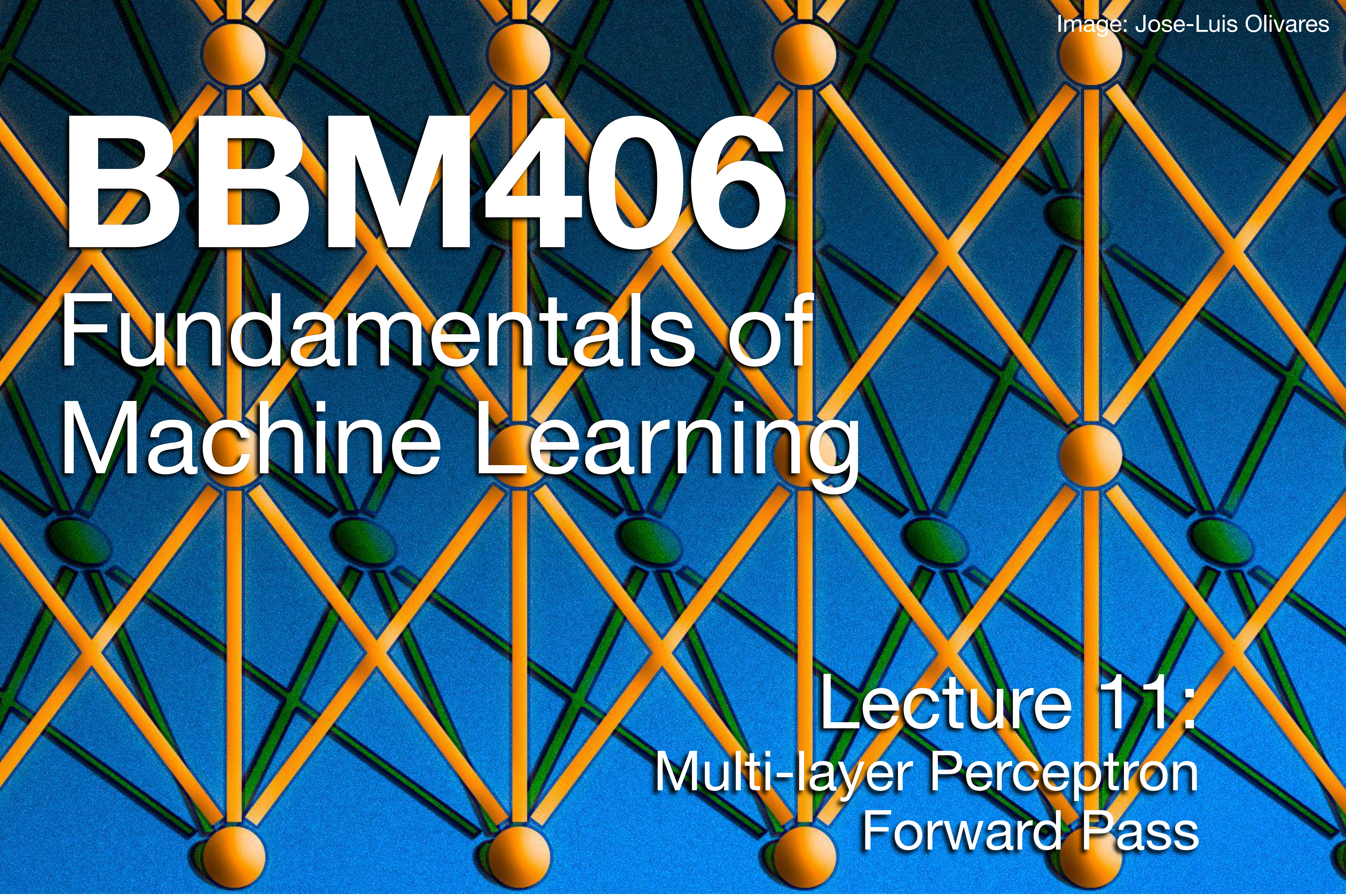


Image: Jose-Luis Olivares

BBM406

Fundamentals of Machine Learning

Lecture 11:

Multi-layer Perceptron
Forward Pass

HACETTEPE UNIVERSITY COMPUTER VISION LAB

Aykut Erdem // Hacettepe University // Fall 2019

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

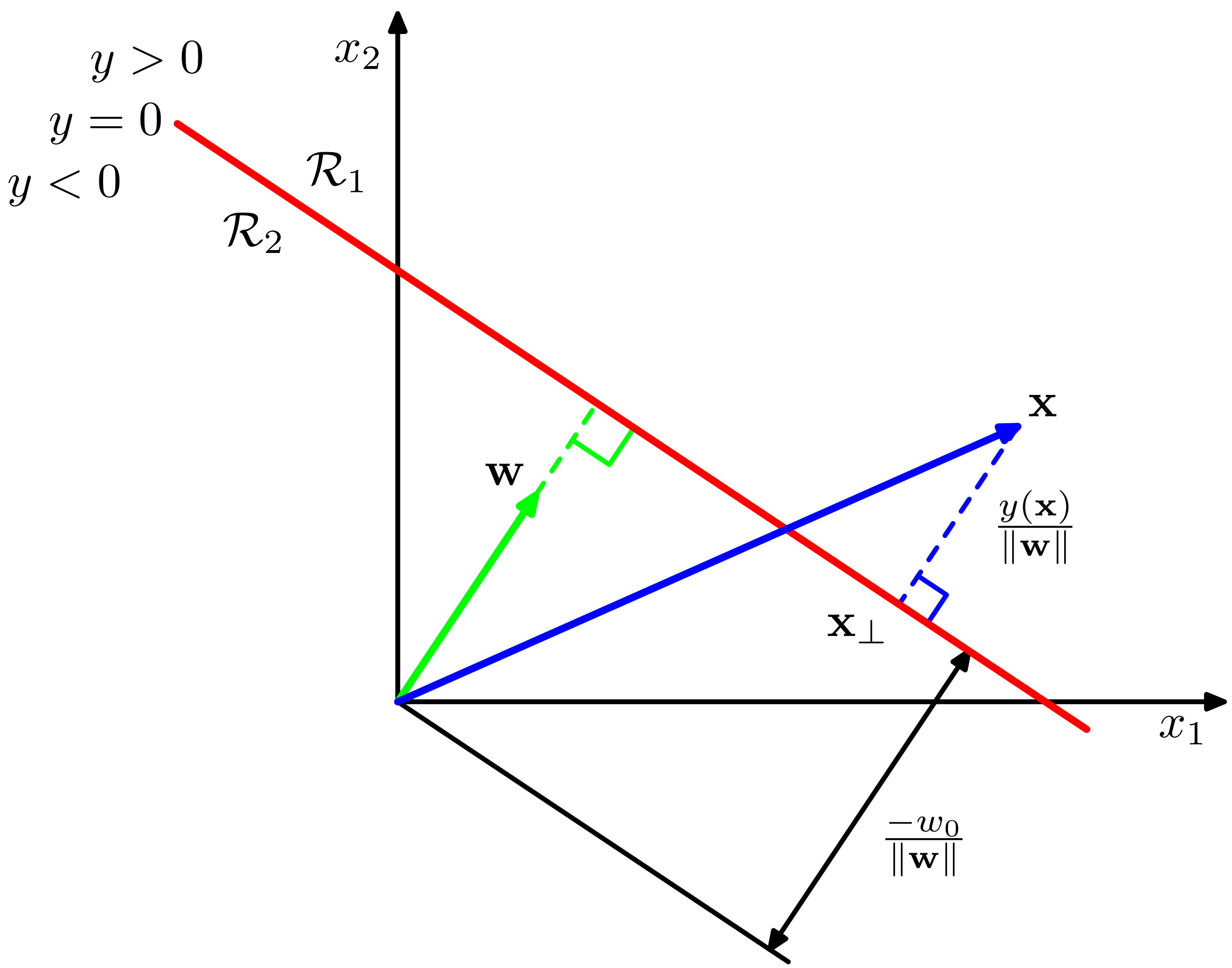
$$C(\mathbf{x}) = \mathrm{sign}(\mathbf{w}^T\mathbf{x} + w_0)$$

where step function sign(·) is defined as

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

slide by Ce Liu

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

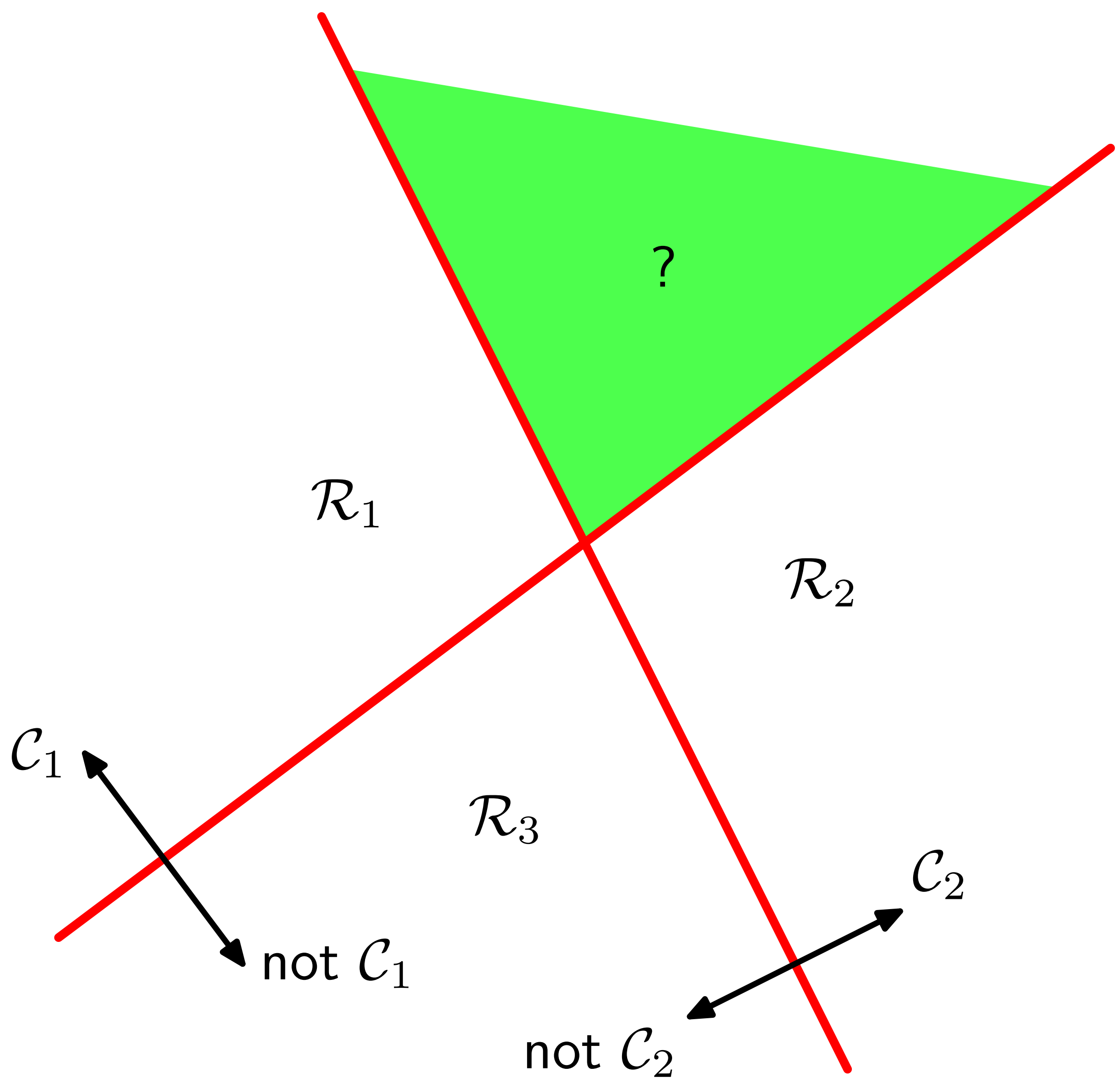
$$\frac{\mathbf{w}^T\mathbf{x}}{\|\mathbf{w}\|} = -\frac{w_0}{\|\mathbf{w}\|}$$

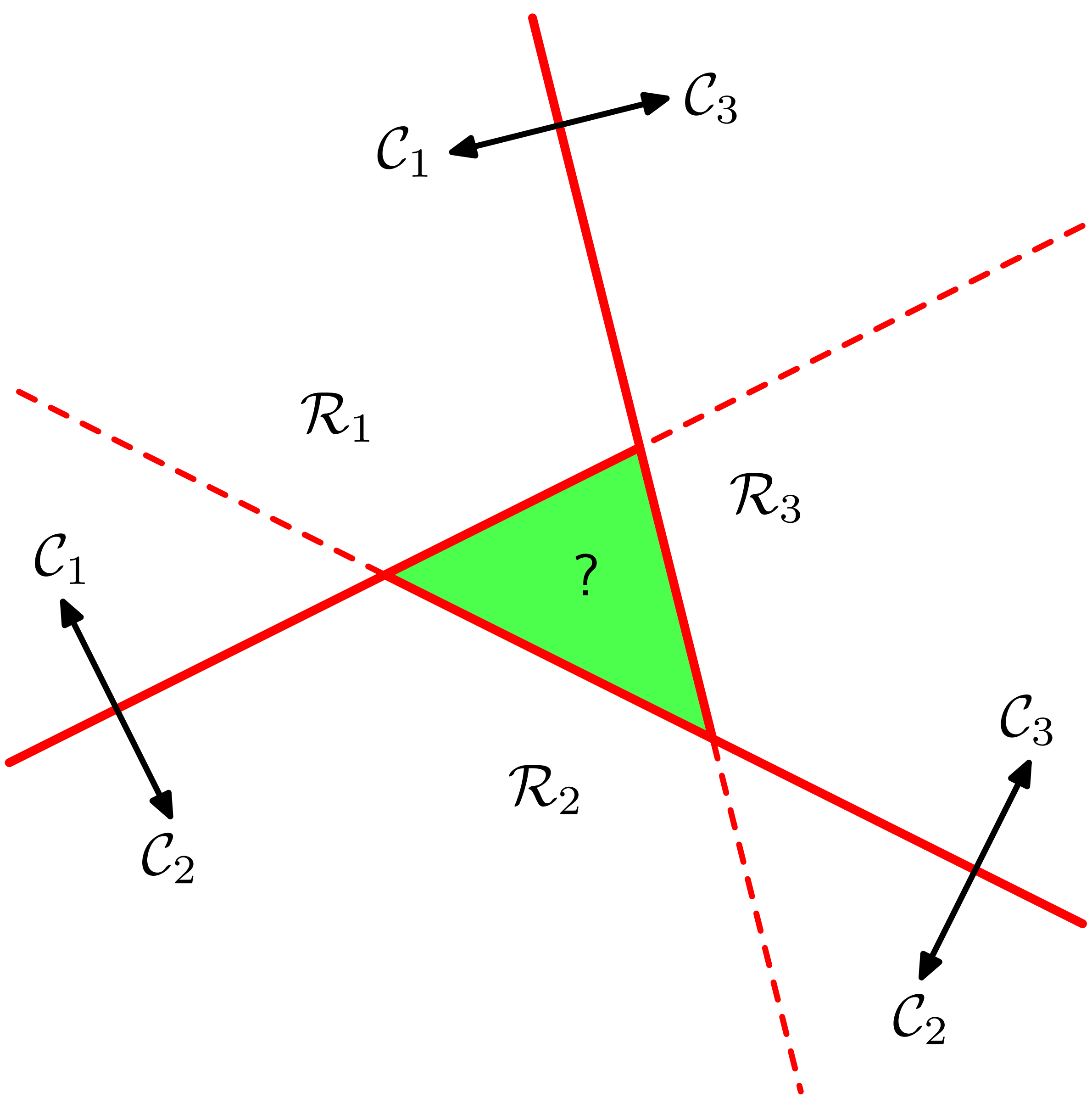
- So w_0 determines the location of the decision surface.

slide by Ce Liu

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.





slide by Ce Liu

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 $y_k(\mathbf{x}) = y_j(\mathbf{x})$

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

slide by Ce Liu

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

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

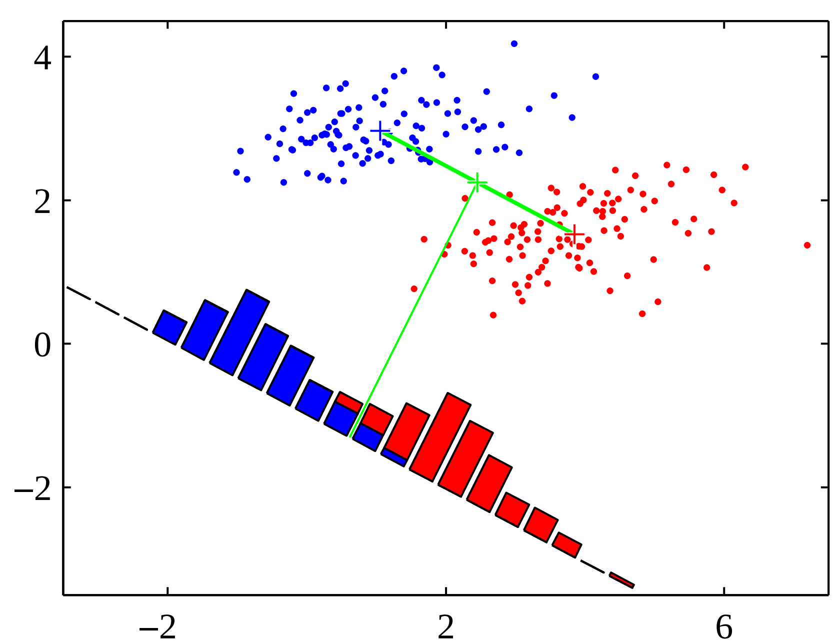
$$y = \mathbf{w}^T \mathbf{x}$$

- The mean vectors of the two classes

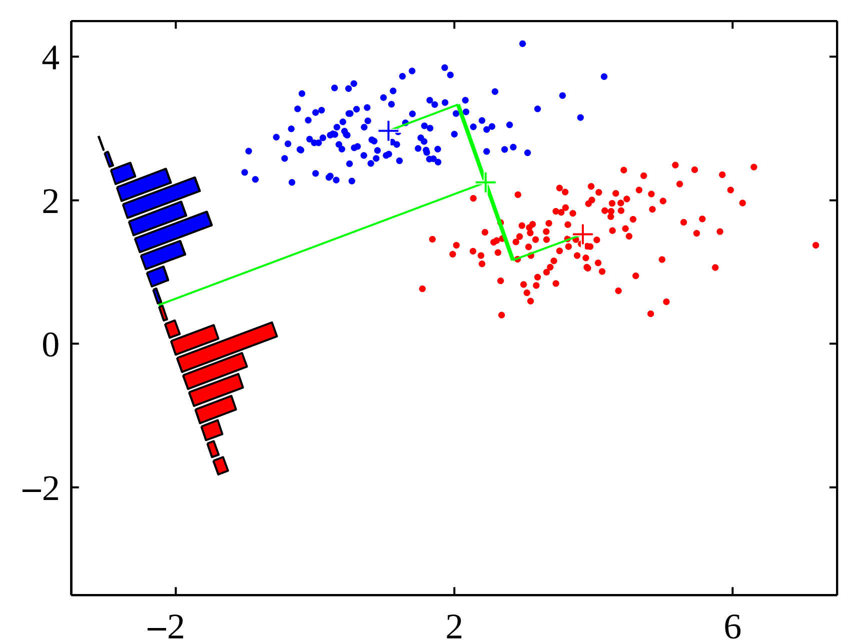
$$\mathbf{m}_1 = \frac{1}{N_1} \sum_{n \in \mathcal{C}_1} \mathbf{x}_n, \quad \mathbf{m}_2 = \frac{1}{N_2} \sum_{n \in \mathcal{C}_2} \mathbf{x}_n$$

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

slide by Ce Liu

Last time… **Linear classification**

$$\underbrace{\boxed{f(x,W)}}_{10\text{x}1} = \underbrace{\boxed{W}}_{10\text{x}3072}\overbrace{\boxed{x}}^{3072\text{x}1} \quad \underbrace{\boxed{(+b)}}_{10\text{x}1}$$

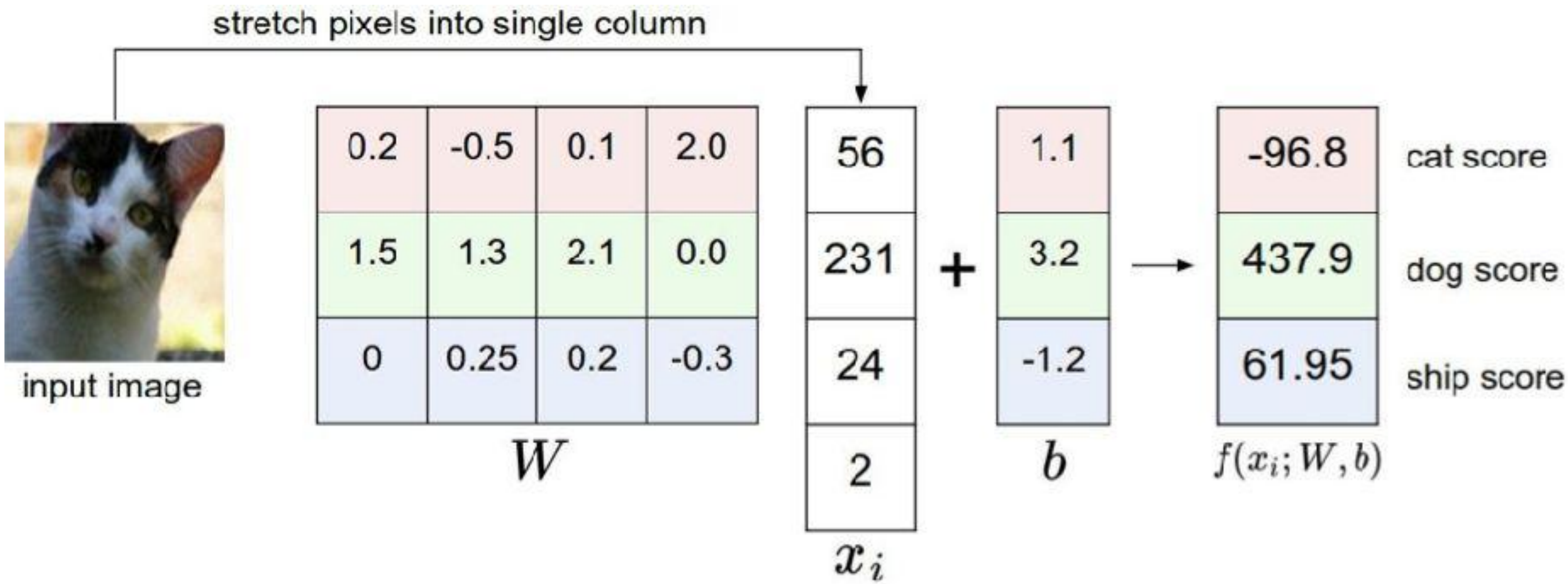
[32x32x3]
array of numbers 0...1

10 numbers, indicating class scores

parameters, or "weights"

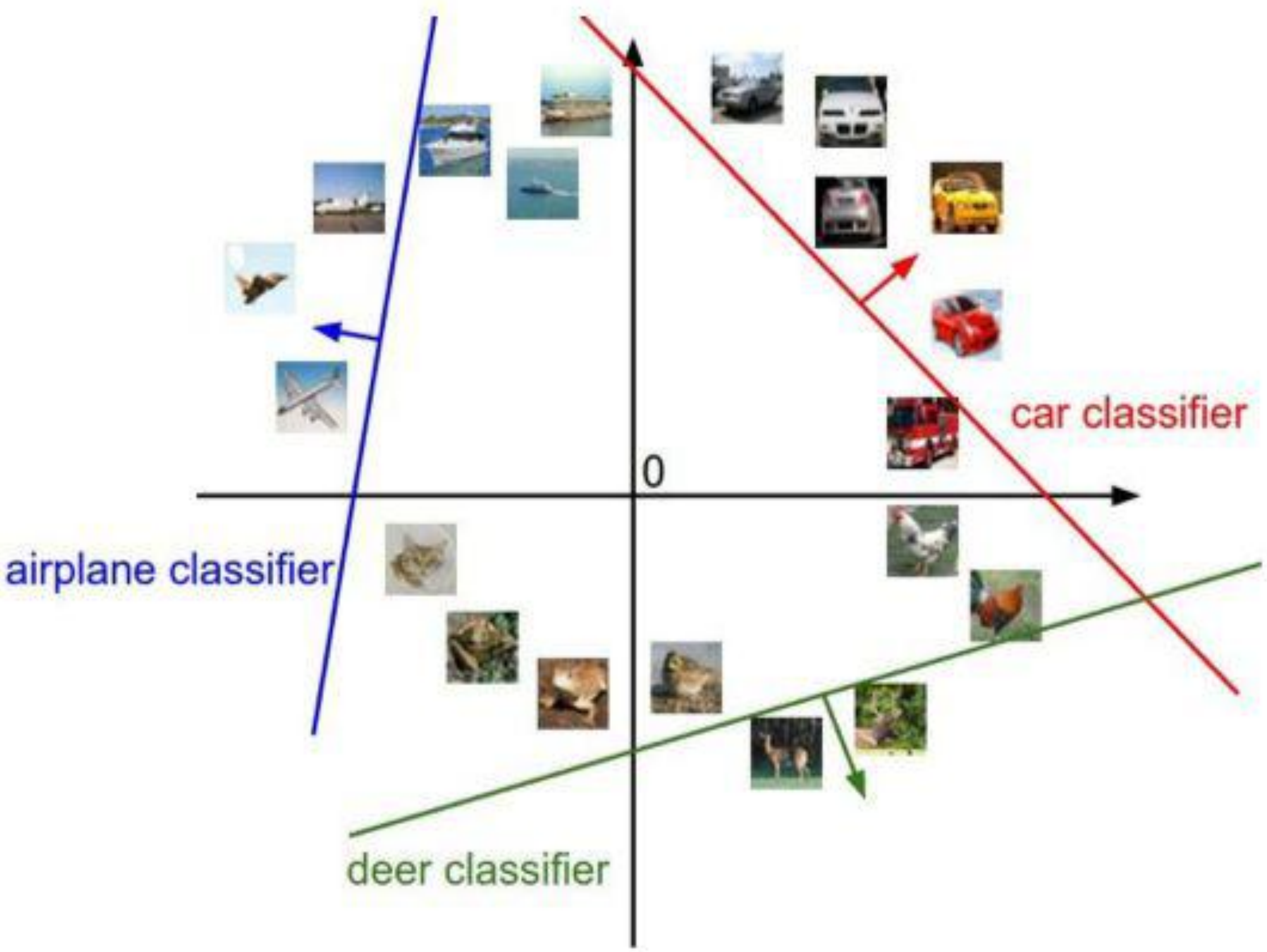
slide by Fei-Fei Li & Andrej Karpathy & Justin Johnson

Last time… **Linear classification**



slide by Fei-Fei Li & Andrej Karpathy & Justin Johnson

Last time... **Linear classification**

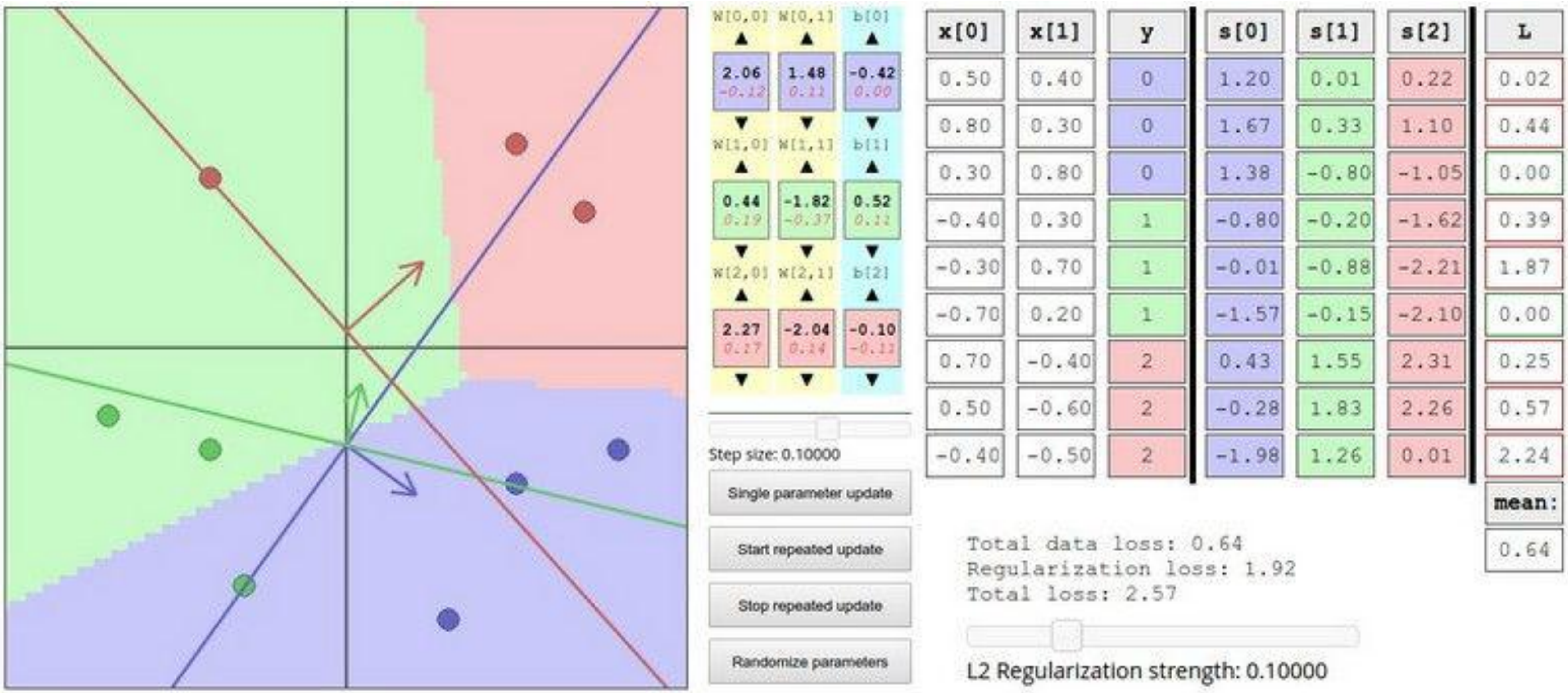


$$f(x_i, W, b) = Wx_i + b$$

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

slide by Fei-Fei Li & Andrej Karpathy & Justin Johnson

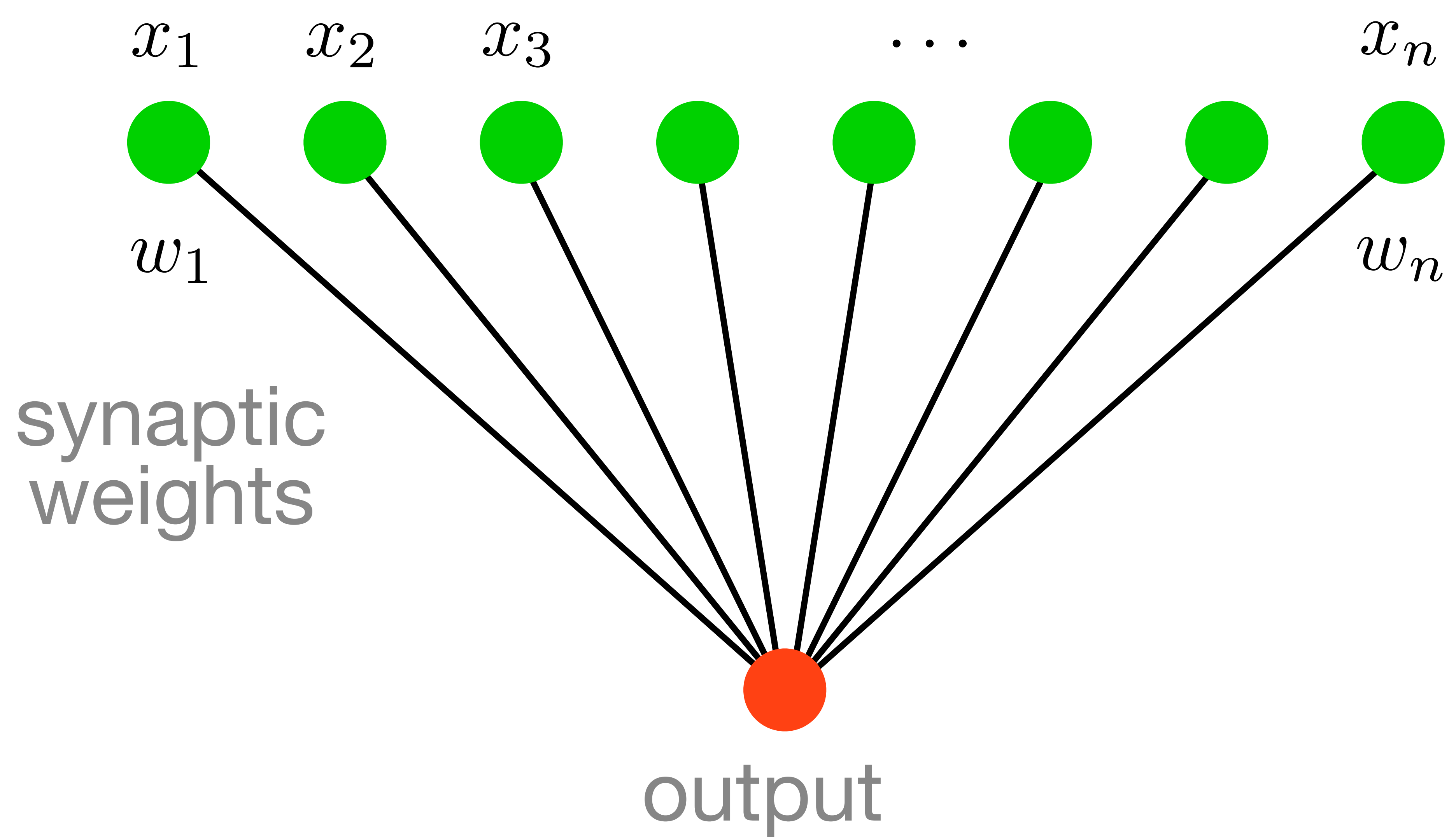
Interactive web demo time....



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

slide by Fei-Fei Li & Andrej Karpathy & Justin Johnson

Last time… **Perceptron**



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

slide by Alex Smola

This Week

- Multi-layer perceptron
- Forward Pass
- Backward Pass

Introduction

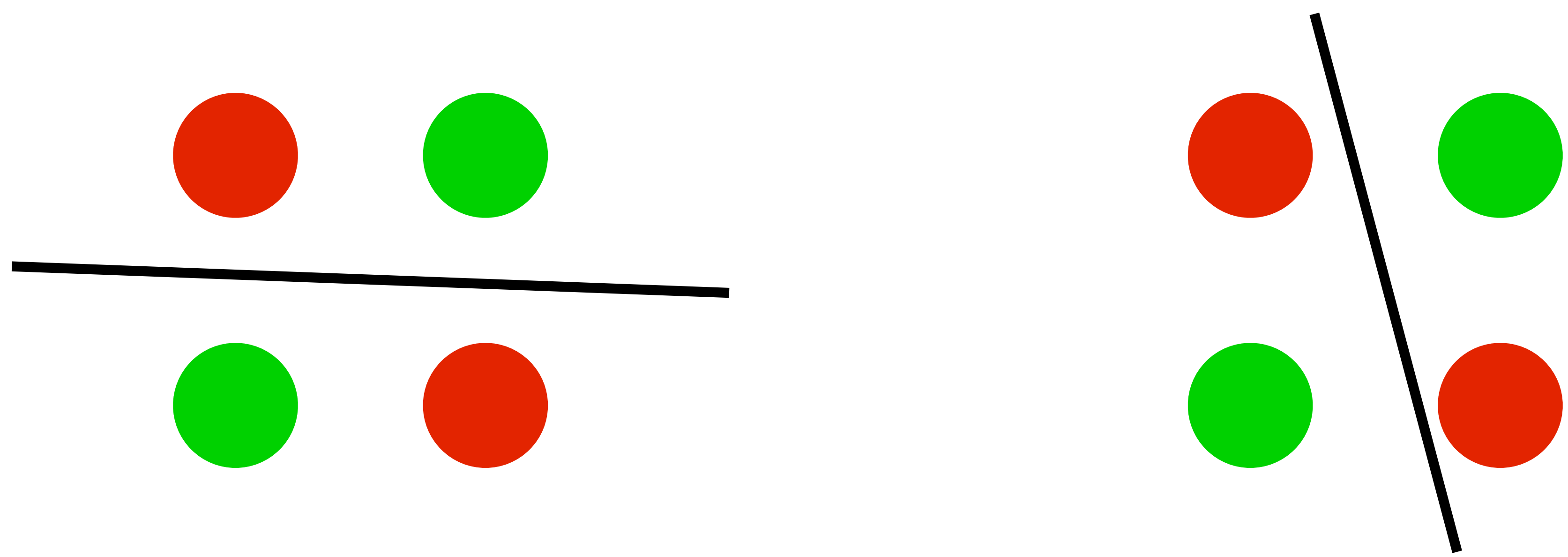
A brief history of computers

	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

- Data grows at higher exponent
- Moore's law (silicon) vs. Kryder's law (disks)
- Early algorithms data bound, now CPU/RAM bound

slide by Alex Smola

Not linearly separable data



- Some datasets are **not linearly separable**!
  - e.g. XOR problem
- Nonlinear separation is trivial

slide by Alex Smola

Addressing non-linearly separable data

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

slide by Dhruv Batra

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 **w**
  - As easy to learn
  - Data is linearly separable in higher dimensional spaces
  - Express via kernels

slide by Dhruv Batra

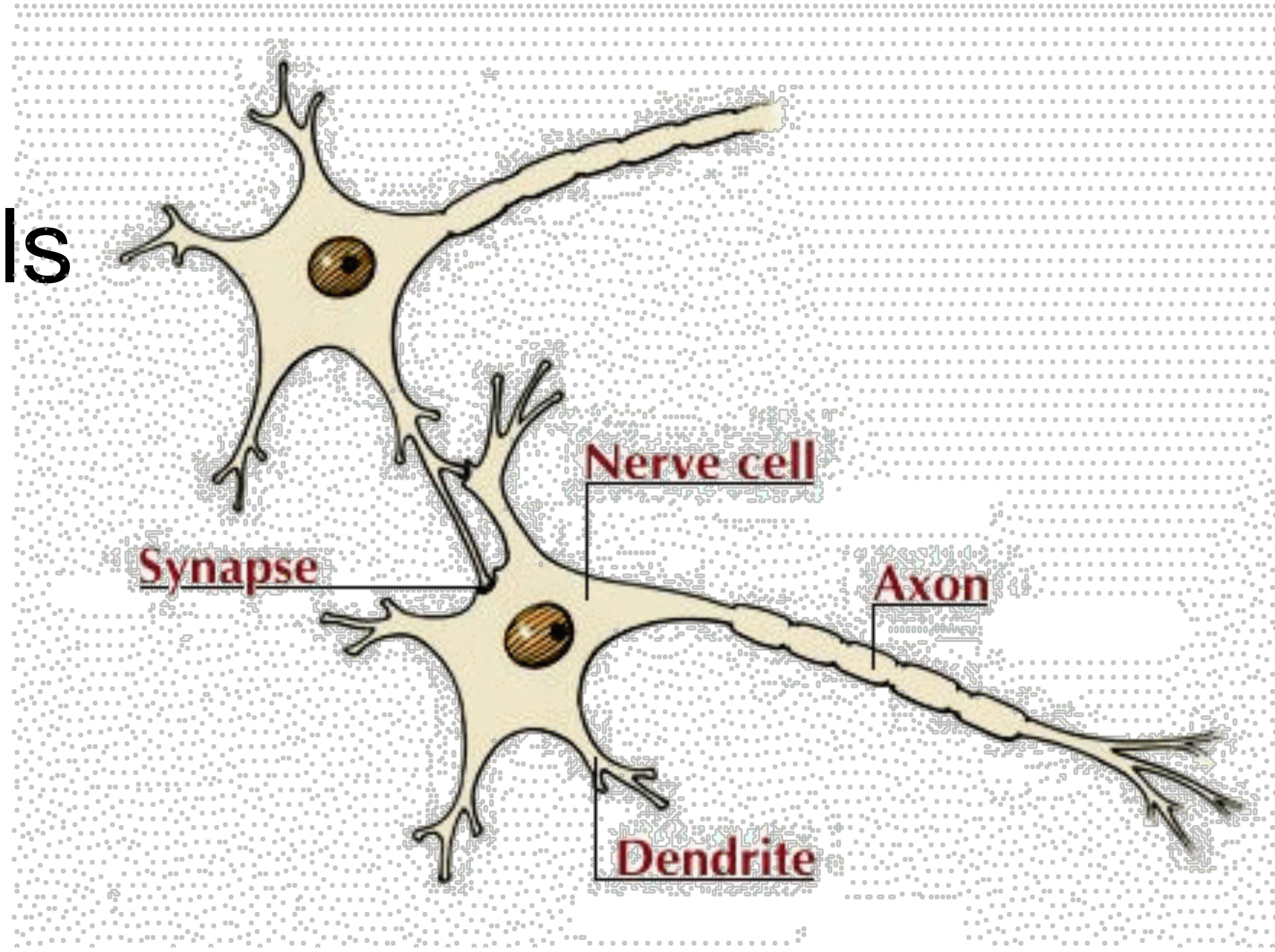
Option 2 — Non-linear classifiers

- Choose a classifier $h_{\mathbf{w}}(\mathbf{x})$ that is non-linear in parameters **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

slide by Dhruv Batra

Biological Neurons



- Soma (CPU)
  Cell body - combines signals

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

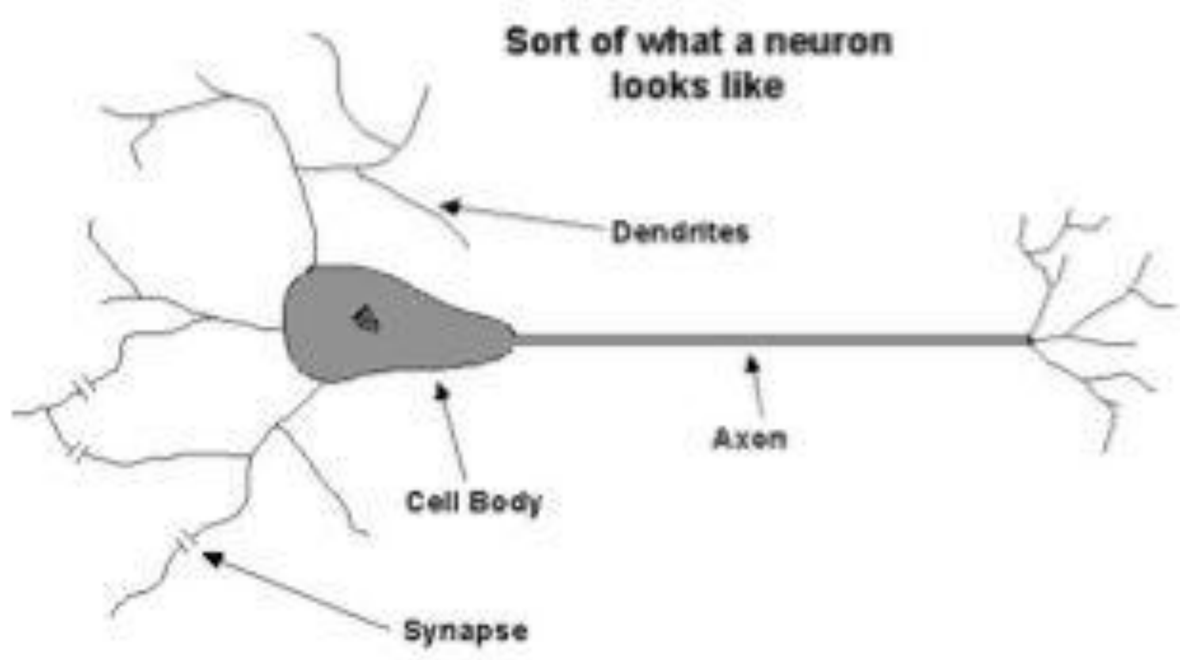
- Synapse (interface)
  Interface and **parameter store** between neurons

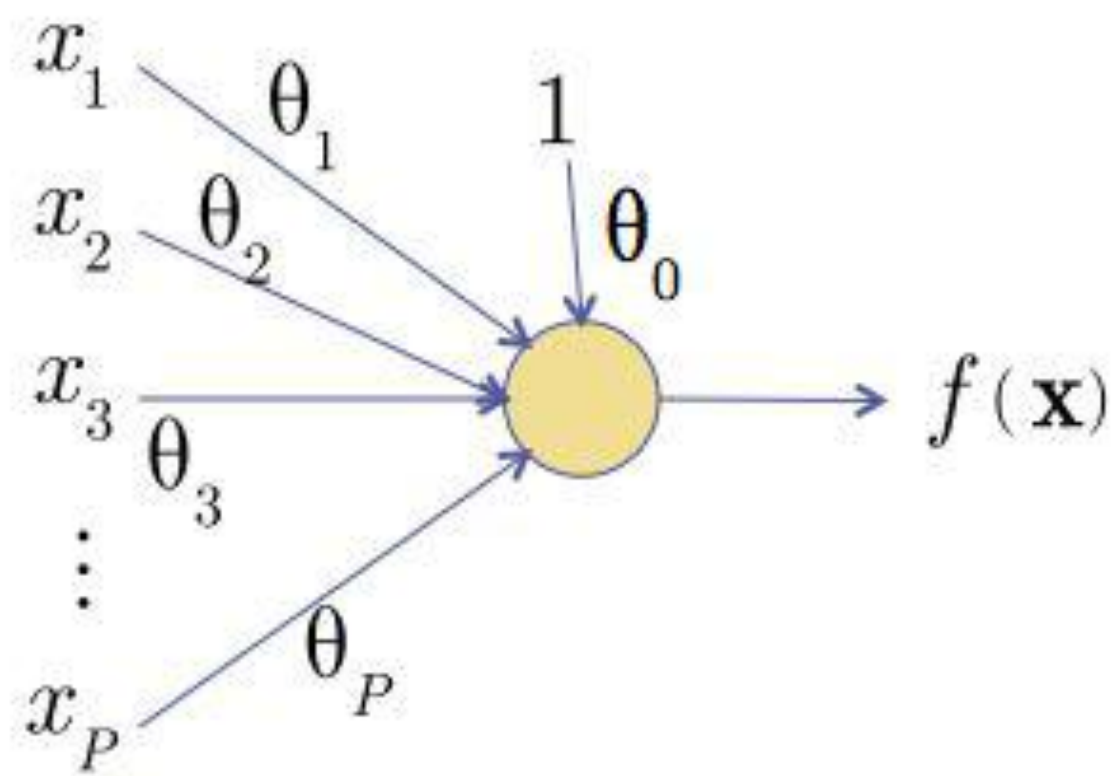
- Axon (cable)
  May be up to 1m long and will transport the activation signal to neurons at different locations

slide by Alex Smola

Recall: The Neuron Metaphor

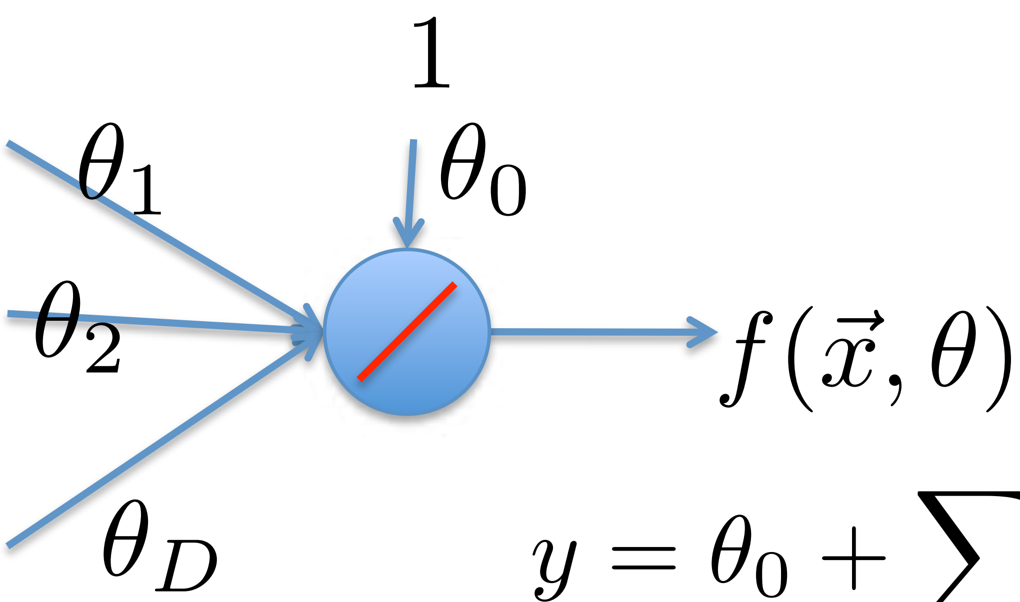
- Neurons
  - accept information from multiple inputs,
  - transmit information to other neurons.
- Multiply inputs by weights along edges
- Apply some function to the set of inputs at each node





slide by Dhruv Batra

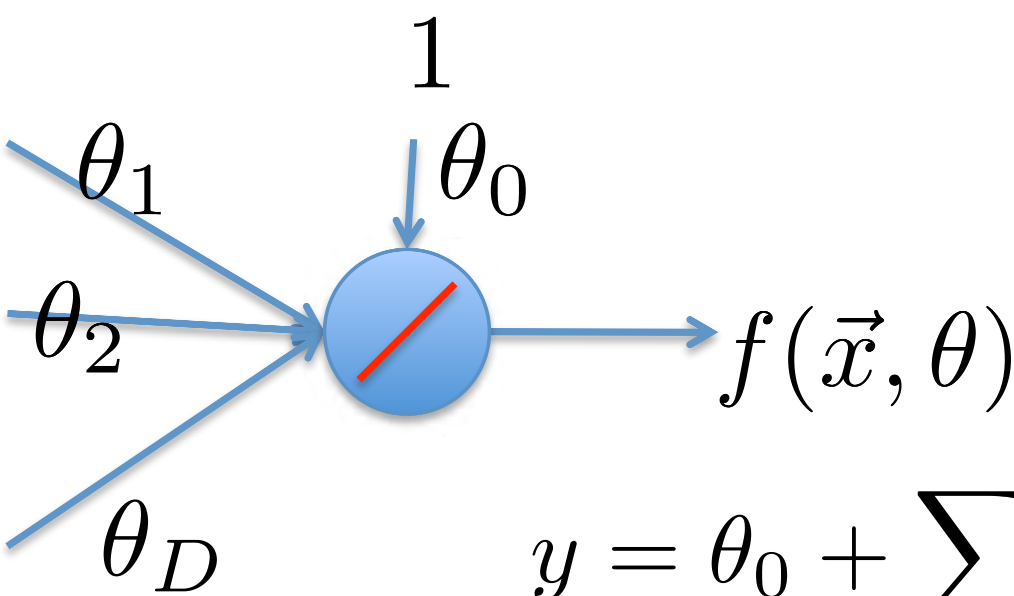
Types of Neuron



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

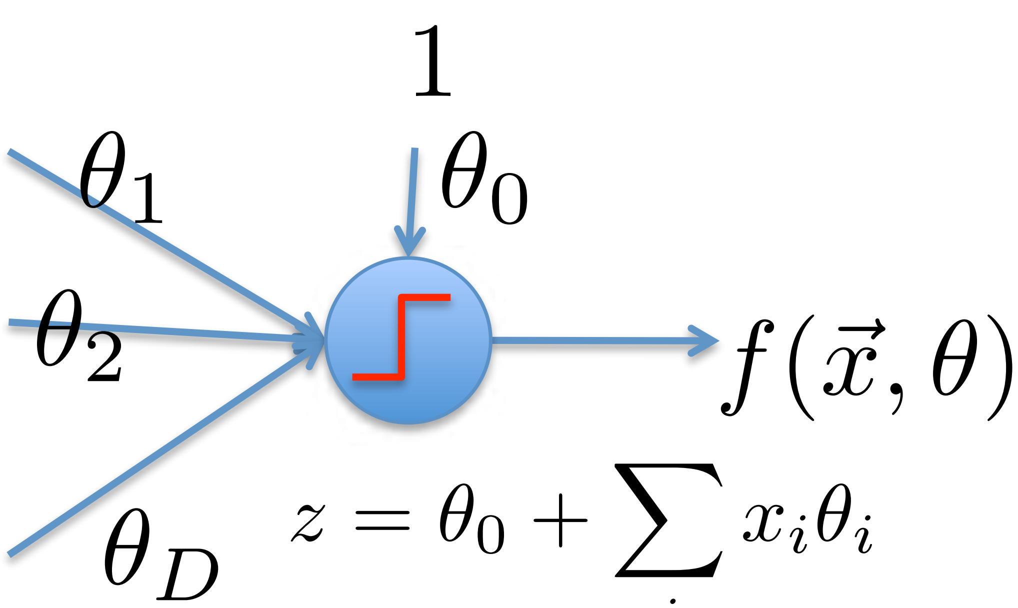
Linear Neuron

Types of Neuron



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

Linear Neuron

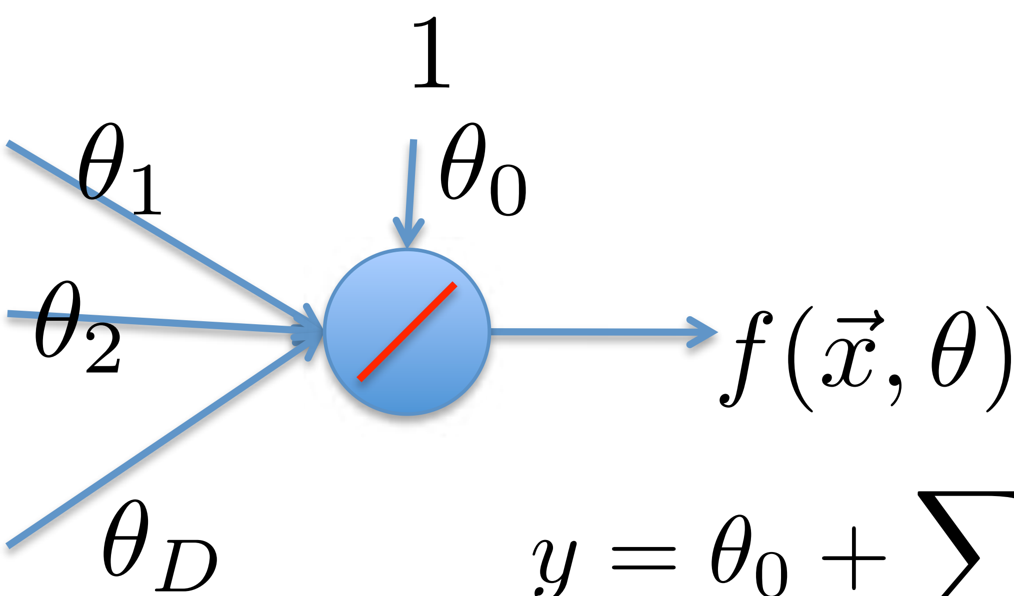


$$z = \theta_0 + \sum_i x_i \theta_i$$

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

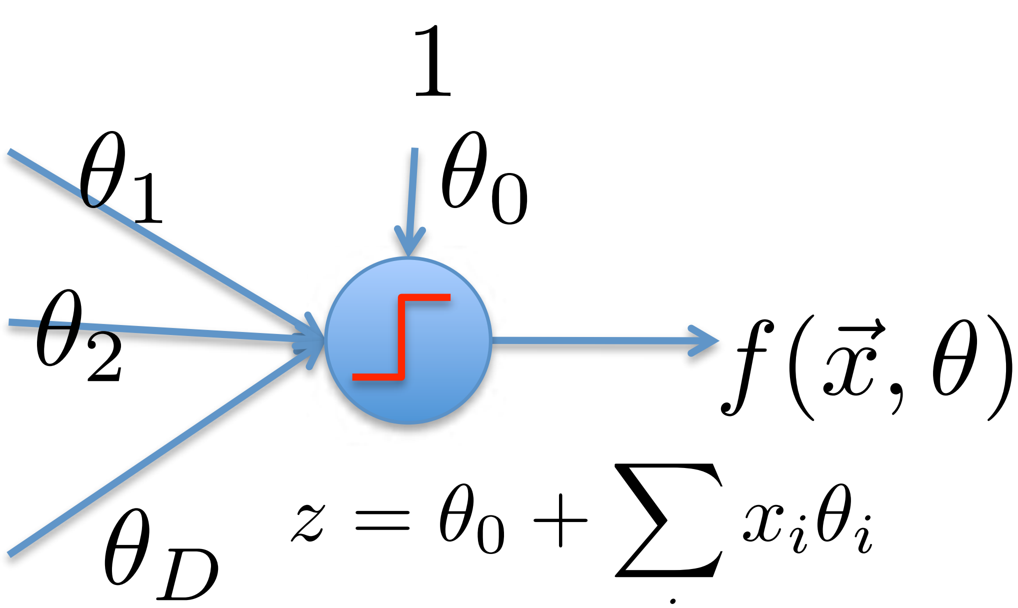
slide by Dhruv Batra

Types of Neuron



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

Linear Neuron

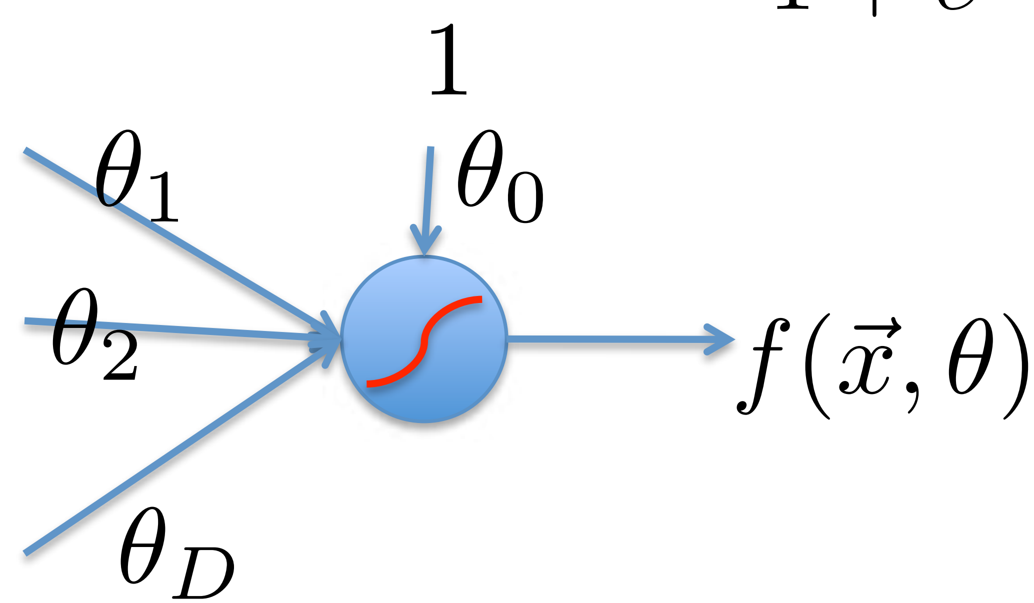


$$z = \theta_0 + \sum_i x_i \theta_i$$

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

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

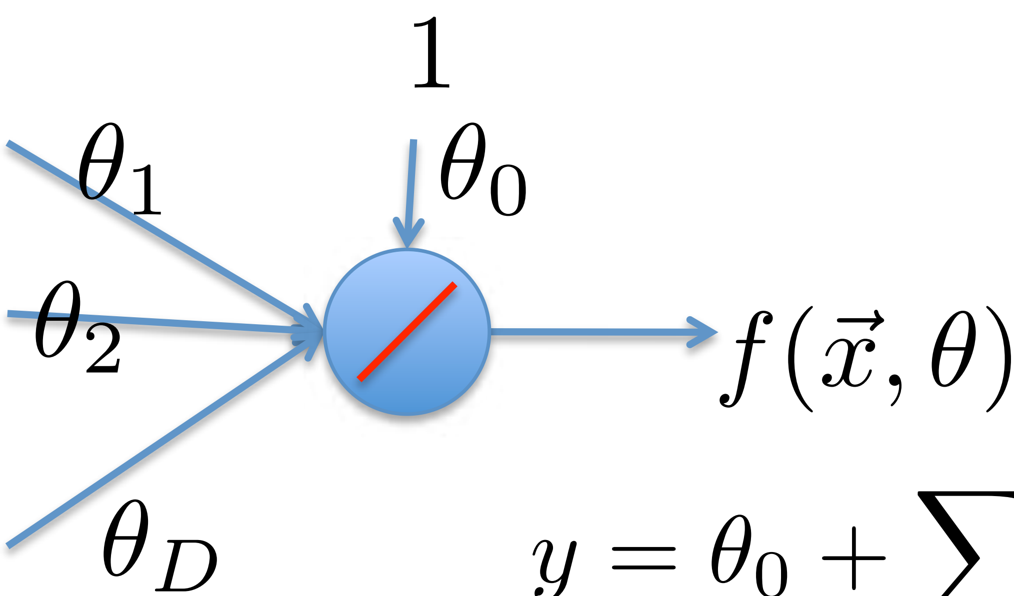
$$y = \frac{1}{1 + e^{-z}}$$



Logistic Neuron

slide by Dhruv Batra

Types of Neuron

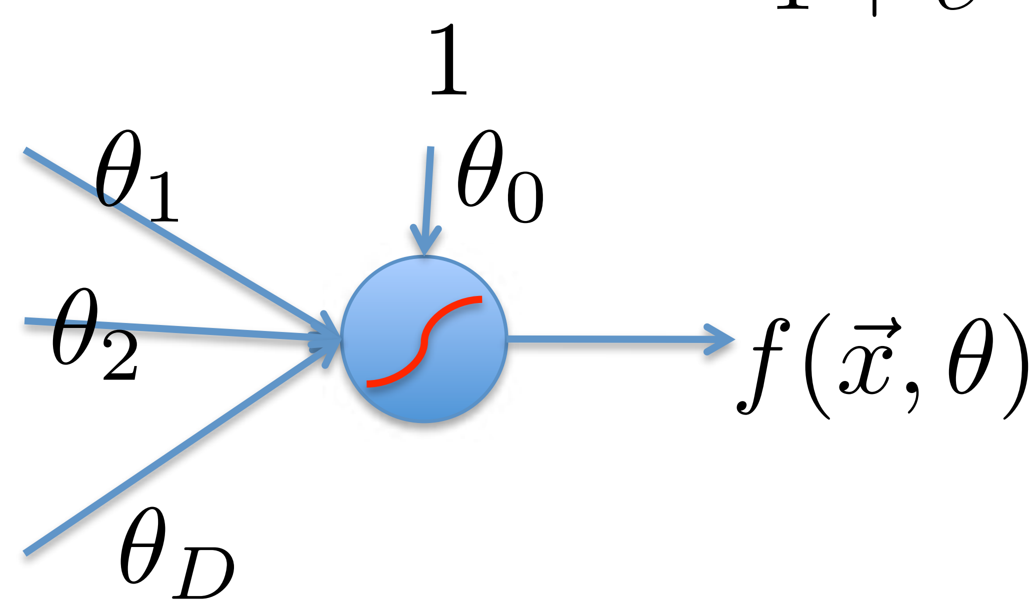


$$y = \theta_0 + \sum_i x_i \theta_i$$

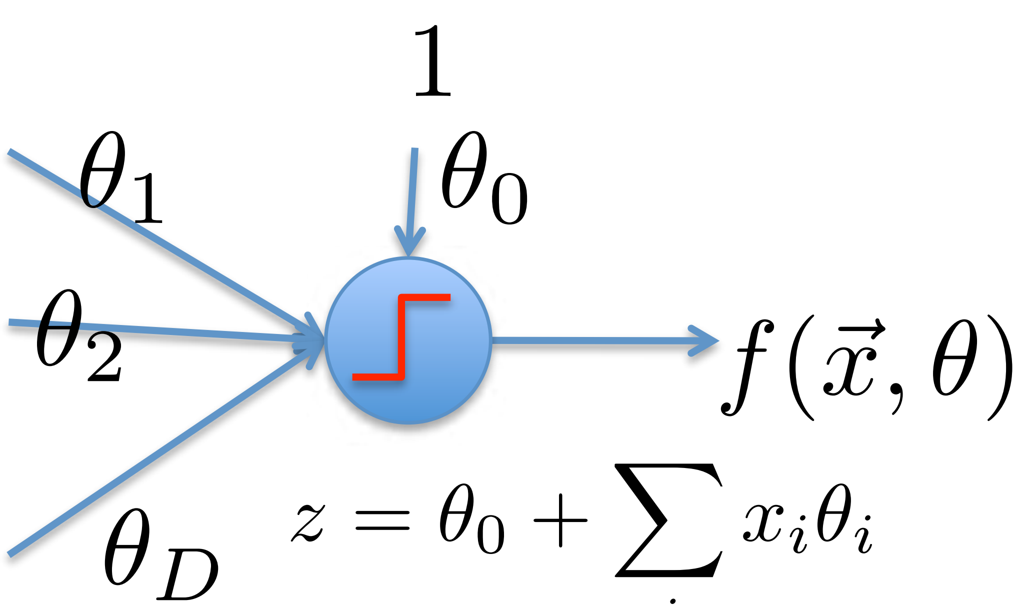
Linear Neuron

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

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



Logistic Neuron



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

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

- Potentially more. Requires a convex loss function for gradient descent training.

slide by Dhruv Batra

Limitation

- A single “neuron” is still a linear decision boundary

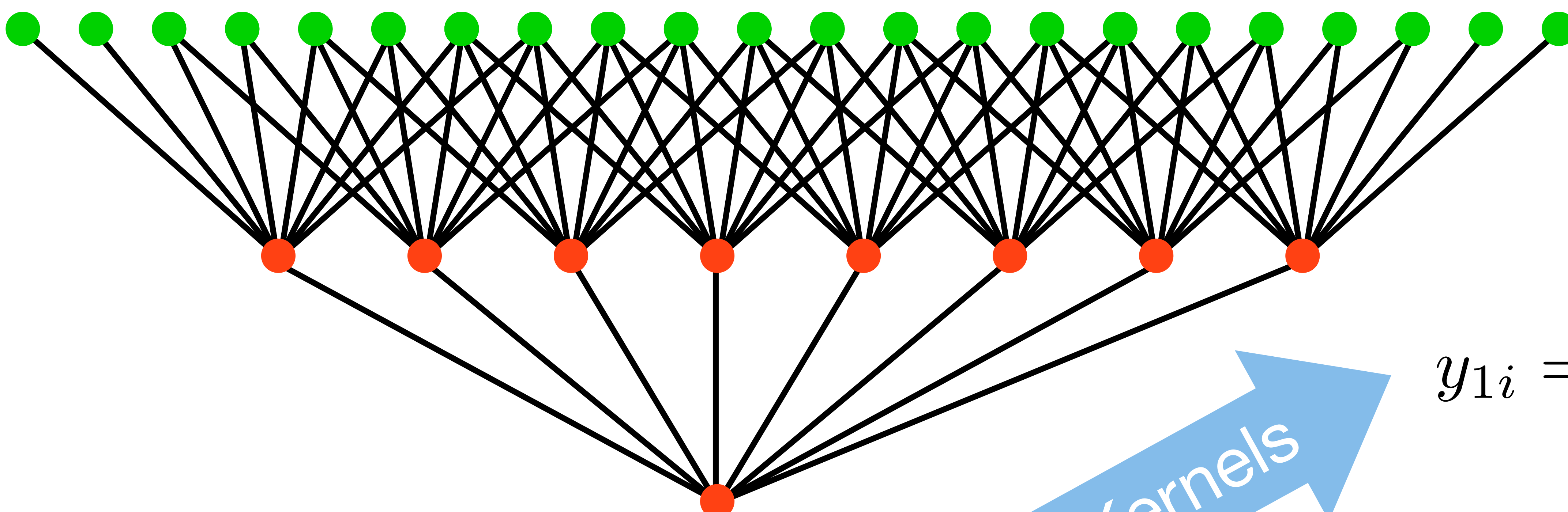
- What to do?

- Idea: Stack a bunch of them together!

slide by Dhruv Batra

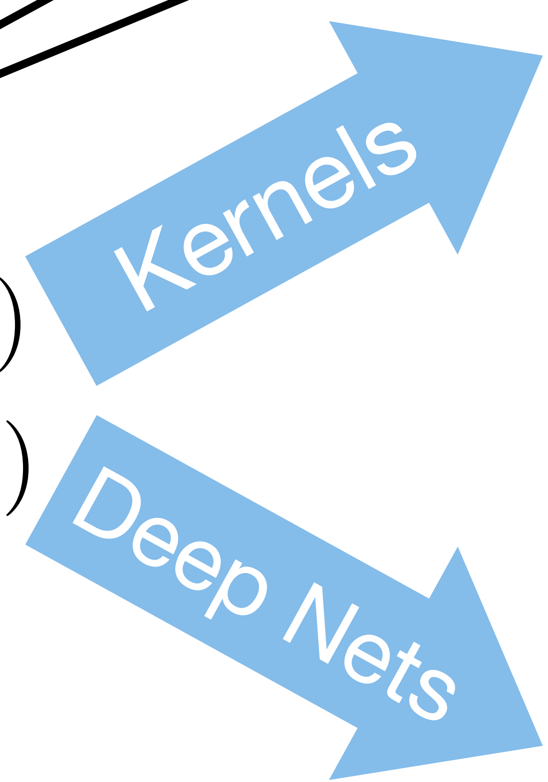
Nonlinearities via Layers

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



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

$$y_2(x) = \sigma(\langle w_2, y_1 \rangle)$$

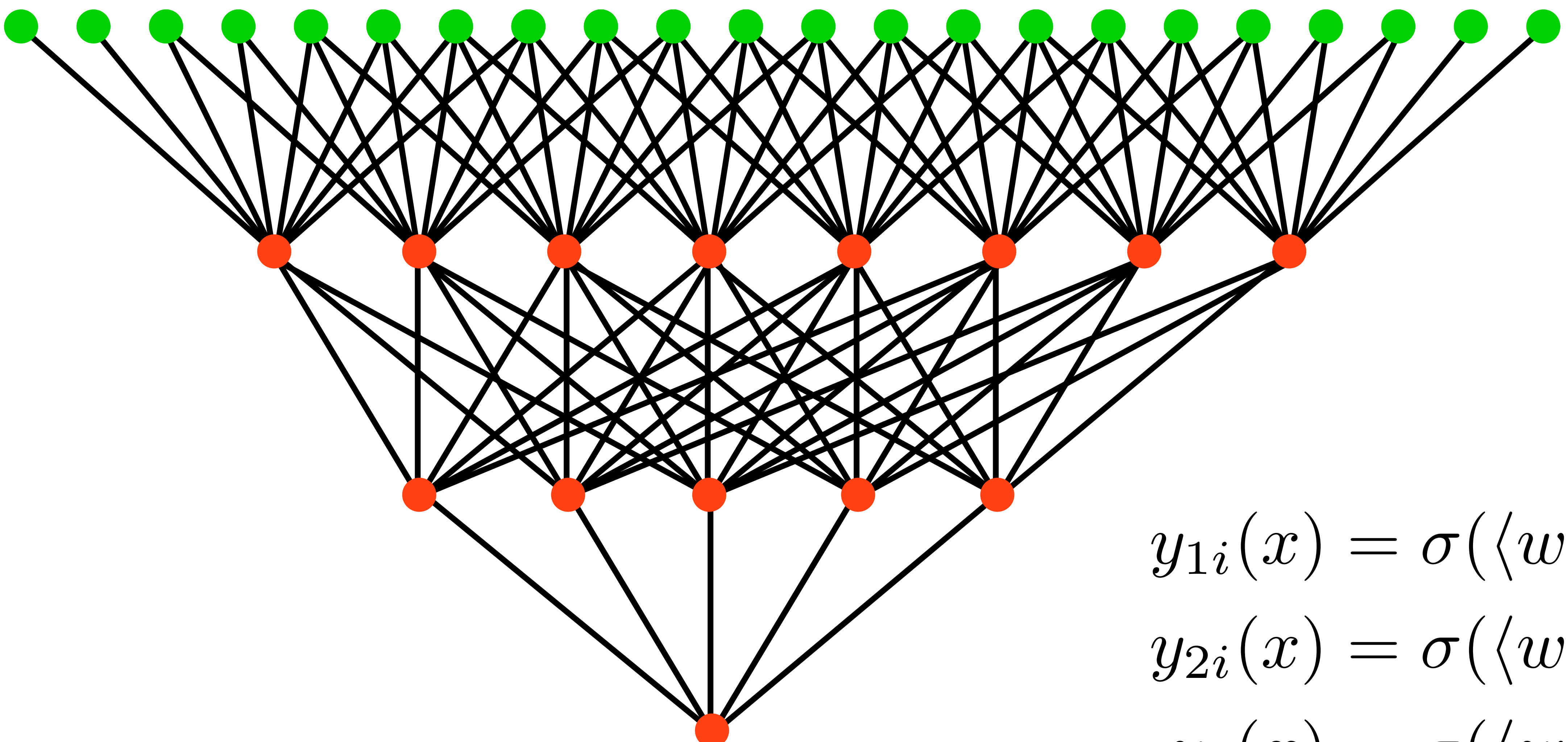


$$y_{1i} = k(x_i, x)$$

optimize all weights

slide by Alex Smola

Nonlinearities via Layers



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

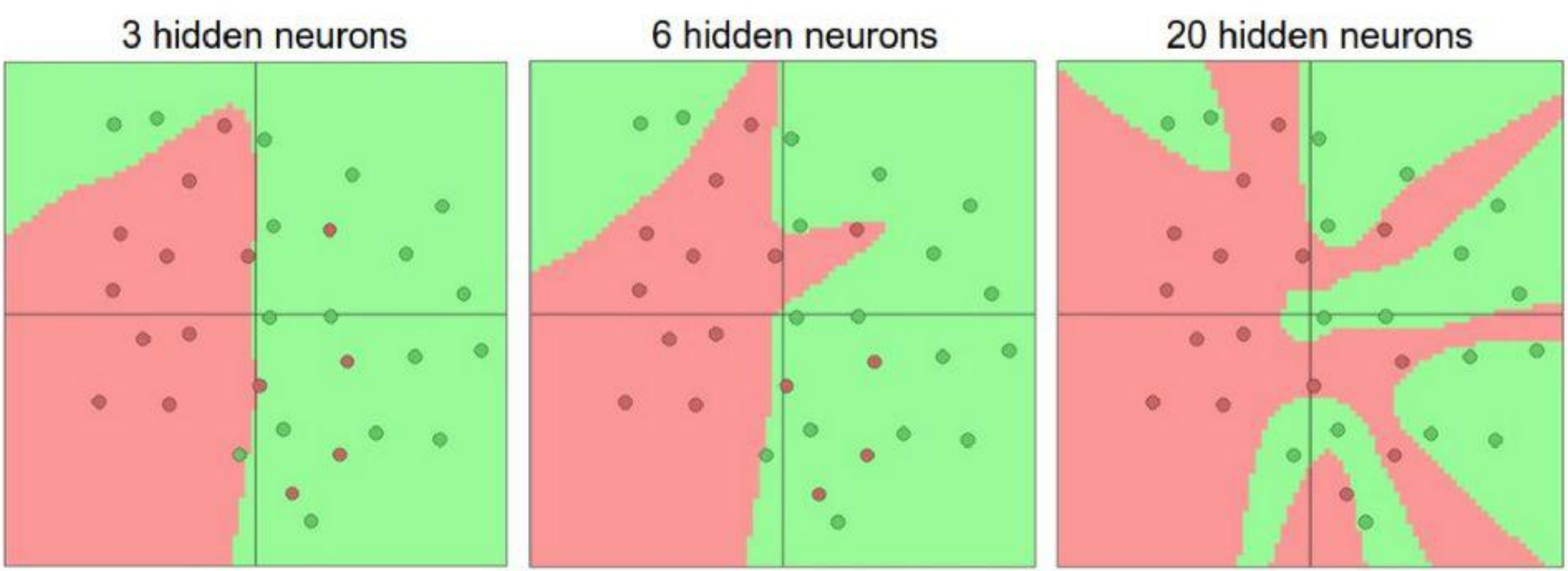
$$y_{2i}(x) = \sigma(\langle w_{2i}, y_1 \rangle)$$

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

slide by Alex Smola

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

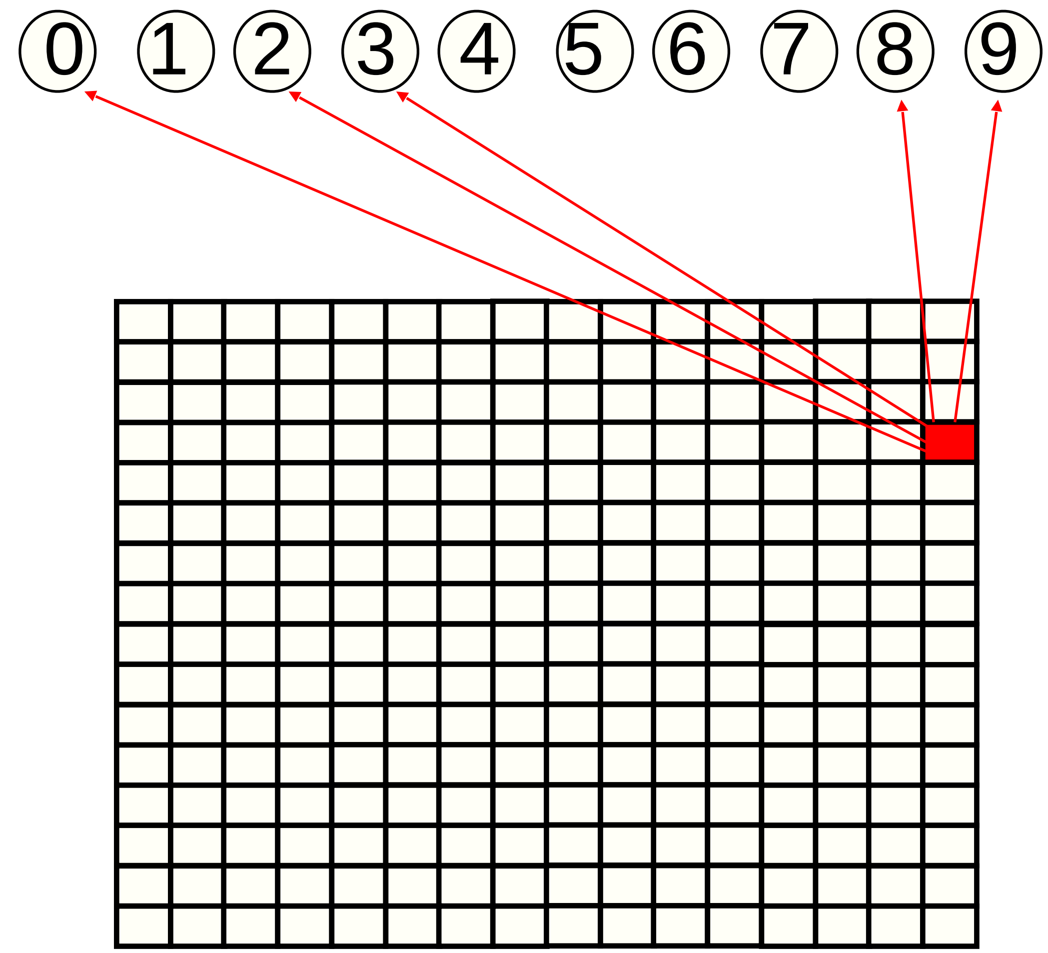


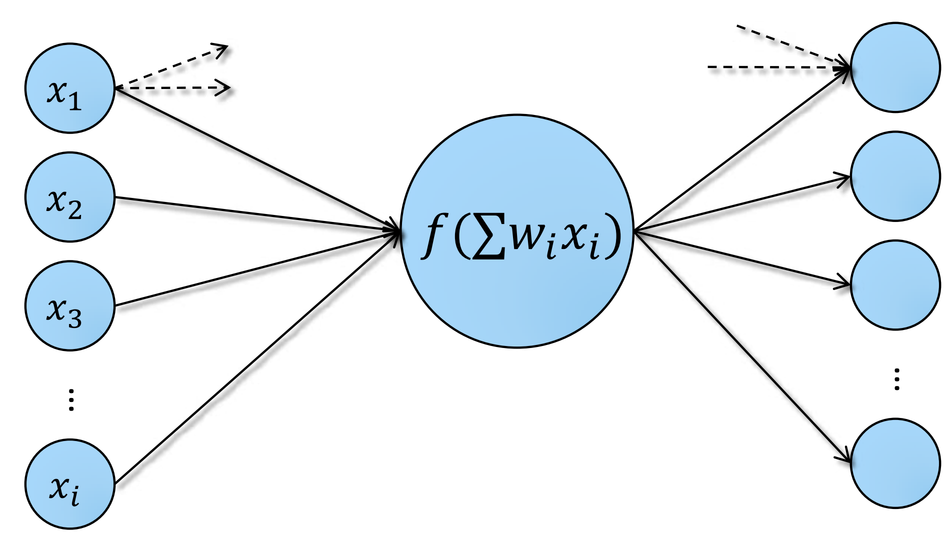
- The capacity of the network increases with more hidden units and more hidden layers

slide by Raquel Urtasun, Richard Zemel, Sanja Fidler

A simple example

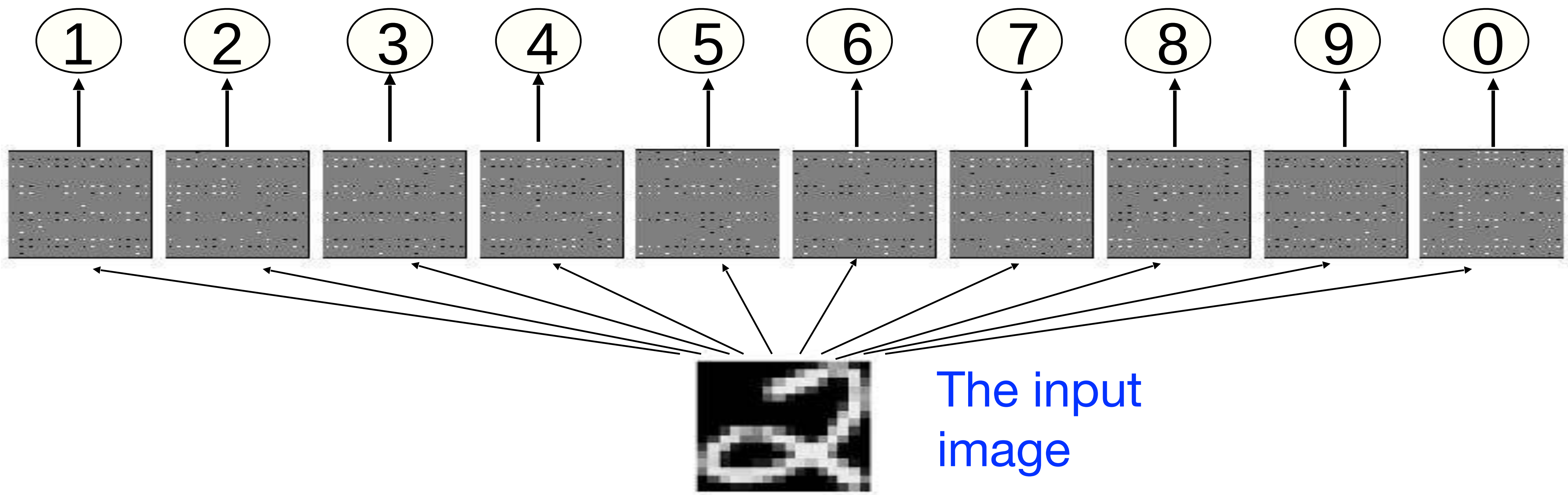
- Consider a neural network with two layers of neurons.
  - neurons in the top layer represent known shapes.
  - neurons in the bottom layer represent pixel intensities.
- A pixel gets to vote if it has ink on it.
  - Each inked pixel can vote for several different shapes.
- The shape that gets the most votes wins.





slide by Geoffrey Hinton

How to display the weights

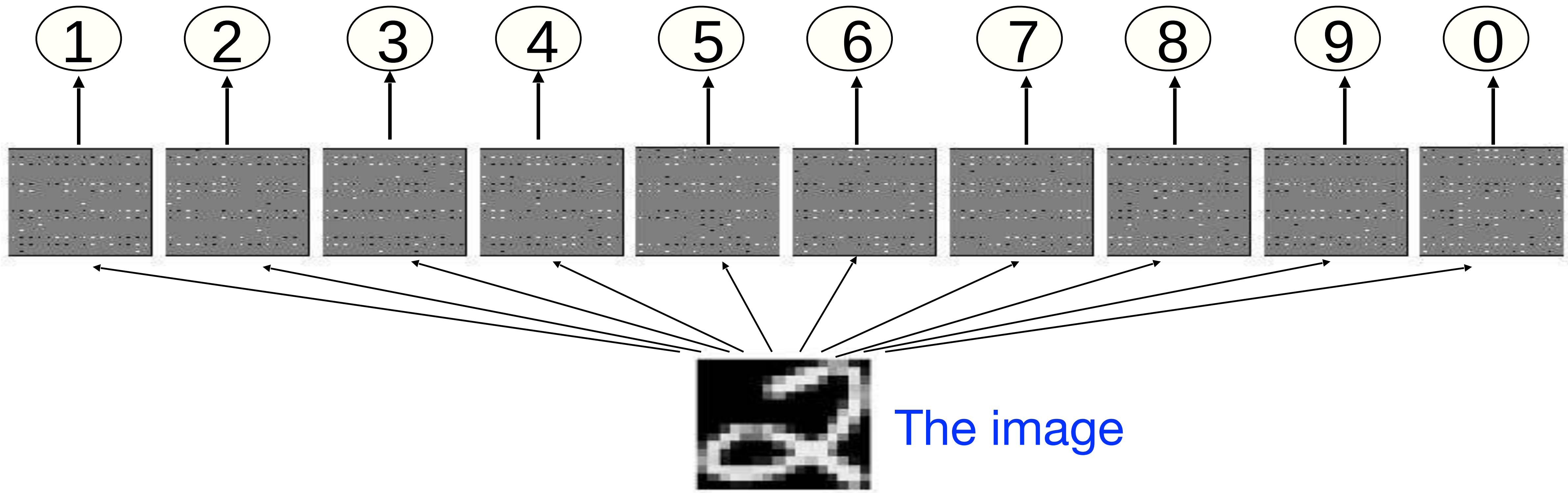


Give each output unit its own “map” of the input image and display the weight coming from each pixel in the location of that pixel in the map.

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

slide by Geoffrey Hinton

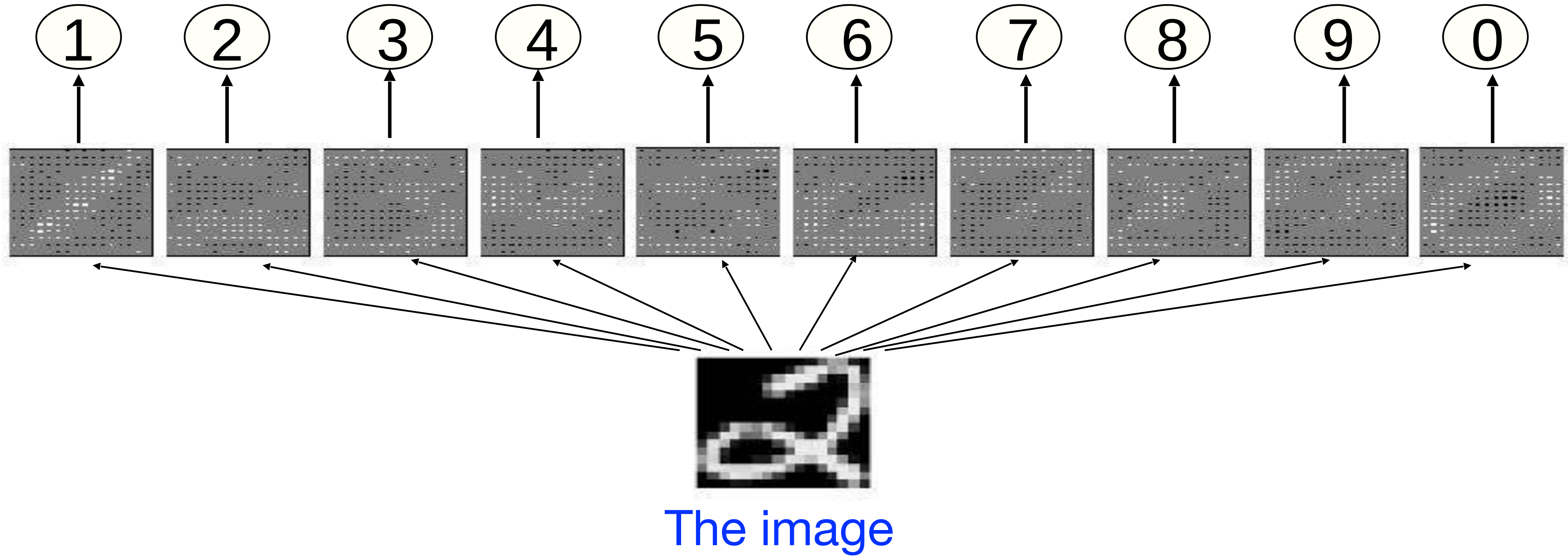
How to learn the weights



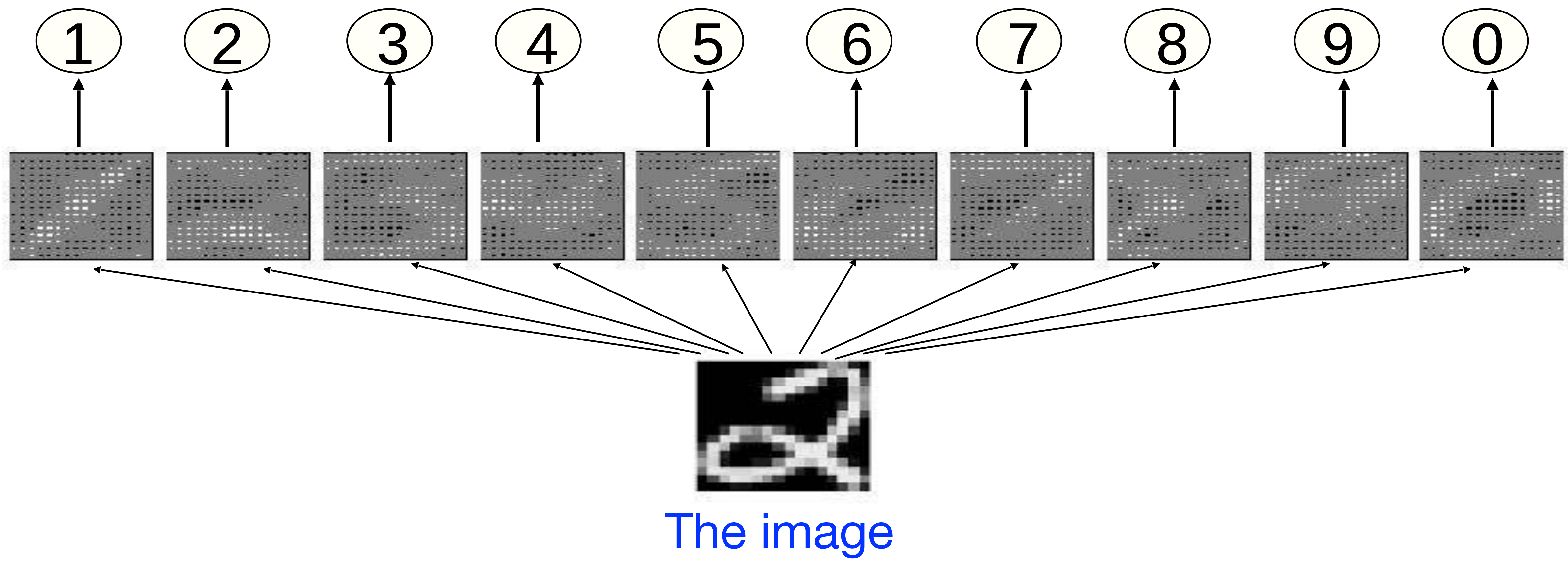
Show the network an image and increment the weights from active pixels to the correct class.

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

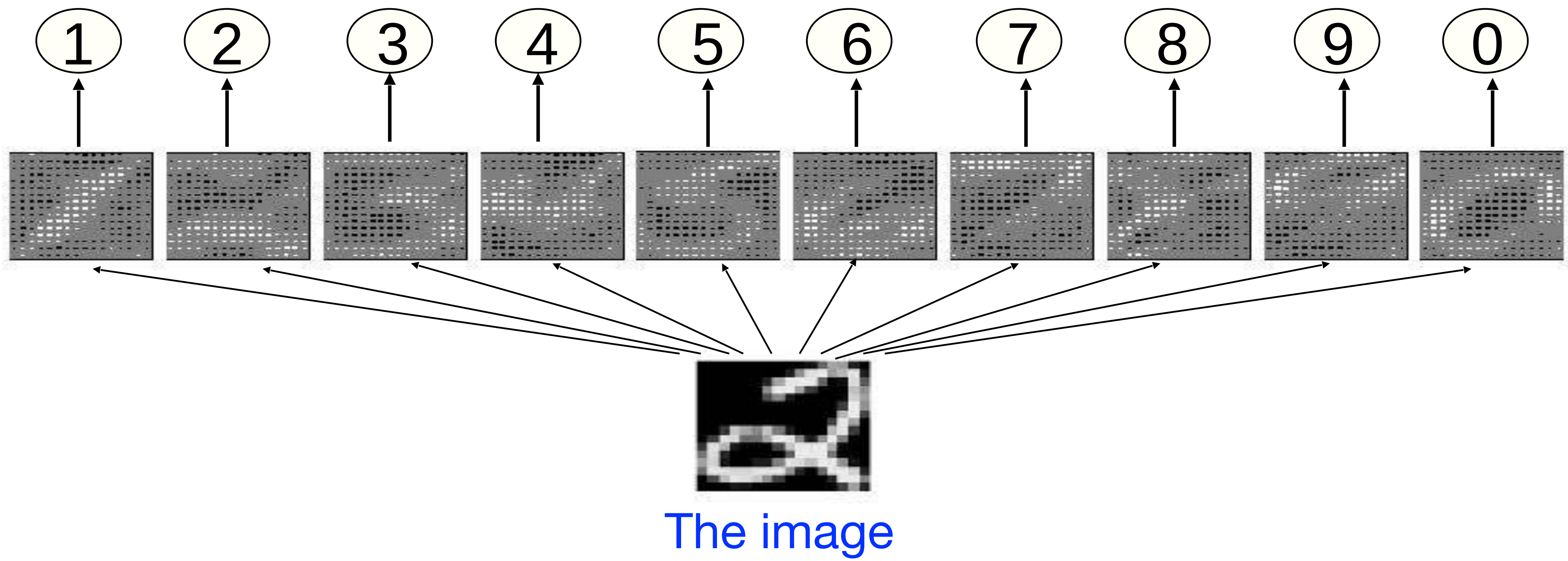
slide by Geoffrey Hinton



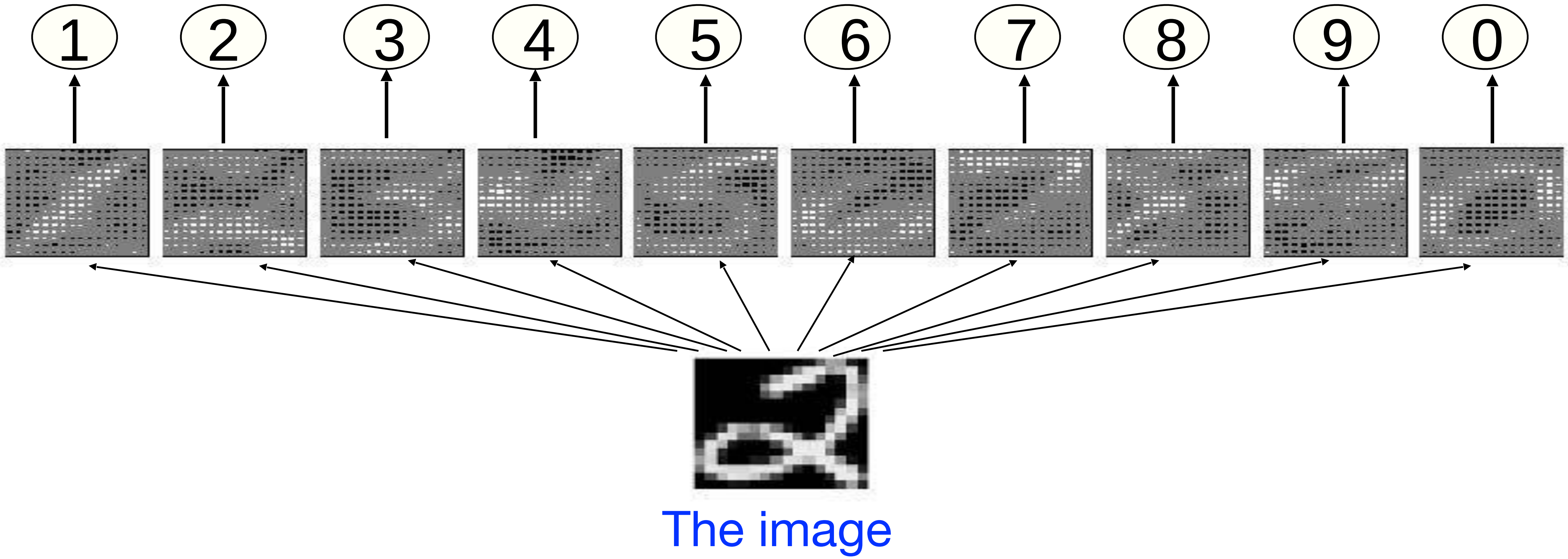
slide by Geoffrey Hinton



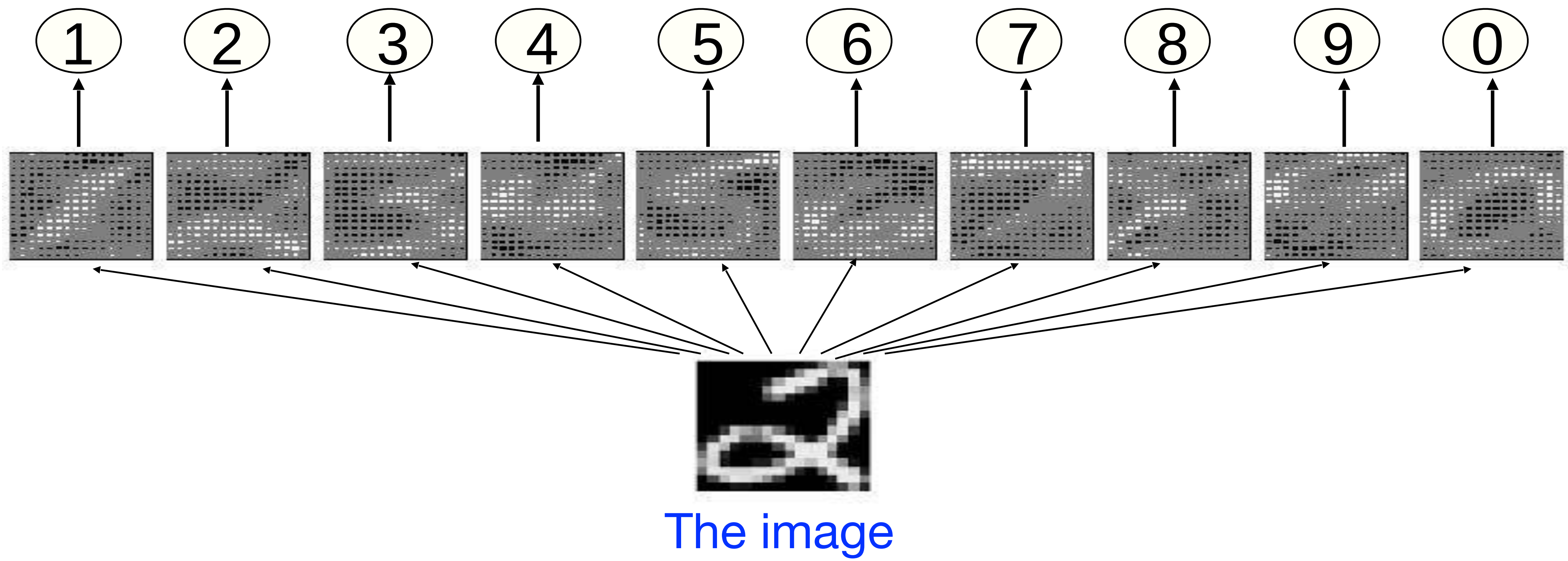
slide by Geoffrey Hinton



slide by Geoffrey Hinton

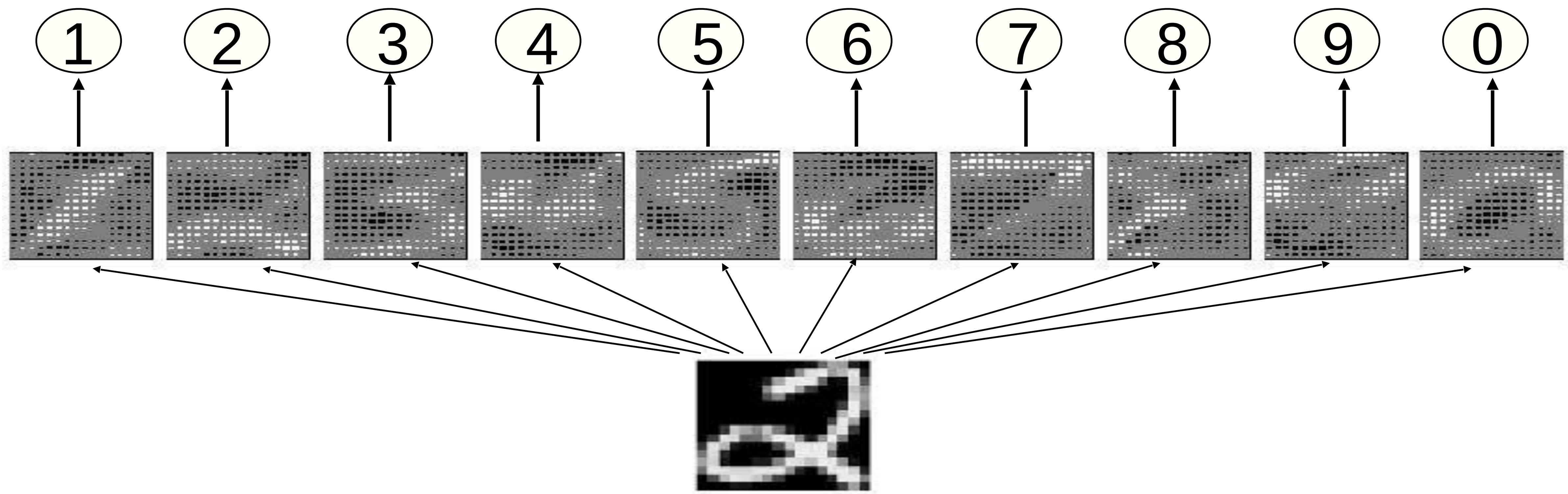


slide by Geoffrey Hinton



slide by Geoffrey Hinton

The learned weights



The details of the learning algorithm will be explained later.

slide by Geoffrey Hinton

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.

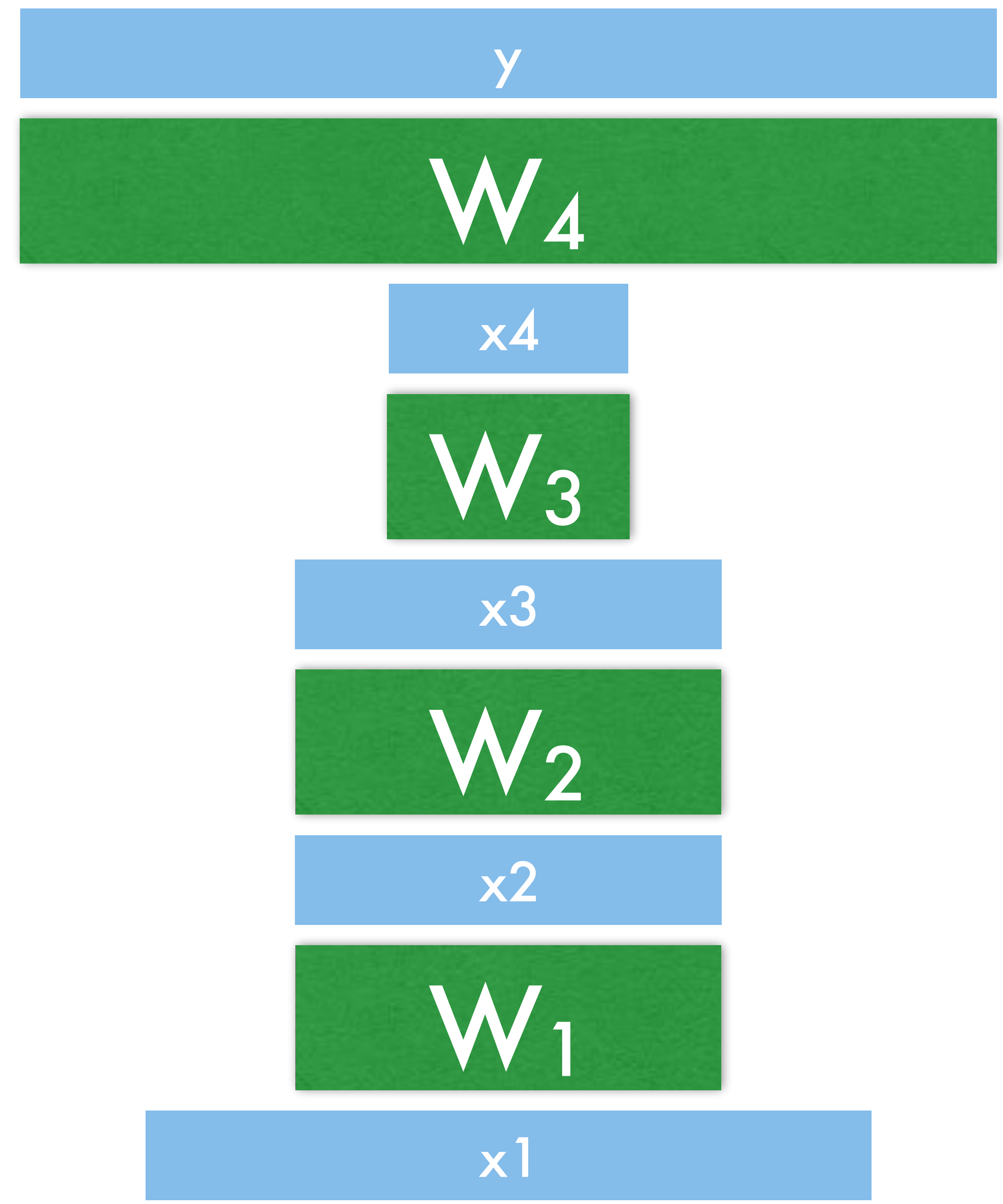
slide by Geoffrey Hinton

Multilayer Perceptron

- Layer Representation

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

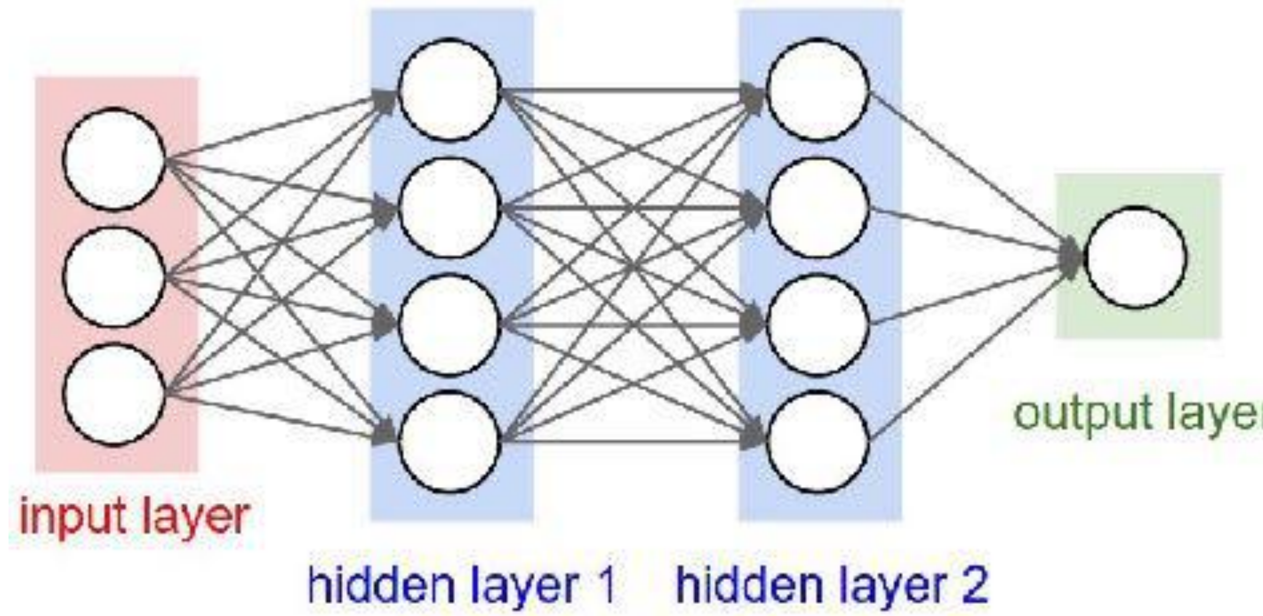
- (typically) iterate between linear mapping Wx and nonlinear function
- Loss function $l(y, y_i)$ to measure quality of estimate so far



slide by Alex Smola

Forward Pass

Forward Pass: What does the Network Compute?



- Output of the network can be written as:

$$
\begin{aligned}
h_j(\mathbf{x}) &= f(v_{j0} + \sum_{i=1}^{D} x_i v_{ji}) \\
o_k(\mathbf{x}) &= g(w_{k0} + \sum_{j=1}^{J} h_j(\mathbf{x}) w_{kj})
\end{aligned}
$$

(j indexing hidden units, k indexing the output units, D number of inputs)

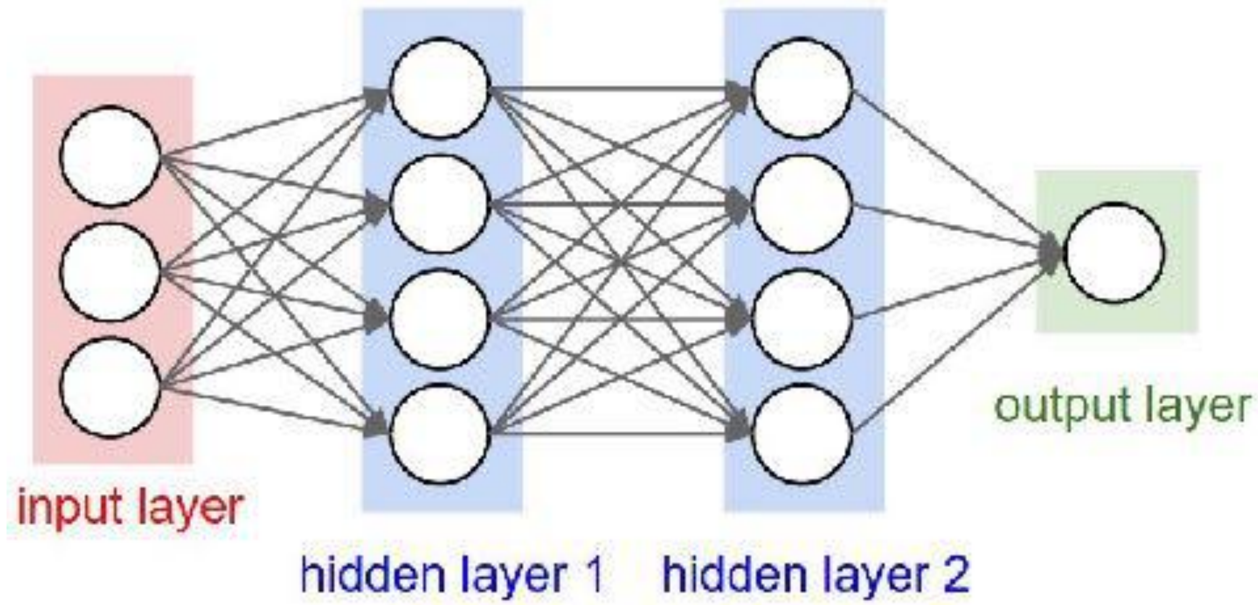
- Activation functions f , g : sigmoid/logistic, tanh, or rectified linear (ReLU)

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

slide by Raquel Urtasun, Richard Zemel, Sanja Fidler

Forward Pass in Python

- Example code for a forward pass for a 3-layer network in 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)
```

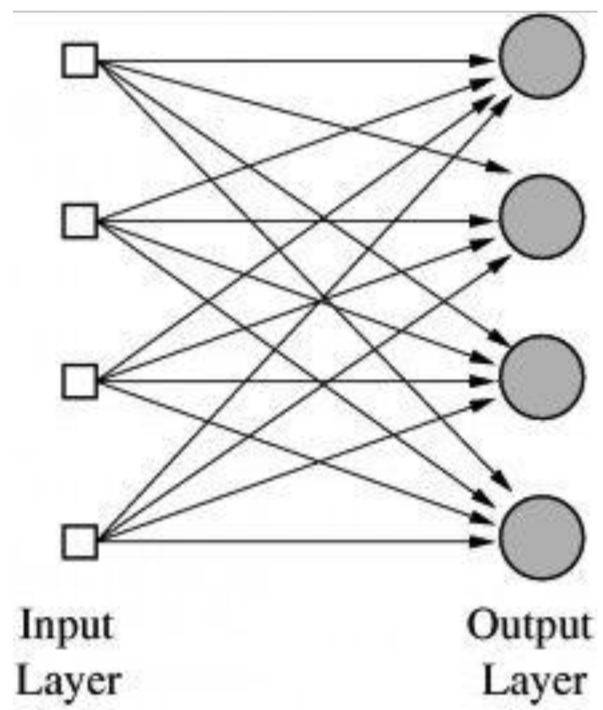
- Can be implemented efficiently using matrix operations
- Example above: W_1 is matrix of size 4×3, W_2 is 4×4. What about biases and W_3?

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

slide by Raquel Urtasun, Richard Zemel, Sanja Fidler

Special Case

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



- Network:

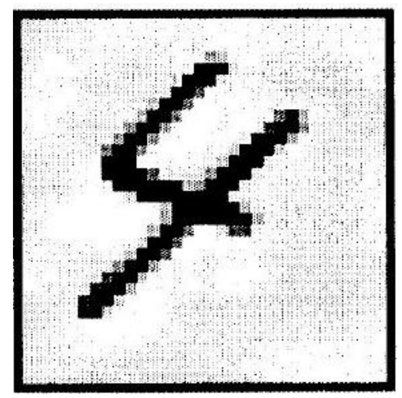
$$
\begin{aligned}
o_k(\mathbf{x}) &= \frac{1}{1+\exp(-z_k)} \\
z_k &= w_{k0} + \sum_{j=1}^{J} x_j w_{kj}
\end{aligned}
$$

- Logistic regression!

slide by Raquel Urtasun, Richard Zemel, Sanja Fidler

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:

$$
\begin{aligned}
o_k(\mathbf{x}) &= \frac{1}{1+\exp(-z_k)} \\
z_k &= w_{k0} + \sum_{j=1}^{J} h_j(\mathbf{x}) w_{kj}
\end{aligned}
$$

- How can we train the network, that is, adjust all the parameters **w**?

slide by Raquel Urtasun, Richard Zemel, Sanja Fidler

Training Neural Networks

- Find weights:

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

  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)

slide by Raquel Urtasun, Richard Zemel, Sanja Fidler

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	$\mathrm{ReLU}(z) = \max(0, z)$	$\begin{cases} 1, & \text{if } z > 0 \\ 0, & \text{if } z \leq 0 \end{cases}$

slide by Raquel Urtasun, Richard Zemel, Sanja Fidler