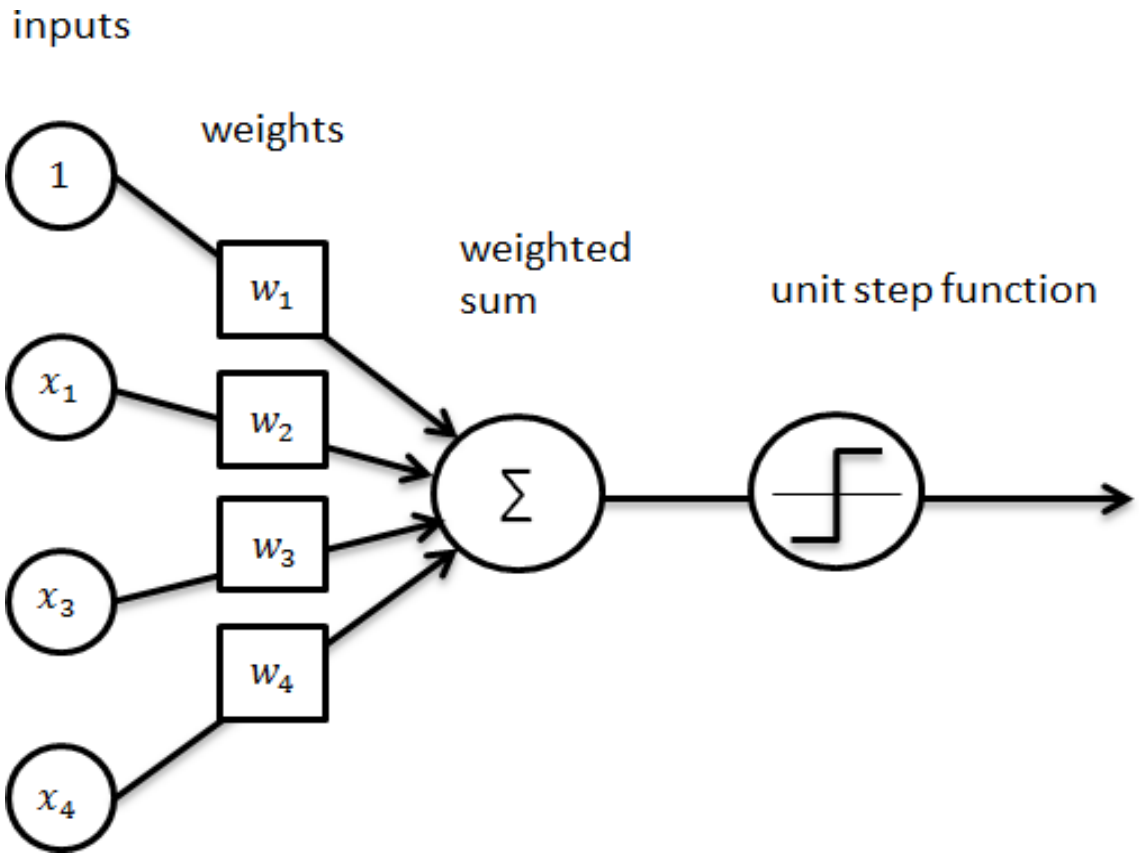
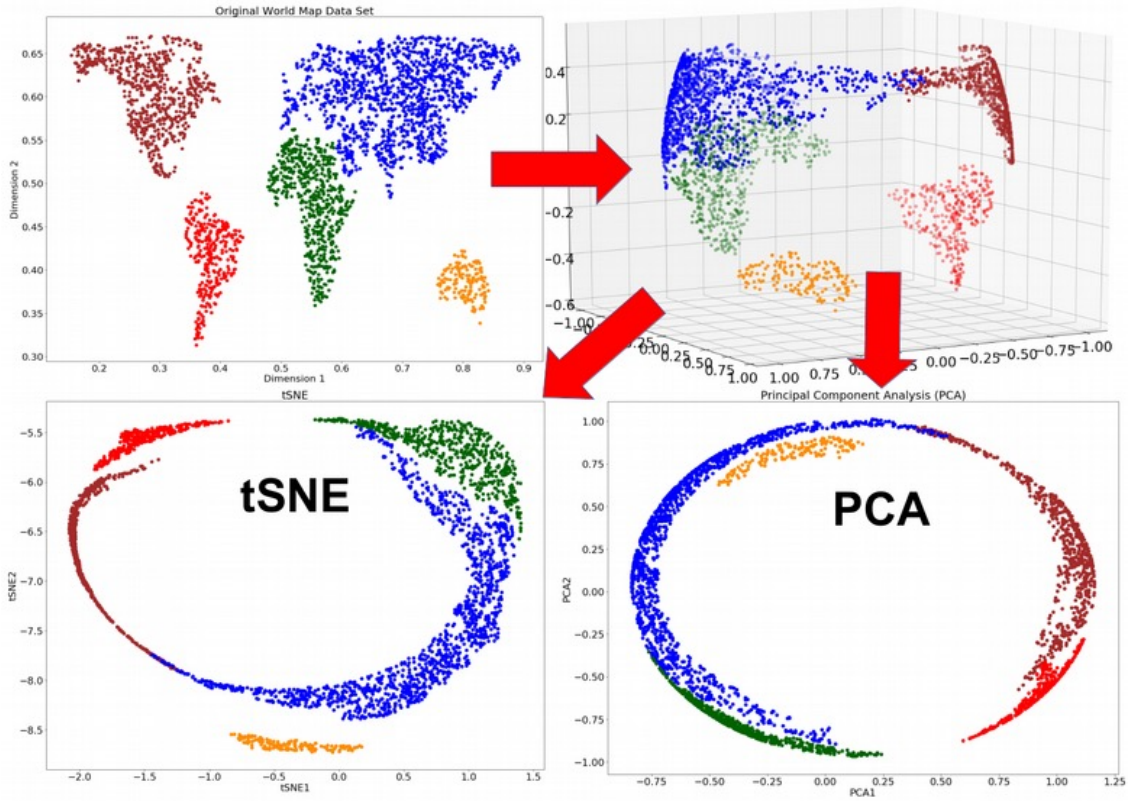
Neural Networks



Feature Visualization

- PCA is good, but it is a linear algorithm, meaning that it cannot represent complex relationship between features
- t-SNE is non-linear dimensionality reduction technique that has better performance. It is designed for visualization purposes.



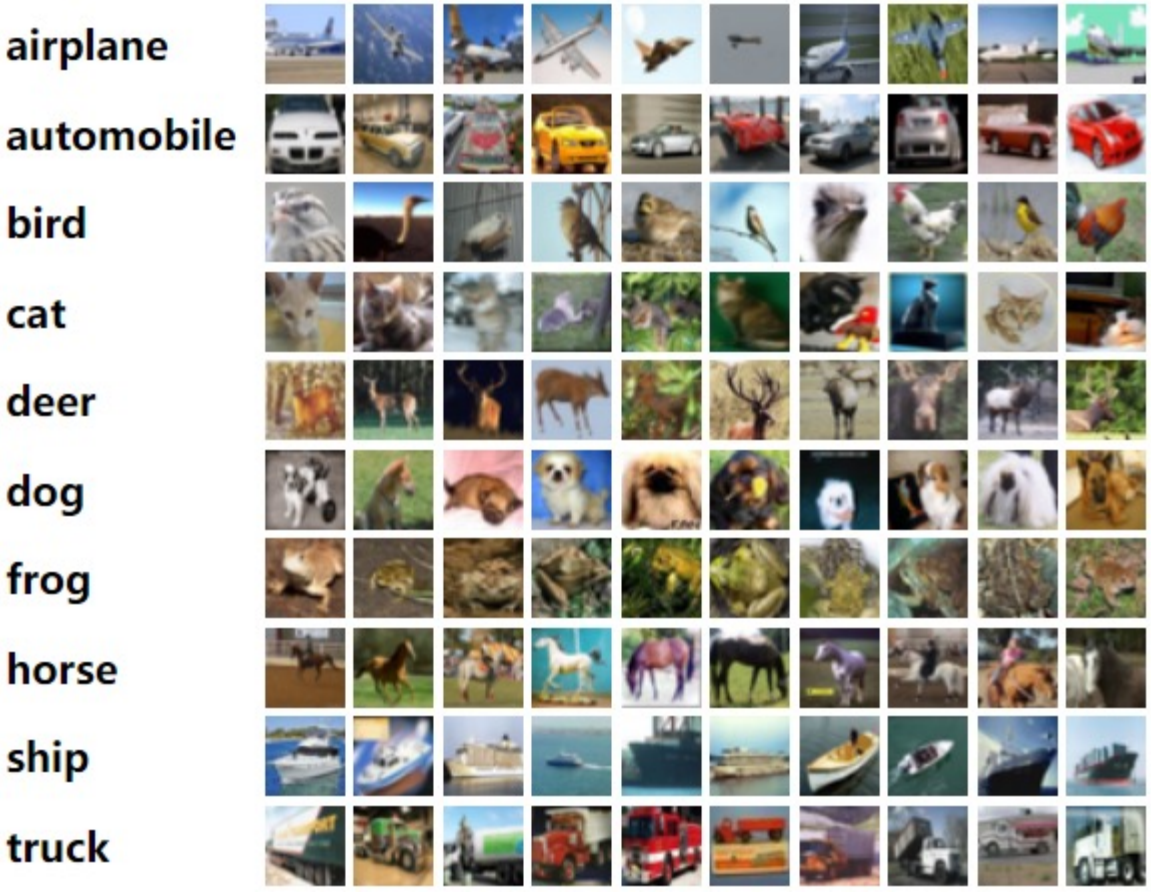
sklearn.decomposition.PCA

sklearn.manifold.TSNE

Example Link

UNIFORM MANIFOLD UMAP APPROXIMATION & PROJECTION

Recall CIFAR10



50,000 training images

10,000 test images

each image is **32x32x3**

Linear Classifier – Score Function

Image

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

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

3072x1

10x1 **10x3072**

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

10 numbers giving class scores

W

parameters or weights

Linear Classifier – Score Function

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

10x1

10x1 10x3072

Image

Array of 32x32x3 numbers
(3072 numbers total)

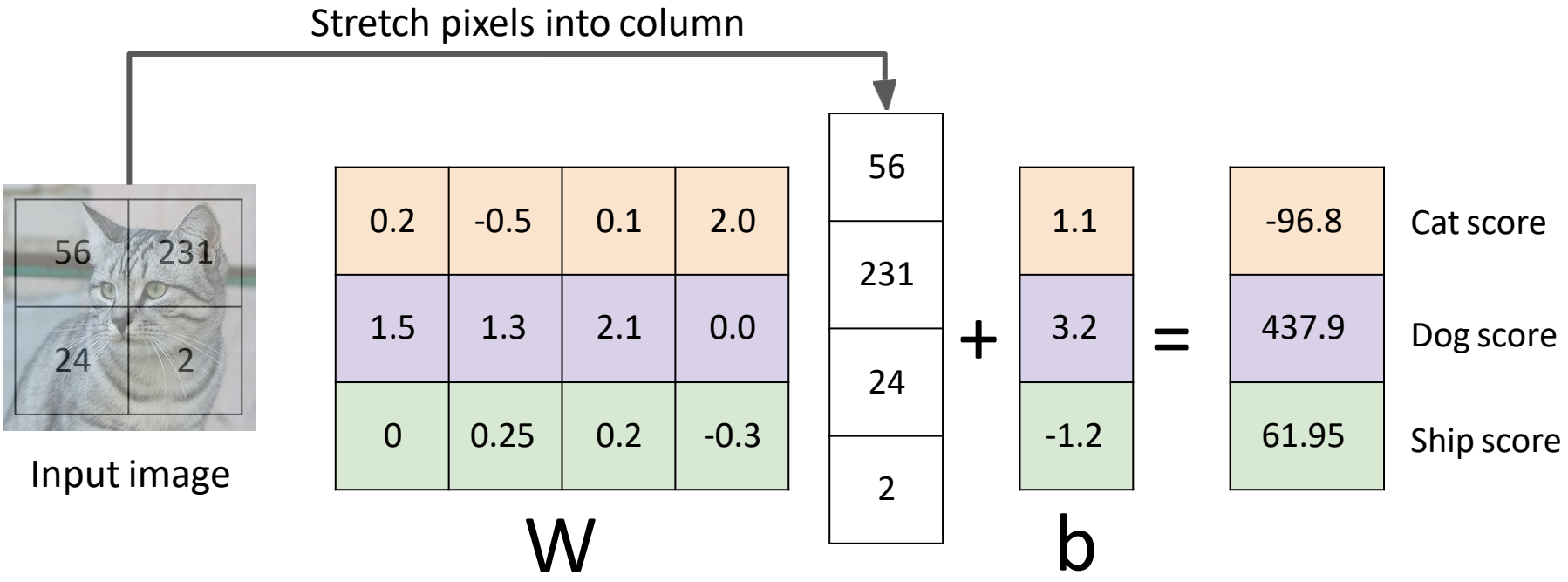
f(x,W)

10 numbers giving class scores

W

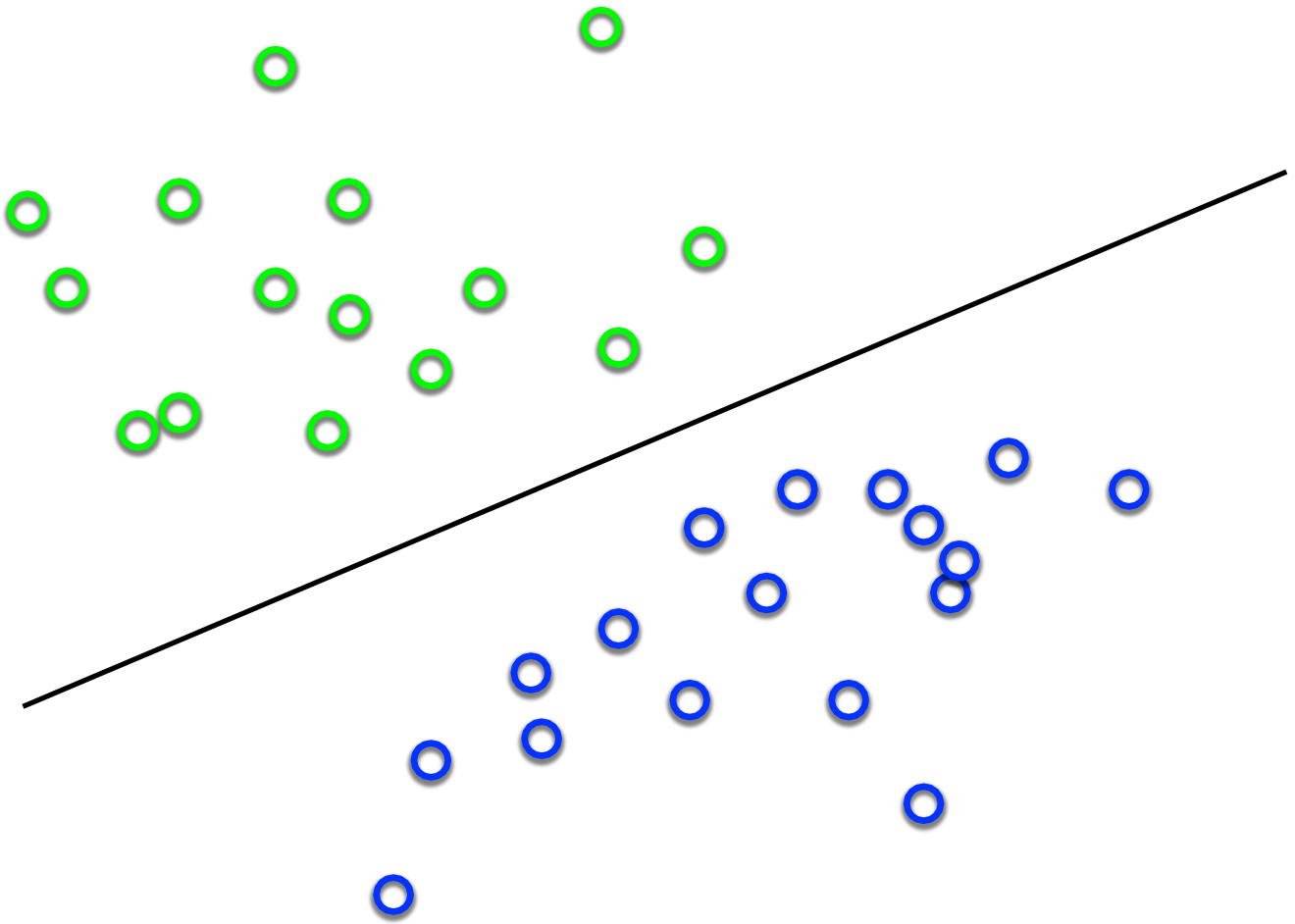
parameters or weights

Example: an image with 4 pixels, and 3 classes (cat/dog/ship)



What's the best w?

Intuitively, the line that is the farthest from all interior points



Maximum Margin solution:
most stable to perturbations of data

SVM can be formulated as a maximization problem

$$\max_{\boldsymbol{w}} \frac{2}{\|\boldsymbol{w}\|}$$

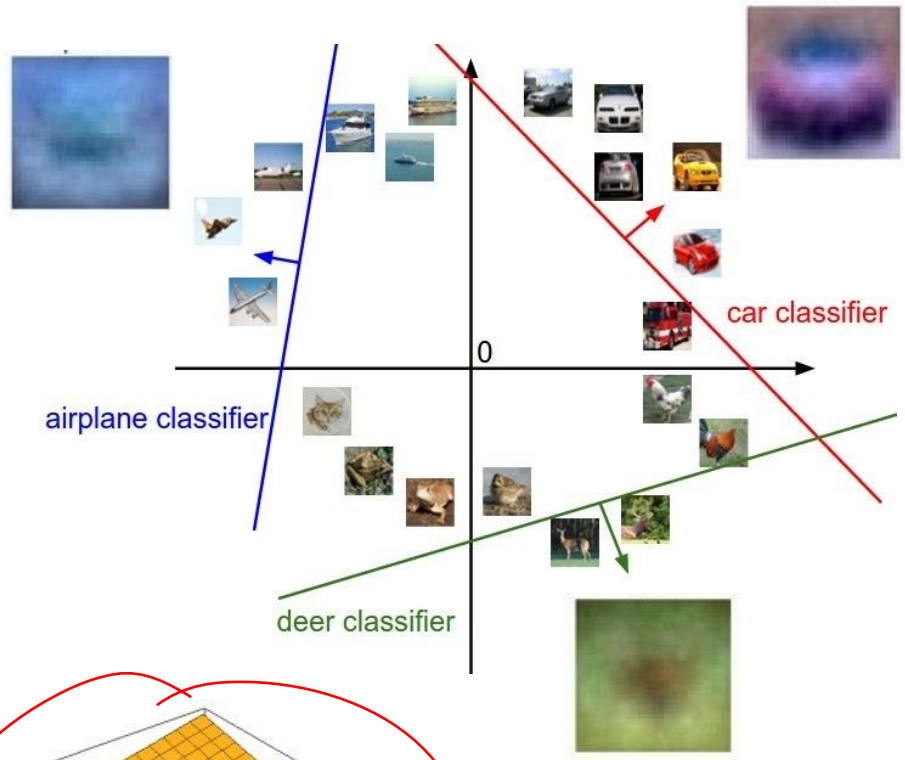
$$\text{subject to } \boldsymbol{w} \cdot \boldsymbol{x}_i + b \begin{array}{l} \geq +1 \text{ if } y_i = +1 \\ \leq -1 \text{ if } y_i = -1 \end{array} \text{ for } i = 1, \ldots, N$$

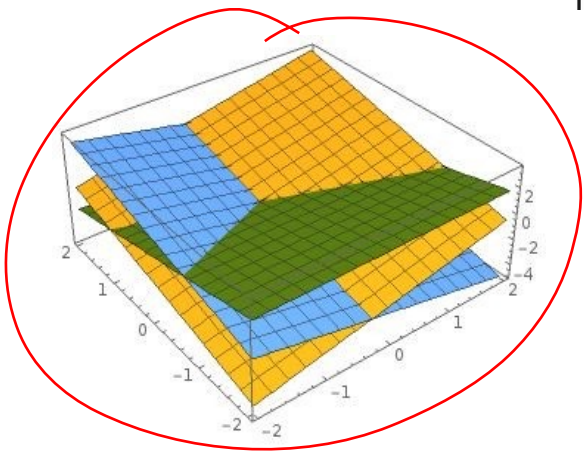
Equivalently,

$$\min_{\boldsymbol{w}} \|\boldsymbol{w}\|$$

$$\text{subject to } y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1 \text{ for } i = 1, \ldots, N$$

Interpreting a Linear Classifier: Geometric Viewpoint





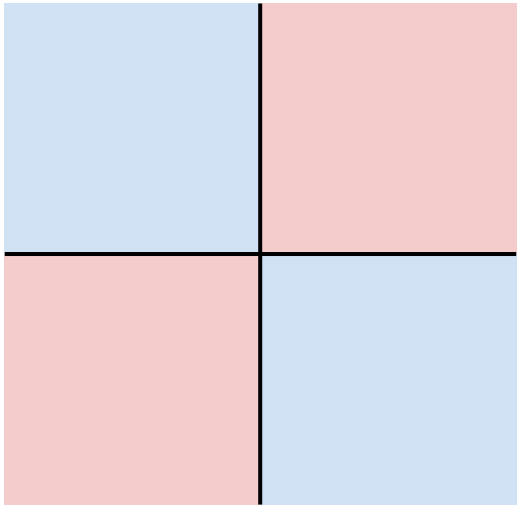
$$f(x,W) = Wx + b$$

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

Hard cases for a linear classifier

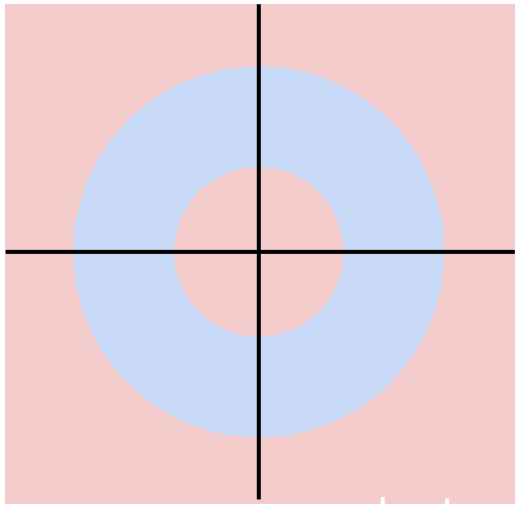
Class 1:
First and third quadrants

Class 2:
Second and fourth quadrants



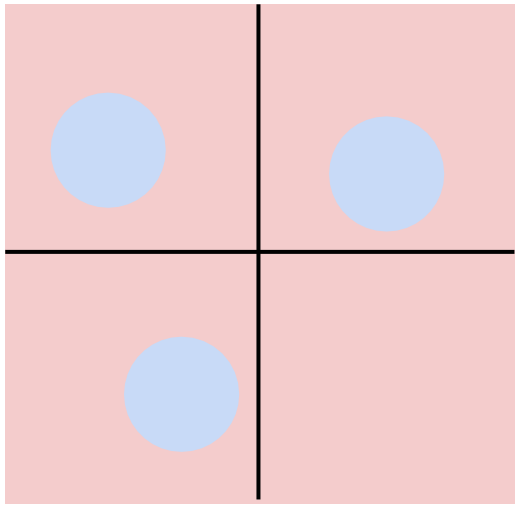
Class 1:
1 <= L2 norm <= 2

Class 2:
Everything else



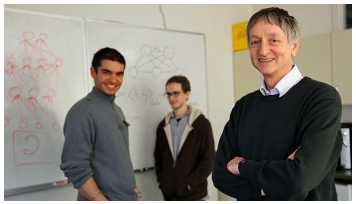
Class 1: Three modes

Class 2:
Everything else



Why is everyone talking about Deep Learning?

- Because a lot of money is invested in it…
  - DeepMind: Acquired by Google for **$400 million**
  - DNNResearch: **Three-person startup** (including Geoff Hinton) acquired by Google for unknown price tag
  - Enlitic, Ersatz, MetaMind, Nervana, Skylab: Deep Learning startups commanding **millions of VC dollars**



MetaMind

- Because it made the **front page** of the New York Times

The New York Times

Why is everyone talking about Deep Learning?

1950s Age of the Perceptron

1957 The Perceptron (Rosenblatt)

1969 Perceptrons (Minsky, Papert)

1980s Age of the Neural Network

1986 Back propagation (Hinton)

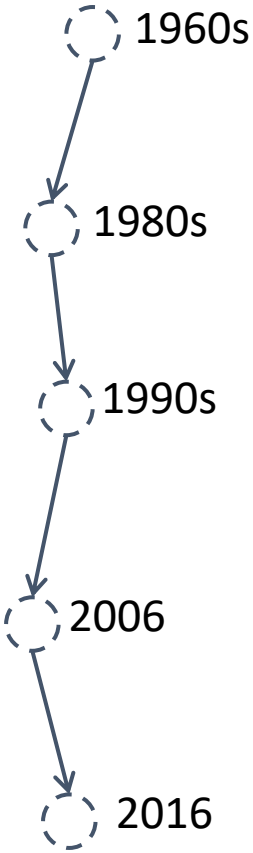
1990s Age of the Graphical Model

2000s Age of the Support Vector Machine

2010s Age of the Deep Network

deep learning = known algorithms + computing power + big data

Why is everyone talking about Deep Learning?



Deep learning:

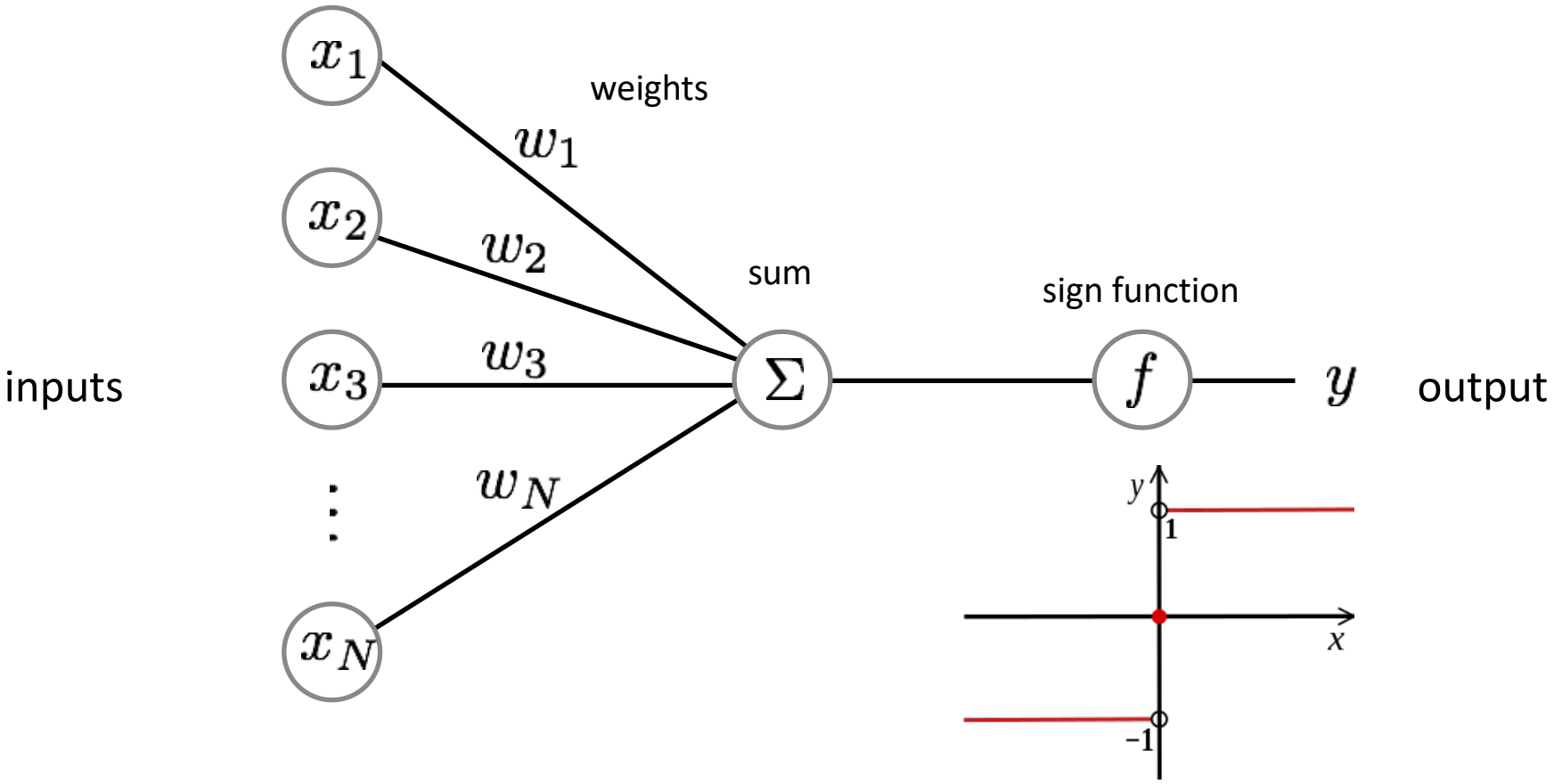
- Has won numerous pattern recognition competitions
- Does so with minimal feature engineering

This wasn't always the case!

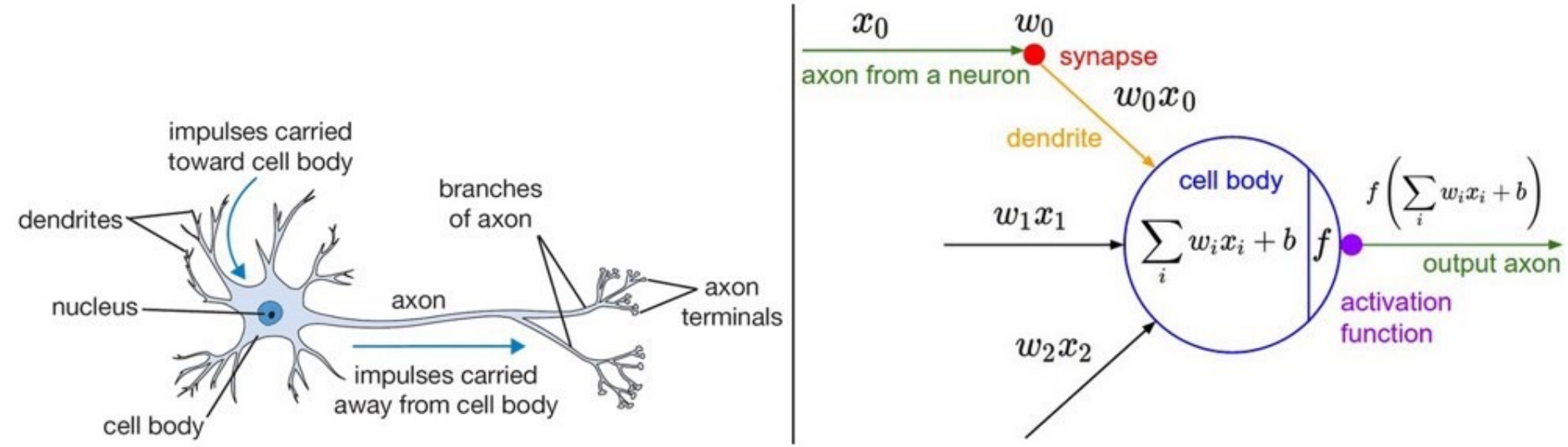
Since 1980s: Form of models hasn't changed much, but lots of new tricks…

- More hidden units
- Better (online) optimization
- New nonlinear functions (ReLUs)
- Faster computers (CPUs and GPUs)

Perceptron


inputs
x_1
x_2
x_3
x_N
weights
w_1
w_2
w_3
w_N
sum
Σ
sign function
f
y
output
y
1
x
-1

Aside: Inspiration from Biology



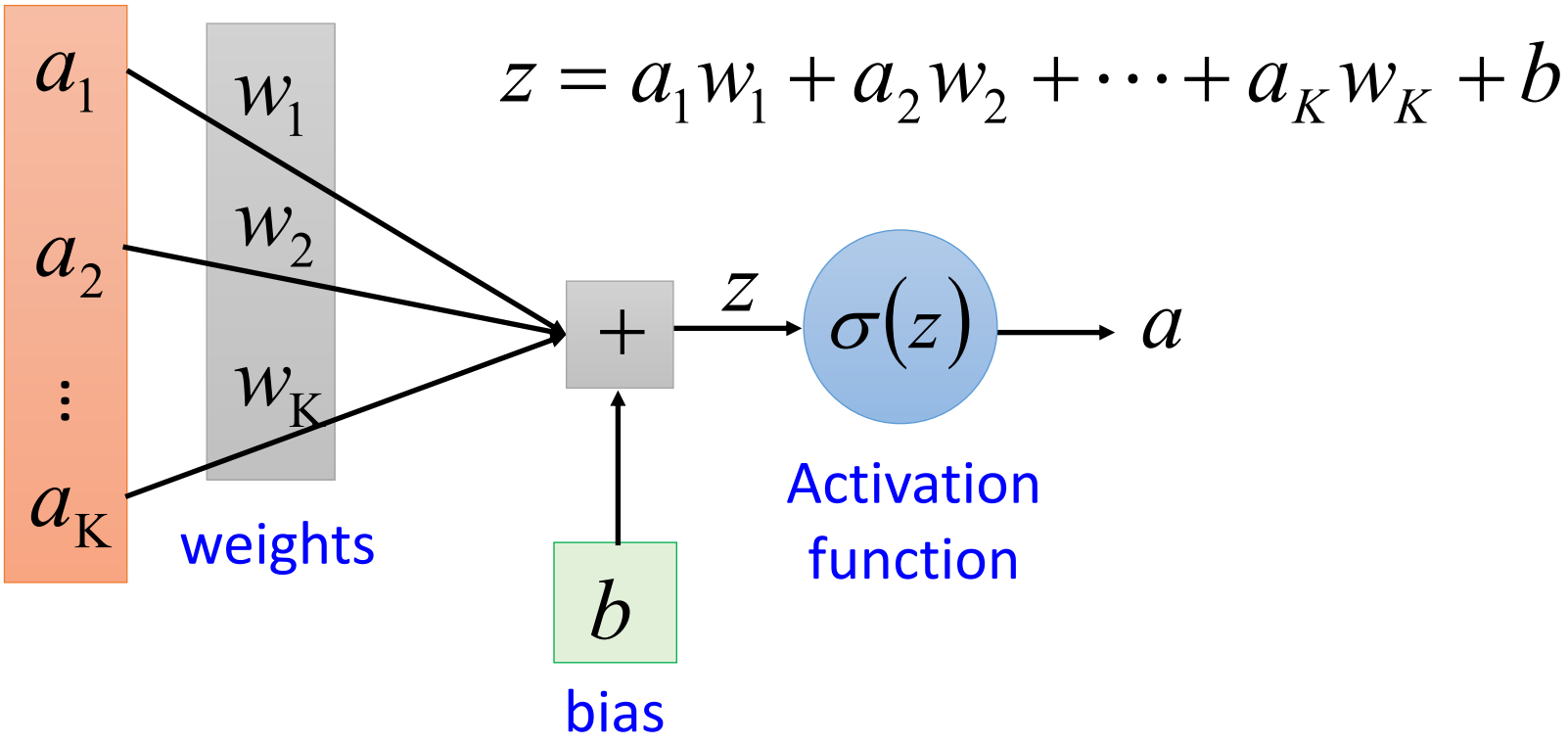
A cartoon drawing of a biological neuron (left) and its mathematical model (right).

Neural nets/perceptrons are loosely inspired by biology.

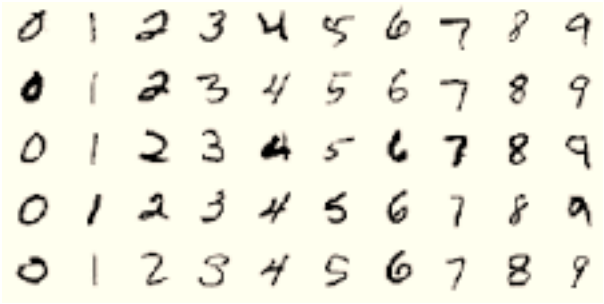
But they certainly are not a model of how the brain works, or even how neurons work.

Perceptron

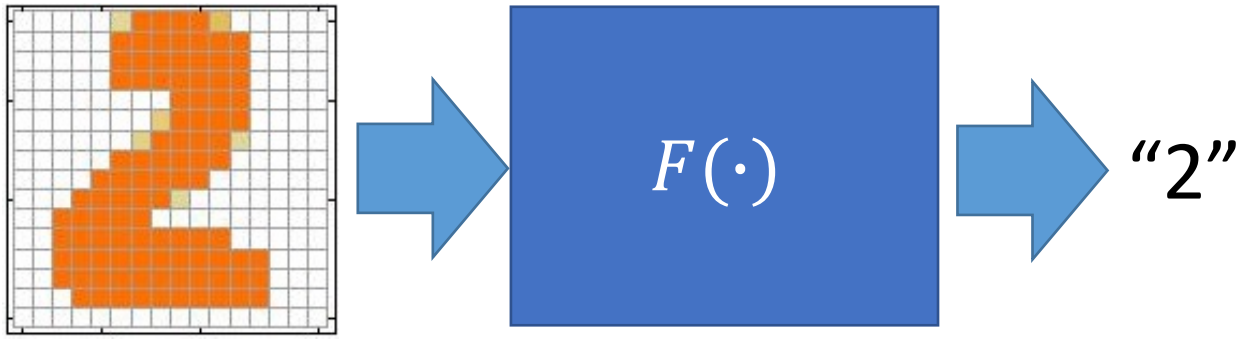
Neuron $f: R^K \rightarrow R$



Example Application

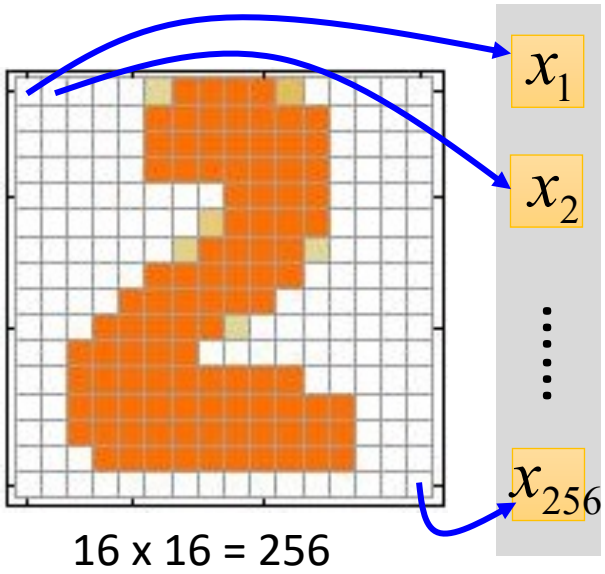


- Handwriting Digit Recognition



Handwriting Digit Recognition

Input

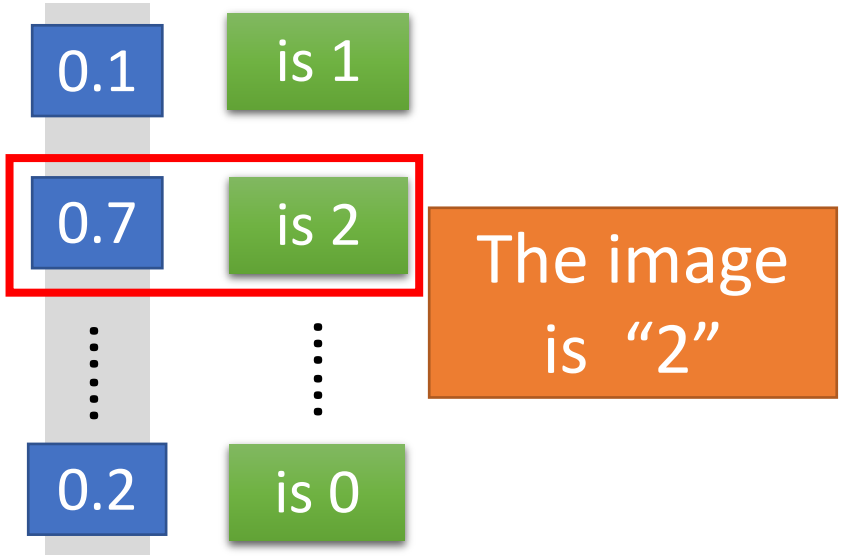


16 x 16 = 256

Ink → 1

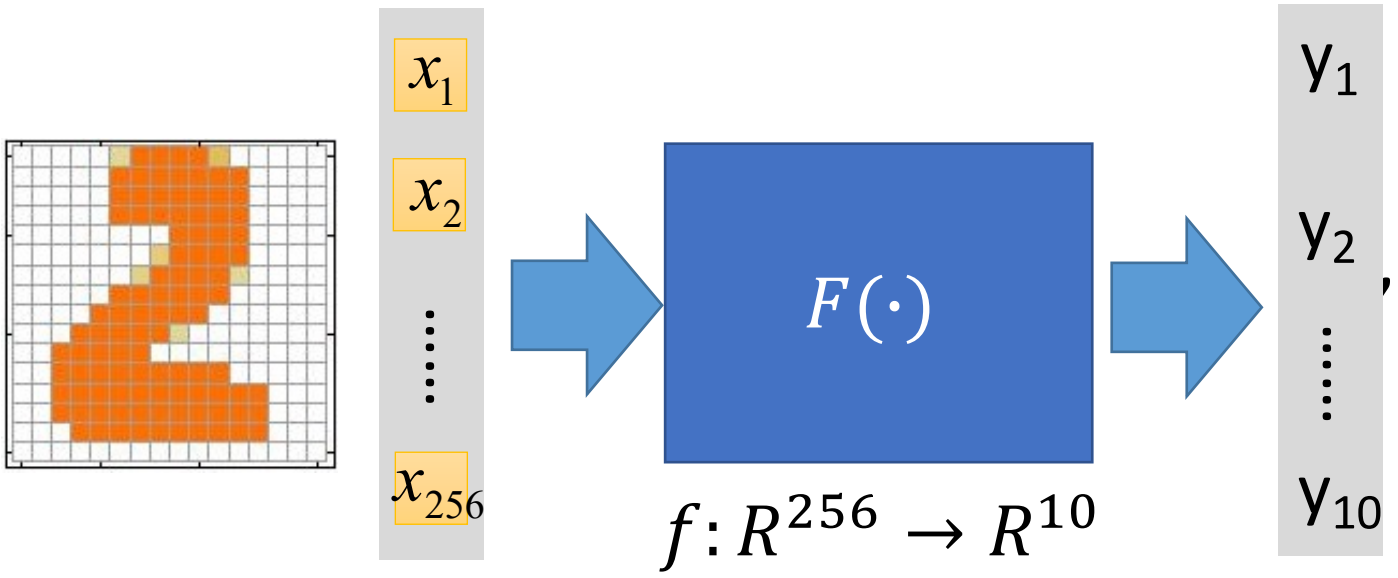
No ink → 0

Output



Each dimension represents the confidence of a digit.

Handwriting Digit Recognition



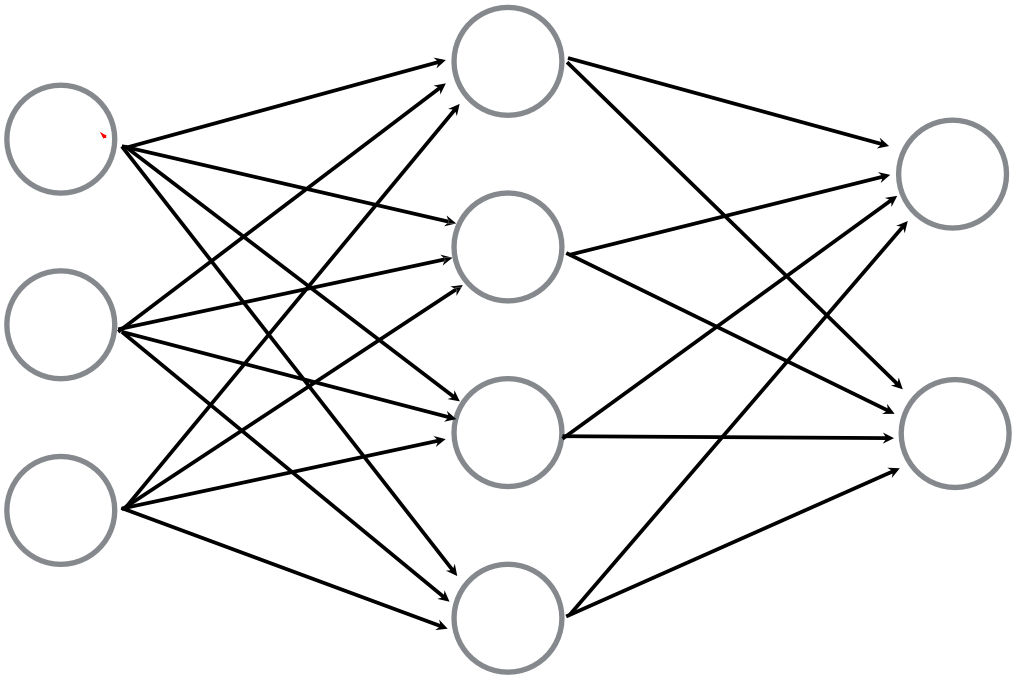
In deep learning, the function $F(\cdot)$ is represented by neural network

Neural Networks

Connect a bunch of perceptrons together …

Neural Network

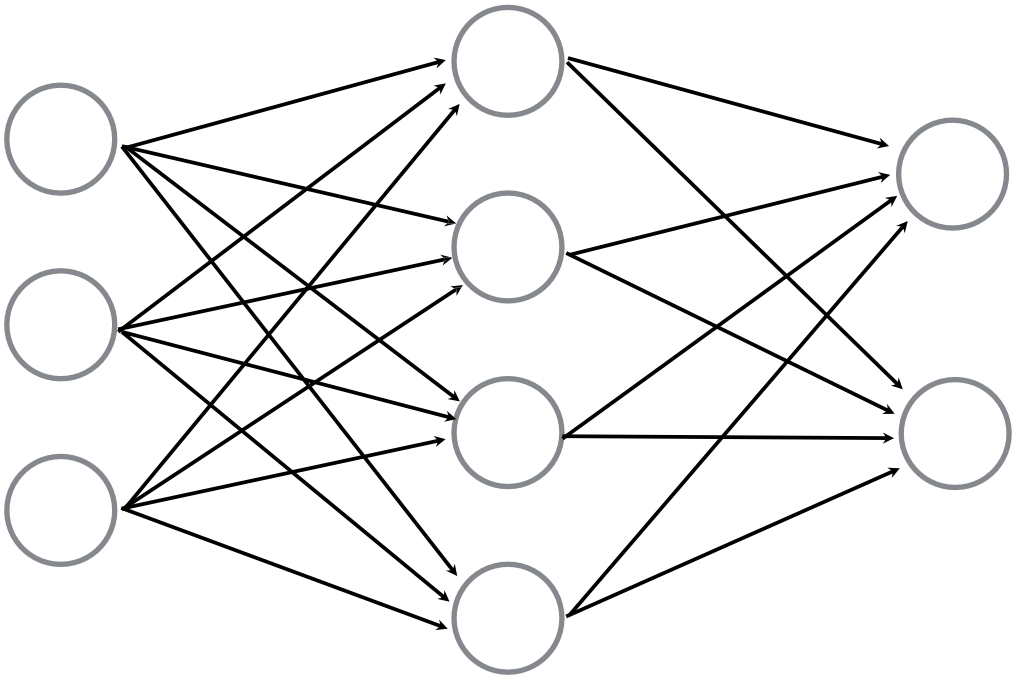
a collection of connected perceptrons



Connect a bunch of perceptrons together …

Neural Network

a collection of connected perceptrons

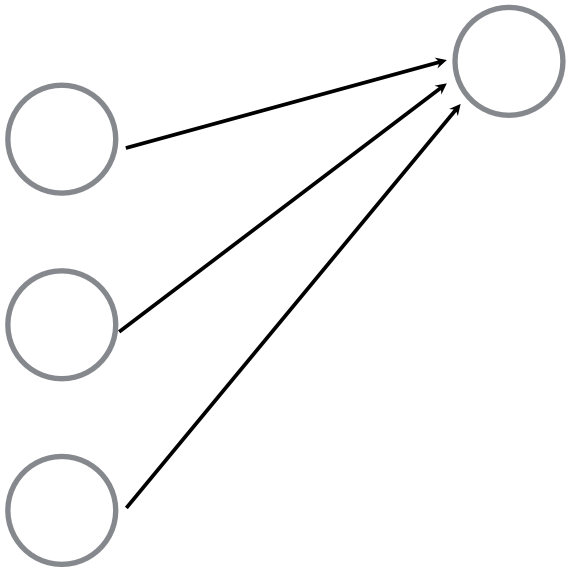


How many perceptrons in this neural network?

Connect a bunch of perceptrons together …

Neural Network

a collection of connected perceptrons

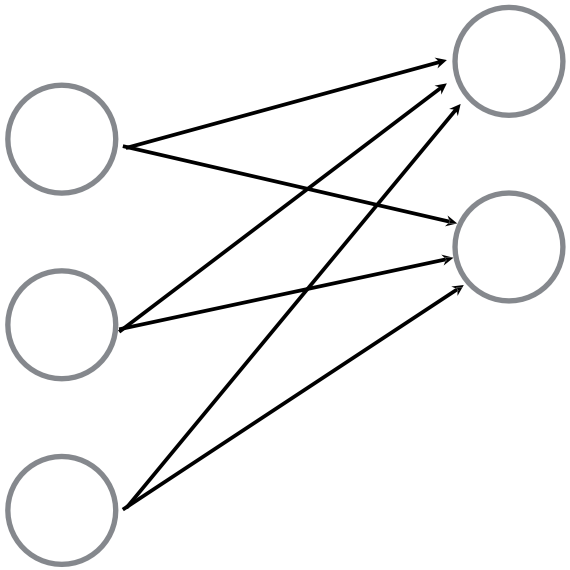


‘one perceptron’

Connect a bunch of perceptrons together …

Neural Network

a collection of connected perceptrons

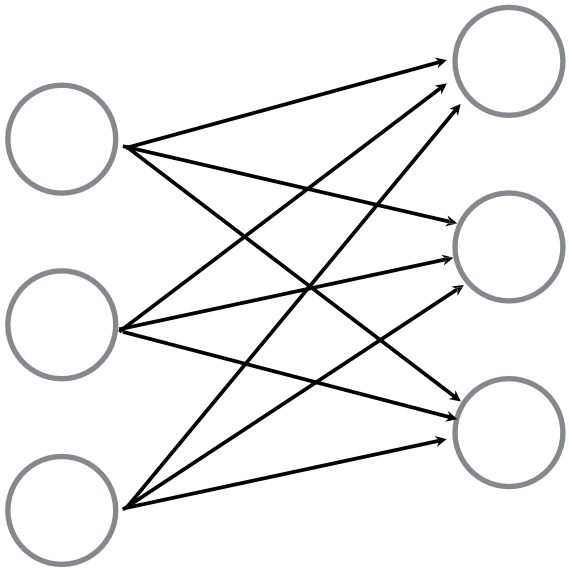


‘two perceptrons’

Connect a bunch of perceptrons together …

Neural Network

a collection of connected perceptrons

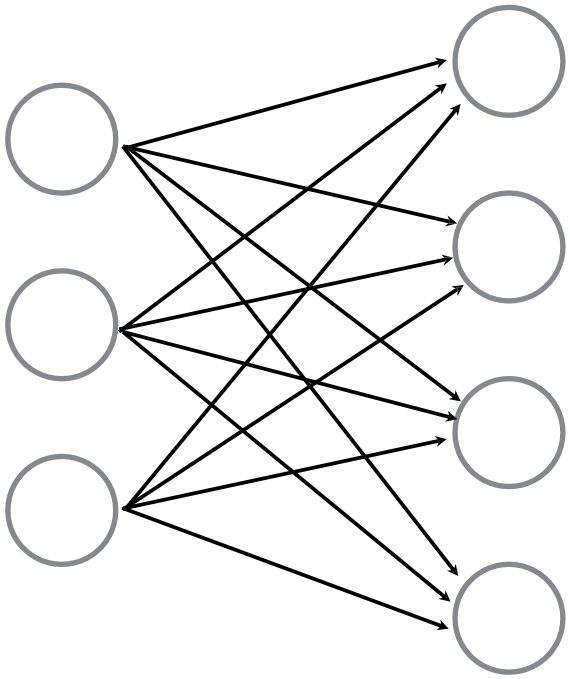


‘three perceptrons’

Connect a bunch of perceptrons together …

Neural Network

a collection of connected perceptrons

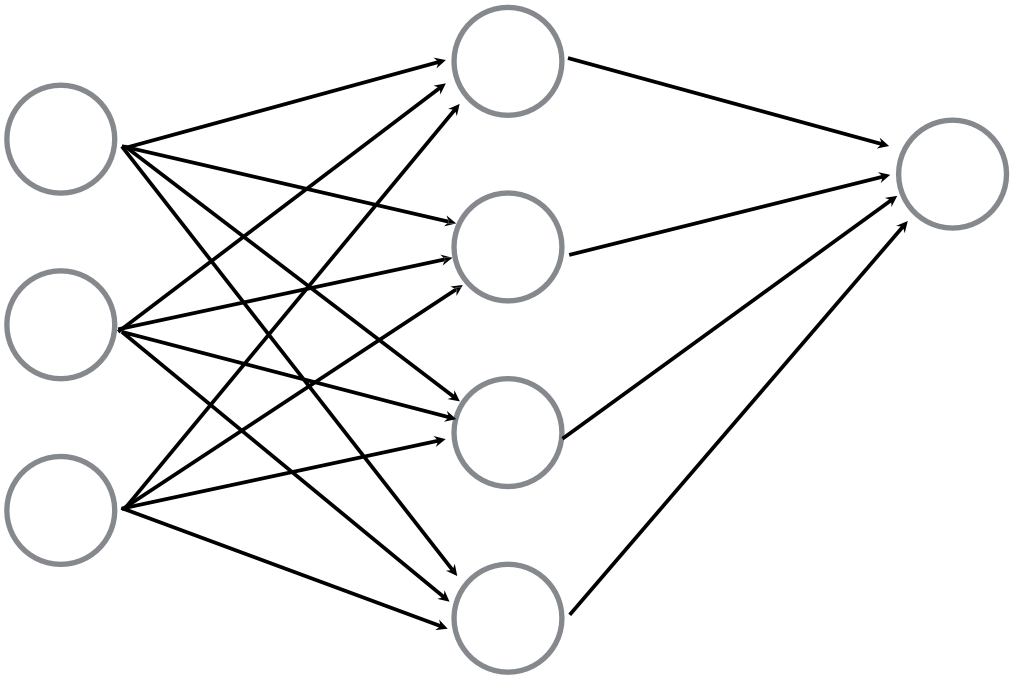


‘four perceptrons’

Connect a bunch of perceptrons together …

Neural Network

a collection of connected perceptrons

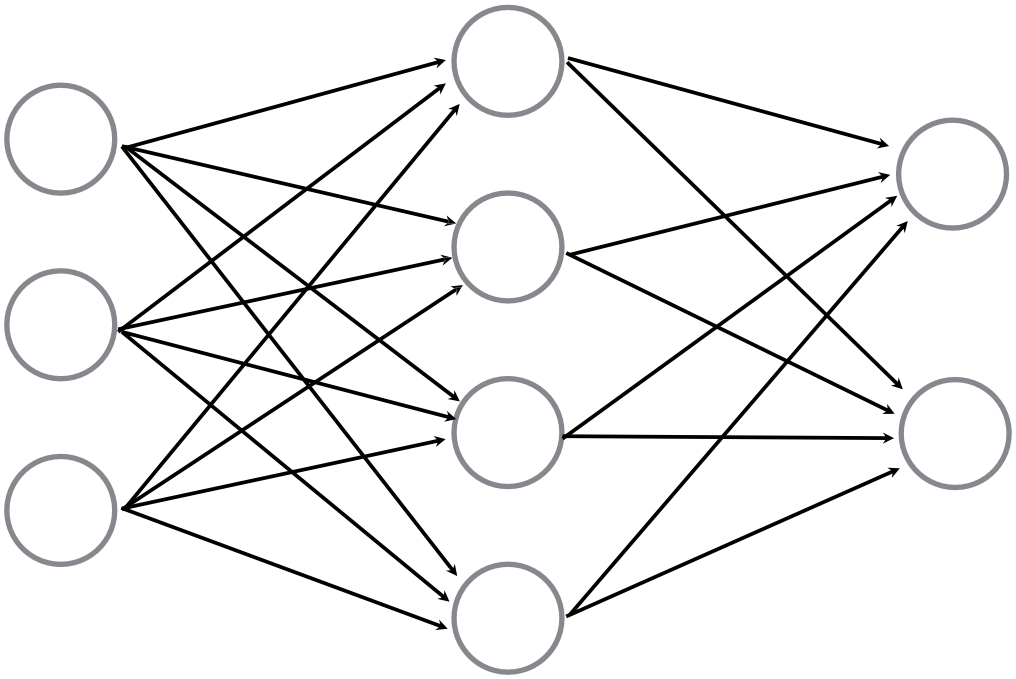


‘five perceptrons’

Connect a bunch of perceptrons together …

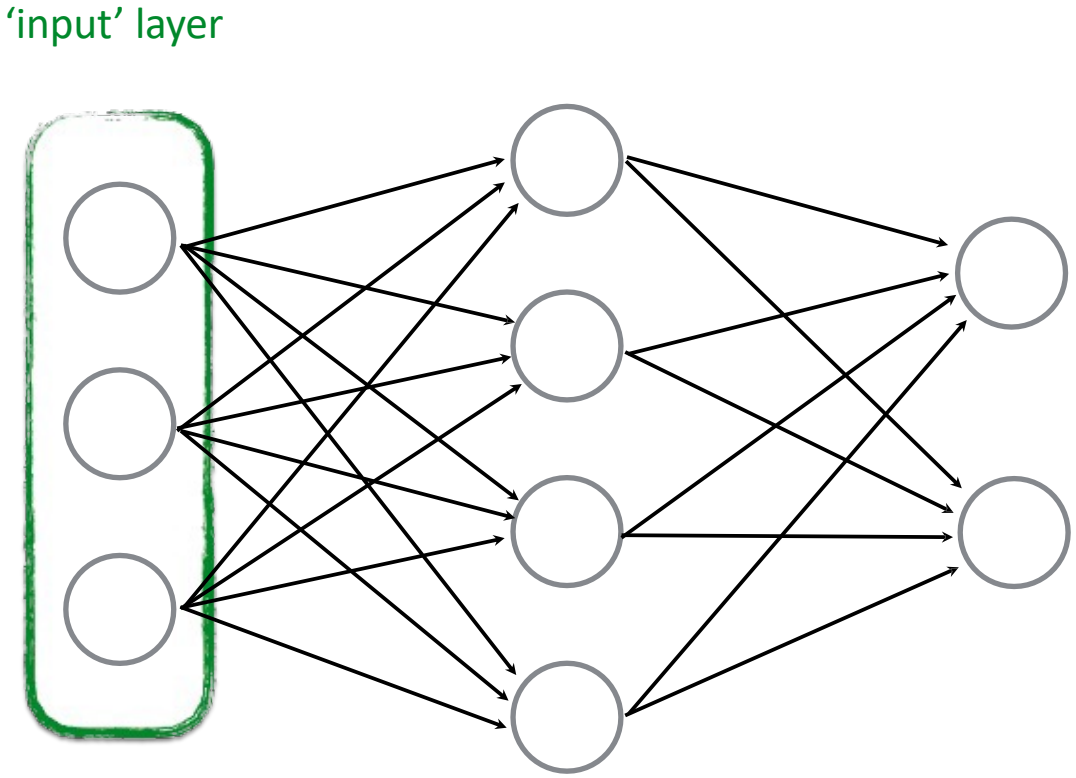
Neural Network

a collection of connected perceptrons



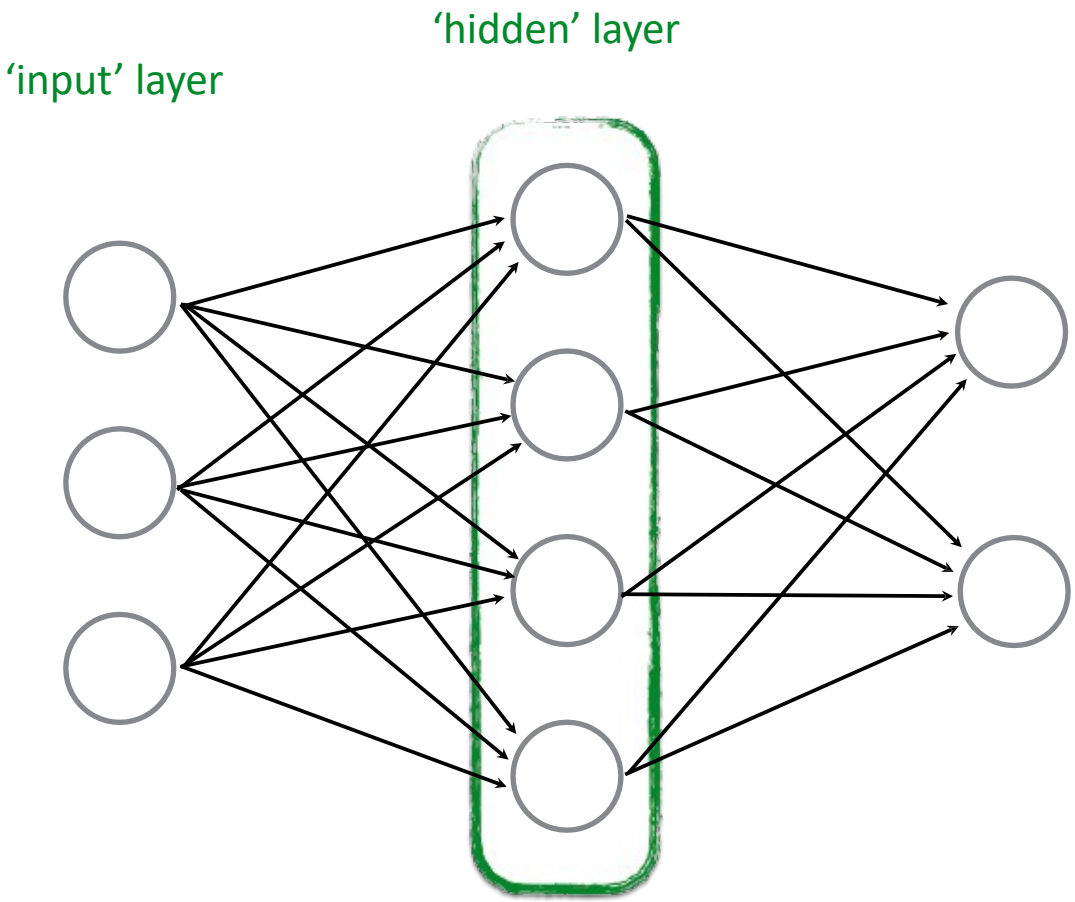
‘six perceptrons’

Some terminology…



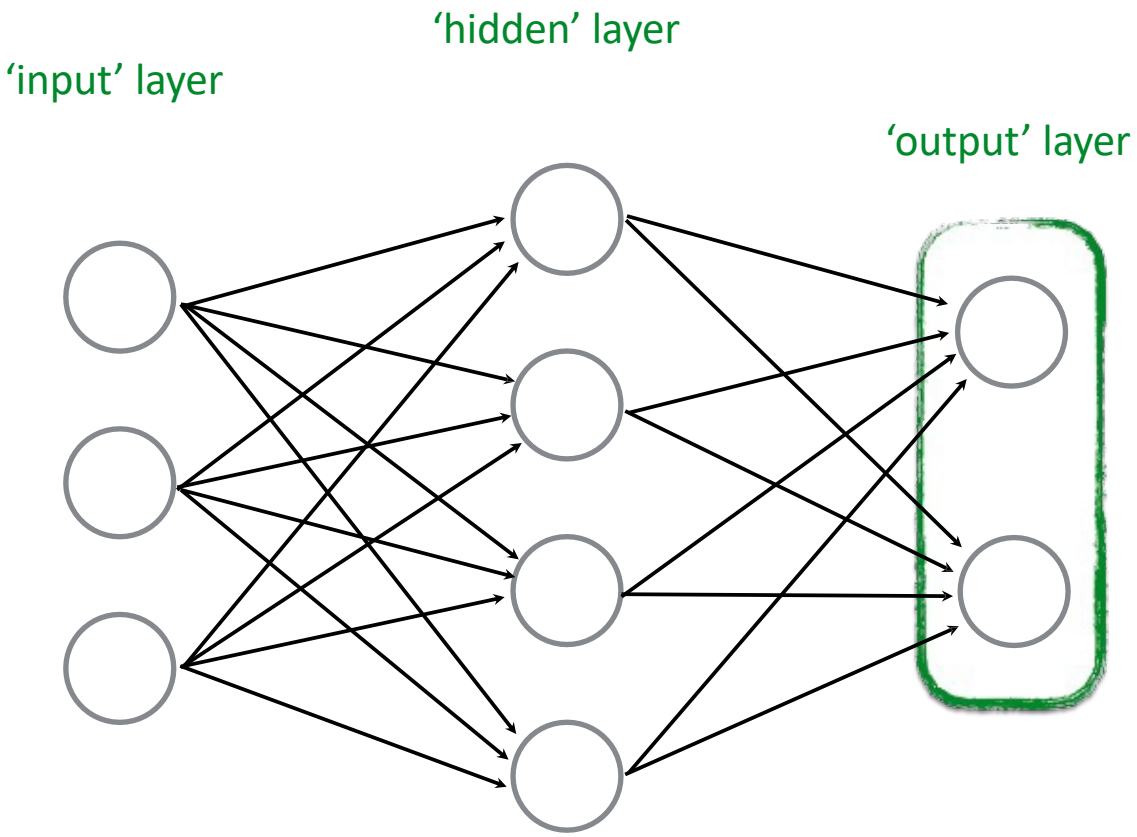
…also called a Multi-layer Perceptron (MLP)

Some terminology…



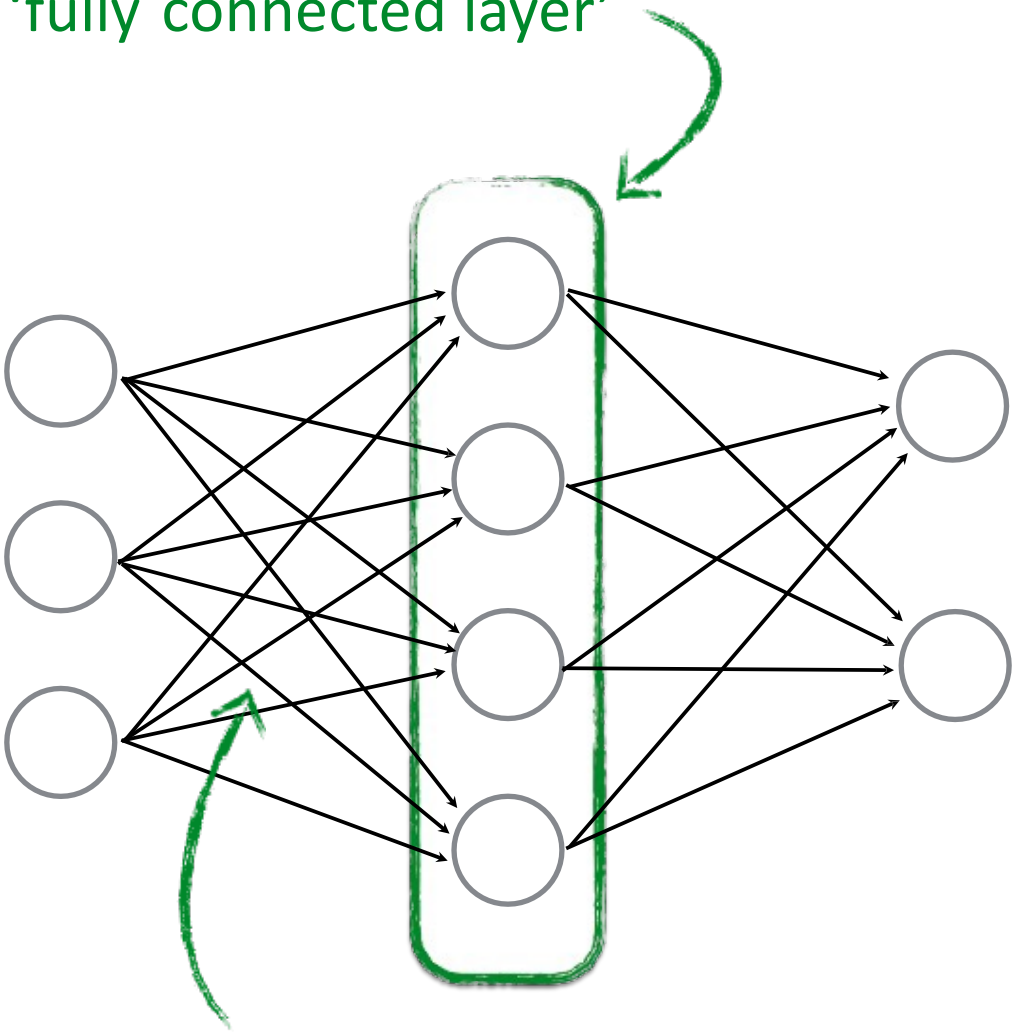
…also called a Multi-layer Perceptron (MLP)

Some terminology…



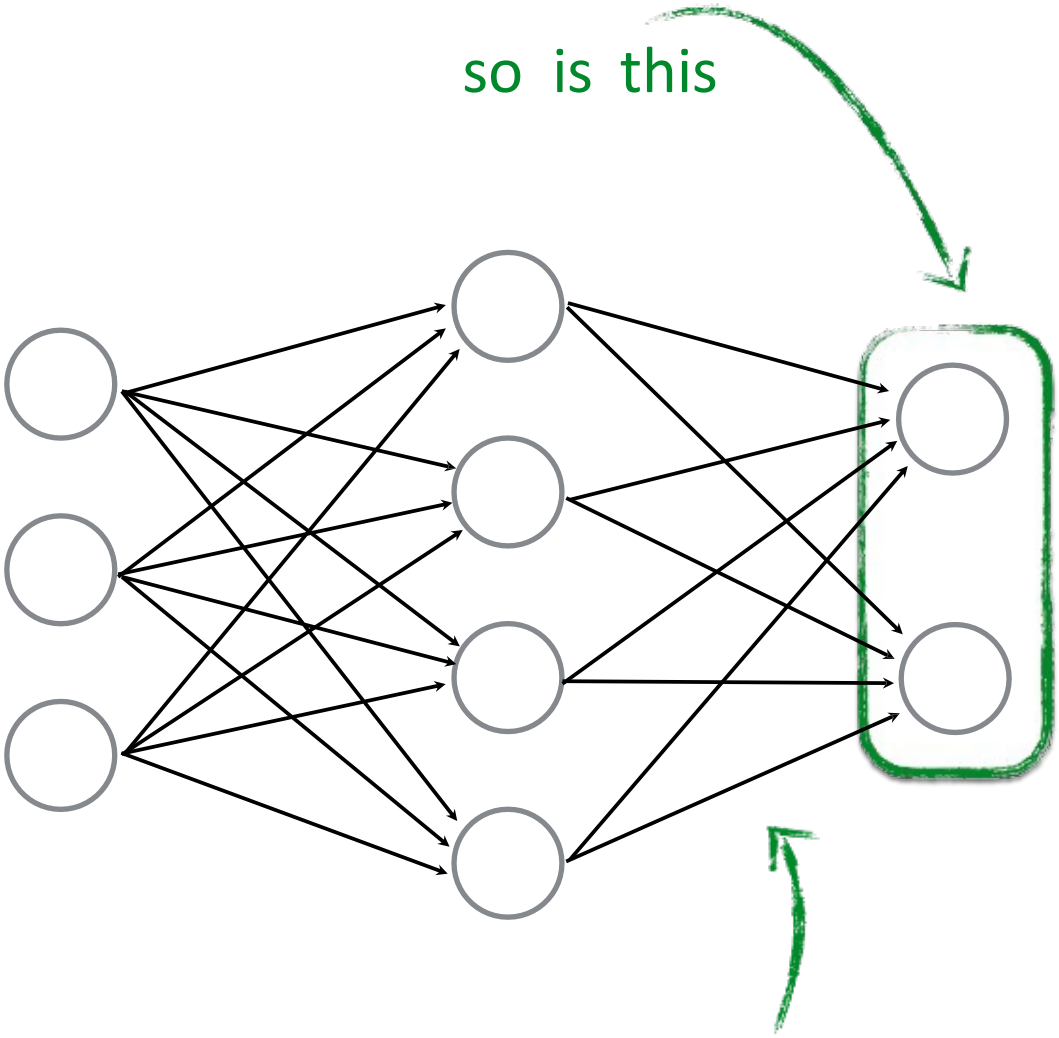
…also called a Multi-layer Perceptron (MLP)

this layer is a
'fully connected layer'



all pairwise neurons between layers are connected

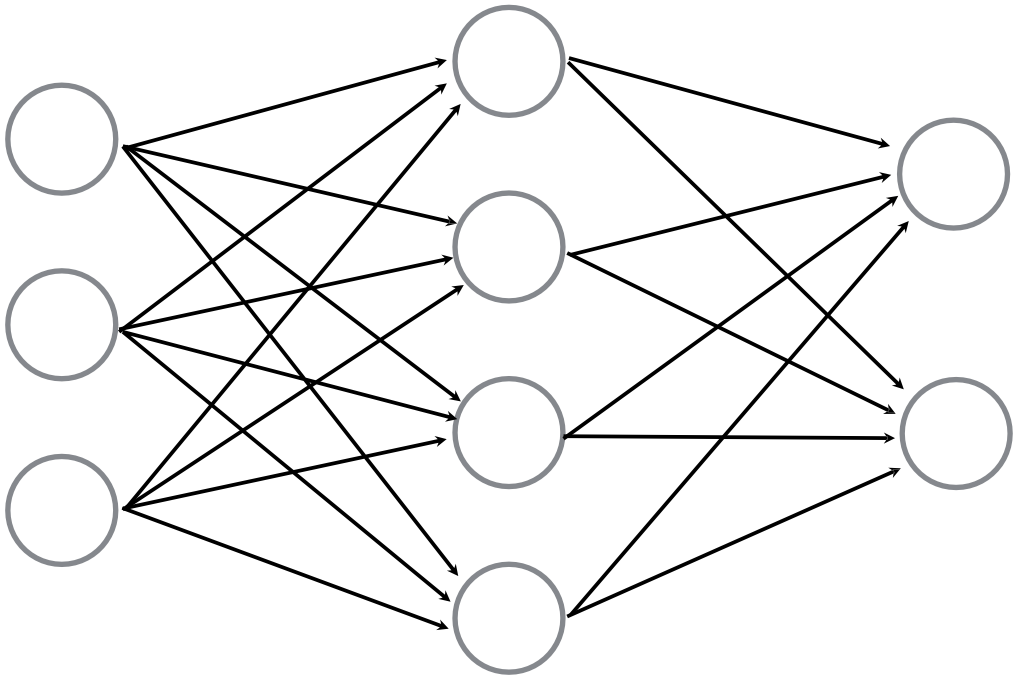
so is this



all pairwise neurons between layers are connected

How many neurons (perceptrons)?

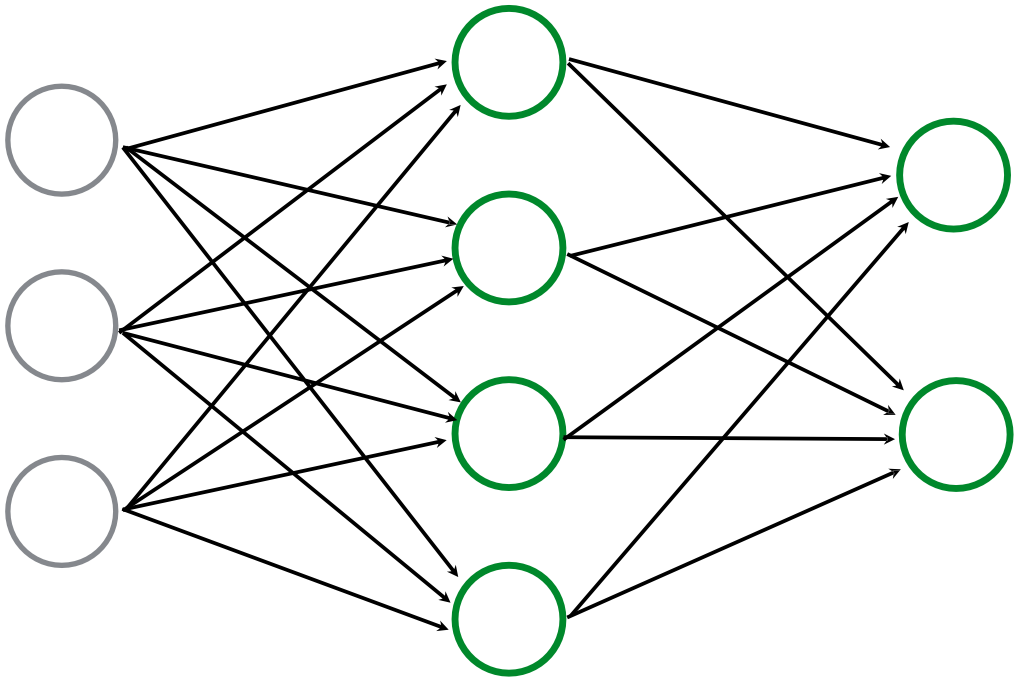
How many weights (edges)?



How many learnable parameters total?

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

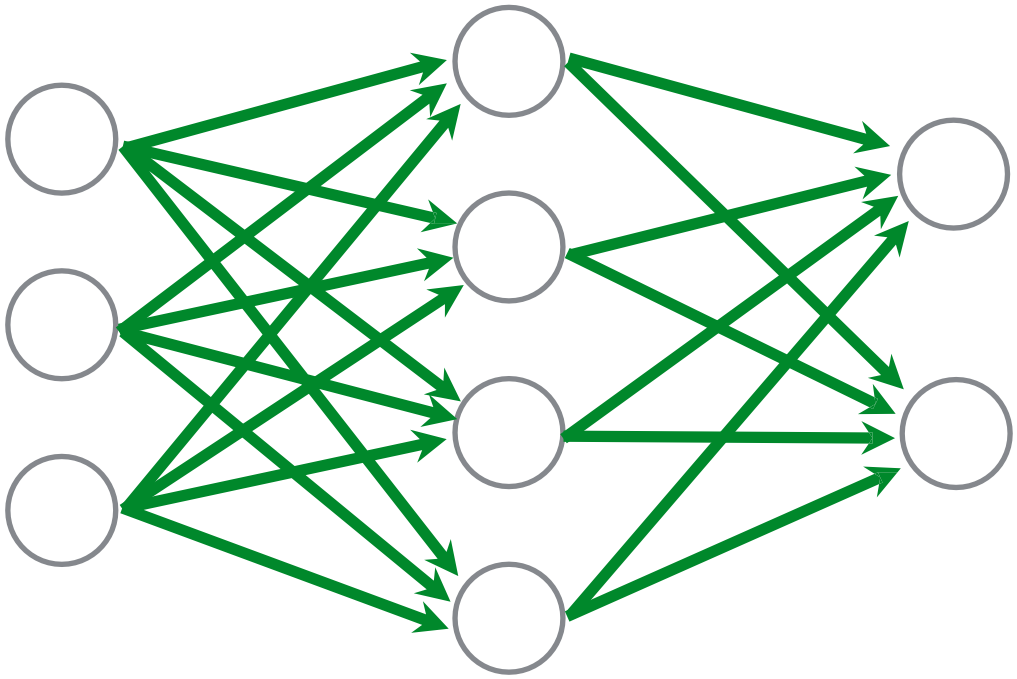
How many weights (edges)?



How many learnable parameters total?

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

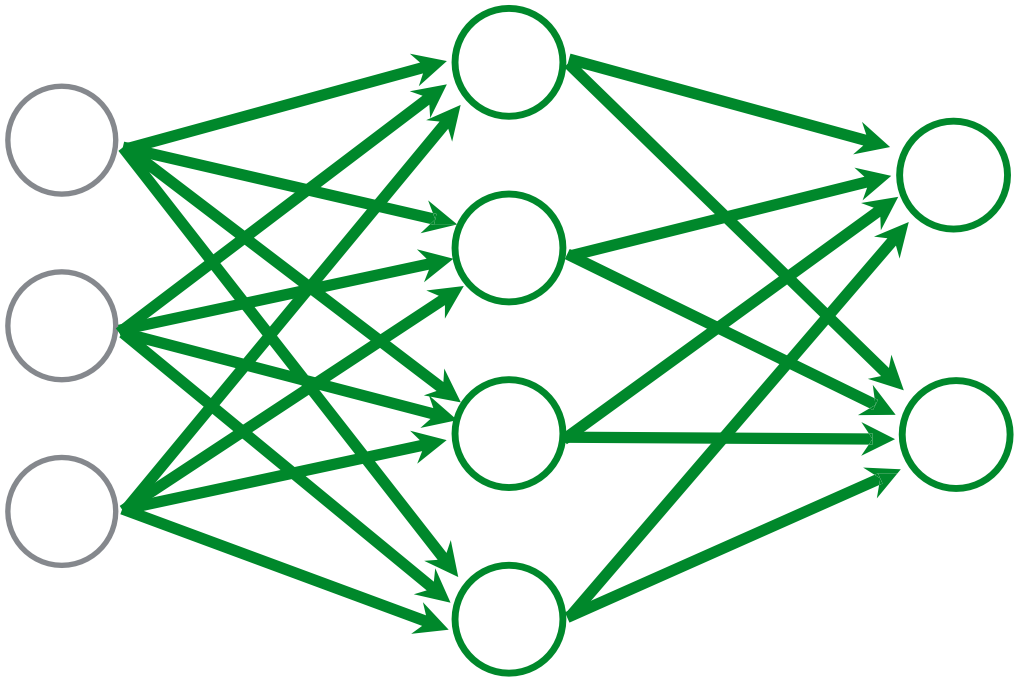
How many weights (edges)? (3 x 4) + (4 x 2) = 20



How many learnable parameters total?

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

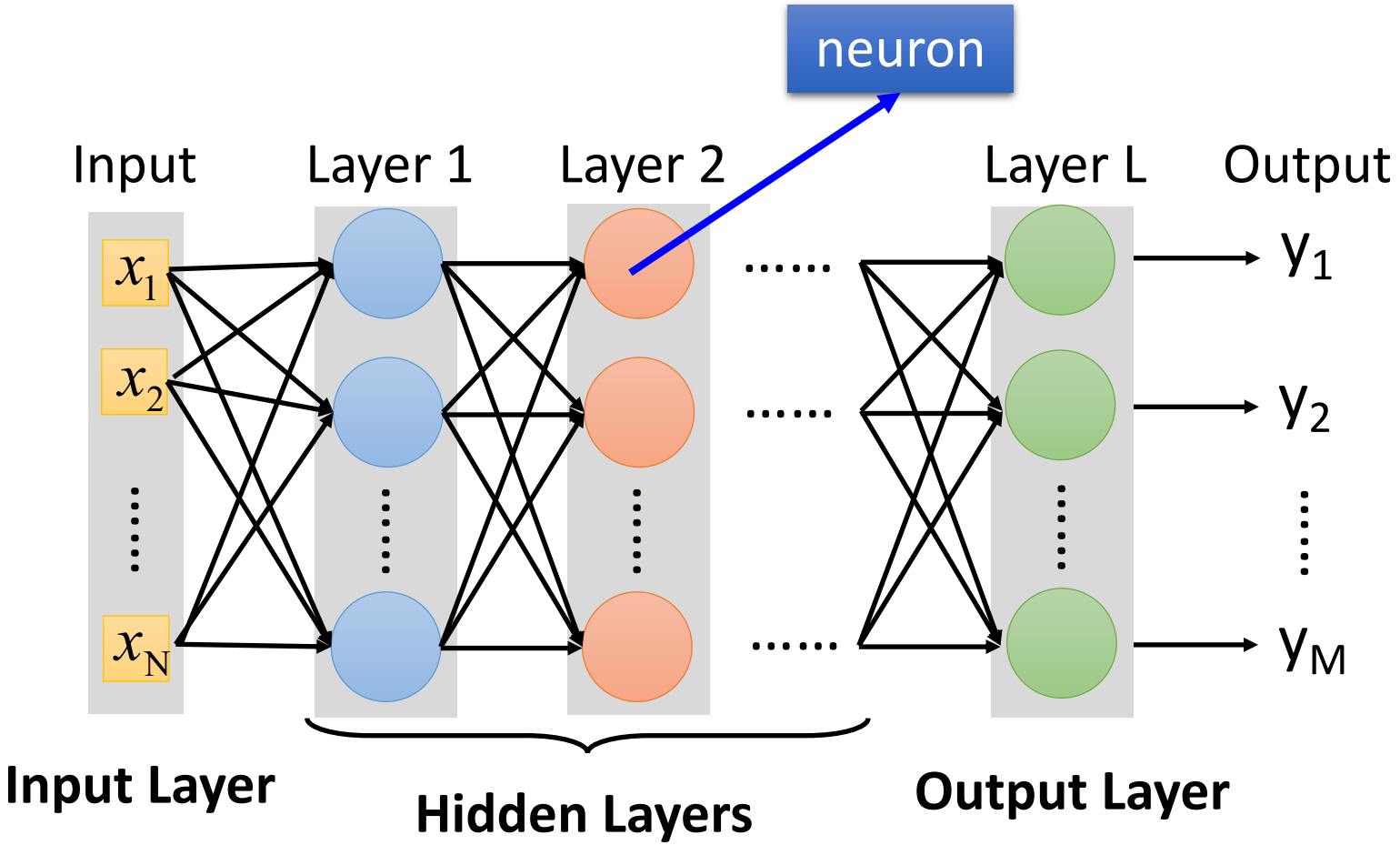
How many weights (edges)? (3 x 4) + (4 x 2) = 20



How many learnable parameters total? 20 + 4 + 2 = 26

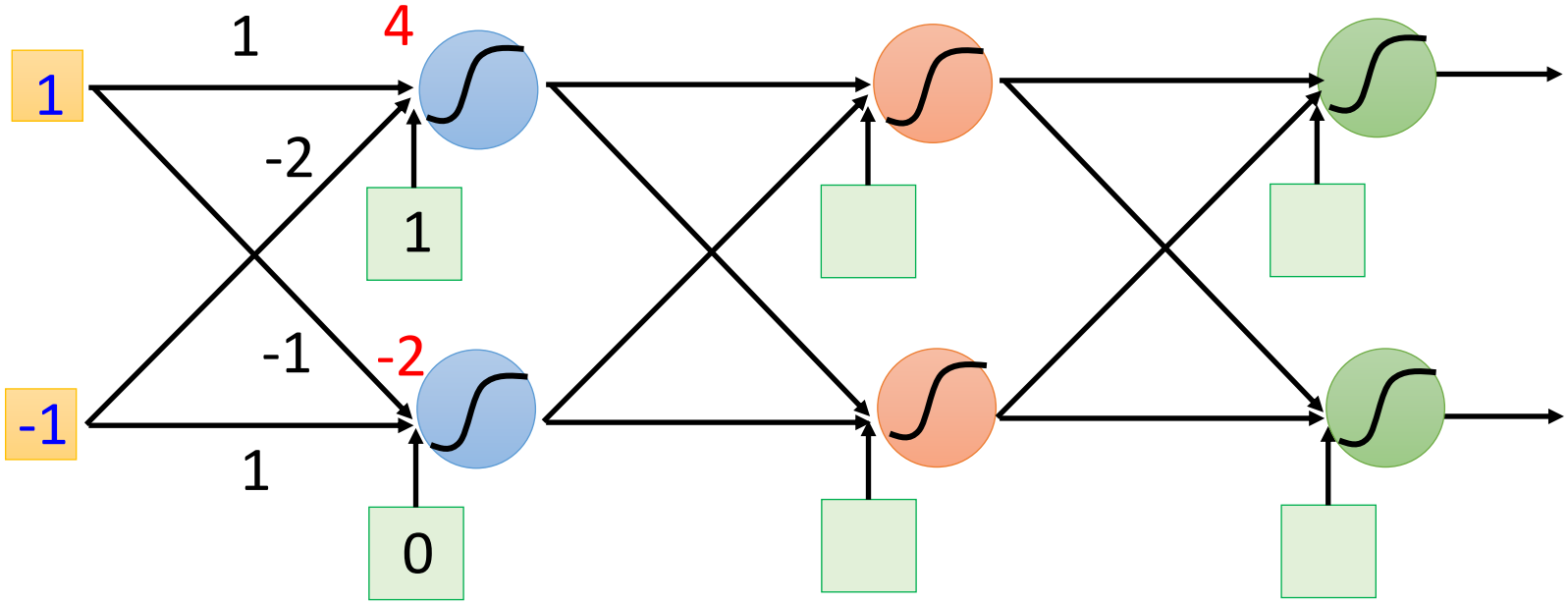
bias terms

Neural Network

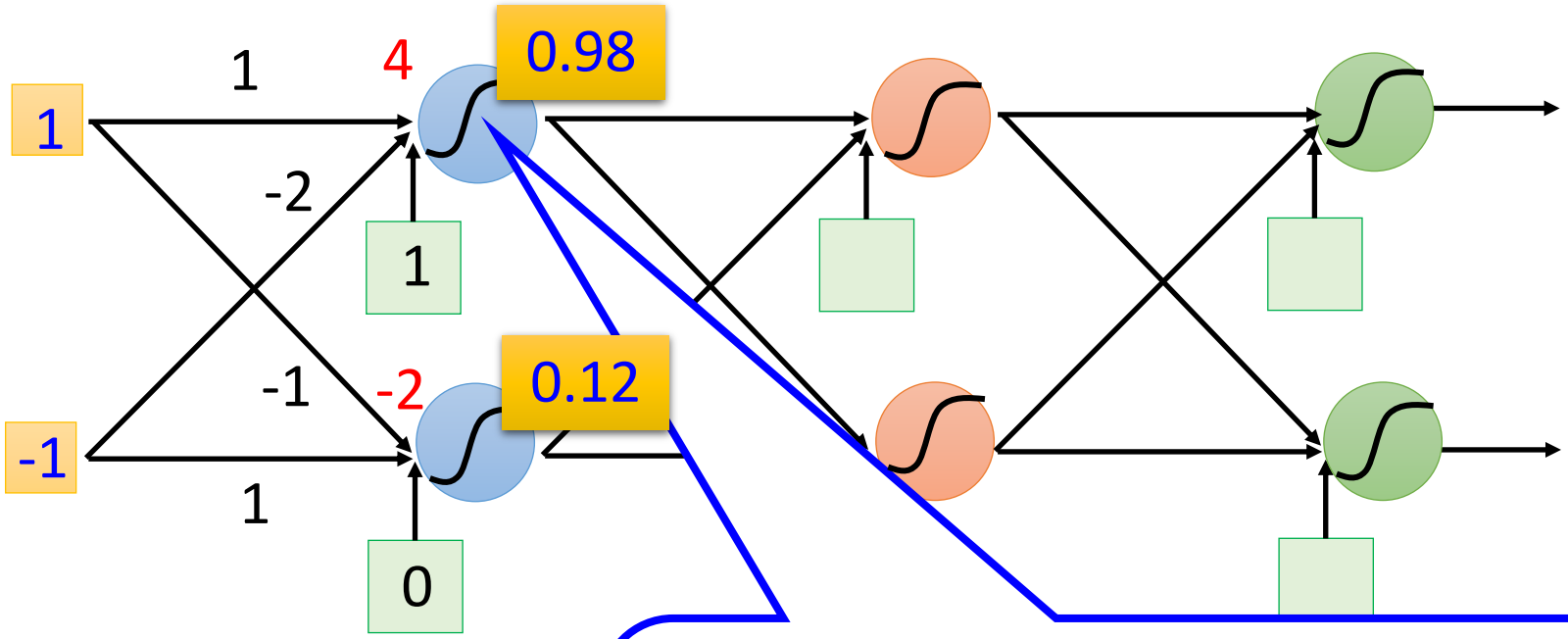


Deep means many hidden layers

Example of Neural Network

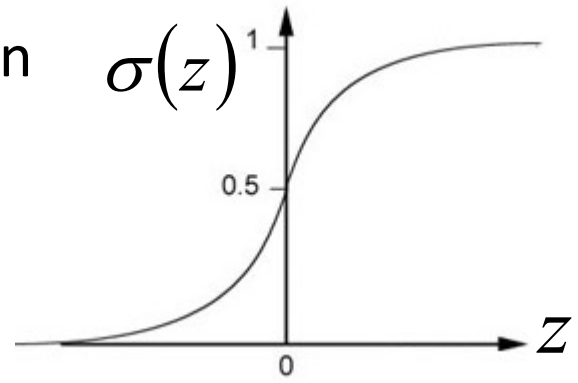


Example of Neural Network



Sigmoid Function

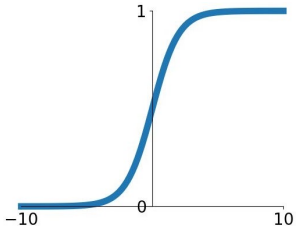
$$\sigma(z) = \frac{1}{1+e^{-z}}$$



Activation functions

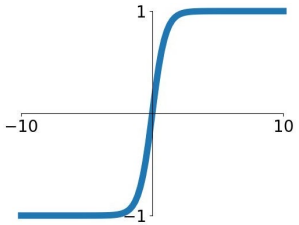
Sigmoid

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



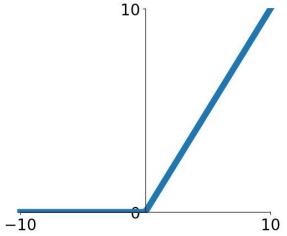
tanh

$$\tanh(x)$$



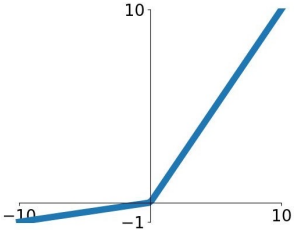
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

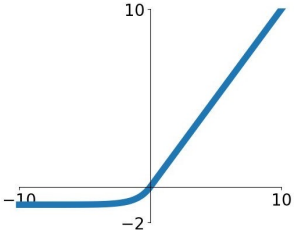


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

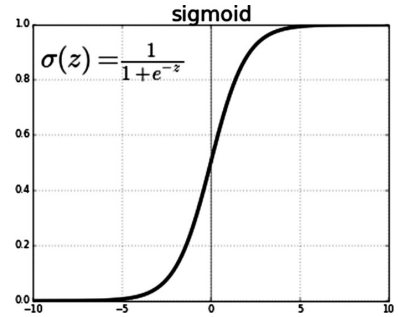
ELU

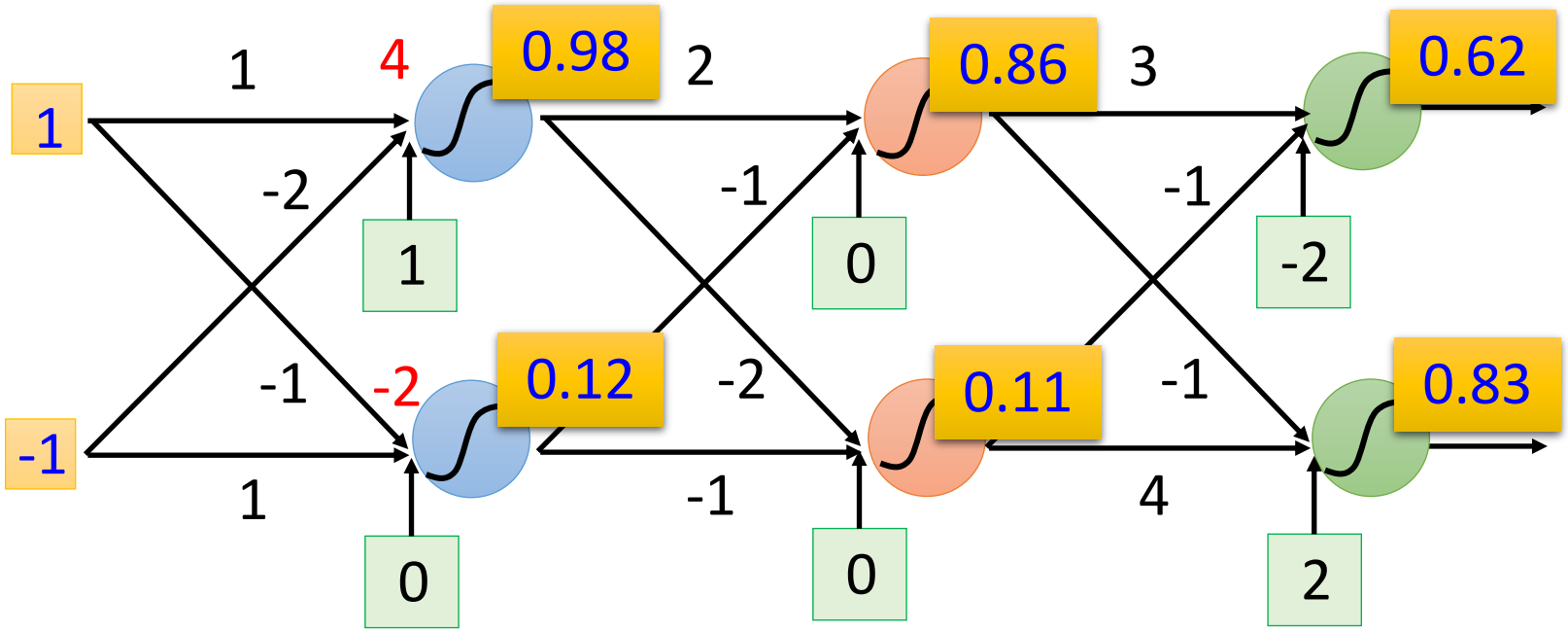
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



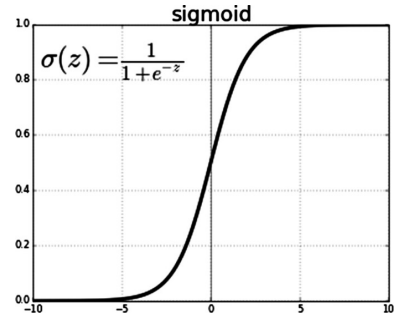
Exponential Linear Unit

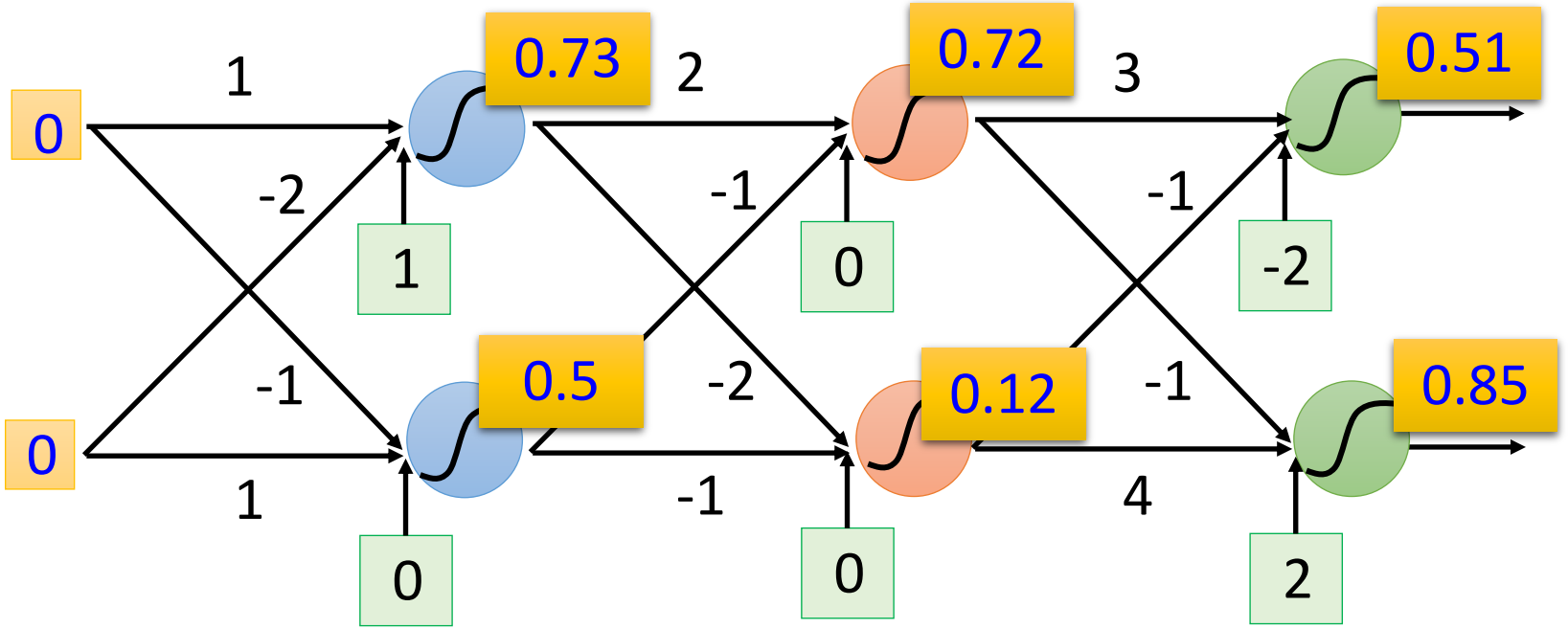
Example of Neural Network





Example of Neural Network



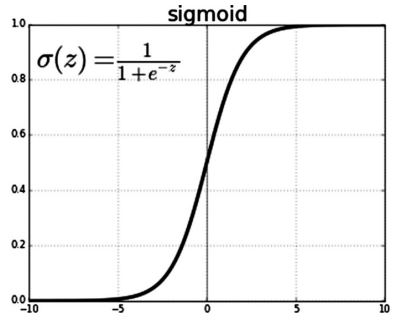


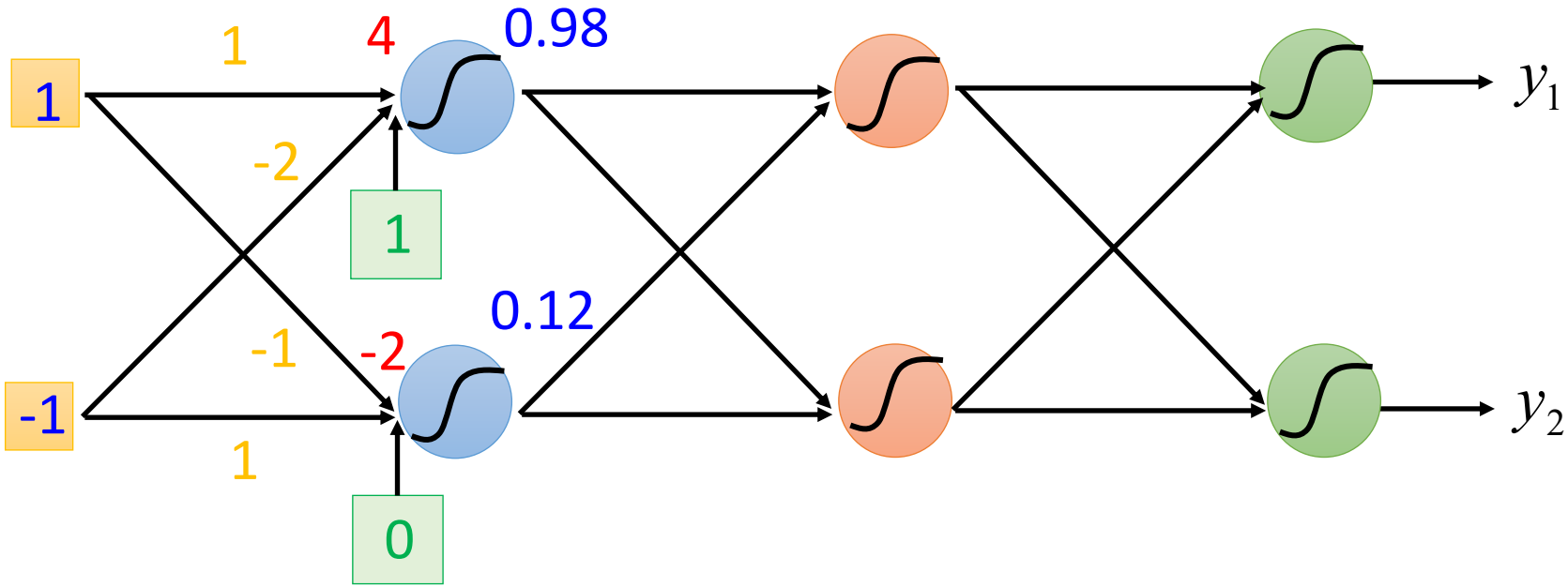
$f: R^2 \rightarrow R^2$

$$f\left(\begin{bmatrix}1\\-1\end{bmatrix}\right) = \begin{bmatrix}0.62\\0.83\end{bmatrix} \quad f\left(\begin{bmatrix}0\\0\end{bmatrix}\right) = \begin{bmatrix}0.51\\0.85\end{bmatrix}$$

Different parameters define different function

Matrix Operation

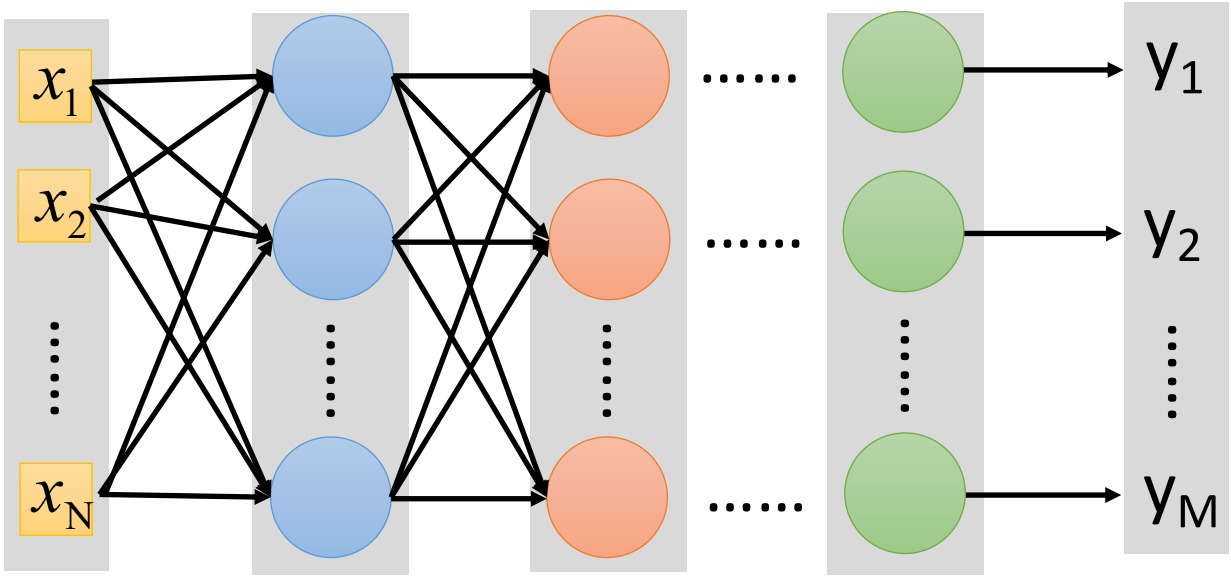




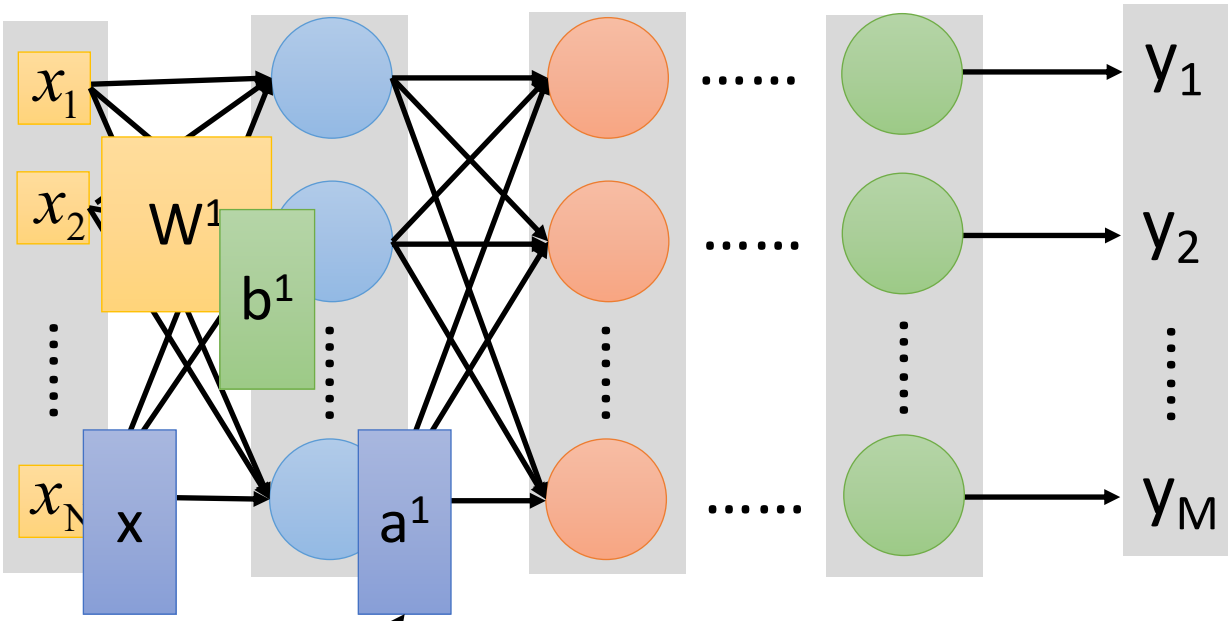
$$\sigma\left(\begin{bmatrix} 1 & -2 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} 0.98 \\ 0.12 \end{bmatrix}$$

$$\begin{bmatrix} 4 \\ -2 \end{bmatrix}$$

Neural Network

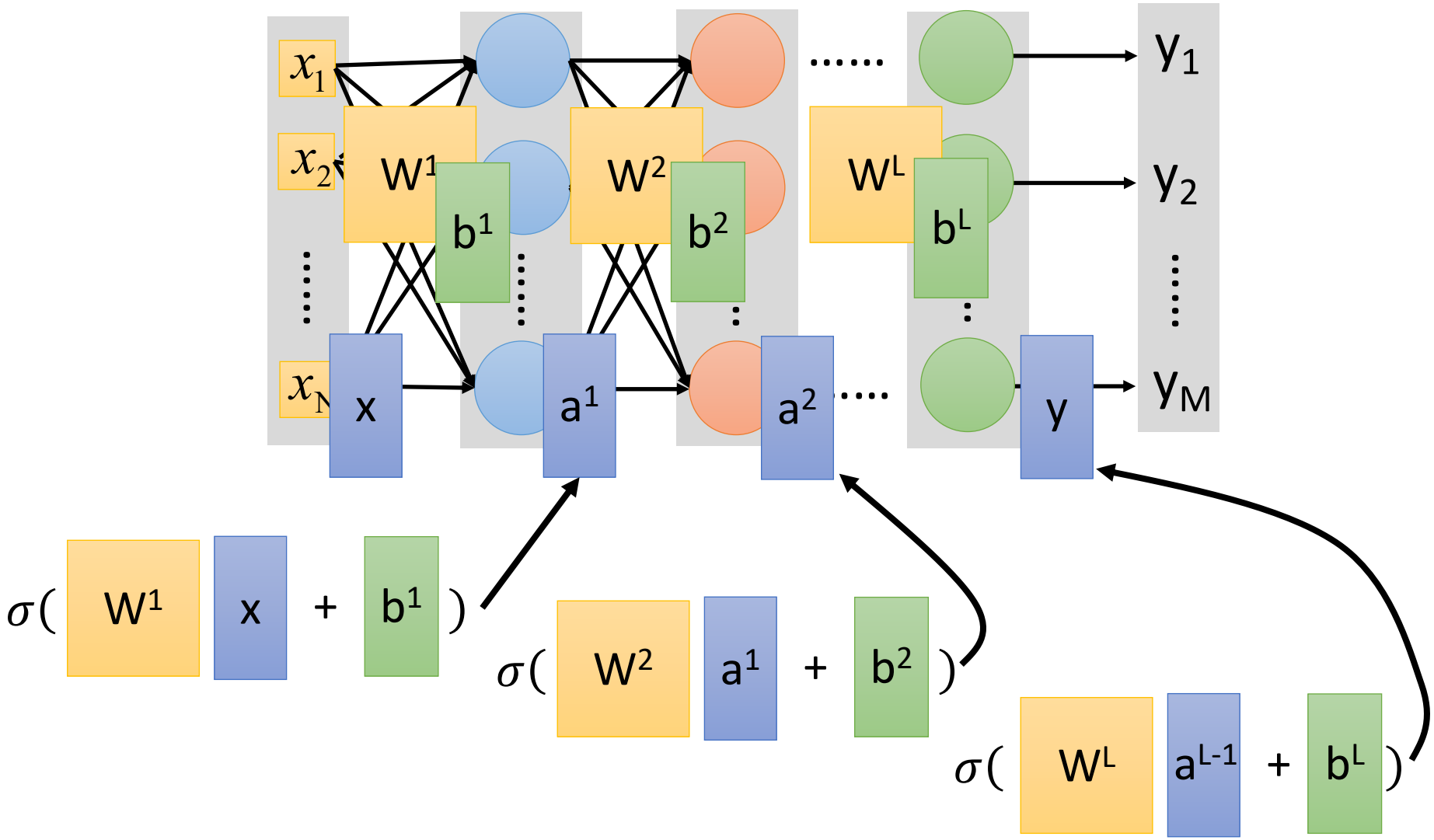


Neural Network

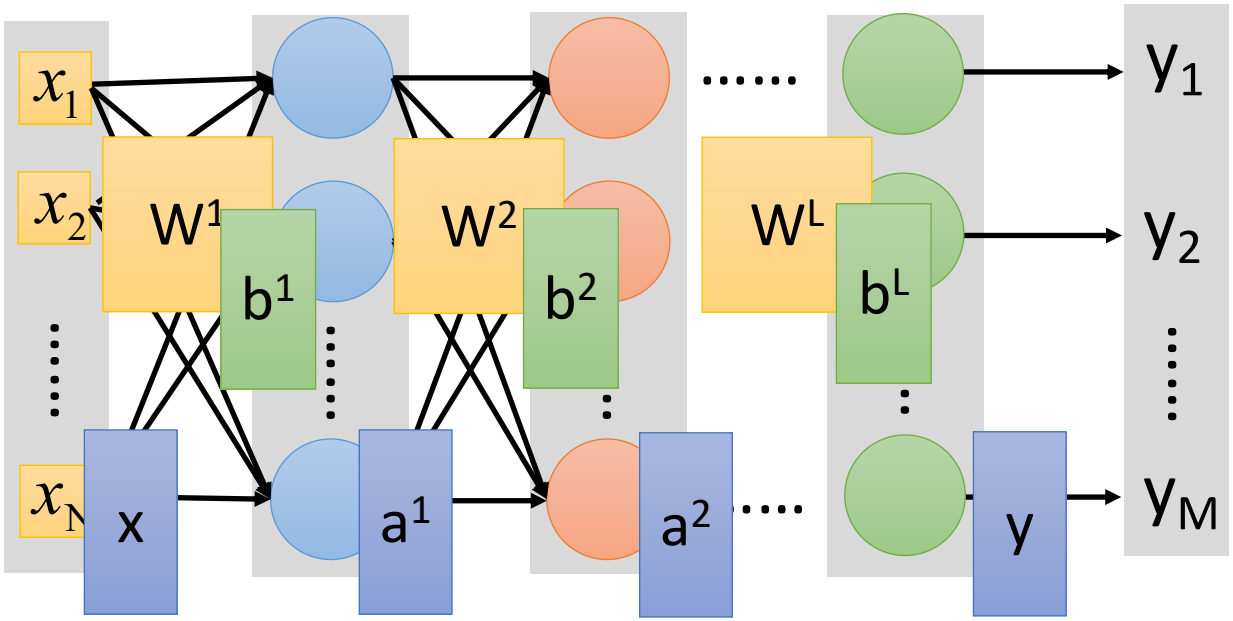


$$\sigma\left(\begin{bmatrix} W^1 \end{bmatrix}\begin{bmatrix} x \end{bmatrix} + \begin{bmatrix} b^1 \end{bmatrix}\right)$$

Neural Network



Neural Network



$$y = f(x)$$

Using parallel computing techniques to speed up matrix operation

$$= \sigma(W^L \cdots \sigma(W^2 \sigma(W^1 x + b^1) + b^2) \cdots + b^L)$$

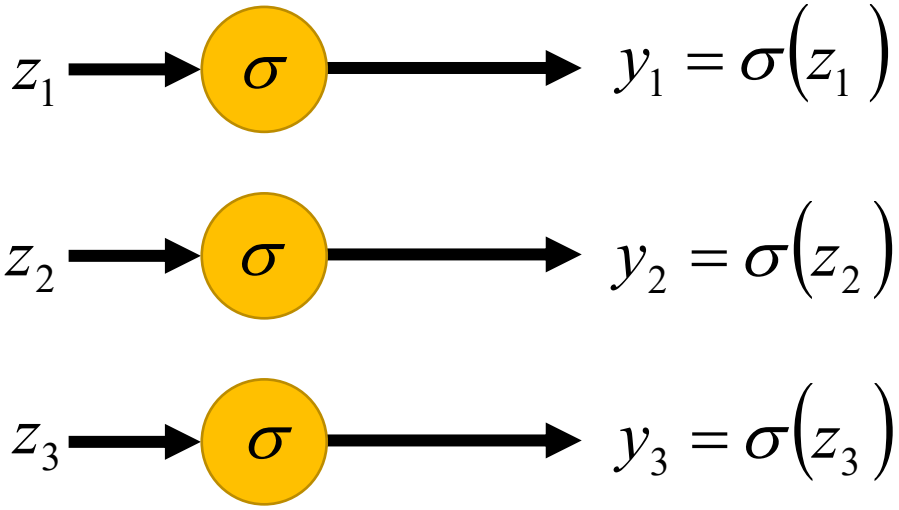
Softmax [Classification]

Probability:

- $1 > y_i > 0$
- $\sum_i y_i = 1$

- Softmax layer as the output layer

Ordinary Layer



In general, the output of network can be any value.

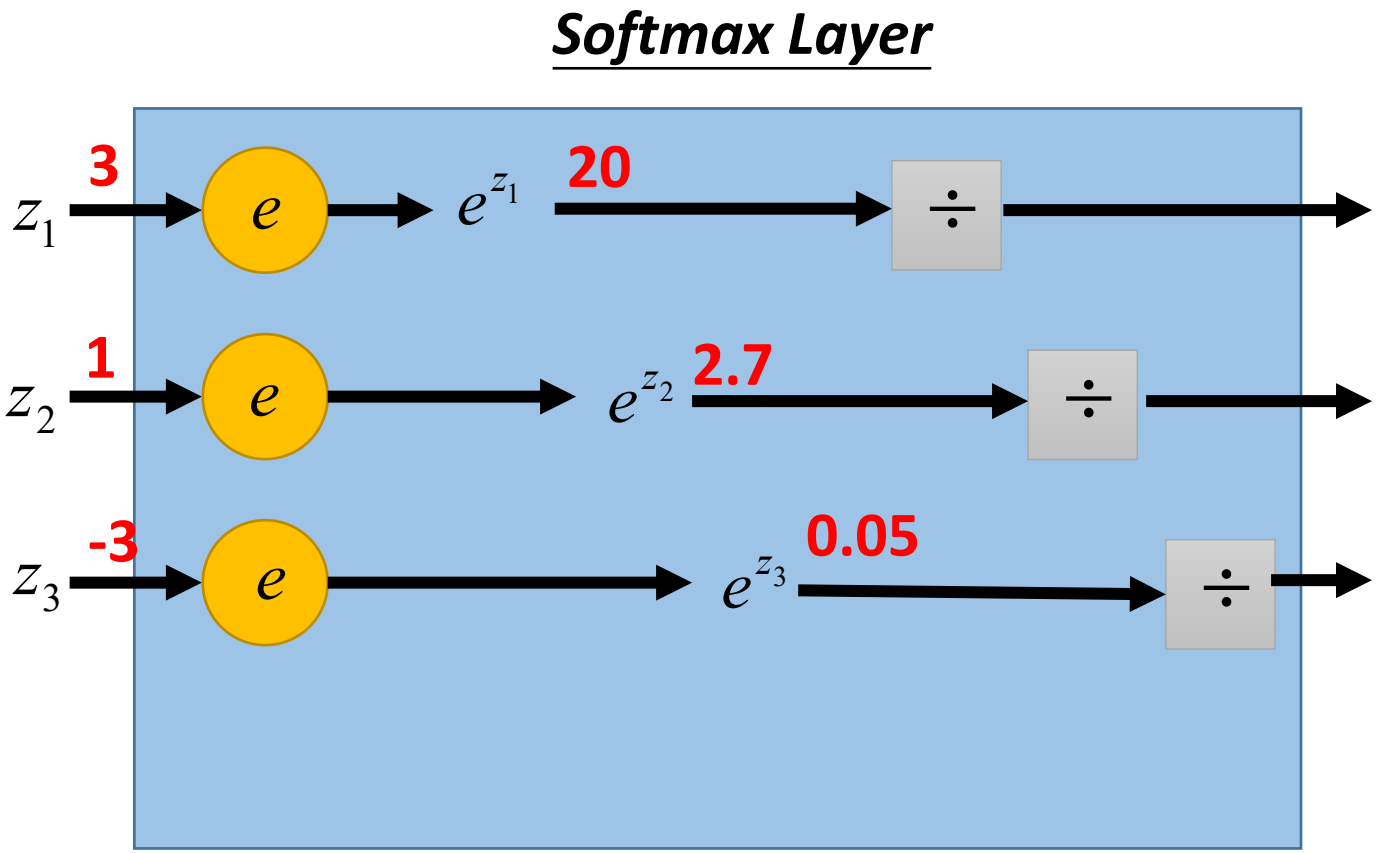
May not be easy to interpret

Softmax [Classification]

Probability:

- $1 > y_i > 0$
- $\sum_i y_i = 1$

- Softmax layer as the output layer

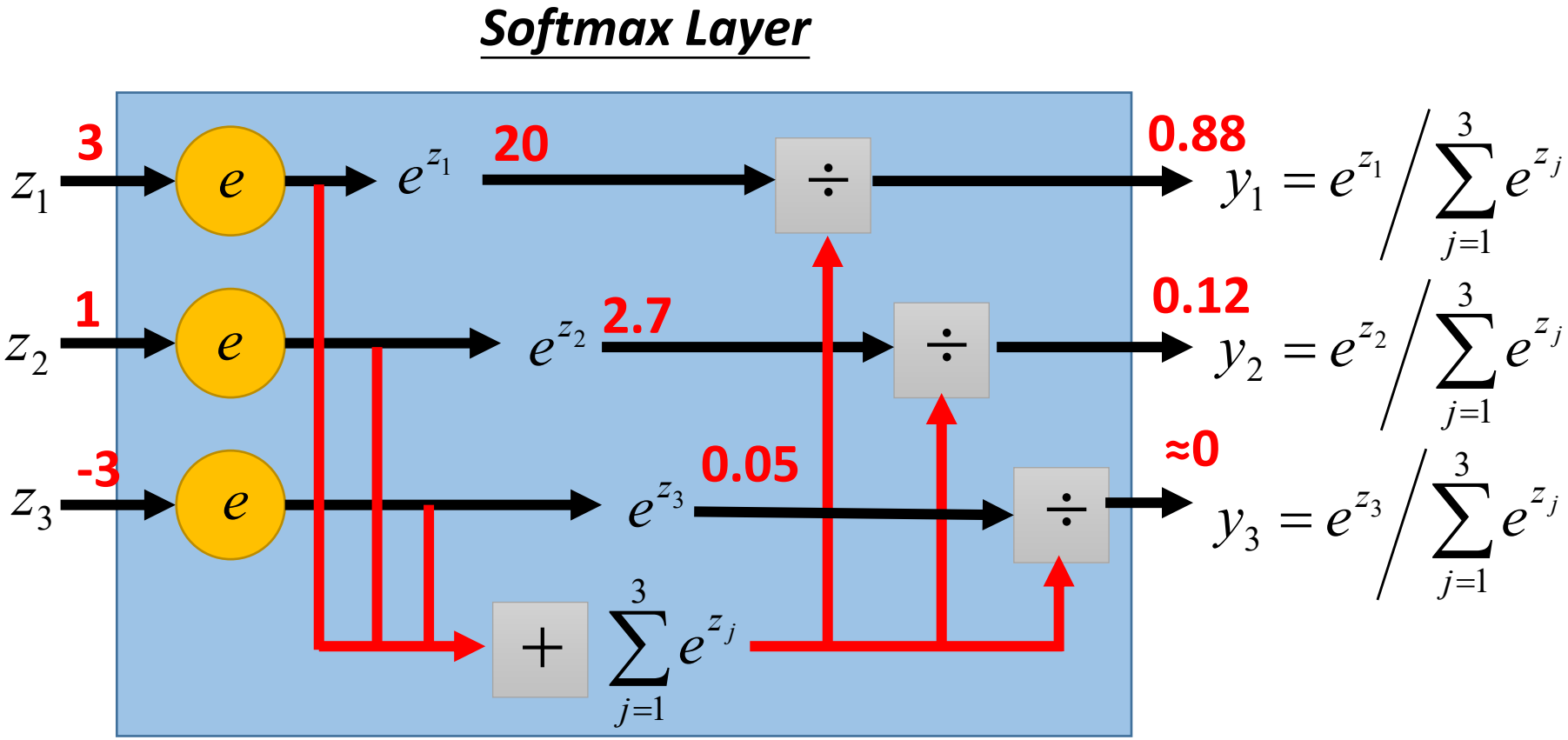


Softmax [Classification]

Probability:

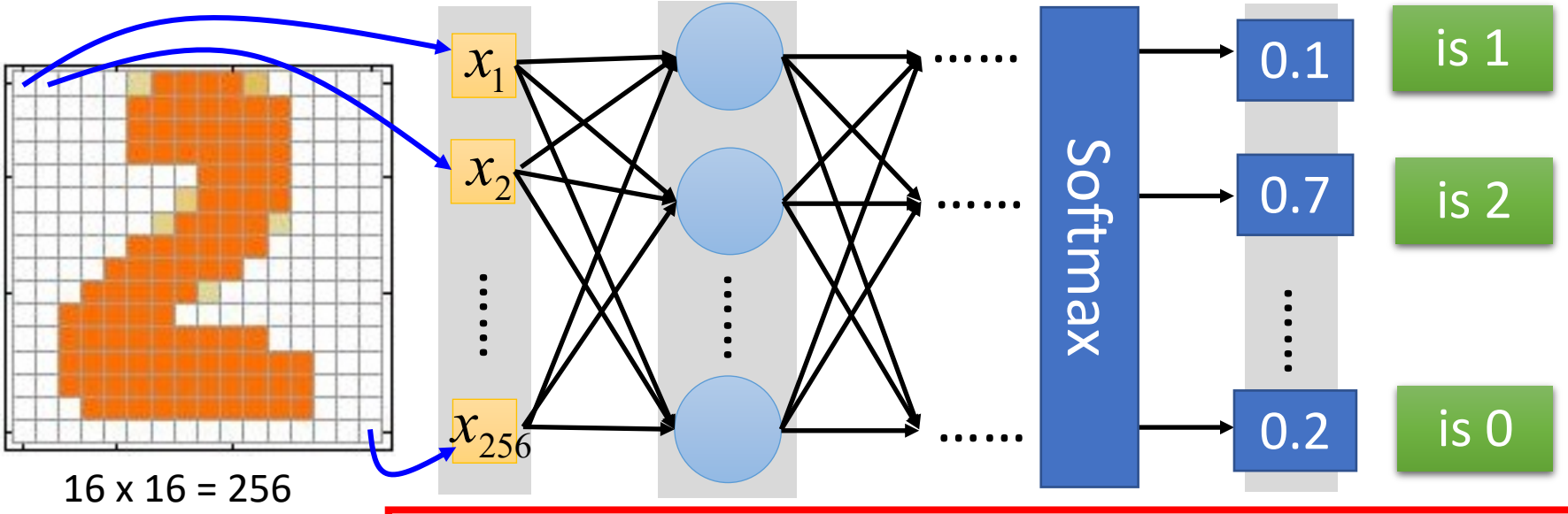
- $1 > y_i > 0$
- $\sum_i y_i = 1$

- Softmax layer as the output layer



How to set network parameters

$$\theta = \{W^1, b^1, W^2, b^2, \cdots W^L, b^L\}$$



16 x 16 = 256

Ink → 1

No ink → 0

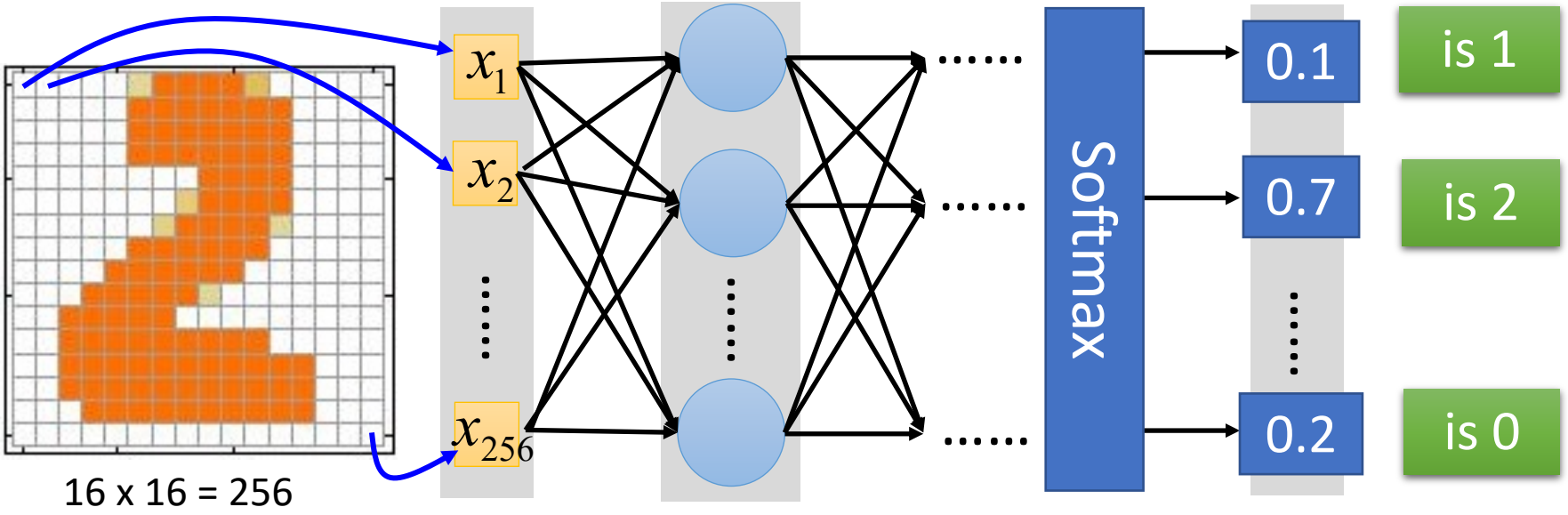
Set the network parameters θ such that ……

Input:  → y_1 has the maximum value

Input:  → y_2 has the maximum value

How to set network parameters

$$\theta = \{W^1, b^1, W^2, b^2, \cdots W^L, b^L\}$$



16 x 16 = 256

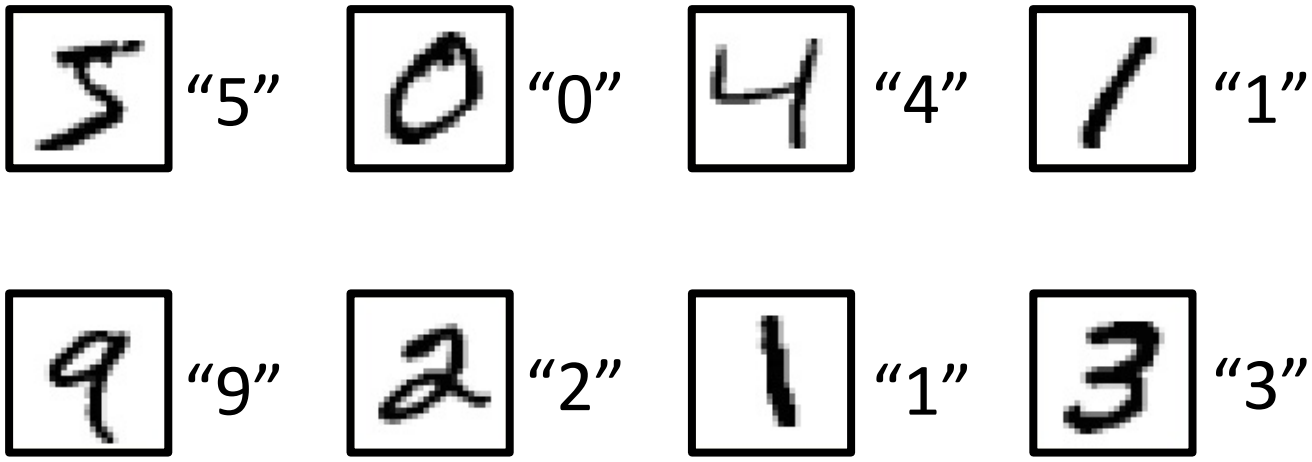
Ink → 1

No ink → 0

How to let the neural network achieve this

Training Data

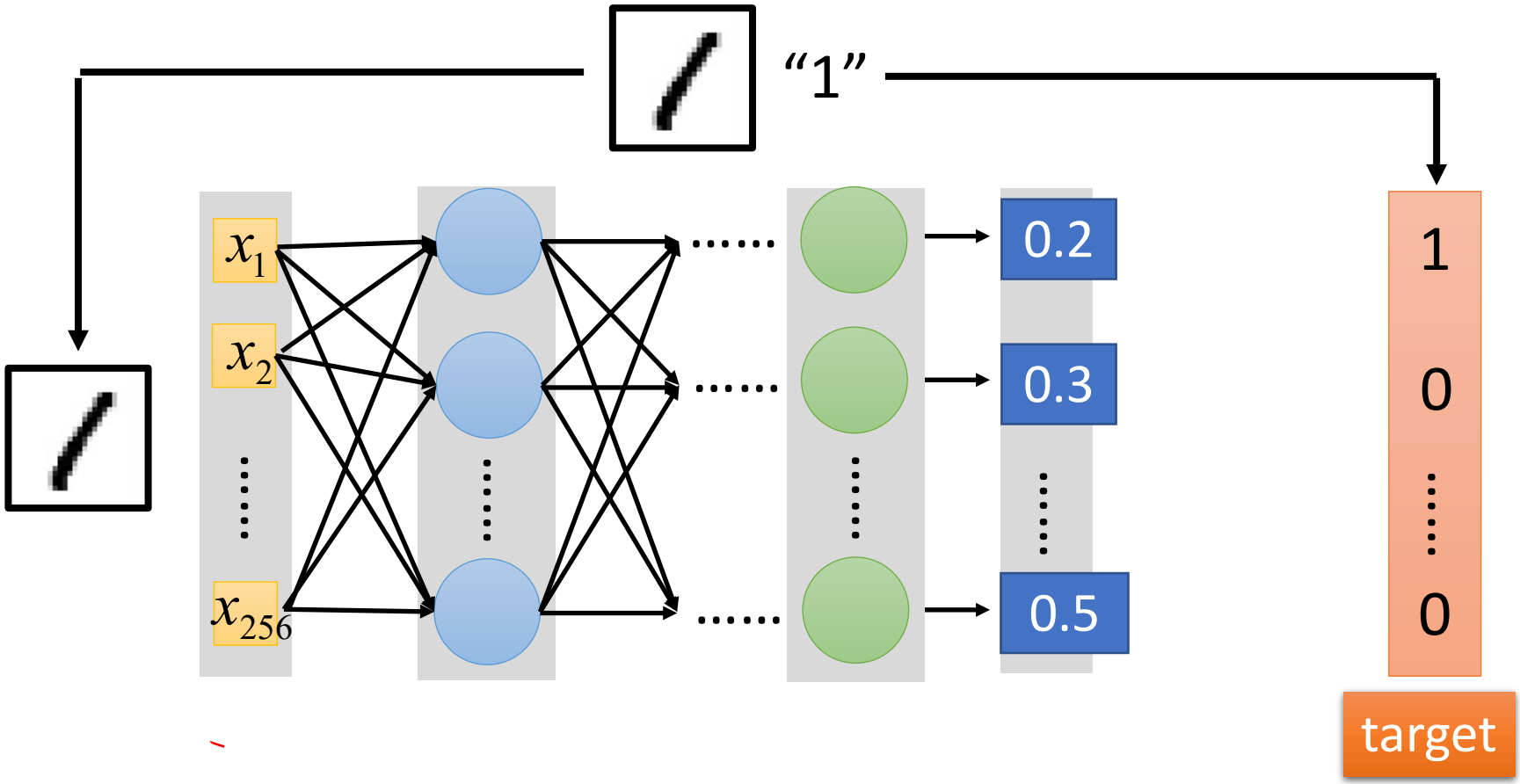
- Preparing training data: images and their labels



Using the training data to find the network parameters.

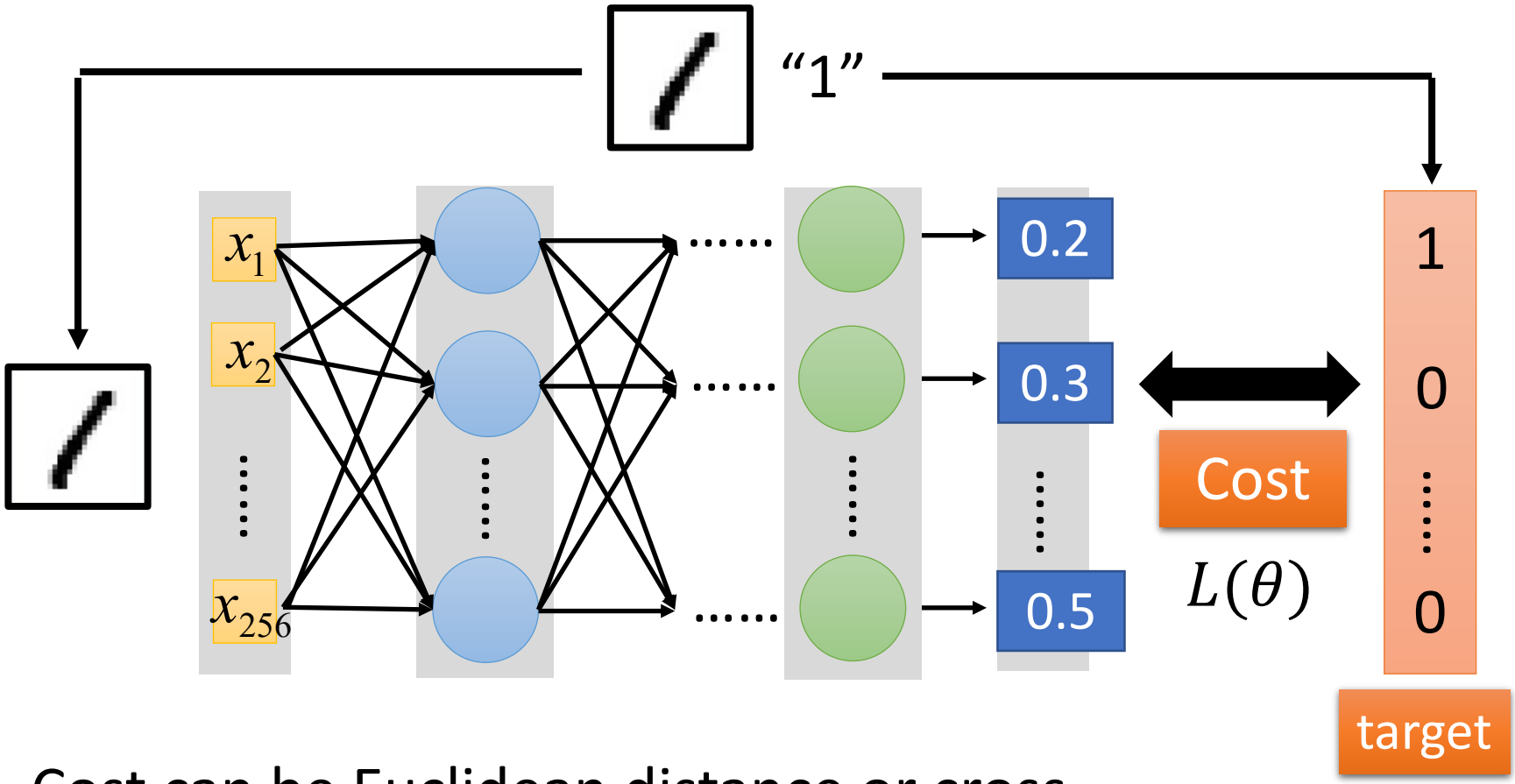
Cost

Given a set of network parameters θ, each example has a cost value.



Cost

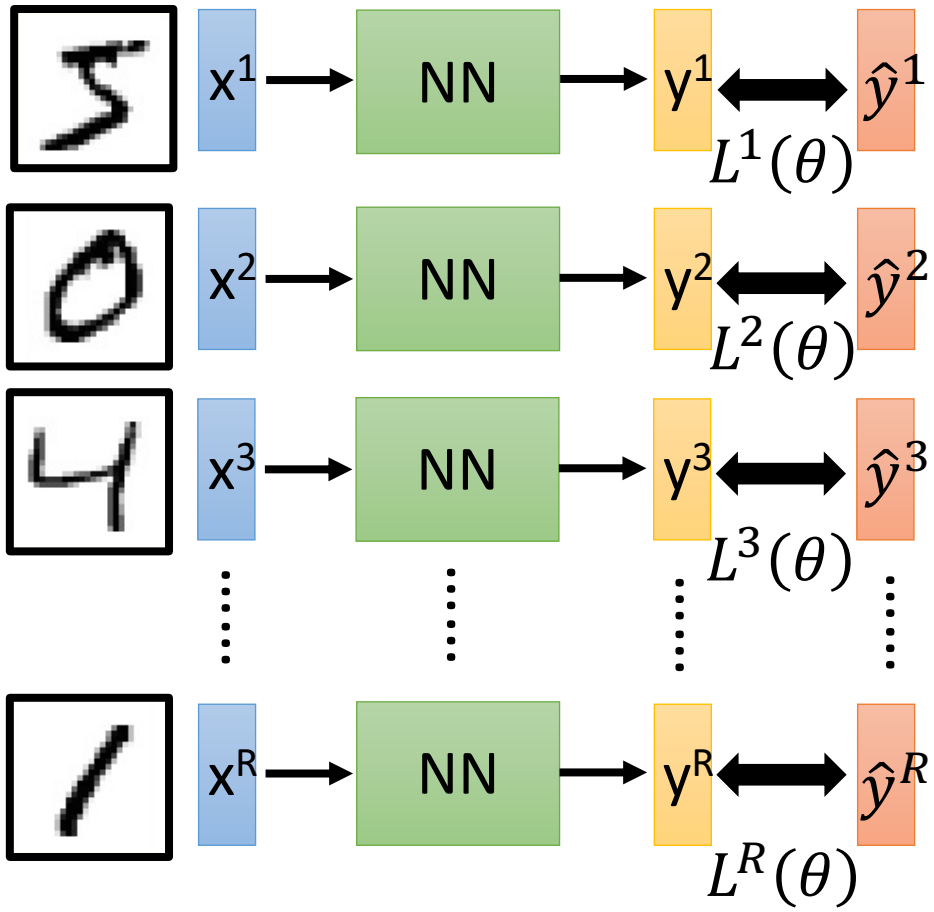
Given a set of network parameters θ, each example has a cost value.



Cost can be Euclidean distance or cross entropy of the network output and target

Total Cost

For all training data …



Total Cost:

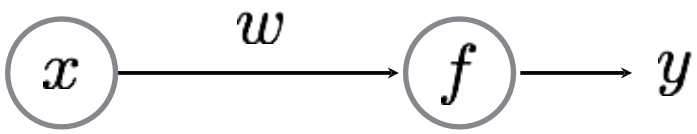
$$C(\theta) = \sum_{r=1}^{R} L^r(\theta)$$

Find the network parameters θ^* that minimize this value

Training Neural Networks

Let's start easy

World's smallest perceptron!



$$y = wx$$

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

Learning a Perceptron

Given a set of samples and a Perceptron

$$\{x_i, y_i\}$$

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

what is this activation function?

linear function! $f(x) = wx$

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?

$$y = wx$$

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

$$\hat{y} = wx$$

what does this mean?

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

perceptron parameter

perceptron output

true label

Before diving into gradient descent, we need to understand …

Loss Function

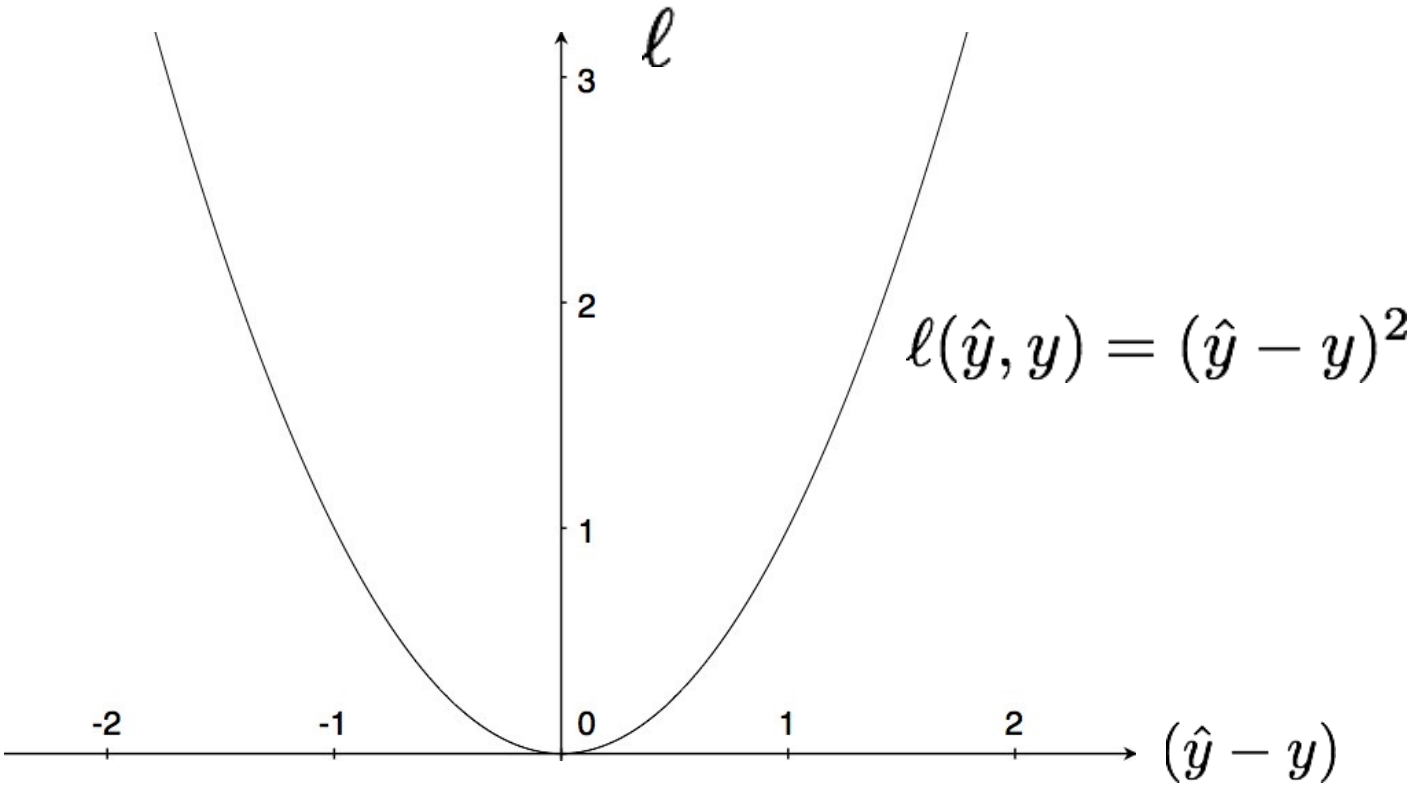
Defines what it means to be close to the true solution

YOU get to choose the loss function!

(some are better than others depending on what you want to do)

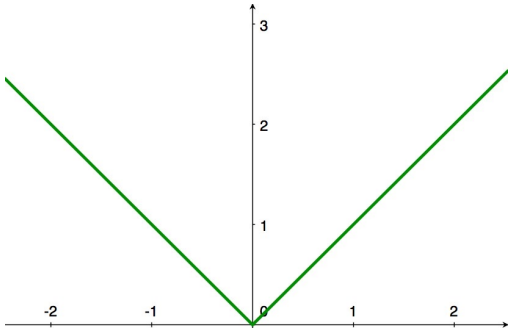
Squared Error (L2)

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



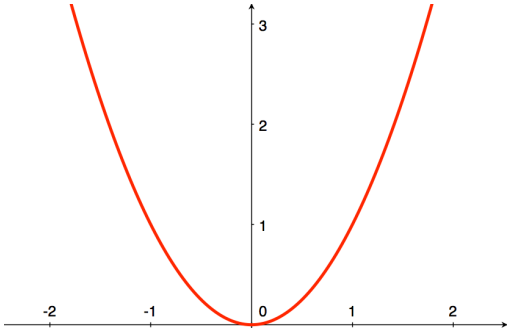
L1 Loss

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



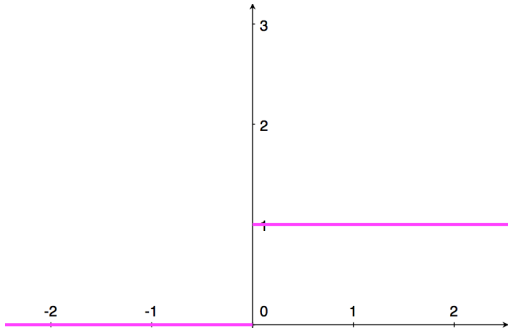
L2 Loss

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



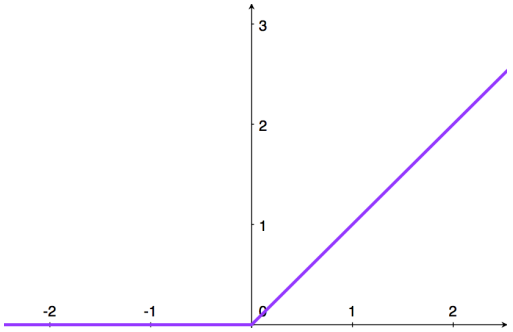
Zero-One Loss

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



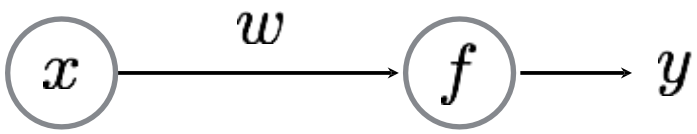
Hinge Loss

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



Back to the…

World's Smallest Perceptron!



$$y = wx$$

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

function of ONE parameter!

Learning 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:

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

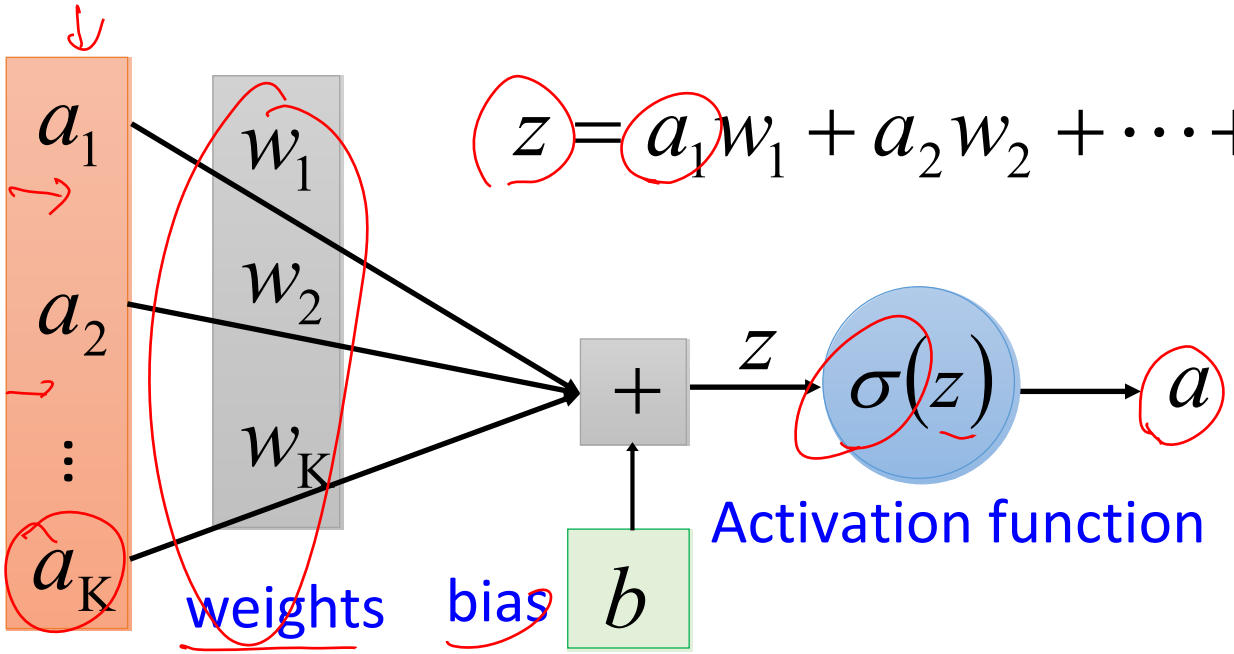
just one line of code!

Now where does this come from?

Review

@10-11-22

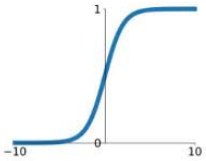
Perceptron **Neuron** $f: R^K \to R$



$$z = a_1 w_1 + a_2 w_2 + \cdots + a_K w_K + b$$

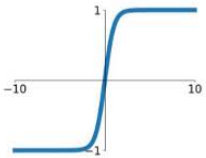
Sigmoid

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



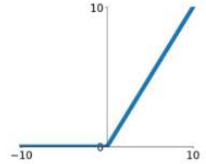
tanh

$\tanh(x)$



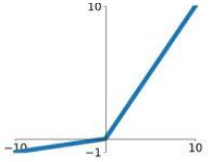
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

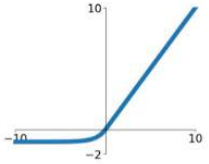


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

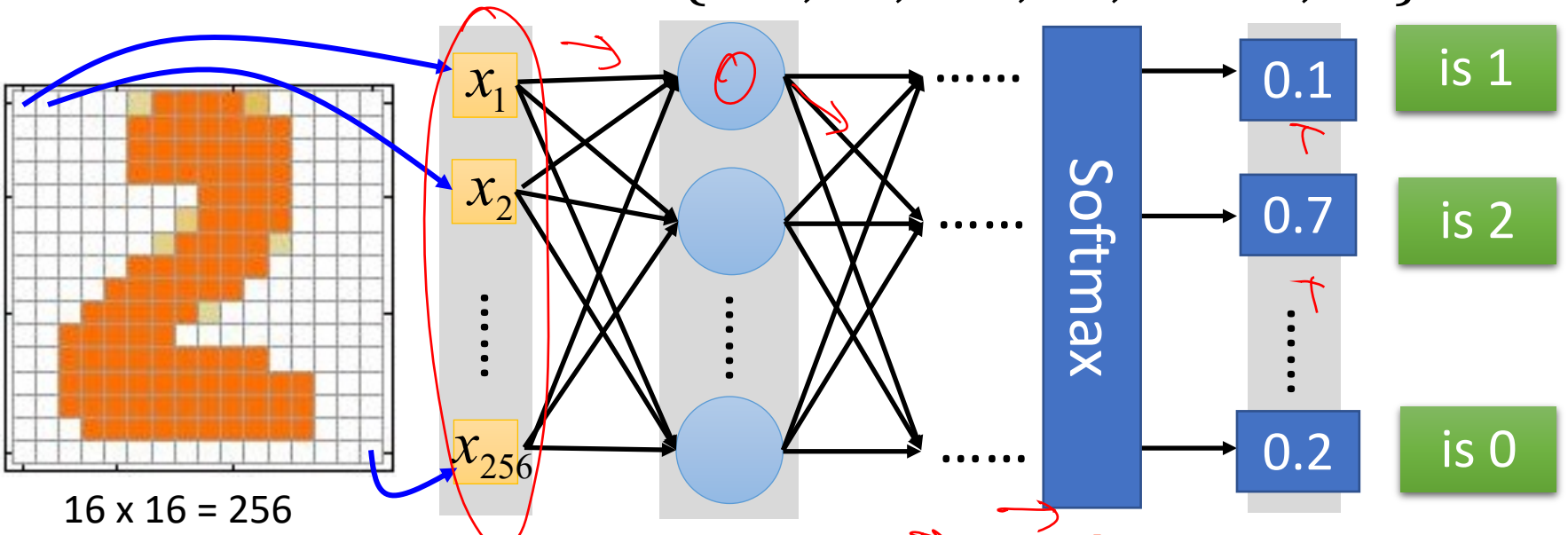
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Neural Network

$$\theta = \{W^1, b^1, W^2, b^2, \cdots W^L, b^L\}$$



16 x 16 = 256

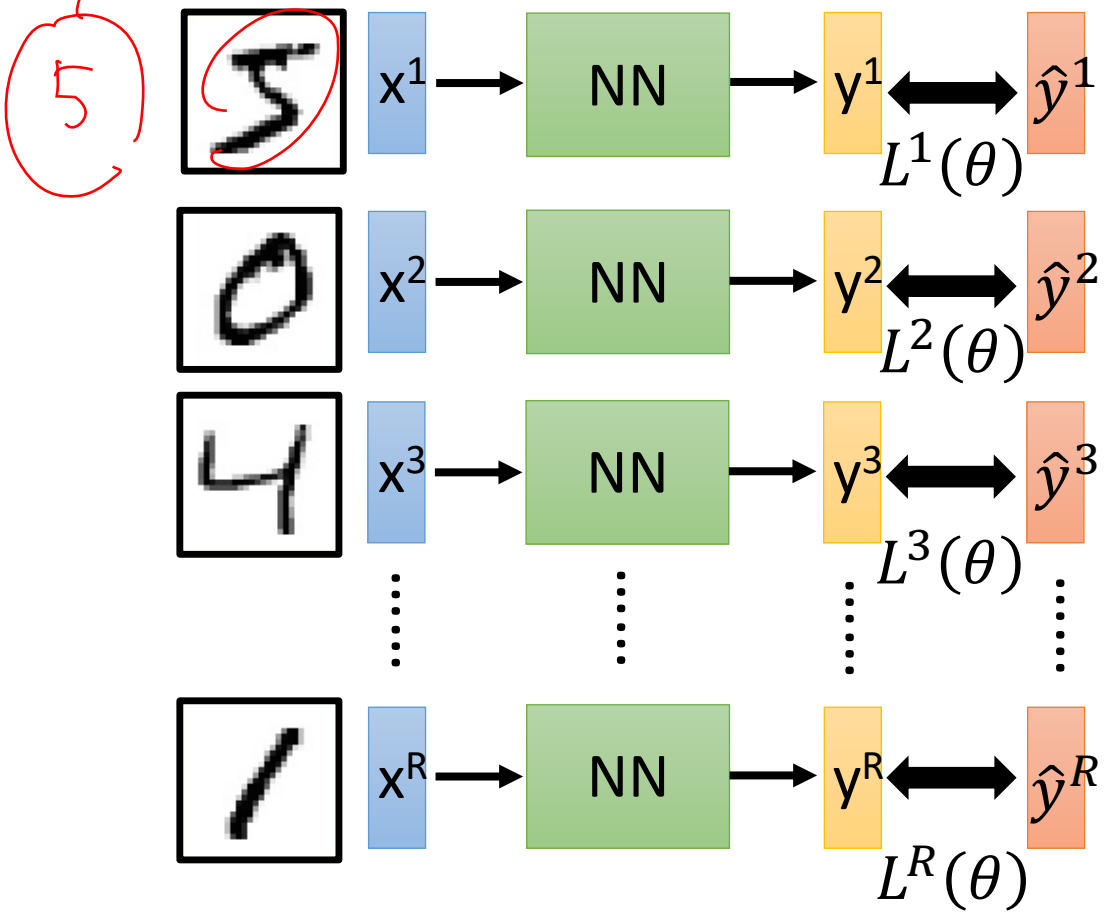
Ink → 1

No ink → 0

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Total Cost

For all training data …



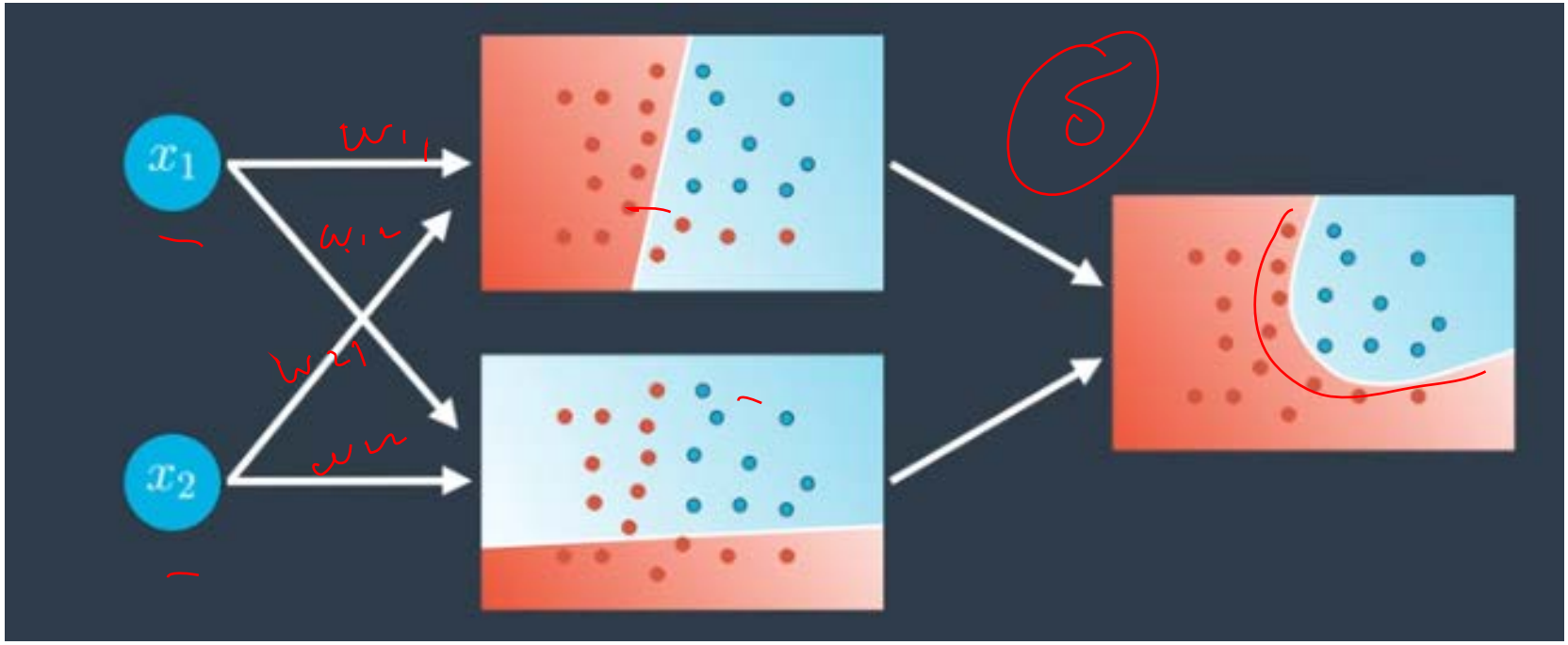
Cross entropy loss

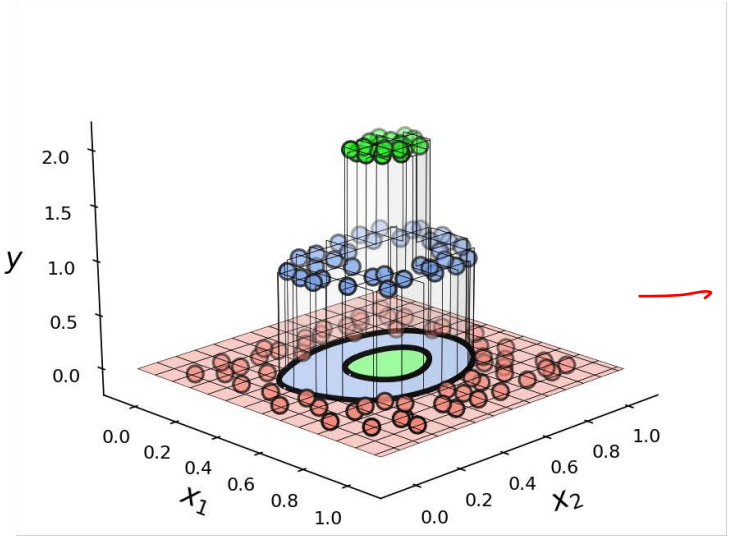
$$L = -\frac{1}{m}\sum_{i=1}^{m} y_i \cdot \log(\hat{y}_i)$$

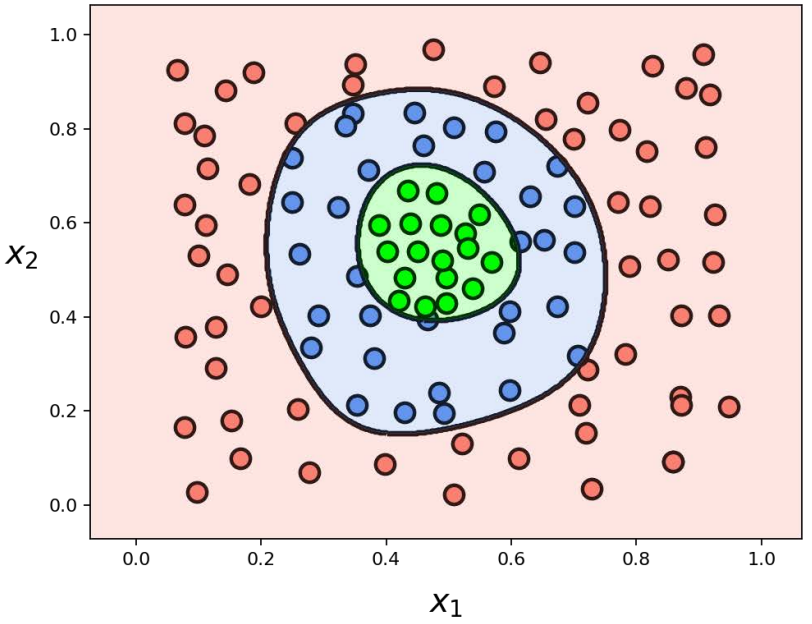
Total Cost:

$$C(\theta) = \sum_{r=1}^{R} L^r(\theta)$$

Find the network parameters θ^* that minimize this value

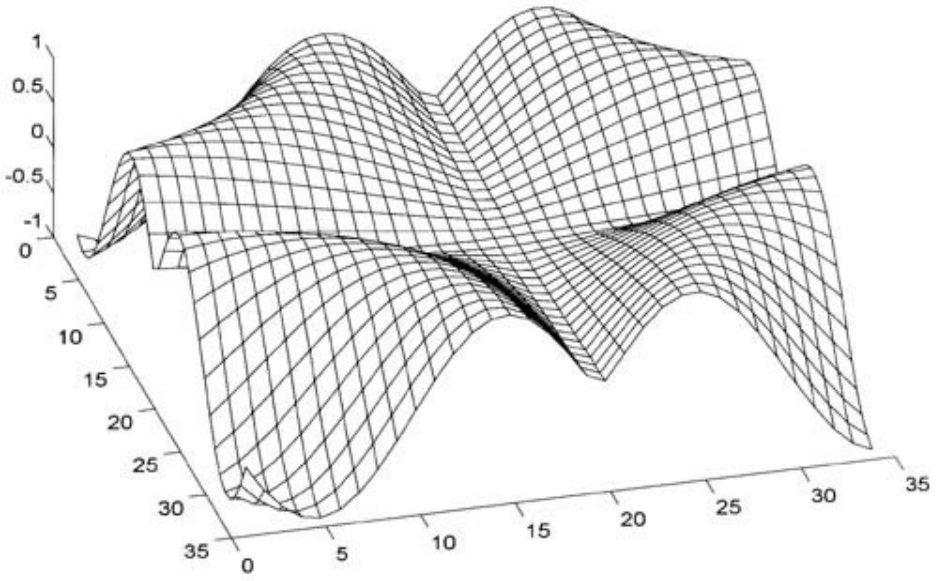

x_1
x_2


y
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8
1.0
x_1
x_2

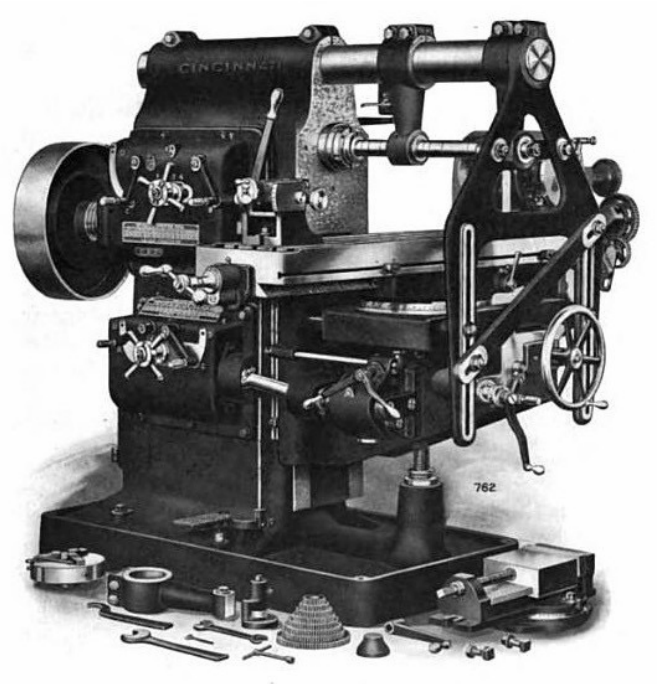

x_2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
x_1

Gradient descent

Two ways to think about them:

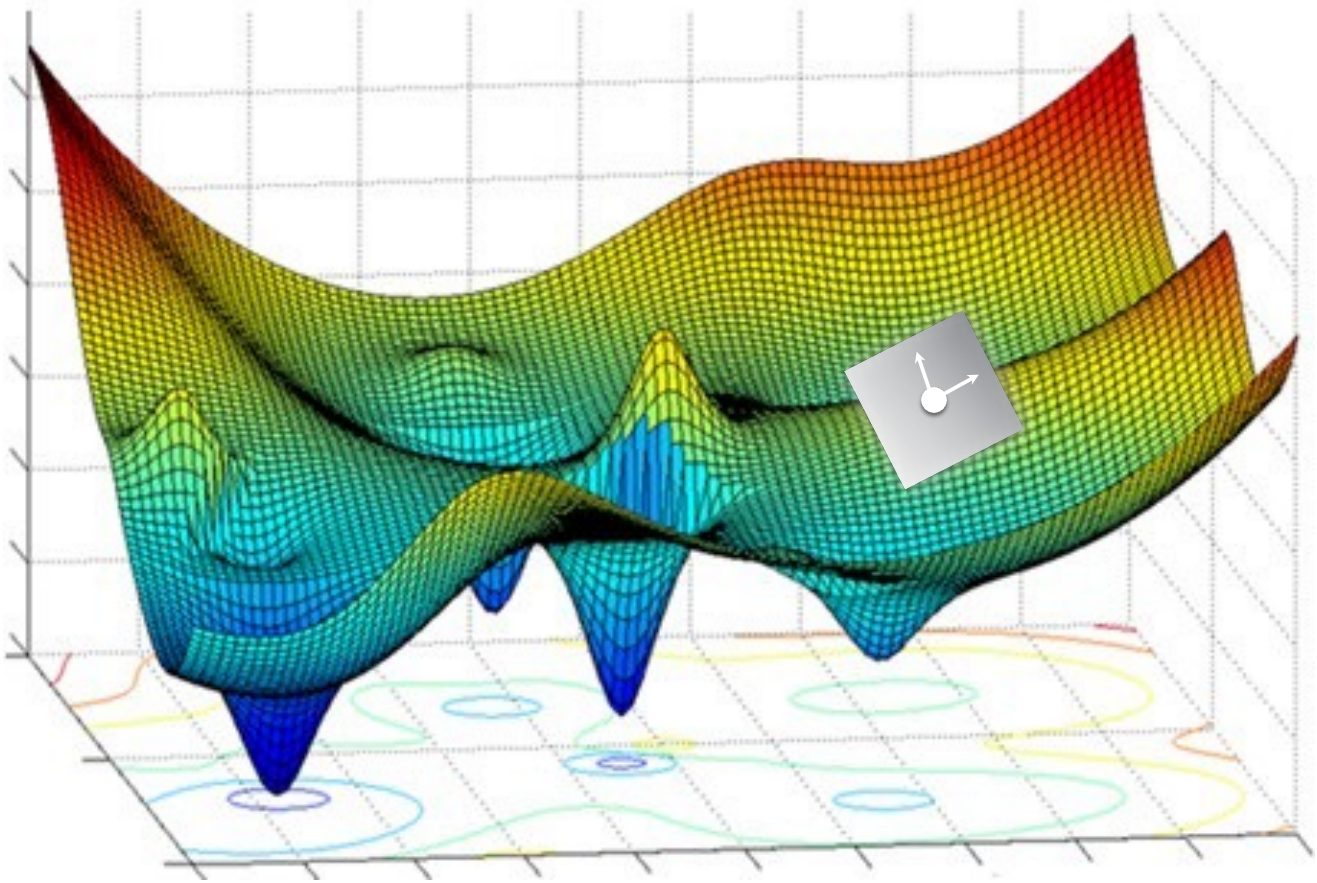


Slope of a function



Knobs on a machine

1. Slope of a function:



$$\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} = \left[\frac{\partial f(\boldsymbol{x})}{\partial x}, \frac{\partial f(\boldsymbol{x})}{\partial y}\right]$$

Describes the slope around a point

2. Knobs on a machine:

input $\boldsymbol{x}$

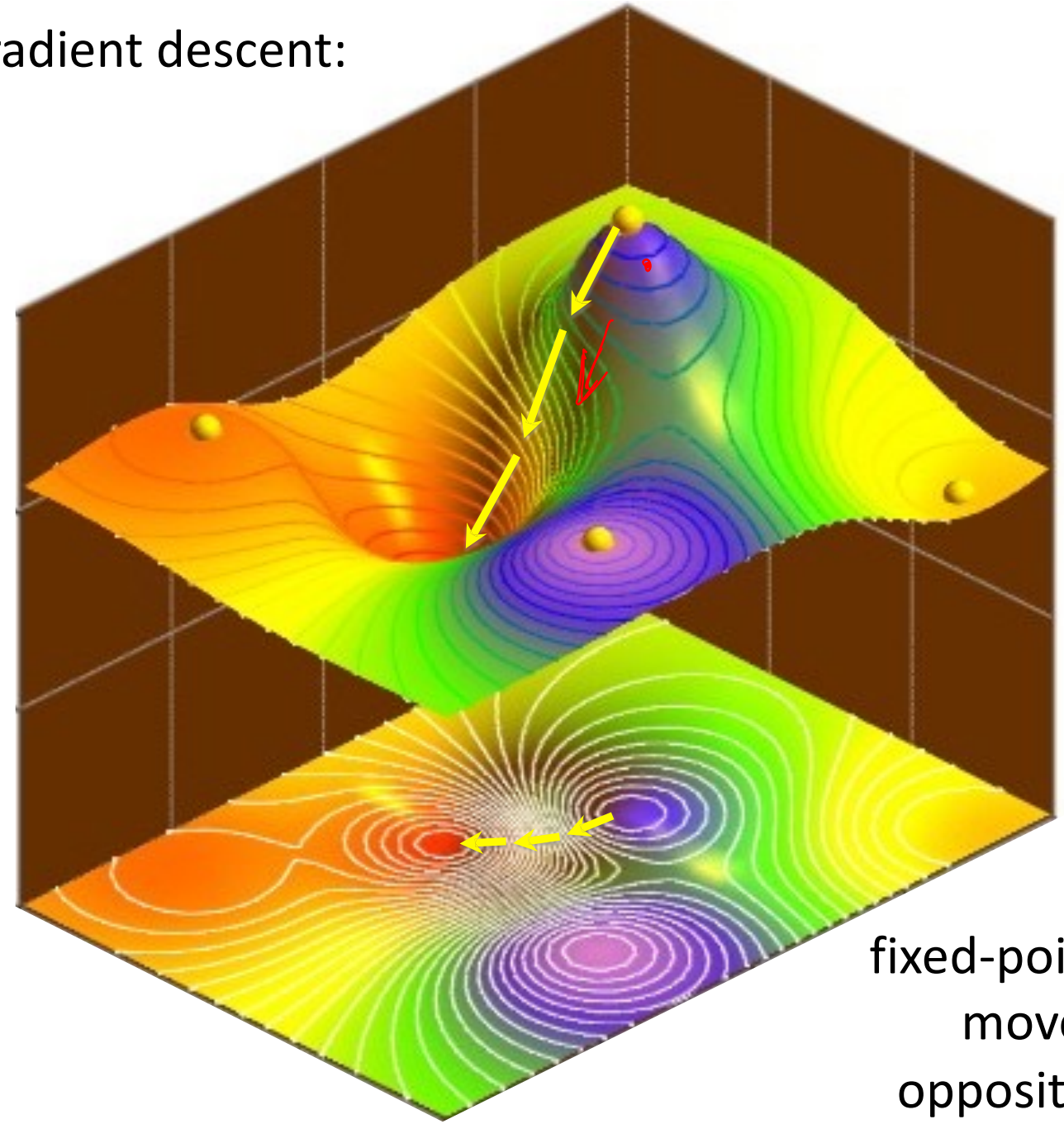
output $f(\boldsymbol{x}; \boldsymbol{w})$

describes how each 'knob' affects the output

$$\frac{\partial f(x)}{\partial w_1} \quad \frac{\partial f(x)}{\partial w_2} \quad \frac{\partial f(x)}{\partial w_3}$$

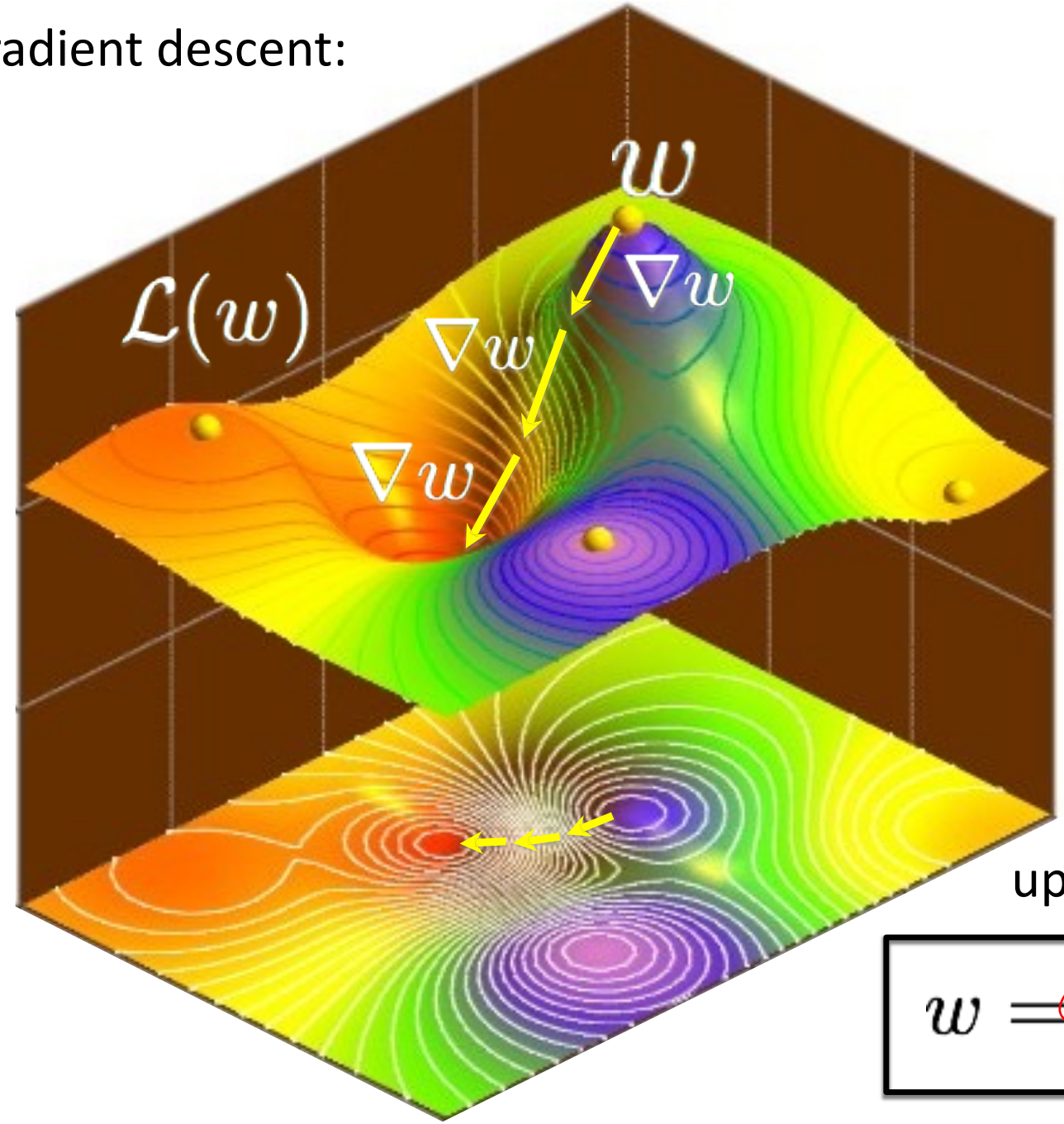
small change in parameter Δw_1 → output will change by $\frac{\partial f(x)}{\partial w_1} \Delta w_1$

Gradient descent:



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

Gradient descent:

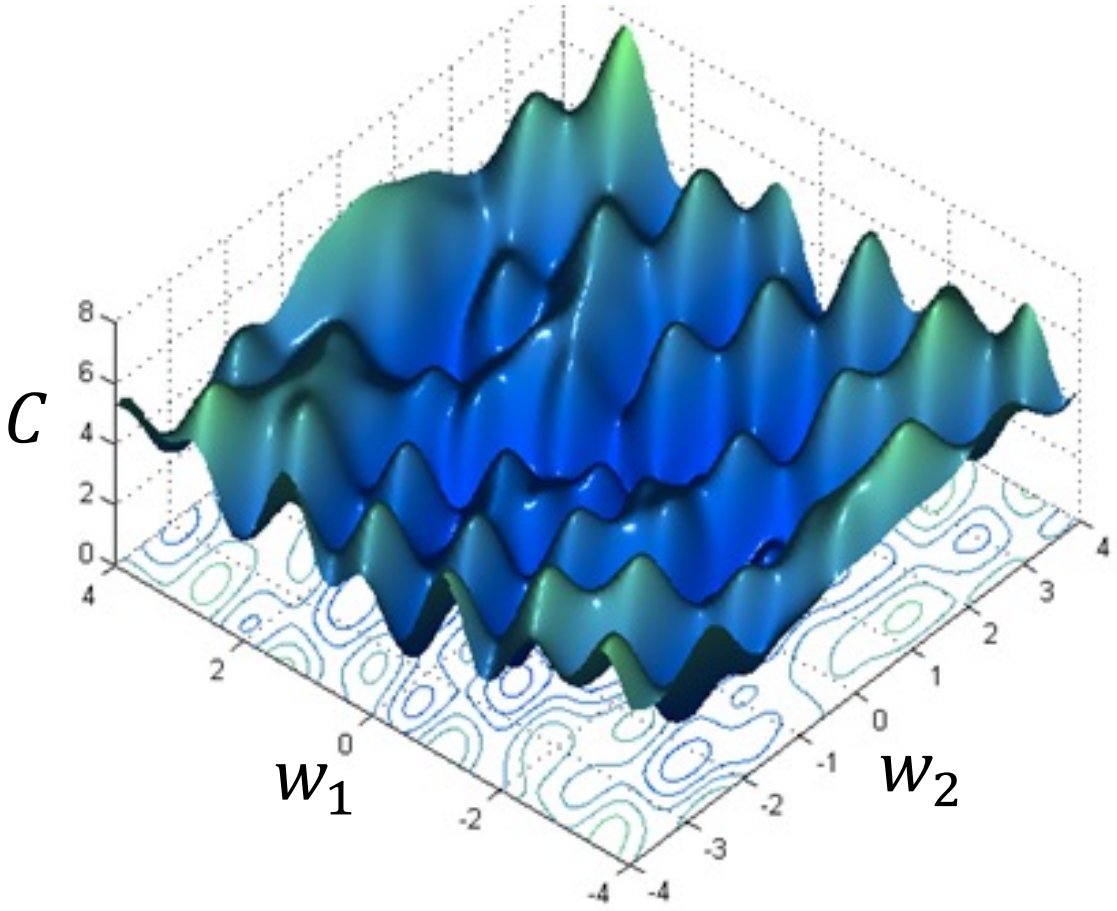


update rule:

$$w = w - \nabla w$$

Local Minima

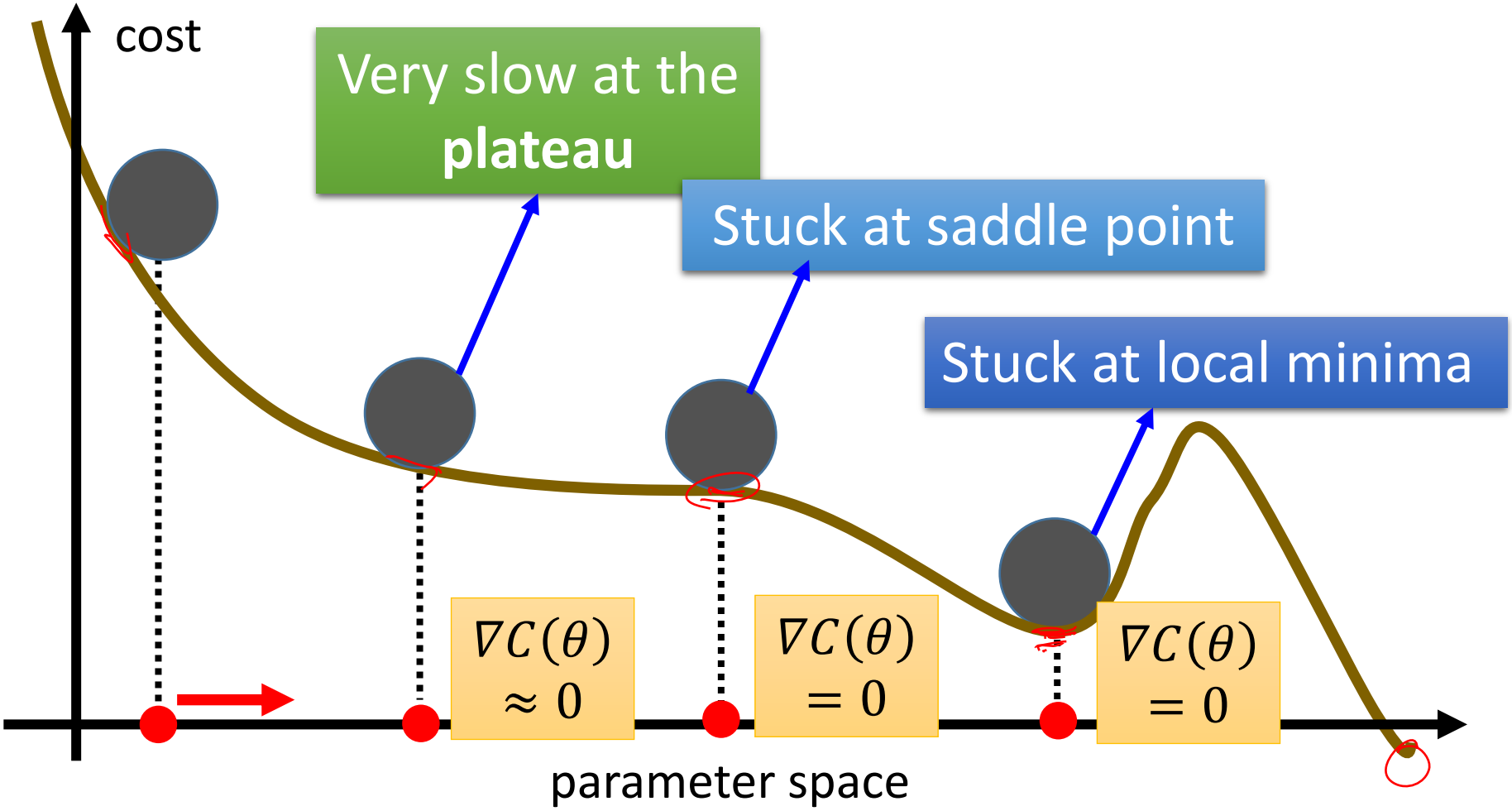
- Gradient descent never guarantee global minima



Different initial point θ^0

Reach different minima, so different results

Besides local minima ……



Backpropagation

Backpropagation

- A network can have millions of parameters.
  - Backpropagation is the way to compute the gradients efficiently
  - ***Ref***: http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/DNN%20backprop.ecm.mp4/index.html
- Many toolkits can compute the gradients automatically

theano

TensorFlow

Ref:

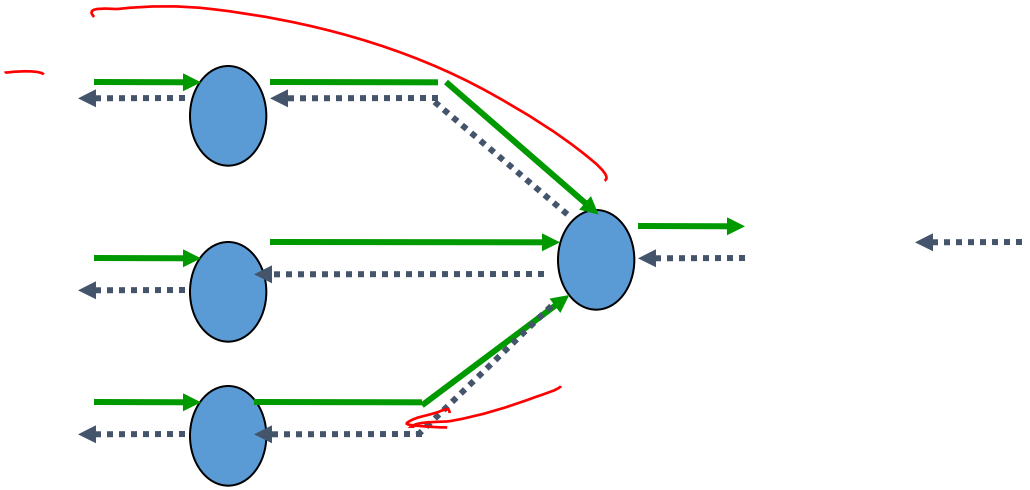
http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/Theano%20DNN.ecm.mp4/index.html

Training: Backprop algorithm

- **The Backprop algorithm searches for weight values that minimize the total error of the network over the set of training examples (training set).**
- **Backprop consists of the repeated application of the following two passes:**
  - **Forward pass**
    The network is activated on one example and the error of (each neuron of) the output layer is computed.
  - **Backward pass**
    The network error is used for updating the weights.
    Starting at the output layer, the error is propagated backwards through the network, layer by layer.
    This is done by recursively computing the local gradient of each neuron.

Back Propagation

- Back-propagation training algorithm



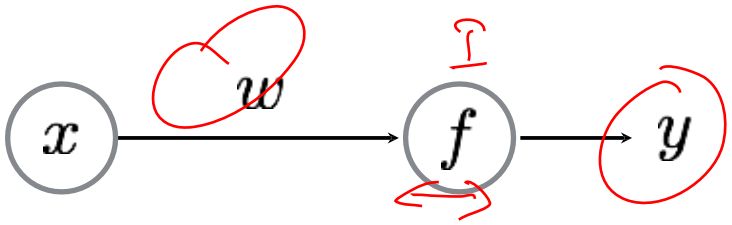
Network activation
Forward Step

Error propagation
Backward Step

- Backprop adjusts the weights of the NN in order to minimize the network total mean squared error.

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

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

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

This is just gradient descent, that means…

this should be the gradient of the loss function

$\frac{d\mathcal{L}}{dw}$ … 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

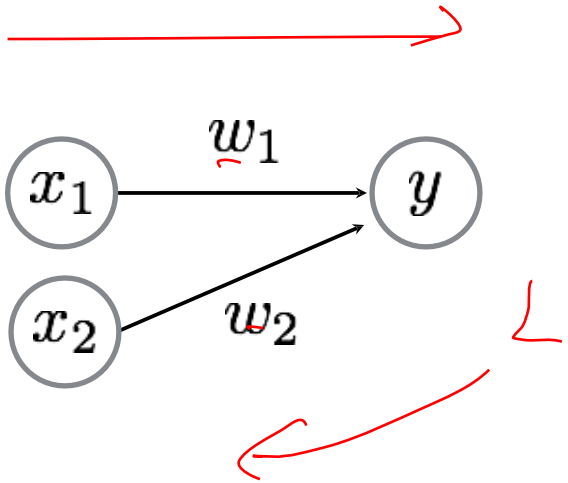
$$= w + (y-\hat{y})x$$

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 $w = w - \nabla w$

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

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

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

Why do we have partial derivatives now?

Derivative computation

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

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

Gradient Update

$$w_1 = w_1 - \eta \nabla w_1$$
$$= w_1 + \eta(y-\hat{y})x_1$$

$$w_2 = w_2 - \eta \nabla w_2$$
$$= w_2 + \eta(y-\hat{y})x_2$$

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
   b. Gradient update

two lines now

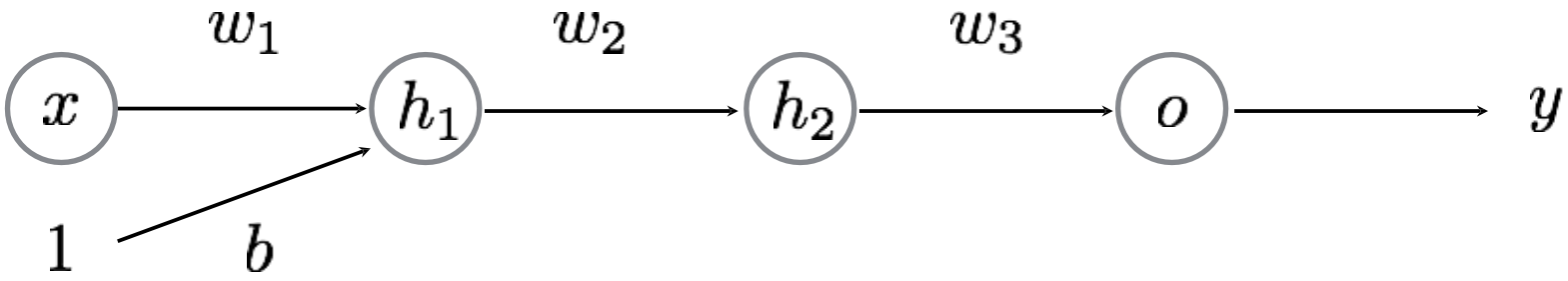
$$\nabla w_{1i} = -(y_i - \hat{y})x_{1i}$$
$$\nabla w_{2i} = -(y_i - \hat{y})x_{2i}$$

$$w_{1i} = w_{1i} + \eta(y - \hat{y})x_{1i}$$
$$w_{2i} = w_{2i} + \eta(y - \hat{y})x_{2i}$$

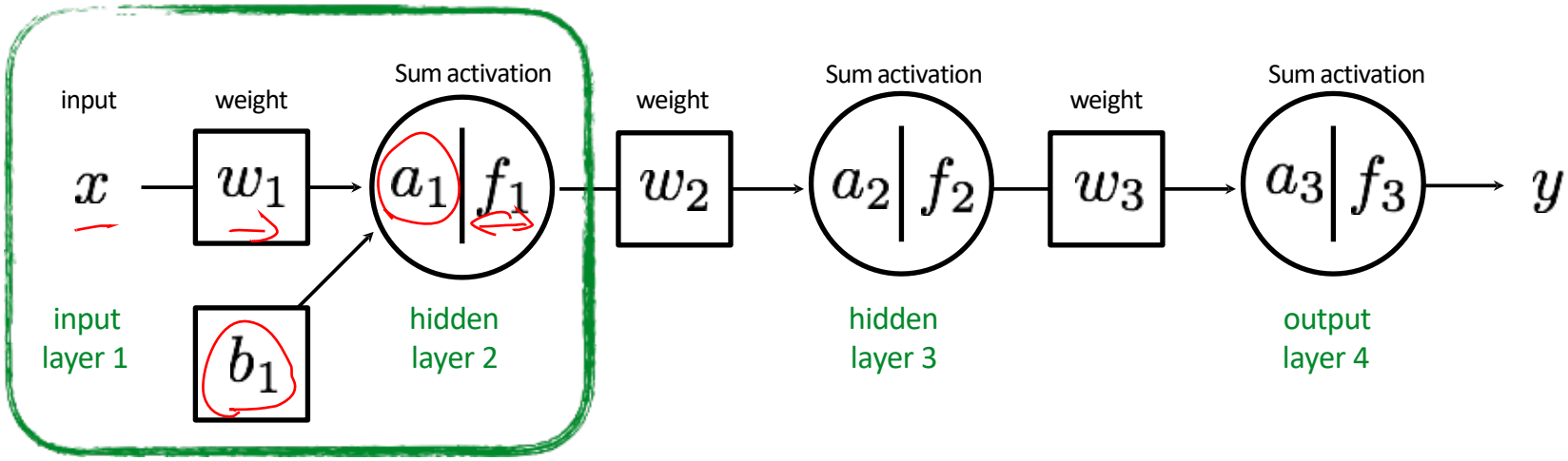
(adjustable step size)

We haven't seen a lot of '**propagation**' yet because our perceptrons only had one layer…

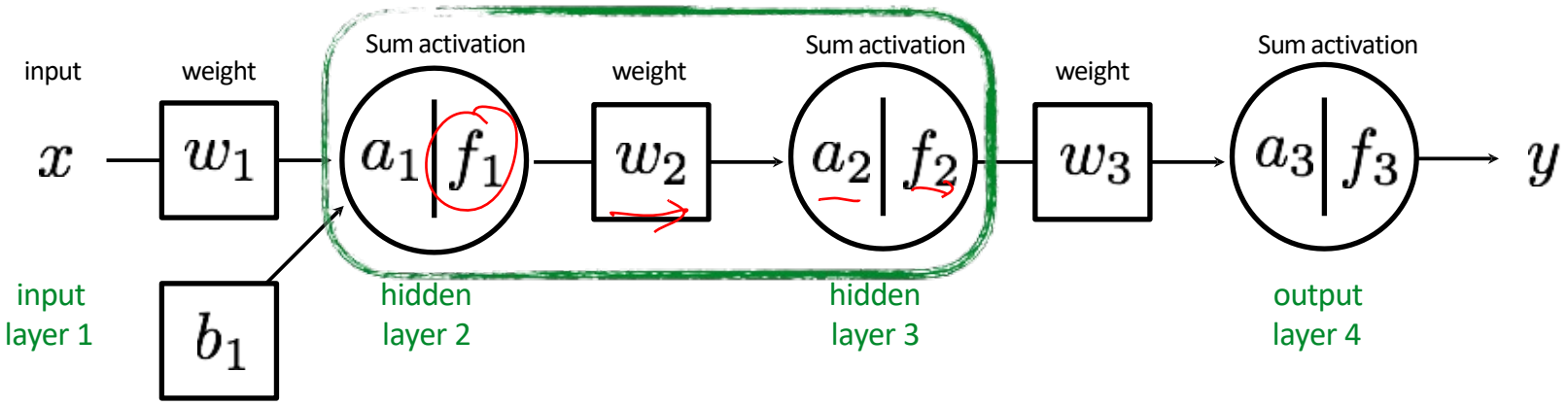
multi-layer perceptron



function of FOUR parameters and FOUR layers!

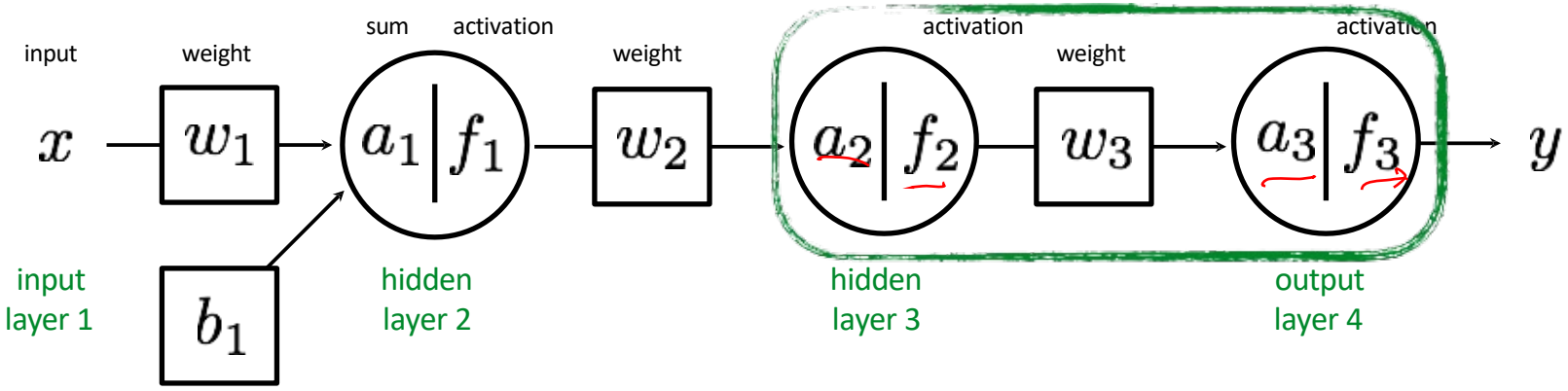


$$a_1 = w_1 \cdot x + b_1$$



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

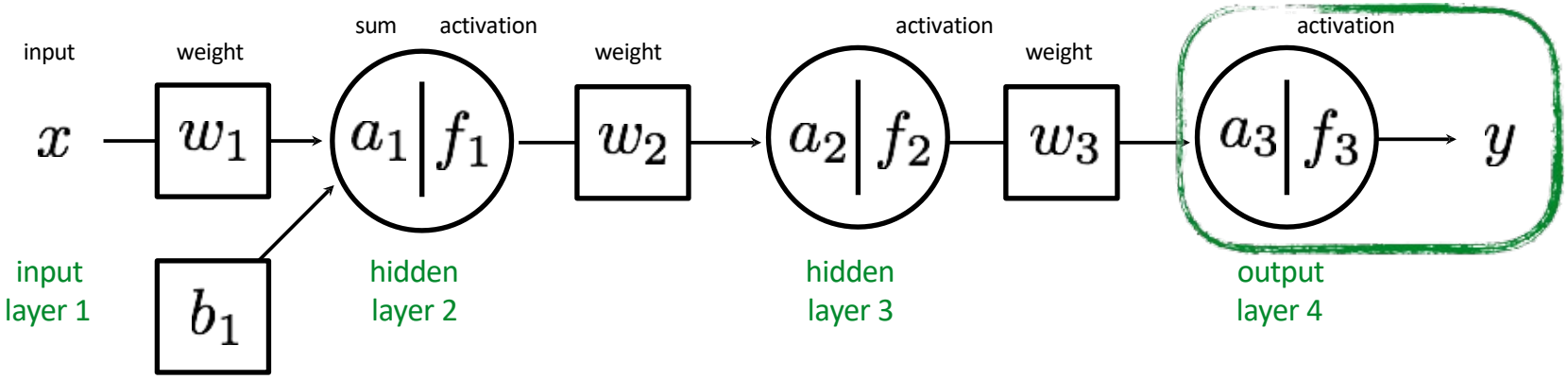
$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$



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

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

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



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

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

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

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

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

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

We need to train the network:

What is known? What is unknown?

Entire network can be written out as a long equation

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

We need to train the network:

What is known? What is unknown?

Learning an MLP

Given a set of samples and an 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 $\{x_i, y_i\}$

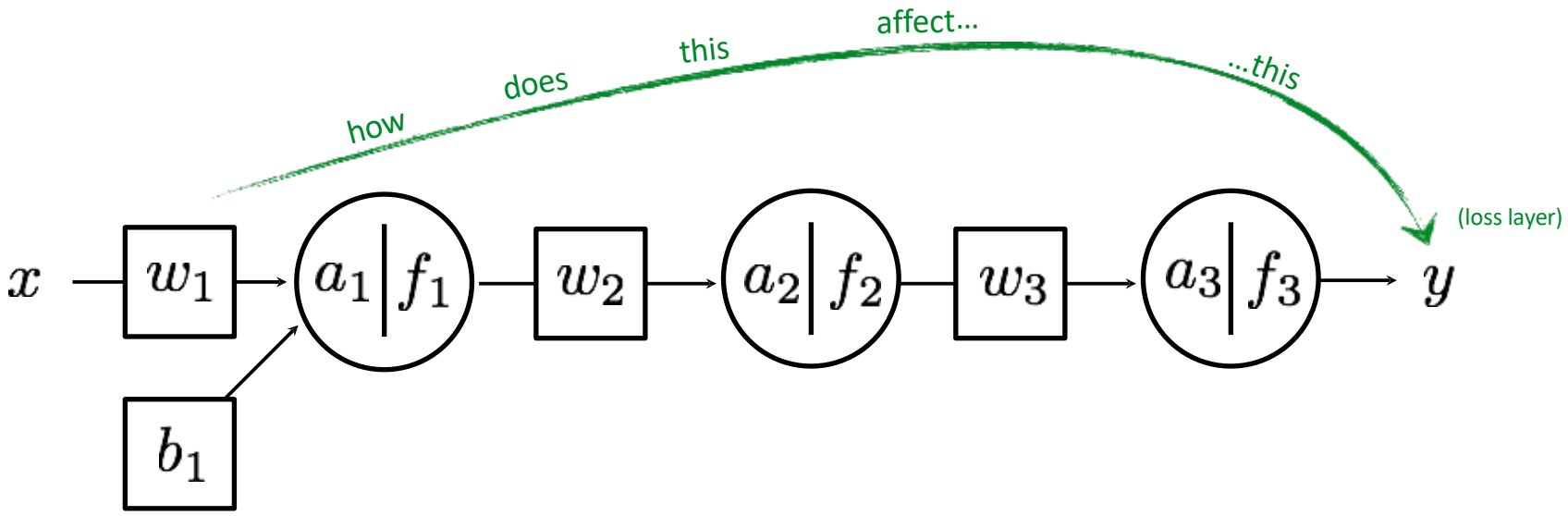
1. Predict
   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$
   b. Compute Loss
2. Update
   a. Back Propagation $\frac{\partial \mathcal{L}}{\partial \theta}$ vector of parameter partial derivatives
   b. Gradient update $\theta \leftarrow \theta - \eta \nabla \theta$

vector of parameter update equations

So we need to compute the partial derivatives

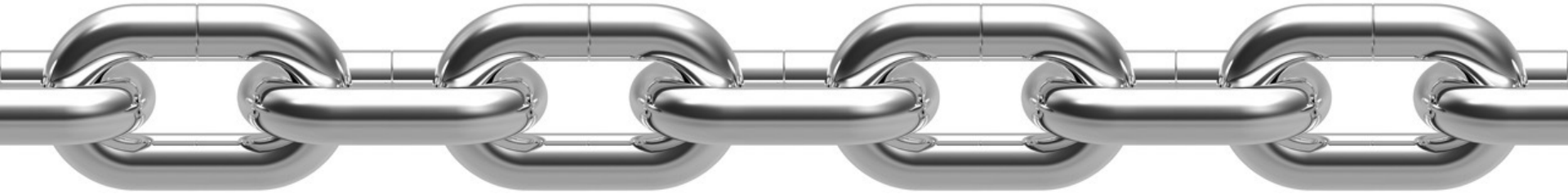
$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \left[\frac{\partial \mathcal{L}}{\partial w_3} \frac{\partial \mathcal{L}}{\partial w_2} \frac{\partial \mathcal{L}}{\partial w_1} \frac{\partial \mathcal{L}}{\partial b} \right]$$

Remember, Partial derivative $\frac{\partial L}{\partial w_1}$ describes…



So, how do you compute it?

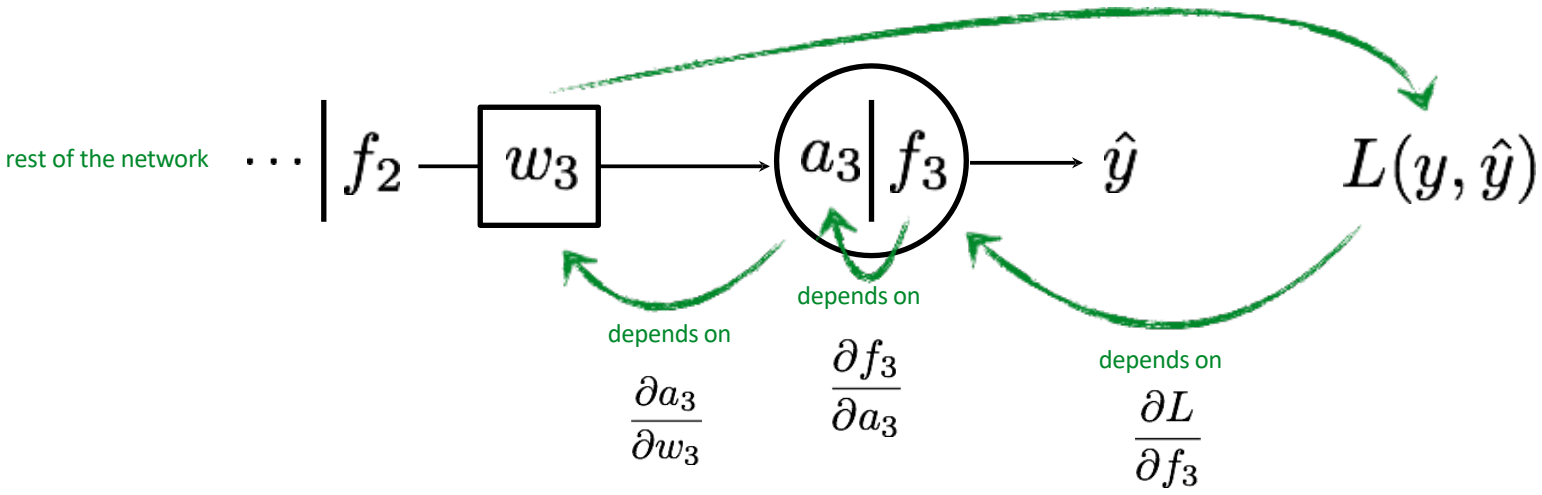
The Chain Rule



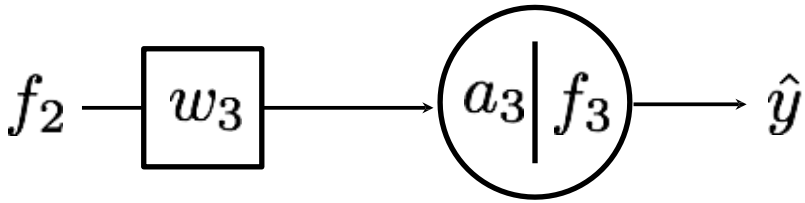
According to the chain rule…

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

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



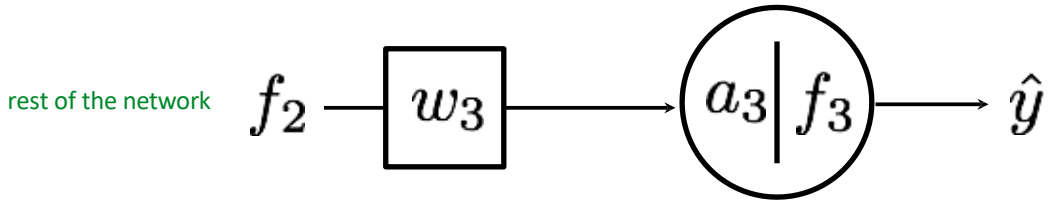
rest of the network



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

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

Chain Rule!

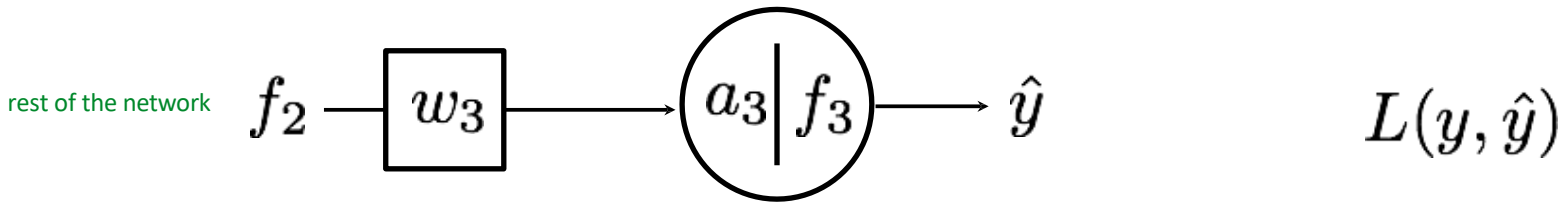


$$L(y, \hat{y})$$

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

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

Just the partial derivative of L2 loss

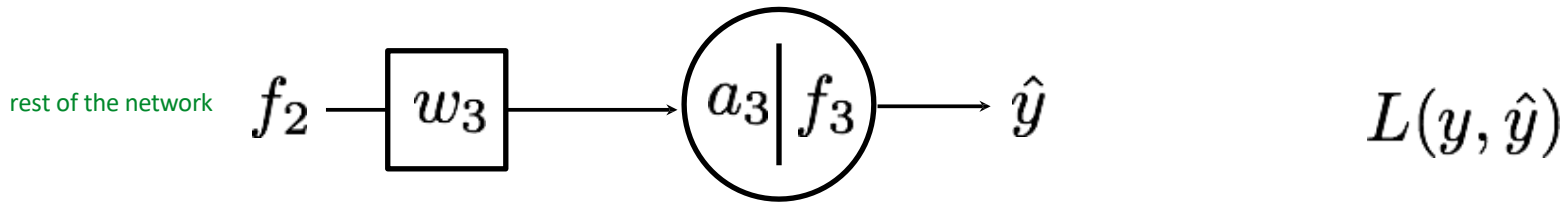


$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

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

Let's use a Sigmoid function

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



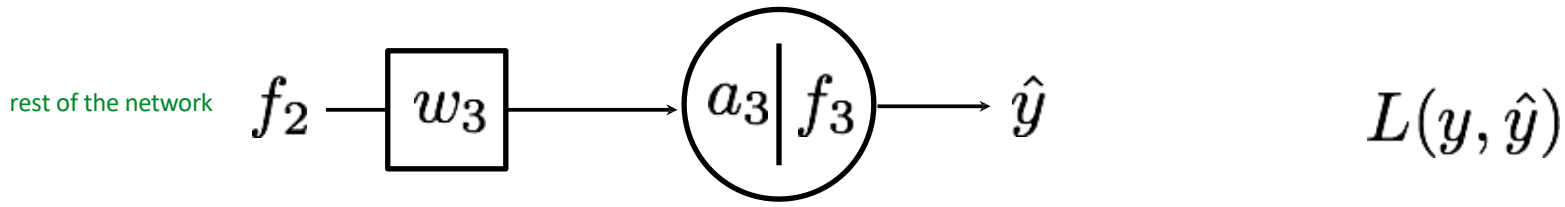
$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

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

$$= -\eta(y - \hat{y}) f_3(1 - f_3)\frac{\partial a_3}{\partial w_3}$$

Let's use a Sigmoid function

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

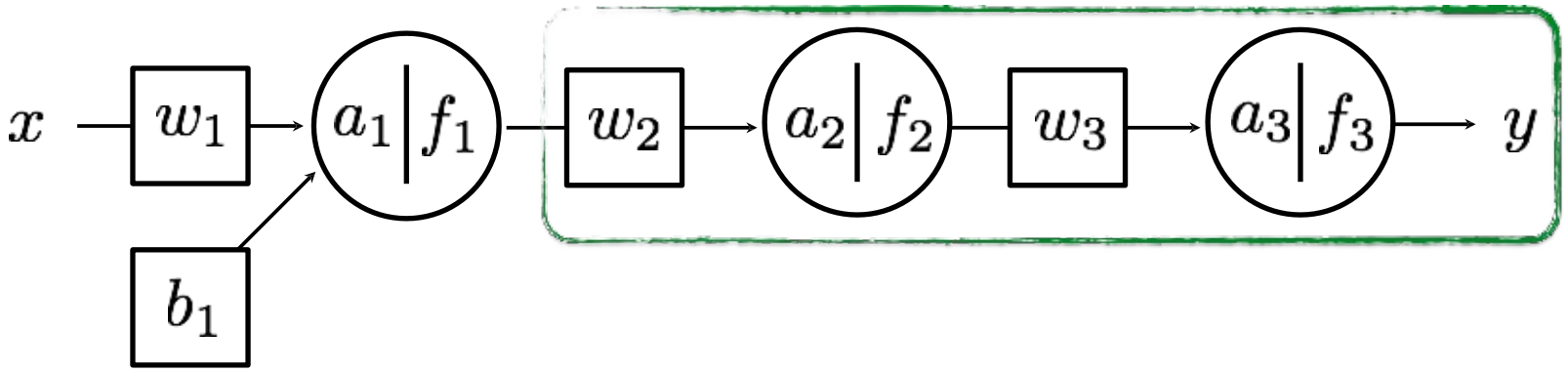


$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

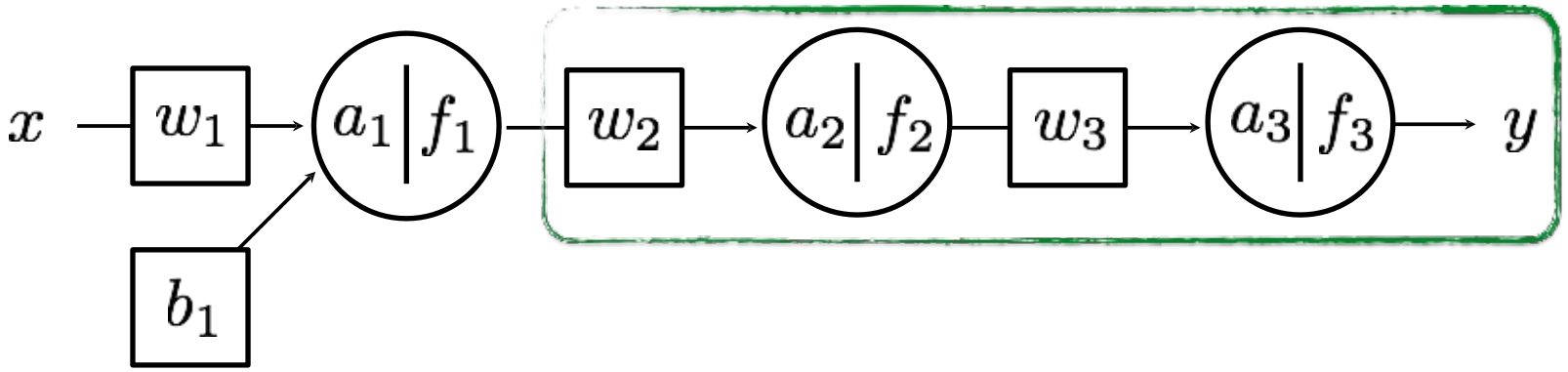
$$= -\eta(y - \hat{y})\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

$$= -\eta(y - \hat{y}) f_3(1 - f_3)\frac{\partial a_3}{\partial w_3}$$

$$= -\eta(y - \hat{y}) f_3(1 - f_3) f_2$$



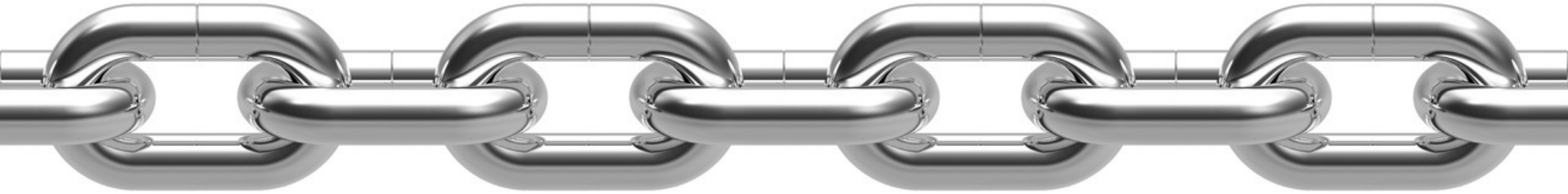
$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial w_2}$$



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

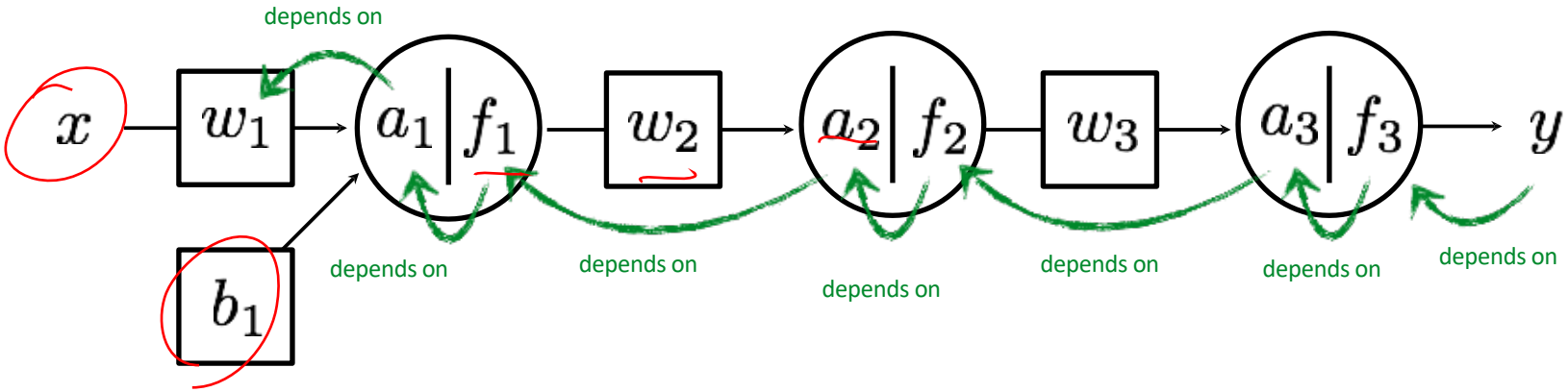
already computed.
re-use (propagate)!

The Chain Rule



a.k.a. backpropagation

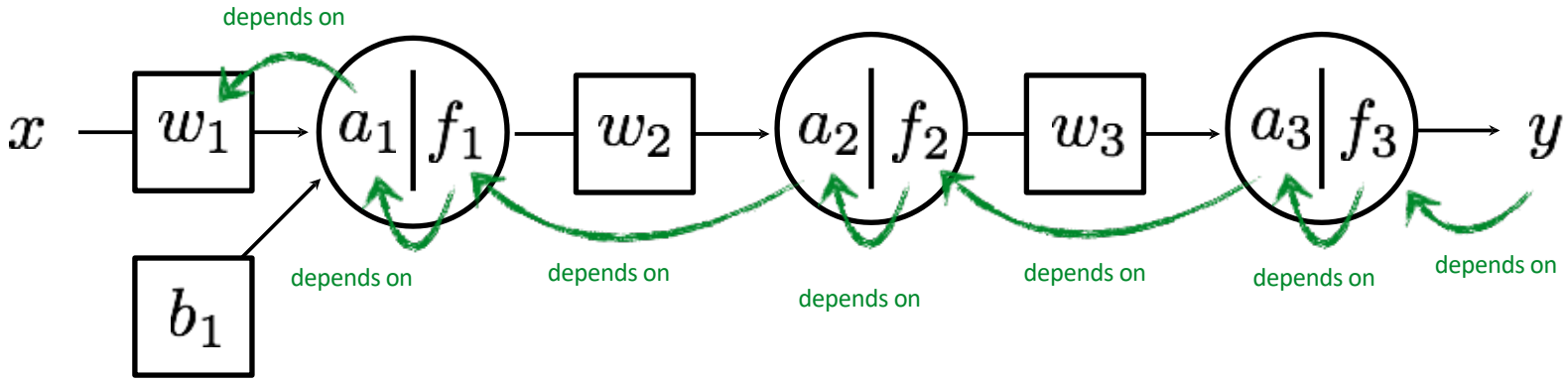
The chain rule says…



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

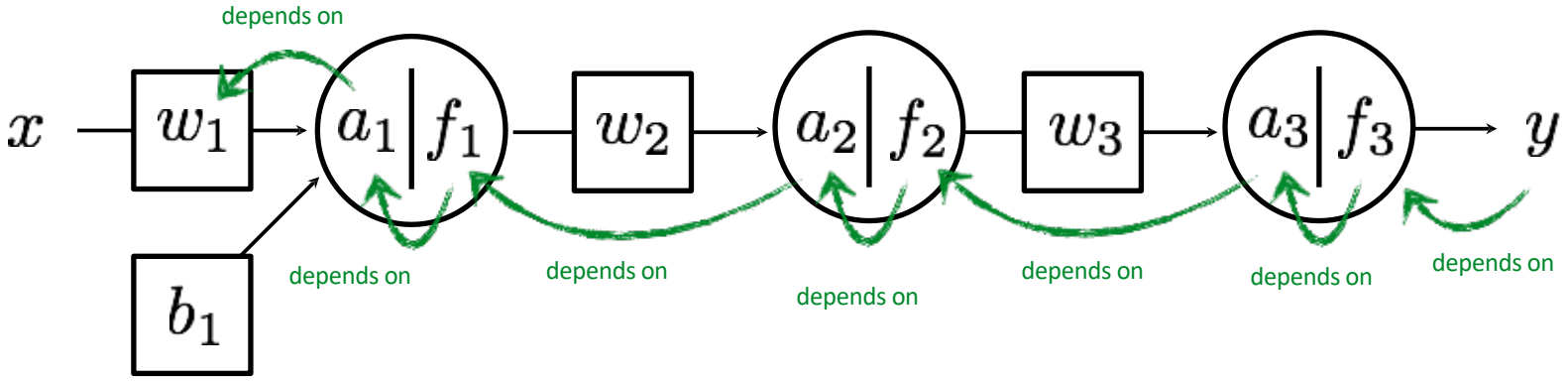
W_2

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)!

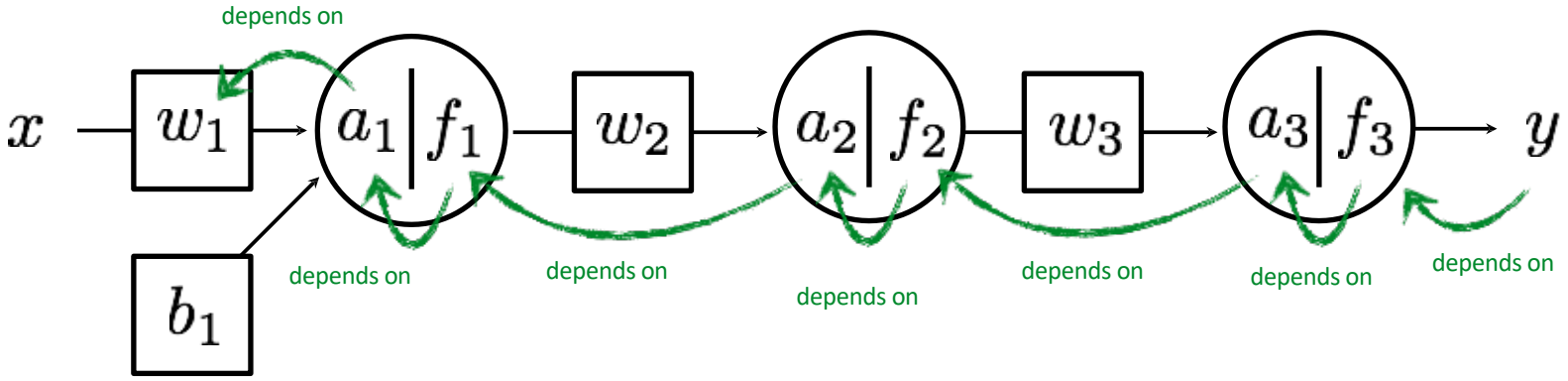


$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

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

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

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

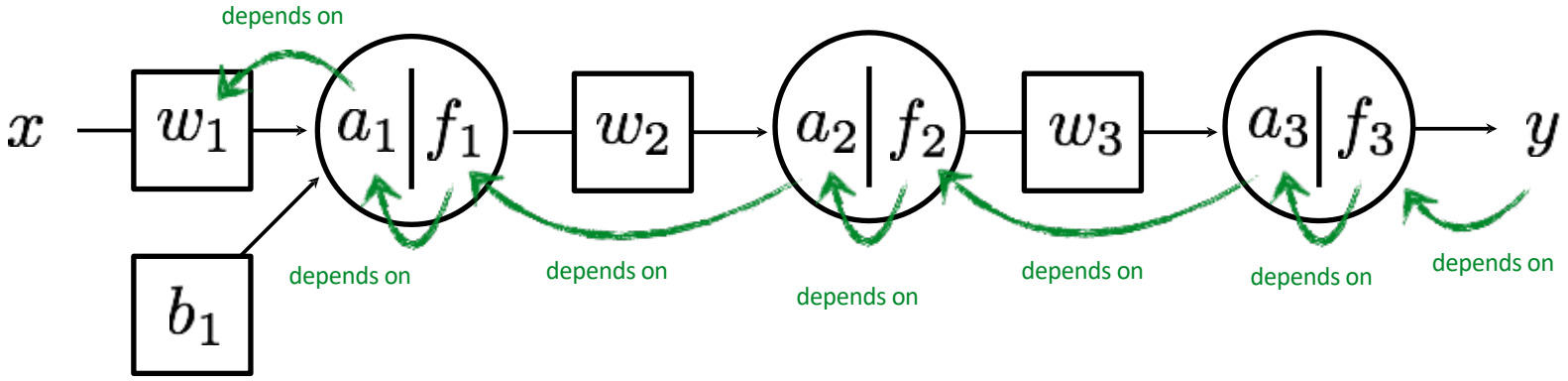


$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

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

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

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



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

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

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

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

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

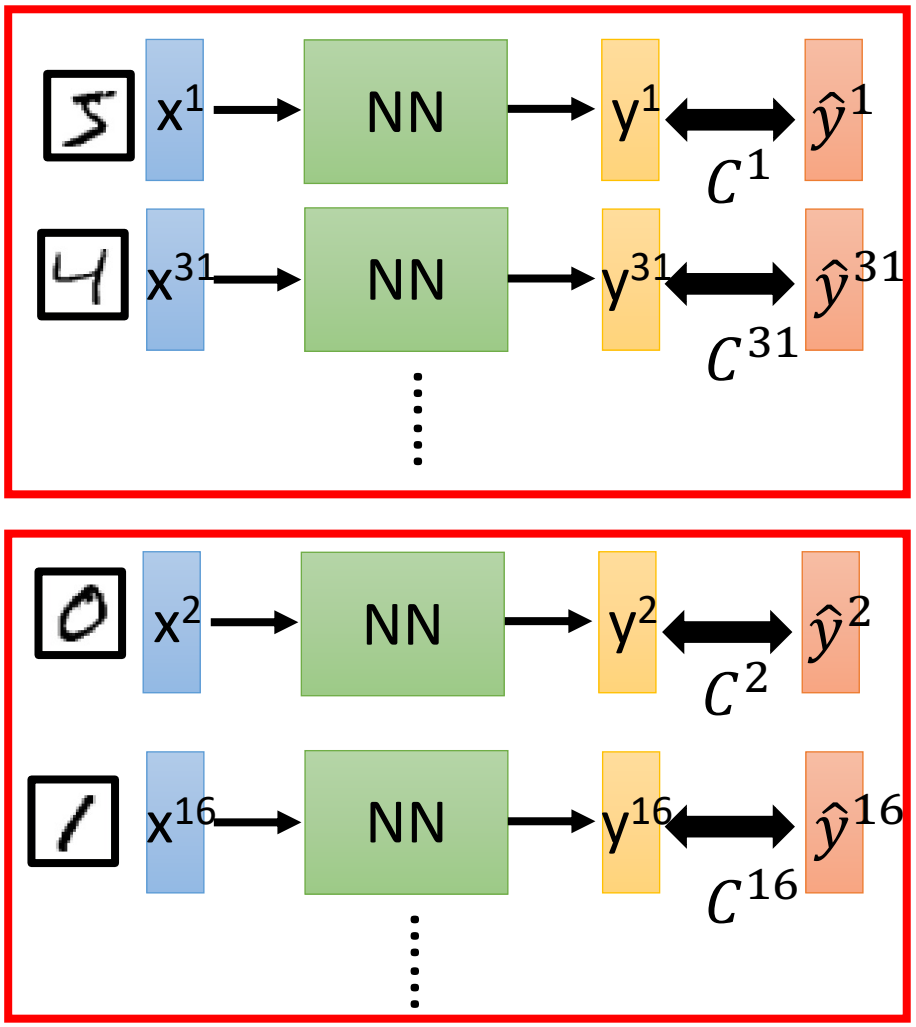
How do we select which sample?

How do we select which sample?

- Select randomly!

Do we need to use only one sample?

Mini-batch



- Randomly initialize θ^0
- Pick the 1st batch

$$C = C^1 + C^{31} + \cdots$$

$$\theta^1 \leftarrow \theta^0 - \eta \nabla C(\theta^0)$$

- Pick the 2nd batch

$$C = C^2 + C^{16} + \cdots$$

$$\theta^2 \leftarrow \theta^1 - \eta \nabla C(\theta^1)$$

$\vdots$

- Until all mini-batches have been picked

one epoch

Repeat the above process

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?

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?

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.

Why Deep?

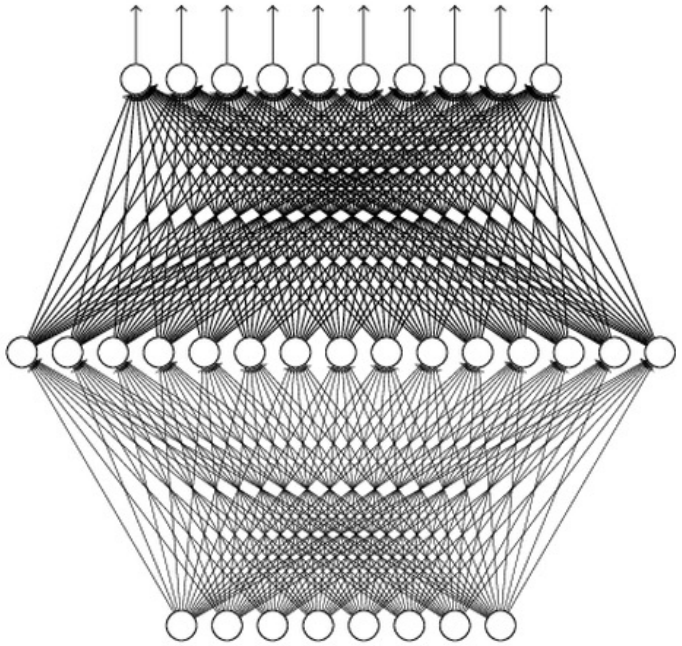
Universality Theorem

Any continuous function f

$$f: R^N \rightarrow R^M$$

Can be realized by a network with one hidden layer

(given **enough** hidden neurons)



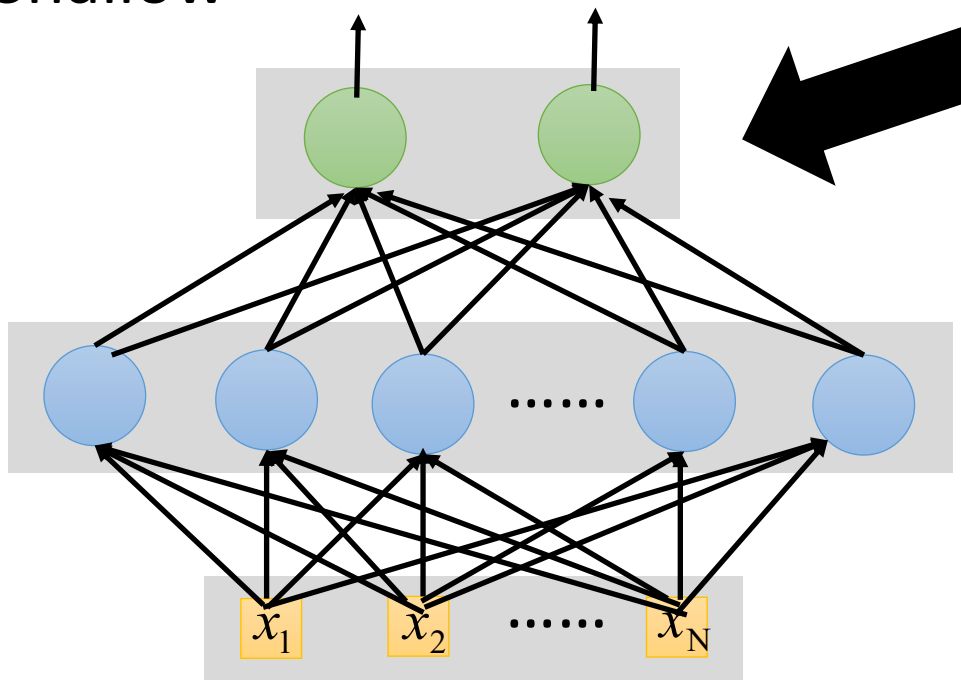
Reference for the reason:
http://neuralnetworksanddeplearning.com/chap4.html

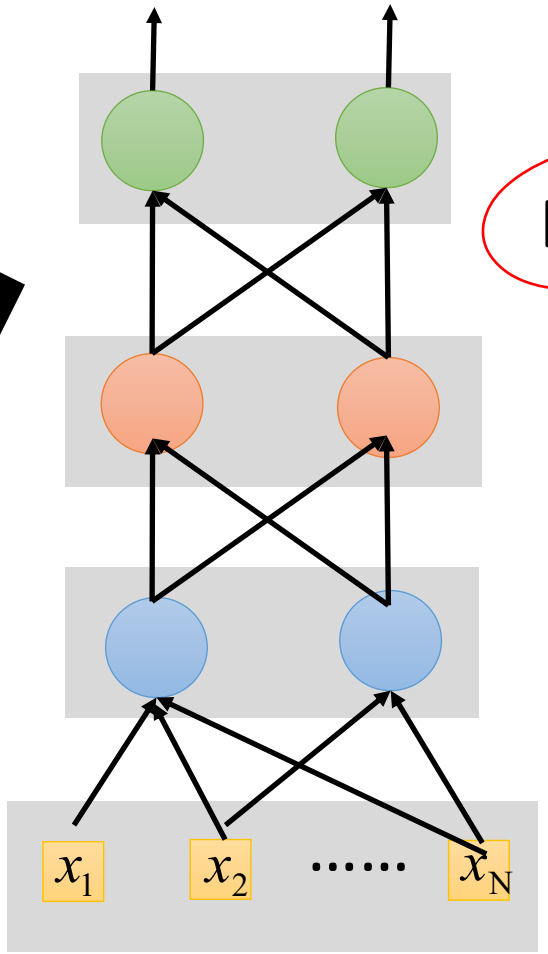
Why “Deep” neural network not “Fat” neural network?

Fat + Short v.s. Thin + Tall

The same number of parameters

Shallow



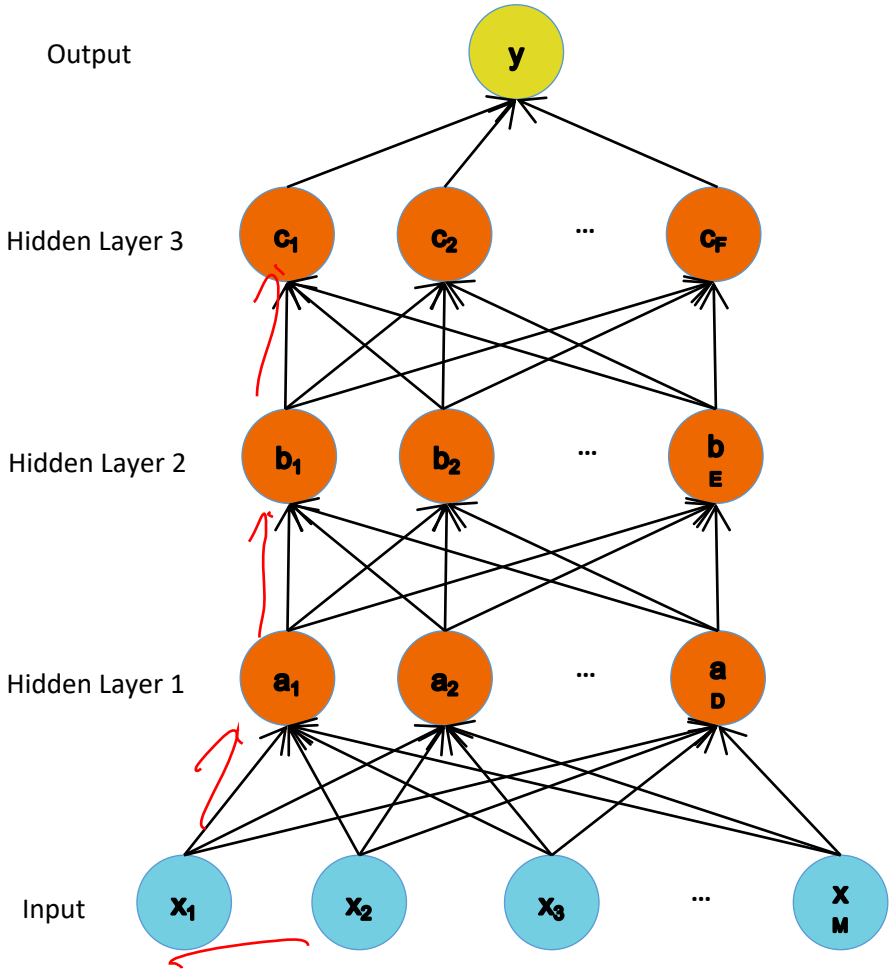


Deep

Which one is better?

Decision Functions

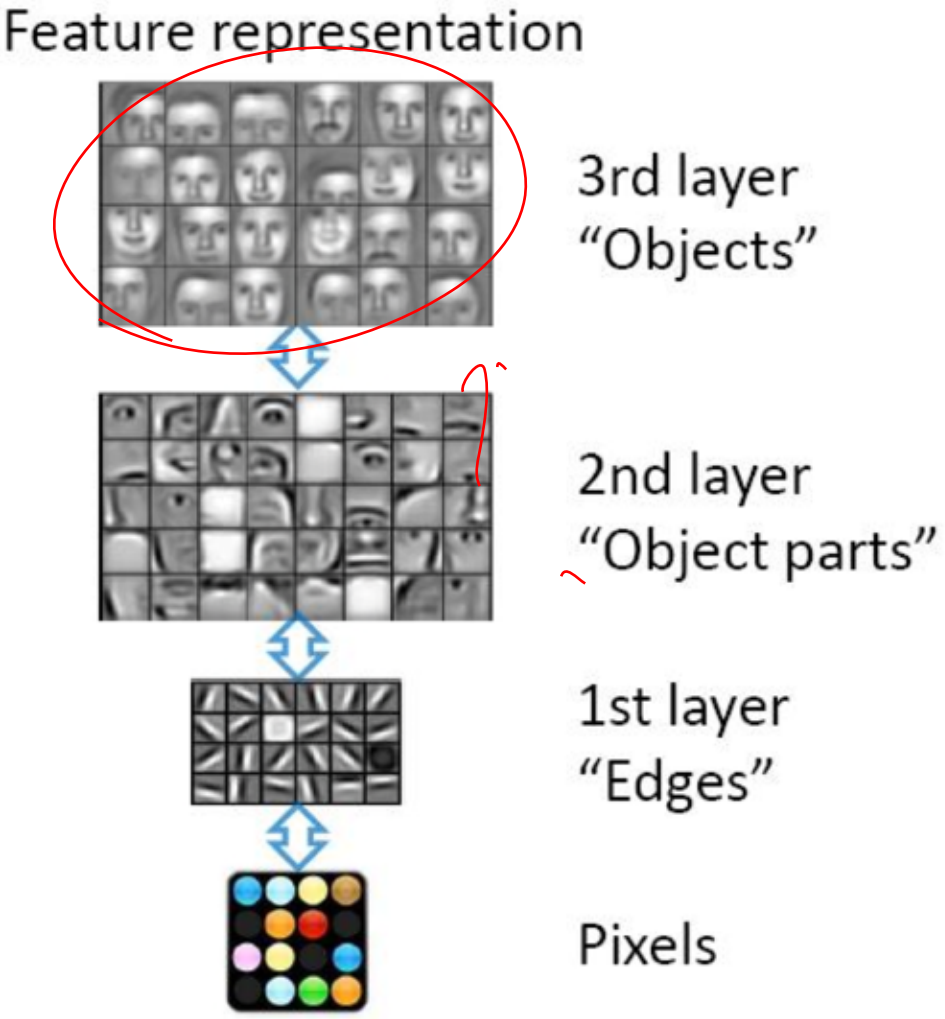
Deeper Networks



Different Levels of Abstraction

Face Recognition:

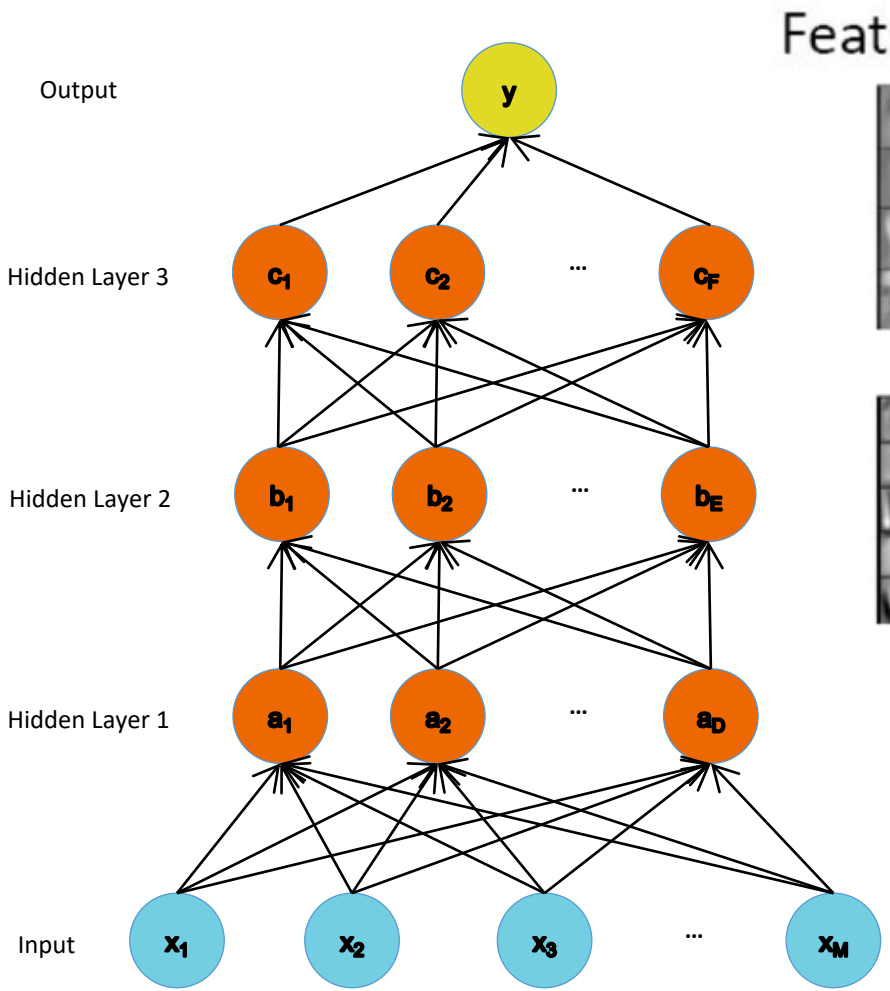
- Deep Network can build up increasingly higher levels of abstraction
- Lines, parts, regions

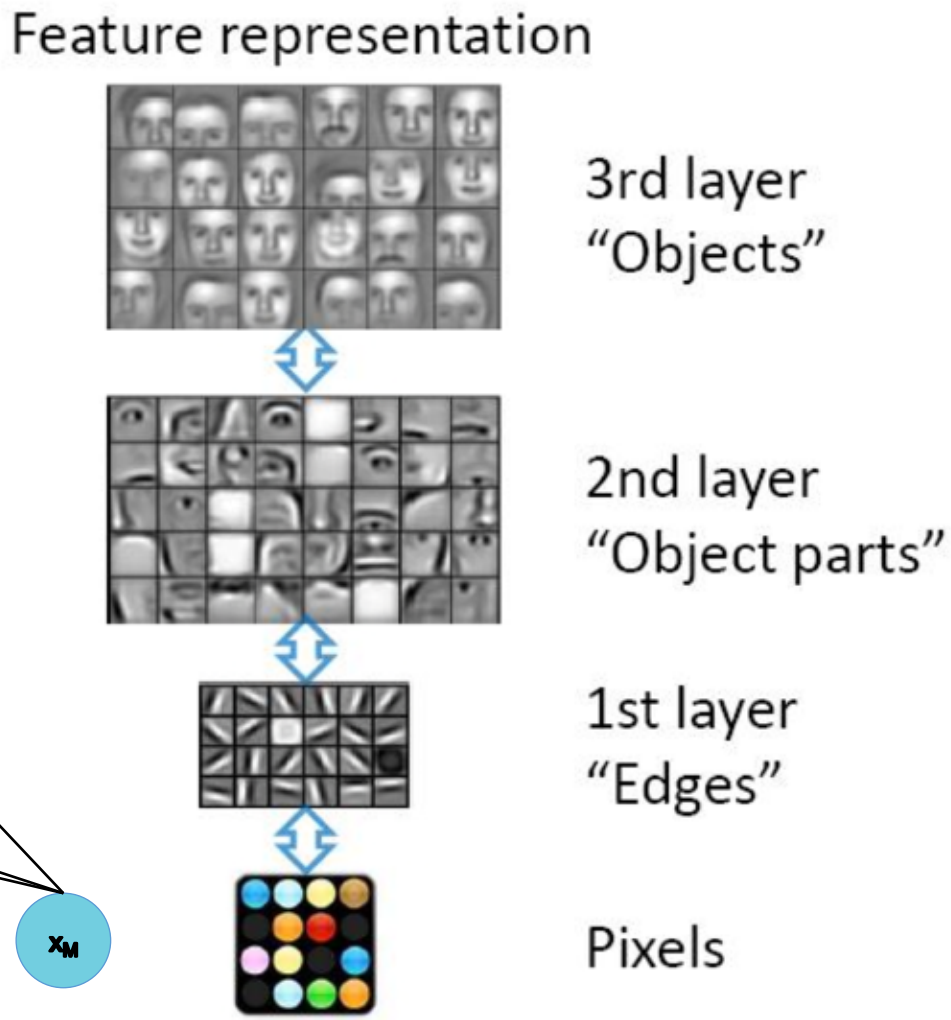


Example from Honglak Lee (NIPS 2010)

Decision Functions

Different Levels of Abstraction





Example from Honglak Lee (NIPS 2010)