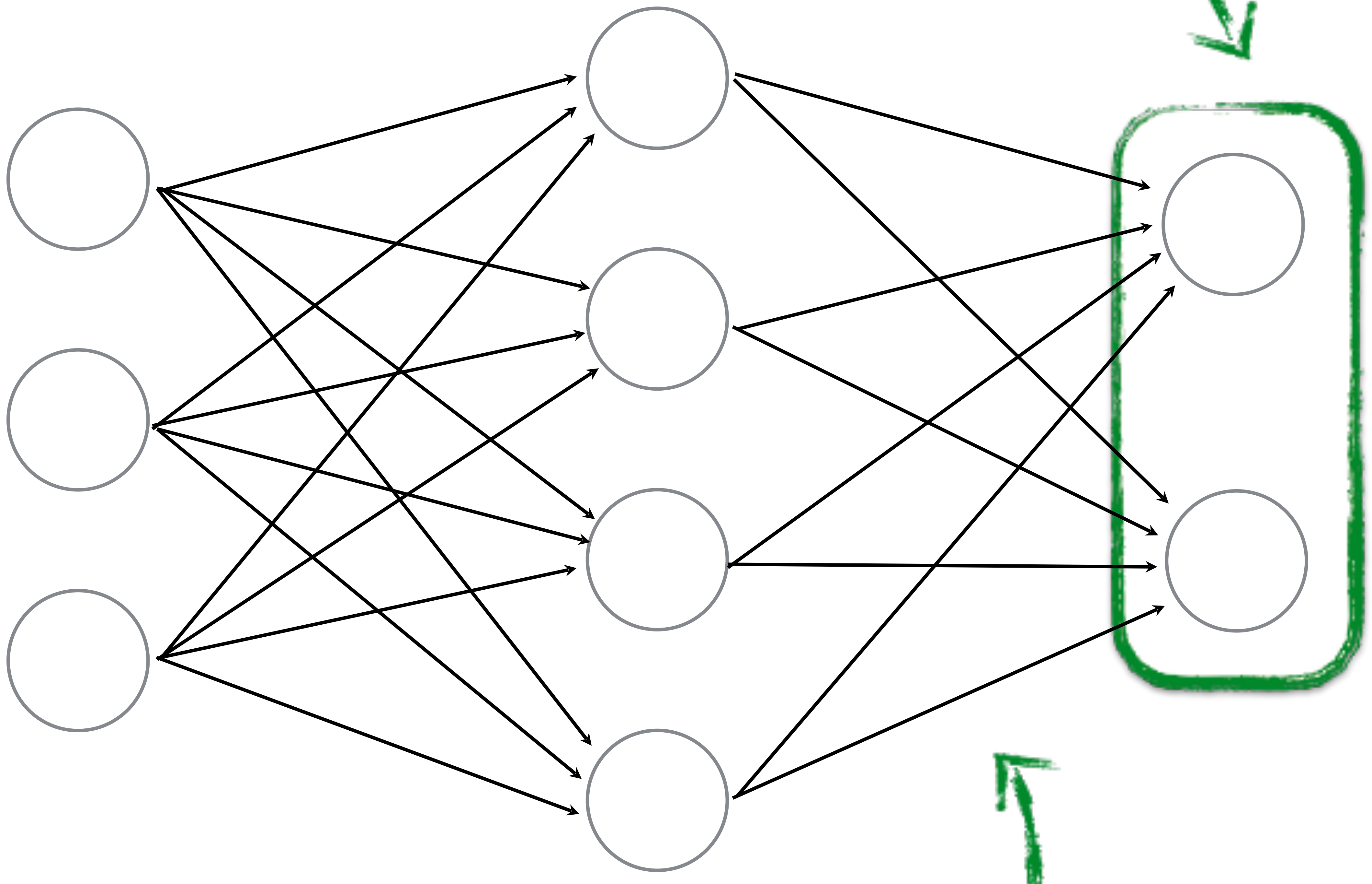


…also called a **Multi-layer Perceptron** (MLP)

this layer is a
'fully connected layer'



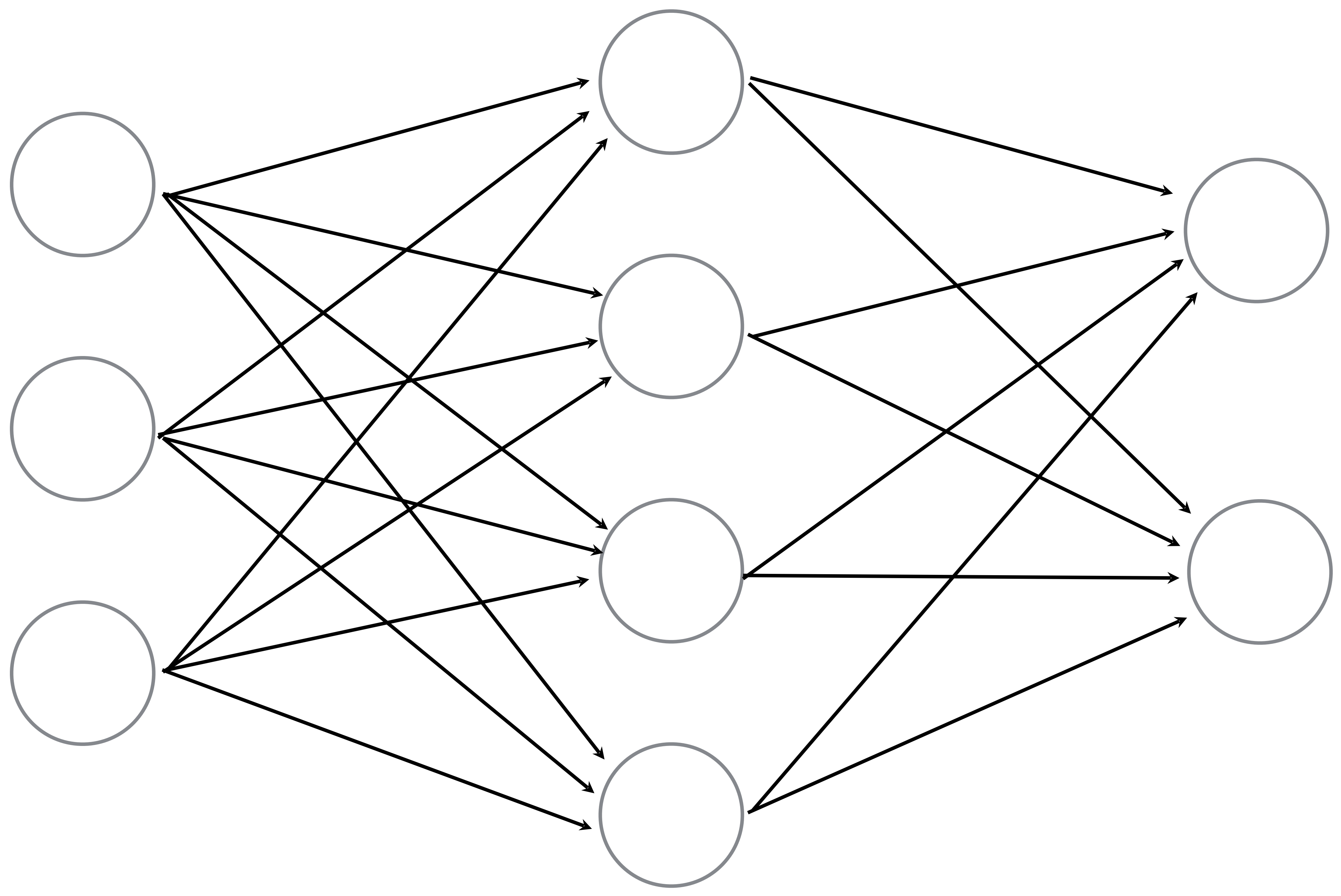
all pairwise neurons between layers are connected so is this



all pairwise neurons <u>between</u> layers are connected

How many neurons (perceptrons)?

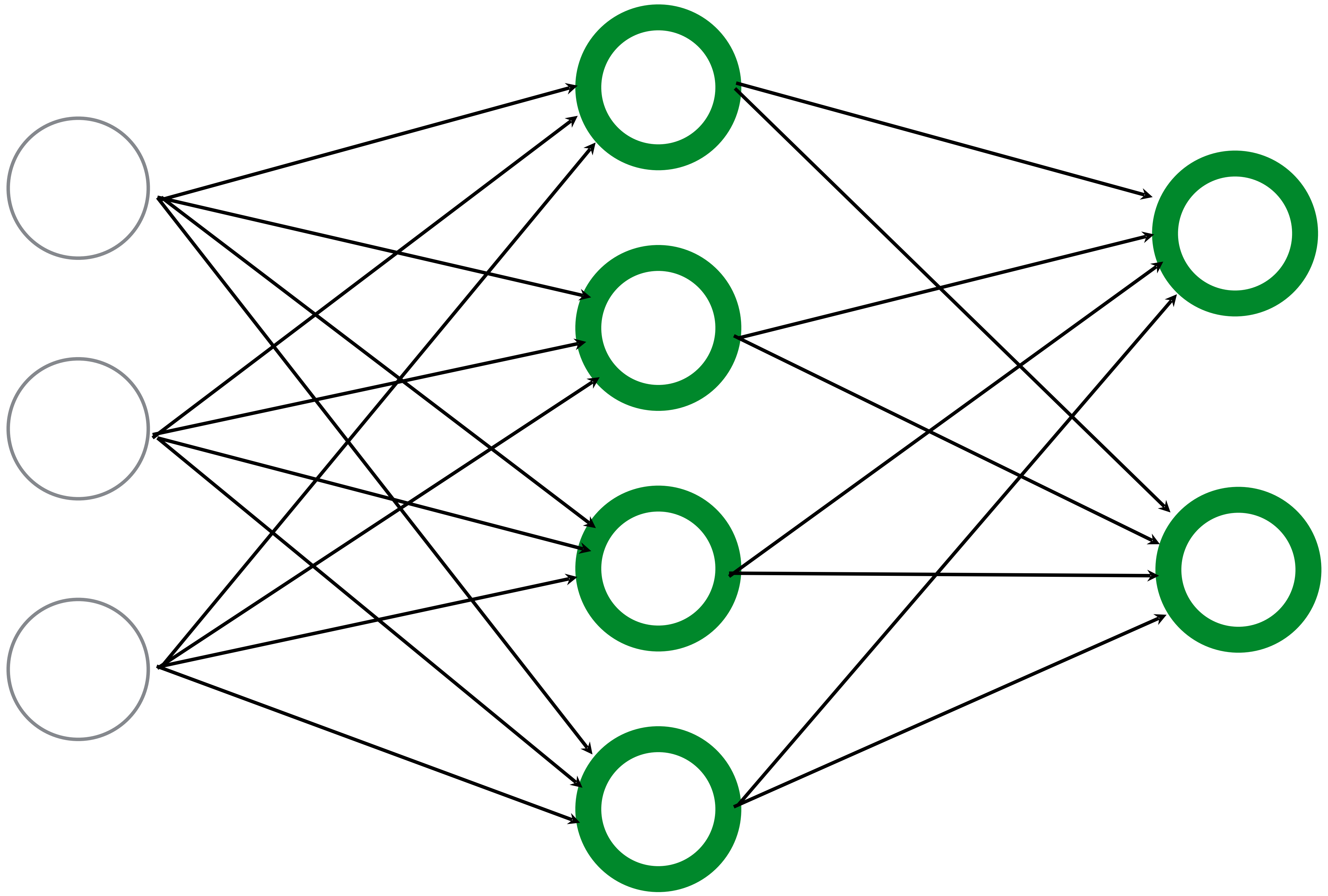
How many weights (edges)?



How many learnable parameters total?

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

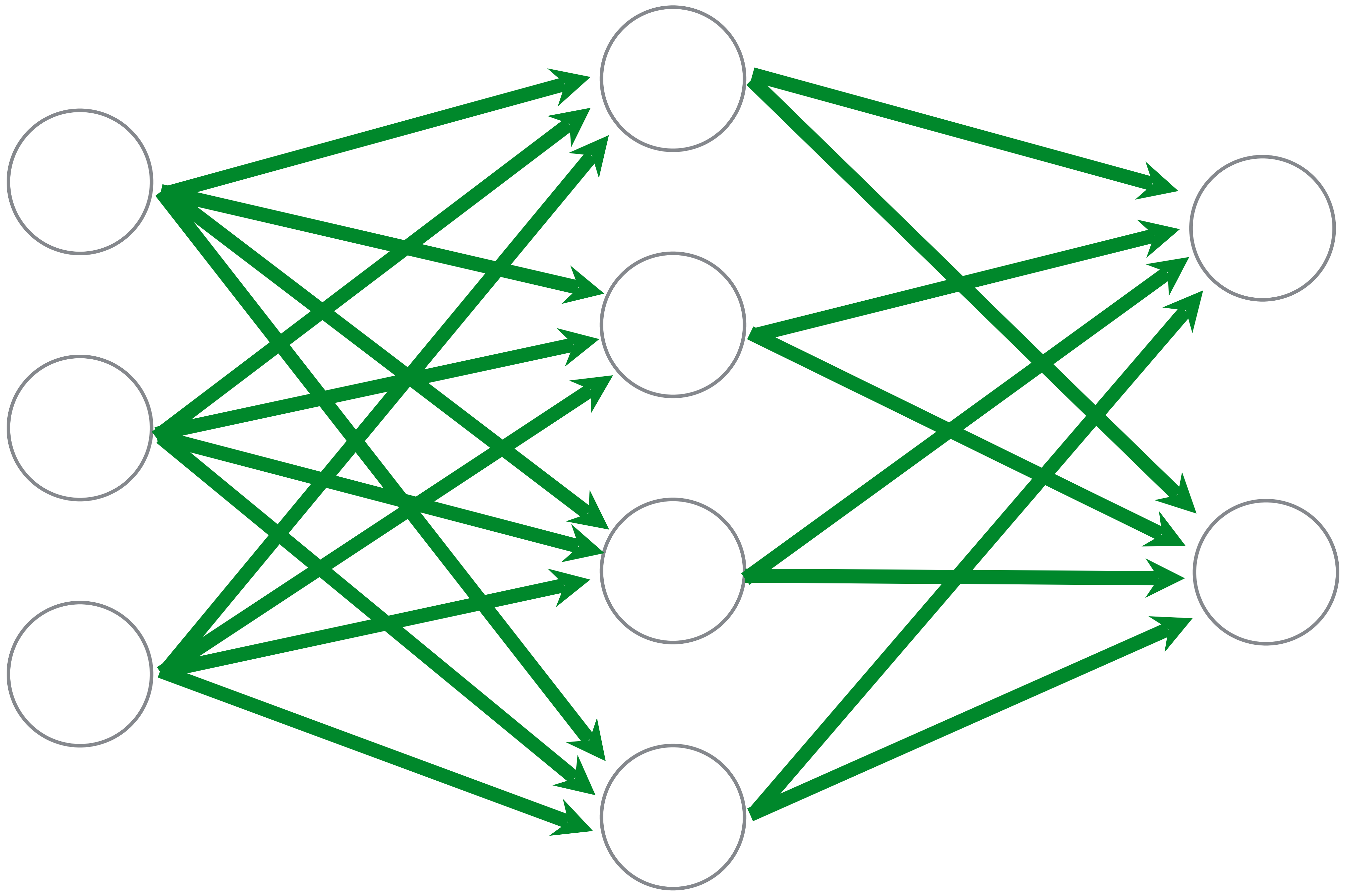
How many weights (edges)?



How many learnable parameters total?

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

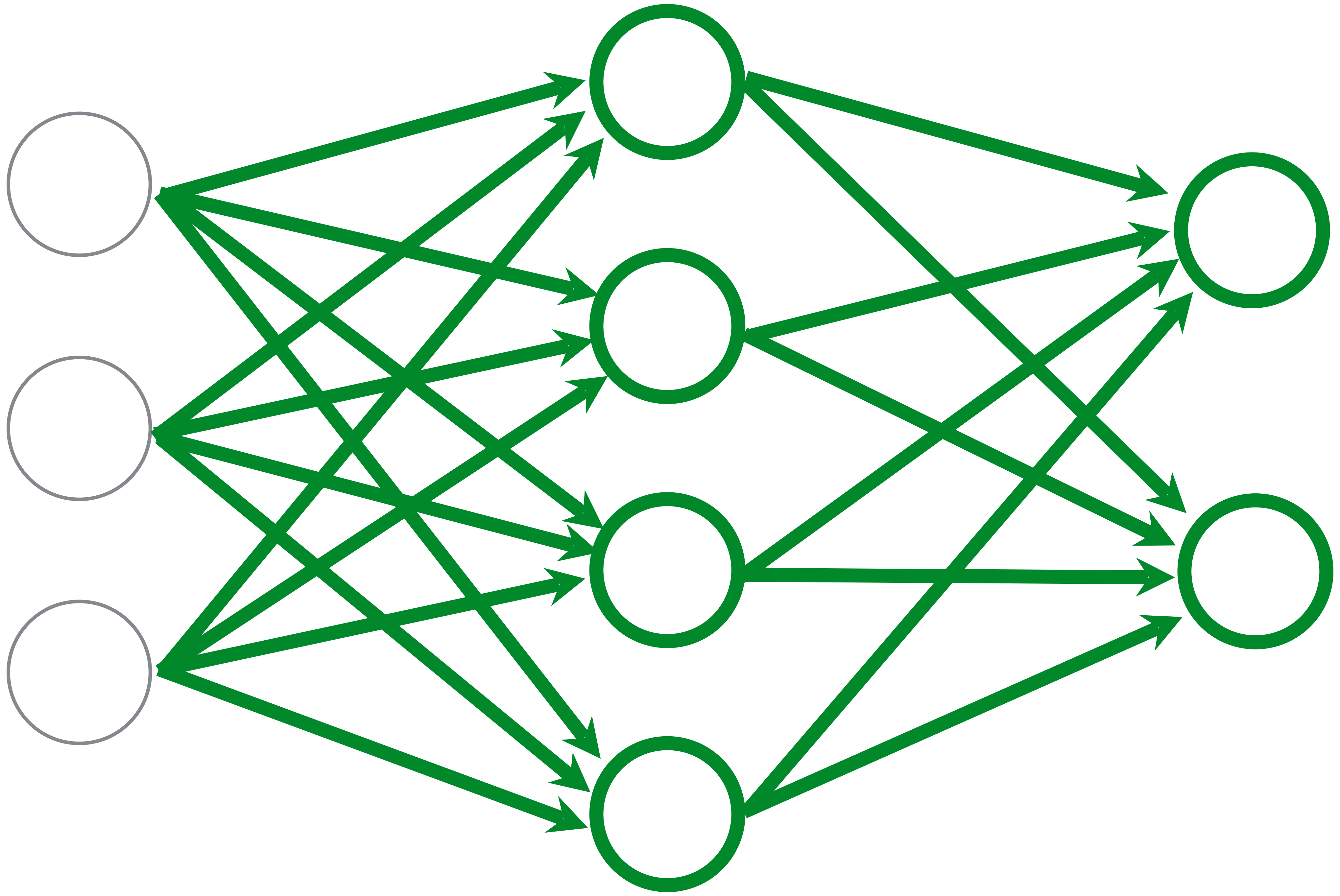
How many weights (edges)? (3 x 4) + (4 x 2) = 20



How many learnable parameters total?

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

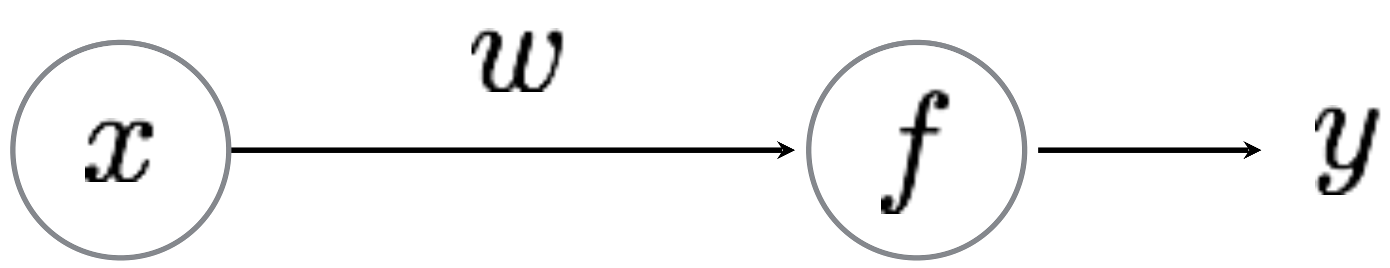
How many weights (edges)? $(3 \times 4) + (4 \times 2) = 20$



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

How to train perceptrons?

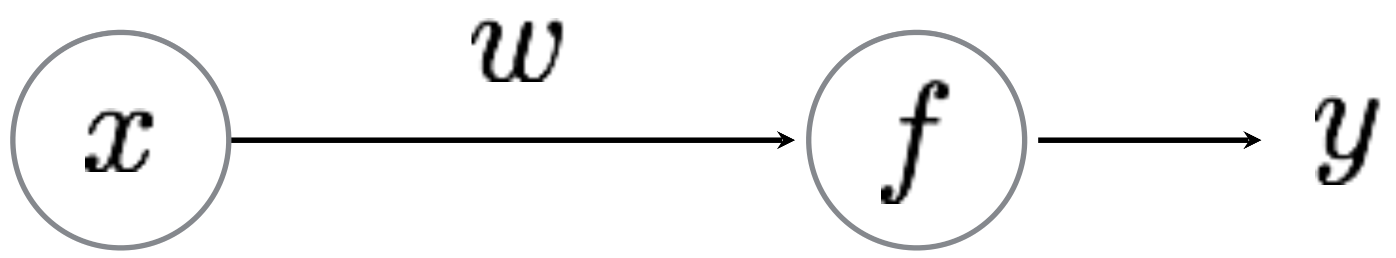
world's smallest perceptron!



$$y = wx$$

What does this look like?

world’s smallest perceptron!



$$y = wx$$

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

Learning a Perceptron

Given a set of samples and a Perceptron

$$\{x_i, y_i\}$$

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

Estimate the parameters 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**

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

what does this mean?

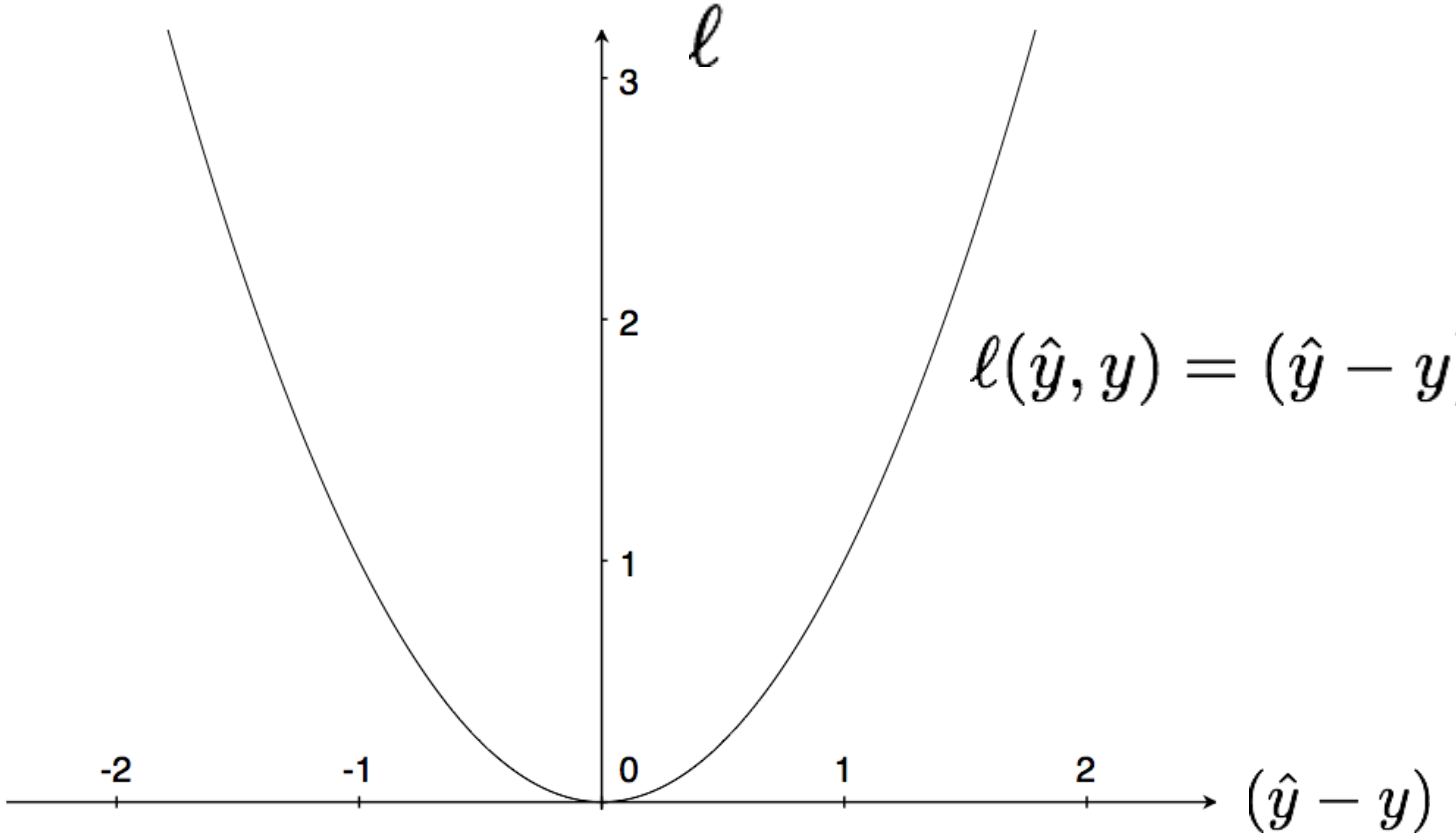
true label

Loss Function
defines what is means to be
close to the true solution

YOU get to chose the loss function!
(some are better than others depending on what you want to do)

Squared Error (L2)

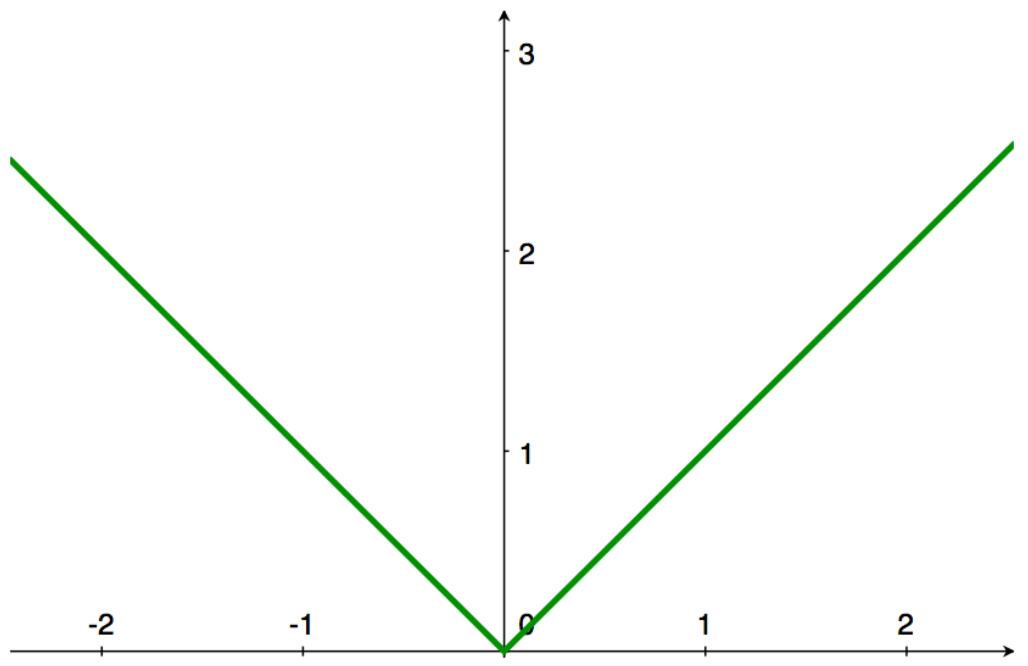
(a popular loss function) ((why?))



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

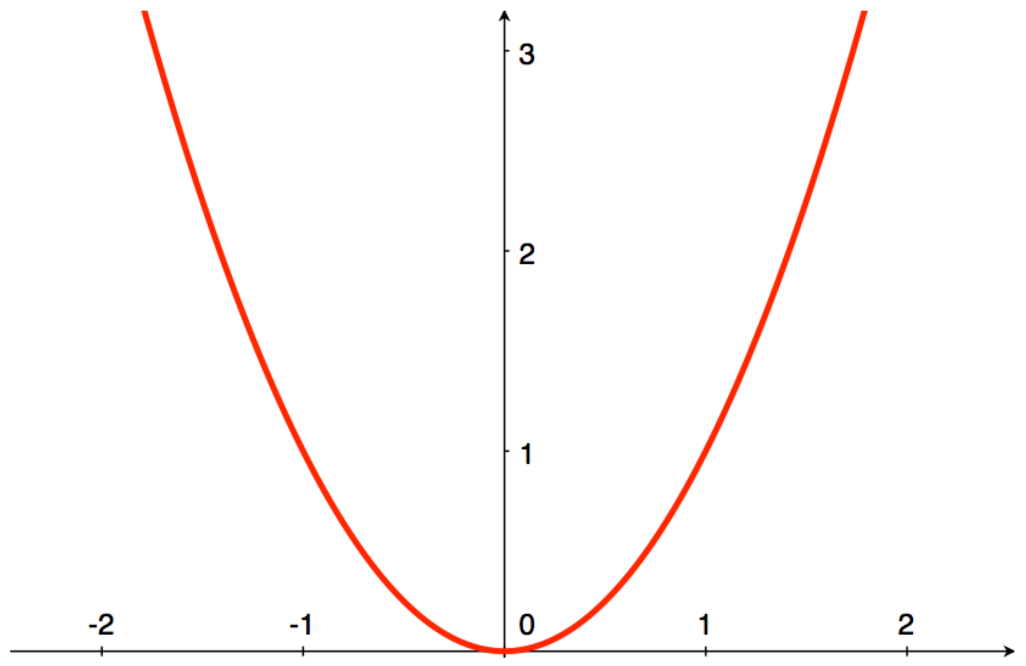
L1 Loss

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



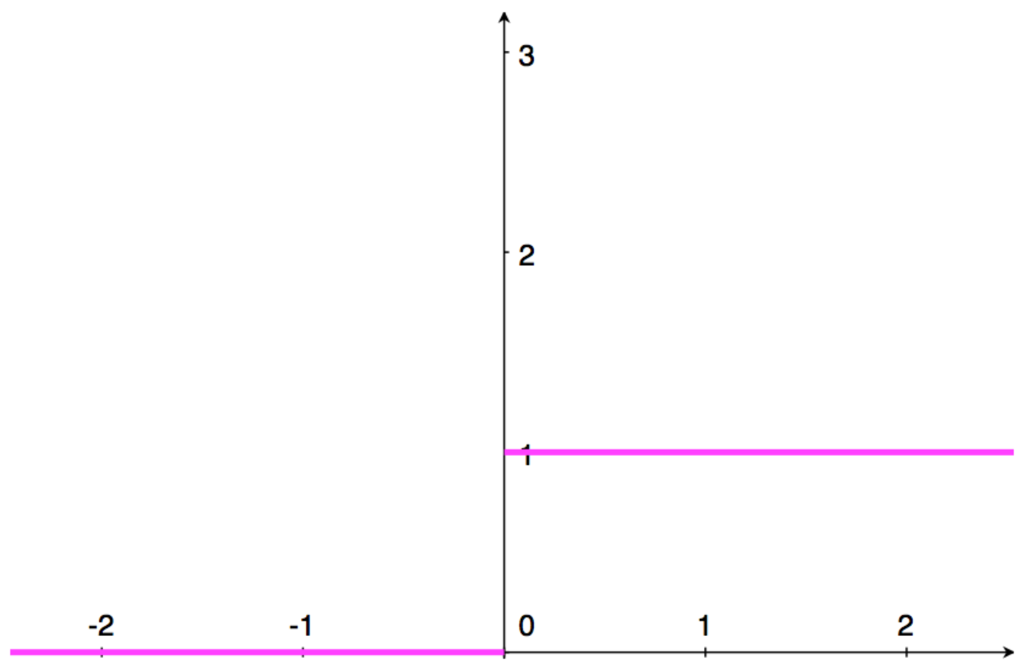
L2 Loss

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



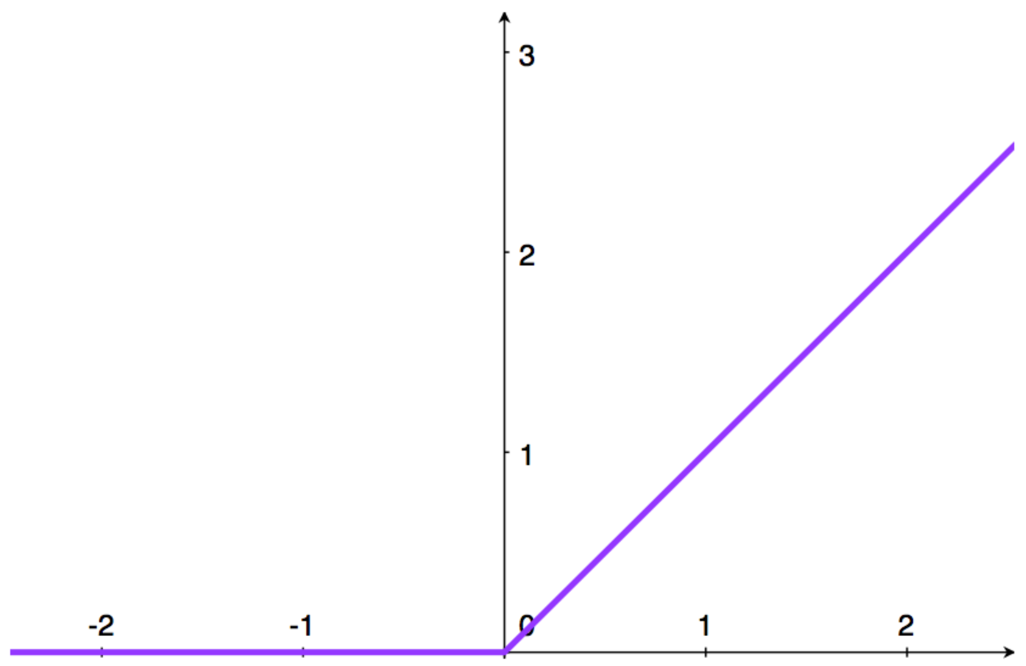
Zero-One Loss

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



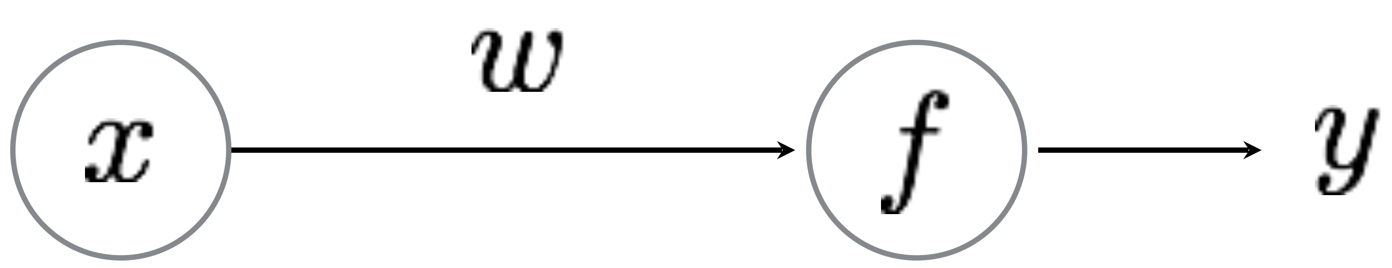
Hinge Loss

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



back to the…

world’s smallest perceptron!



$$y = wx$$

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

function of **ONE** parameter!

Learning a Perceptron

Given a set of samples and a Perceptron

$$\{x_i, y_i\}$$

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

what is this activation function?

Estimate the parameter of the Perceptron

$$w$$

Learning a Perceptron

Given a set of samples and a Perceptron

$$\{x_i, y_i\}$$

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

what is this activation function?

linear function! $f(x) = wx$

Estimate the parameter of the Perceptron

$$w$$

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

Code to train your perceptron:

$$\text{for } \; n = 1 \ldots N$$
$$\quad w = w + (y_n - \hat{y})x_n;$$

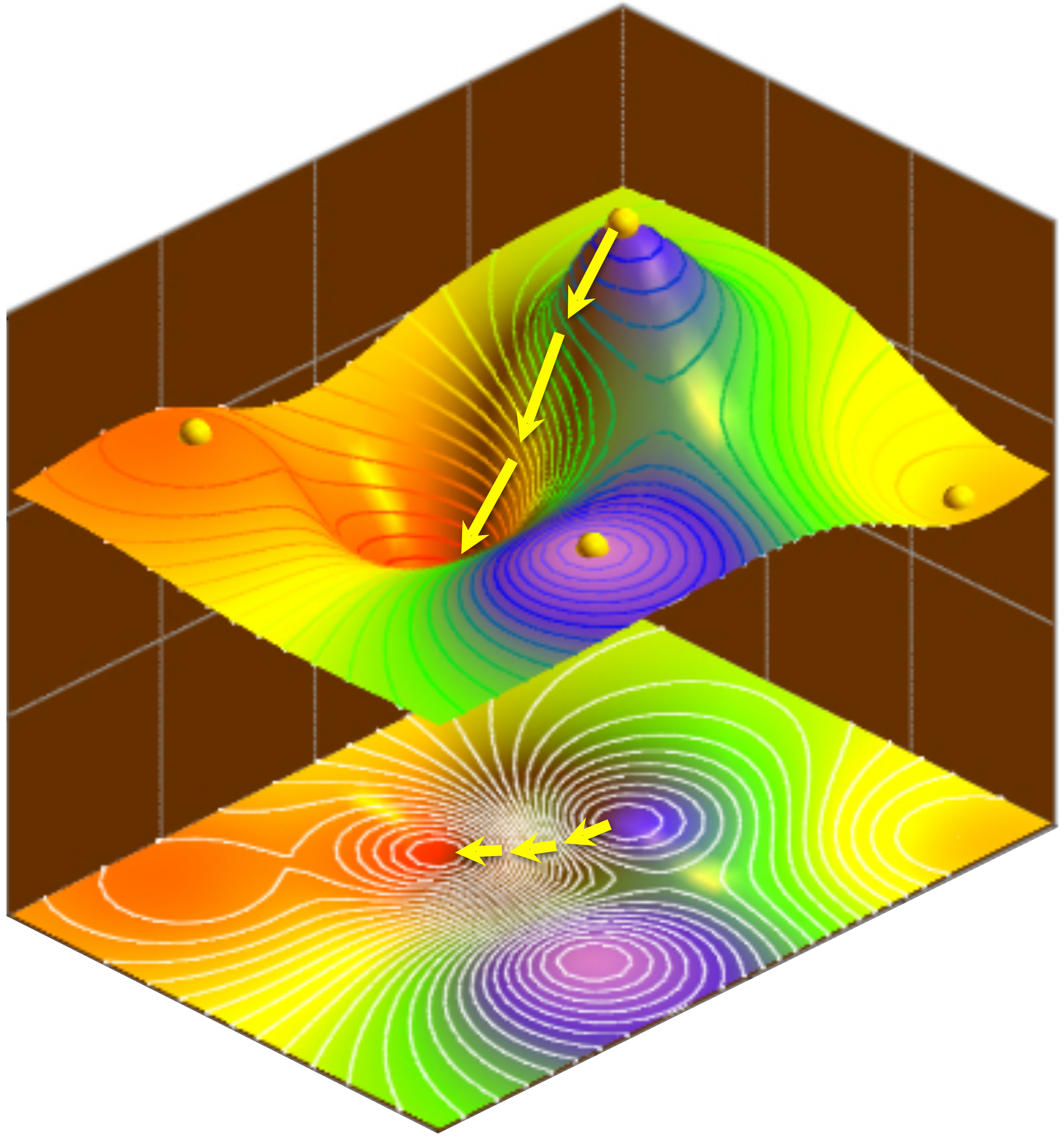
just one line of code!

Gradient descent

(partial) derivatives tell us how much one variable affects another

Gradient descent

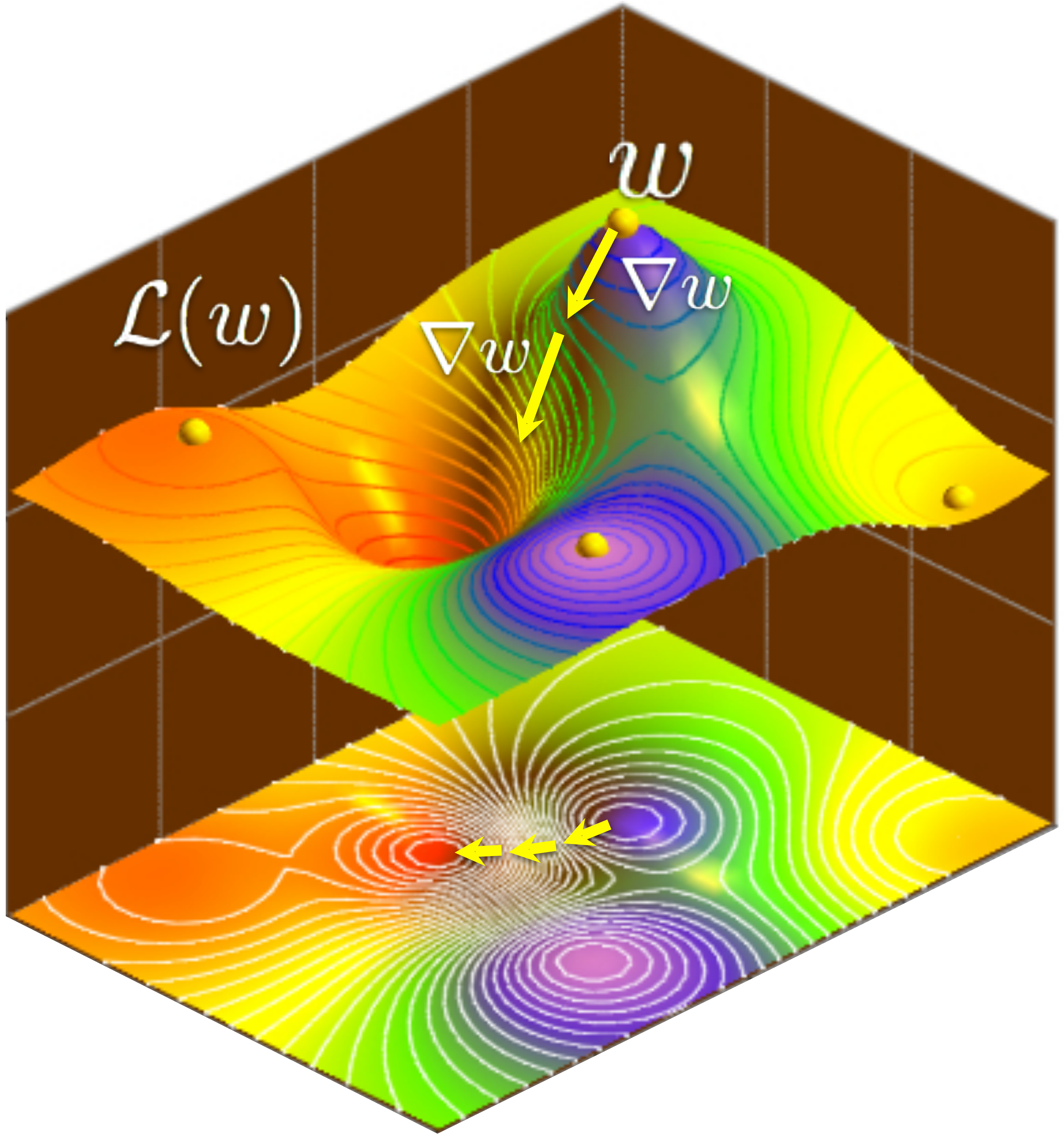
Given a fixed-point or a function, move in the direction opposite of the gradient



Gradient descent

update rule:

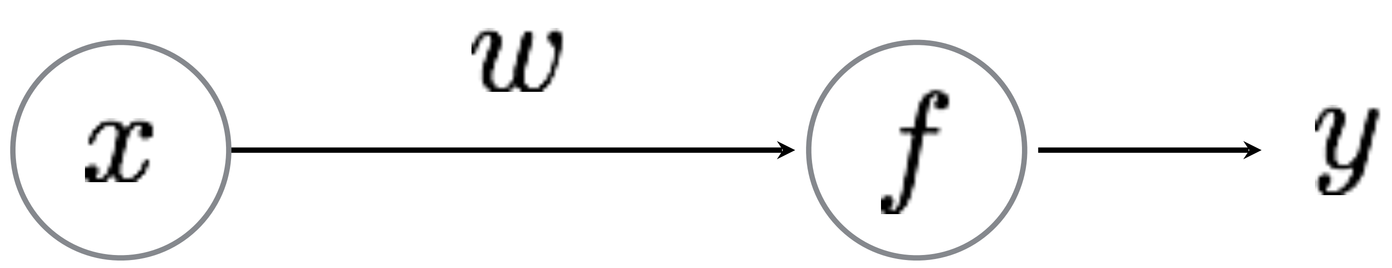
$$\boxed{w = w - \nabla w}$$



Backpropagation

back to the…

World’s Smallest Perceptron!



$$y = wx$$

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

function of **ONE** parameter!

Training the world's smallest perceptron

$$\text{for } n = 1 \ldots N$$

$$w = w + \underline{(y_n - \hat{y})x_n};$$

This is just gradient descent, that means…

this should be the gradient of the loss function

Now where does this come from?

$$\frac{d\mathcal{L}}{dw}$$ …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

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

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

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

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

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

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

Gradient Descent (world's smallest perceptron)

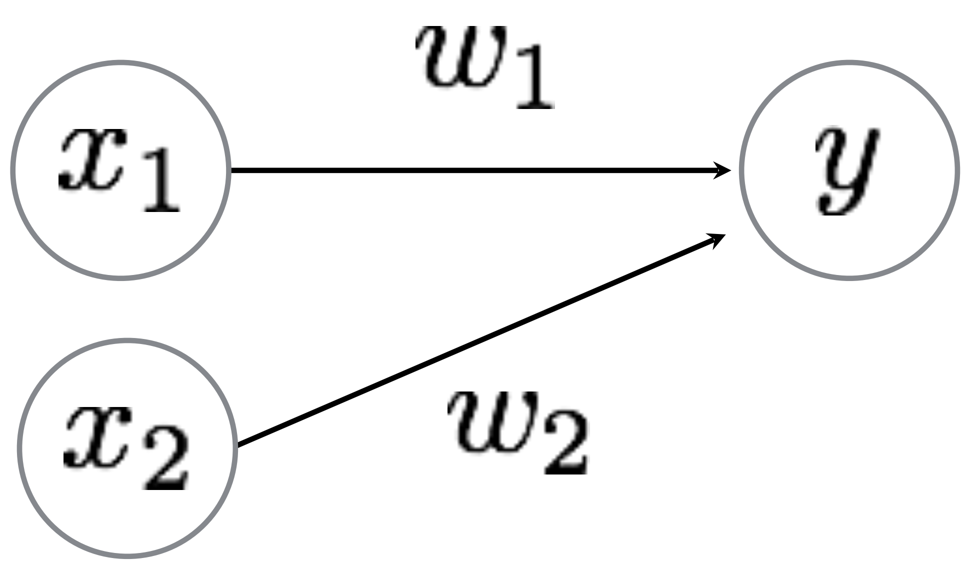
For each sample $\{x_i, y_i\}$

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 $w = w - \nabla w$

Training the world's smallest perceptron

$$\text{for } n = 1 \ldots N$$
$$w = w + (y_n - \hat{y})x_n;$$

world's (second) smallest perceptron!



function of **two** parameters!

Gradient Descent

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

1. Predict
   a. Forward pass
   b. Compute Loss
2. Update
   a. Back Propagation
   b. Gradient update

we just need to compute partial derivatives for this network

Derivative computation

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_1} &= \frac{\partial}{\partial w_1}\left\{\frac{1}{2}(y-\hat{y})^2\right\} \\ &= -(y-\hat{y})\frac{\partial \hat{y}}{\partial w_1} \\ &= -(y-\hat{y})\frac{\partial \sum_i w_i x_i}{\partial w_1} \\ &= -(y-\hat{y})\frac{\partial w_1 x_1}{\partial w_1} \\ &= -(y-\hat{y})x_1 = \nabla w_1\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_2} &= \frac{\partial}{\partial w_2}\left\{\frac{1}{2}(y-\hat{y})^2\right\} \\ &= -(y-\hat{y})\frac{\partial \hat{y}}{\partial w_2} \\ &= -(y-\hat{y})\frac{\partial \sum_i w_i x_i}{\partial w_1} \\ &= -(y-\hat{y})\frac{\partial w_2 x_2}{\partial w_2} \\ &= -(y-\hat{y})x_2 = \nabla w_2\end{aligned}$$

Why do we have partial derivatives now?

Derivative computation

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_1} &= \frac{\partial}{\partial w_1}\left\{\frac{1}{2}(y-\hat{y})^2\right\} \\ &= -(y-\hat{y})\frac{\partial \hat{y}}{\partial w_1} \\ &= -(y-\hat{y})\frac{\partial \sum_i w_i x_i}{\partial w_1} \\ &= -(y-\hat{y})\frac{\partial w_1 x_1}{\partial w_1} \\ &= -(y-\hat{y})x_1 = \nabla w_1\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_2} &= \frac{\partial}{\partial w_2}\left\{\frac{1}{2}(y-\hat{y})^2\right\} \\ &= -(y-\hat{y})\frac{\partial \hat{y}}{\partial w_2} \\ &= -(y-\hat{y})\frac{\partial \sum_i w_i x_i}{\partial w_1} \\ &= -(y-\hat{y})\frac{\partial w_2 x_2}{\partial w_2} \\ &= -(y-\hat{y})x_2 = \nabla w_2\end{aligned}$$

Gradient Update

$$\begin{aligned}w_1 &= w_1 - \eta \nabla w_1 \\ &= w_1 + \eta(y-\hat{y})x_1\end{aligned}$$

$$\begin{aligned}w_2 &= w_2 - \eta \nabla w_2 \\ &= w_2 + \eta(y-\hat{y})x_2\end{aligned}$$

Gradient Descent

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

1. Predict
   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$
   b. Compute Loss $\mathcal{L}_i = \frac{1}{2}(y_i - \hat{y})$ (side computation to track loss. not needed for backprop)
2. Update
   a. Back Propagation

two lines now

$$\boxed{\begin{aligned}\nabla w_{1i} &= -(y_i - \hat{y})x_{1i} \\ \nabla w_{2i} &= -(y_i - \hat{y})x_{2i}\end{aligned}}$$

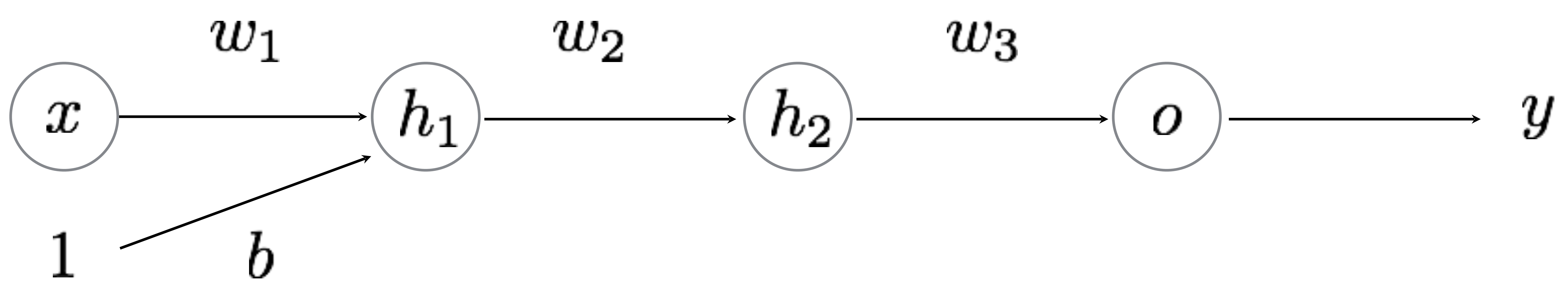
   b. Gradient update

$$\boxed{\begin{aligned}w_{1i} &= w_{1i} + \eta(y - \hat{y})x_{1i} \\ w_{2i} &= w_{2i} + \eta(y - \hat{y})x_{2i}\end{aligned}}$$

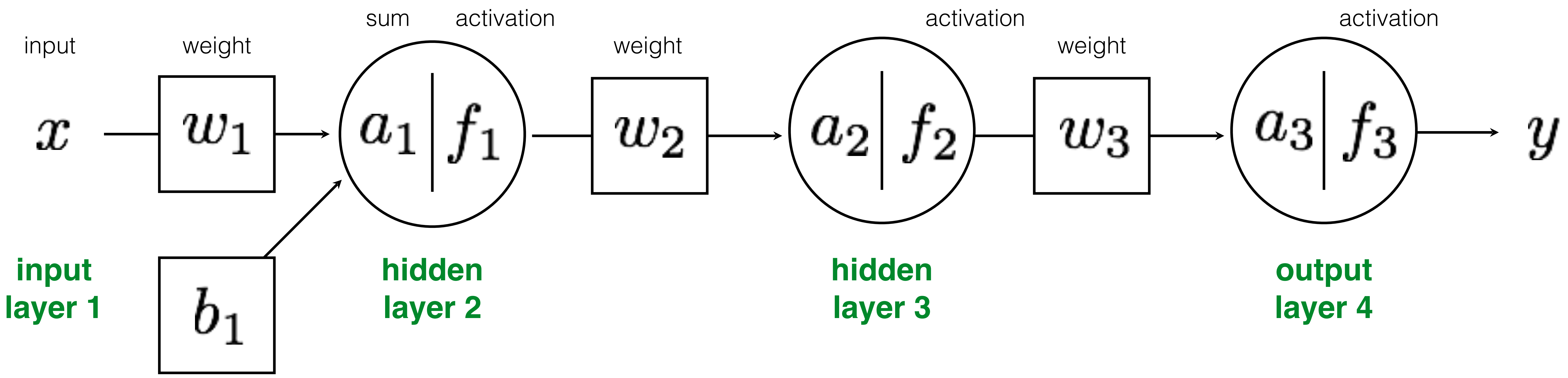
(adjustable step size)

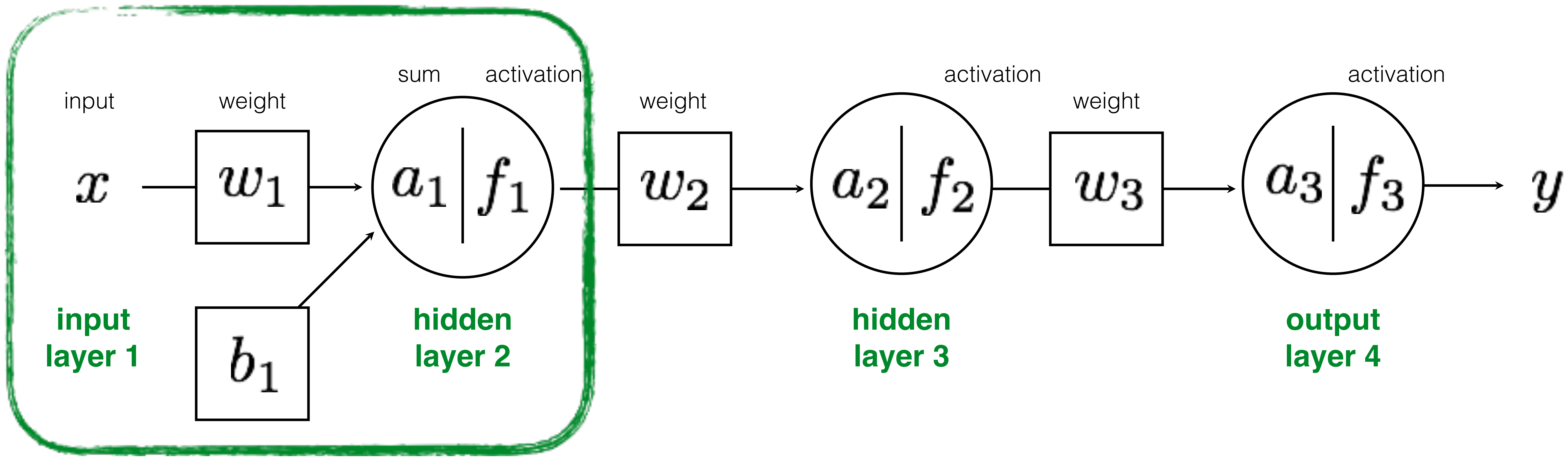
We haven’t seen a lot of ‘propagation’ yet
because our perceptrons only had <u>one</u> layer…

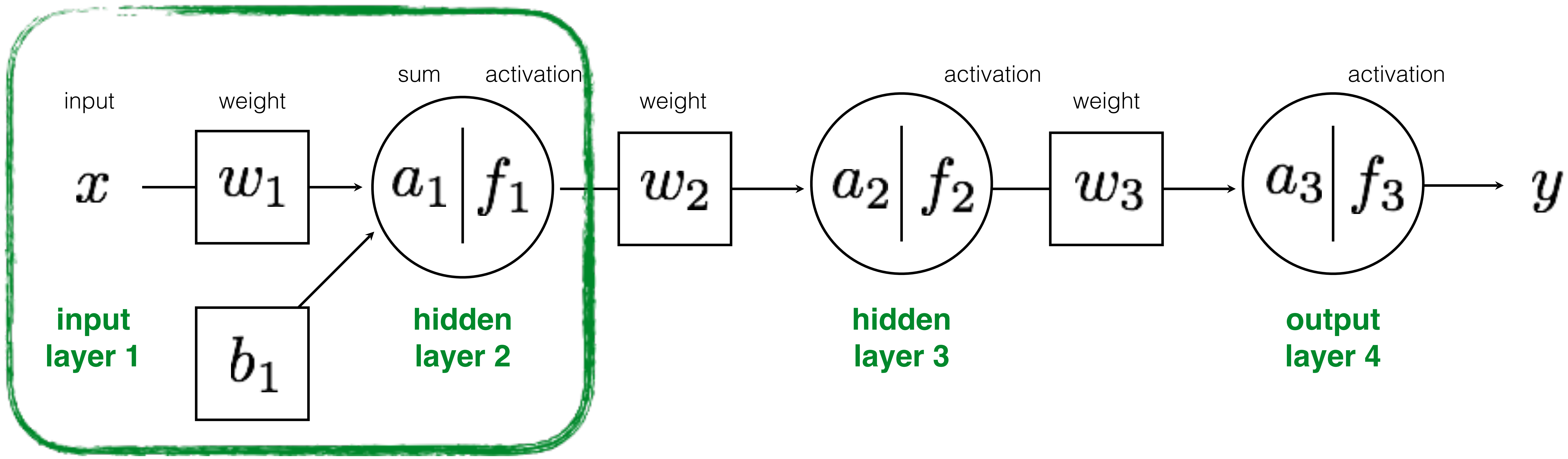
Multi-layer perceptron



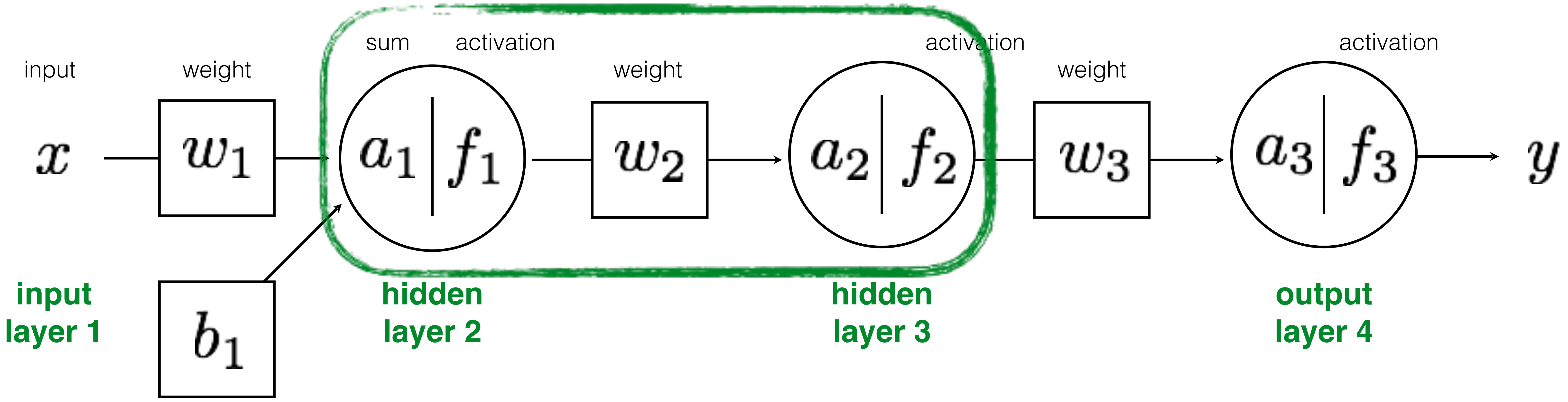
function of **FOUR** parameters and **FOUR** layers!


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

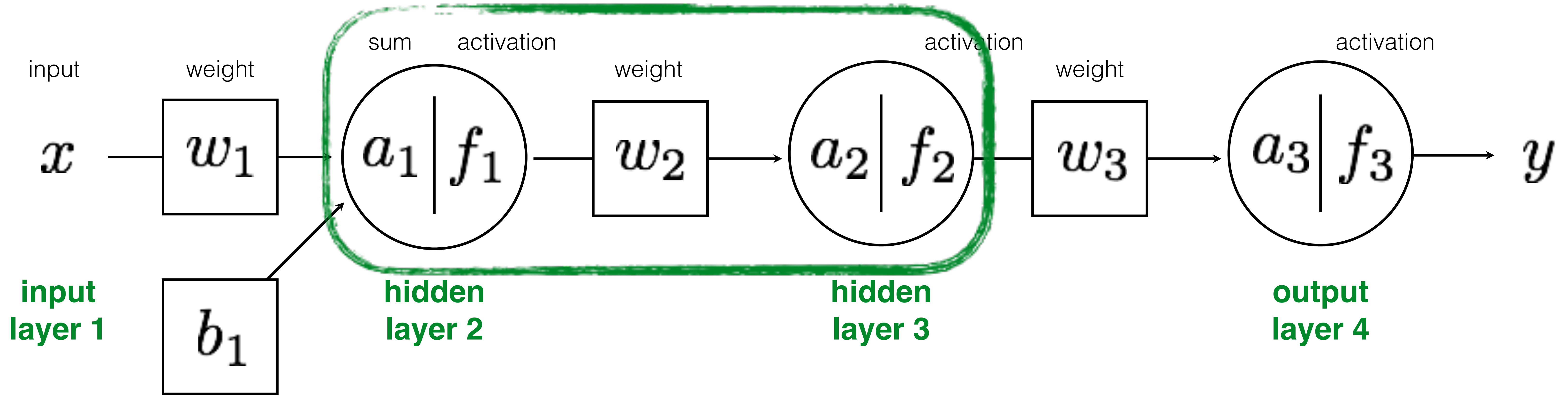

input
weight
sum
activation
weight
activation
weight
activation
x
w_1
a_1 | f_1
w_2
a_2 | f_2
w_3
a_3 | f_3
y
b_1
input layer 1
hidden layer 2
hidden layer 3
output layer 4



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

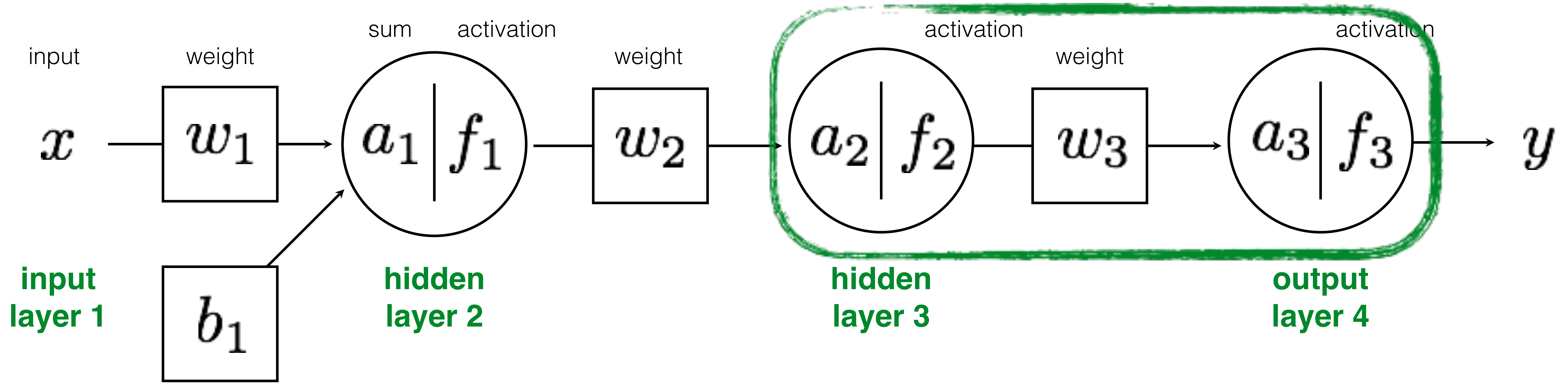


$$a_1 = w_1 \cdot x + b_1$$



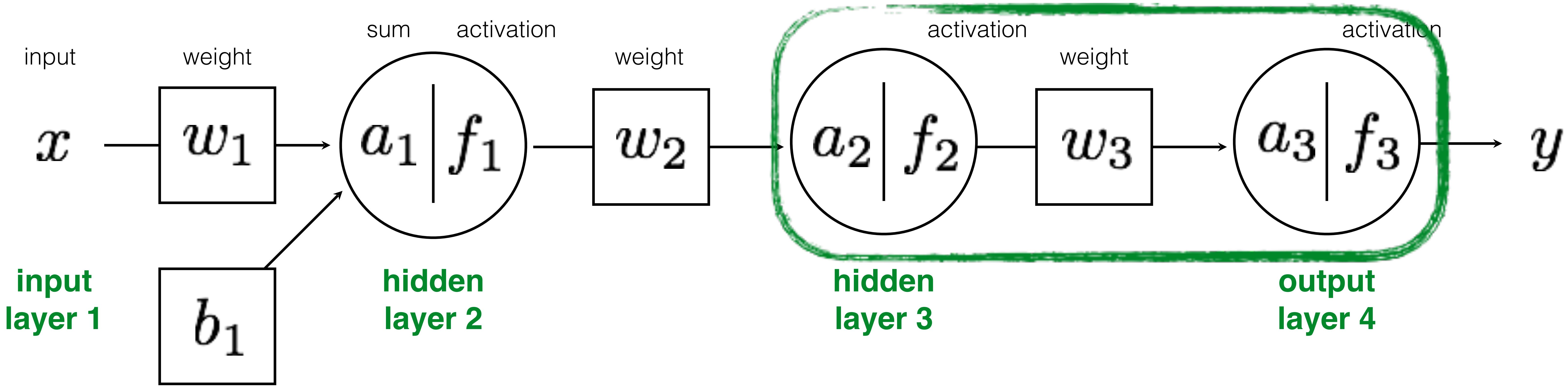
$$a_1 = w_1 \cdot x + b_1$$

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



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

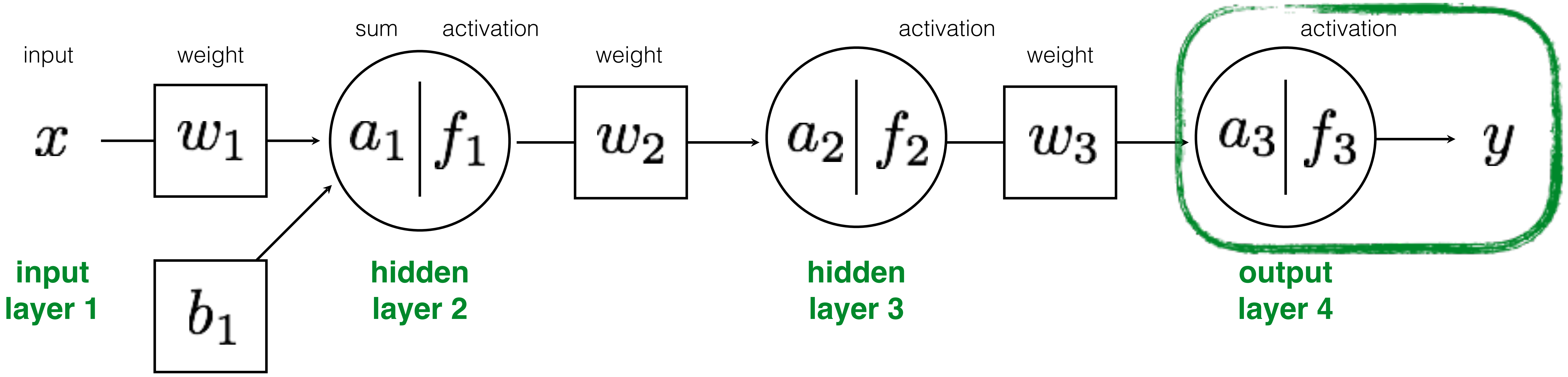
$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$



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

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

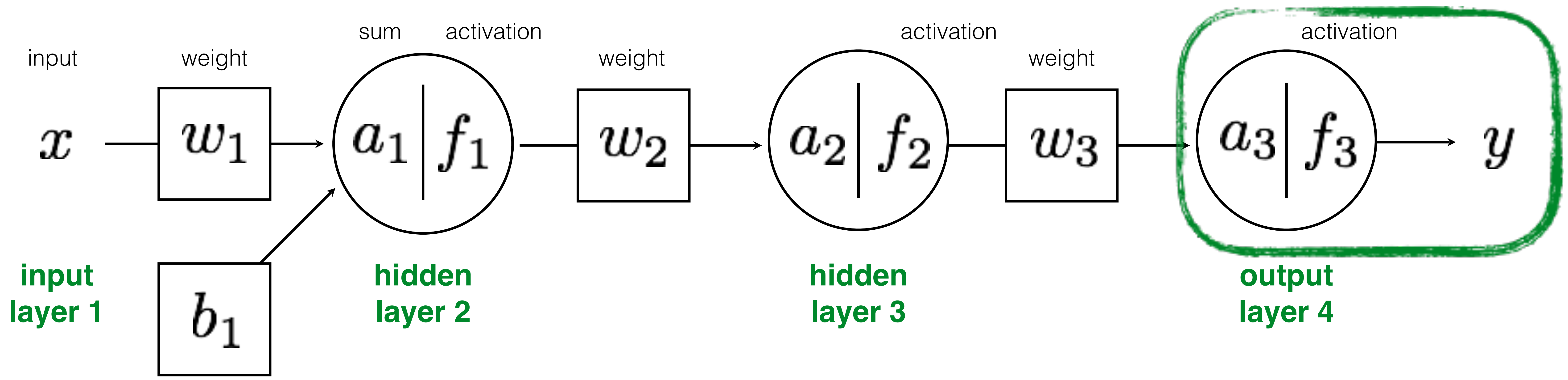
$$a_3 = w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1))$$



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

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

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



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

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

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

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

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 a 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 a 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 $\boxed{\frac{\partial \mathcal{L}}{\partial \theta}}$ vector of parameter partial derivatives
   b. Gradient update $\boxed{\theta \leftarrow \theta - \eta \nabla \theta}$

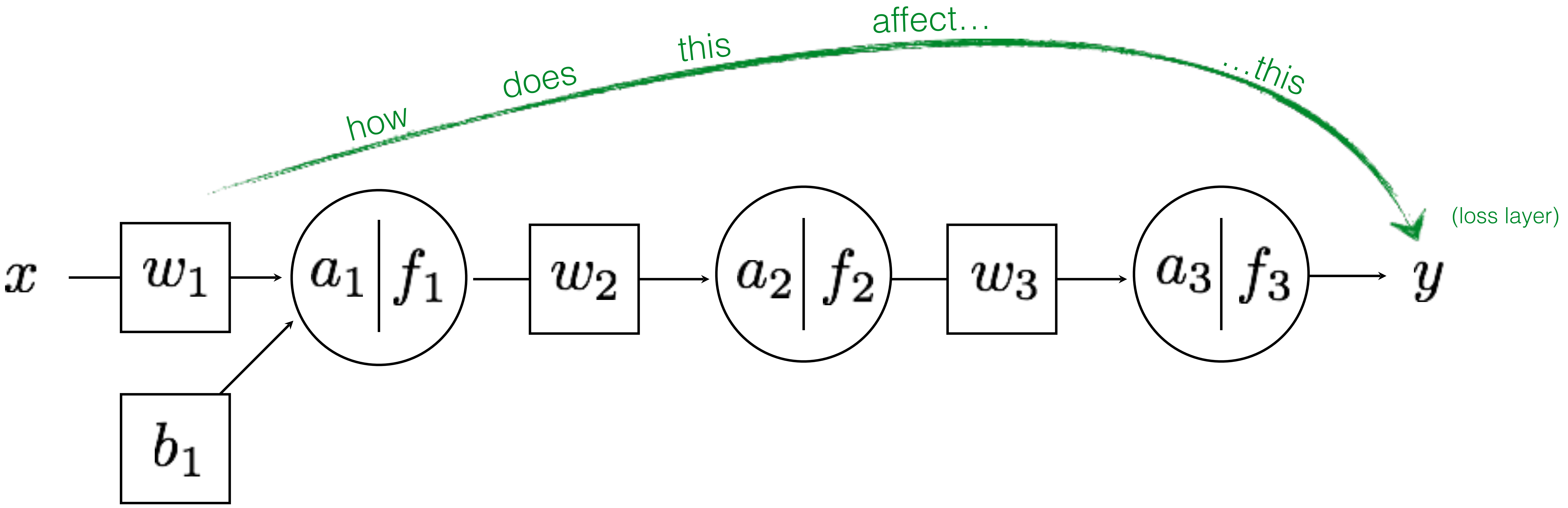
vector of parameter update equations

So we need to compute the partial derivatives

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

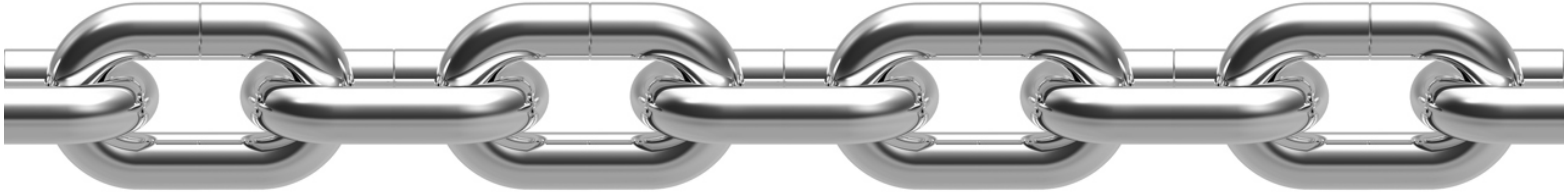
Remember,

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



So, how do you compute it?

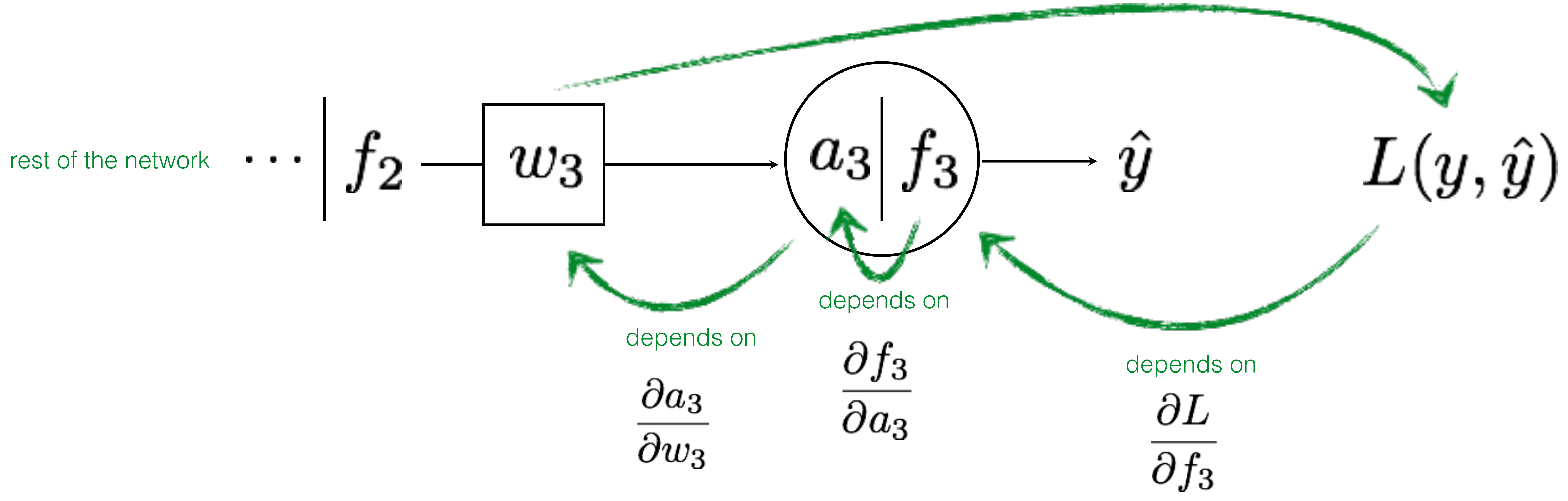
The Chain Rule

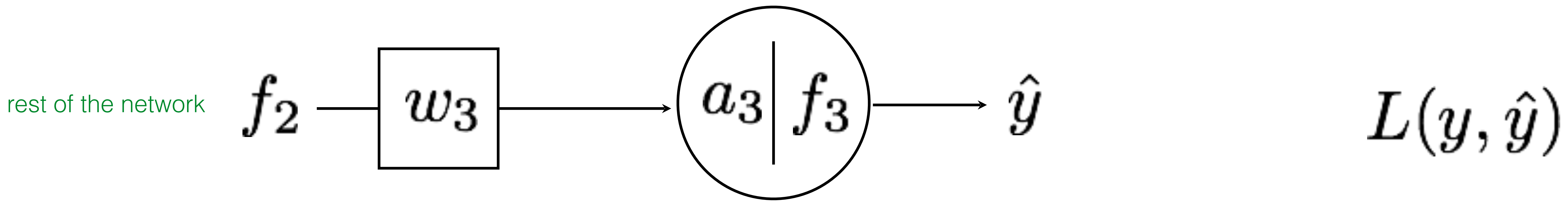


According to the chain rule…

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

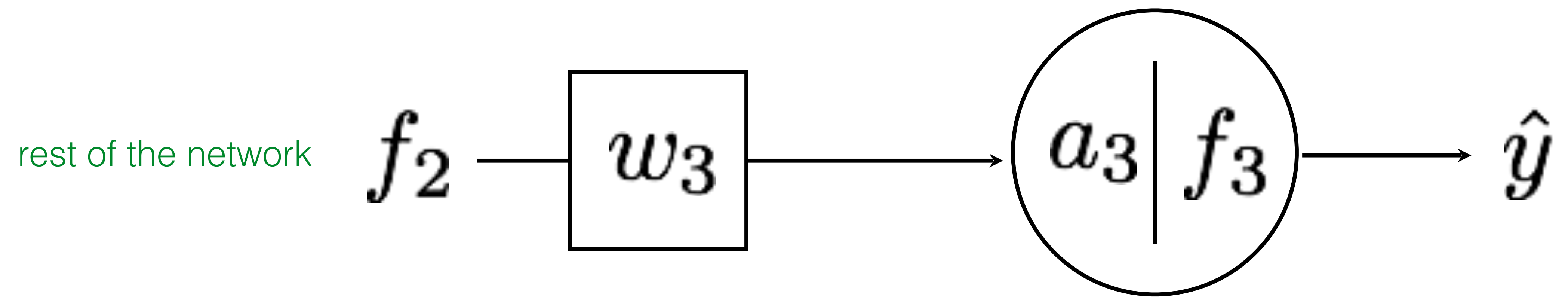
Intuitively, the effect of weight on loss function : $\frac{\partial L}{\partial w_3}$





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

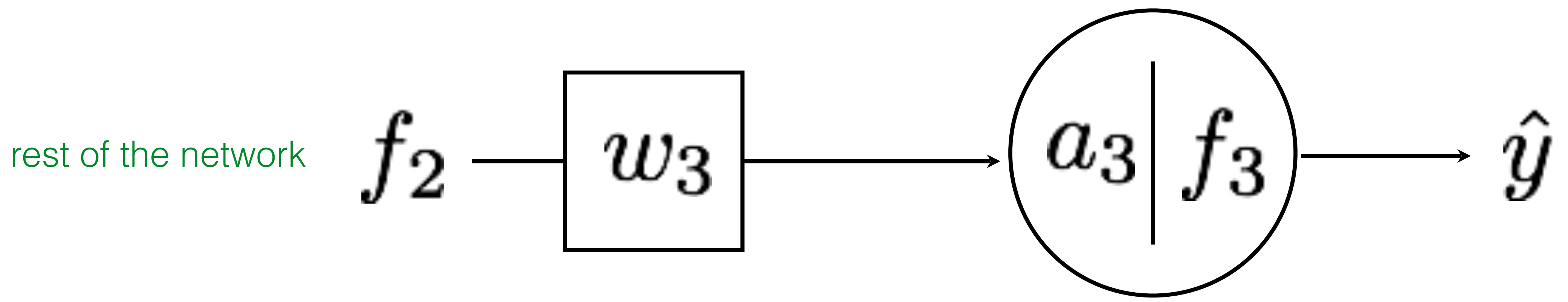
Chain Rule!



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

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$
$$= -\eta(y - \hat{y})\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

Just the partial
derivative of L2 loss

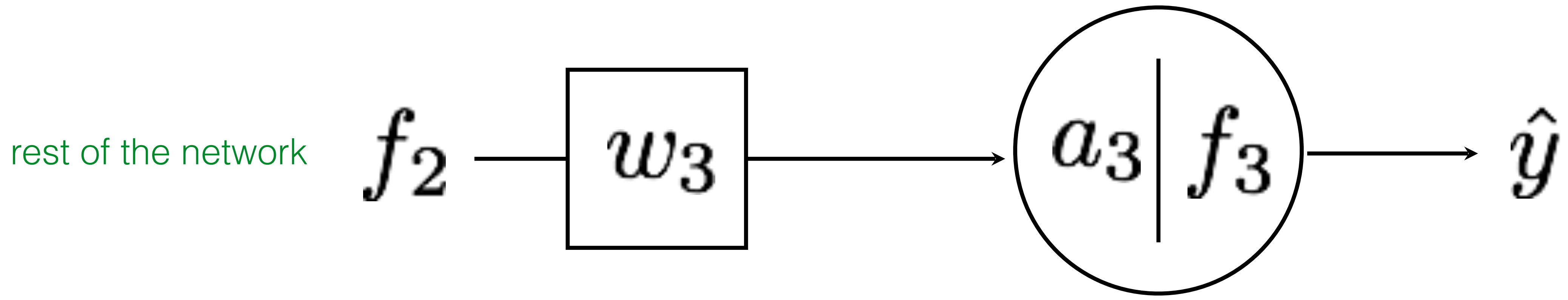


$$L(y, \hat{y})$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$
$$= -\eta(y - \hat{y})\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

Let's use a Sigmoid function

$$\frac{ds(x)}{dx} = s(x)(1 - s(x))$$

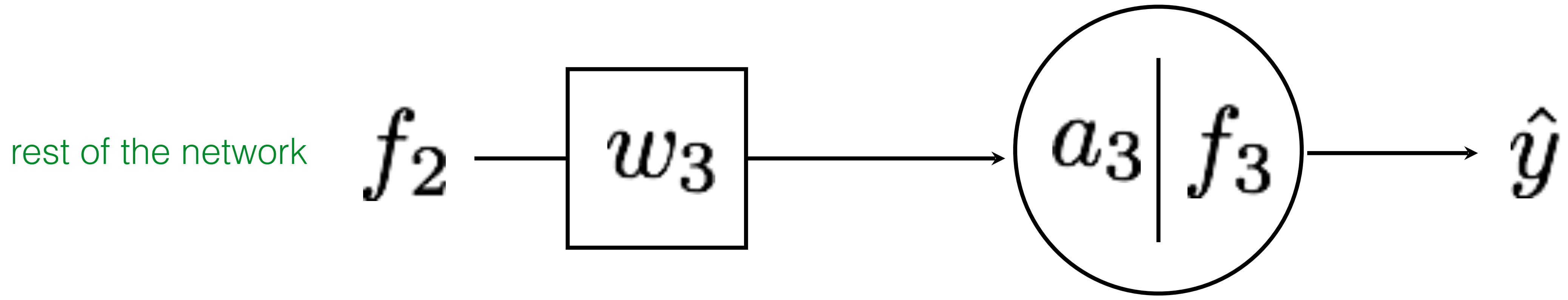


$$L(y, \hat{y})$$

$$\begin{aligned}
\frac{\partial L}{\partial w_3} &= \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3} \\
&= -\eta(y - \hat{y})\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3} \\
&= -\eta(y - \hat{y}) f_3(1 - f_3)\frac{\partial a_3}{\partial w_3}
\end{aligned}$$

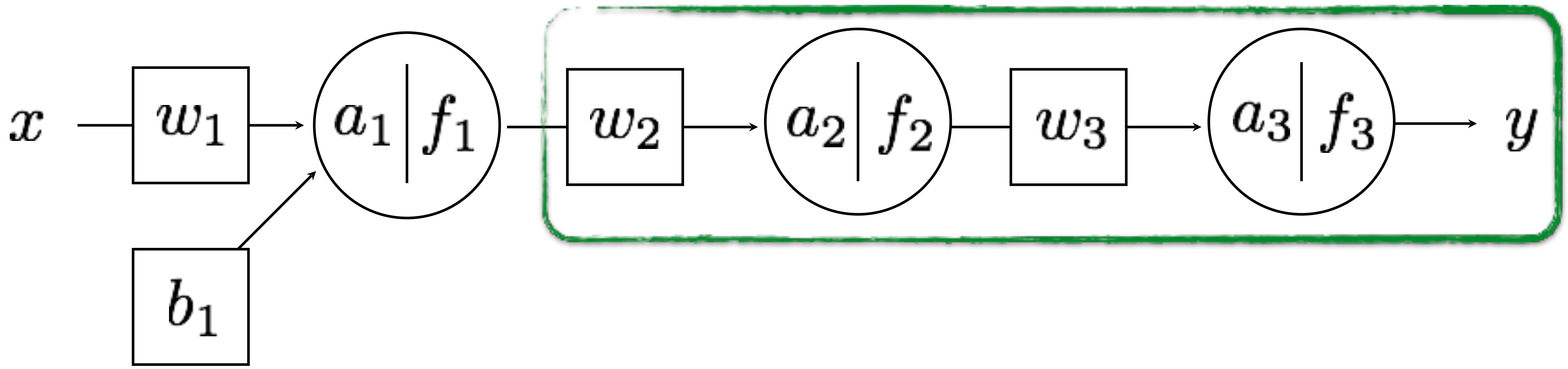
Let's use a Sigmoid function

$$\frac{ds(x)}{dx} = s(x)(1 - s(x))$$

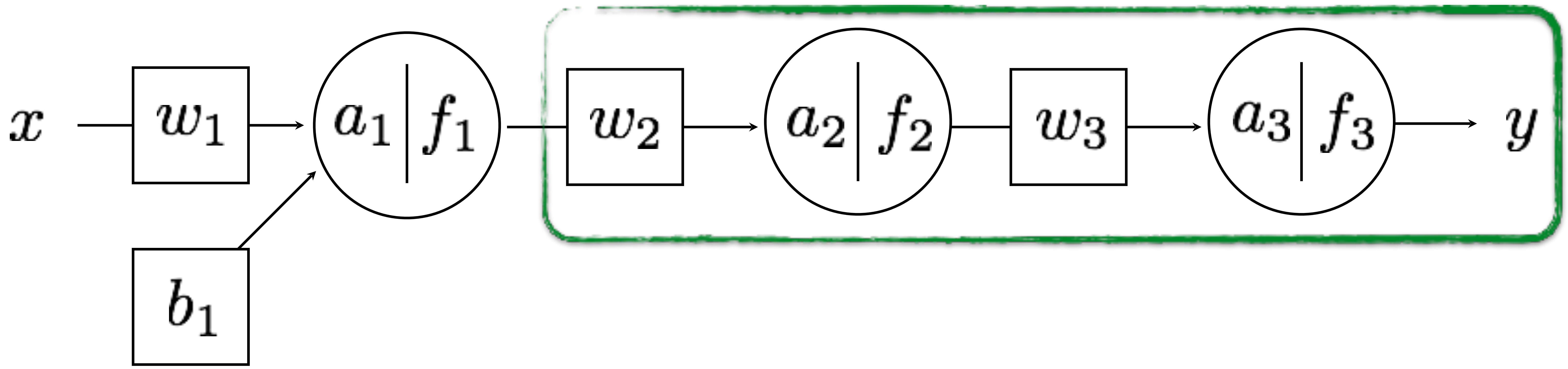


$L(y, \hat{y})$

$$
\begin{aligned}
\frac{\partial L}{\partial w_3} &= \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3} \\
&= -\eta(y - \hat{y})\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3} \\
&= -\eta(y - \hat{y})f_3(1 - f_3)\frac{\partial a_3}{\partial w_3} \\
&= -\eta(y - \hat{y})f_3(1 - f_3)f_2
\end{aligned}
$$



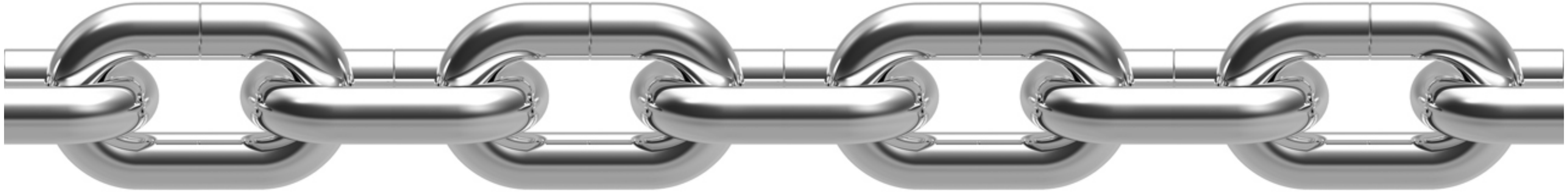
$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial w_2}$$



$$\frac{\partial L}{\partial w_2} = \boxed{\frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial w_2}$$

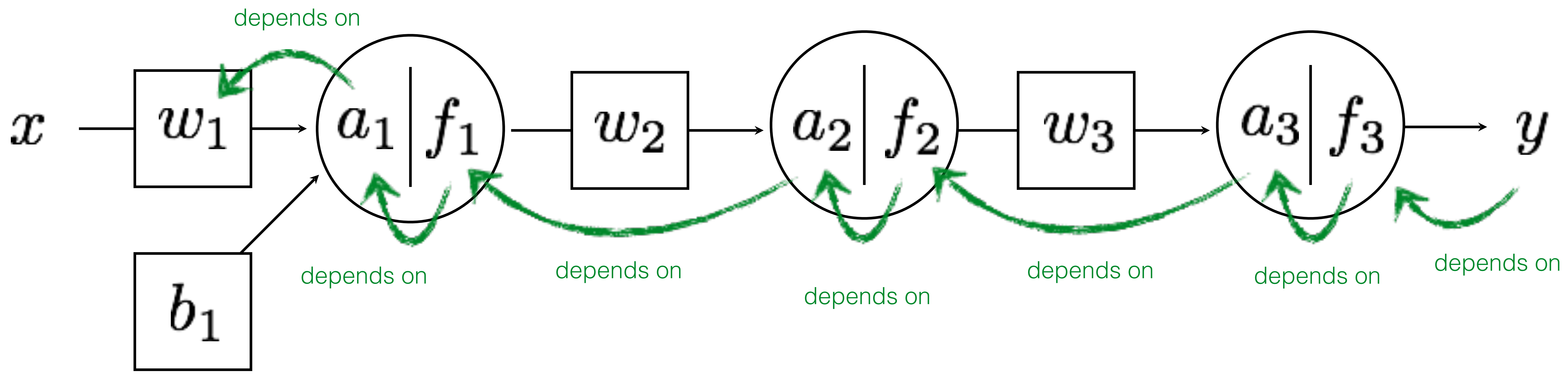
already computed.
re-use (propagate)!

The Chain Rule



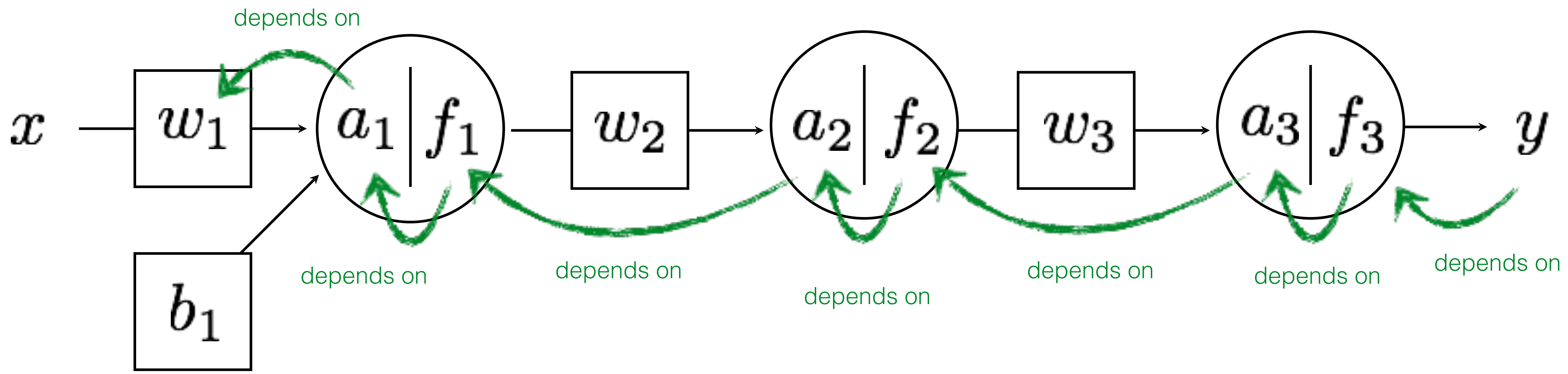
a.k.a. backpropagation

The chain rule says…



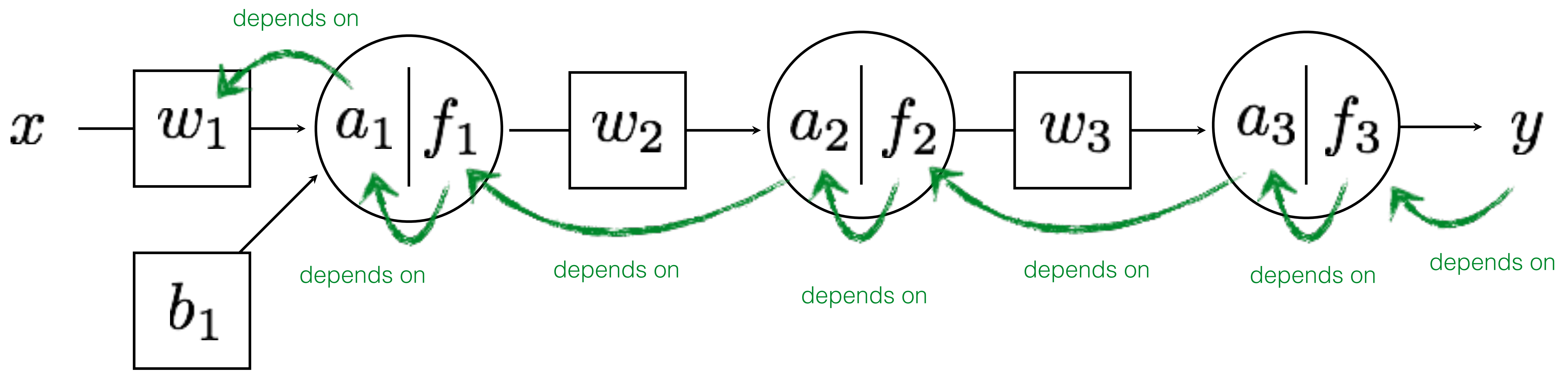
$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

The chain rule says…



$$\frac{\partial L}{\partial w_1} = \boxed{\frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

already computed.
re-use (propagate)!



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

$$\frac{\partial \mathcal{L}}{\partial w_2} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}}\frac{\partial a_2}{\partial w_2}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial b}$$

Gradient Descent

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

1. Predict
   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$
   b. Compute Loss $\mathcal{L}_i$
2. Update
   a. Back Propagation

$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$
$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial w_2}$$
$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$
$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial b}$$

   b. Gradient update

$$w_3 = w_3 - \eta \nabla w_3$$
$$w_2 = w_2 - \eta \nabla w_2$$
$$w_1 = w_1 - \eta \nabla w_1$$
$$b = b - \eta \nabla b$$

Gradient Descent

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

1. Predict
   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$
   b. Compute Loss $\mathcal{L}_i$
2. Update
   a. Back Propagation $\frac{\partial \mathcal{L}}{\partial \theta}$

      vector of parameter partial derivatives

   b. Gradient update $\theta \leftarrow \theta + \eta \frac{\partial \mathcal{L}}{\partial \theta}$

      vector of parameter update equations

Stochastic gradient descent

What we are truly minimizing:

$$\min_{\theta} \sum_{i=1}^{N} L(y_i, f_{MLP}(x_i))$$

The gradient is:

What we are truly minimizing:

$$\min_{\theta} \sum_{i=1}^{N} L(y_i, f_{MLP}(x_i))$$

The gradient is:

$$\sum_{i=1}^{N} \frac{\partial L(y_i, f_{MLP}(x_i))}{\partial \theta}$$

What we use for gradient update is:

What we are truly minimizing:

$$\min_{\theta} \sum_{i=1}^{N} L(y_i, f_{MLP}(x_i))$$

The gradient is:

$$\sum_{i=1}^{N} \frac{\partial L(y_i, f_{MLP}(x_i))}{\partial \theta}$$

What we use for gradient update is:

$$\frac{\partial L(y_i, f_{MLP}(x_i))}{\partial \theta} \quad \text{for some i}$$

Stochastic Gradient Descent

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

1. Predict
   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$
   b. Compute Loss $\mathcal{L}_i$
2. Update
   a. Back Propagation $\frac{\partial \mathcal{L}}{\partial \theta}$

      vector of parameter partial derivatives

   b. Gradient update $\theta \leftarrow \theta + \eta \frac{\partial \mathcal{L}}{\partial \theta}$

      vector of parameter update equations

How do we select which sample?

- Select randomly!

Do we need to use only one sample?

- You can use a *minibatch* of size $B < N$.

Why not do gradient descent with all samples?

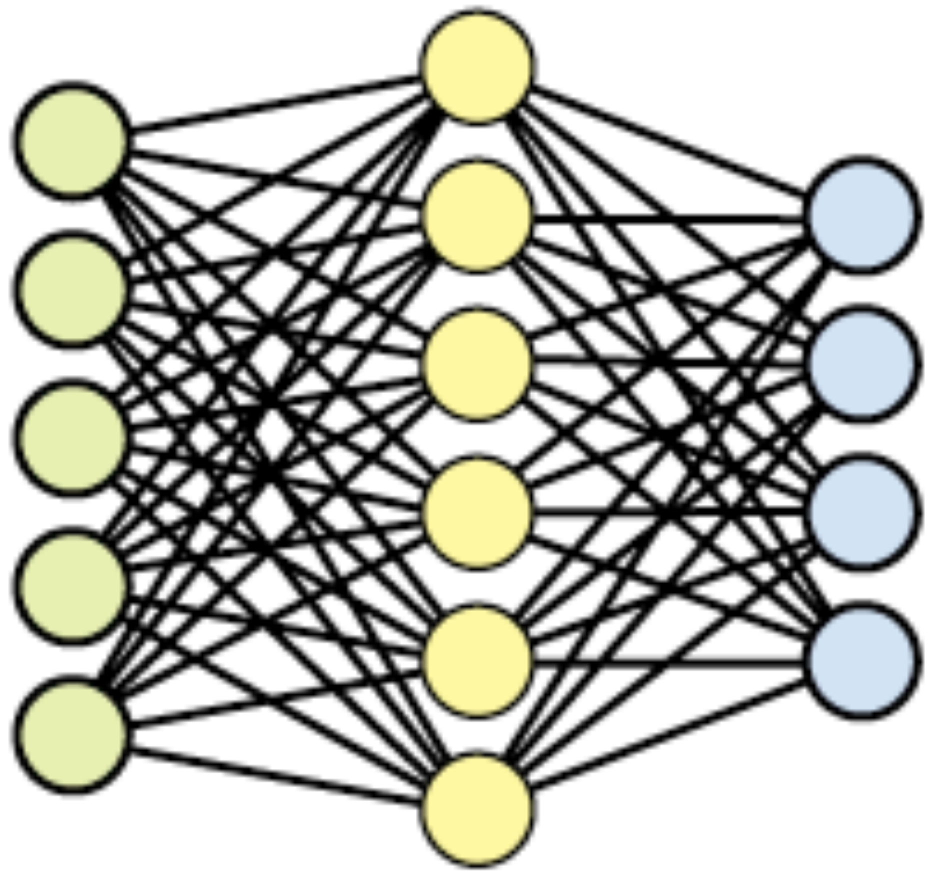
- It's very expensive when N is large (big data).

Do I lose anything by using stochastic GD?

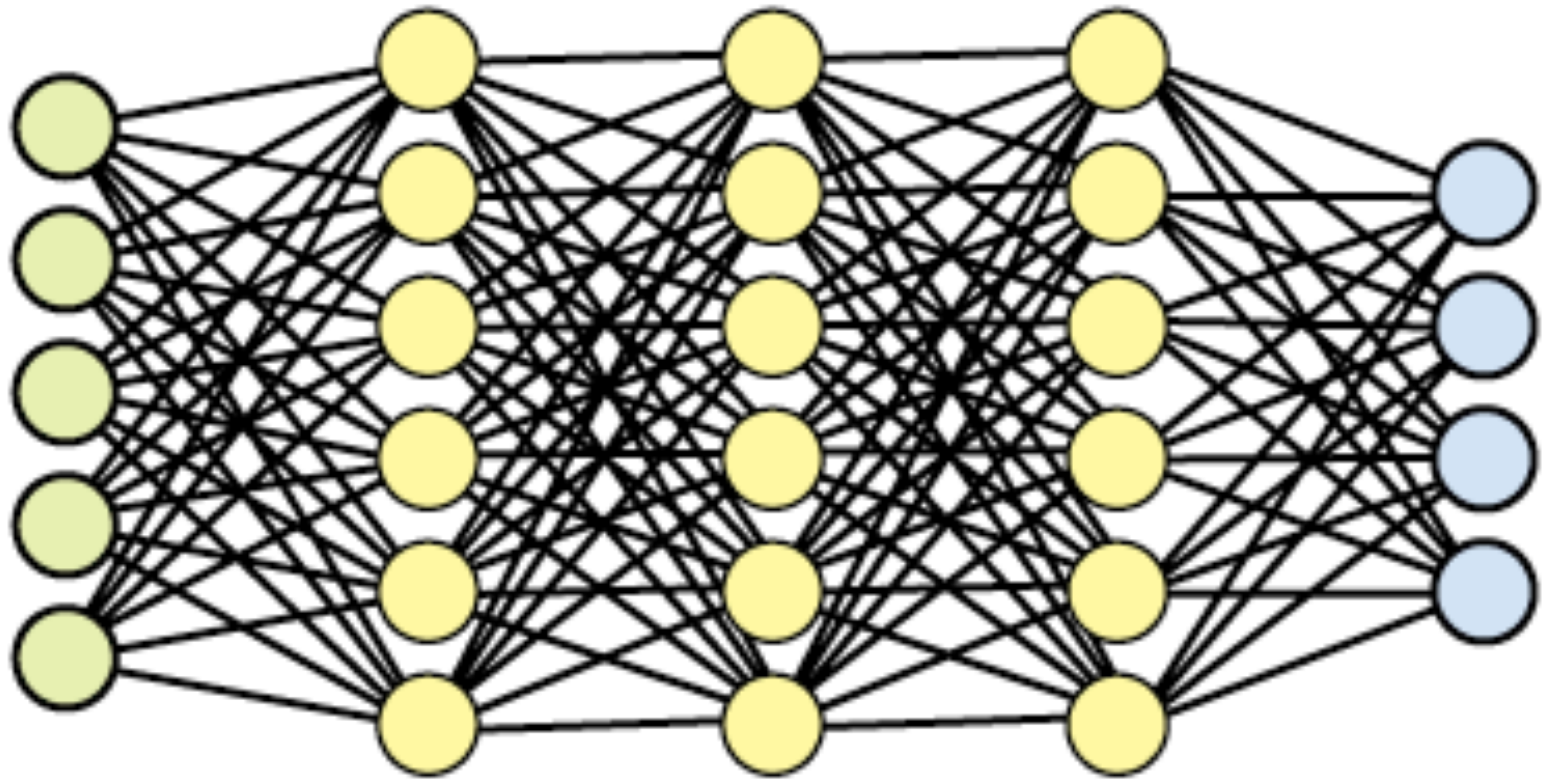
- Same convergence guarantees and complexity!
- Better generalization.

From Neural Networks to Deep Neural Networks

A neural Network



A deep neural Network



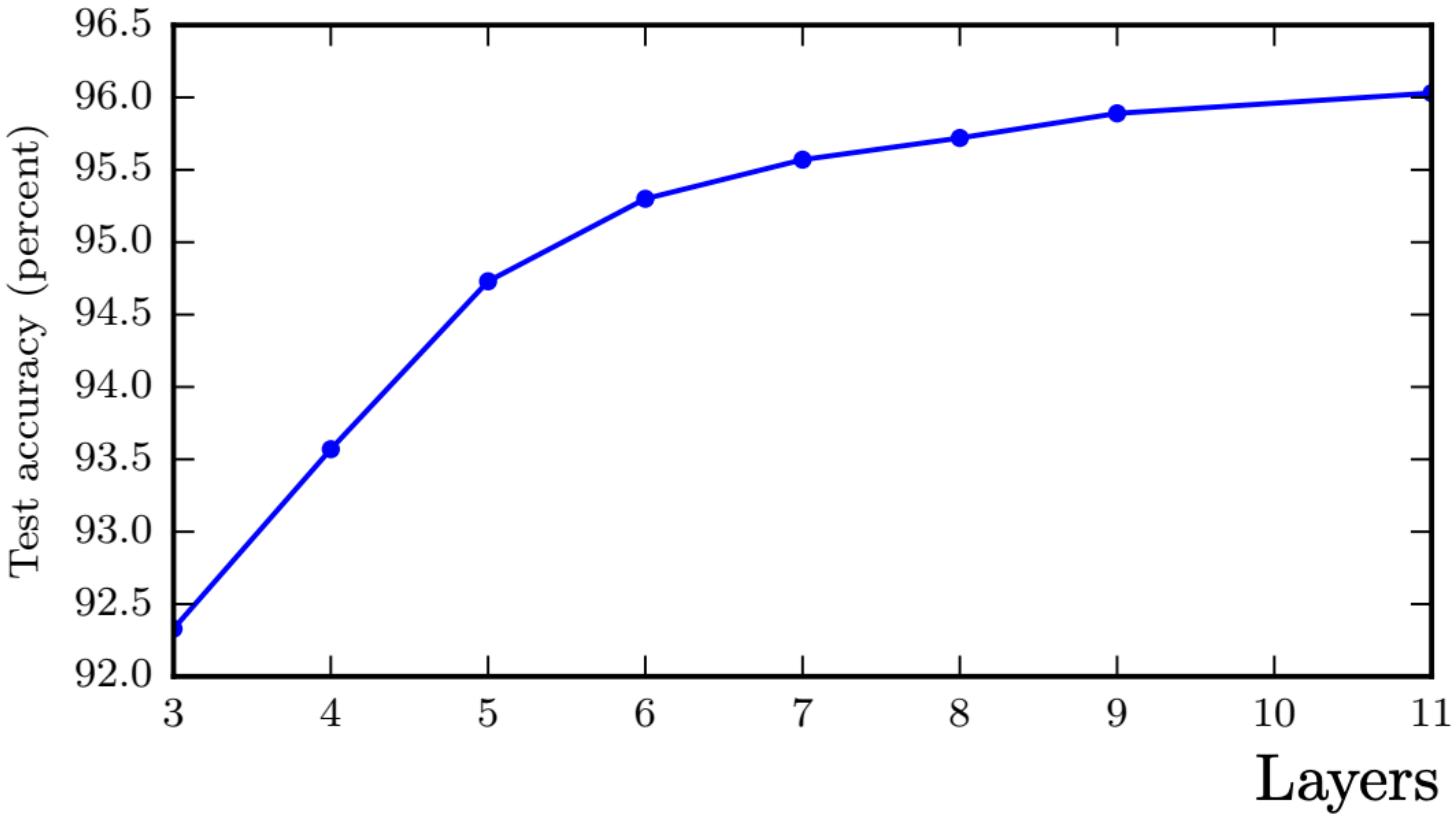
Modern deep learning provides a powerful framework for supervised learning. By adding more layers and more units within a layer, a deep network can represent functions of increasing complexity.

Deep Learning — Part II, p.163
http://www.deeplearningbook.org/contents/part_practical.html

Hyperparameters

1. **Model specific**
   - Activation functions (output & hidden), Network size
2. **Optimisation Objective**
   - Regularization, Early-stopping, Dropout
3. **Optimization procedure**
   - Momentum, Adaptive learning rates

Wide or Deep?



[Figure 6.6, Deep Learning, book]