# CS540 Introduction to Artificial Intelligence

## (Deep) Neural Networks Summary

Yingyu Liang
University of Wisconsin-Madison

**Nov 9, 2021**

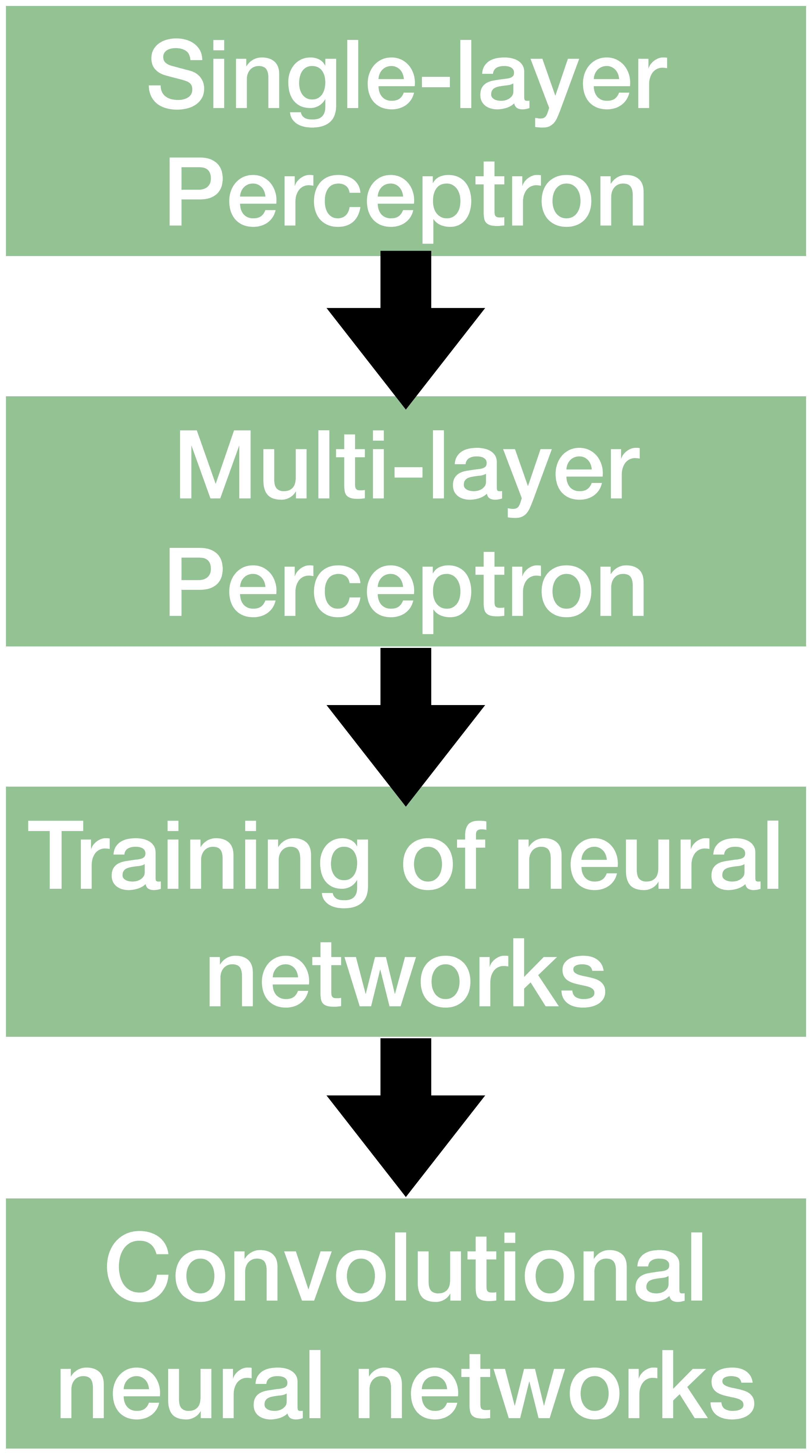
Slides created by Sharon Li [modified by Yingyu Liang]

# How to classify

**Cats vs. dogs?**

# How to classify

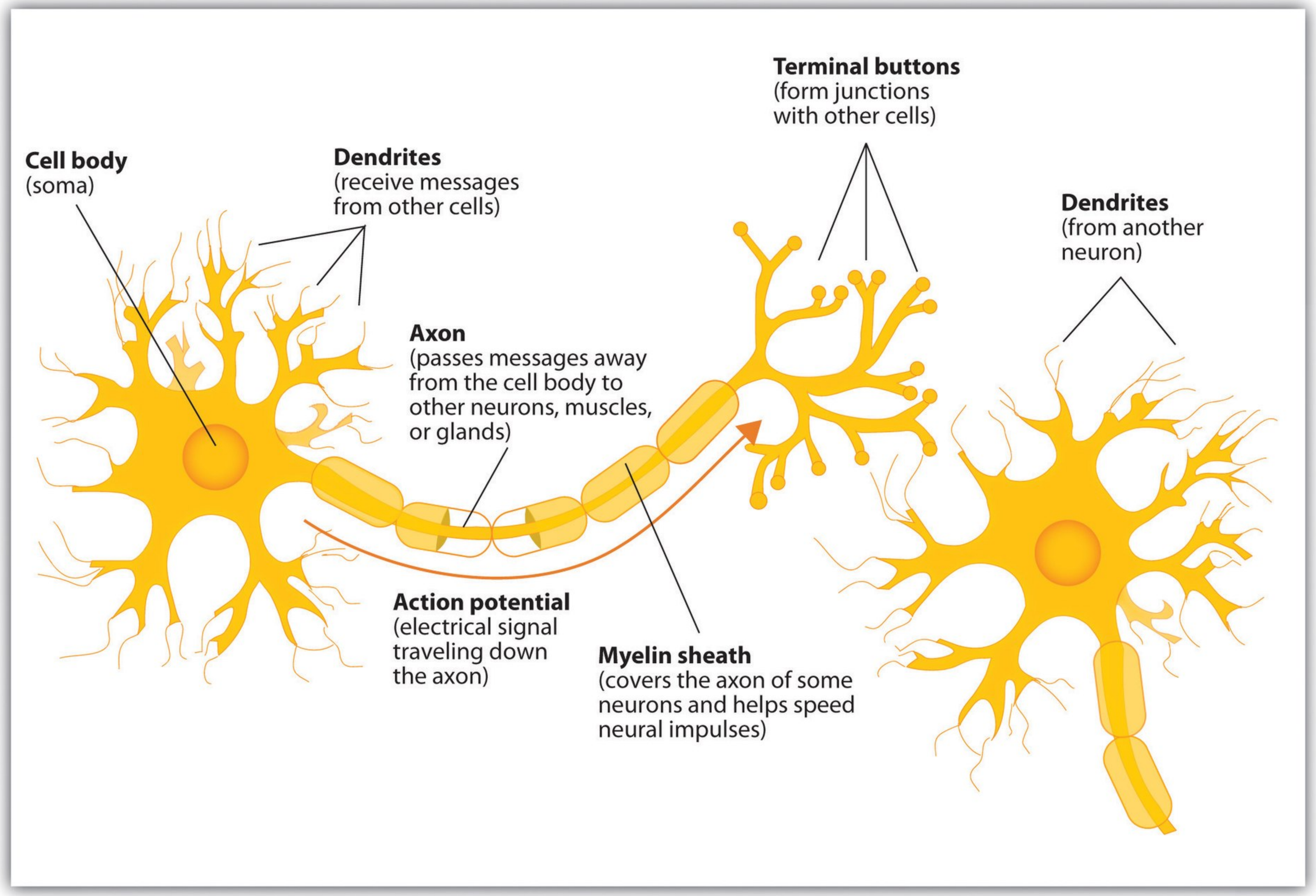
**Cats vs. dogs?**



# Inspiration from neuroscience

- Inspirations from human brains
- Networks of simple and homogenous units (a.k.a **neuron**)

(wikipedia)

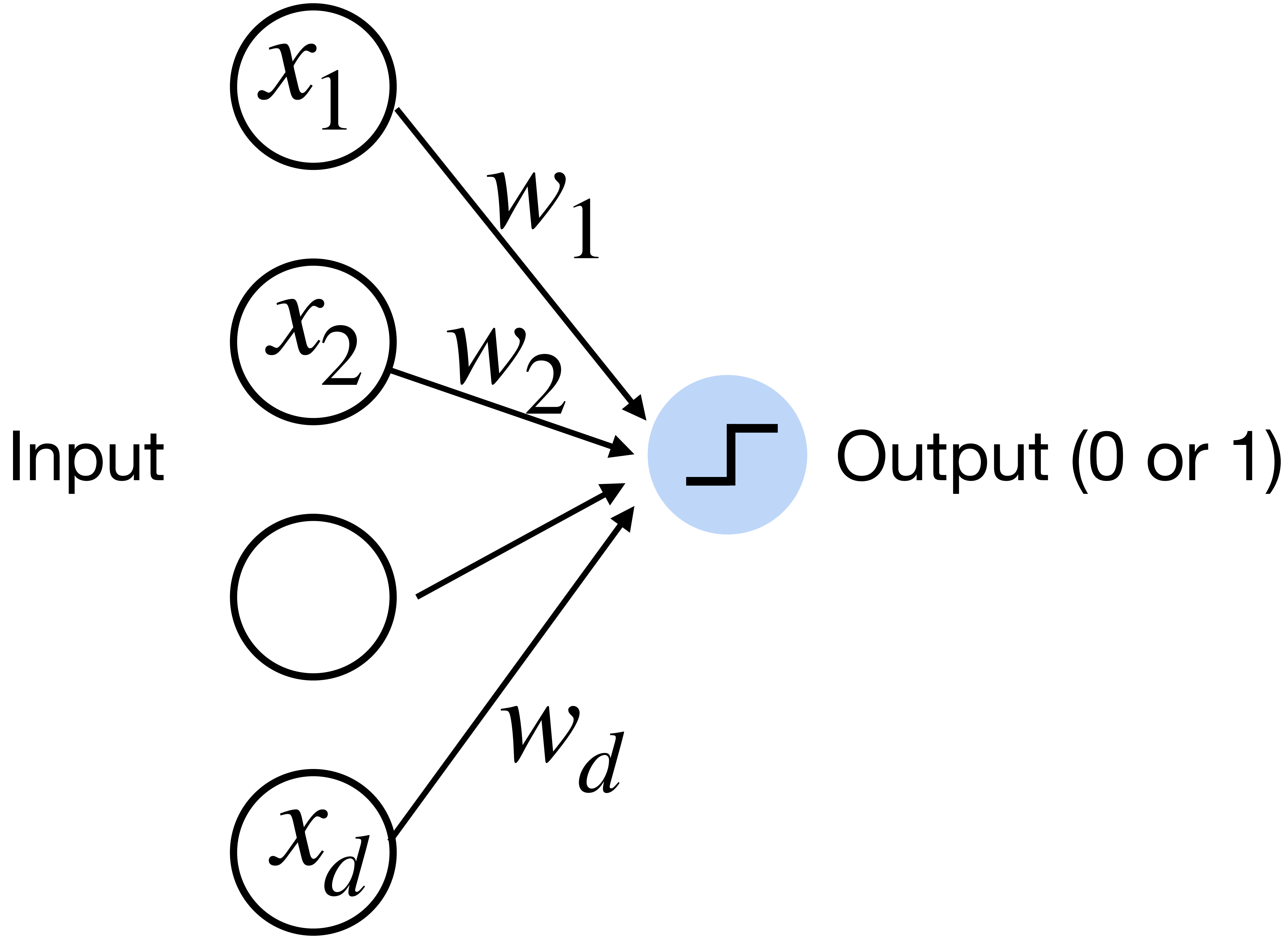


# Perceptron

- Given input $\mathbf{x}$, weight $\mathbf{w}$ and bias $b$, perceptron outputs:

$$o = \sigma\left(\mathbf{w}^\top \mathbf{x} + b\right) \qquad \sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

**Cats vs. dogs?**



# Perceptron

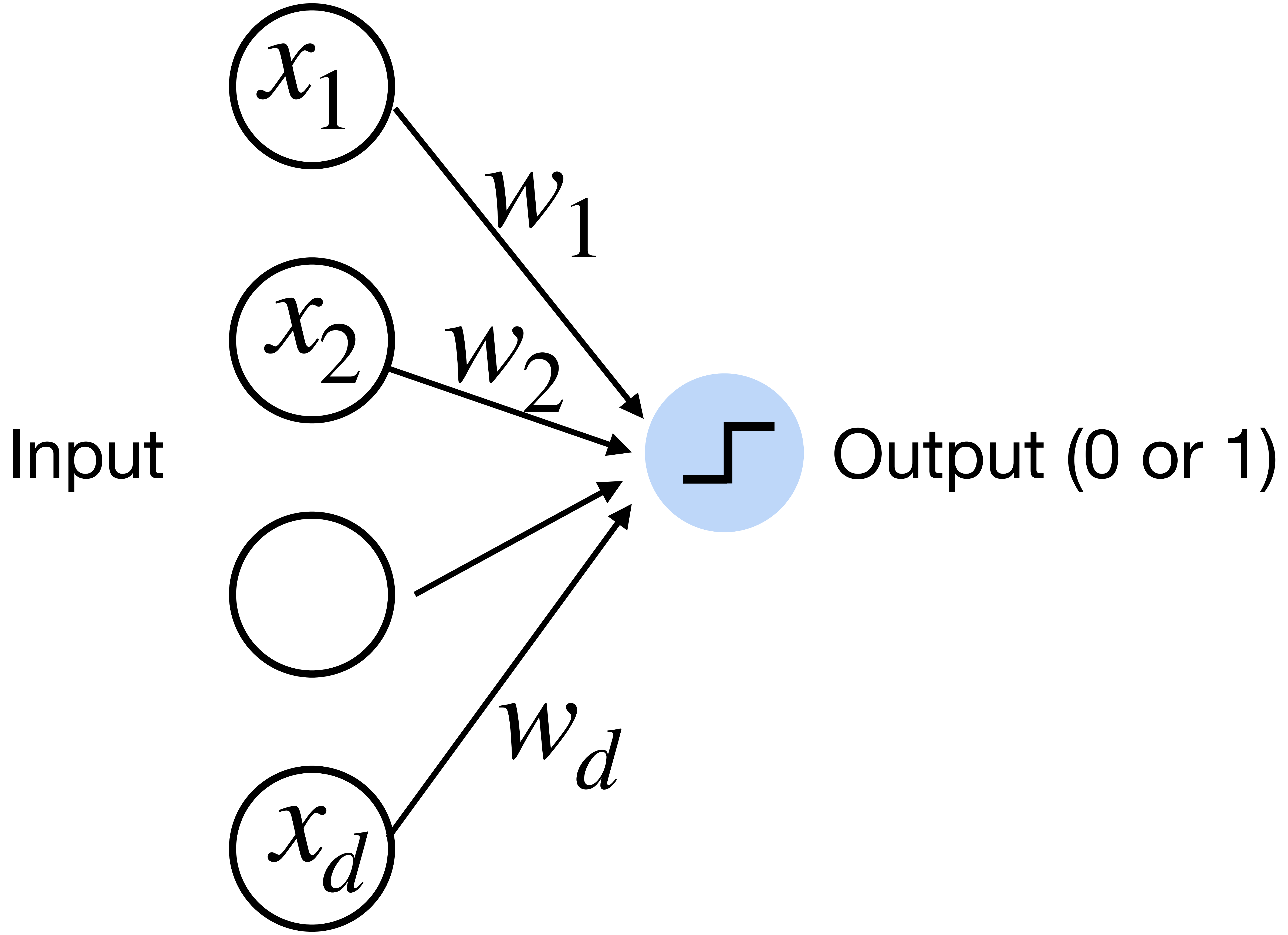
- Given input $\mathbf{x}$, weight $\mathbf{w}$ and bias $b$, perceptron outputs:

$$o = \sigma\left(\mathbf{w}^\top \mathbf{x} + b\right)$$

$$\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

**Activation function**

**Cats vs. dogs?**



# Training the Perceptron

**Perceptron Algorithm**

```
Initialize w⃗ = 0⃗                    // Initialize w⃗. w⃗ = 0⃗ misclassifies everything.
while TRUE do                        // Keep looping
    m = 0                            // Count the number of misclassifications, m
    for (x_i, y_i) ∈ D do            // Loop over each (data, label) pair in the dataset, D
        if o_i ≠ y_i then            // If the pair (x⃗_i, y_i) is misclassified
            w⃗ ← w⃗ + x_i if y_i = 1,   w⃗ ← w⃗ − x_i if y_i = 0
            m ← m + 1                // Counter the number of misclassification
        end if
    end for
    if m = 0 then                    // If the most recent w⃗ gave 0 misclassifications
        break                        // Break out of the while-loop
    end if
end while                            // Otherwise, keep looping!
```

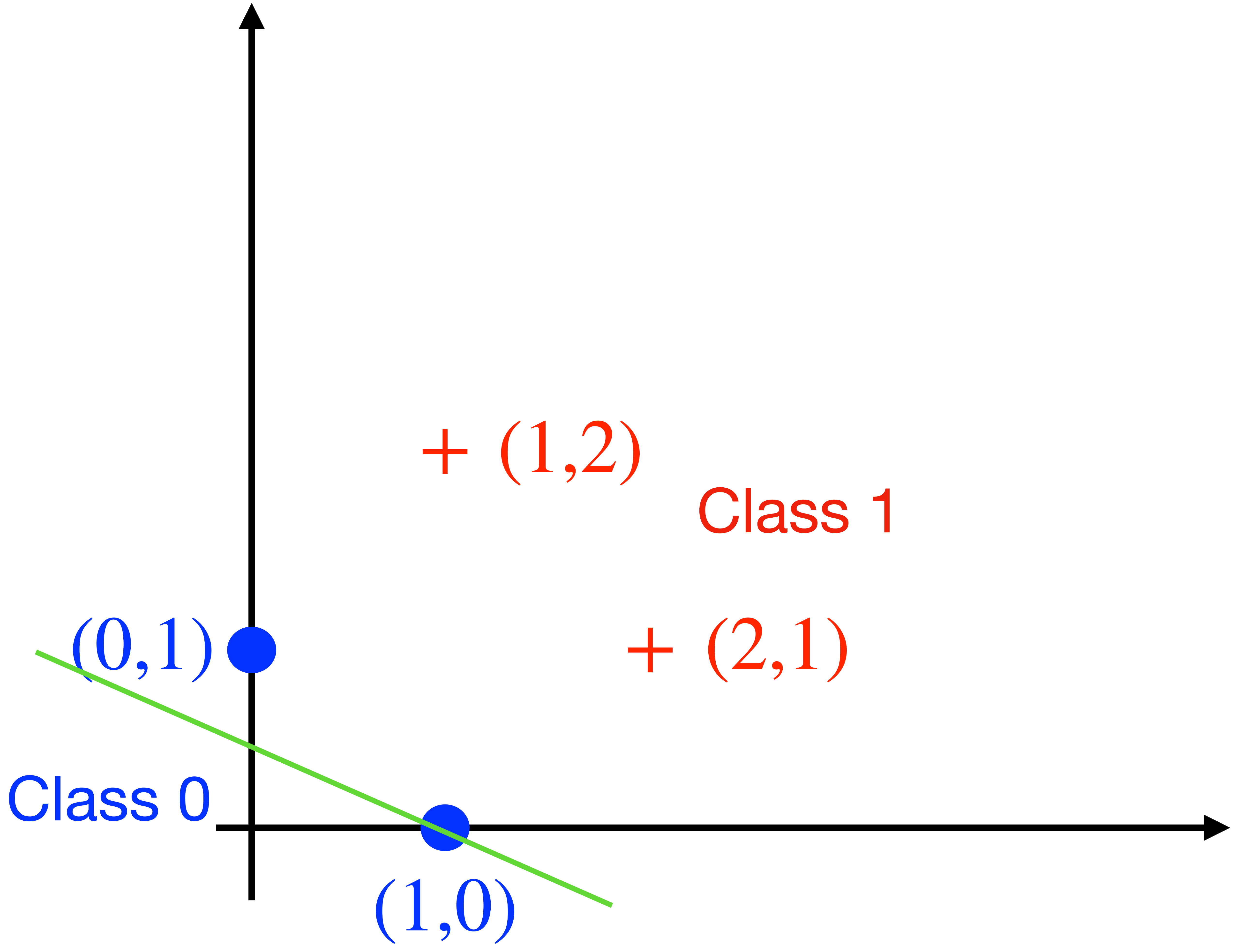
For simplicity, the weight vector and input vector are extended vectors (including the bias or the constant 1).

# Example: Training the Perceptron

- Suppose we begin with:

  $\mathbf{w} = (1,2), b = -1$

- Extended vectors:



# Example: Training the Perceptron

- Suppose we begin with:

  $\mathbf{w} = (1,2), b = -1$

- Extended vectors:

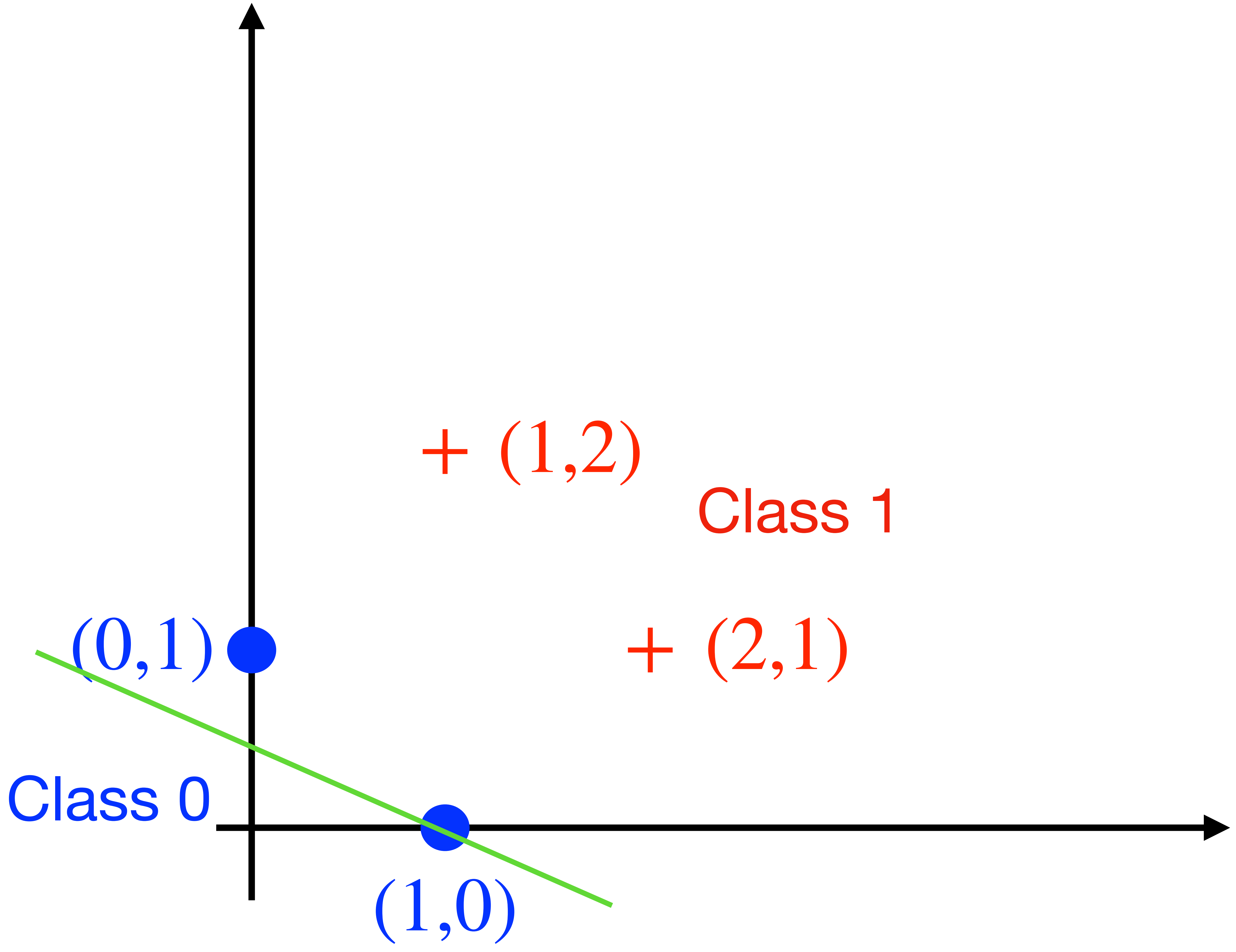
  $\vec{w} = (1,2,-1)$

  $\vec{x_1} = (0,1,1)$

  $\vec{x_2} = (1,0,1)$

  $\vec{x_3} = (2,1,1)$

  $\vec{x_4} = (1,2,1)$



# Example: Training the Perceptron

$\vec{w} = (1,2,-1)$

$\vec{x_1} = (0,1,1)$

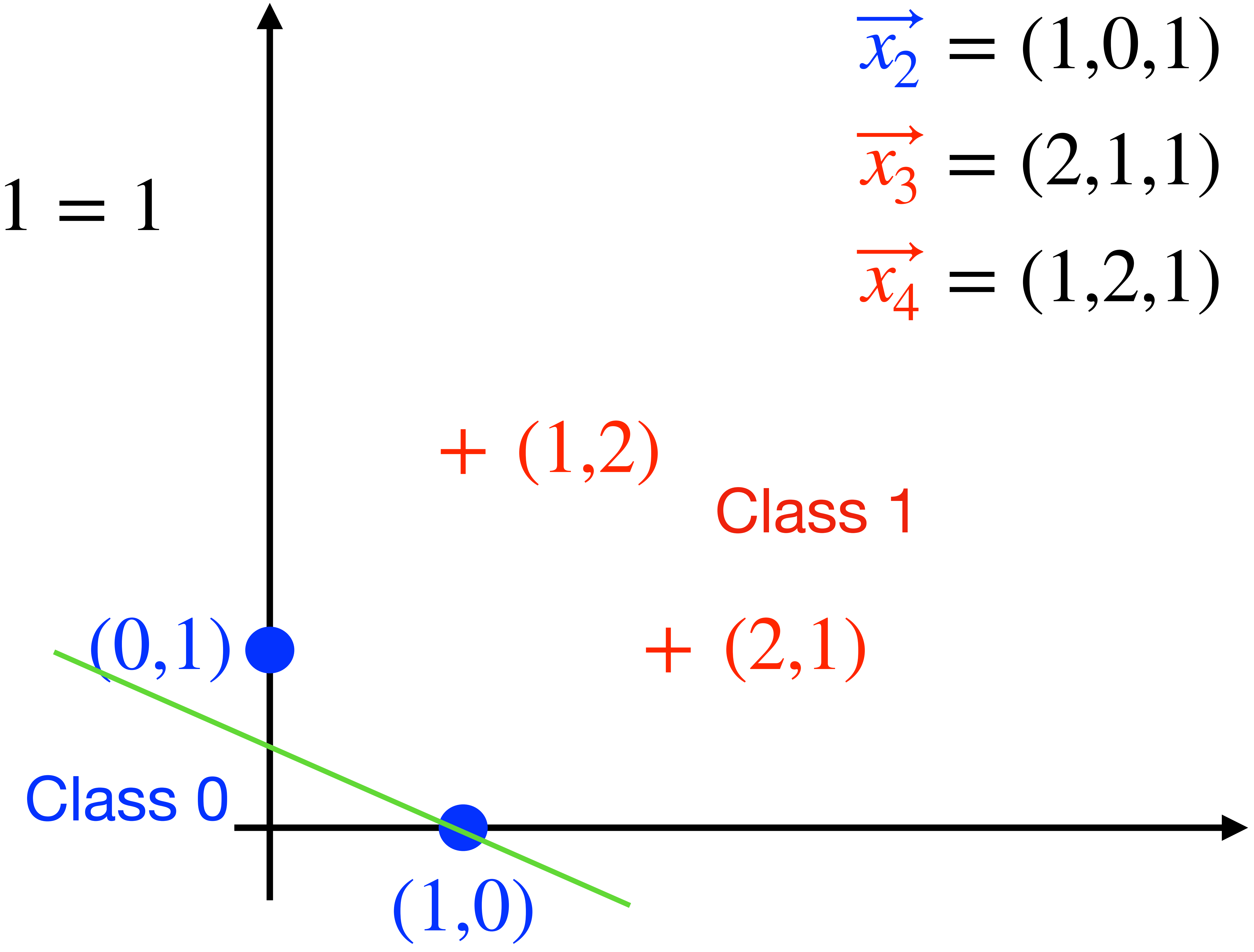
$\vec{x_2} = (1,0,1)$

$\vec{x_3} = (2,1,1)$

$\vec{x_4} = (1,2,1)$

- First Epoch:

$\vec{x_1} : \langle \vec{w}, \vec{x_1} \rangle = 1 \times 0 + 2 \times 1 + (-1) \times 1 = 1$



# Example: Training the Perceptron

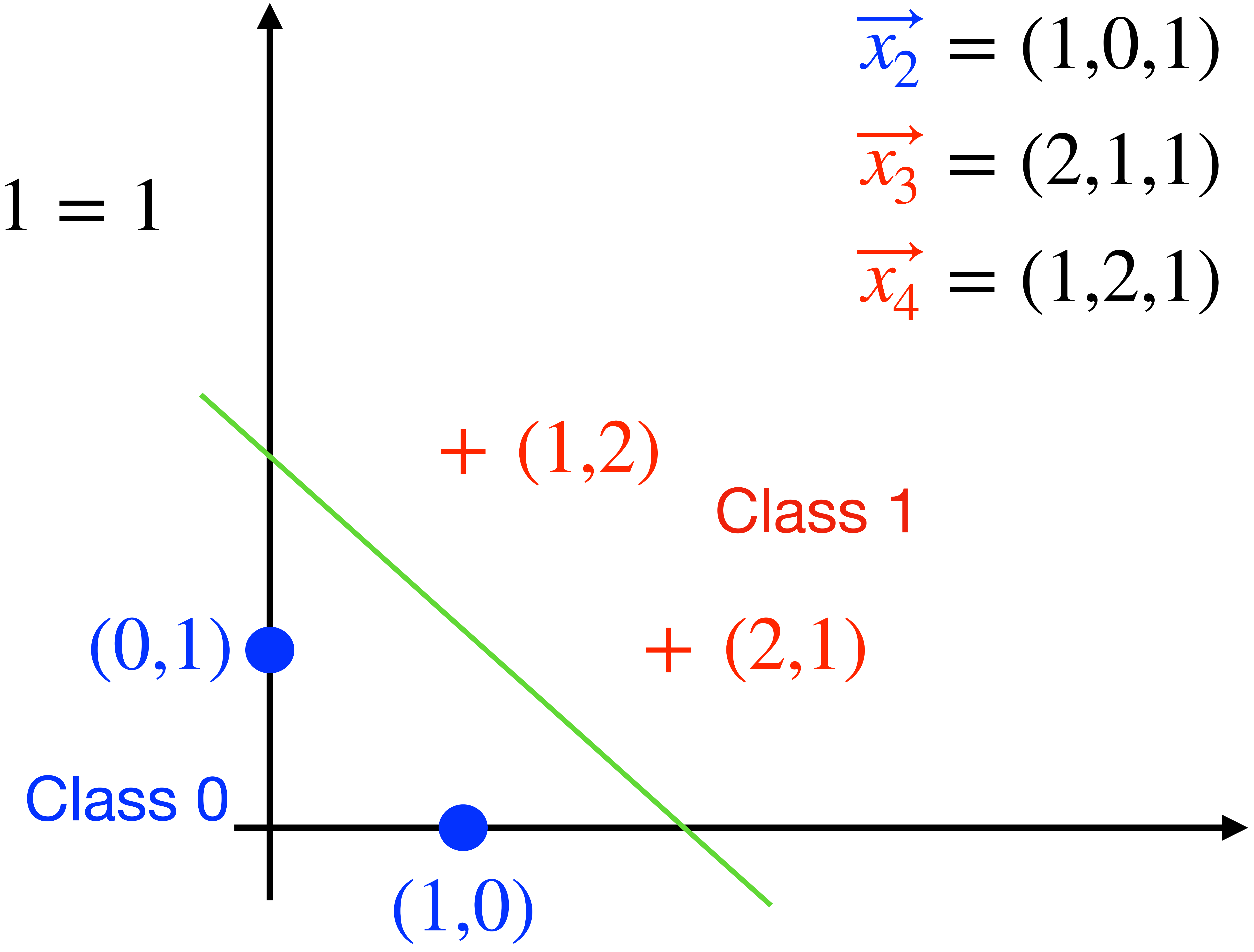
$\vec{w} = (1,1,-2)$
$\vec{x_1} = (0,1,1)$
$\vec{x_2} = (1,0,1)$
$\vec{x_3} = (2,1,1)$
$\vec{x_4} = (1,2,1)$

- First Epoch:

$\vec{x_1} : \langle \vec{w}, \vec{x_1} \rangle = 1 \times 0 + 2 \times 1 + (-1) \times 1 = 1$

wrong prediction

update $\vec{w} \leftarrow \vec{w} - \vec{x_1} = (1,1,-2)$



# Example: Training the Perceptron

$\vec{w} = (1,1,-2)$

$\vec{x_1} = (0,1,1)$

$\vec{x_2} = (1,0,1)$

$\vec{x_3} = (2,1,1)$

$\vec{x_4} = (1,2,1)$

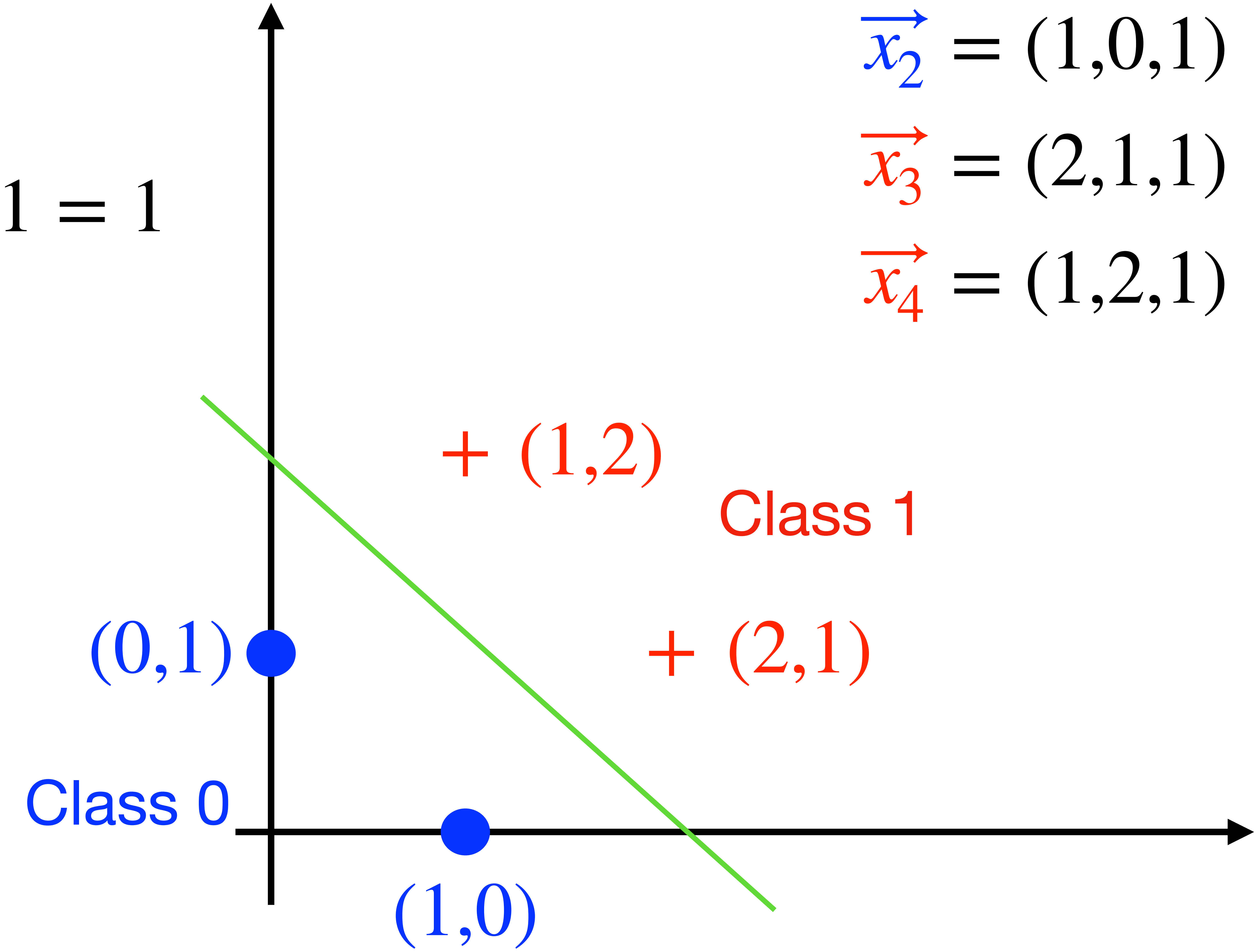
- First Epoch:

$\vec{x_1} : \langle \vec{w}, \vec{x_1} \rangle = 1 \times 0 + 2 \times 1 + (-1) \times 1 = 1$

wrong prediction

update $\vec{w} \leftarrow \vec{w} - \vec{x_1} = (1,1,-2)$

$\vec{x_2} : \langle \vec{w}, \vec{x_2} \rangle = -1$



# Example: Training the Perceptron

$\vec{w} = (1,1,-2)$
$\vec{x_1} = (0,1,1)$
$\vec{x_2} = (1,0,1)$
$\vec{x_3} = (2,1,1)$
$\vec{x_4} = (1,2,1)$

- First Epoch:

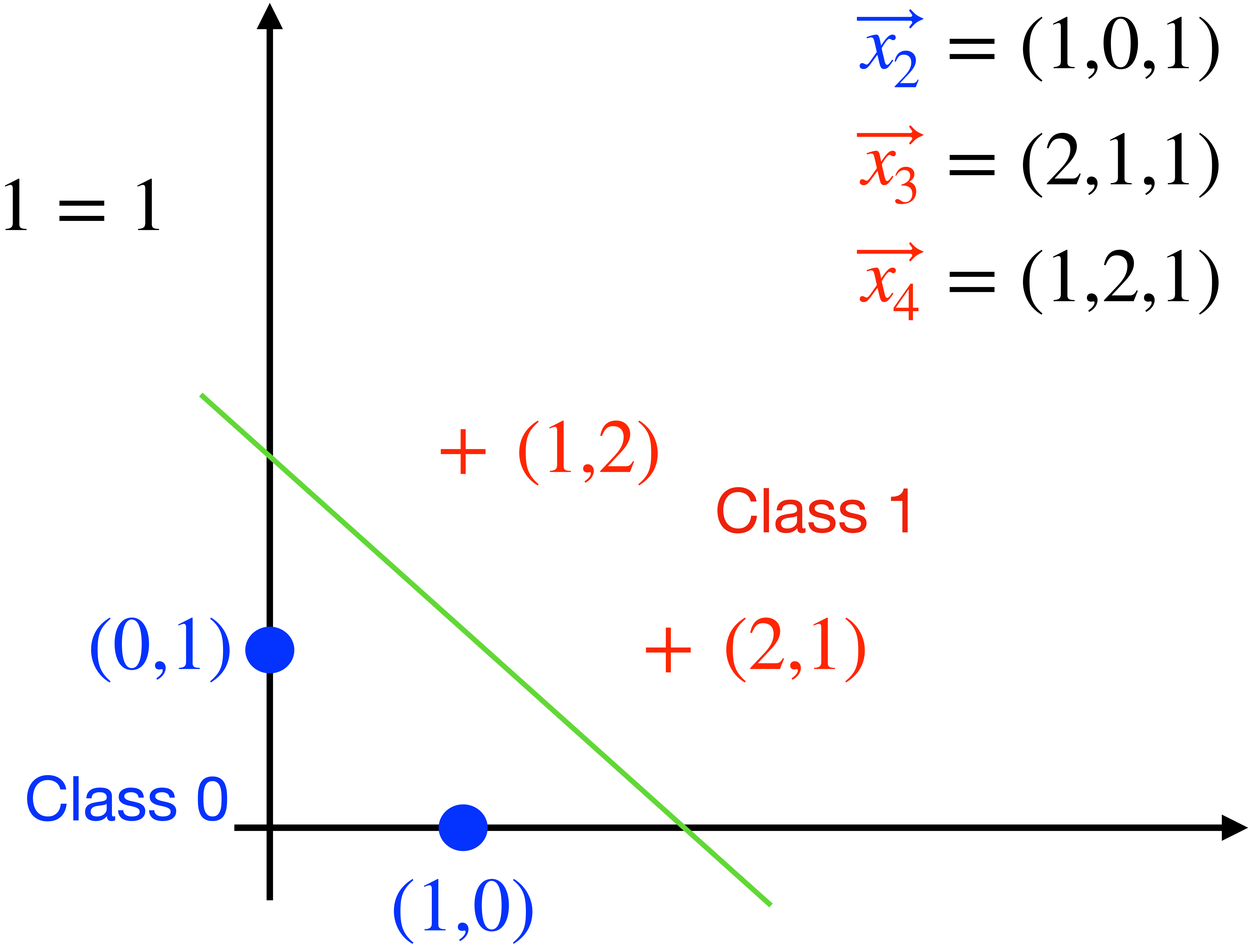
$\vec{x_1}: \langle \vec{w}, \vec{x_1} \rangle = 1 \times 0 + 2 \times 1 + (-1) \times 1 = 1$

wrong prediction

update $\vec{w} \leftarrow \vec{w} - \vec{x_1} = (1,1,-2)$

$\vec{x_2}: \langle \vec{w}, \vec{x_2} \rangle = -1$

correct prediction, no update



# Example: Training the Perceptron

$\vec{w} = (1,1,-2)$
$\vec{x_1} = (0,1,1)$
$\vec{x_2} = (1,0,1)$
$\vec{x_3} = (2,1,1)$
$\vec{x_4} = (1,2,1)$

- First Epoch:

$\vec{x_1}: \langle \vec{w}, \vec{x_1} \rangle = 1 \times 0 + 2 \times 1 + (-1) \times 1 = 1$

wrong prediction

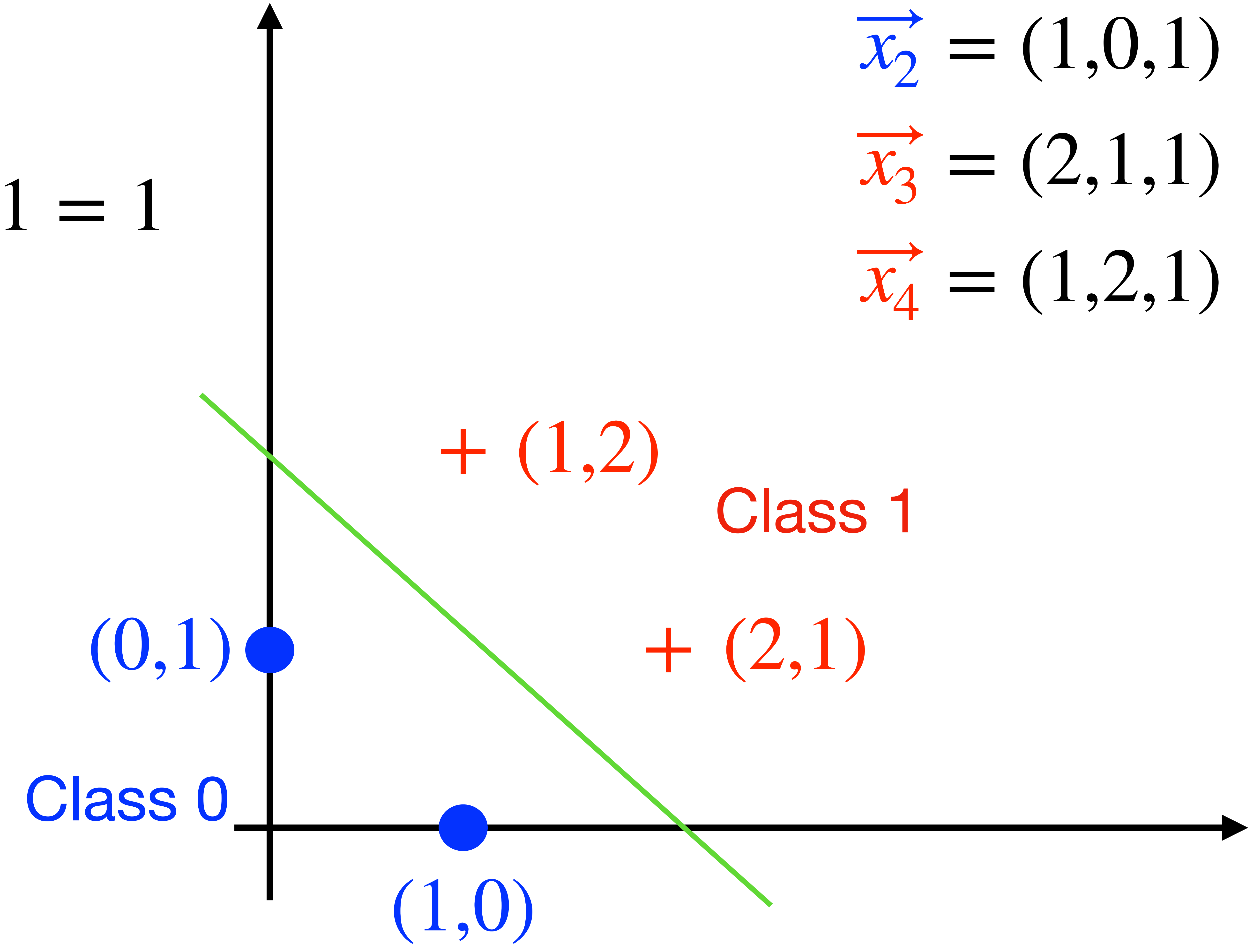
update $\vec{w} \leftarrow \vec{w} - \vec{x_1} = (1,1,-2)$

$\vec{x_2}: \langle \vec{w}, \vec{x_2} \rangle = -1$

correct prediction, no update

$\vec{x_3}: \langle \vec{w}, \vec{x_3} \rangle = 1$

correct prediction, no update



# Example: Training the Perceptron

$\vec{w} = (1,1,-2)$

$\vec{x_1} = (0,1,1)$

$\vec{x_2} = (1,0,1)$

$\vec{x_3} = (2,1,1)$

$\vec{x_4} = (1,2,1)$

- First Epoch:

$\vec{x_1} : \langle \vec{w}, \vec{x_1} \rangle = 1 \times 0 + 2 \times 1 + (-1) \times 1 = 1$

wrong prediction

update $\vec{w} \leftarrow \vec{w} - \vec{x_1} = (1,1,-2)$

$\vec{x_2} : \langle \vec{w}, \vec{x_2} \rangle = -1$

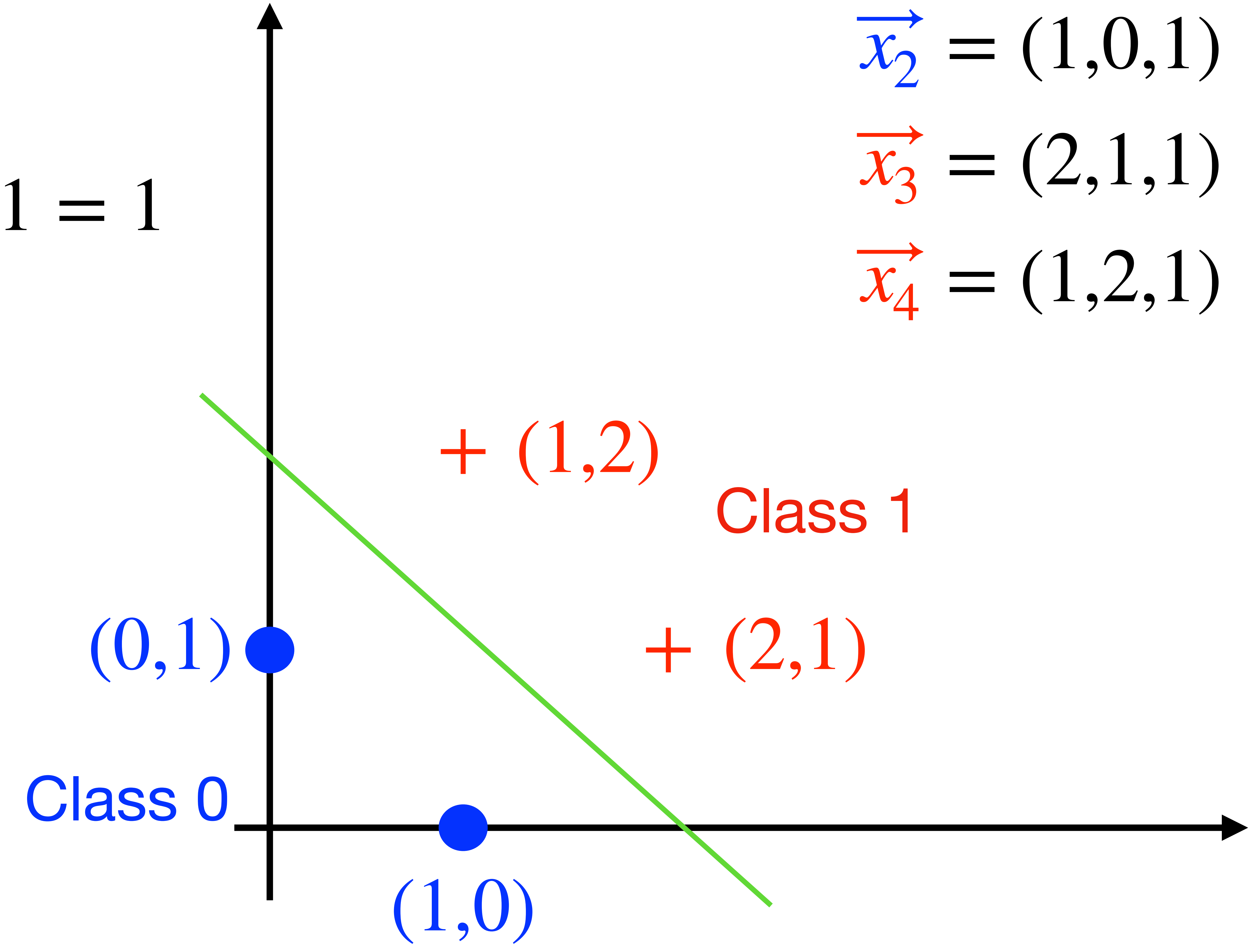
correct prediction, no update

$\vec{x_3} : \langle \vec{w}, \vec{x_3} \rangle = 1$

correct prediction, no update

$\vec{x_4} : \langle \vec{w}, \vec{x_4} \rangle = 1$

correct prediction, no update



# Example: Training the Perceptron

$\vec{w} = (1,1,-2)$

$\vec{x}_1 = (0,1,1)$

$\vec{x}_2 = (1,0,1)$

$\vec{x}_3 = (2,1,1)$

$\vec{x}_4 = (1,2,1)$

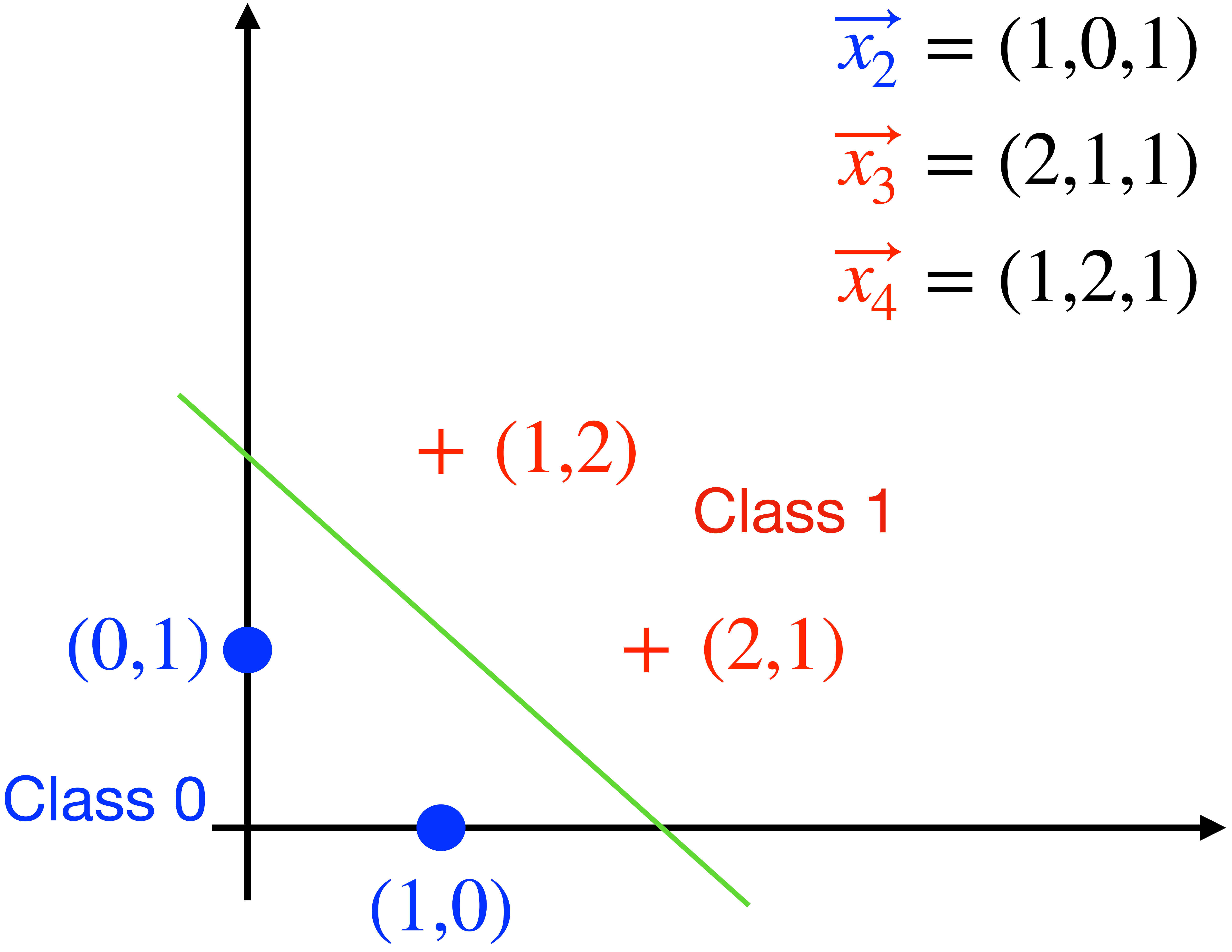
- Second Epoch:

$\vec{x}_1 : \langle \vec{w}, \vec{x}_1 \rangle = -1,$ correct

$\vec{x}_2 : \langle \vec{w}, \vec{x}_2 \rangle = -1,$ correct

$\vec{x}_3 : \langle \vec{w}, \vec{x}_3 \rangle = 1,$ correct

$\vec{x}_4 : \langle \vec{w}, \vec{x}_4 \rangle = 1,$ correct



# Example: Training the Perceptron

$\vec{w} = (1,1,-2)$

$\vec{x_1} = (0,1,1)$

$\vec{x_2} = (1,0,1)$

$\vec{x_3} = (2,1,1)$

$\vec{x_4} = (1,2,1)$

- Second Epoch:

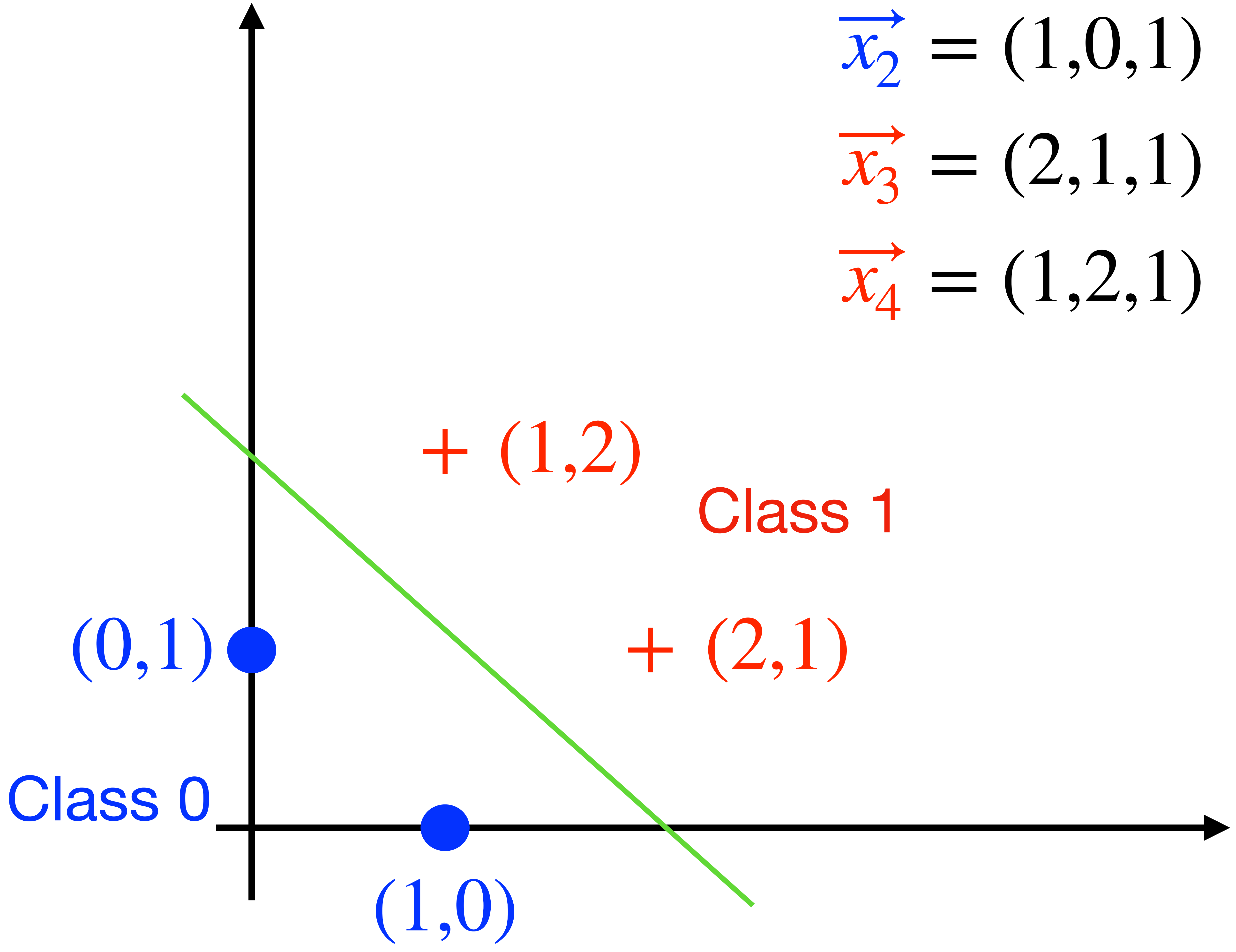
$\vec{x_1}: \langle \vec{w}, \vec{x_1} \rangle = -1,$ correct

$\vec{x_2}: \langle \vec{w}, \vec{x_2} \rangle = -1,$ correct

$\vec{x_3}: \langle \vec{w}, \vec{x_3} \rangle = 1,$ correct

$\vec{x_4}: \langle \vec{w}, \vec{x_4} \rangle = 1,$ correct

- Success!



# Limitation: XOR Problem (Minsky & Papert, 1969)

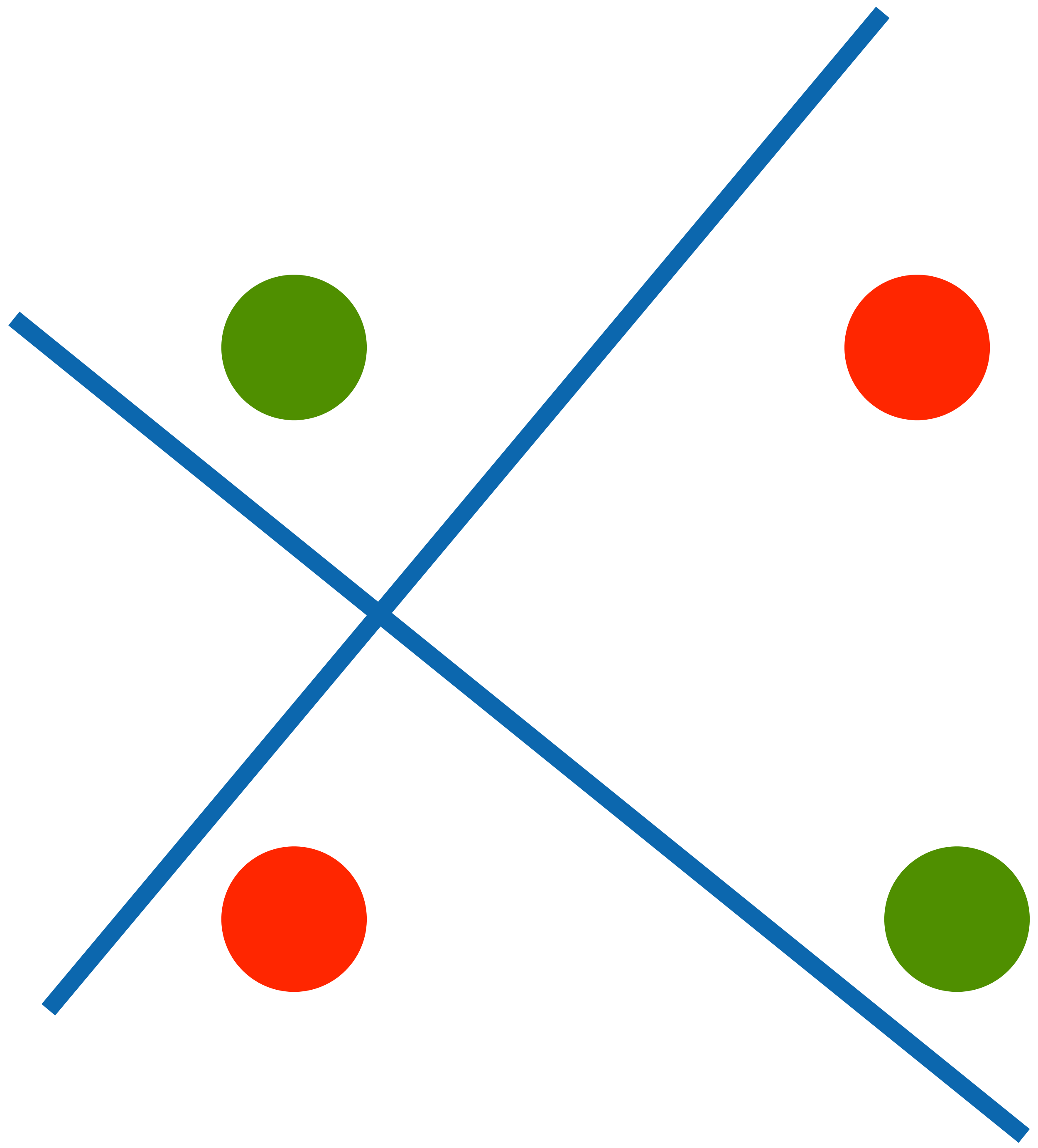
The perceptron cannot learn an XOR function
(neurons can only generate linear separators)

$x_1 = 1, x_2 = 1, y = 0$

$x_1 = 1, x_2 = 0, y = 1$

$x_1 = 0, x_2 = 1, y = 1$

$x_1 = 0, x_2 = 0, y = 0$



# Limitation: XOR Problem (Minsky & Papert, 1969)

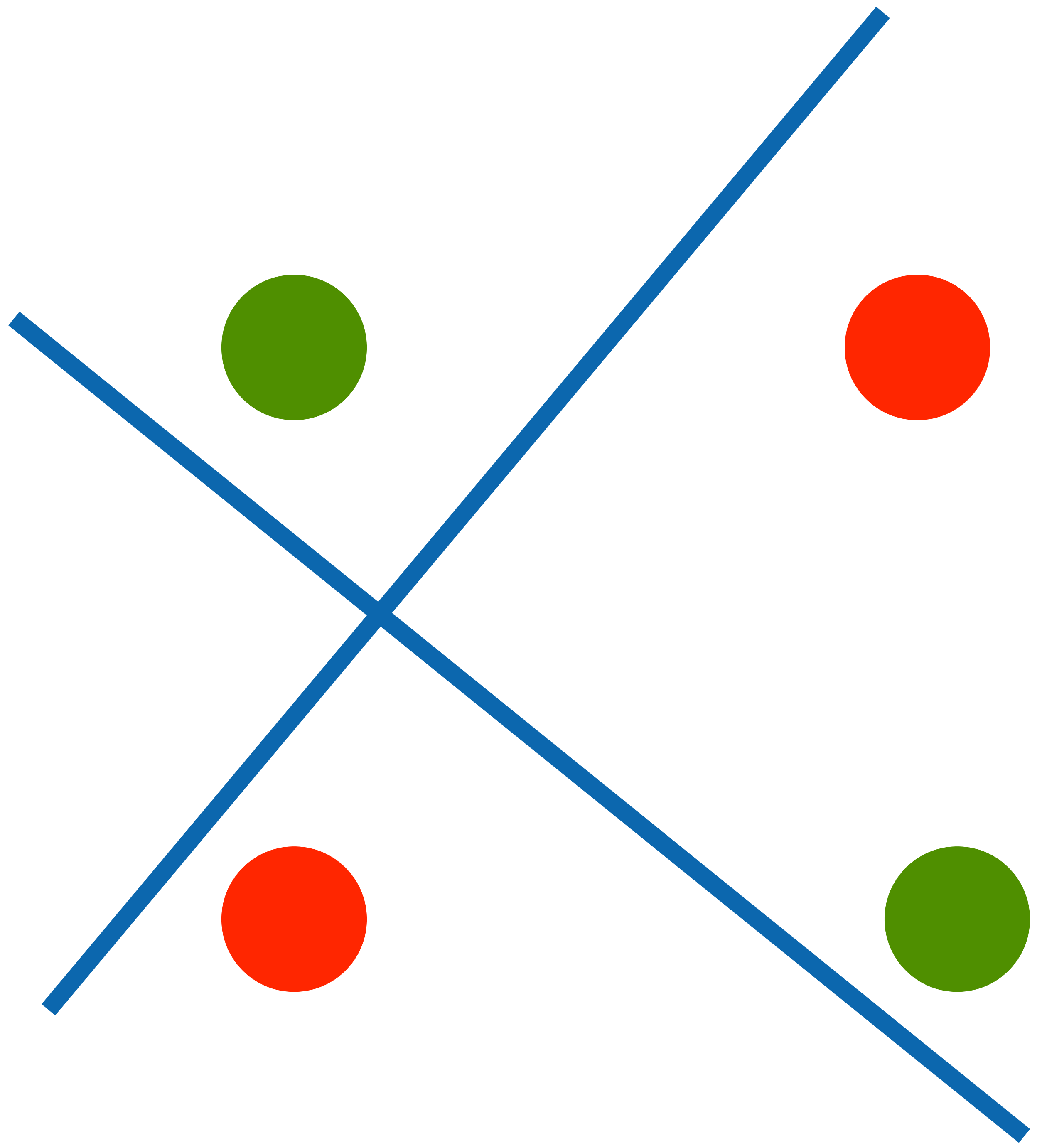
The perceptron cannot learn an XOR function
(neurons can only generate linear separators)

$$x_1 = 1, x_2 = 1, y = 0$$

$$x_1 = 1, x_2 = 0, y = 1$$

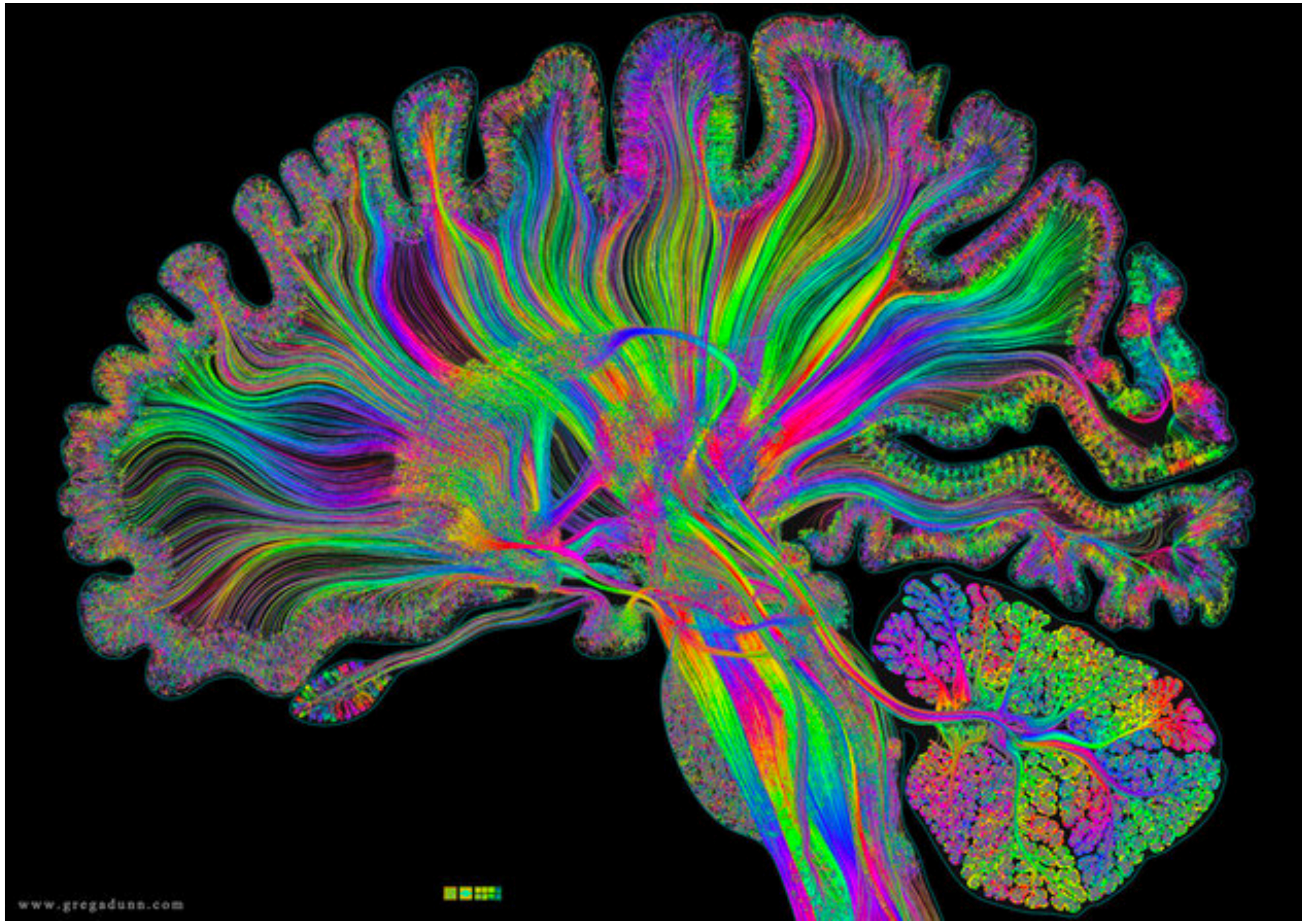
$$x_1 = 0, x_2 = 1, y = 1$$

$$x_1 = 0, x_2 = 0, y = 0$$



On the other hand, perceptron can represent AND OR NOT, and their composition can represent all logic functions

# Multilayer Perceptron

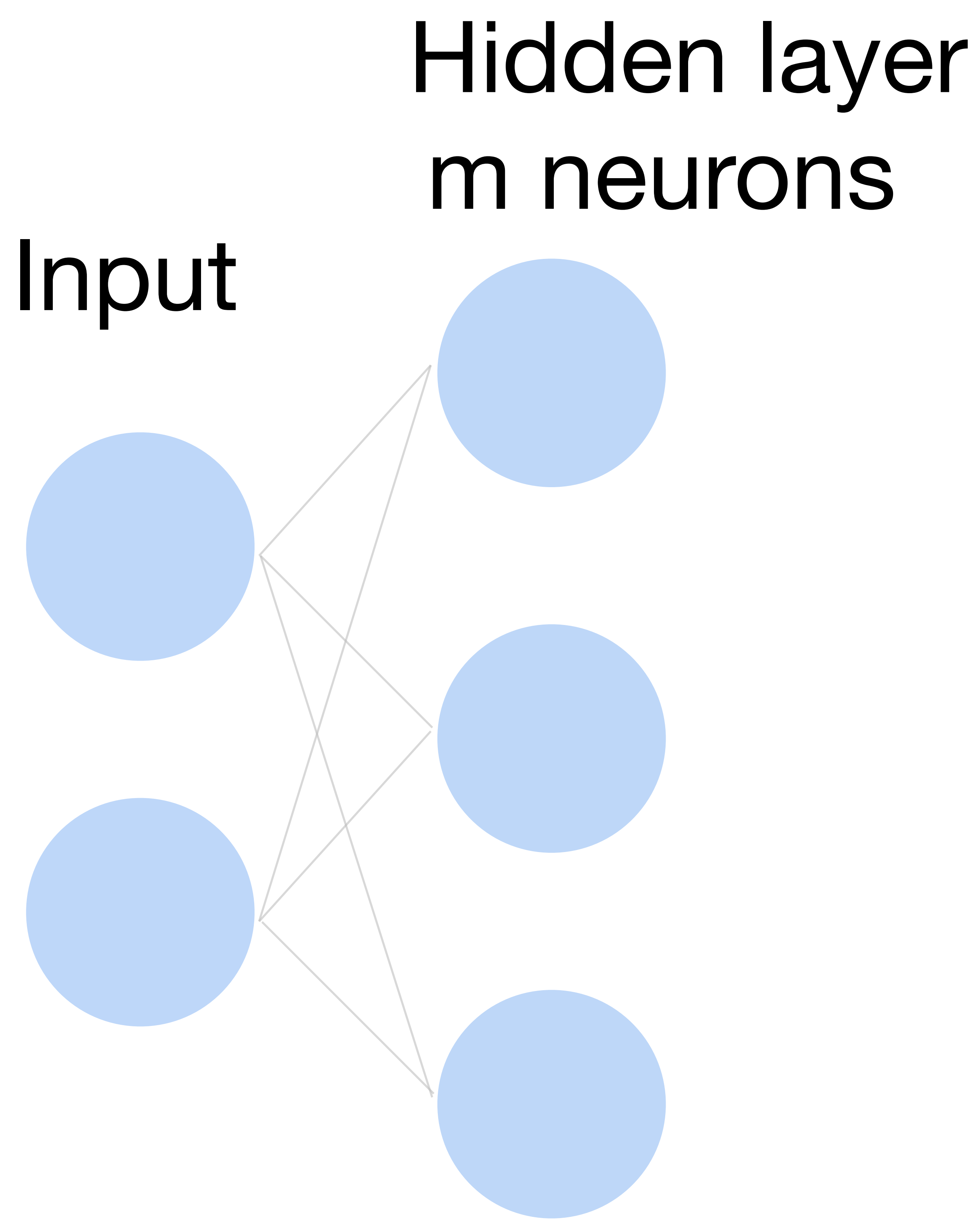


# Single Hidden Layer

- Input $\mathbf{x} \in \mathbb{R}^d$
- Hidden $\mathbf{W} \in \mathbb{R}^{m \times d}, \mathbf{b} \in \mathbb{R}^m$
- Intermediate output

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$\sigma$ is an element-wise activation function

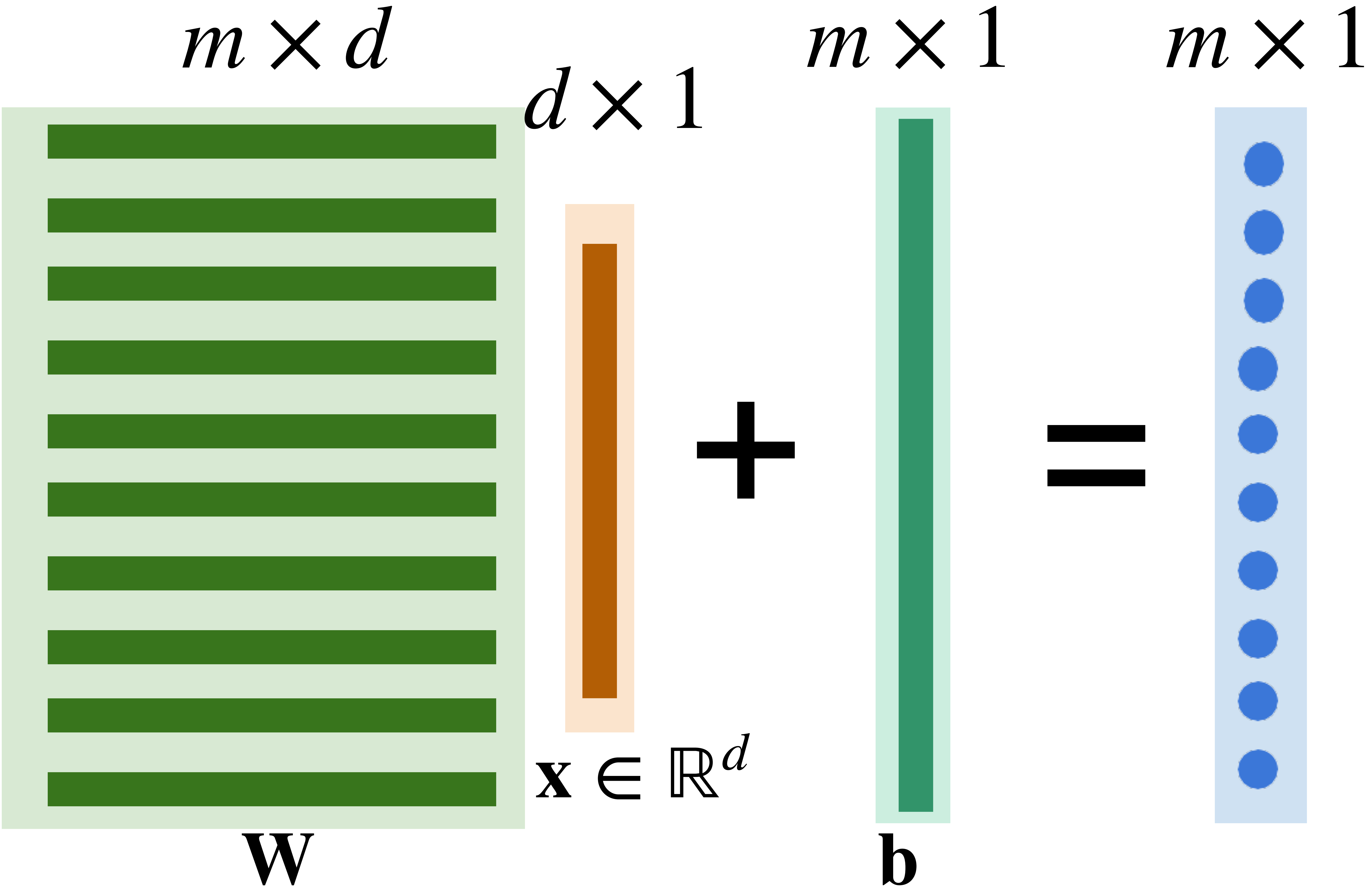


# Neural networks with one hidden layer

**Key elements**: linear operations + nonlinear activations

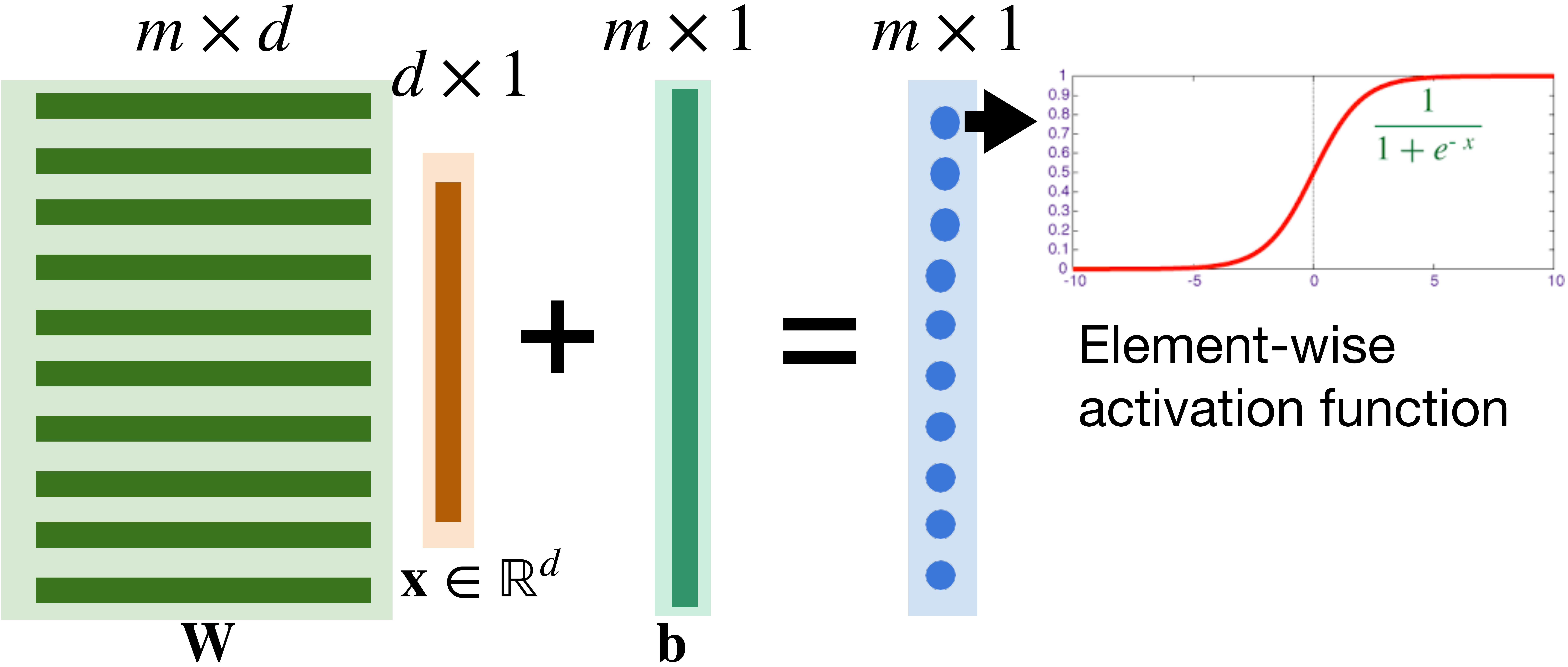
# Neural networks with one hidden layer

**Key elements:** linear operations + nonlinear activations



# Neural networks with one hidden layer

**Key elements:** linear operations + nonlinear activations

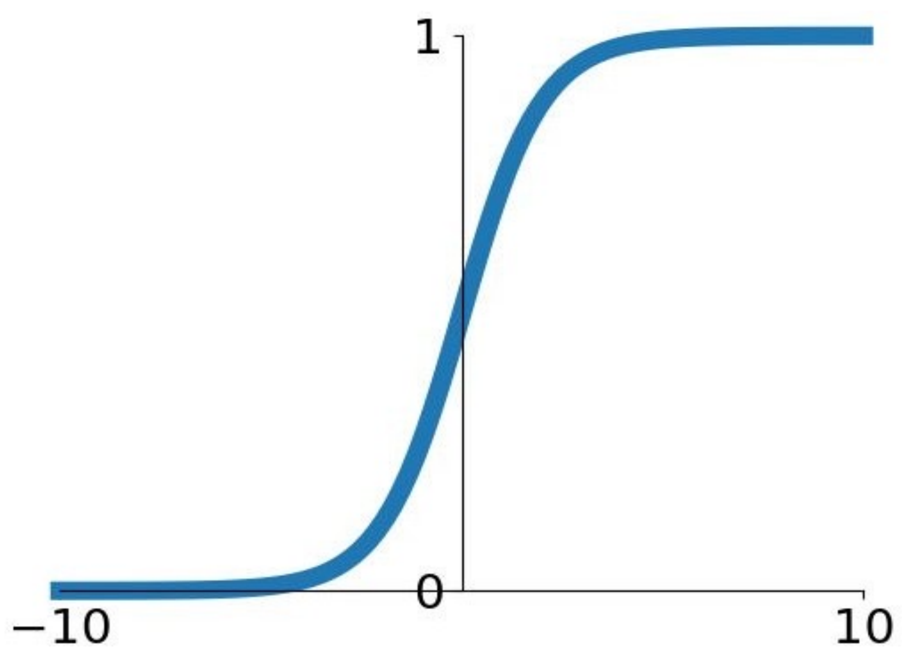


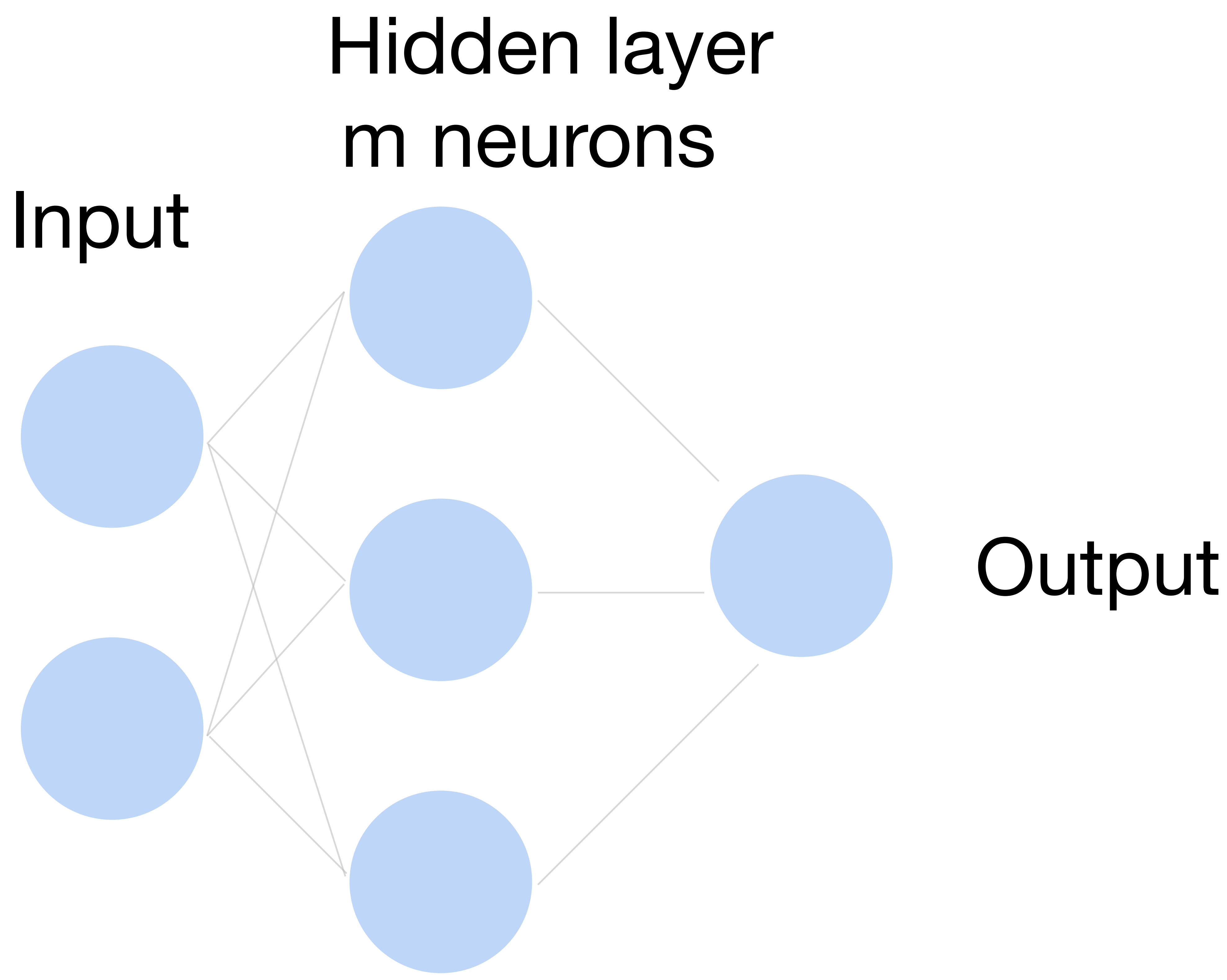
# Single Hidden Layer

- Output $f = \mathbf{w}_2^\top \mathbf{h} + b_2$
- Normalize the output into probability using sigmoid

$$p(y = 1 \mid \mathbf{x}) = \frac{1}{1 + e^{-f}}$$

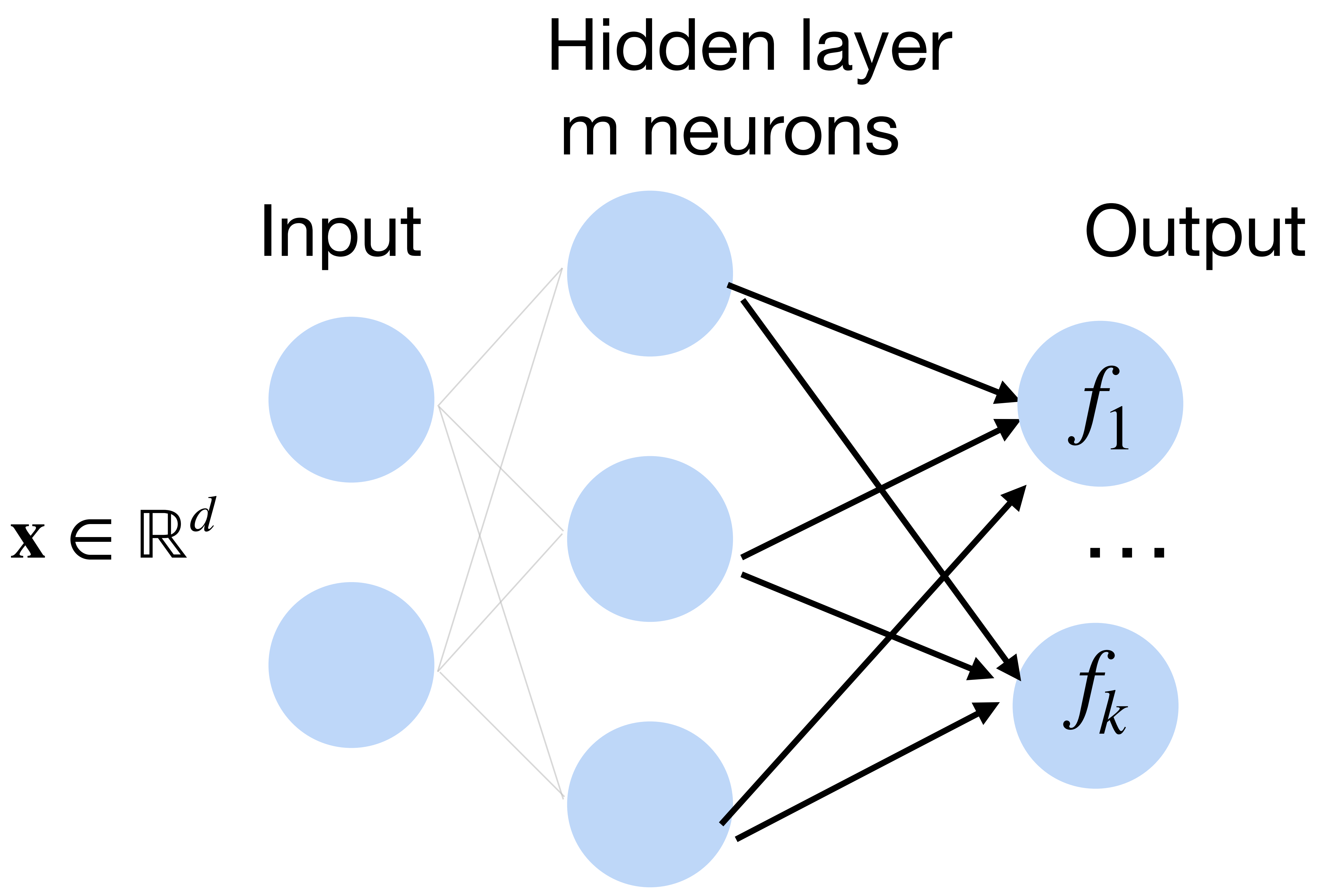
**Sigmoid**



# Multi-class classification

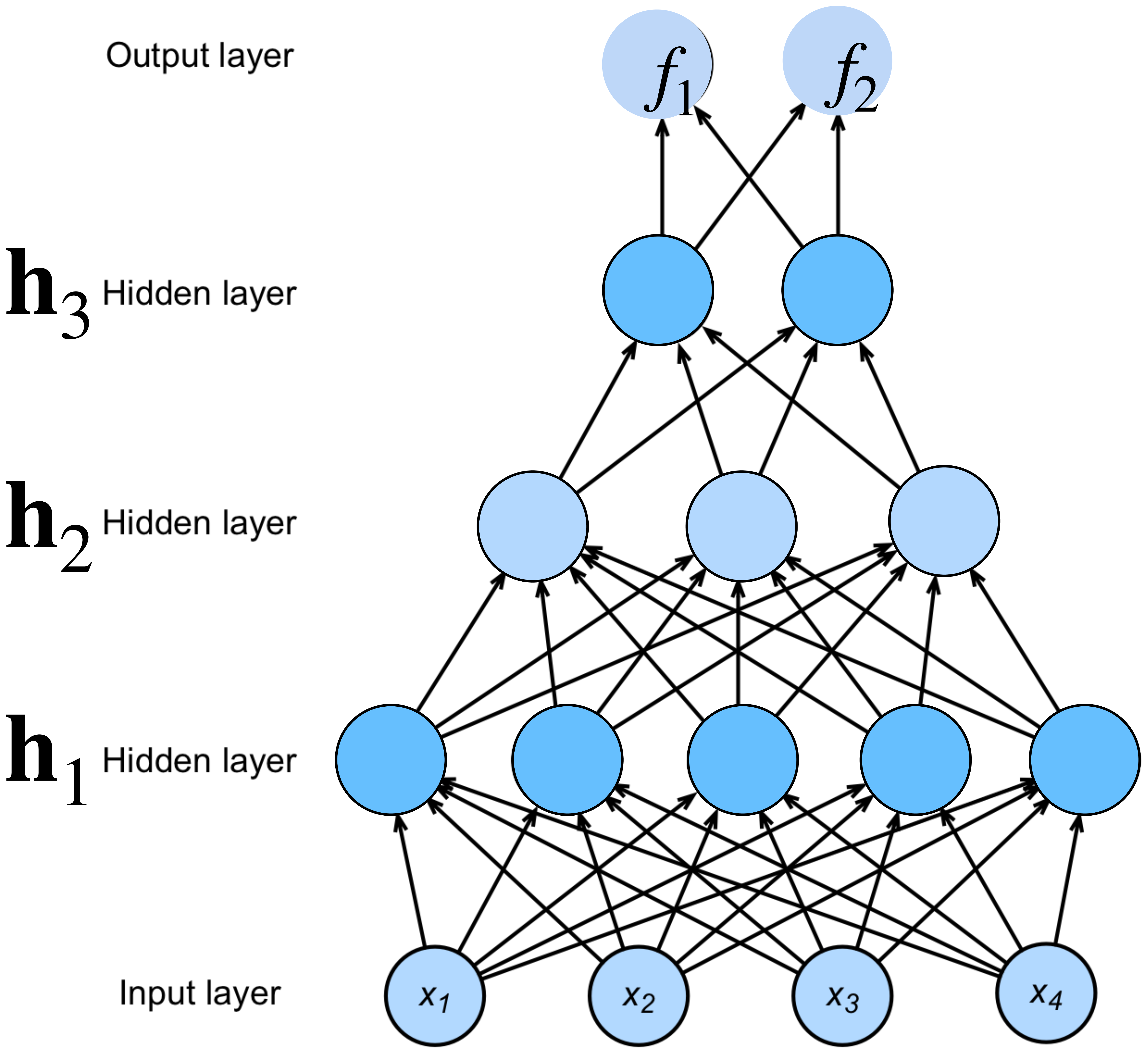
Turns outputs f into k probabilities (sum up to 1 across k classes)




$$p(y \mid \mathbf{x}) = \text{softmax}(\mathbf{f})$$

$$= \frac{\exp f_y(x)}{\sum_i^k \exp f_i(x)}$$

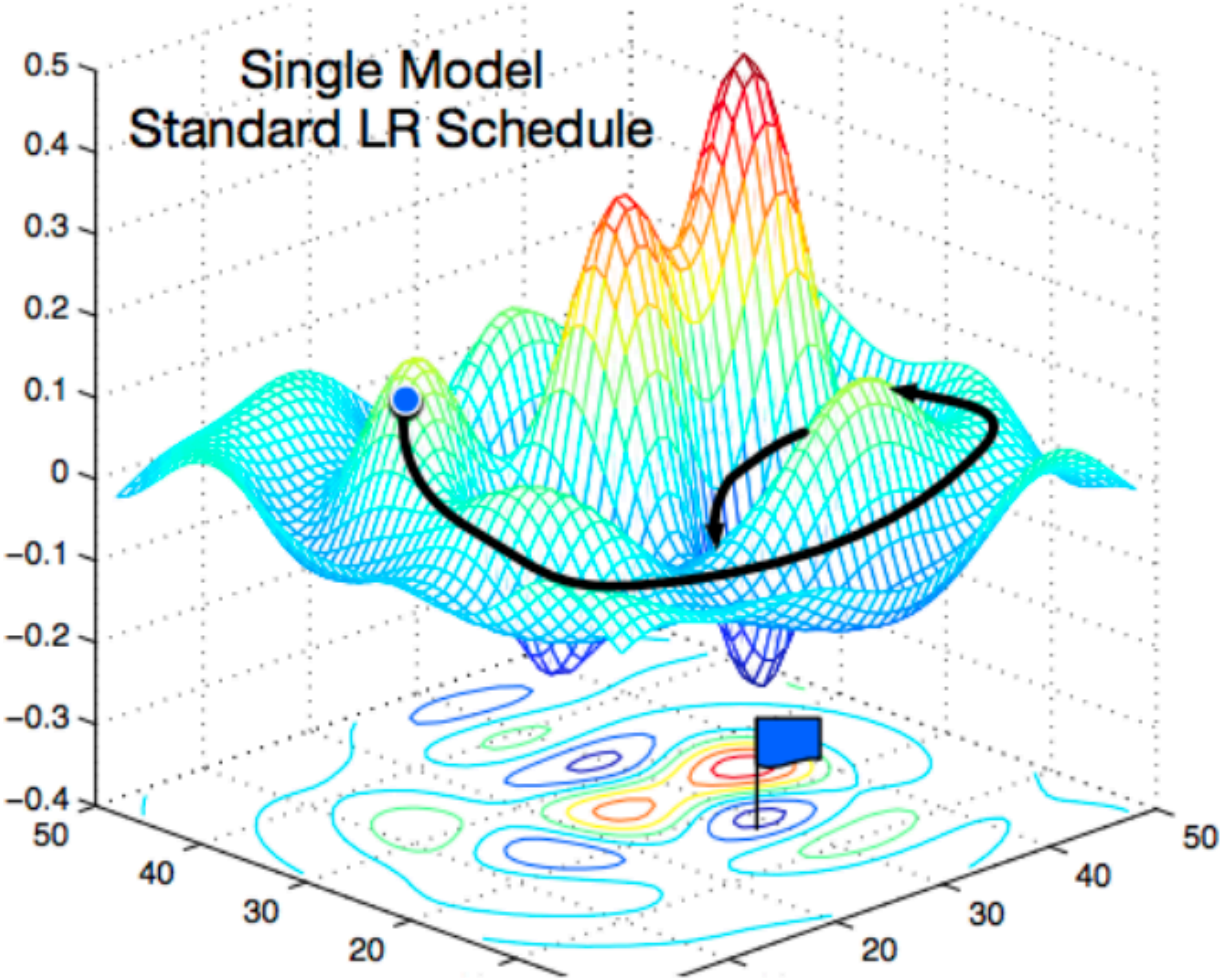
# Deep neural networks (DNNs)




$$\mathbf{h}_1 = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$
$$\mathbf{h}_2 = \sigma(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)$$
$$\mathbf{h}_3 = \sigma(\mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3)$$
$$\mathbf{f} = \mathbf{W}_4\mathbf{h}_3 + \mathbf{b}_4$$
$$\mathbf{y} = \text{softmax}(\mathbf{f})$$

NNs are composition of nonlinear functions

# Training Neural Networks




[Gao and Li et al., 2018]

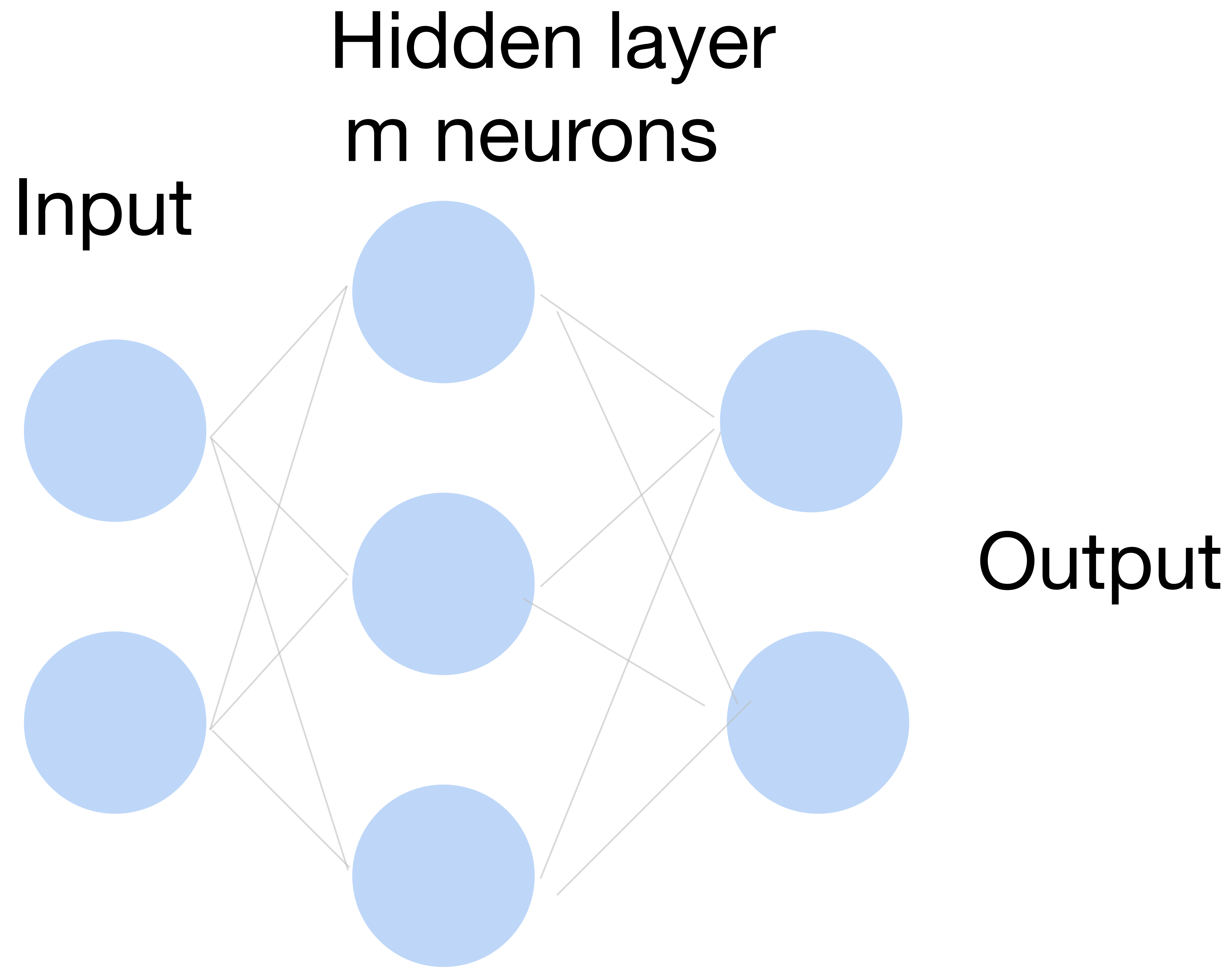
# Cross-Entropy Loss

**Loss:** $$\frac{1}{|D|} \sum_{(\mathbf{x},y)\in D} \ell(\mathbf{x}, y)$$

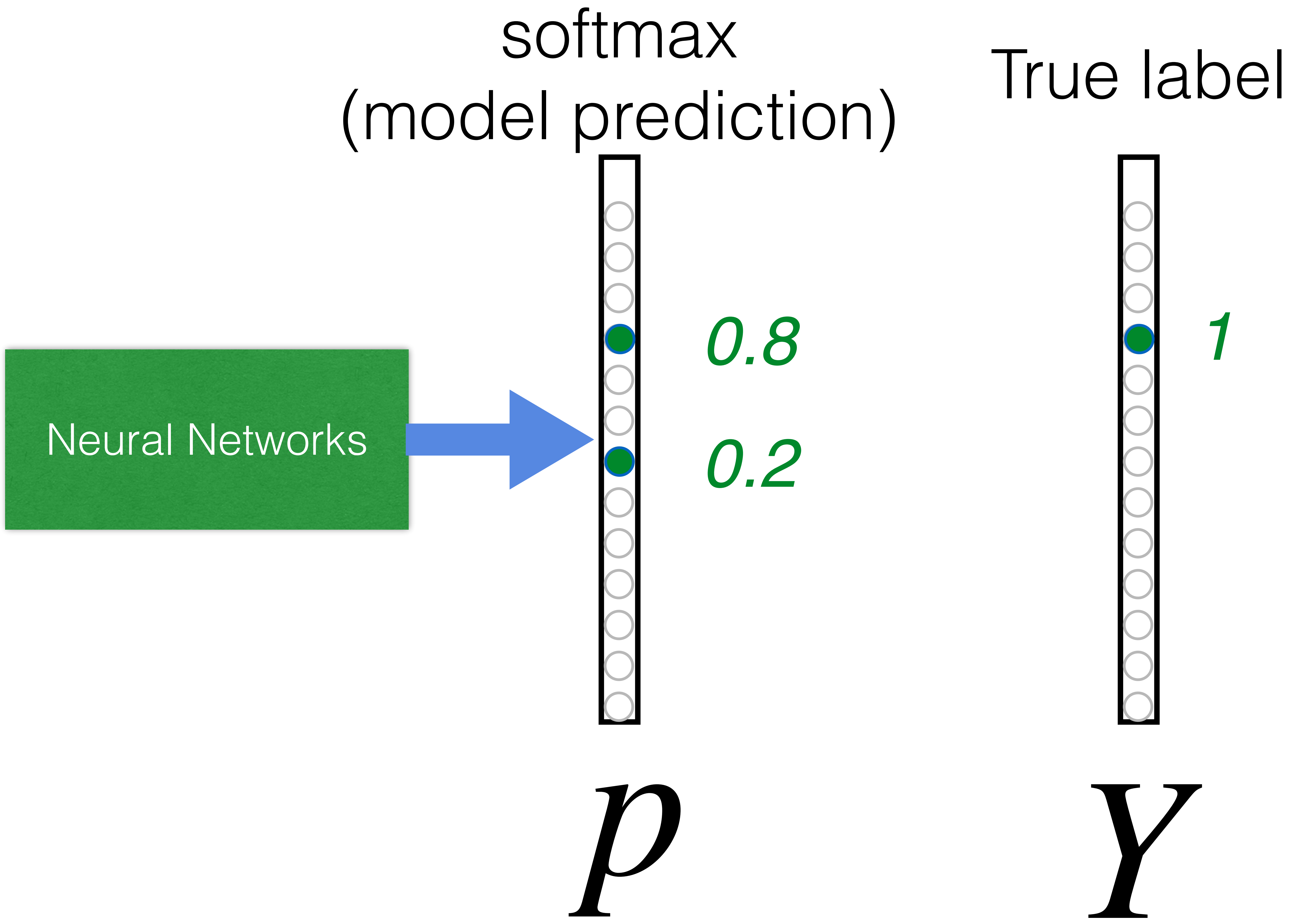
**Per sample loss (cross-entropy or softmax loss):**

$$\ell(\mathbf{x}, y) = \sum_{j=1}^{K} -Y_j \log p_j = -\log p_y$$

where $Y$ is one-hot encoding of $y$



# Cross-Entropy Loss



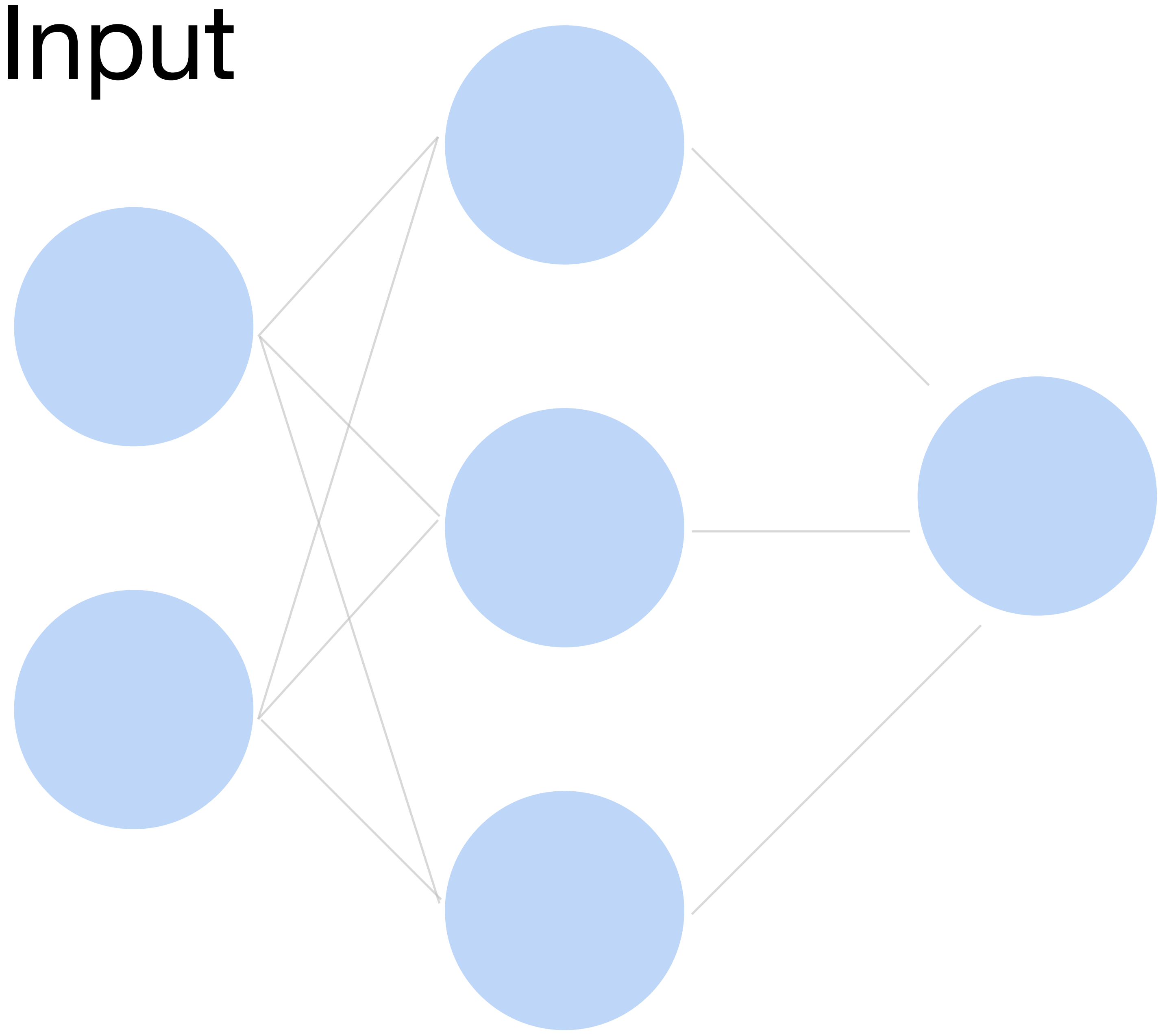

$$L_{CE} = \sum_j - Y_j \log(p_j)$$

$$= -\log(0.8)$$

**Goal**: push **p** and **Y** to be identical

# Binary Cross-Entropy Loss

**Loss:** $$\frac{1}{|D|}\sum_{(\mathbf{x},y)\in D}\ell(\mathbf{x},y)$$




**Binary cross-entropy loss can be viewed as a special case of cross-entropy loss:**

$$\ell(\mathbf{x},y) = -y\log p - (1-y)\log(1-p)$$

Think of the output as a probability vector $(p_0 = 1-p, p_1 = p)$ over the classes {0, 1}:

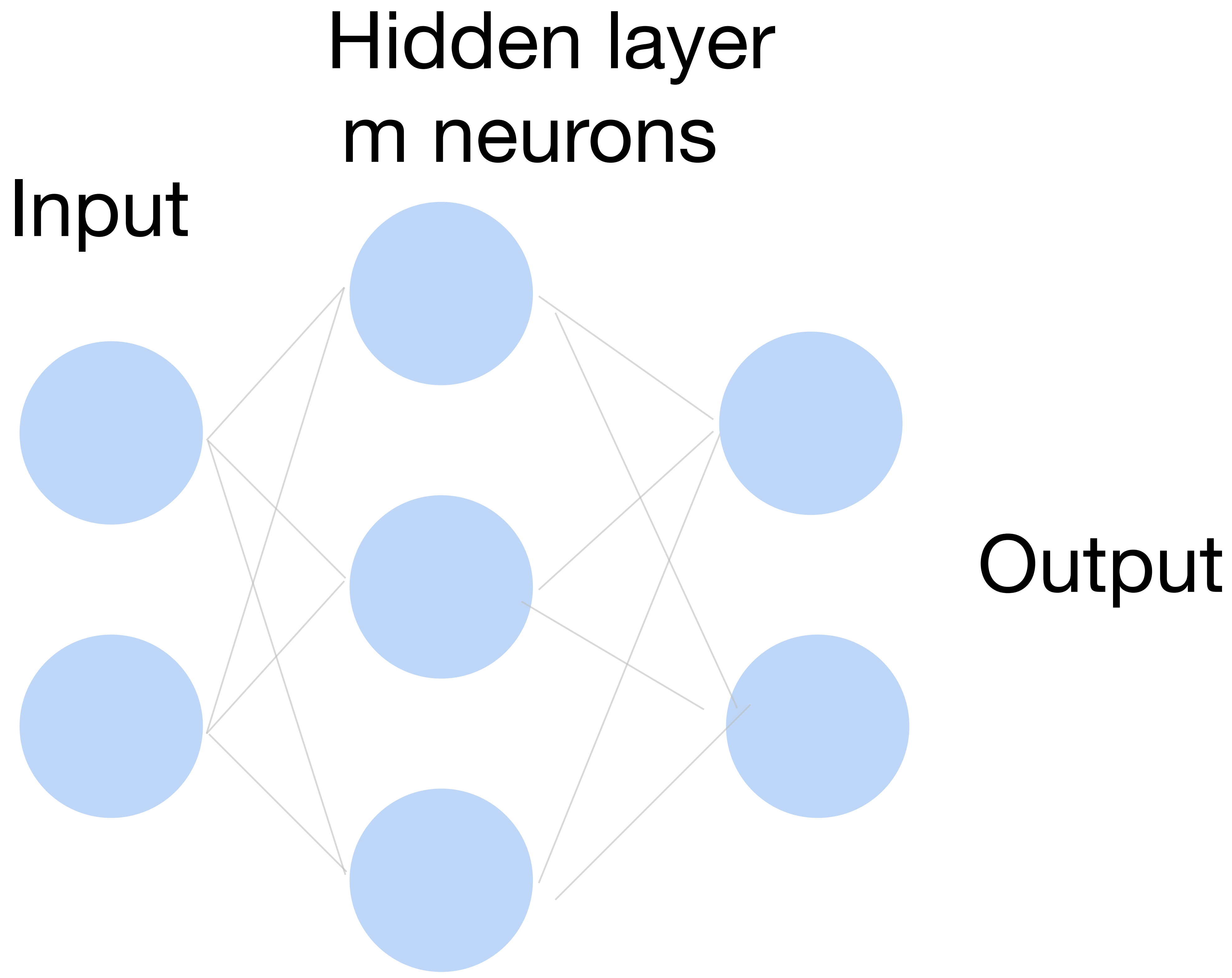
$$\ell(\mathbf{x},y) = -\log p_y$$

# How to train a neural network?

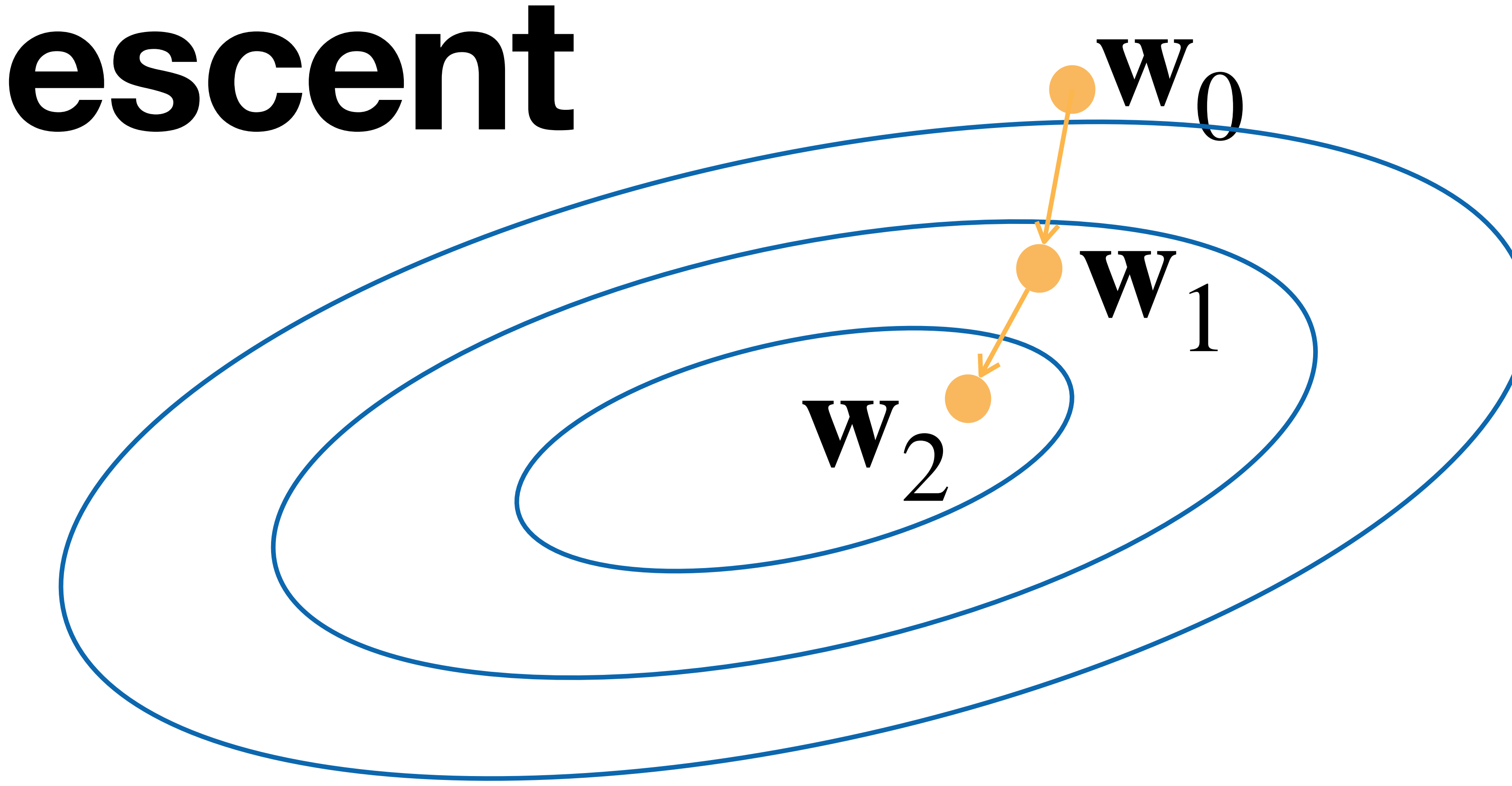
Update the weights W to minimize the loss function

$$L = \frac{1}{|D|} \sum_{(\mathbf{x},y)\in D} \ell(\mathbf{x}, y)$$

**Use gradient descent!**



# Minibatch Stochastic Gradient Descent

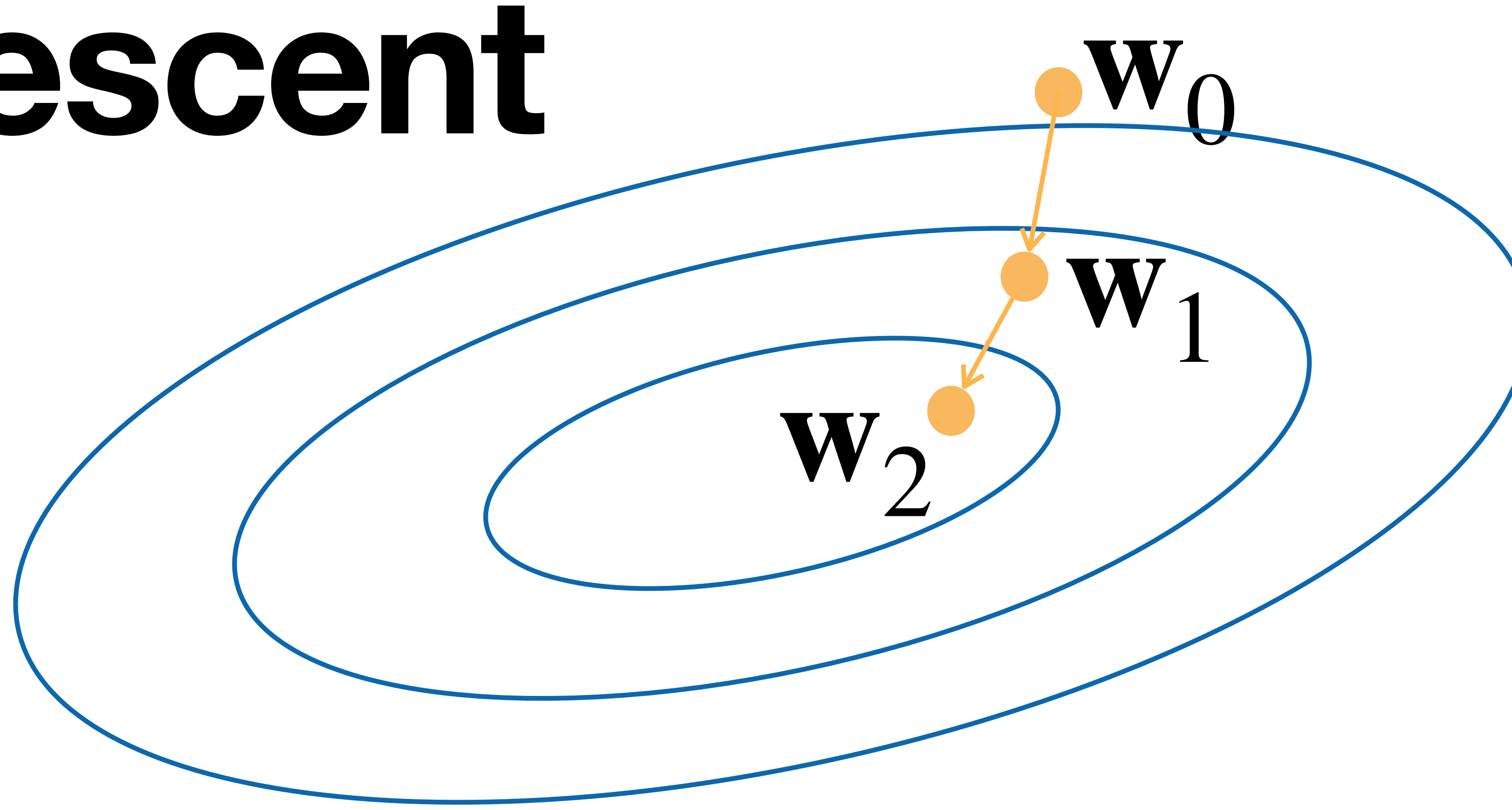



- Choose a learning rate $\alpha > 0$
- Initialize the model parameters $w_0$
- For t =1,2,…
    - **Randomly sample a subset (mini-batch)** $B \subset D$
      Update parameters:

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \alpha \frac{1}{|B|} \sum_{\mathbf{x} \in B} \frac{\partial \ell(\mathbf{x}_i, y_i)}{\partial \mathbf{w}_{t-1}}$$

- Repeat

# Minibatch Stochastic Gradient Descent



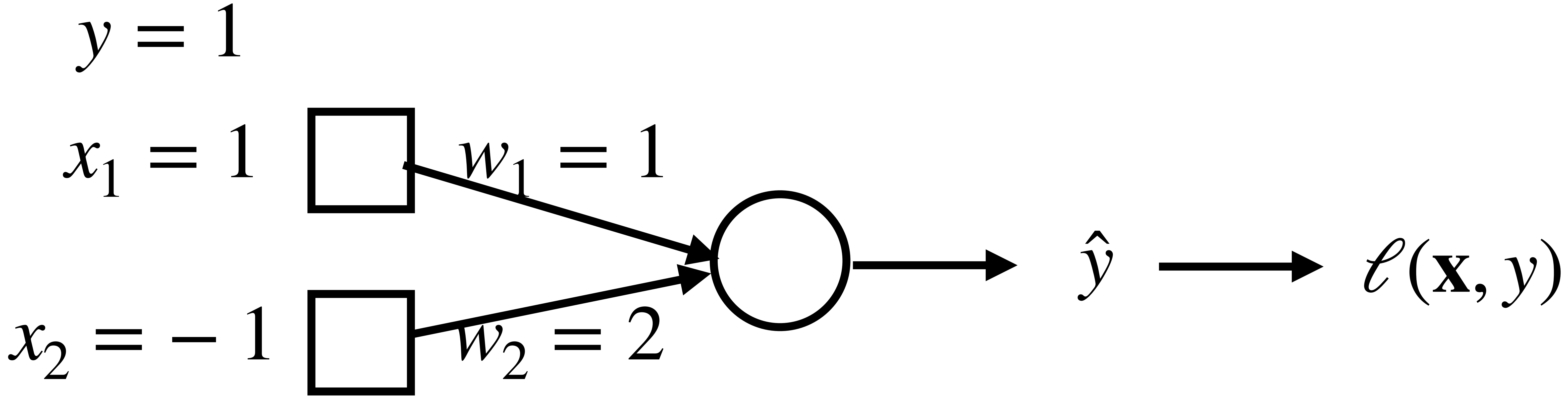

- Choose a learning rate $\alpha > 0$
- Initialize the model parameters $w_0$
- For t =1,2,…
  - **Randomly sample a subset (mini-batch)** $B \subset D$
    Update parameters:

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \alpha \frac{1}{|B|} \sum_{\mathbf{x} \in B} \frac{\partial \ell(\mathbf{x}_i, y_i)}{\partial \mathbf{w}_{t-1}}$$

The gradient w.r.t. all parameters is obtained by concatenating the partial derivatives w.r.t. each parameter

- Repeat

# Example: Backpropagation

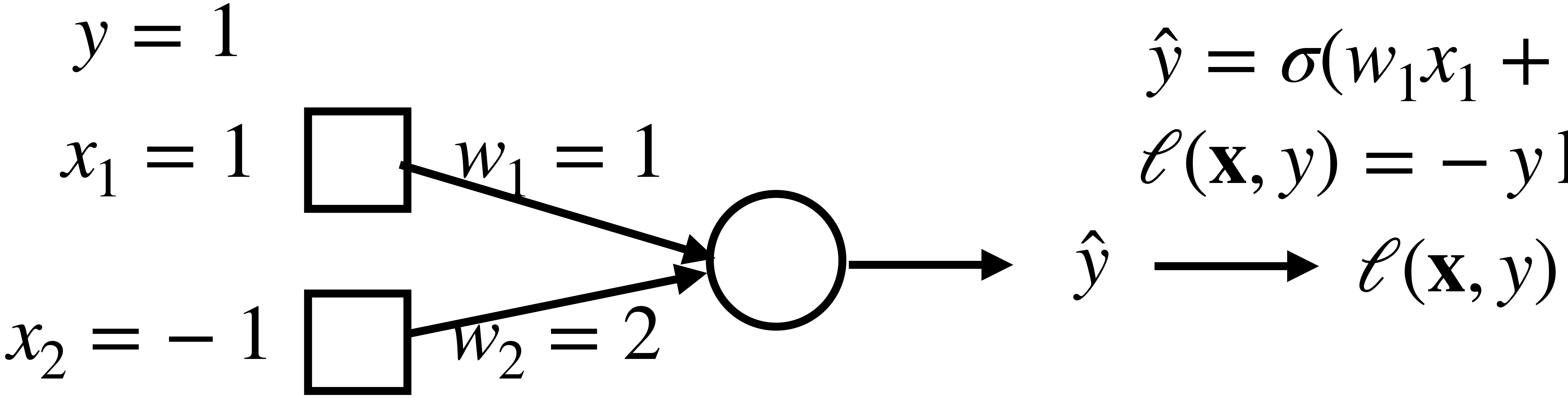


# Example: Backpropagation

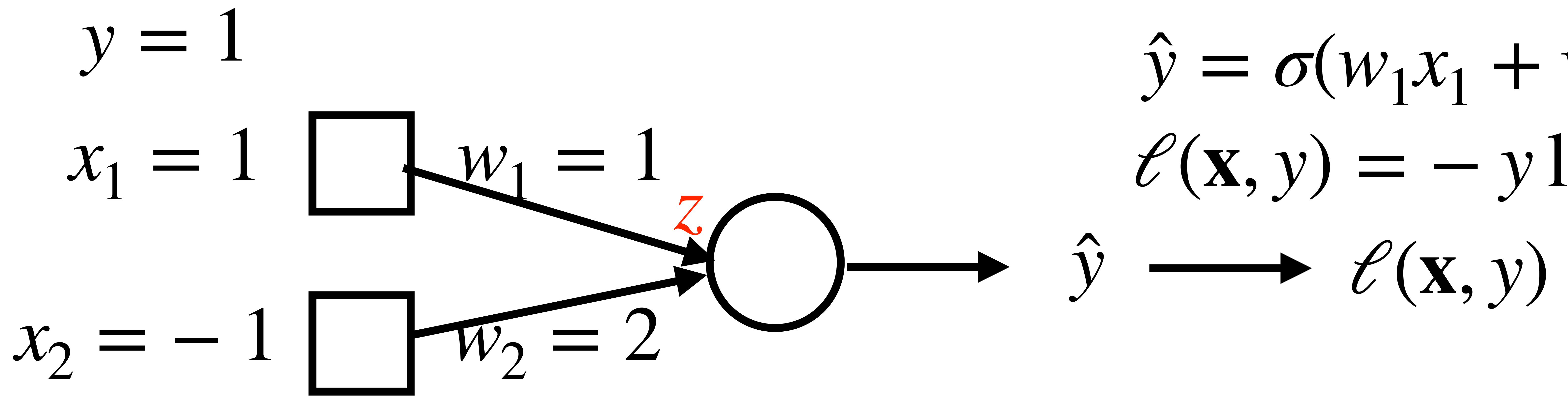
sigmoid activation, binary cross-entropy loss

$$\hat{y} = \sigma(w_1 x_1 + w_2 x_2)$$

$$\ell(\mathbf{x}, y) = -y \log \hat{y} - (1-y)\log(1-\hat{y})$$



# Example: Backpropagation




sigmoid activation, binary cross-entropy loss

$$\hat{y} = \sigma(w_1 x_1 + w_2 x_2)$$

$$\ell(\mathbf{x}, y) = -y \log \hat{y} - (1 - y)\log(1 - \hat{y})$$

- Forward:

$$z = w_1 x_1 + w_2 x_2 = 1 \times 1 + (-1) \times 2 = -1$$

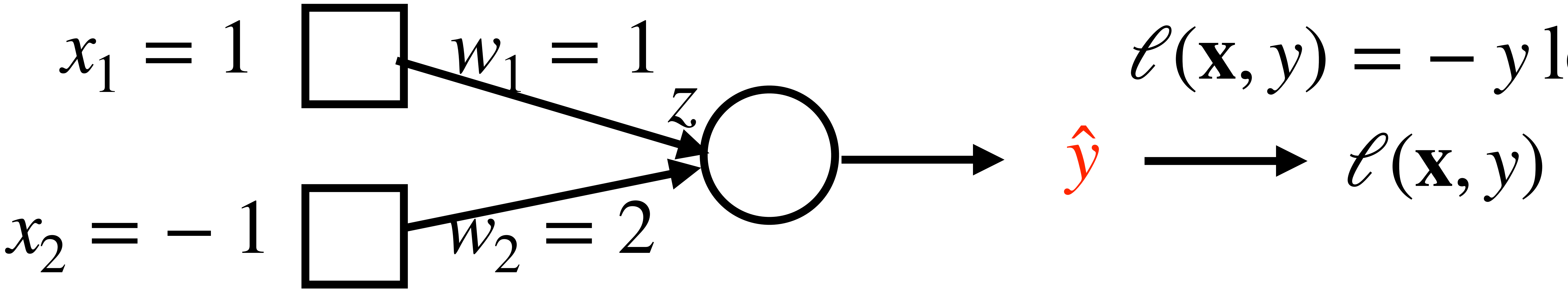
# Example: Backpropagation

sigmoid activation, binary cross-entropy loss

$$\hat{y} = \sigma(w_1 x_1 + w_2 x_2)$$

$$\ell(\mathbf{x}, y) = -y \log \hat{y} - (1-y)\log(1-\hat{y})$$

$y = 1$




- Forward:

$$z = w_1 x_1 + w_2 x_2 = 1 \times 1 + (-1) \times 2 = -1$$

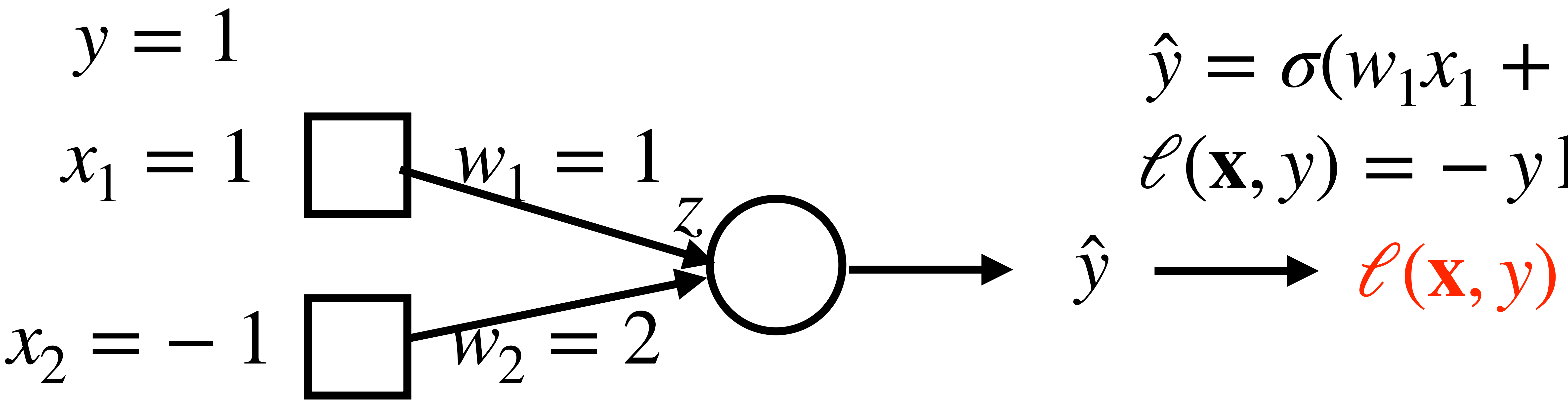
$$\hat{y} = \sigma(z) = \sigma(-1)$$

# Example: Backpropagation

sigmoid activation, binary cross-entropy loss

$$\hat{y} = \sigma(w_1 x_1 + w_2 x_2)$$

$$\ell(\mathbf{x}, y) = -y \log \hat{y} - (1-y)\log(1-\hat{y})$$



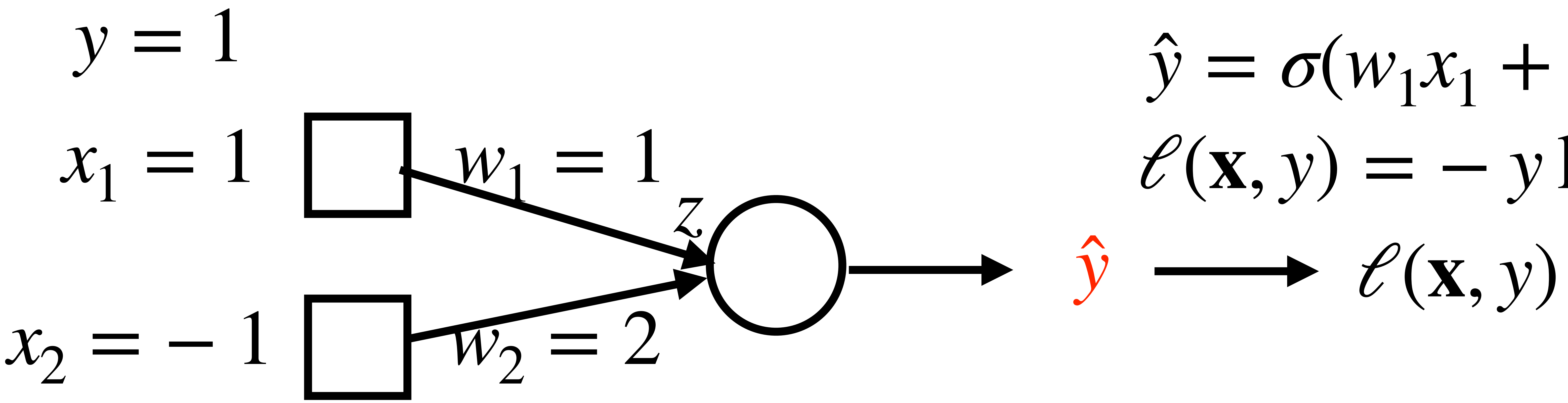

- Forward:

$$z = w_1 x_1 + w_2 x_2 = 1 \times 1 + (-1) \times 2 = -1$$

$$\hat{y} = \sigma(z) = \sigma(-1)$$

$$\ell(\mathbf{x}, y) = -y \log \hat{y} - (1-y)\log(1-\hat{y}) = -\log \hat{y} = -\log \sigma(-1)$$

# Example: Backpropagation




sigmoid activation, binary cross-entropy loss

$$\hat{y} = \sigma(w_1 x_1 + w_2 x_2)$$

$$\ell(\mathbf{x}, y) = -y \log \hat{y} - (1 - y)\log(1 - \hat{y})$$

- Forward:

  $z = -1, \hat{y} = \sigma(-1), \ell(\mathbf{x}, y) = -\log \sigma(-1)$

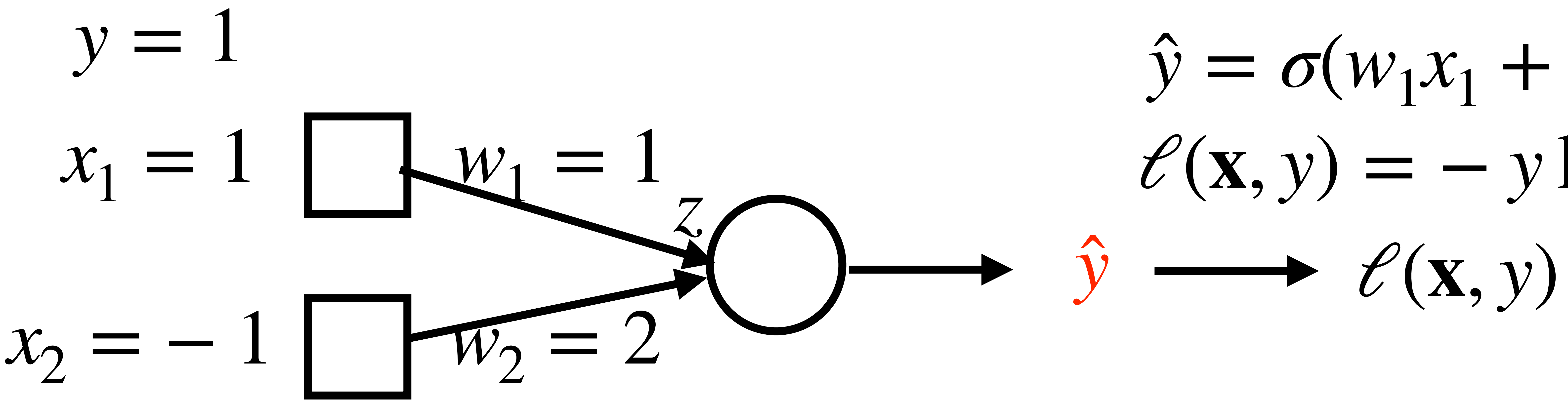
- Backward:

# Example: Backpropagation

sigmoid activation, binary cross-entropy loss

$$\hat{y} = \sigma(w_1 x_1 + w_2 x_2)$$

$$\ell(\mathbf{x}, y) = -y \log \hat{y} - (1-y)\log(1-\hat{y})$$




- Forward:

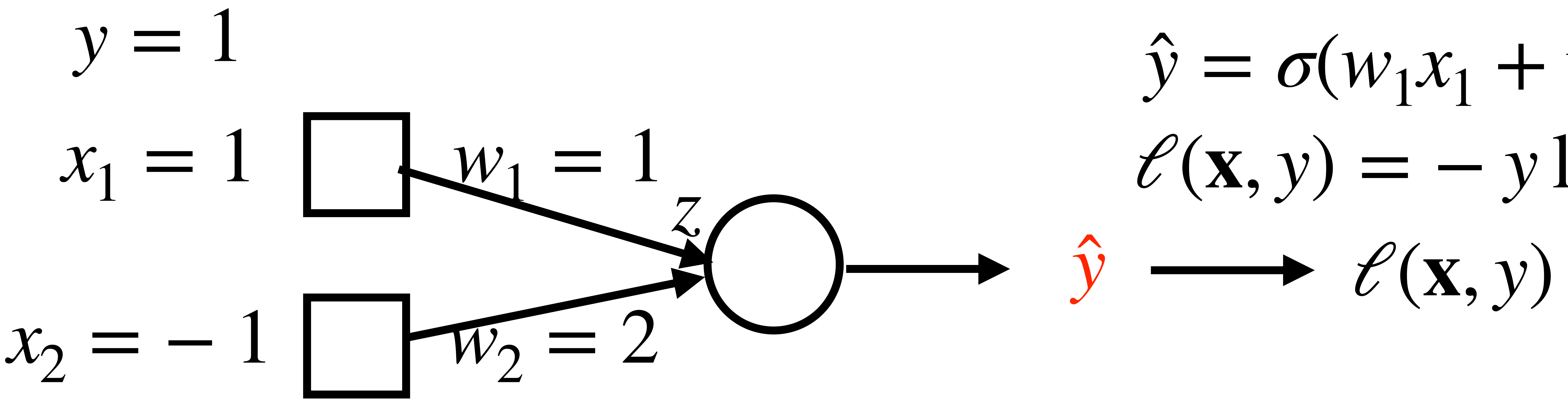
$$z = -1, \hat{y} = \sigma(-1), \ell(\mathbf{x}, y) = -\log \sigma(-1)$$

- Backward:

$$\frac{\partial \ell}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}}(-y \log \hat{y} - (1-y)\log(1-\hat{y}))$$

$$= \frac{-y}{\hat{y}} + \frac{1-y}{1-\hat{y}} = \frac{\hat{y}-y}{\hat{y}(1-\hat{y})} = \frac{\sigma(-1)-1}{\sigma(-1)(1-\sigma(-1))}$$

# Example: Backpropagation




sigmoid activation, binary cross-entropy loss

$$\hat{y} = \sigma(w_1 x_1 + w_2 x_2)$$

$$\ell(\mathbf{x}, y) = -y \log \hat{y} - (1 - y)\log(1 - \hat{y})$$

Partial derivative: view all the other variables as constants, and compute the gradient

- Forward:

$$z = -1, \hat{y} = \sigma(-1), \ell(\mathbf{x}, y) = -\log \sigma(-1)$$

- Backward:

$$\frac{\partial \ell}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}}(-y \log \hat{y} - (1 - y)\log(1 - \hat{y}))$$

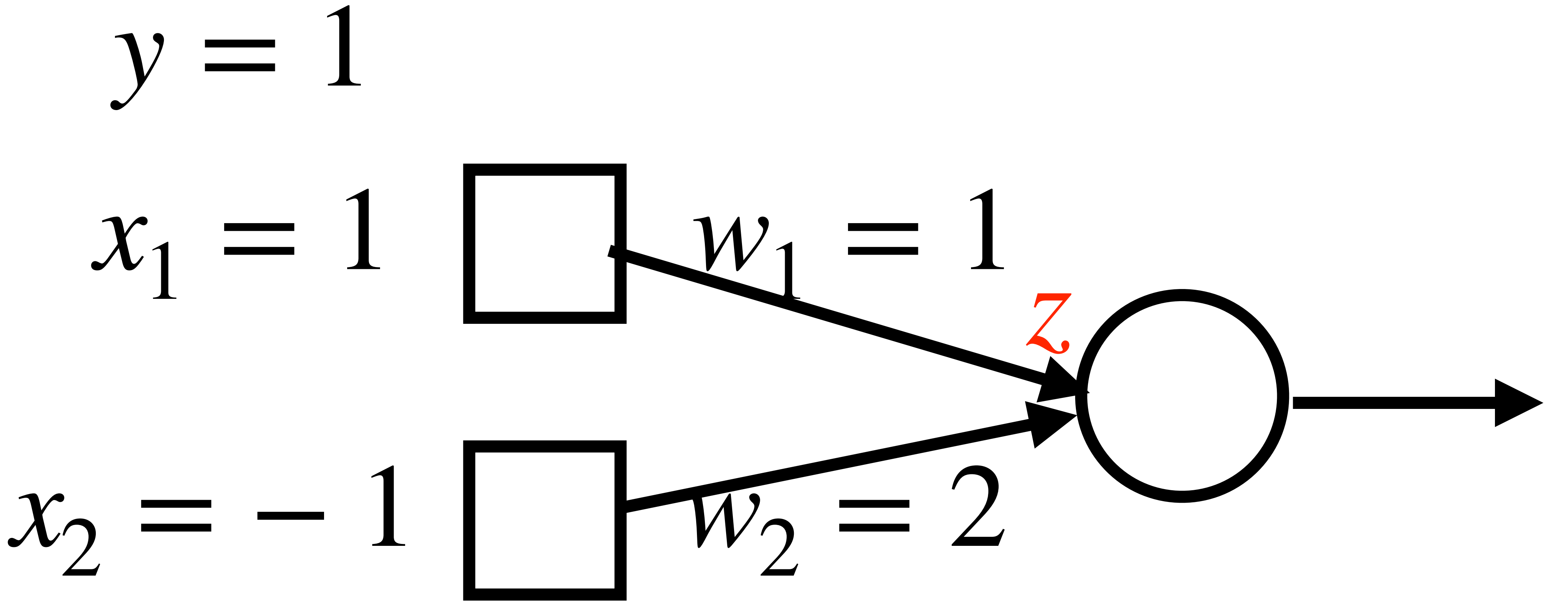
$$= \frac{-y}{\hat{y}} + \frac{1 - y}{1 - \hat{y}} = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})} = \frac{\sigma(-1) - 1}{\sigma(-1)(1 - \sigma(-1))}$$

# Example: Backpropagation

sigmoid activation, binary cross-entropy loss

$$\hat{y} = \sigma(w_1 x_1 + w_2 x_2)$$

$$\ell(\mathbf{x}, y) = -y \log \hat{y} - (1-y)\log(1-\hat{y})$$



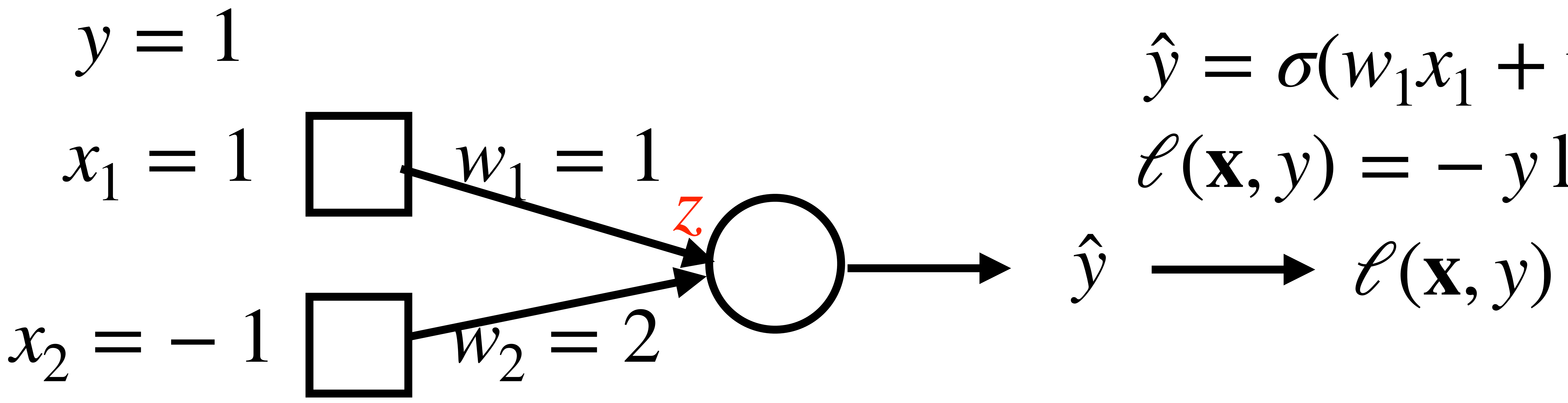

$$\hat{y} \longrightarrow \ell(\mathbf{x}, y)$$

- Forward:

$$z = -1, \hat{y} = \sigma(-1), \ell(\mathbf{x}, y) = -\log \sigma(-1)$$

- Backward: $$\frac{\partial \ell}{\partial \hat{y}} = \frac{\sigma(-1) - 1}{\sigma(-1)(1 - \sigma(-1))}$$

# Example: Backpropagation




sigmoid activation, binary cross-entropy loss

$$\hat{y} = \sigma(w_1 x_1 + w_2 x_2)$$

$$\ell(\mathbf{x}, y) = -y \log \hat{y} - (1 - y)\log(1 - \hat{y})$$

- Forward:

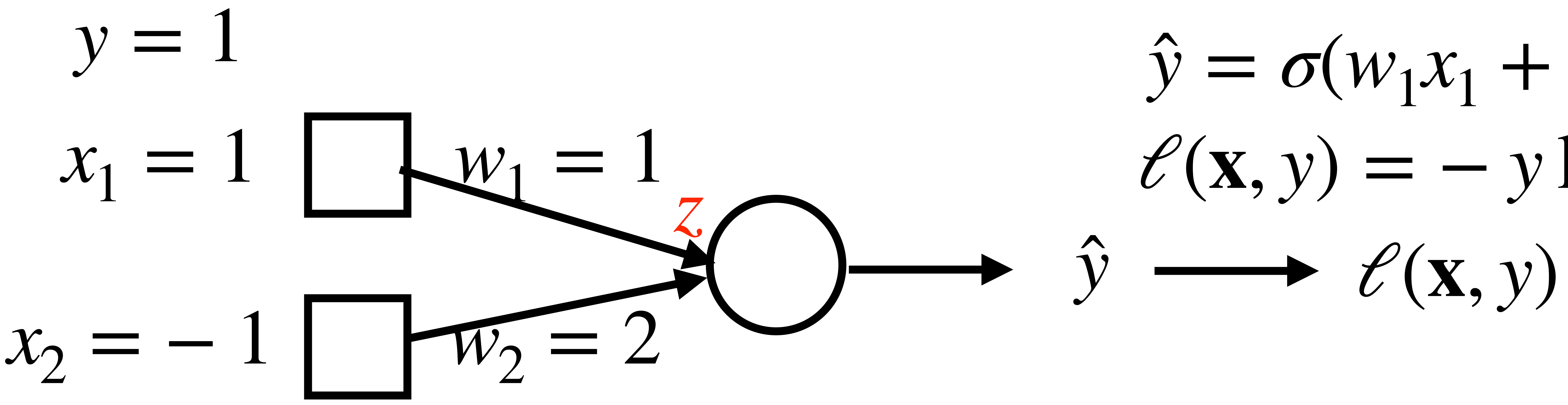
$$z = -1, \hat{y} = \sigma(-1), \ell(\mathbf{x}, y) = -\log \sigma(-1)$$

- Backward:

$$\frac{\partial \ell}{\partial \hat{y}} = \frac{\sigma(-1) - 1}{\sigma(-1)(1 - \sigma(-1))}$$

$$\frac{\partial \ell}{\partial z} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z}$$

# Example: Backpropagation




sigmoid activation, binary cross-entropy loss

$$\hat{y} = \sigma(w_1 x_1 + w_2 x_2)$$

$$\ell(\mathbf{x}, y) = -y \log \hat{y} - (1-y)\log(1-\hat{y})$$

- Forward:

$$z = -1, \hat{y} = \sigma(-1), \ell(\mathbf{x}, y) = -\log \sigma(-1)$$

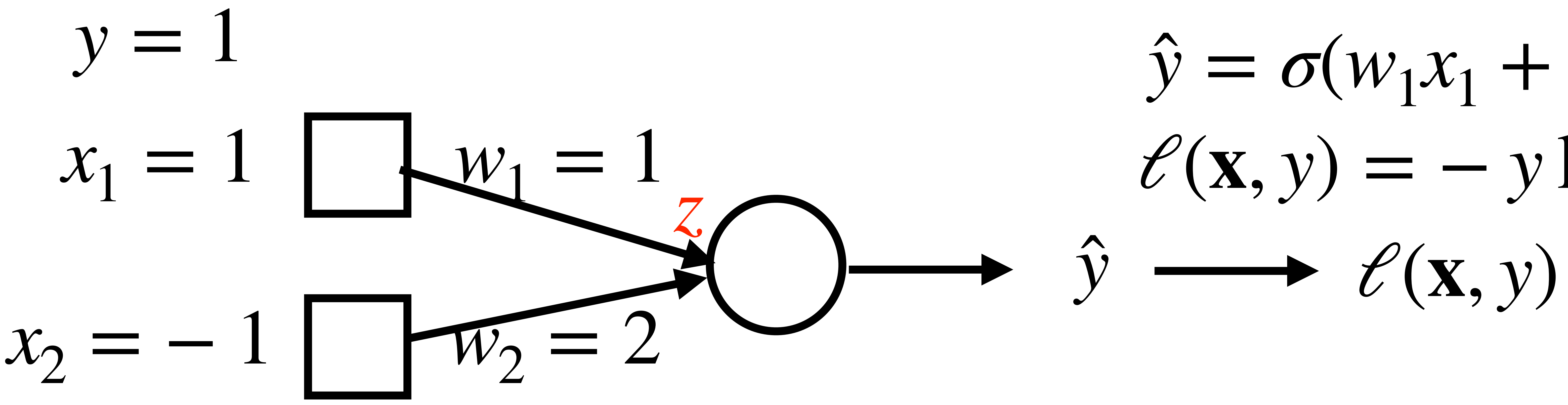
- Backward:

$$\frac{\partial \ell}{\partial \hat{y}} = \frac{\sigma(-1) - 1}{\sigma(-1)(1 - \sigma(-1))}$$

$$\frac{\partial \ell}{\partial z} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z}$$

$$\frac{\partial \hat{y}}{\partial z} = \frac{\partial}{\partial z}\sigma(z) = \sigma(z)(1 - \sigma(z)) = \hat{y}(1 - \hat{y}) = \sigma(-1)(1 - \sigma(-1))$$

# Example: Backpropagation




sigmoid activation, binary cross-entropy loss

$\hat{y} = \sigma(w_1 x_1 + w_2 x_2)$

$\ell(\mathbf{x}, y) = -y \log \hat{y} - (1-y)\log(1-\hat{y})$

- Forward:

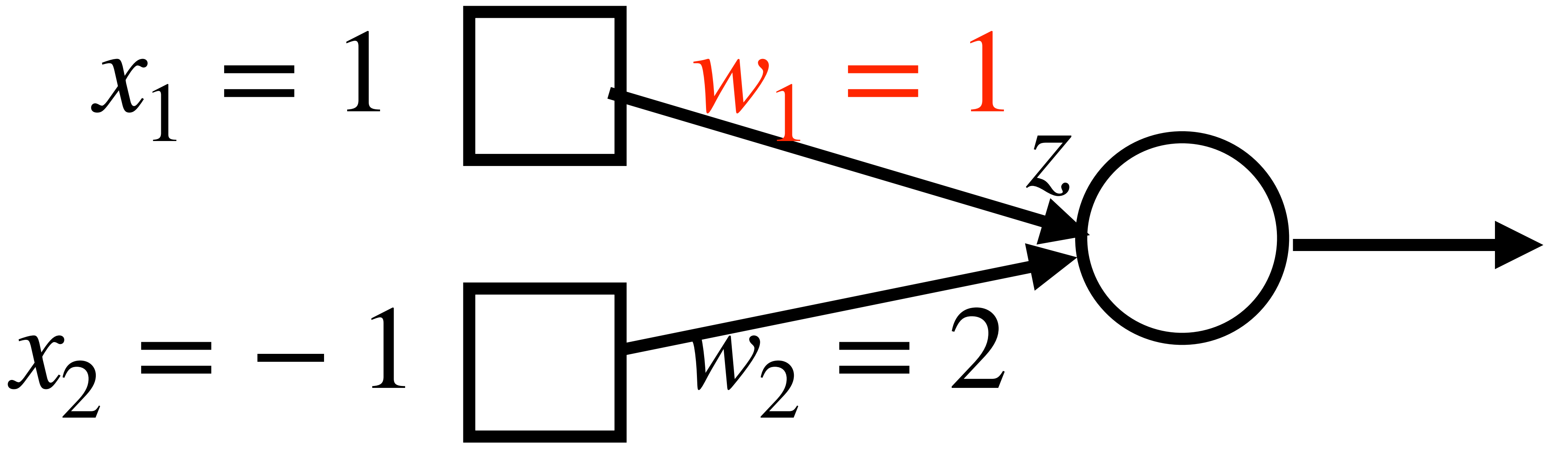
  $z = -1, \hat{y} = \sigma(-1), \ell(\mathbf{x}, y) = -\log \sigma(-1)$

- Backward: $\dfrac{\partial \ell}{\partial z} = \dfrac{\partial \ell}{\partial \hat{y}} \dfrac{\partial \hat{y}}{\partial z} = \sigma(-1) - 1$

# Example: Backpropagation

sigmoid activation, binary cross-entropy loss

$y = 1$




$\hat{y} \longrightarrow \ell(\mathbf{x}, y)$

$$\hat{y} = \sigma(w_1 x_1 + w_2 x_2)$$

$$\ell(\mathbf{x}, y) = -y \log \hat{y} - (1-y)\log(1-\hat{y})$$

- Forward:

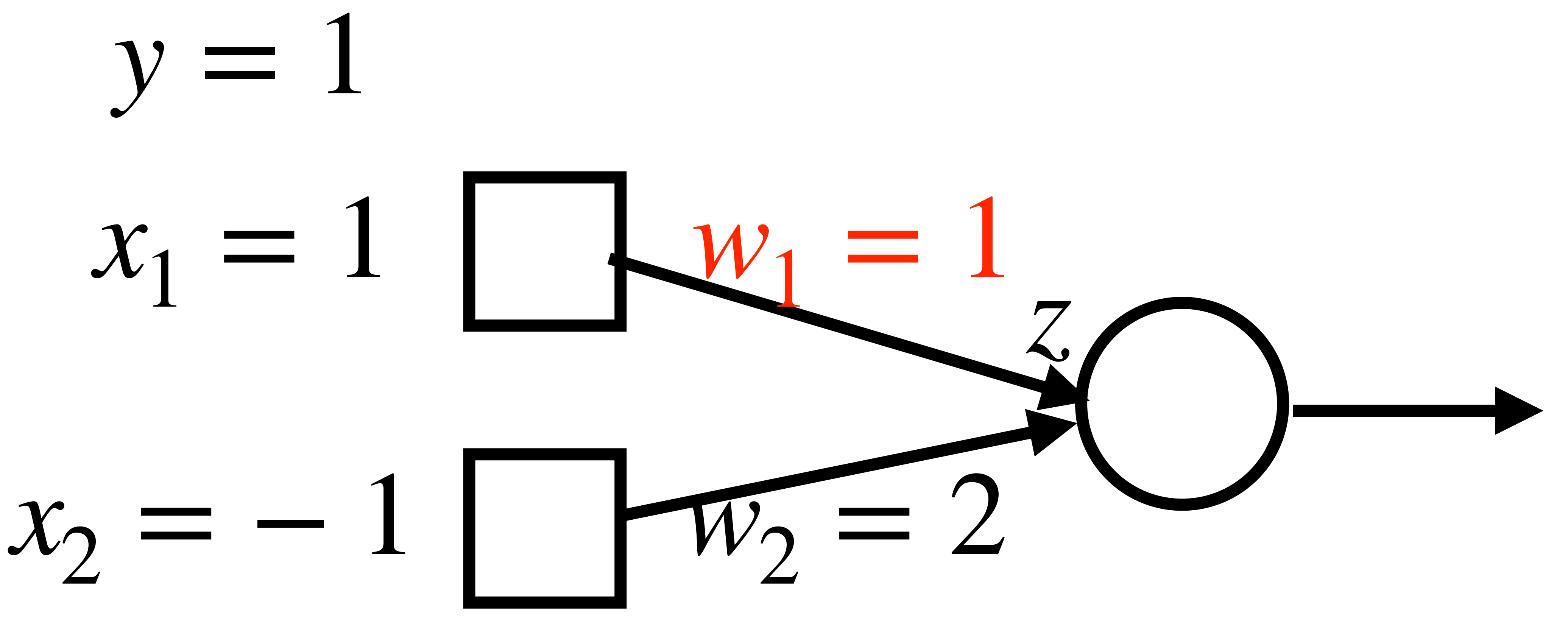
$$z = -1, \hat{y} = \sigma(-1), \ell(\mathbf{x}, y) = -\log \sigma(-1)$$

- Backward:

$$\frac{\partial \ell}{\partial z} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} = \sigma(-1) - 1$$

$$\frac{\partial \ell}{\partial w_1} = \frac{\partial \ell}{\partial z} \frac{\partial z}{\partial w_1}$$

# Example: Backpropagation




$\hat{y} \longrightarrow \ell(\mathbf{x}, y)$

sigmoid activation, binary cross-entropy loss

$$\hat{y} = \sigma(w_1 x_1 + w_2 x_2)$$

$$\ell(\mathbf{x}, y) = -y \log \hat{y} - (1 - y)\log(1 - \hat{y})$$

- Forward:

$$z = -1, \hat{y} = \sigma(-1), \ell(\mathbf{x}, y) = -\log \sigma(-1)$$

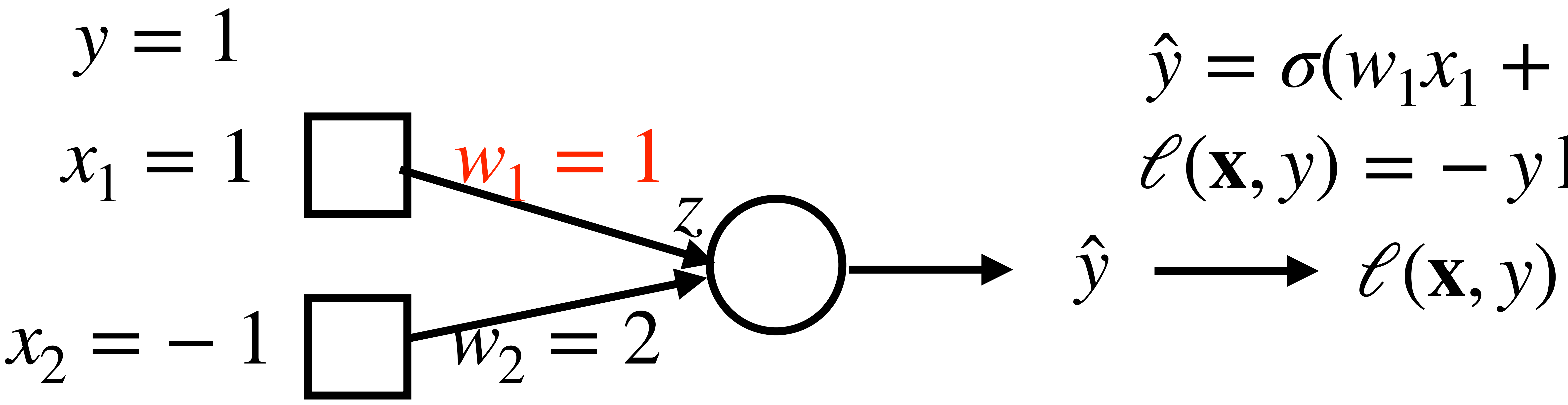
- Backward:

$$\frac{\partial \ell}{\partial z} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} = \sigma(-1) - 1$$

$$\frac{\partial \ell}{\partial w_1} = \frac{\partial \ell}{\partial z} \frac{\partial z}{\partial w_1}$$

$$\frac{\partial z}{\partial w_1} = \frac{\partial}{\partial w_1}(w_1 x_1 + w_2 x_2) = x_1 = 1$$

# Example: Backpropagation




sigmoid activation, binary cross-entropy loss

$$\hat{y} = \sigma(w_1 x_1 + w_2 x_2)$$

$$\ell(\mathbf{x}, y) = -y \log \hat{y} - (1 - y)\log(1 - \hat{y})$$

- Forward:

$$z = -1, \hat{y} = \sigma(-1), \ell(\mathbf{x}, y) = -\log \sigma(-1)$$

- Backward:

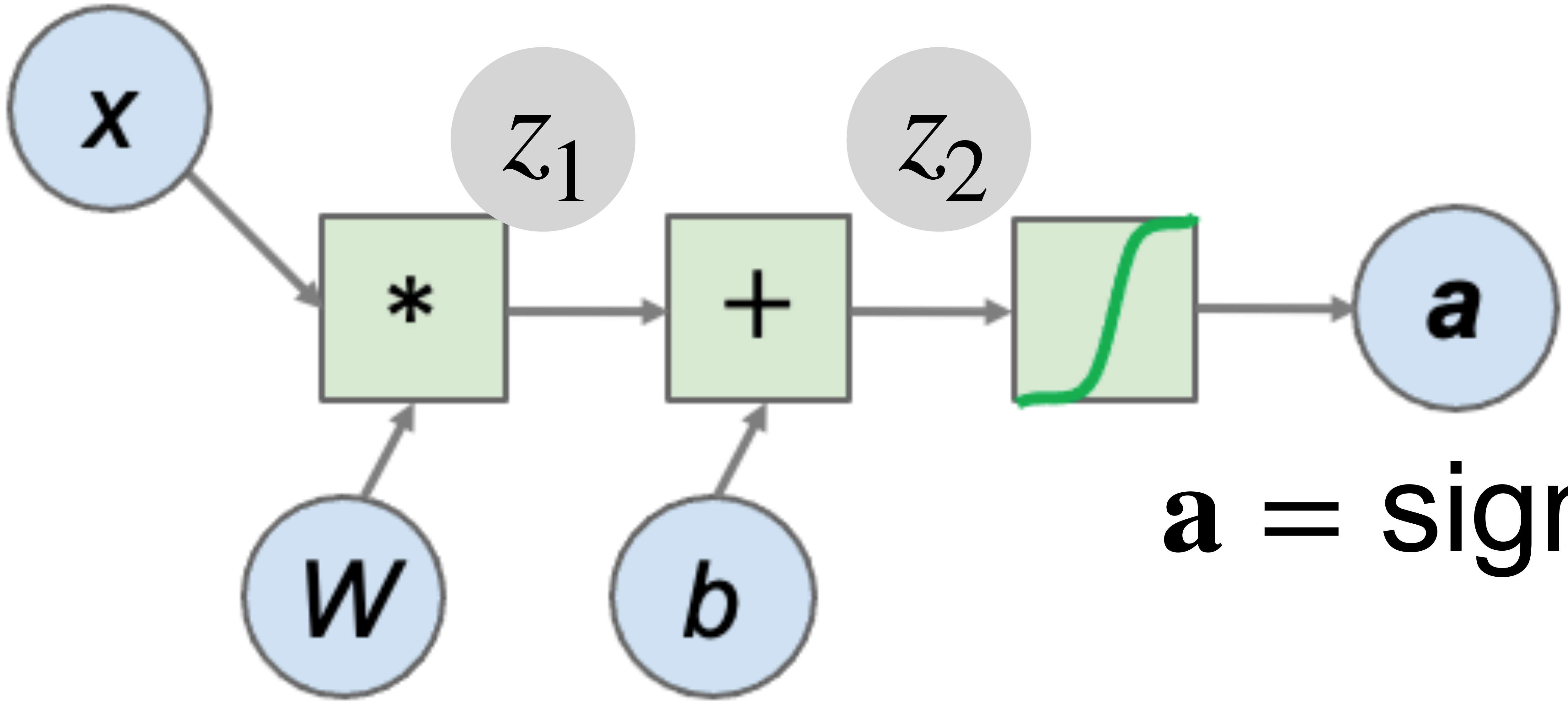
$$\frac{\partial \ell}{\partial w_1} = \frac{\partial \ell}{\partial z}\frac{\partial z}{\partial w_1} = (\sigma(-1) - 1) \times 1 = \sigma(-1) - 1$$

# Calculate Gradient: Backpropagation with Chain Rule

- Define a loss function L
- Gradient to a variable =

gradient on the top x gradient from the current operation

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \boldsymbol{z_1}} \frac{\partial \boldsymbol{z_1}}{\partial W}$$

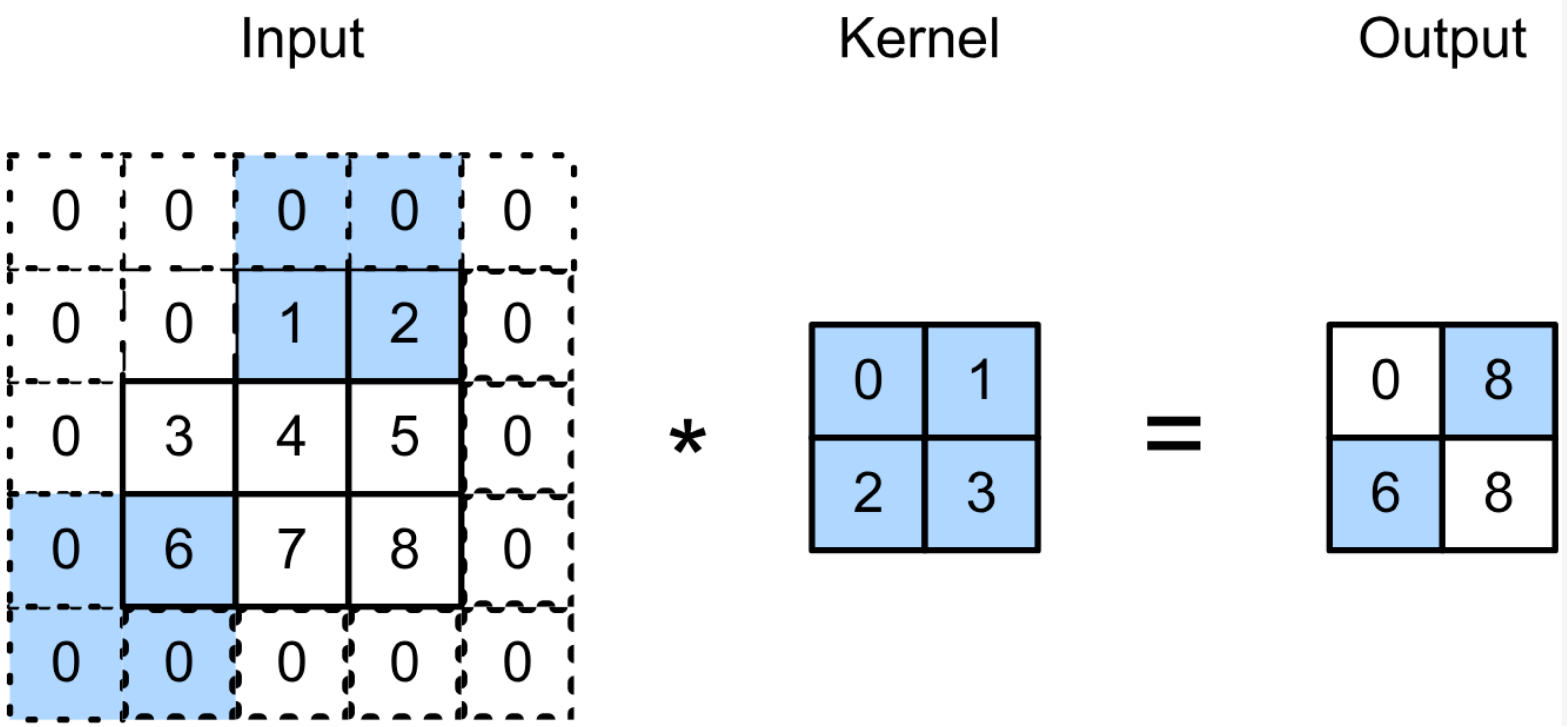



$$\mathbf{a} = \text{sigmoid}(\mathbf{Wx} + \mathbf{b})$$

# Convolutional Neural Networks

# 2-D Convolution Layer with Stride and Padding

- Stride is the #rows/#columns per slide
- Padding adds rows/columns around input
- Output shape



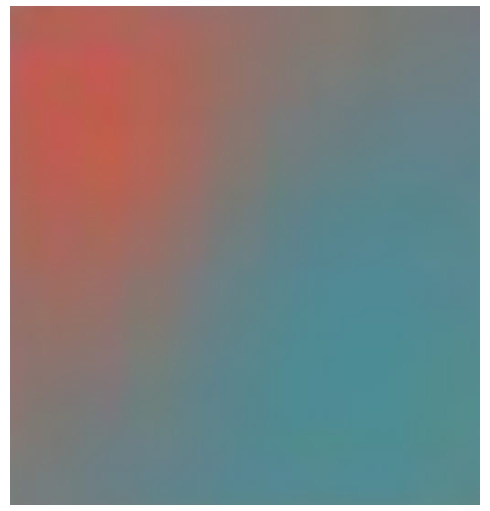

**Kernel/filter size**

$$\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor$$

**Input size** **Pad** **Stride**

# Multiple Input Channels

- Input and kernel can be 3D, e.g., an RGB image have 3 channels
- Also call each 3D kernel a "**filter**", which produce only **one** output channel (due to summation over channels)



*

One filter
(3 channels)

RGB (3 input channels)

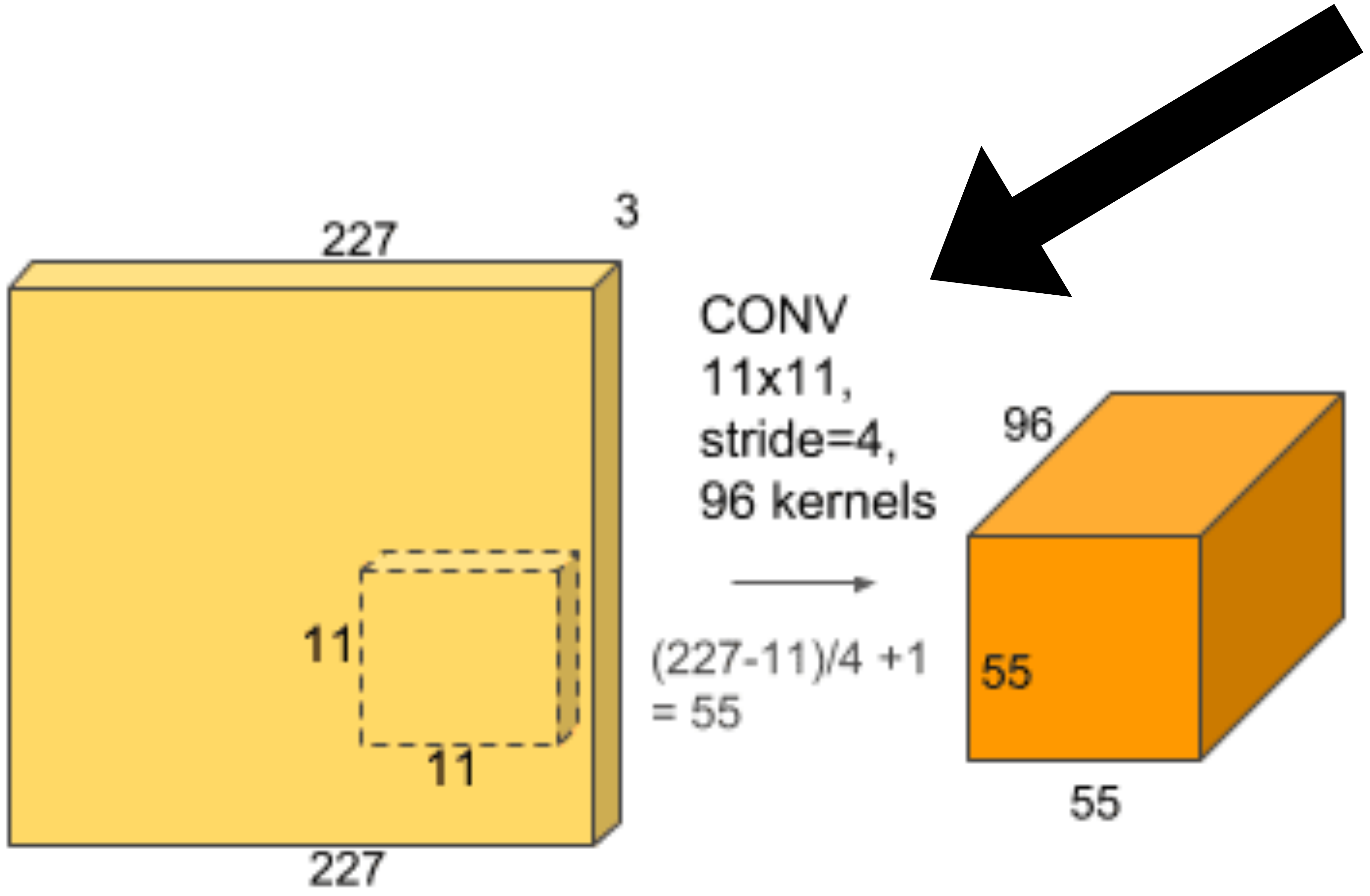
# Multiple Filters Lead to Multiple Output Channels

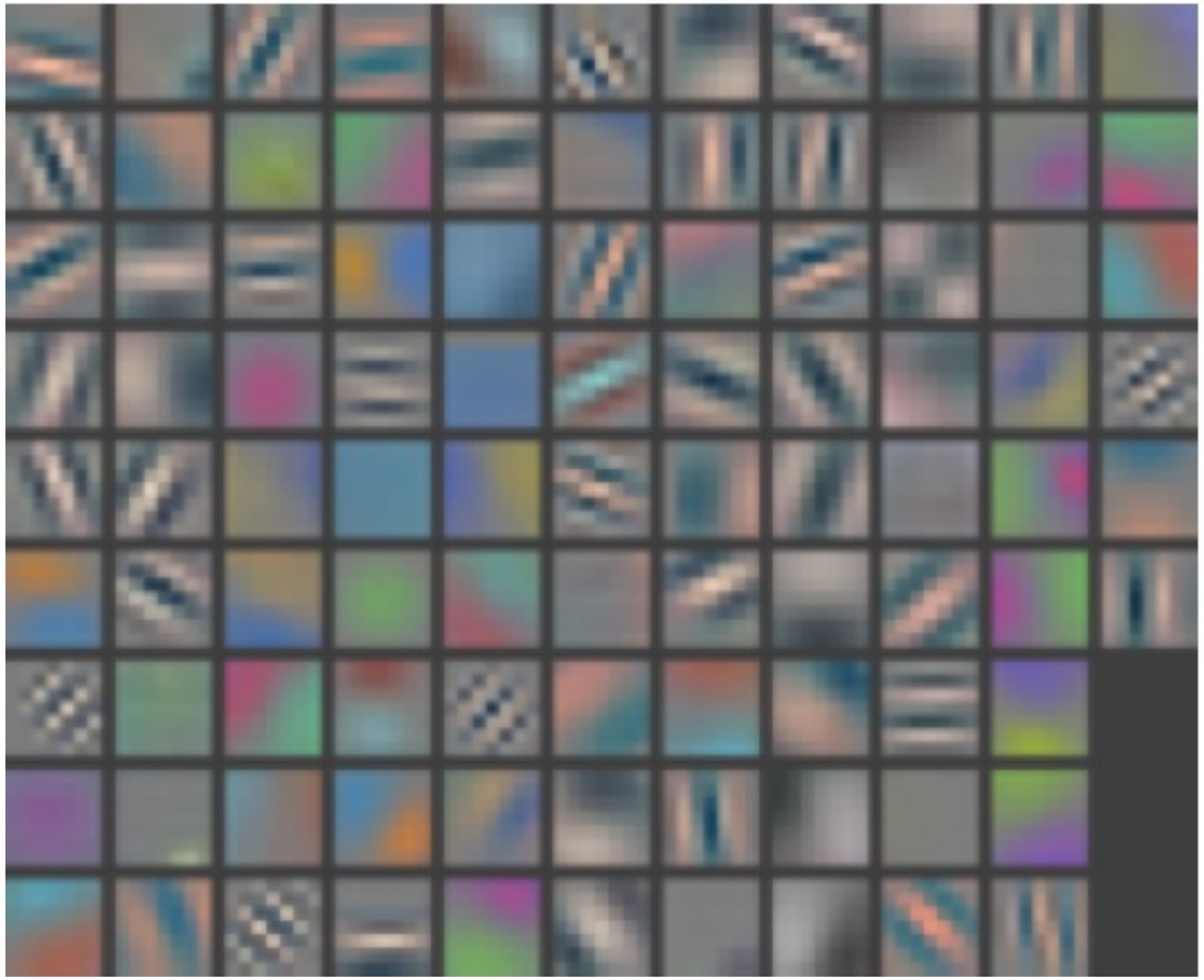
- Apply multiple filters on the input
- Each filter may learn different features about the input
- Each filter (3D kernel) produces one output channel


RGB (3 input channels)

# Conv1 Filters in AlexNet

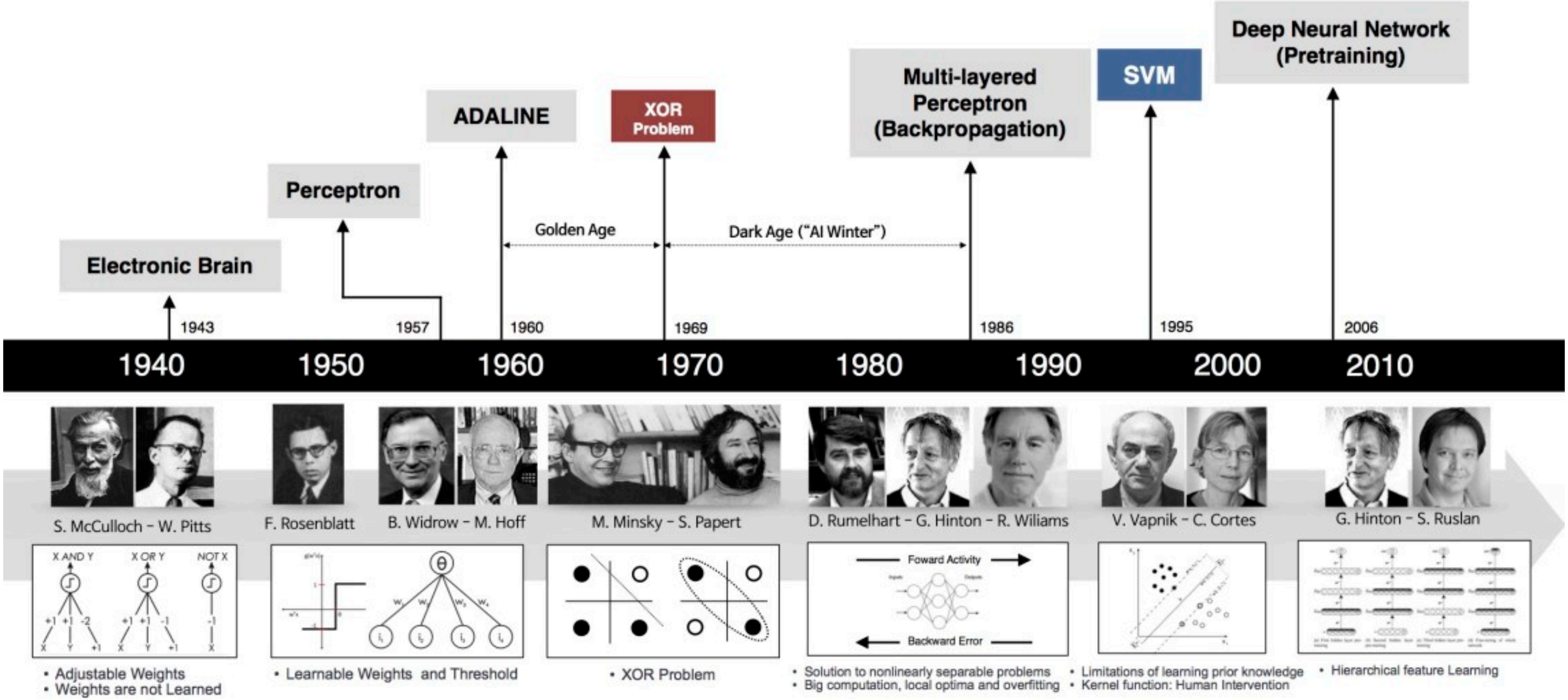
- 96 filters (each of size 11x11x3)
- Gabor filters



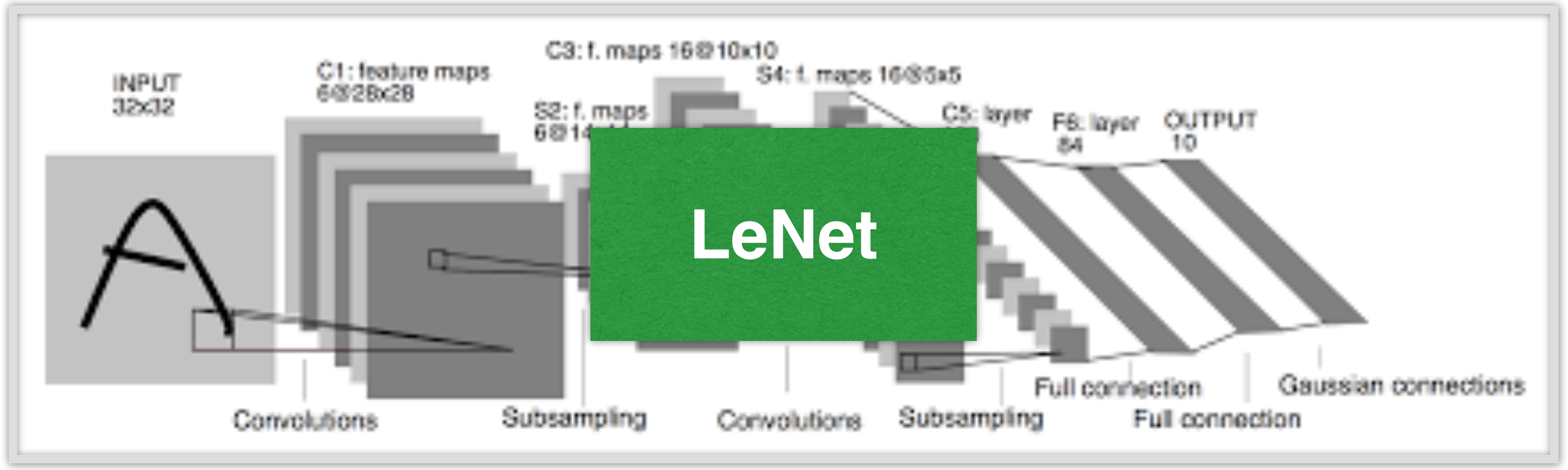



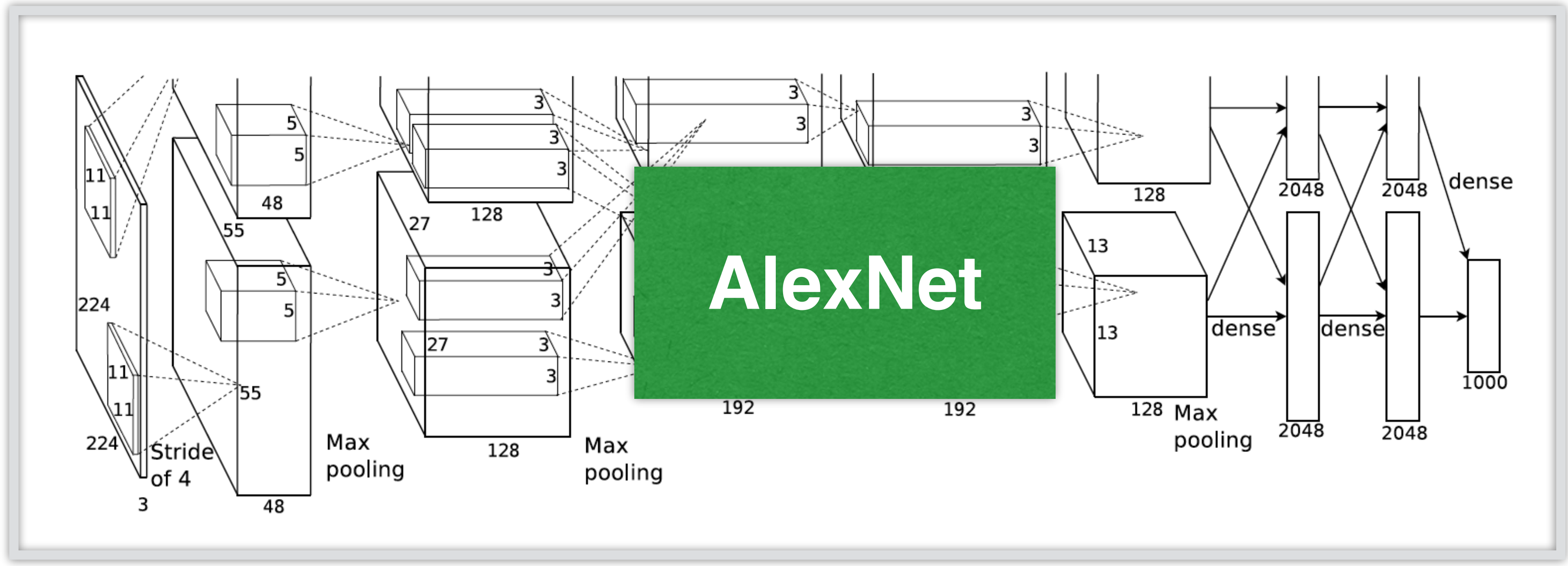
Figures from Visualizing and Understanding Convolutional Networks
by *M. Zeiler and R. Fergus*

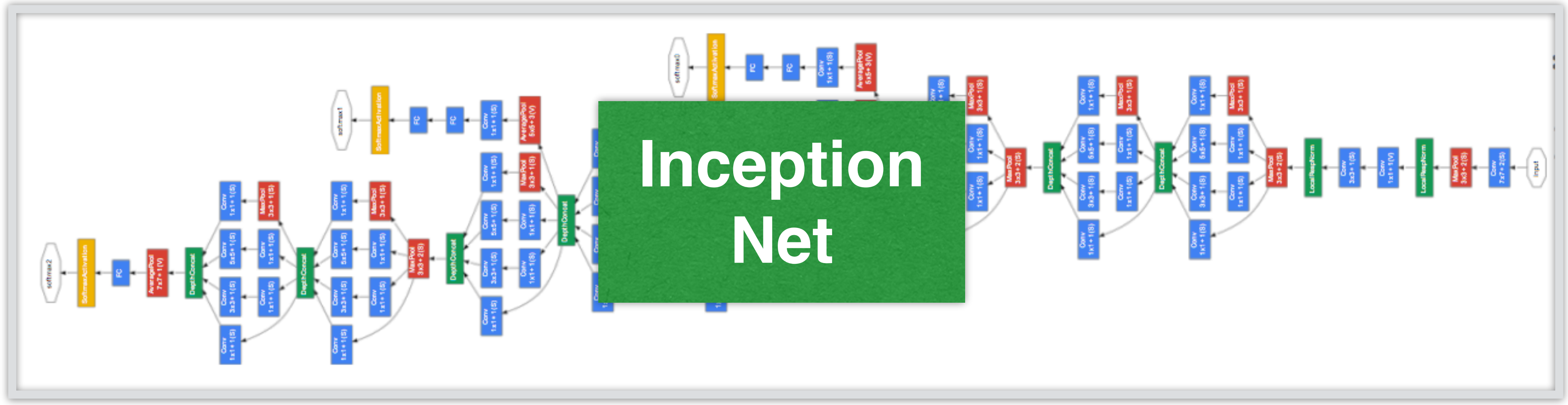
# Brief history of neural networks

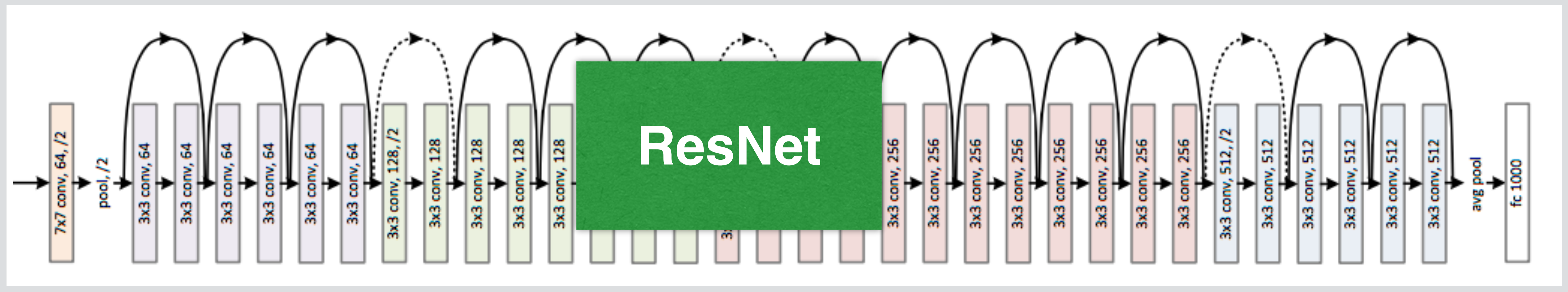


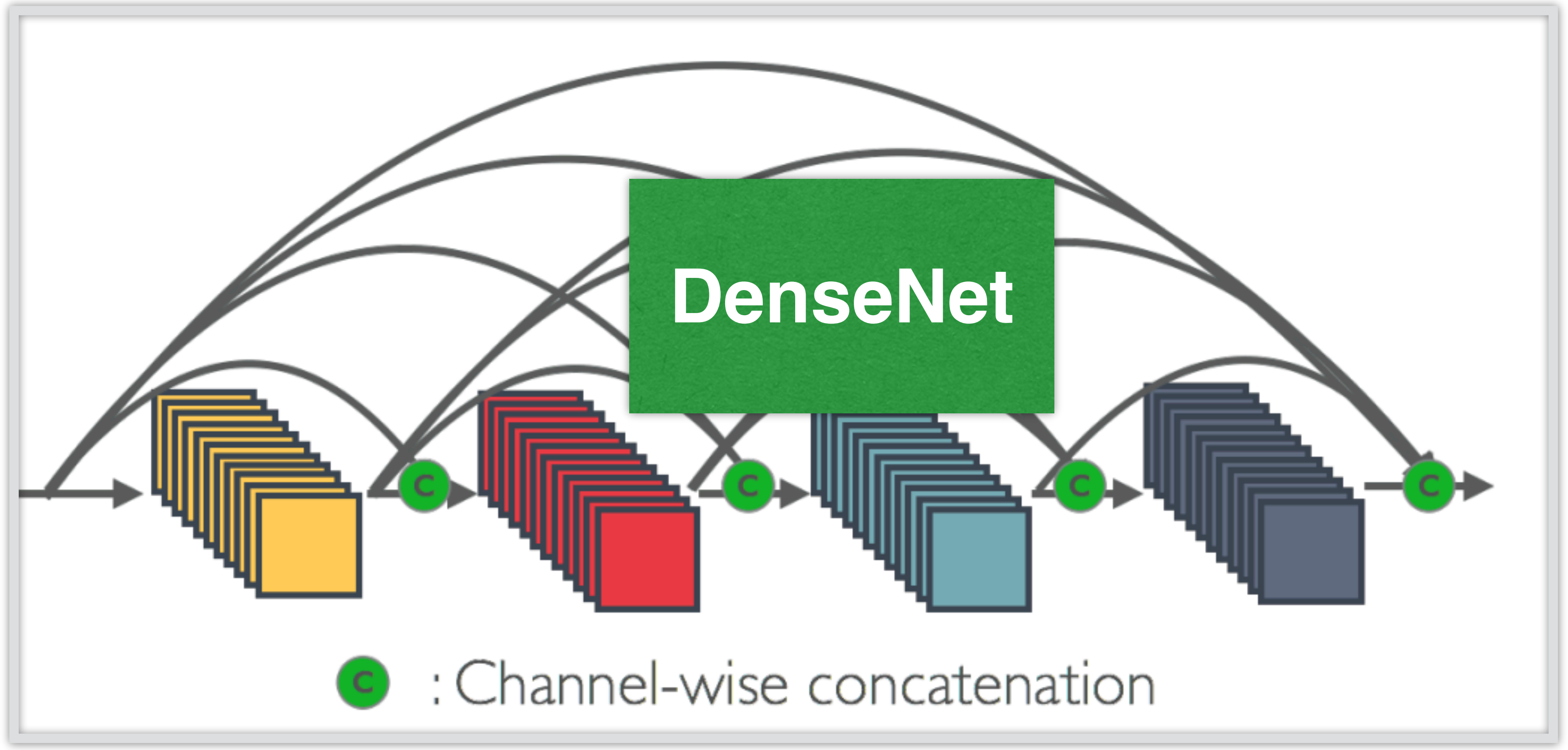
# Evolution of modern deep neural net architectures



# What we've learned today…

- Modeling a single neuron
  - Perceptron
  - Limited power of a single neuron
- Multi-layer perceptron
- Training of neural networks
  - Loss function (cross entropy)
  - Backpropagation and SGD
- Convolutional neural networks
  - Convolution, pooling, stride, padding
  - Basic architectures (LeNet etc.)
  - More advanced architectures (AlexNet, ResNet etc)

# Thank you!

Some of the slides in these lectures have been adapted from materials developed by Alex Smola and Mu Li:
https://courses.d2l.ai/berkeley-stat-157/index.html