Image: Jose-Luis Olivares

AIN311
Fundamentals of Machine Learning

Lecture 11:
Multi-layer Perceptron
Forward Pass

HACETTEPE UNIVERSITY COMPUTER VISION LAB

Erkut Erdem // Hacettepe University // Fall 2021

Last time... **Linear Discriminant Function**

- Linear discriminant function for a vector $\mathbf{x}$

$$y(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$$

where $\mathbf{w}$ is called weight vector, and w_0 is a bias.
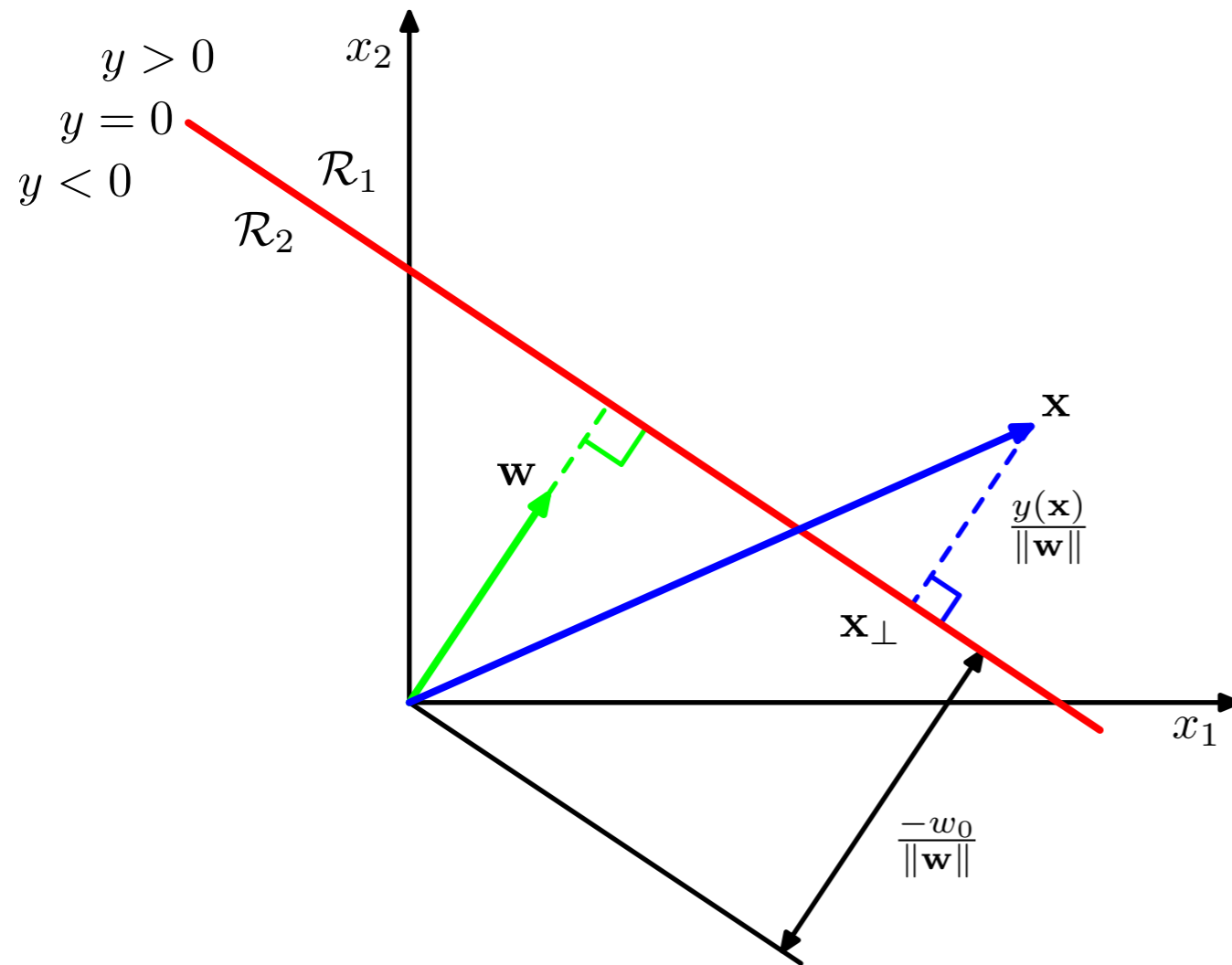
- The classification function is

$$C(\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x} + w_0)$$

where step function sign($\cdot$) is defined as

$$\text{sign}(a) = \begin{cases} +1, & a \geqslant 0 \\ -1, & a < 0 \end{cases}$$

Last time... **Properties of Linear Discriminant Functions**

- The decision surface, shown in red, is perpendicular to $\mathbf{w}$, and its displacement from the origin is controlled by the bias parameter w_0.

- The signed orthogonal distance of a general point $\mathbf{x}$ from the decision surface is given by $y(\mathbf{x})/\|\mathbf{w}\|$

- $y(\mathbf{x})$ gives a signed measure of the perpendicular distance r of the point $\mathbf{x}$ from the decision surface
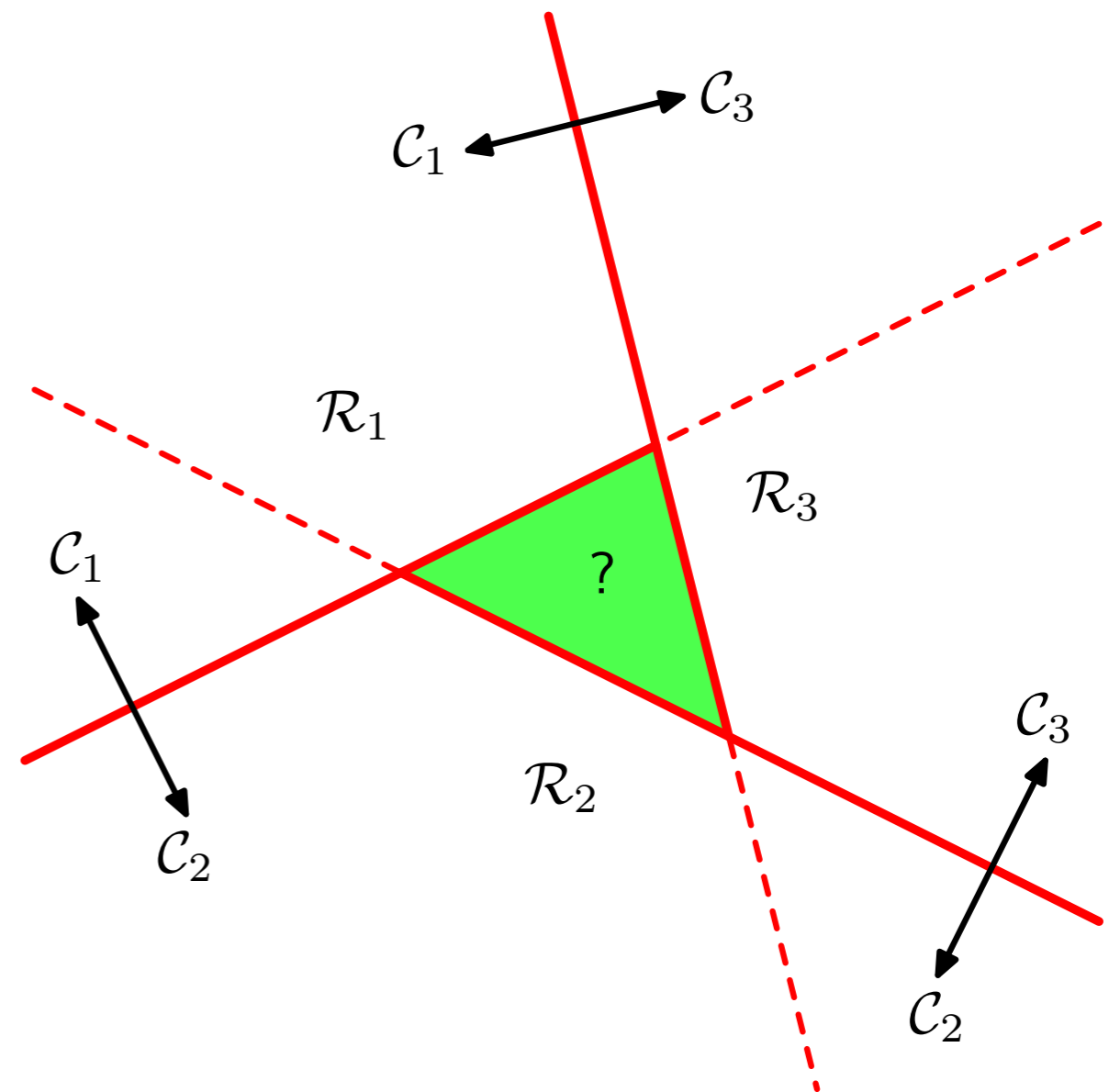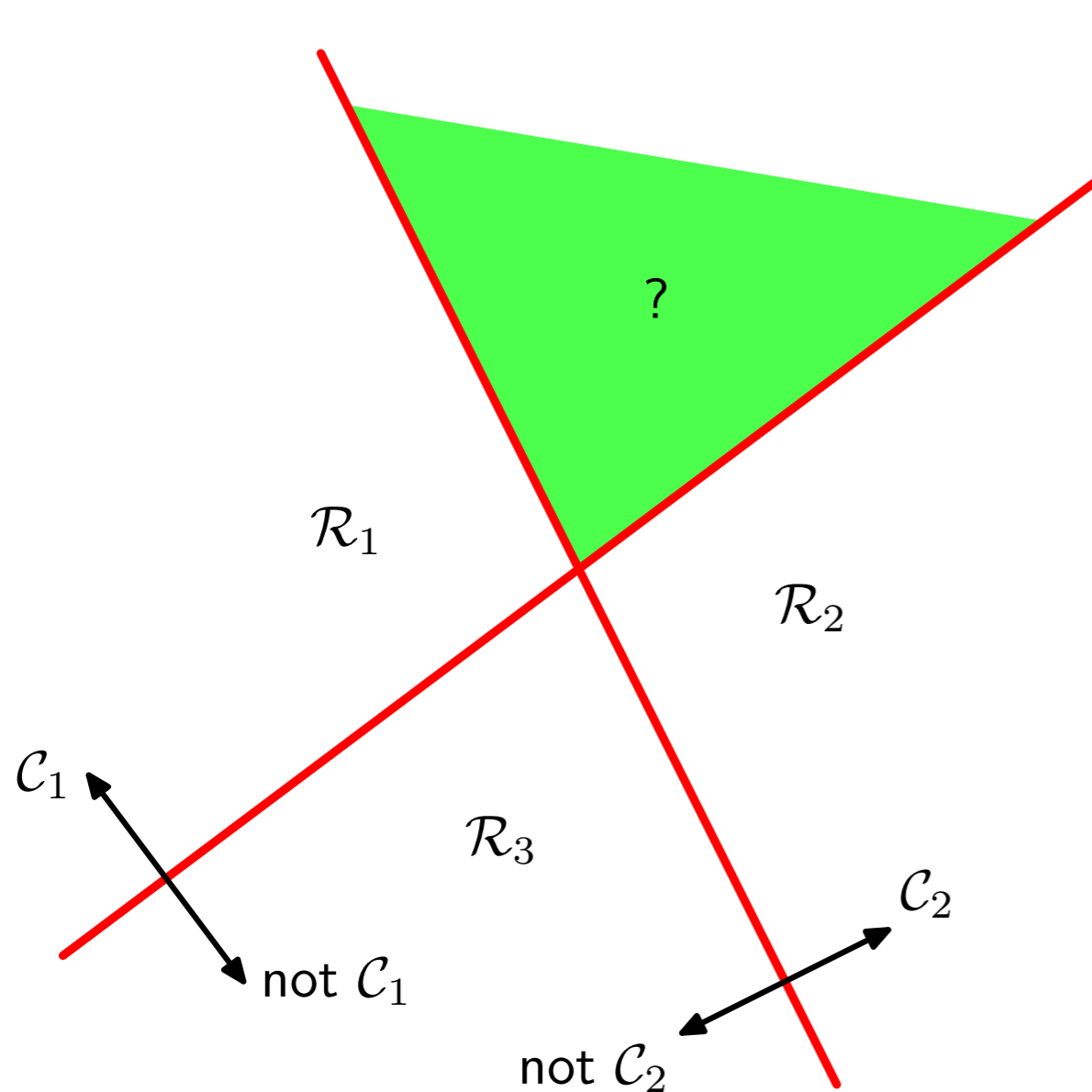
- $y(\mathbf{x}) = 0$ for $\mathbf{x}$ on the decision surface. The normal distance from the origin to the decision surface is

$$\frac{\mathbf{w}^T \mathbf{x}}{\|\mathbf{w}\|} = -\frac{w_0}{\|\mathbf{w}\|}$$

- So w_0 determines the location of the decision surface.

3

Last time… **Multiple Classes: Simple Extension**

- **One-versus-the-rest** classifier: classify C_k and samples not in C_k.
- **One-versus-one** classifier: classify every pair of classes.

Last time… **Multiple Classes: K-Class Discriminant**

- A single *K*-class discriminant comprising *K* linear functions

$$y_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + w_{k0}$$

- Decision function

$$\mathcal{C}(\mathbf{x}) = k, \text{ if } y_k(\mathbf{x}) > y_j(\mathbf{x}) \ \forall \, j \neq k$$

- The decision boundary between class C_k and C_j is given by $y_k(\mathbf{x}) = y_j(\mathbf{x})$

$$(\mathbf{w}_k - \mathbf{w}_j)^T \mathbf{x} + (w_{k0} - w_{j0}) = 0$$

Last time…**Fisher's Linear Discriminant**

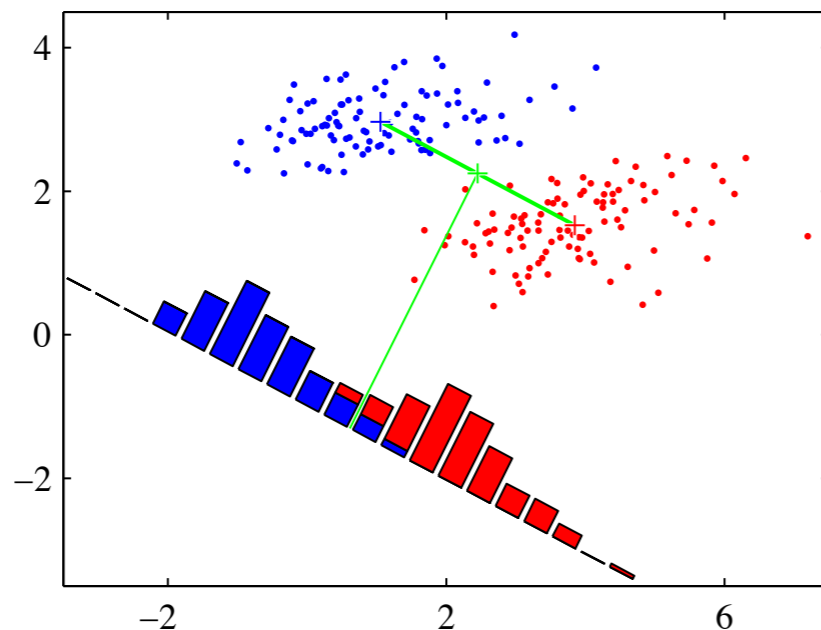- Pursue the optimal linear projection on which the two classes can be maximally separated

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

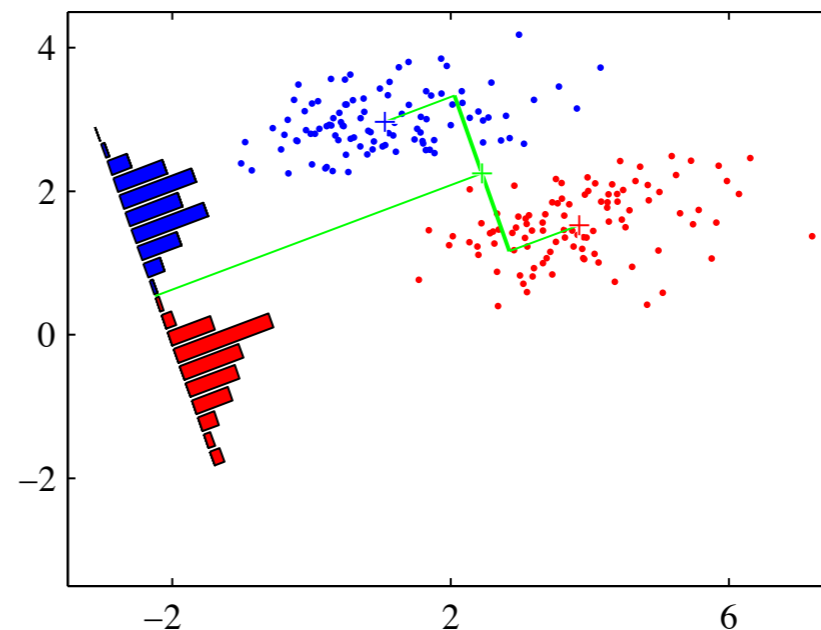- The mean vectors of the two classes

$$\mathbf{m}_1 = \frac{1}{N_1} \sum_{n \in \mathcal{C}_1} \mathbf{x}_n, \quad \mathbf{m}_2 = \frac{1}{N_2} \sum_{n \in \mathcal{C}_2} \mathbf{x}_n$$

A way to view a linear classification model is in terms of dimensionality reduction.

$$J(\mathbf{w}) = \frac{\texttt{Between-class variance}}{\texttt{Within-class variance}} = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}}$$

Difference of means

Fisher's Linear Discriminant

6

Last time… **Linear classification**

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

3072×1 10×1

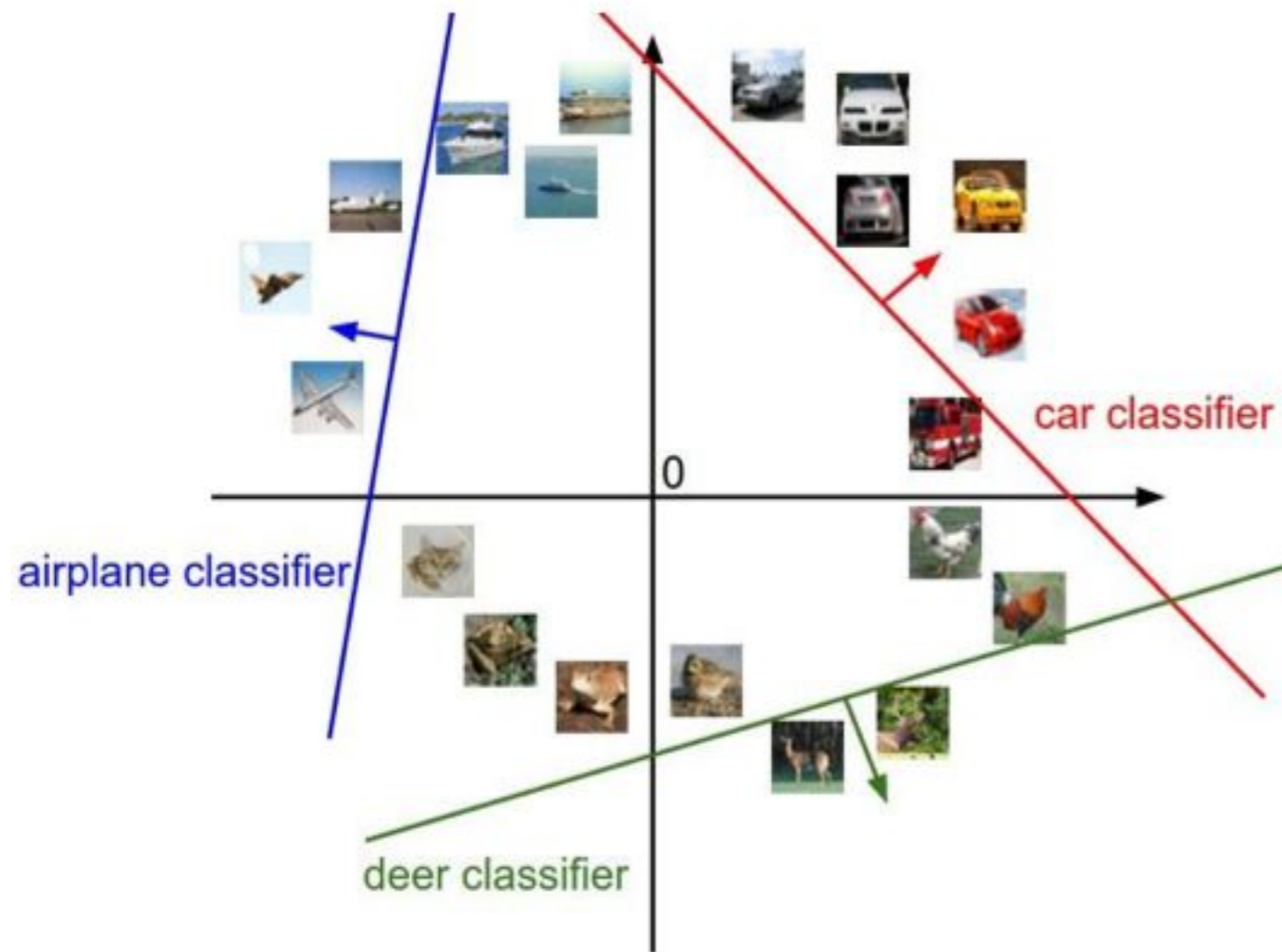10×1 10×3072

[32x32x3]
array of numbers 0...1

10 numbers, indicating class scores

parameters, or "weights"

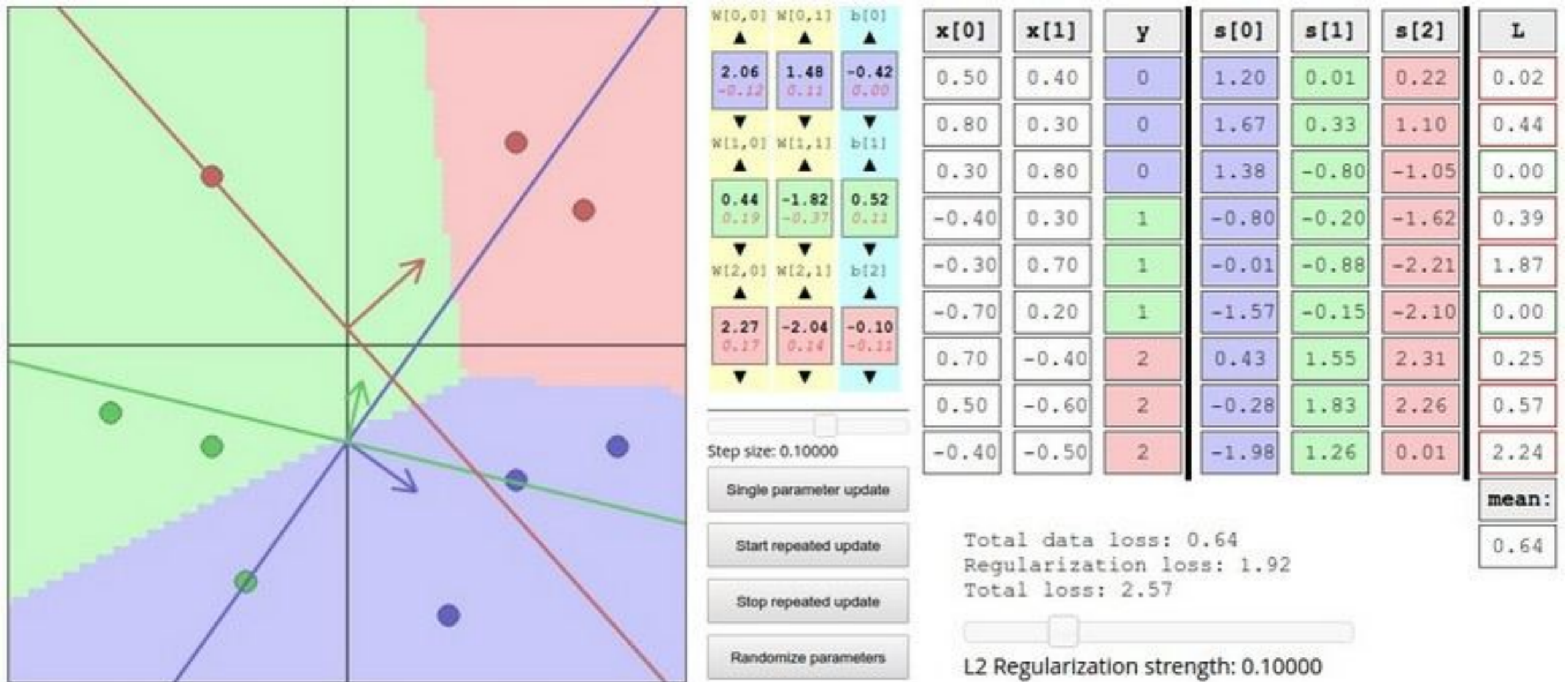# Last time... **Linear classification**

Last time… **Linear classification**

car classifier
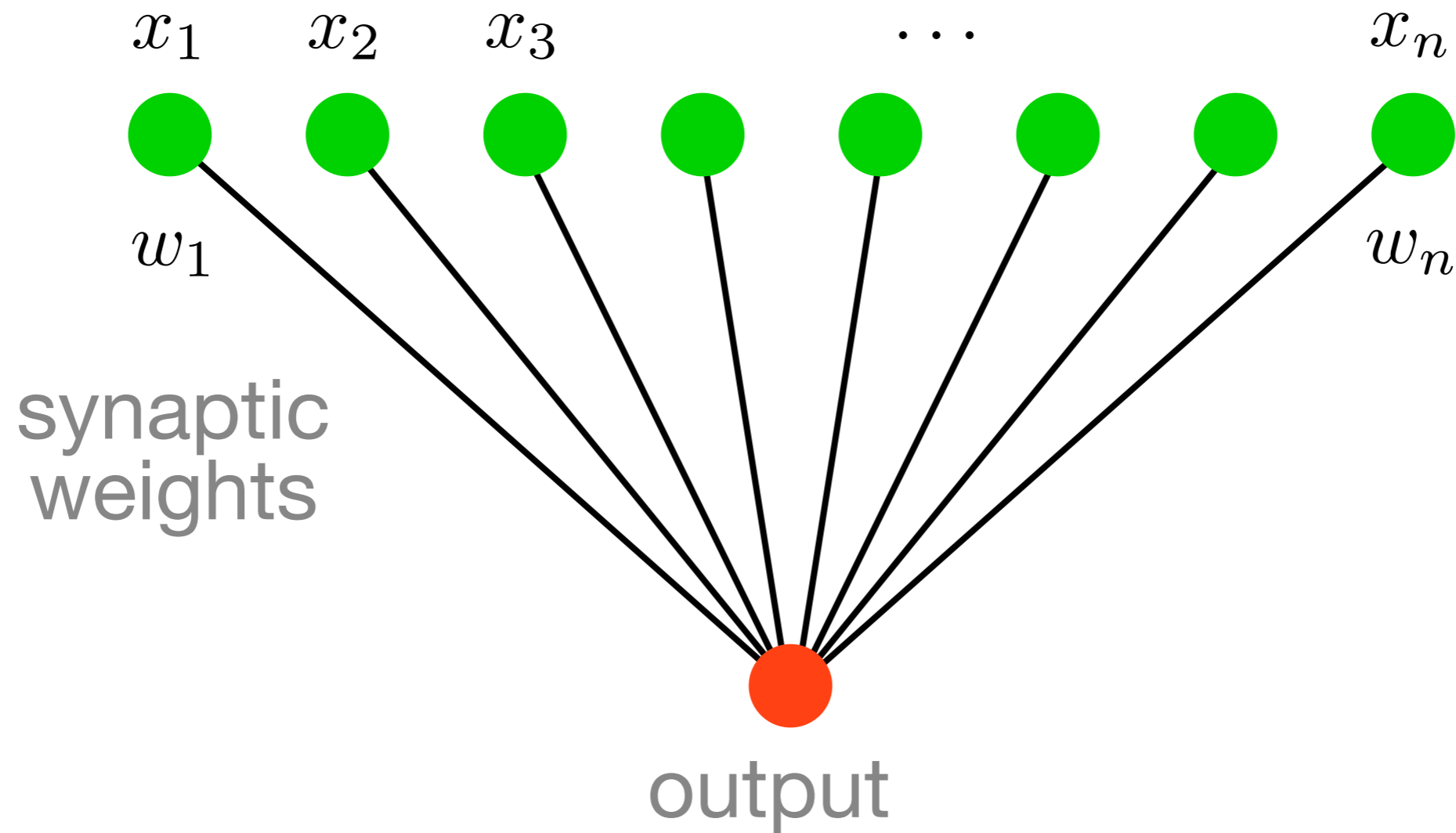
airplane classifier

deer classifier
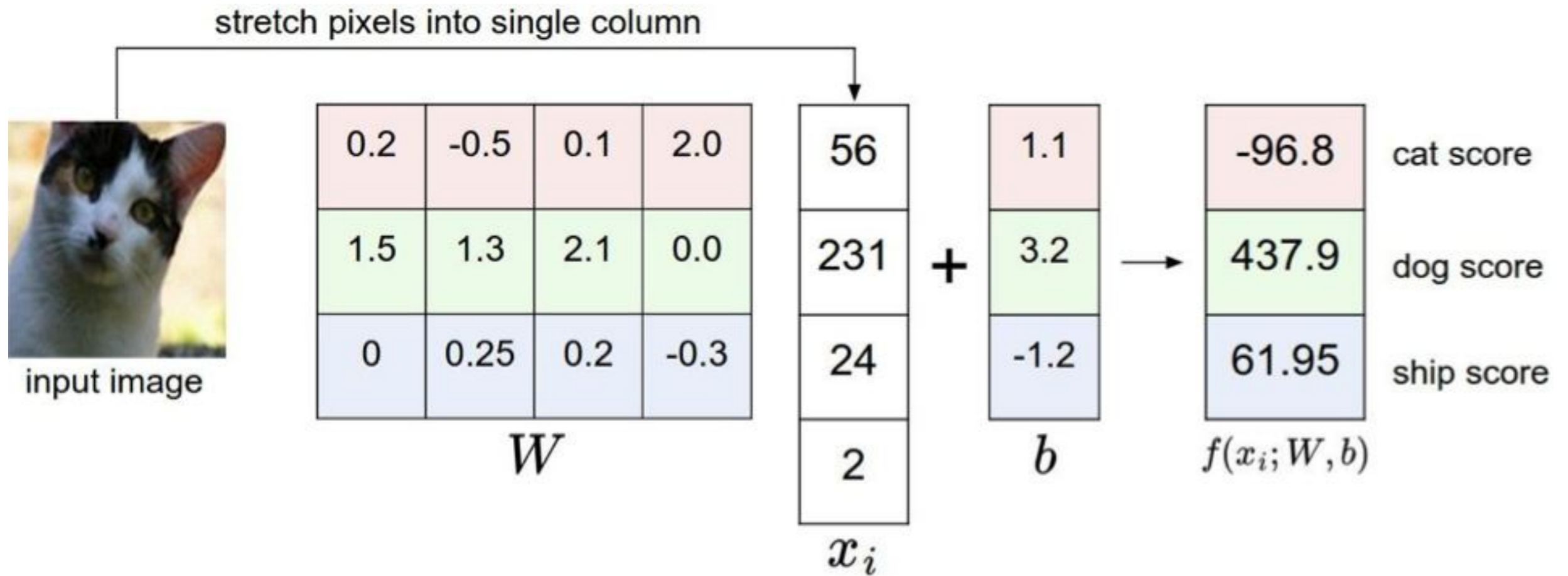
$$f(x_i, W, b) = W x_i + b$$

[32x32x3]
array of numbers 0…1
(3072 numbers total)

Interactive web demo time….

http://vision.stanford.edu/teaching/cs231n/linear-classify-demo/

10

Last time… **Perceptron**

$$x_1 \quad x_2 \quad x_3 \quad \cdots \quad x_n$$

synaptic
weights

w_1 $\qquad\qquad\qquad\qquad\qquad$ w_n

output

$$f(x) = \sum_i w_i x_i = \langle w, x \rangle$$

Last time... **Perceptron**

initialize $w = 0$ and $b = 0$
repeat
   if $y_i \left[\langle w, x_i \rangle + b \right] \leq 0$ **then**
      $w \leftarrow w + y_i x_i$ and $b \leftarrow b + y_i$
   end if
until all classified correctly

- Nothing happens if classified correctly
- Weight vector is linear combination $w = \sum_{i \in I} y_i x_i$
- Classifier is linear combination of inner products $f(x) = \sum_{i \in I} y_i \langle x_i, x \rangle + b$

Last time… **Perceptron on features**

initialize $w, b = 0$
repeat
    Pick (x_i, y_i) from data
    if $y_i(w \cdot \Phi(x_i) + b) \leq 0$ then
$$w' = w + y_i \Phi(x_i)$$
$$b' = b + y_i$$
until $y_i(w \cdot \Phi(x_i) + b) > 0$ for all i

- Nothing happens if classified correctly
- Weight vector is linear combination $w = \sum_{i \in I} y_i \phi(x_i)$
- Classifier is linear combination of inner products $f(x) = \sum_{i \in I} y_i \langle \phi(x_i), \phi(x) \rangle + b$

Today

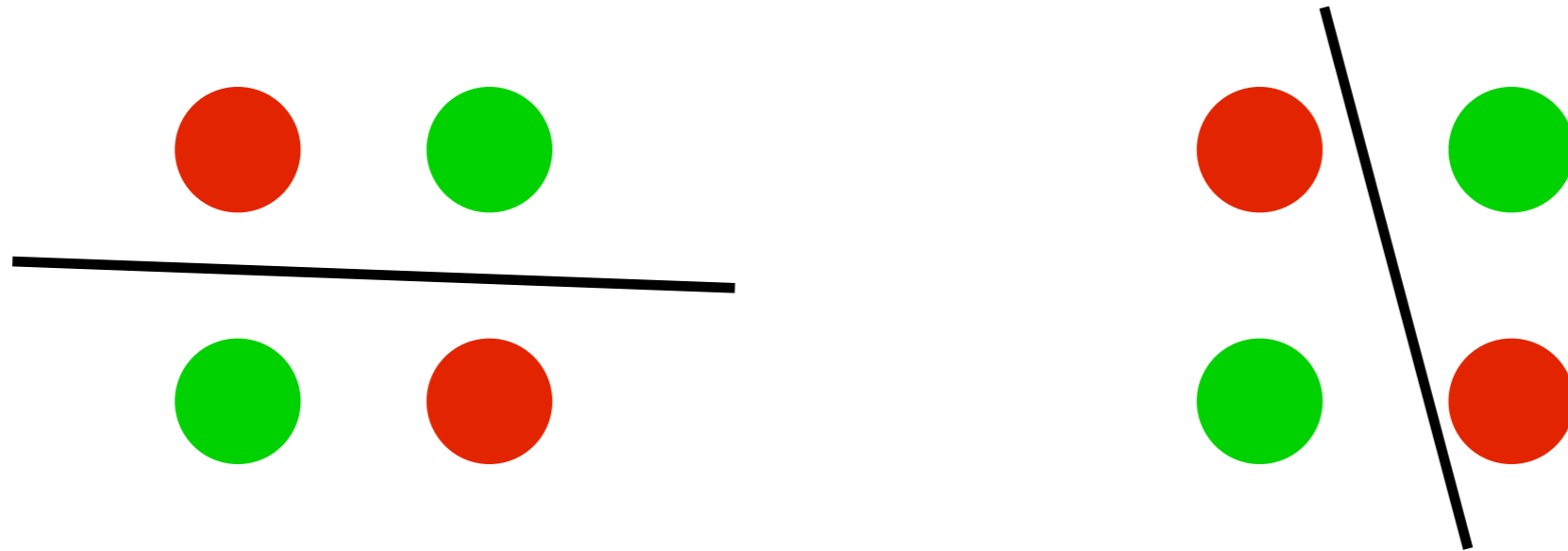- Multi-layer perceptron

- Forward Pass

Introduction

A brief history of computers

	1970s	1980s	1990s	2000s	2010s
Data	10^2	10^3	10^5	10^8	10^{11}
RAM	?	1MB	100MB	10GB	1TB
CPU	?	10MF	1GF	100GF	1PF GPU

deep nets   kernel methods   deep nets

- Data grows at higher exponent

- Moore's law (silicon) vs. Kryder's law (disks)

- Early algorithms data bound, now CPU/RAM bound

Not linearly separable data

- Some datasets are **not linearly separable**!
  - e.g. XOR problem

- Nonlinear separation is trivial

17

Addressing non-linearly separable data

- Two options:
  - Option 1: Non-linear features
  - Option 2: Non-linear classifiers

Option 1 — Non-linear features

- Choose non-linear features, e.g.,
  - Typical linear features: $w_0 + \Sigma_i w_i x_i$
  - Example of non-linear features:
    - Degree 2 polynomials, $w_0 + \Sigma_i w_i x_i + \Sigma_{ij} w_{ij} x_i x_j$

- Classifier $h_\mathbf{w}(\mathbf{x})$ still linear in parameters $\mathbf{w}$
  - As easy to learn
  - Data is linearly separable in higher dimensional spaces
  - Express via kernels

19

Option 2 — Non-linear classifiers

- Choose a classifier $h_\mathbf{w}(\mathbf{x})$ that is non-linear in parameters $\mathbf{w}$, e.g.,
  - Decision trees, neural networks,…

- More general than linear classifiers
- But, can often be harder to learn (non-convex optimization required)
- Often very useful (outperforms linear classifiers)
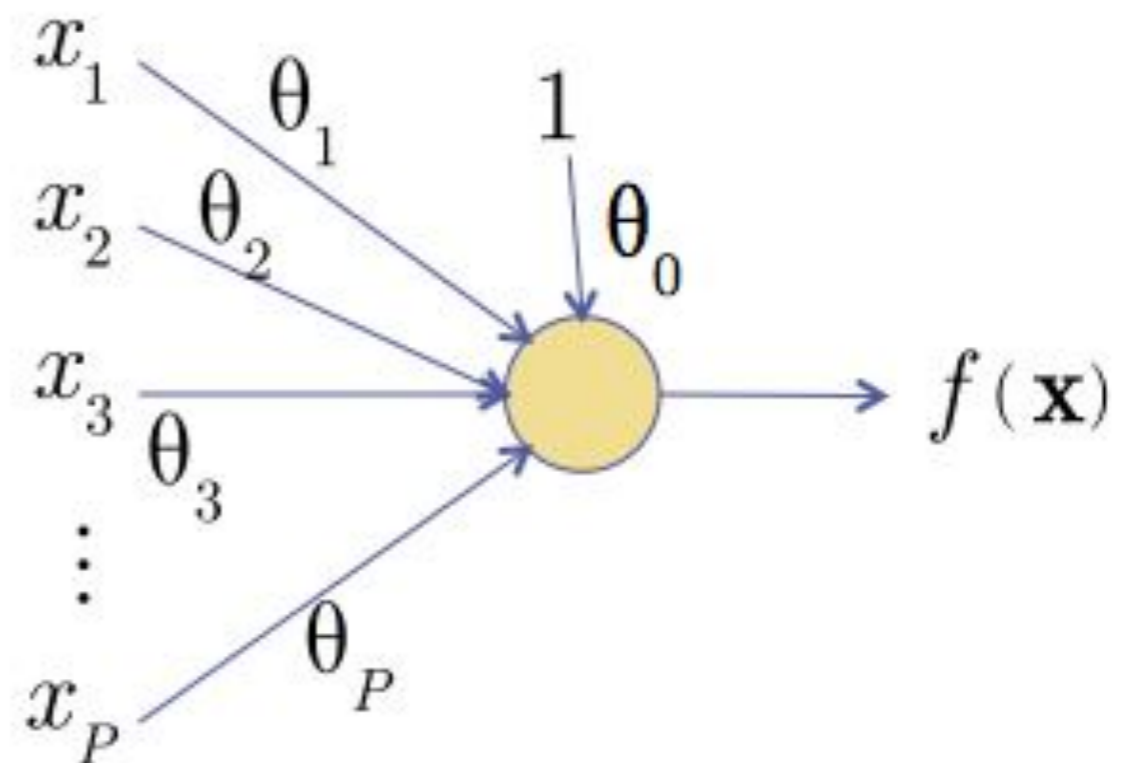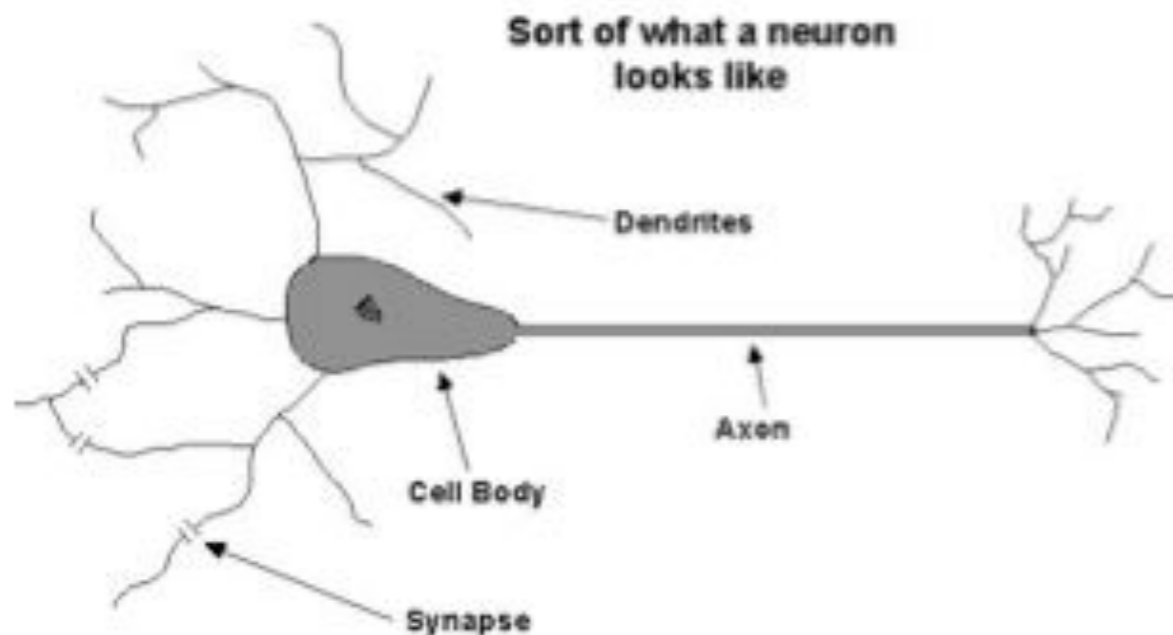- In a way, both ideas are related

Biological Neurons

- Soma (CPU)
  Cell body - combines signals

- Dendrite (input bus)
  Combines the inputs from
  several other nerve cells

- Synapse (interface)
  Interface and **parameter store** between neurons

- Axon (cable)
  May be up to 1m long and will transport the
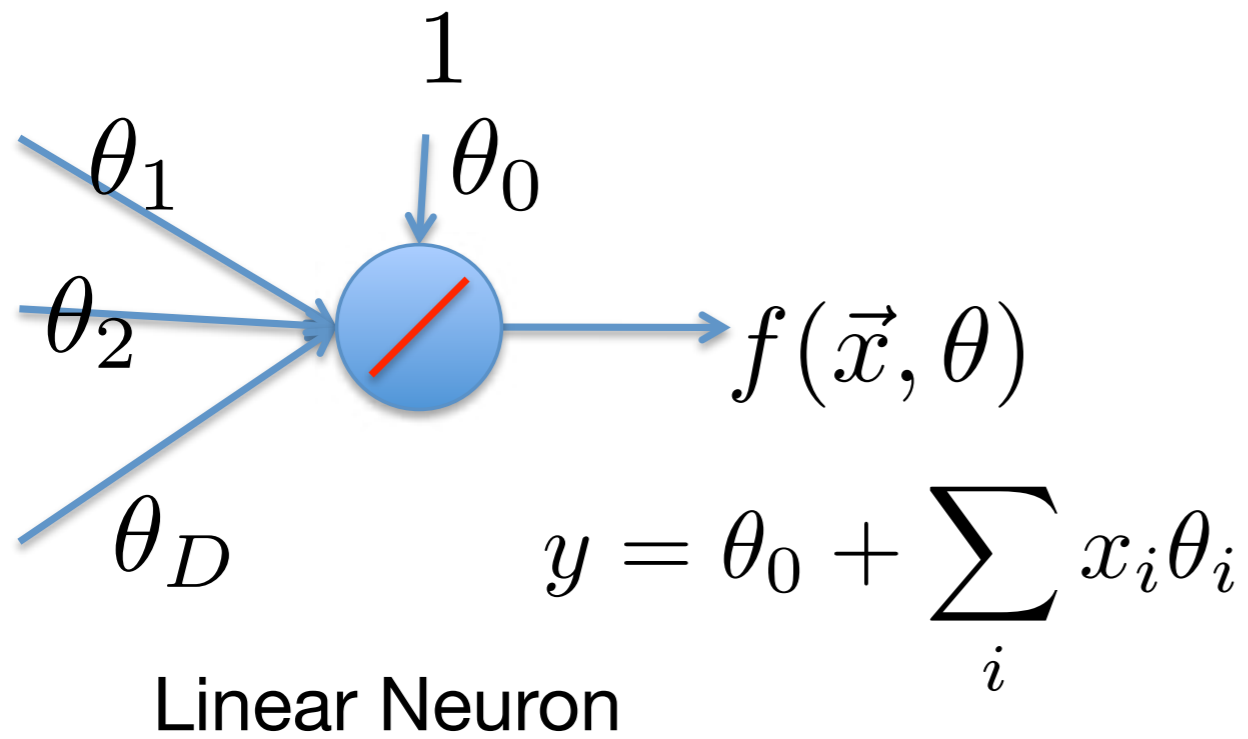 activation signal to neurons at different locations
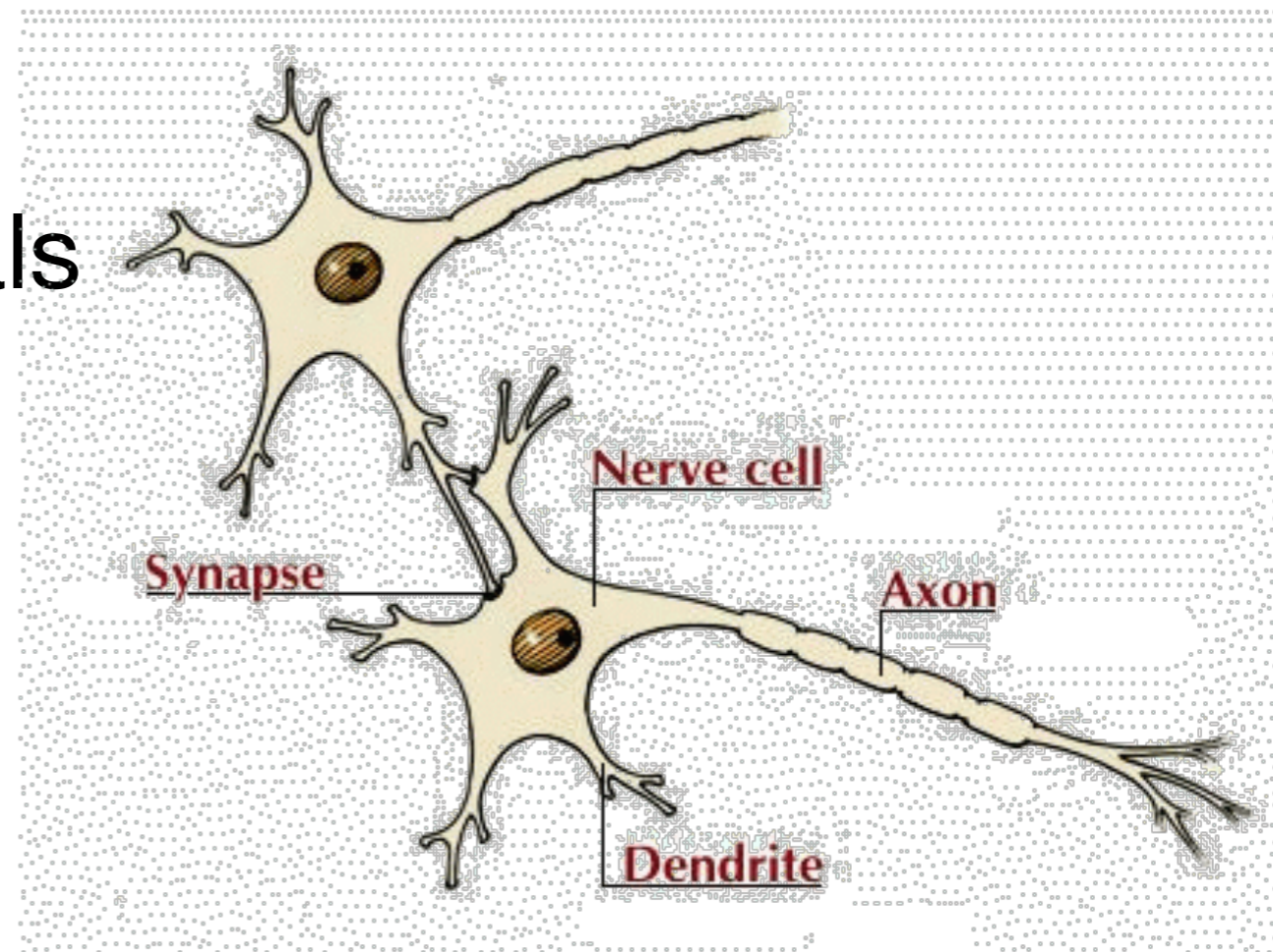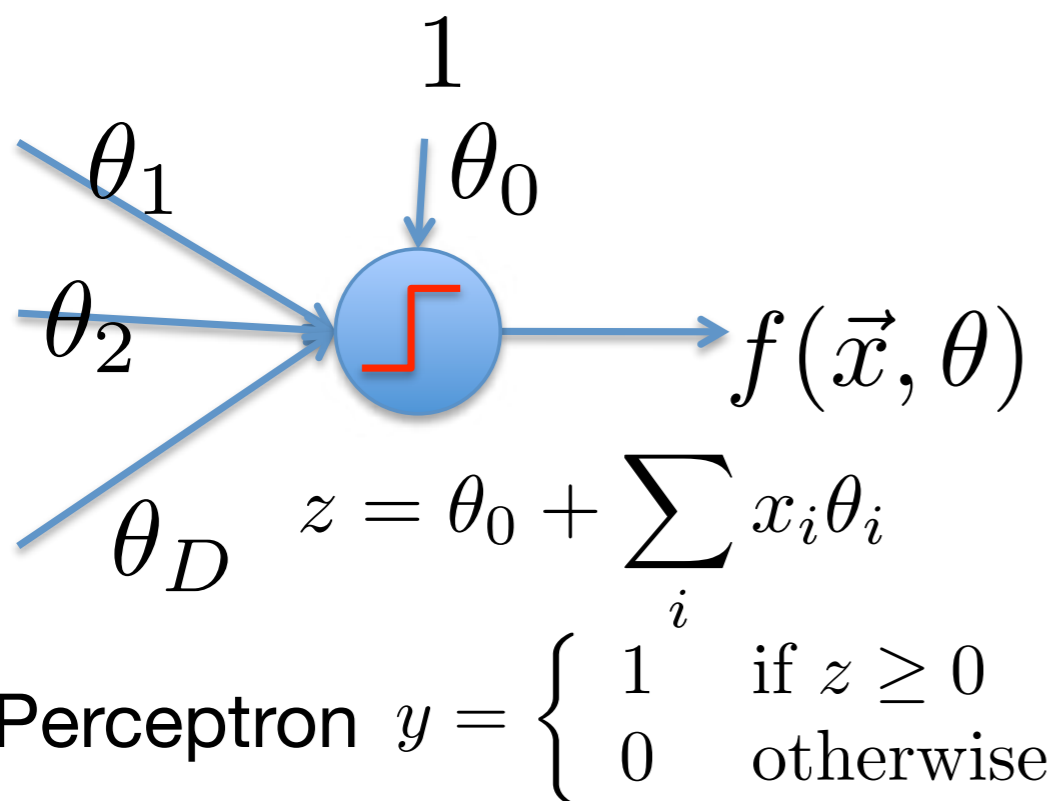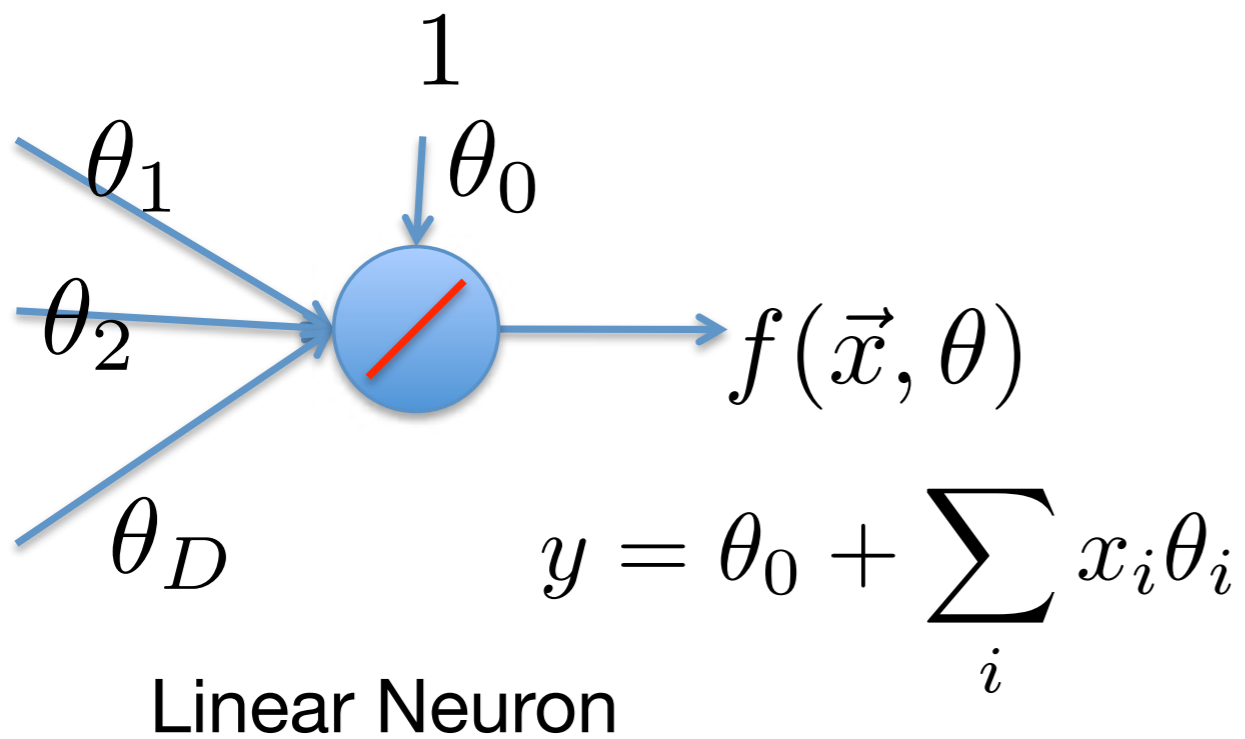
Recall: The Neuron Metaphor

- Neurons
  - accept information from multiple inputs,
  - transmit information to other neurons.
- Multiply inputs by weights along edges
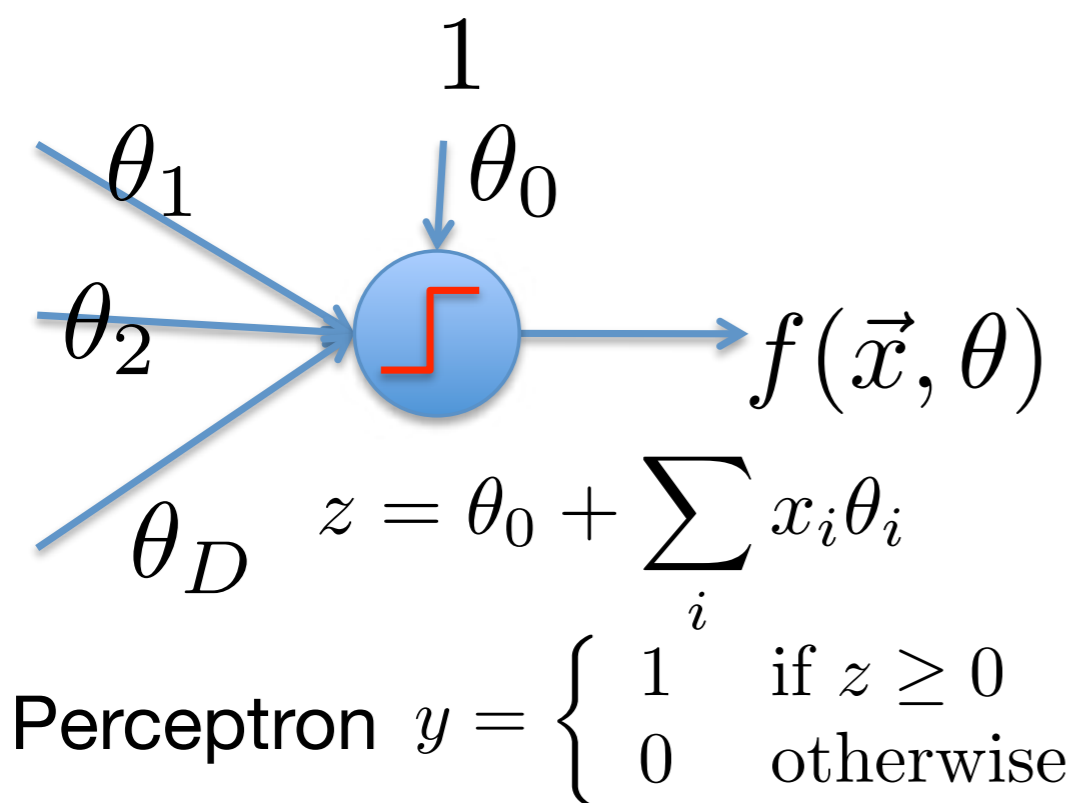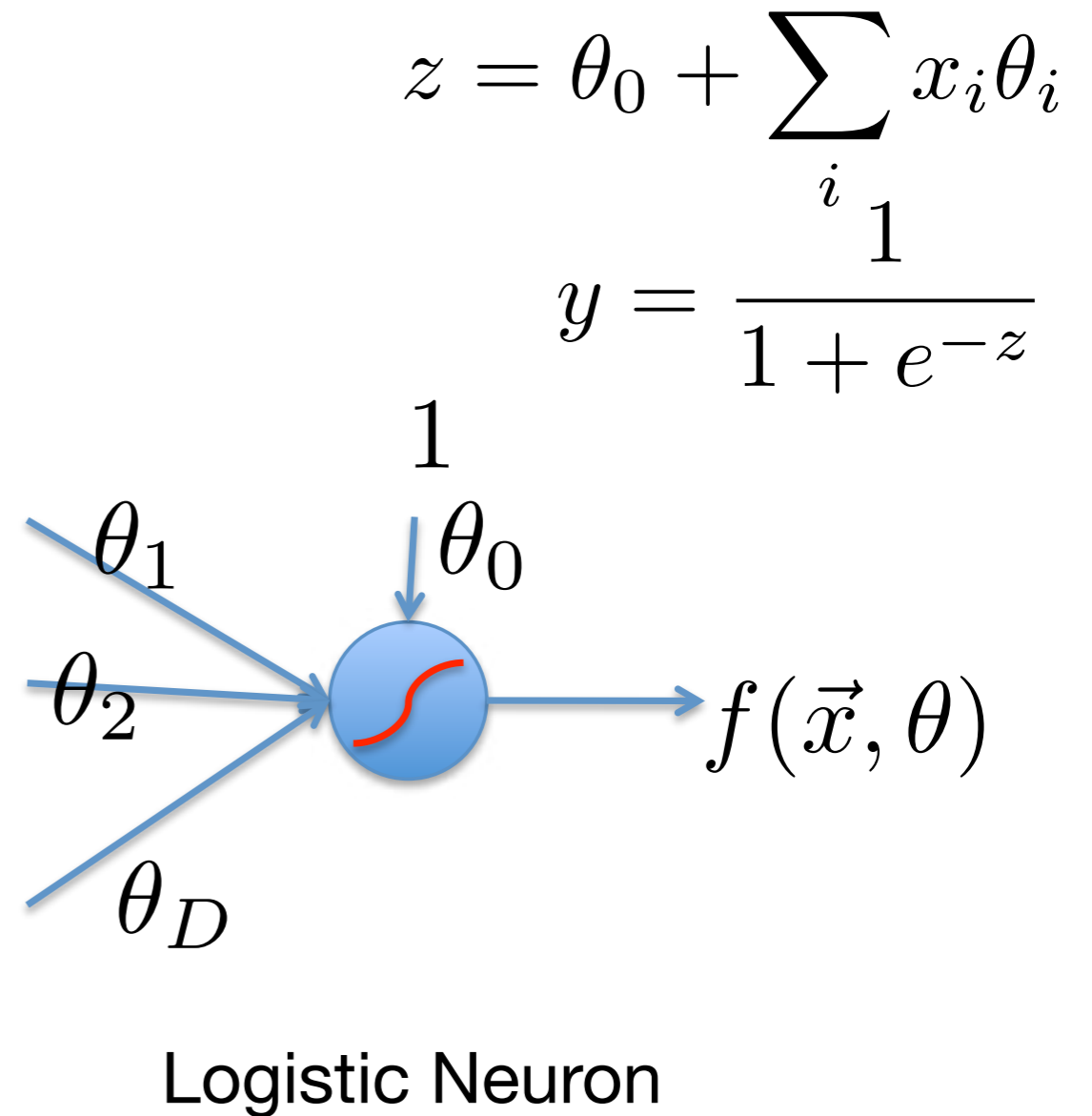- Apply some function to the set of inputs at each node

Sort of what a neuron looks like

Dendrites

Axon

Cell Body

Synapse

x_1 θ_1 1 θ_0

x_2 θ_2

x_3 θ_3 $f(\mathbf{x})$

x_P θ_P

22

Types of Neuron

1

θ_1

θ_0

θ_2

θ_D

$f(\vec{x}, \theta)$

$$y = \theta_0 + \sum_i x_i \theta_i$$

Linear Neuron

Types of Neuron

θ_1

θ_2

θ_D

1

θ_0

$f(\vec{x}, \theta)$

$$y = \theta_0 + \sum_i x_i \theta_i$$
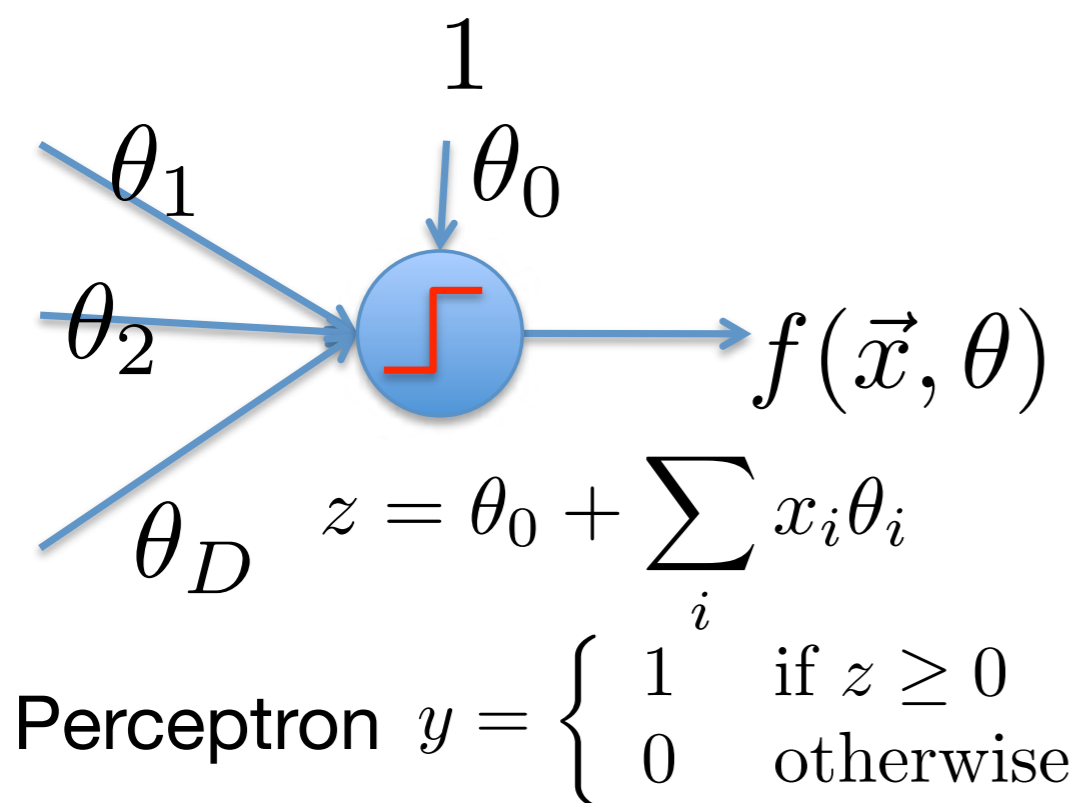
Linear Neuron

θ_1

θ_2

θ_D

1

θ_0

$f(\vec{x}, \theta)$

$$z = \theta_0 + \sum_i x_i \theta_i$$

Perceptron $y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$

24

Types of Neuron

$$1$$

$$\theta_1$$
$$\theta_2$$
$$\theta_0$$

$$f(\vec{x}, \theta)$$

$$\theta_D$$

$$y = \theta_0 + \sum_i x_i \theta_i$$

Linear Neuron

$$z = \theta_0 + \sum_i x_i \theta_i$$

$$y = \frac{1}{1 + e^{-z}}$$

$$1$$

$$\theta_1$$
$$\theta_2$$
$$\theta_0$$

$$f(\vec{x}, \theta)$$

$$\theta_D$$

Logistic Neuron

$$1$$

$$\theta_1$$
$$\theta_2$$
$$\theta_0$$

$$f(\vec{x}, \theta)$$

$$\theta_D$$

$$z = \theta_0 + \sum_i x_i \theta_i$$

Perceptron $y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$

Types of Neuron

$$1$$

$$\theta_1$$
$$\theta_2$$
$$\theta_D$$

$$\theta_0$$

$$f(\vec{x}, \theta)$$

$$y = \theta_0 + \sum_i x_i \theta_i$$

Linear Neuron

$$z = \theta_0 + \sum_i x_i \theta_i$$

$$y = \frac{1}{1 + e^{-z}}$$

$$1$$

$$\theta_1$$
$$\theta_2$$
$$\theta_D$$

$$\theta_0$$

$$f(\vec{x}, \theta)$$

Logistic Neuron

$$1$$

$$\theta_1$$
$$\theta_2$$
$$\theta_D$$

$$\theta_0$$

$$f(\vec{x}, \theta)$$

$$z = \theta_0 + \sum_i x_i \theta_i$$

Perceptron $\quad y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$
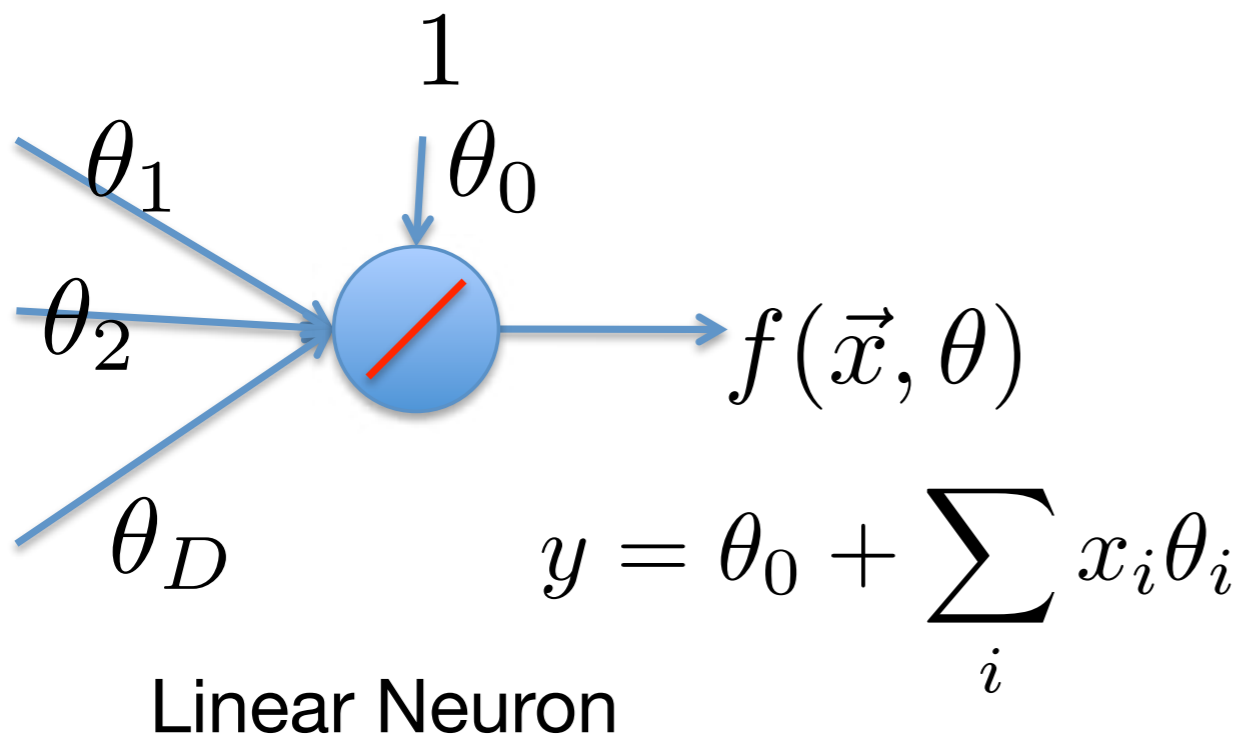
- Potentially more. Requires a convex loss function for gradient descent training.
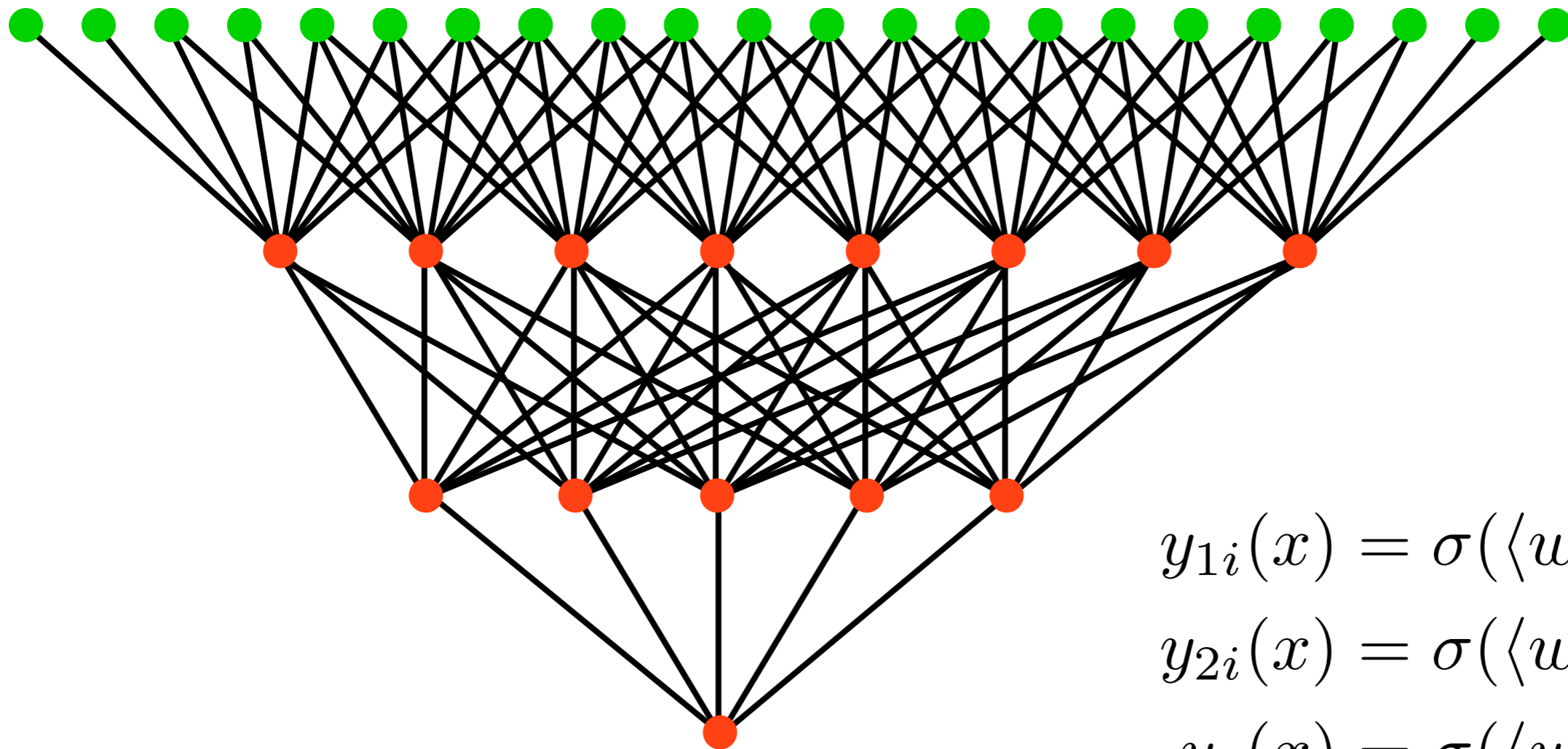
26

Limitation

- A single "neuron" is still a linear decision boundary

- What to do?

- Idea: Stack a bunch of them together!

Nonlinearities via Layers

- Cascade neurons together
- The output from one layer is the input to the next
- Each layer has its own sets of weights

$$y_{1i} = k(x_i, x)$$

Kernels

$$y_{1i}(x) = \sigma(\langle w_{1i}, x \rangle)$$
$$y_2(x) = \sigma(\langle w_2, y_1 \rangle)$$

Deep Nets

optimize
all weights
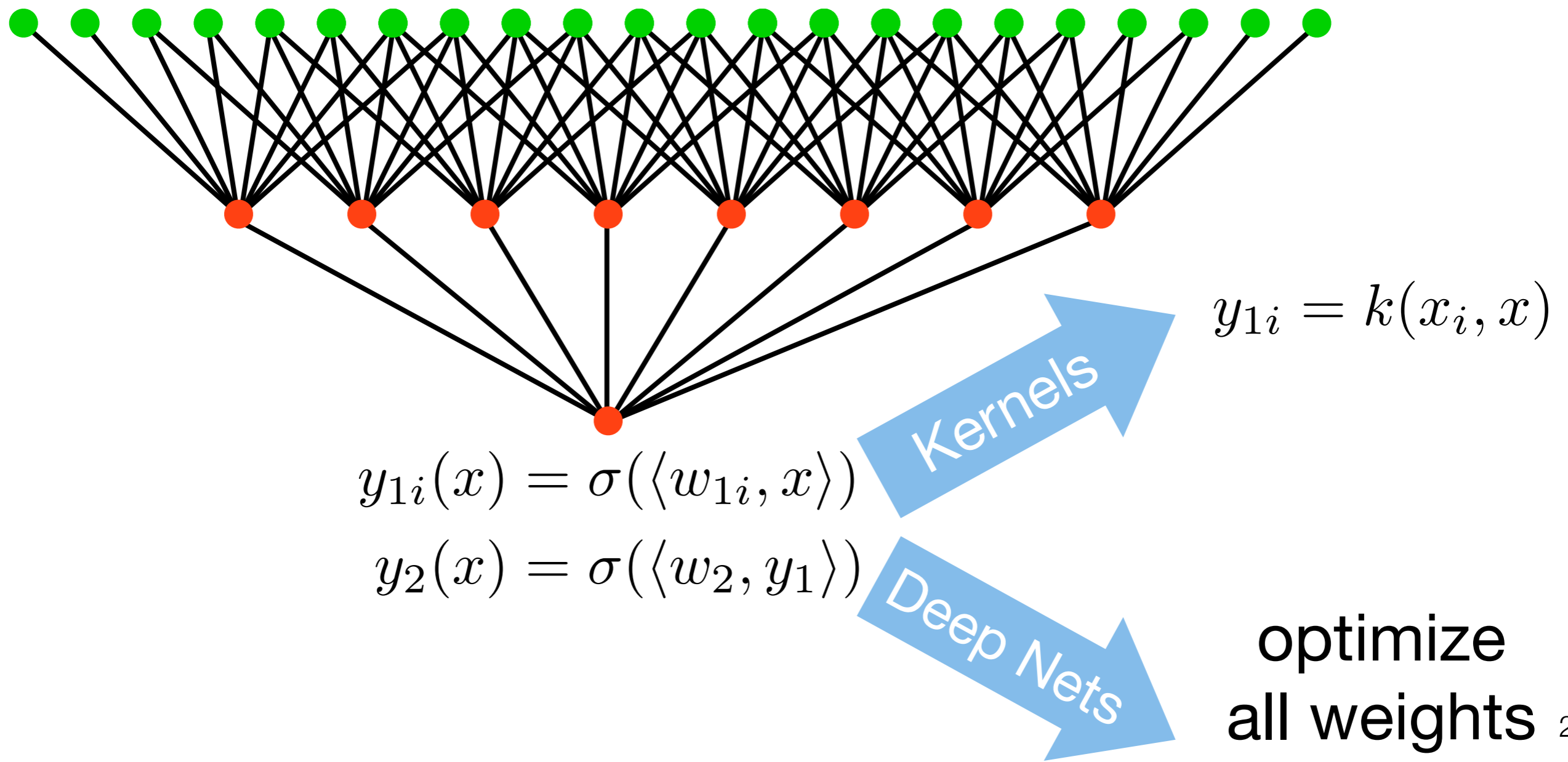
Nonlinearities via Layers

$$y_{1i}(x) = \sigma(\langle w_{1i}, x \rangle)$$

$$y_{2i}(x) = \sigma(\langle w_{2i}, y_1 \rangle)$$

$$y_3(x) = \sigma(\langle w_3, y_2 \rangle)$$

Representational Power

- Neural network with at least one hidden layer is a universal approximator (can represent any function).
Proof in: Approximation by Superpositions of Sigmoidal Function, Cybenko, paper

3 hidden neurons   6 hidden neurons   20 hidden neurons
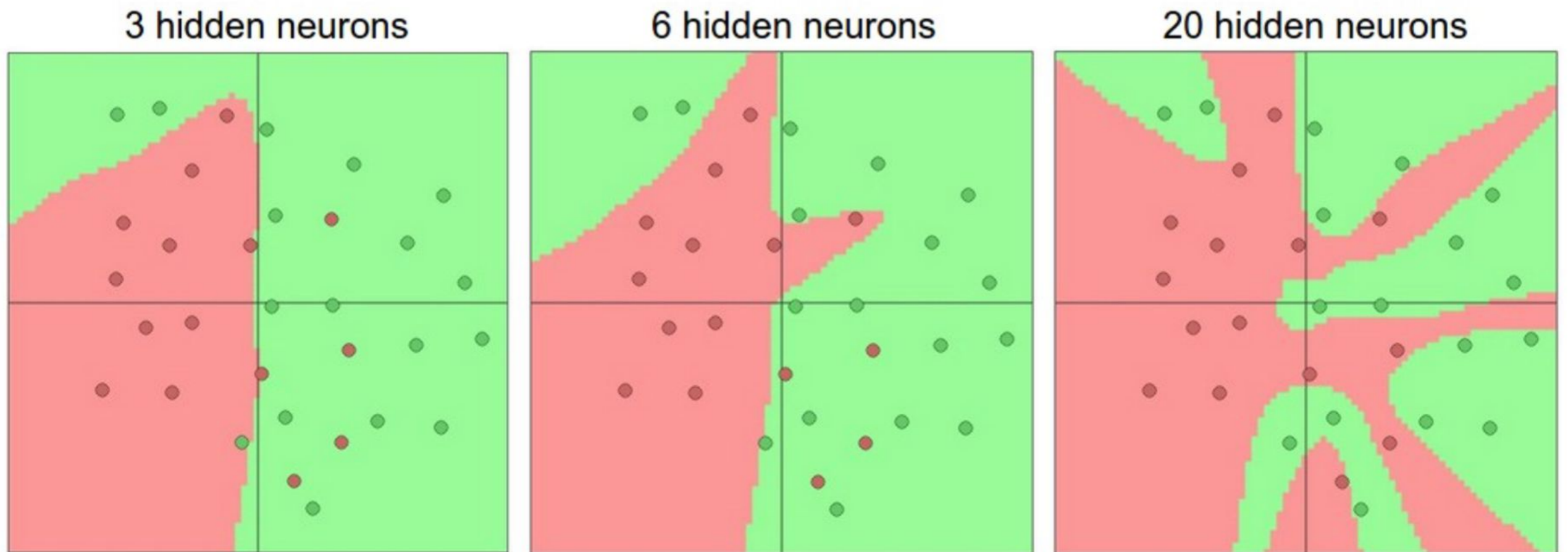
- The capacity of the network increases with more hidden units and more hidden layers

A simple example

- Consider a neural network with two layers of neurons.
  - neurons in the top layer represent known shapes.
  - neurons in the bottom layer represent pixel intensities.

- A pixel gets to vote if it has ink on it.
  - Each inked pixel can vote for several different shapes.

- The shape that gets the most votes wins.

⓪ ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨

x_1

x_2

x_3

$\vdots$

x_i

$f(\sum w_i x_i)$
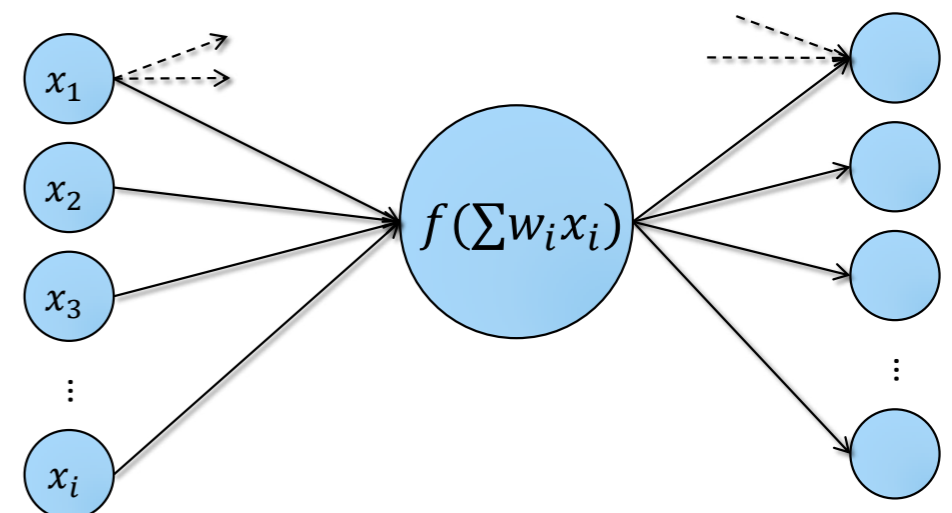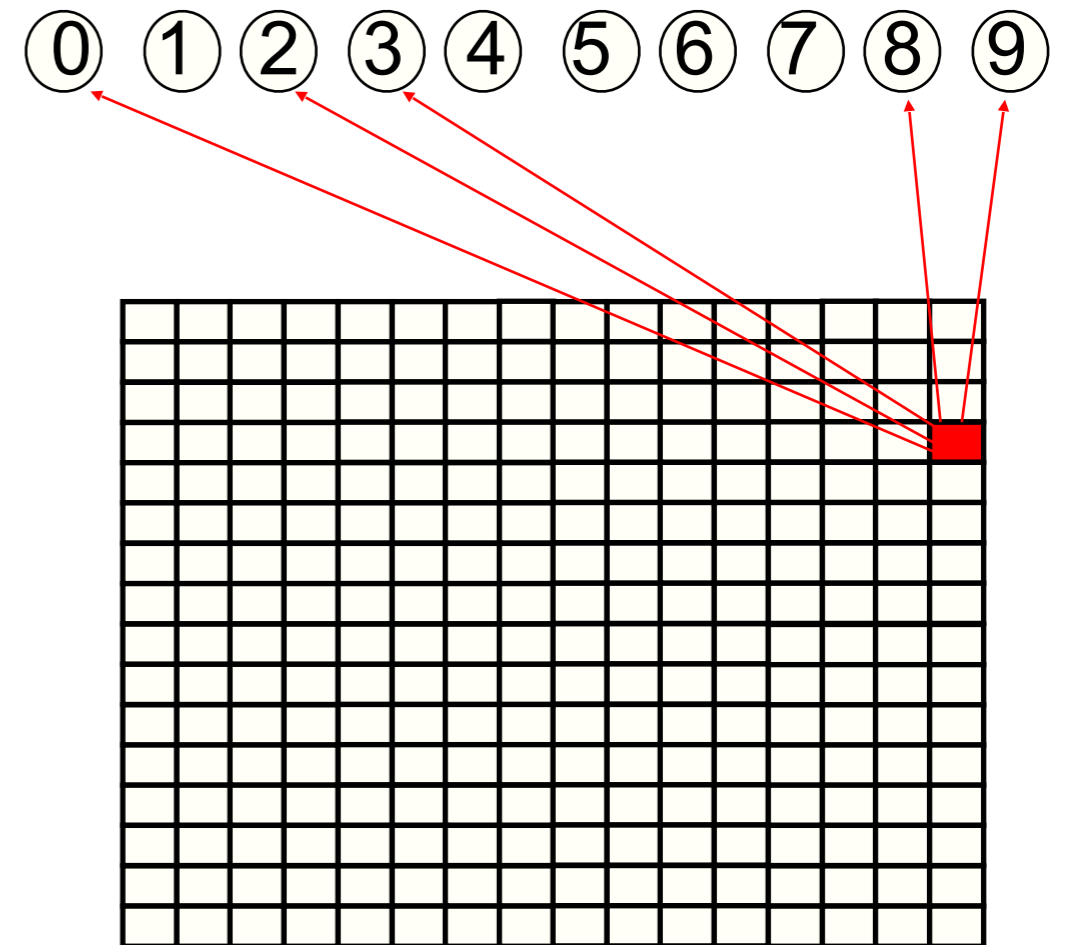
How to display the weights
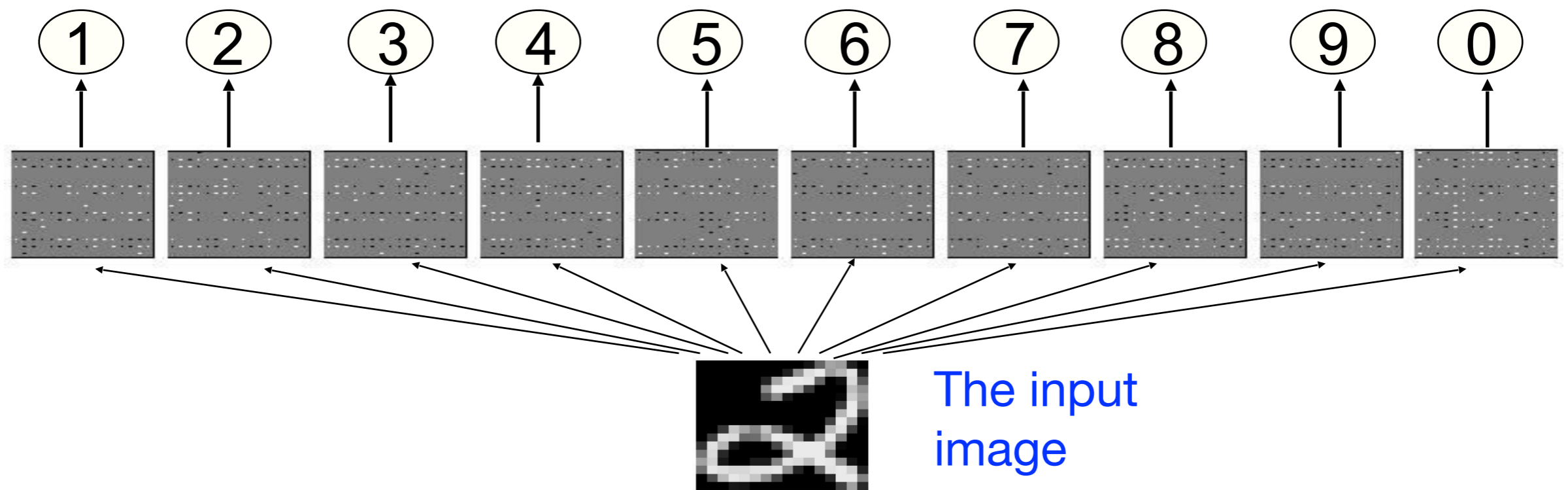
The input image

Give each output unit its own "map" of the input image and display the weight coming from each pixel in the location of that pixel in the map.

Use a black or white blob with the area representing the magnitude of the weight and the color representing the sign.
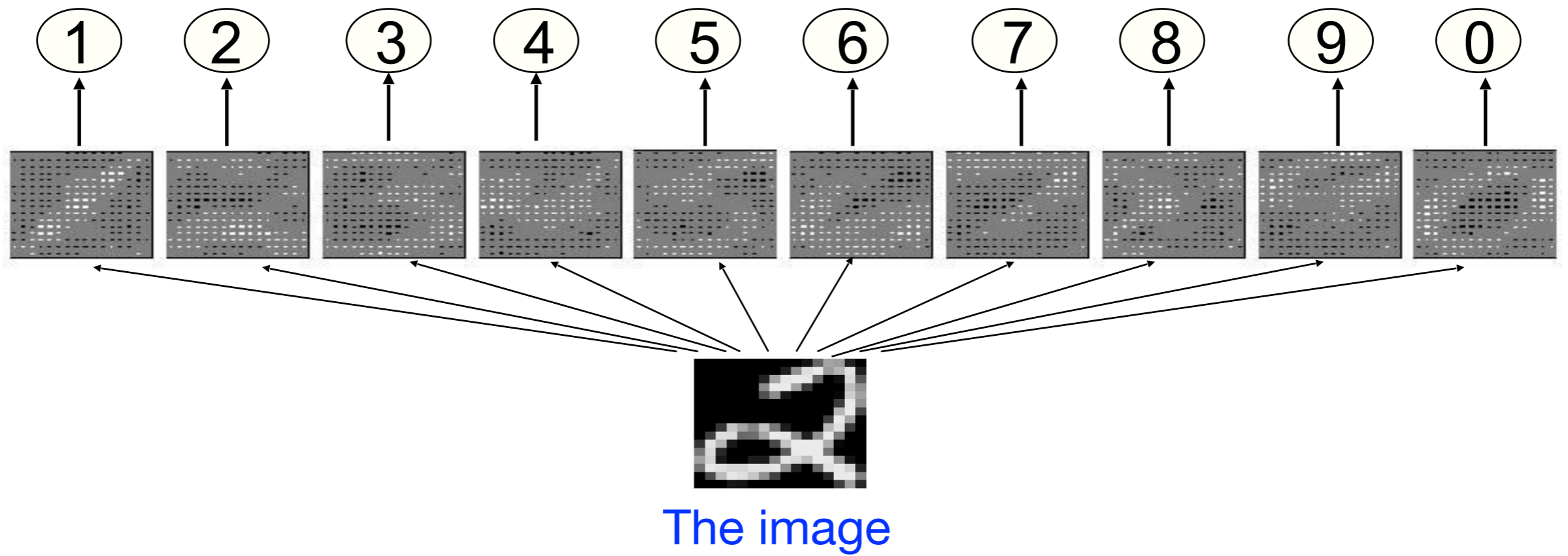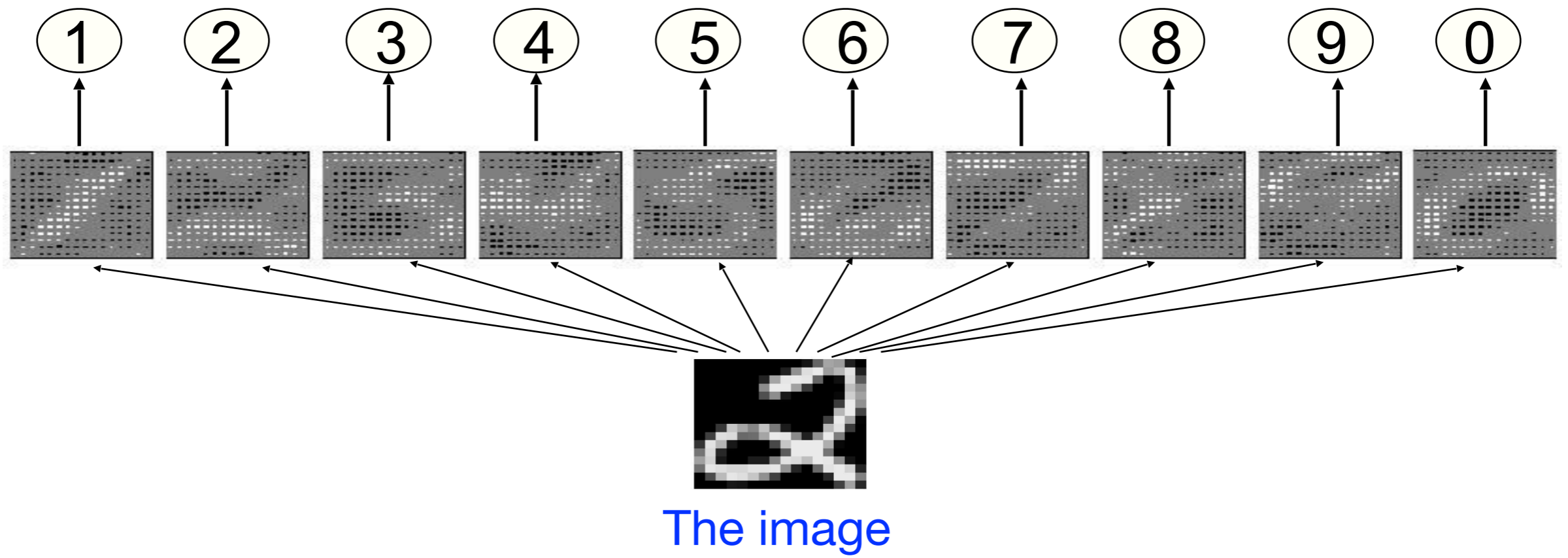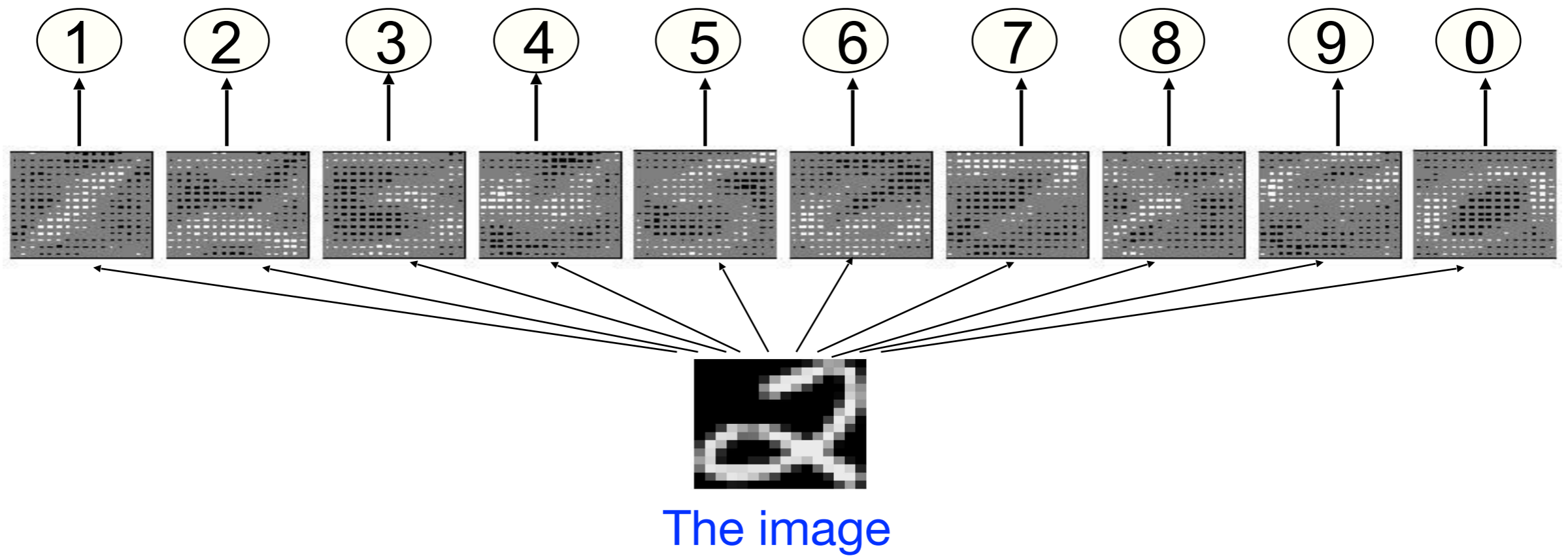
32

How to learn the weights

The image
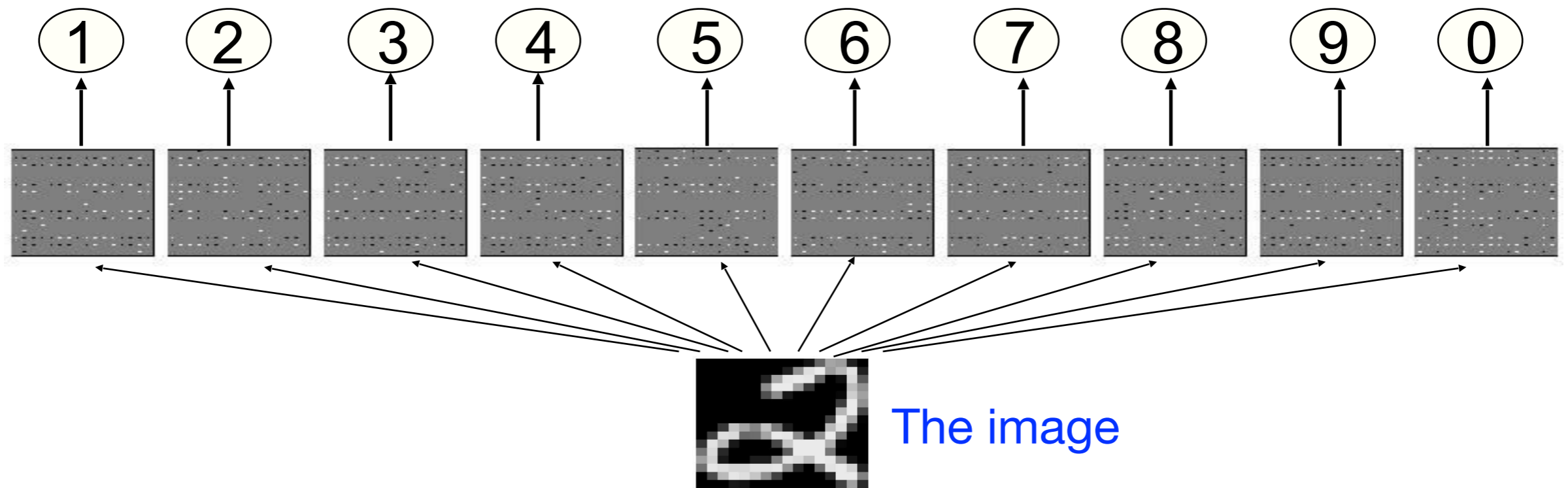
Show the network an image and increment the weights from active pixels to the correct class.

Then decrement the weights from active pixels to whatever class the network guesses.

33

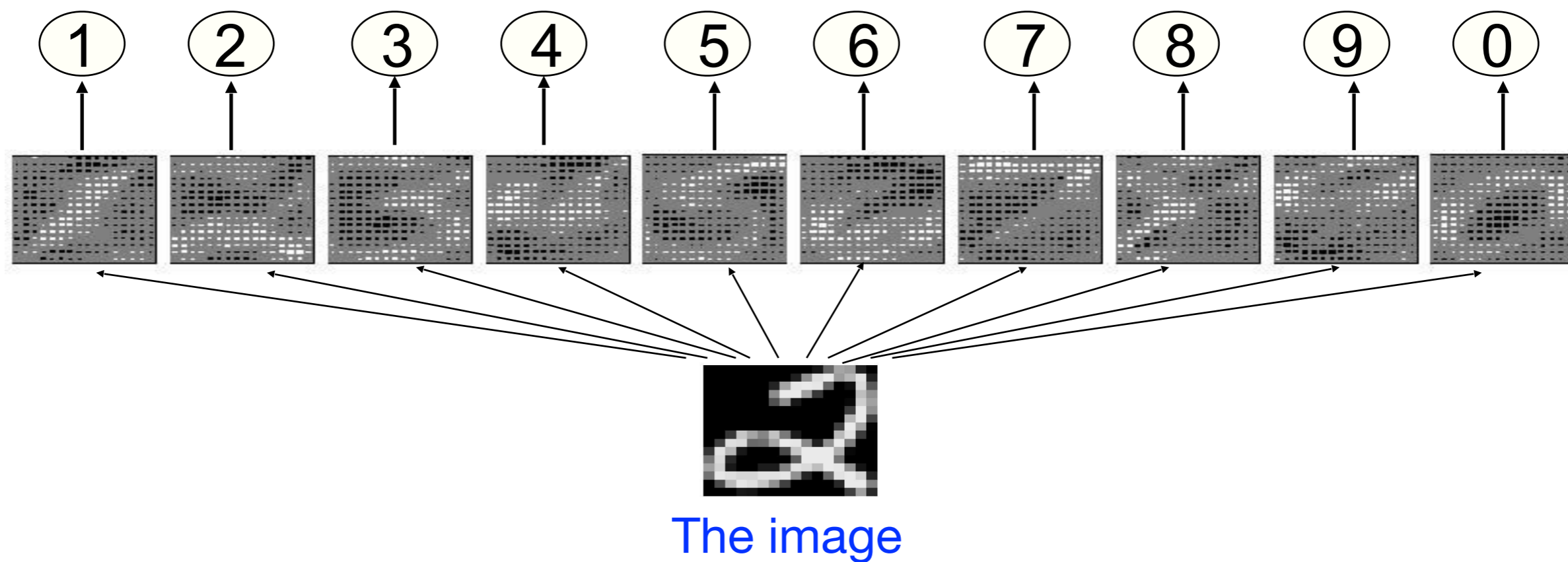The image

34

The image

35

The image

The image

The image
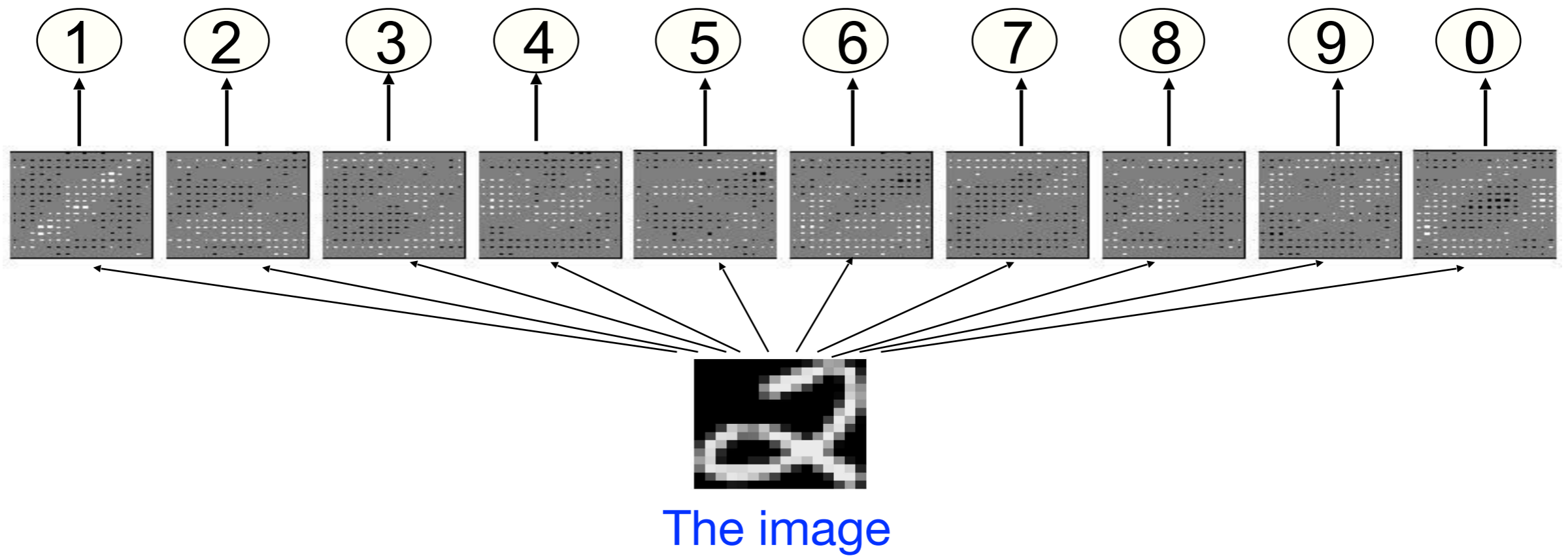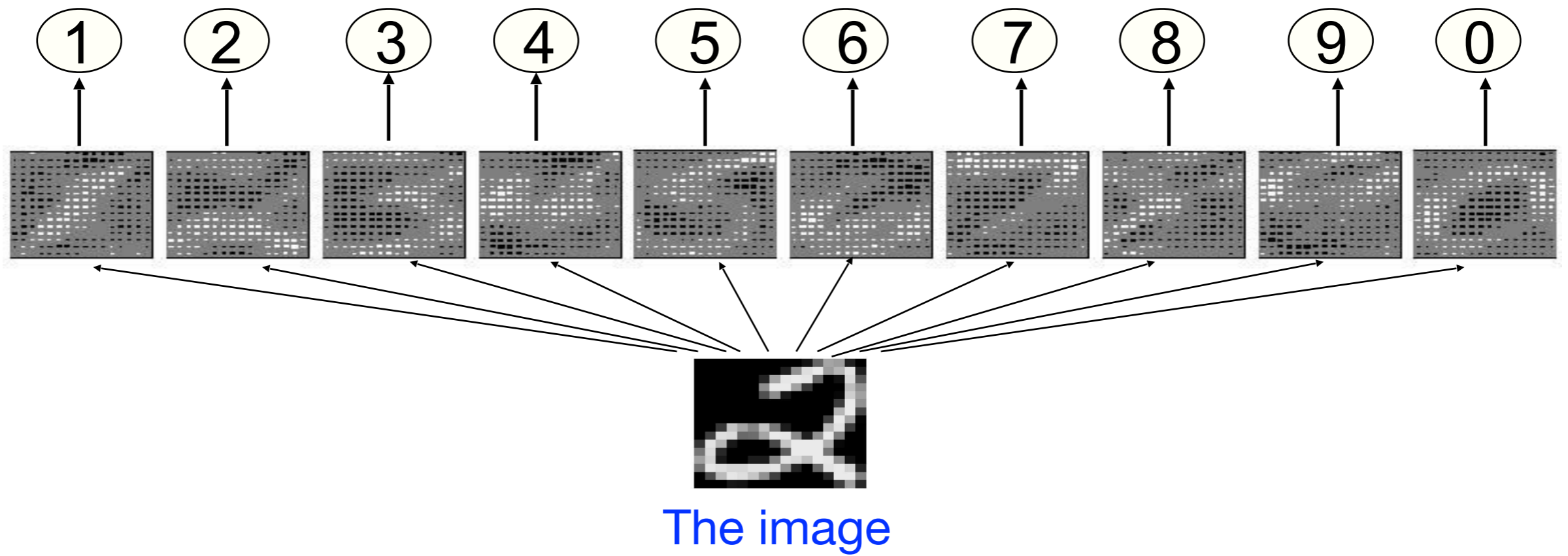
The learned weights

The image

Why insufficient

- A two layer network with a single winner in the top layer is equivalent to having a rigid template for each shape.
  - The winner is the template that has the biggest overlap with the ink.

- The ways in which hand-written digits vary are much too complicated to be captured by simple template matches of whole shapes.
  - To capture all the allowable variations of a digit we need to learn the features that it is composed of.
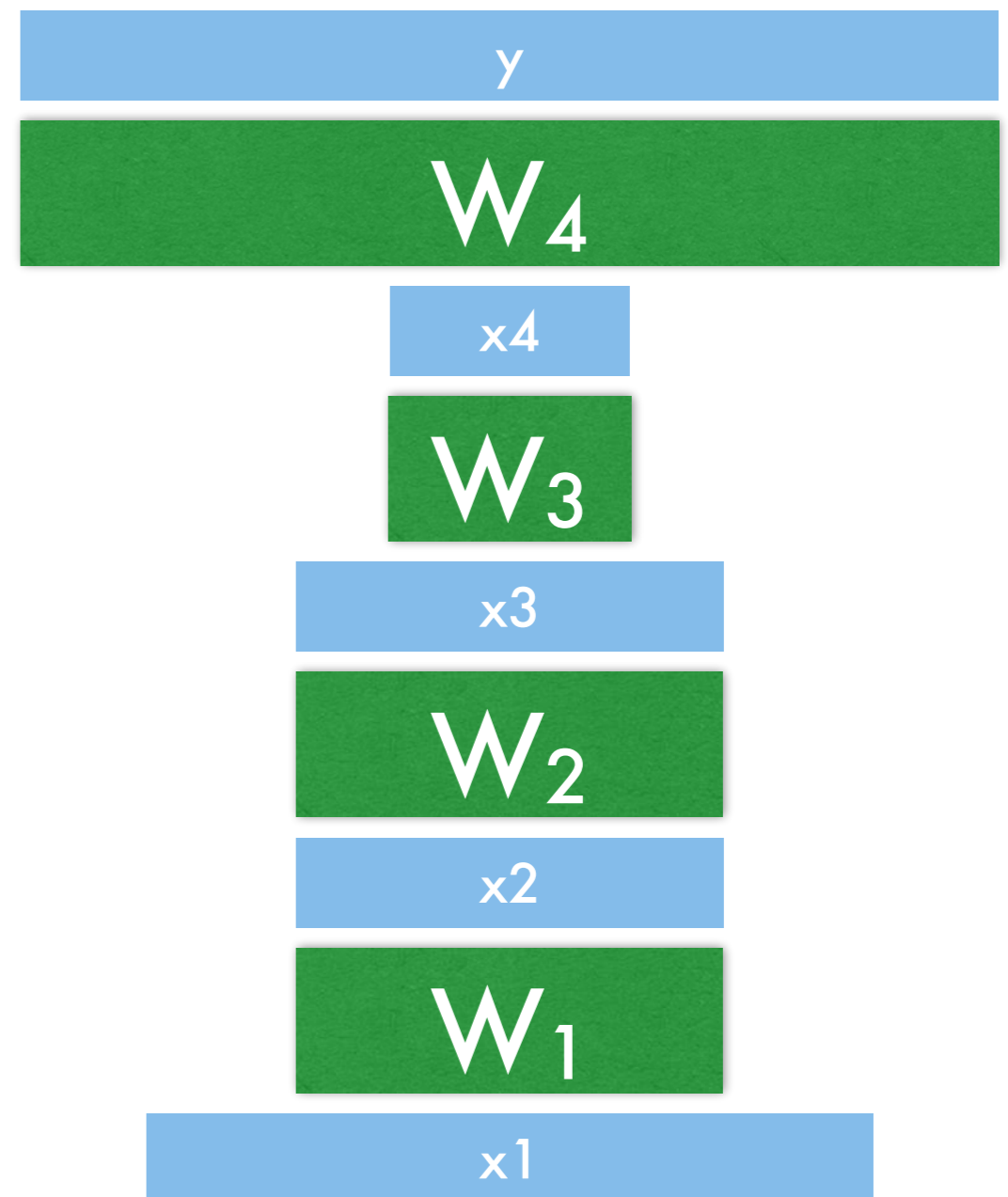
Multilayer Perceptron

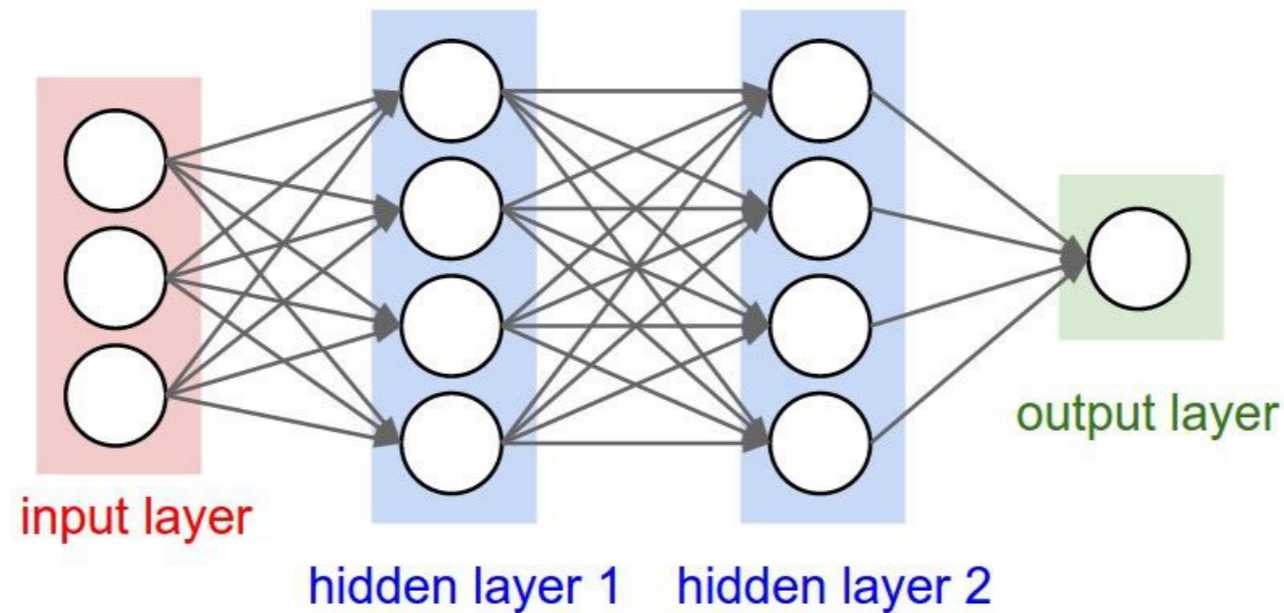- Layer Representation

$$y_i = W_i x_i$$
$$x_{i+1} = \sigma(y_i)$$

- (typically) iterate between linear mapping Wx and nonlinear function

- Loss function $l(y, y_i)$ to measure quality of estimate so far

Forward Pass

Forward Pass: What does the Network Compute?

input layer

hidden layer 1   hidden layer 2

output layer

- Output of the network can be written as:

$$h_j(\mathbf{x}) \quad = \quad f\left(v_{j0} + \sum_{i=1}^{D} x_i v_{ji}\right)$$

$$o_k(\mathbf{x}) \quad = \quad g\left(w_{k0} + \sum_{j=1}^{J} h_j(\mathbf{x}) w_{kj}\right)$$
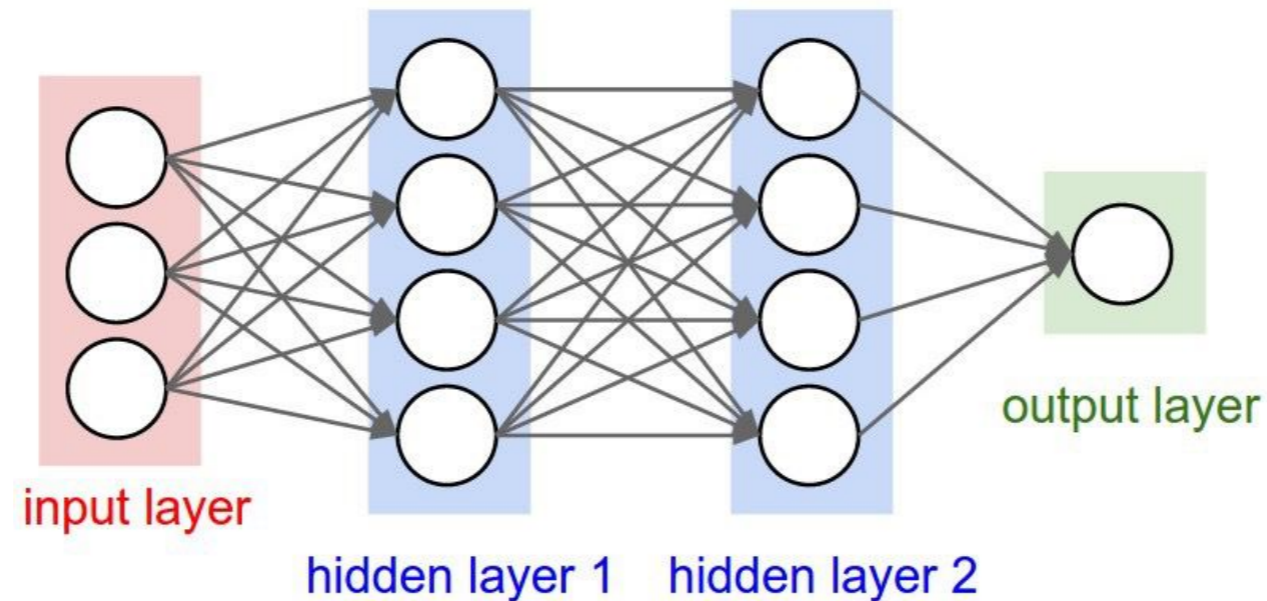
(j indexing hidden units, k indexing the output units, D number of inputs)

- Activation functions f , g : sigmoid/logistic, tanh, or rectified linear (ReLU)

$$\sigma(z) = \frac{1}{1 + \exp(-z)}, \quad \tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}, \quad \mathrm{ReLU}(z) = \max(0, z)$$

Forward Pass in Python

- Example code for a forward pass for a 3-layer network in Python:


```python
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

- Can be implemented efficiently using matrix operations
- Example above: W_1 is matrix of size 4×3, W_2 is 4×4. What about biases and W_3?

[http://cs231n.github.io/neural-networks-1/]

Special Case

- What is a single layer (no hiddens) network with a sigmoid act. function?

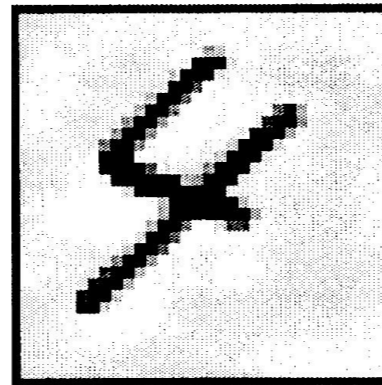Input Layer   Output Layer
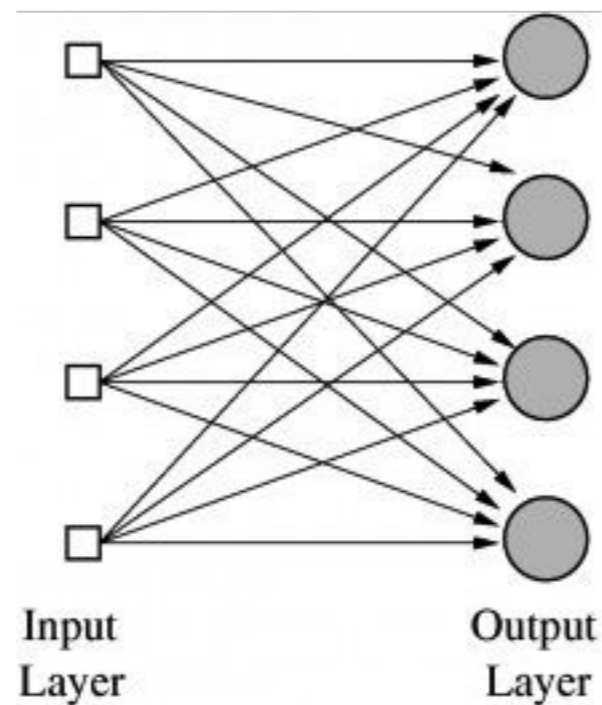
- Network:

$$o_k(\mathbf{x}) = \frac{1}{1 + \exp(-z_k)}$$

$$z_k = w_{k0} + \sum_{j=1}^{J} x_j w_{kj}$$

- Logistic regression!

Example

- Classify image of handwritten digit (32x32 pixels): 4 vs non-4

- How would you build your network?

- For example, use one hidden layer and the sigmoid activation function:

$$o_k(\mathbf{x}) = \frac{1}{1 + \exp(-z_k)}$$

$$z_k = w_{k0} + \sum_{j=1}^{J} h_j(\mathbf{x}) w_{kj}$$

- How can we train the network, that is, adjust all the parameters $\mathbf{w}$?

Training Neural Networks

- Find weights:

$$\mathbf{w}^* = \operatorname*{argmin}_{\mathbf{w}} \sum_{n=1}^{N} \operatorname{loss}(\mathbf{o}^{(n)}, \mathbf{t}^{(n)})$$

where $\mathbf{o} = f(\mathbf{x};\mathbf{w})$ is the output of a neural network

- Define a loss function, e.g.:
  - Squared loss: $\sum_k \frac{1}{2}(o_k^{(n)} - t_k^{(n)})^2$
  - Cross-entropy loss: $-\sum_k t_k^{(n)} \log o_k^{(n)}$
- Gradient descent:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{\partial E}{\partial \mathbf{w}^t}$$

where η is the learning rate (and E is error/loss)

Useful derivatives

name	function	derivative
Sigmoid	$\sigma(z) = \frac{1}{1+\exp(-z)}$	$\sigma(z) \cdot (1 - \sigma(z))$
Tanh	$\tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$	$1/\cosh^2(z)$
ReLU	$\mathrm{ReLU}(z) = \max(0, z)$	$\begin{cases} 1, & \text{if } z > 0 \\ 0, & \text{if } z \leq 0 \end{cases}$

Next Lecture:

Backpropagation