Neural Networks
(aka Deep Learning)

Ties together many ideas from the course

- logistic regression
- template matching/nearest neighbors
- decision forests / ensembles of weak learners
- stochastic gradient descent

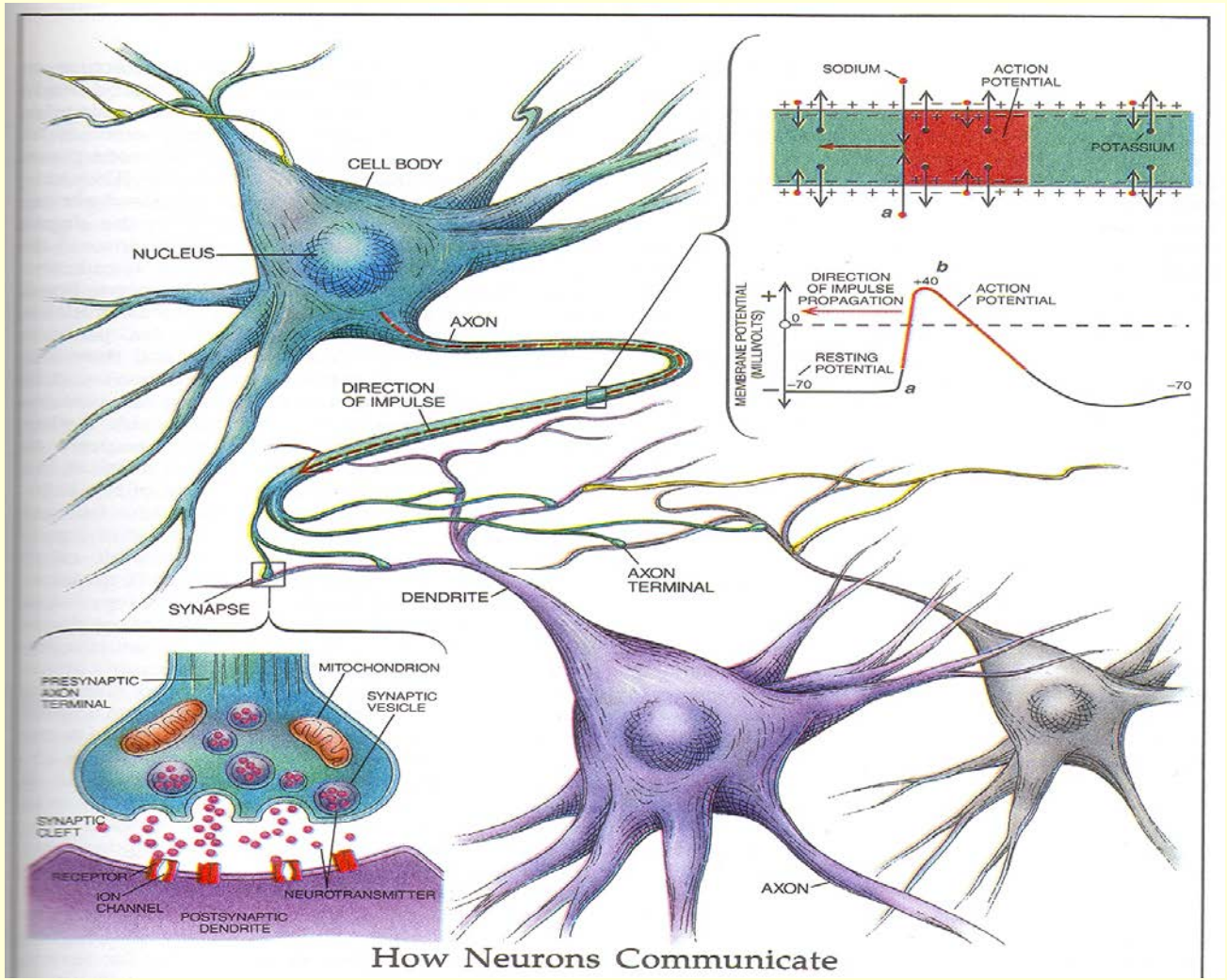
What's new:

- feature learning ("end-to-end training")
- Inspired by the brain

Reasons to study neural computation

- To understand how the brain actually works.
  - Its very big and very complicated and made of stuff that dies when you poke it around. So we need to use computer simulations.
- To understand a style of parallel computation inspired by neurons and their adaptive connections.
  - Very different style from sequential computation.
    - should be good for things that brains are good at (e.g. vision)
    - Should be bad for things that brains are bad at (e.g. 23 x 71)
- To solve practical problems by using novel learning algorithms inspired by the brain (this course)
  - Learning algorithms can be very useful even if they are not how the brain actually works.

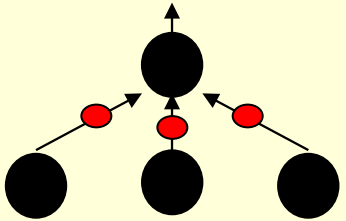
Slide by Geoff Hinton



How Neurons Communicate

“How the brain works” on one slide!

- Each neuron receives inputs from other neurons
  - A few neurons also connect to receptors.
  - Cortical neurons use spikes to communicate.
- The effect of each input line on the neuron is controlled by a synaptic weight
  - The weights can be positive or negative.
- The synaptic weights adapt so that the whole network learns to perform useful computations
  - Recognizing objects, understanding language, making plans, controlling the body.
- You have about 10^{11} neurons each with about 10^{4} weights.
  - A huge number of weights can affect the computation in a very short time. Much better bandwidth than a computer.

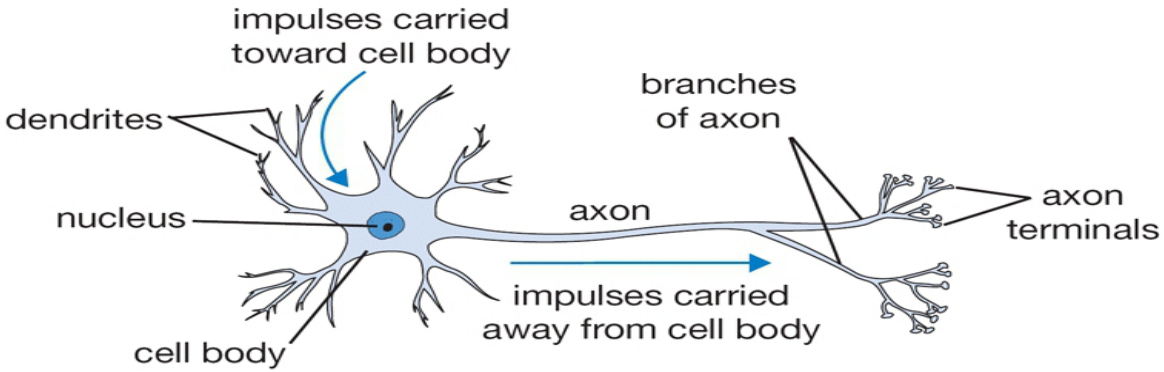


Slide by Geoff Hinton

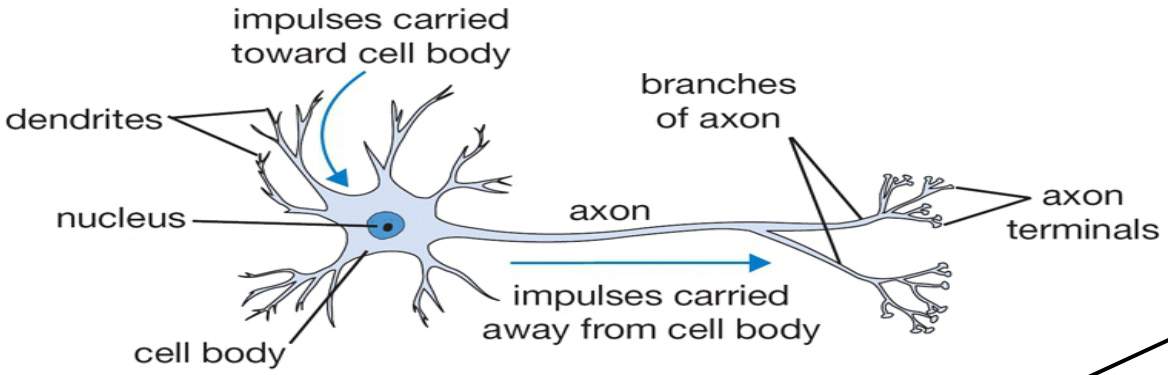
Modularity and the brain

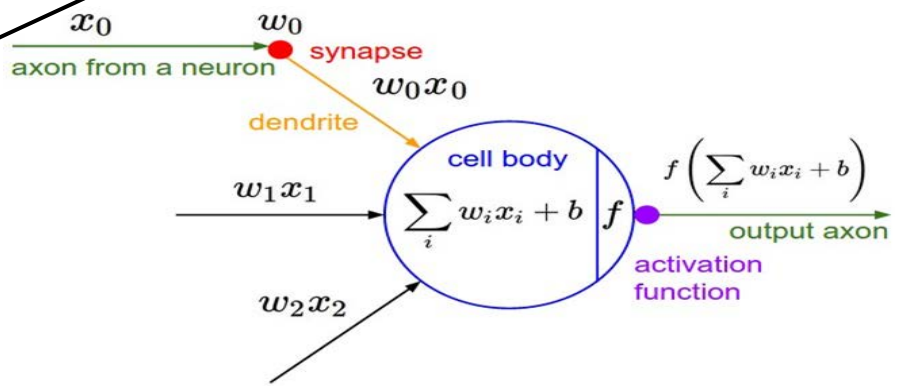
- Different bits of the cortex do different things.
  - Local damage to the brain has specific effects.
  - Specific tasks increase the blood flow to specific regions.
- But cortex looks pretty much the same all over.
  - Early brain damage makes functions relocate.
- Cortex is made of general purpose stuff that has the ability to turn into special purpose hardware in response to experience.
  - This gives rapid parallel computation plus flexibility.
  - Conventional computers get flexibility by having stored sequential programs, but this requires very fast central processors to perform long sequential computations.

Slide by Geoff Hinton

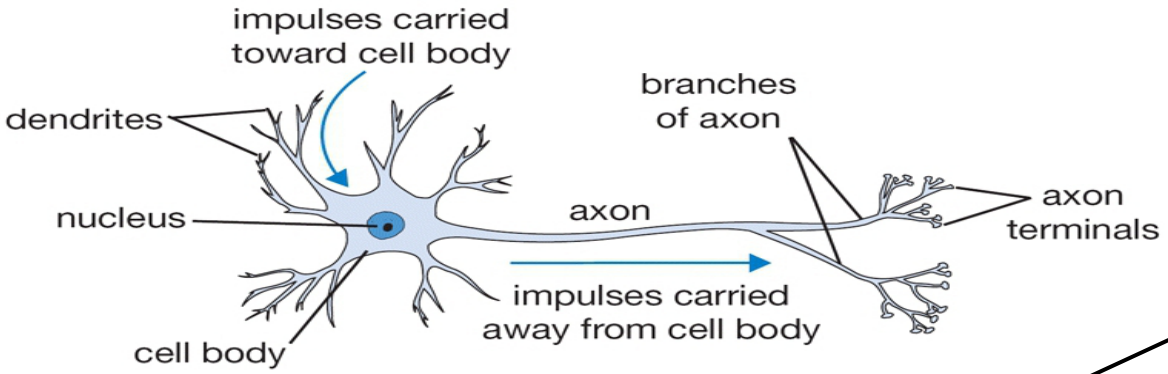


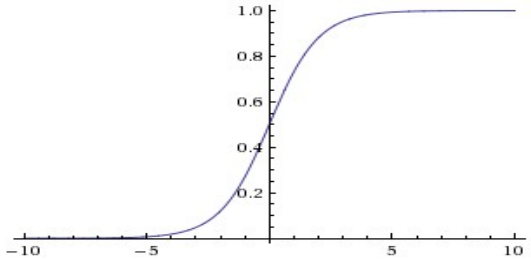
Source: Andrej Karpathy & Fei-Fei Li





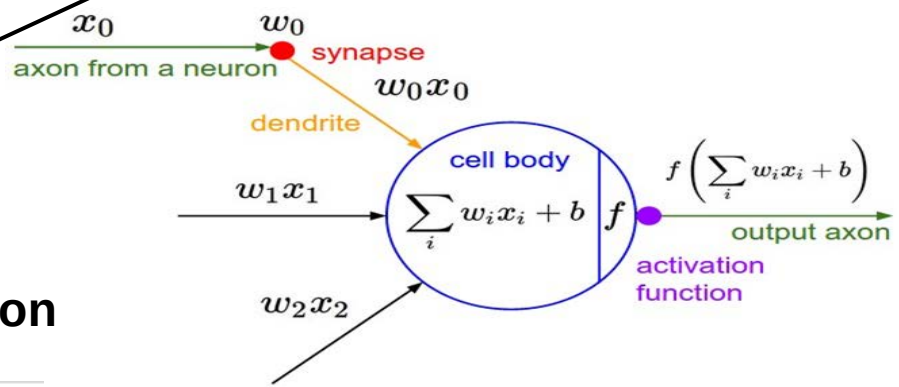
Source: Andrej Karpathy & Fei-Fei Li





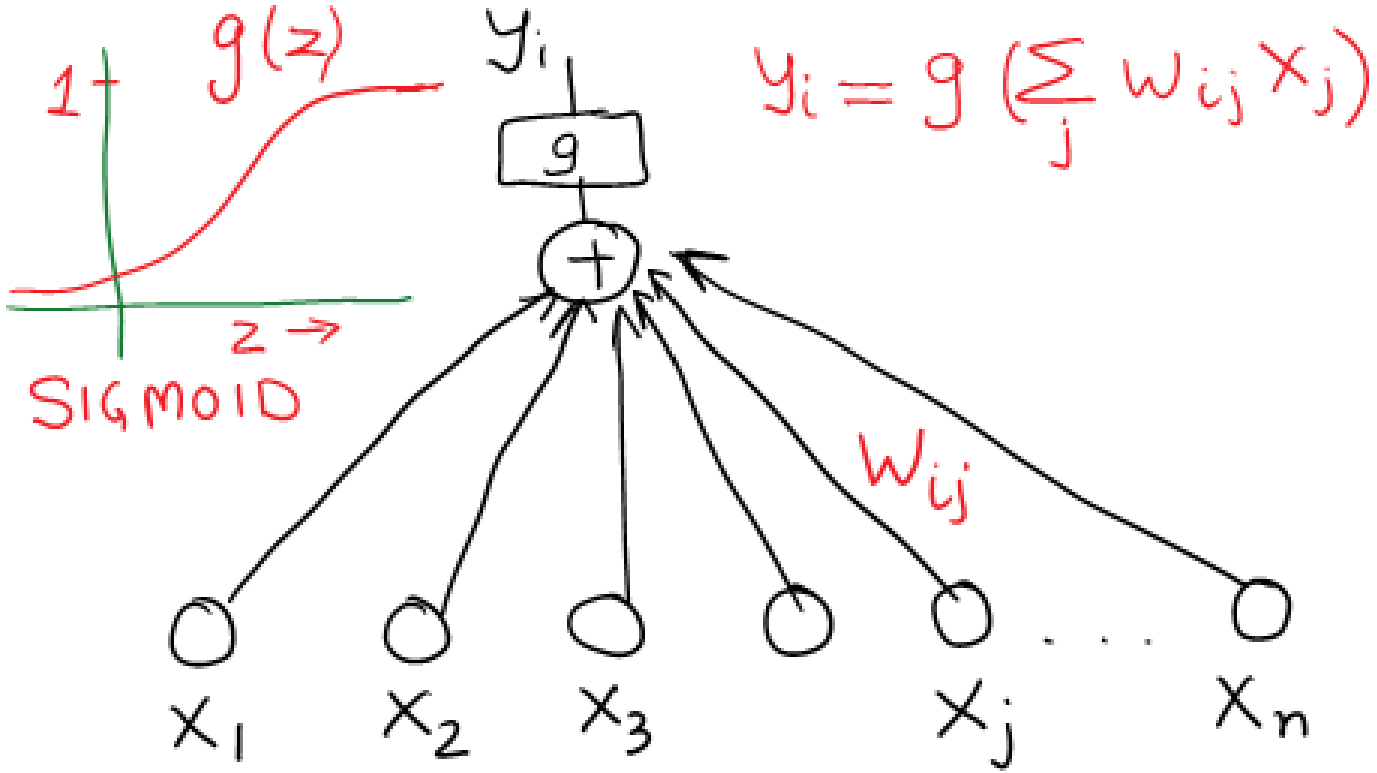
sigmoid activation function

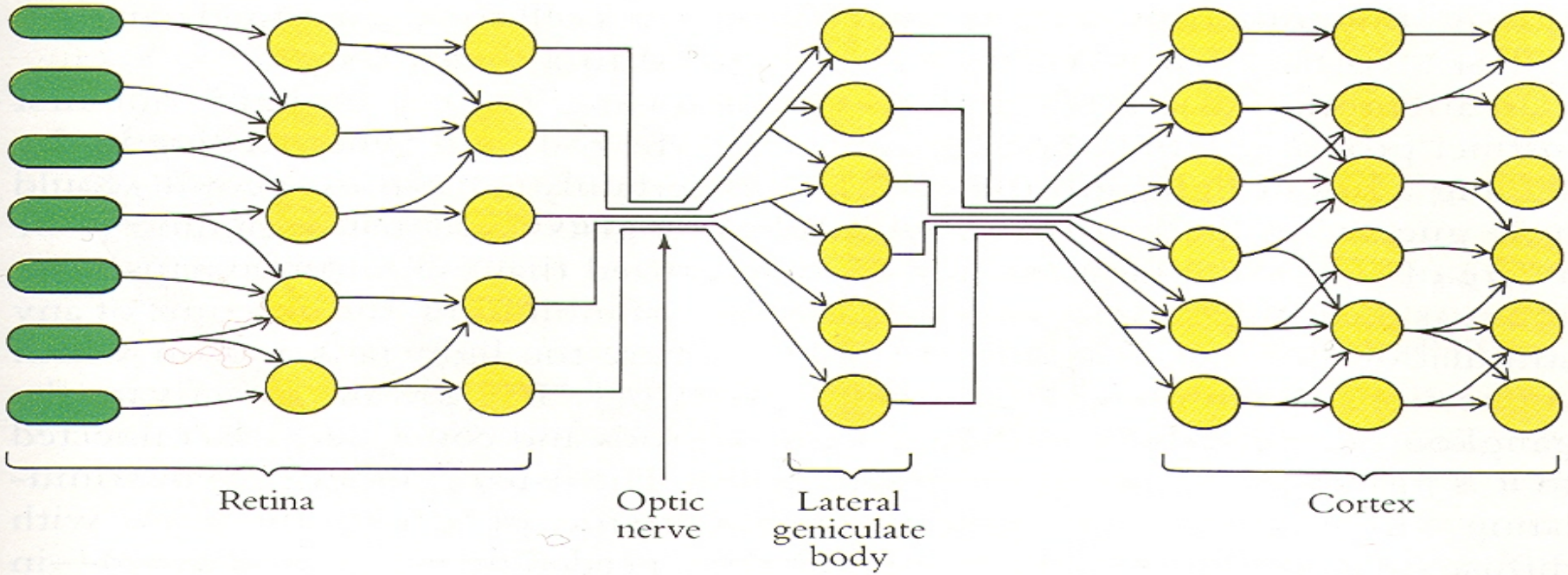
$$\frac{1}{1+e^{-x}}$$



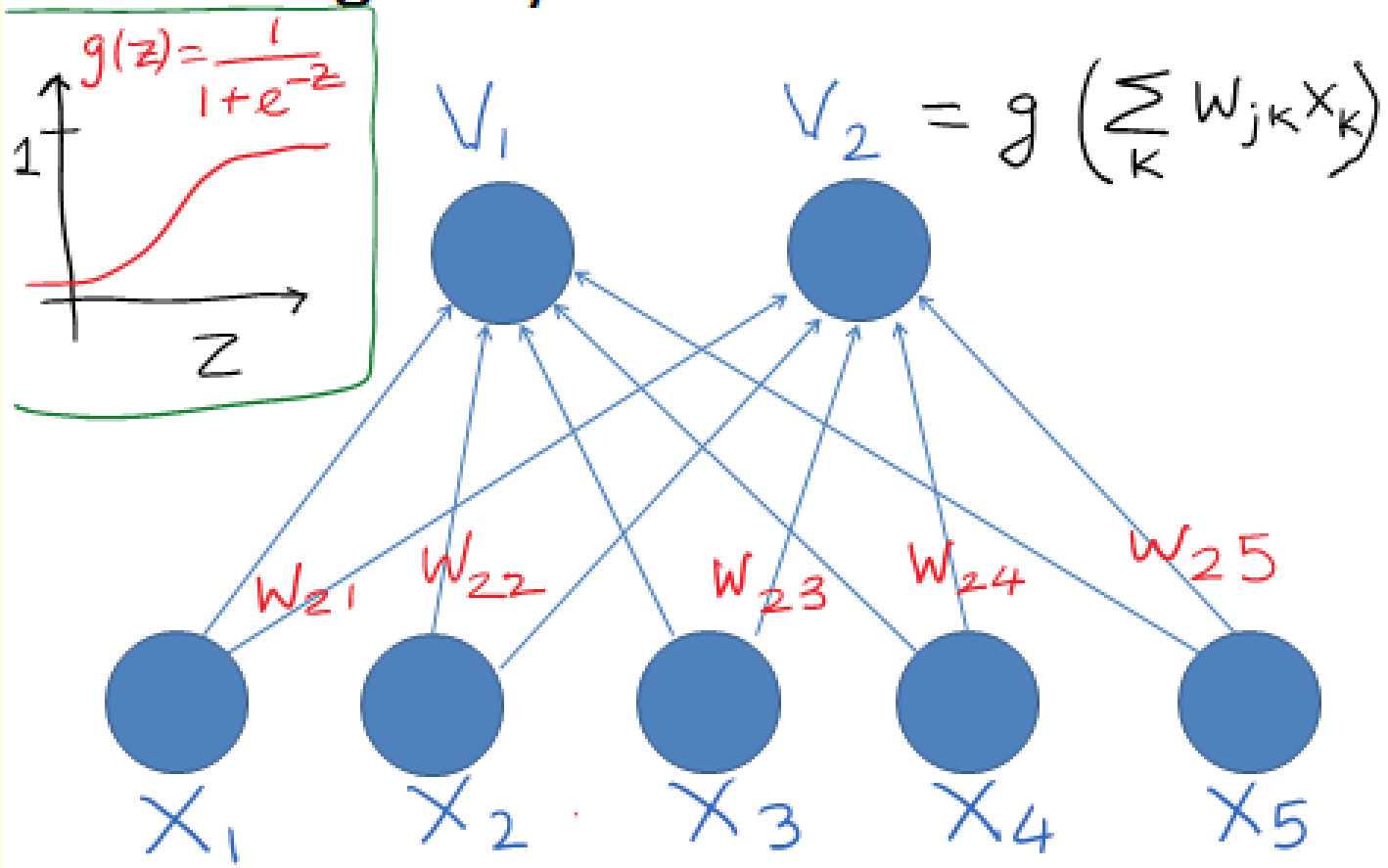
Source: Andrej Karpathy & Fei-Fei Li

Mathematical Abstraction



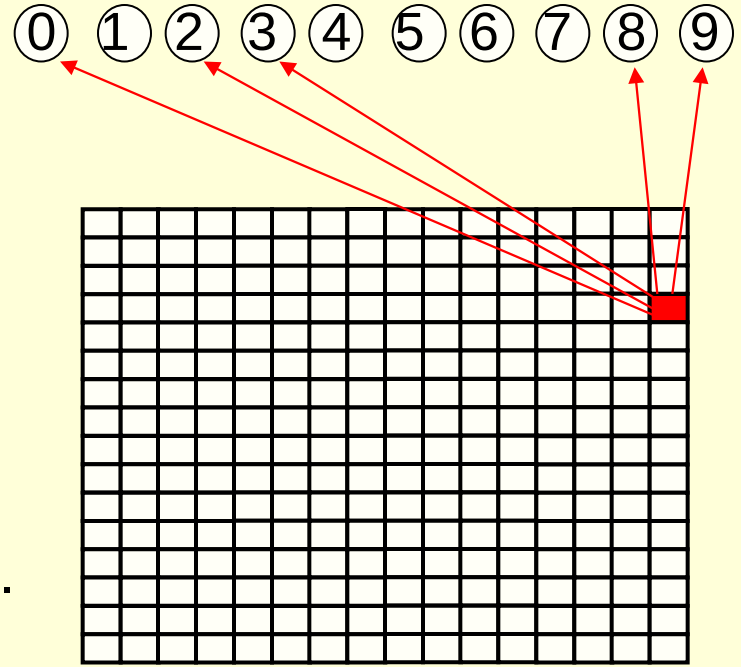

Retina
Optic nerve
Lateral geniculate body
Cortex

Single layer neural network



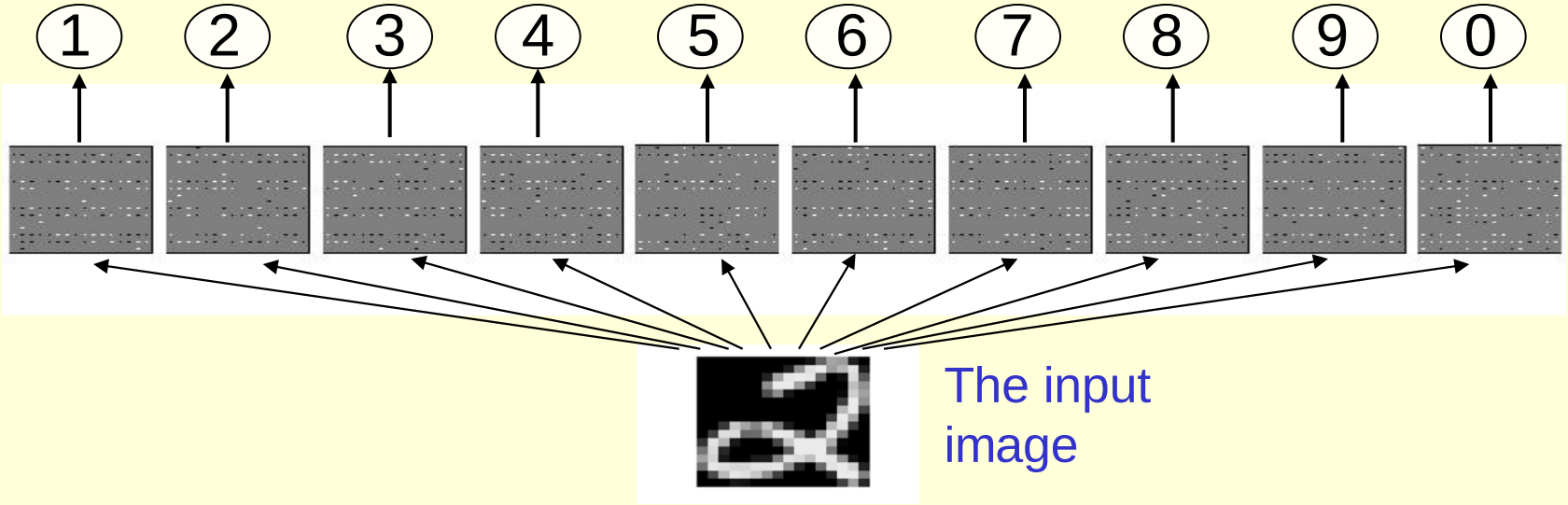
A very simple way to recognize handwritten shapes

- 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 Geoff Hinton

How to display the weights

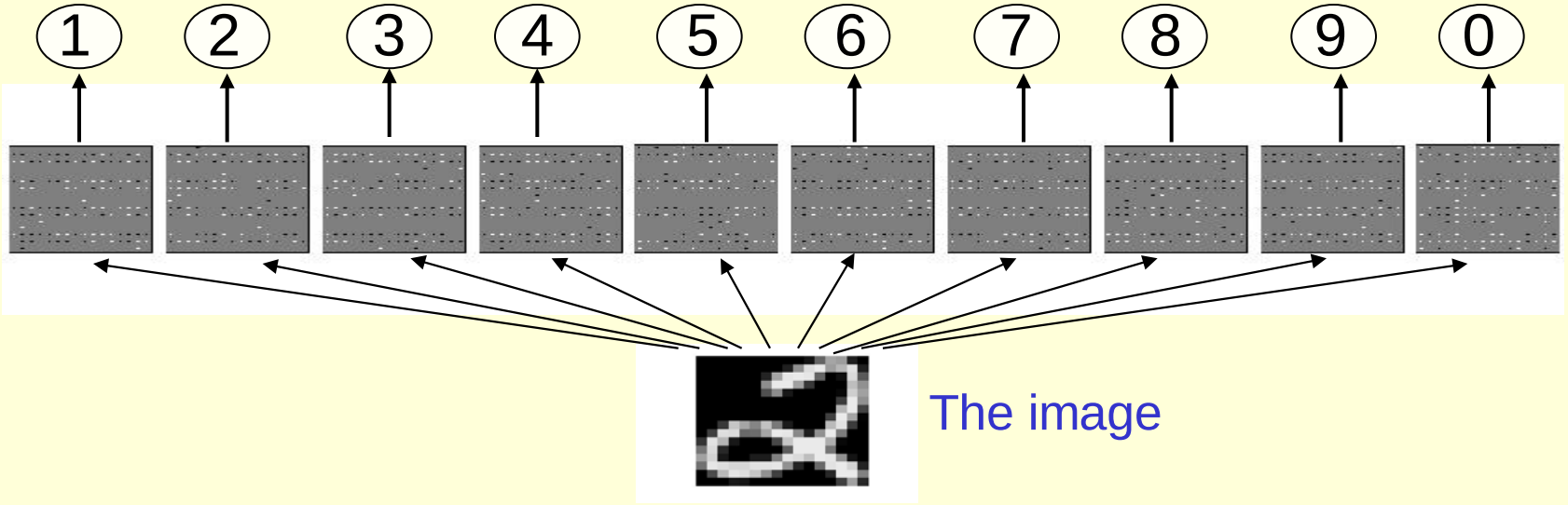


Give each output unit its own “map” of the input image and display the weight coming from each pixel in the location of that pixel in the map.

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

Slide by Geoff Hinton

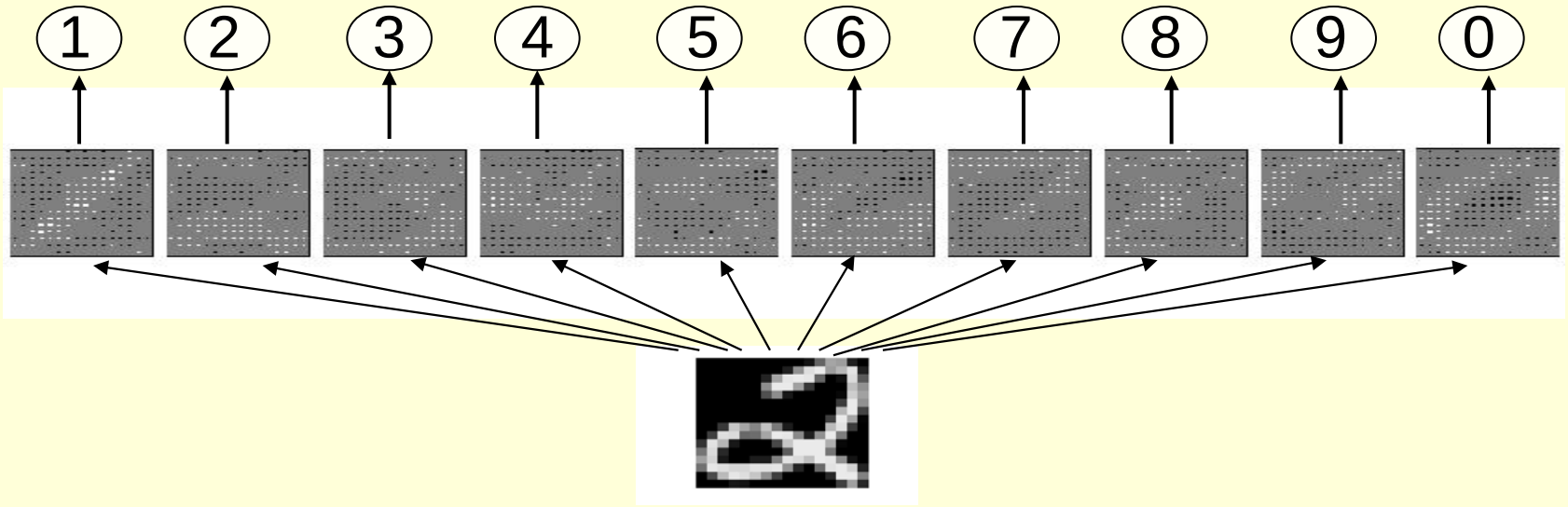
How to learn the weights



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

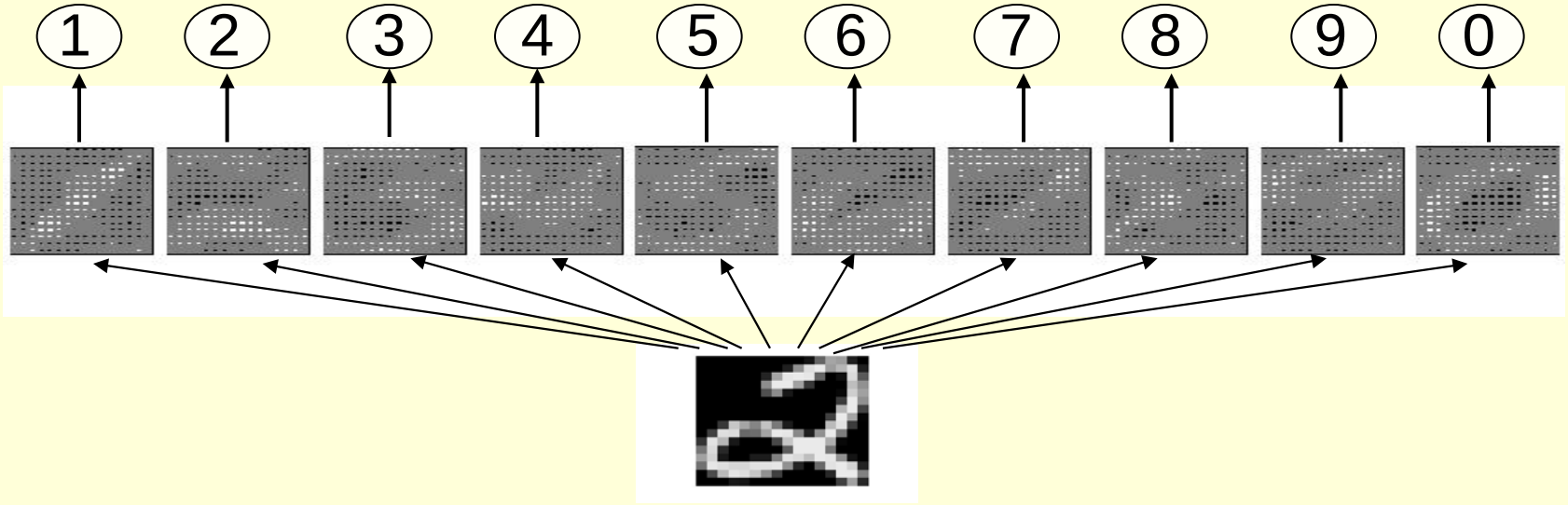
Then decrement the weights from active pixels to whatever class the network guesses.

Slide by Geoff Hinton



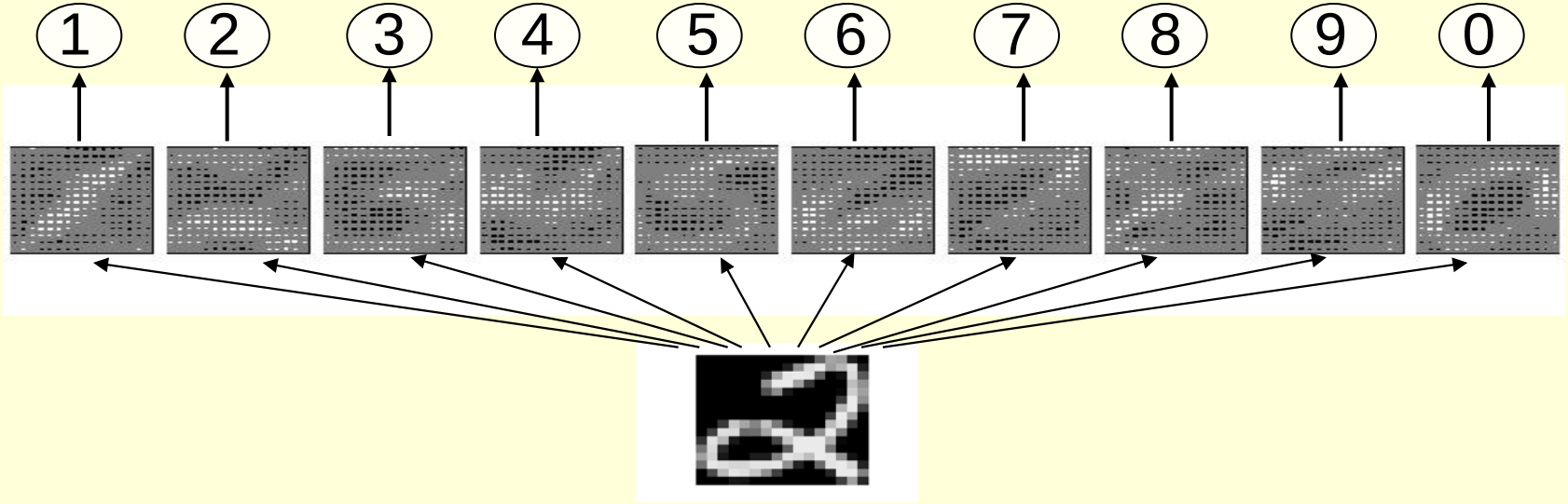
The image

Slide by Geoff Hinton



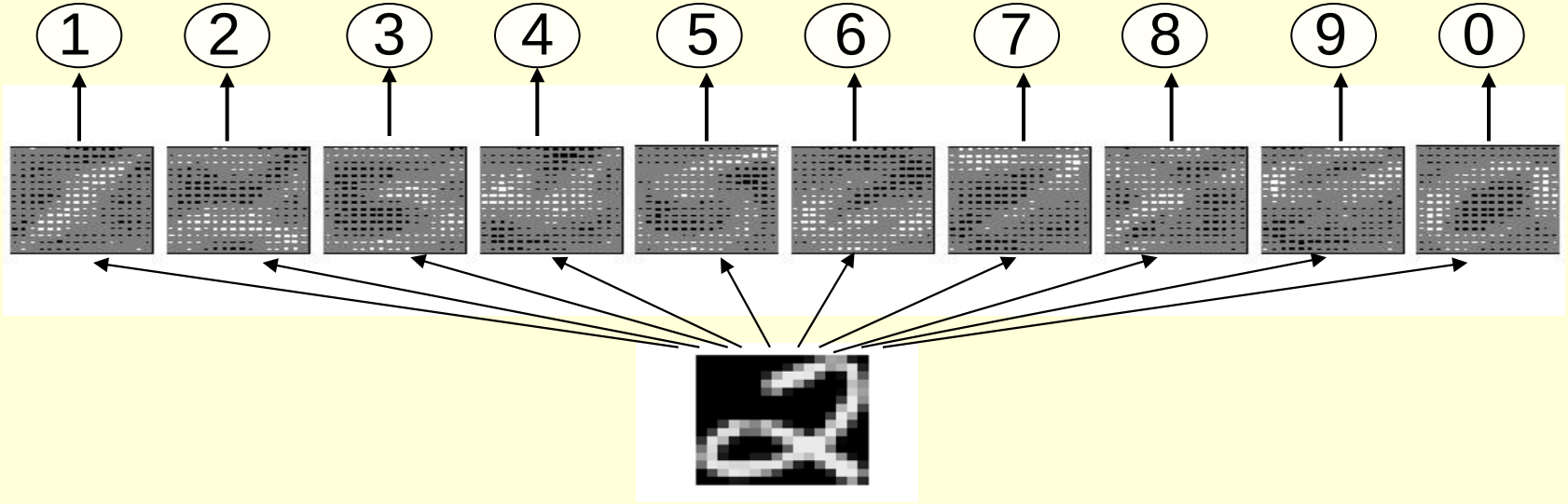
The image

Slide by Geoff Hinton



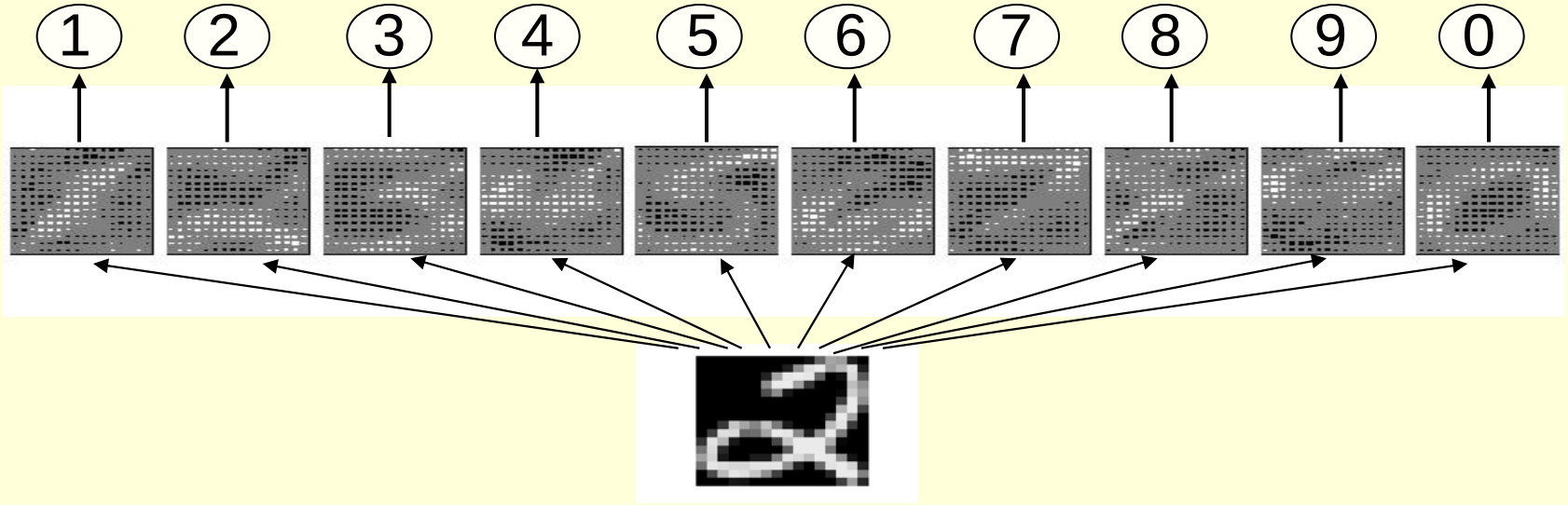
The image

Slide by Geoff Hinton



The image

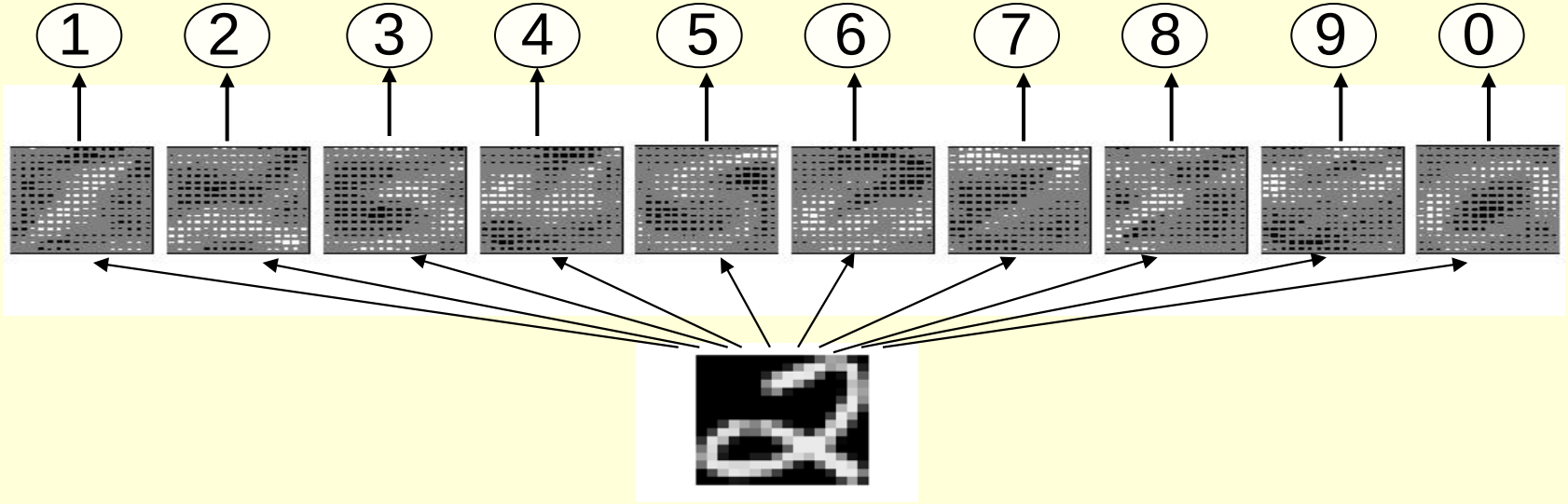
Slide by Geoff Hinton



The image

Slide by Geoff Hinton

The learned weights



The image

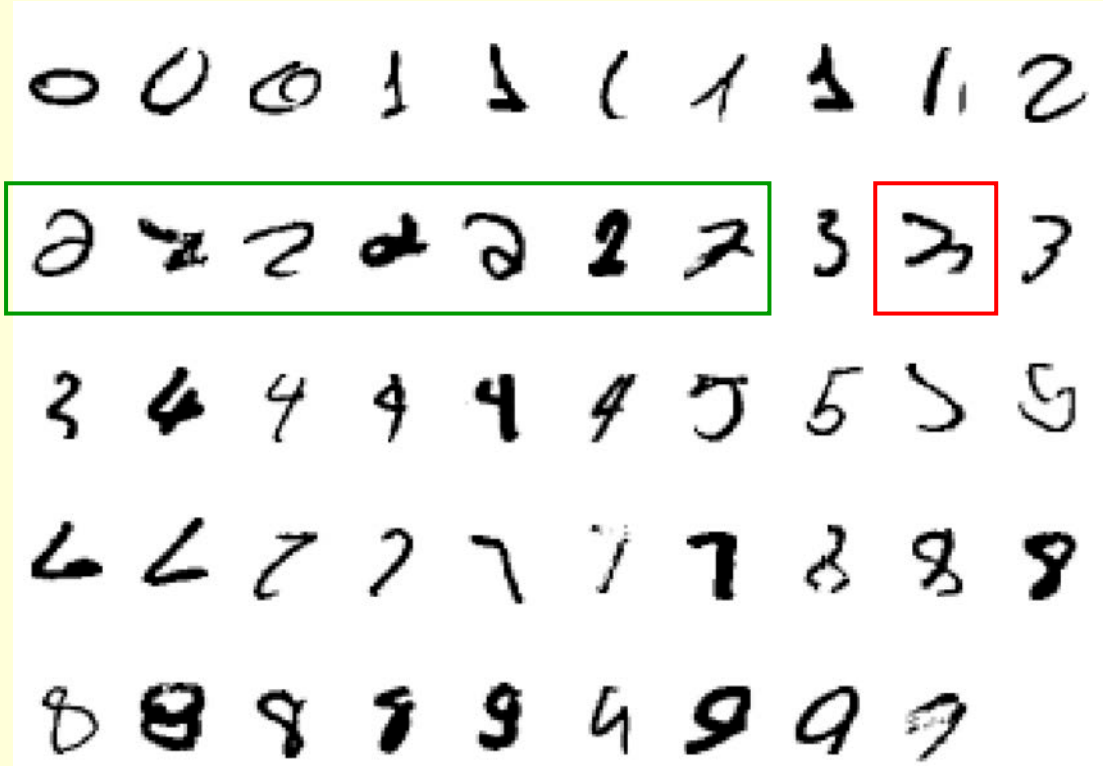
Slide by Geoff Hinton

Why the simple learning algorithm is 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.

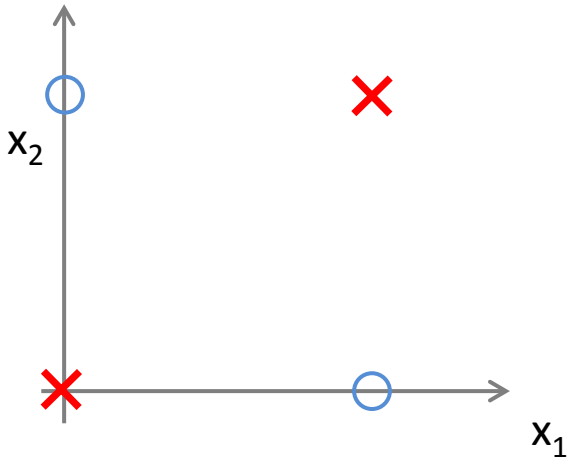
Slide by Geoff Hinton

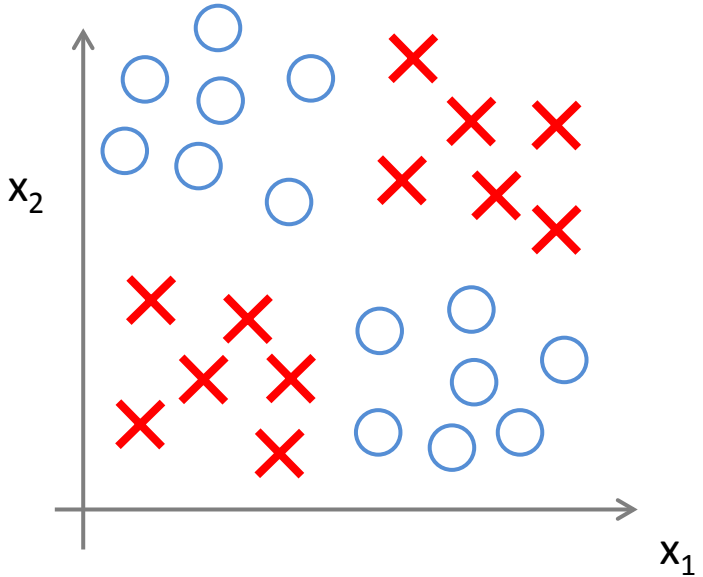
Examples of handwritten digits that can be recognized correctly the first time they are seen



Non-linear classification example: XOR/XNOR

x_1, x_2 are binary (0 or 1).





$y = x_1 \text{ XOR } x_2$

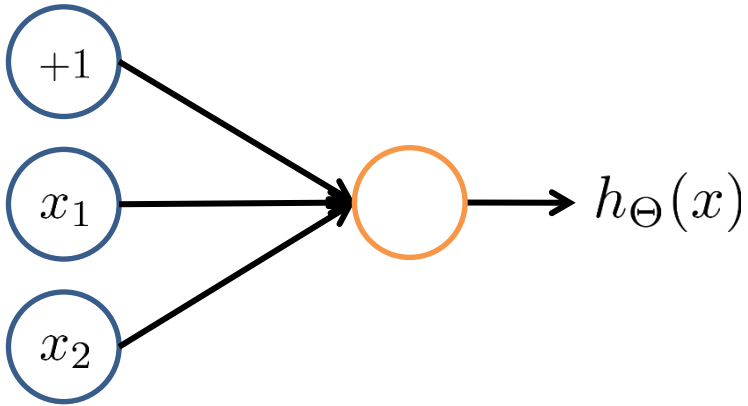
$x_1 \text{ XNOR } x_2$

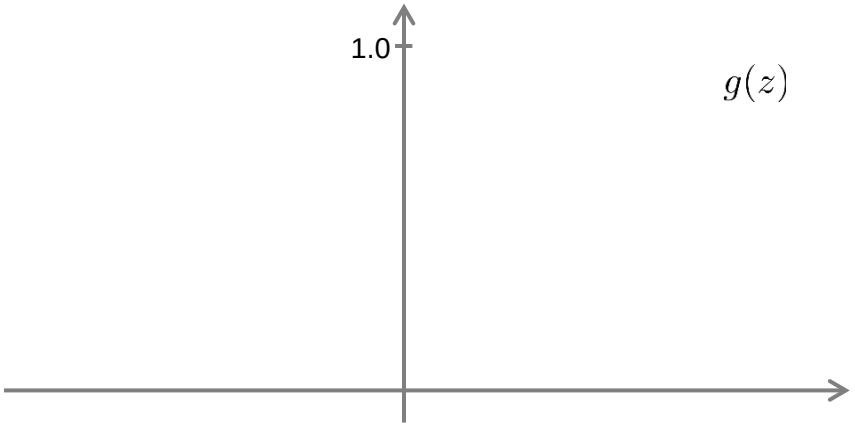
$\text{NOT } (x_1 \text{ XOR } x_2)$

Simple example: AND

$x_1, x_2 \in \{0, 1\}$

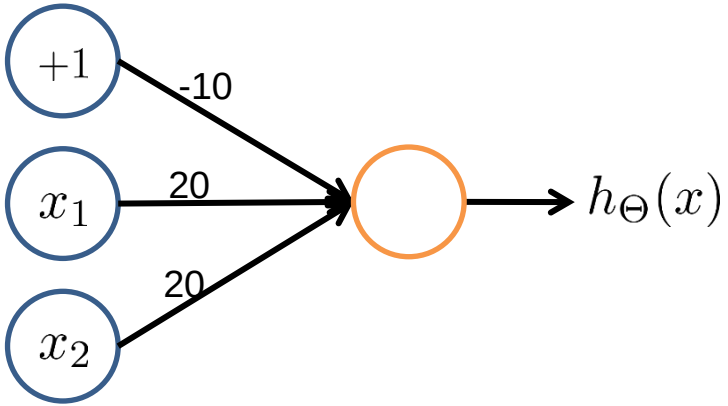
$y = x_1 \text{ AND } x_2$





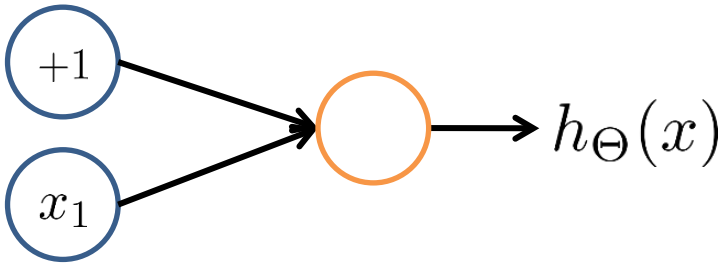
x_1	x_2	$h_\Theta(x)$
0	0	
0	1	
1	0	
1	1	

Example: OR function



x_1	x_2	$h_\Theta(x)$
0	0	
0	1	
1	0	
1	1	

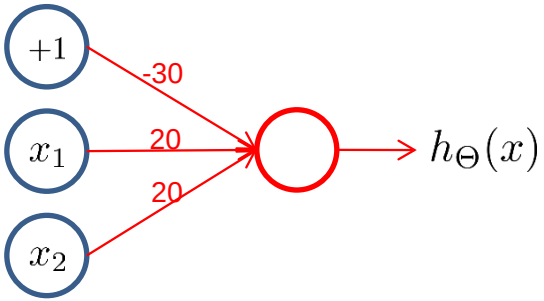
Negation:



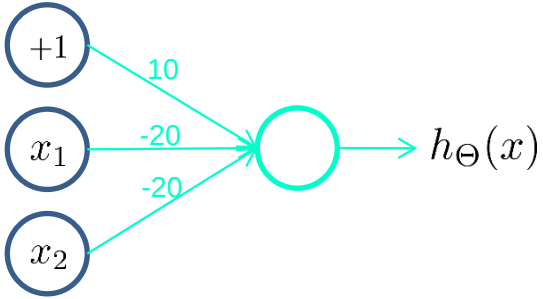
x_1	$h_\Theta(x)$
0	
1	

$$h_\Theta(x) = g(10 - 20x_1)$$

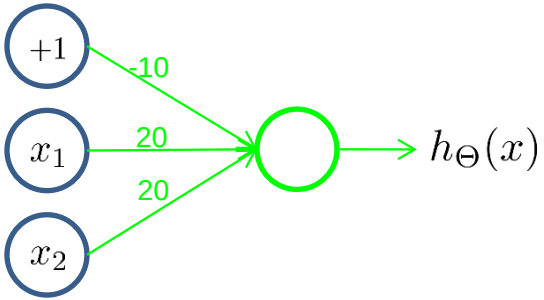
Putting it together: x_1 XNOR x_2



x_1 AND x_2



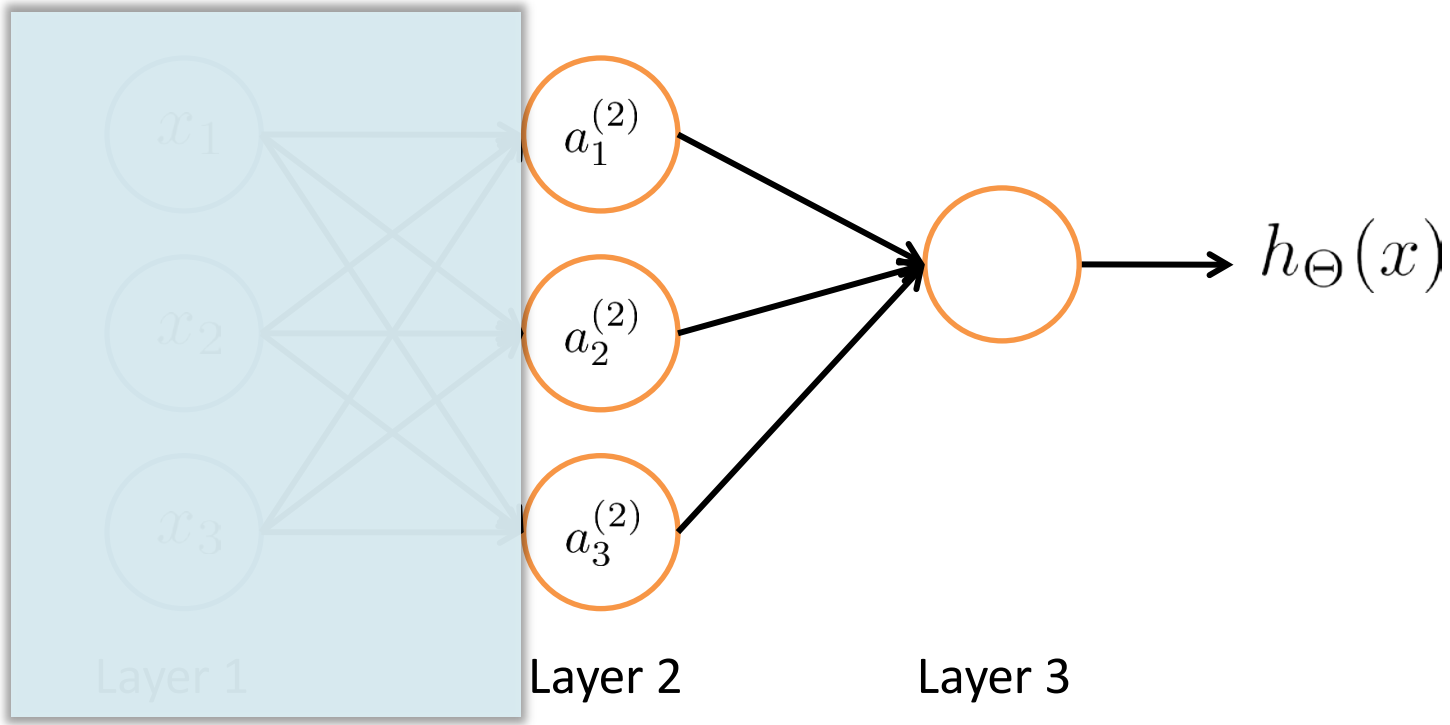
(NOT x_1) AND (NOT x_2)



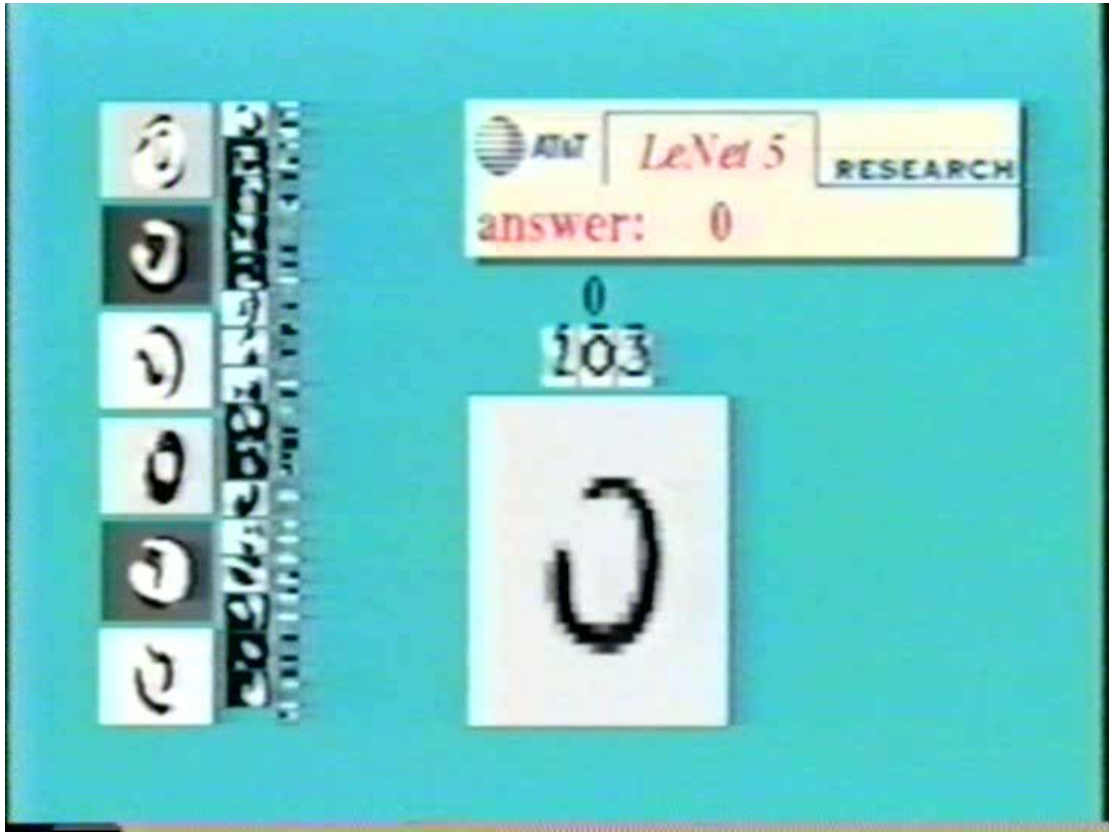
x_1 OR x_2

x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0			
0	1			
1	0			
1	1			

Neural Network learning its own features

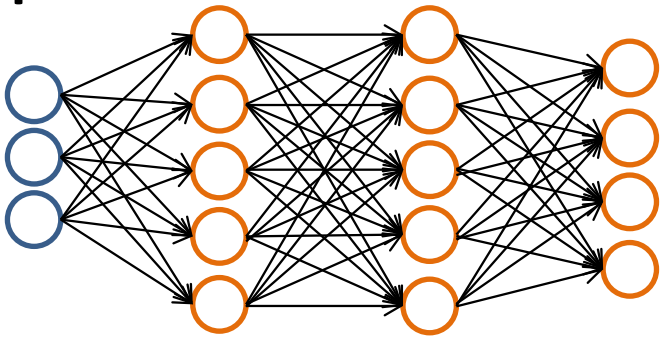


Handwritten digit classification



[Courtesy of Yann LeCun]

Multiple output units: One-vs-all.



$h_\Theta(x) \in \mathbb{R}^4$

Want $h_\Theta(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$, etc.

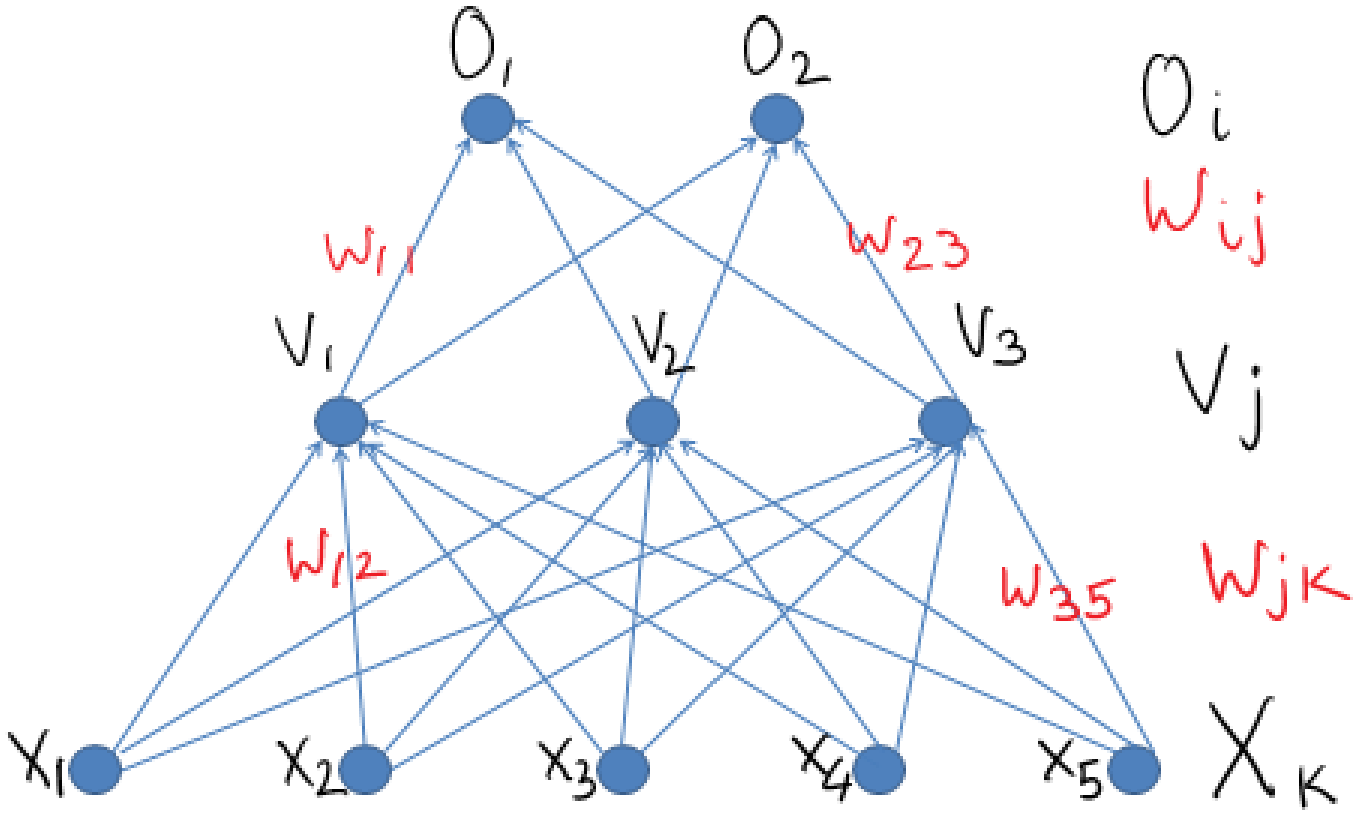
when pedestrian when car when motorcycle

Training set: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$

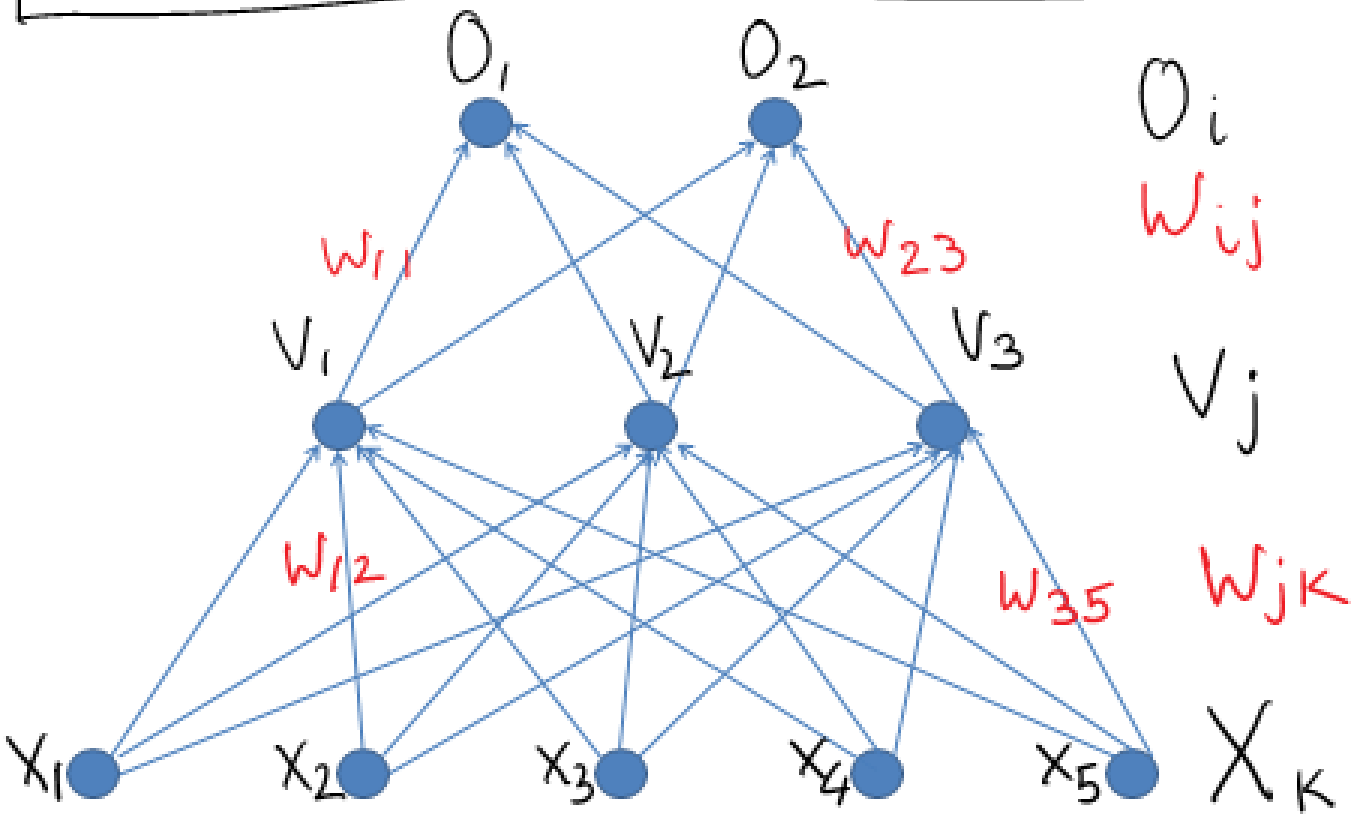
$y^{(i)}$ one of $\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$, $\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$, $\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$

pedestrian car motorcycle truck

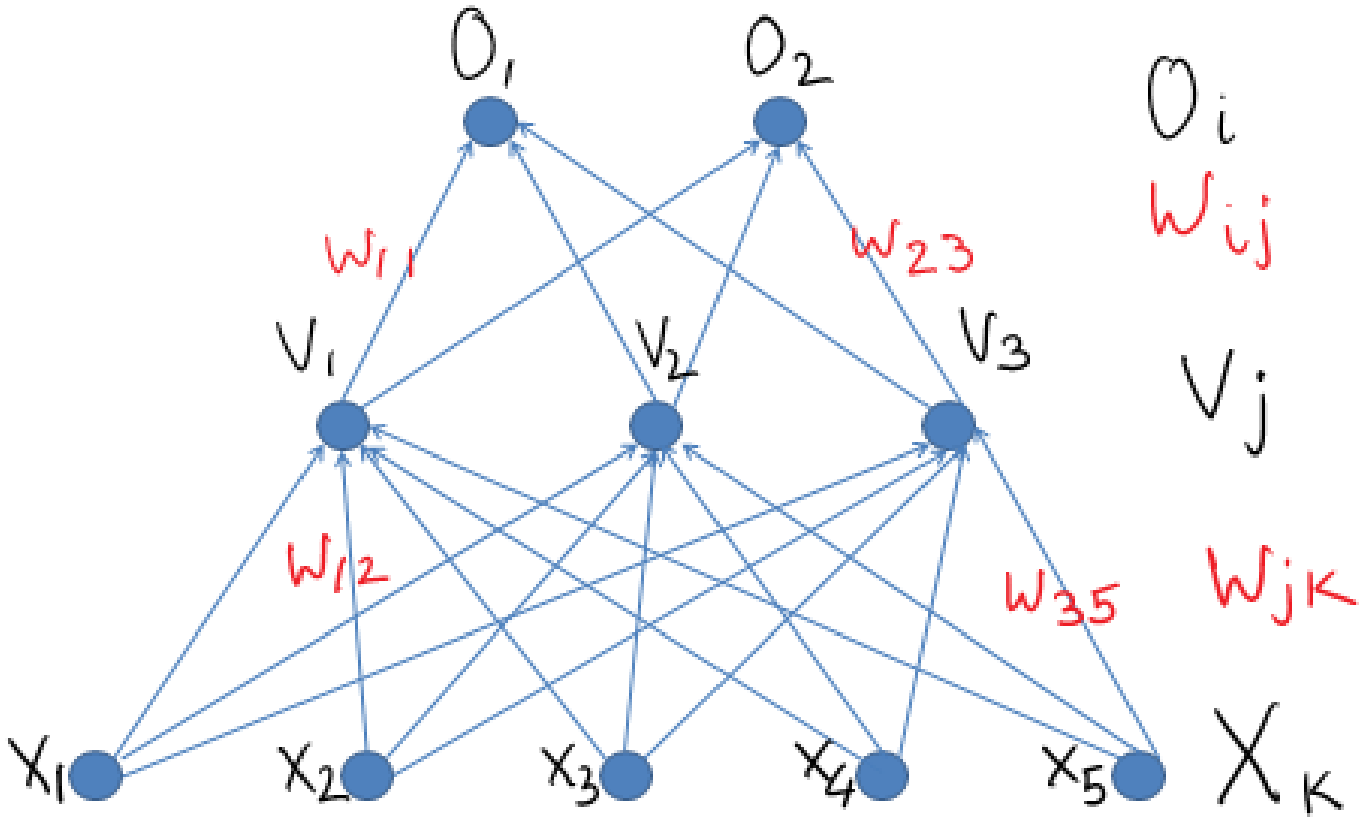
Two layer neural network



$$V_j = g\left(\sum_k w_{jk} x_k\right); \quad O_i = g\left(\sum w_{ij} V_j\right)$$



$$O_i = g\left(\sum_j W_{ij}\, g\left(\sum_K W_{jK}\, x_K\right)\right)$$



O_i

W_{ij}

V_j

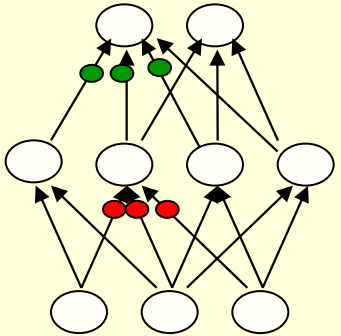
W_{jK}

X_K

Training a neural network

Learning by perturbing weights
(this idea occurs to everyone who knows about evolution)

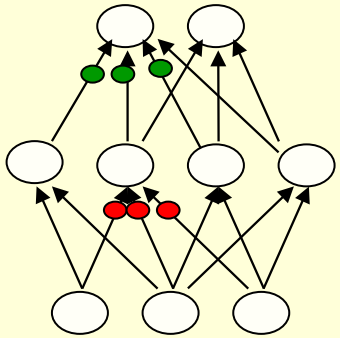
- Randomly perturb one weight and see if it improves performance. If so, save the change.
  - This is a form of reinforcement learning.
  - Very inefficient. We need to do multiple forward passes on a representative set of training cases just to change one weight.
  - Towards the end of learning, large weight perturbations will nearly always make things worse, because the weights need to have the right relative values.



Slide by Geoff Hinton

Learning by using perturbations

- We could randomly perturb all the weights in parallel and correlate the performance gain with the weight changes.
  - Not any better because we need lots of trials on each training case to "see" the effect of changing one weight through the noise created by all the changes to other weights.
- A better idea: Randomly perturb the activities of the hidden units.
  - Once we know how we want a hidden activity to change on a given training case, we can compute how to change the weights.
  - There are fewer activities than weights, but backpropagation still wins by a factor of the number of neurons.



Slide by Geoff Hinton

The idea behind backpropagation

- We don't know what the hidden units ought to do, but we can compute how fast the error changes as we change a hidden activity.
  - Instead of using desired activities to train the hidden units, use error derivatives w.r.t. hidden activities.
  - Each hidden activity can affect many output units and can therefore have many separate effects on the error. These effects must be combined.
- We can compute error derivatives for all the hidden units efficiently at the same time.
  - Once we have the error derivatives for the hidden activities, its easy to get the error derivatives for the weights going into a hidden unit.

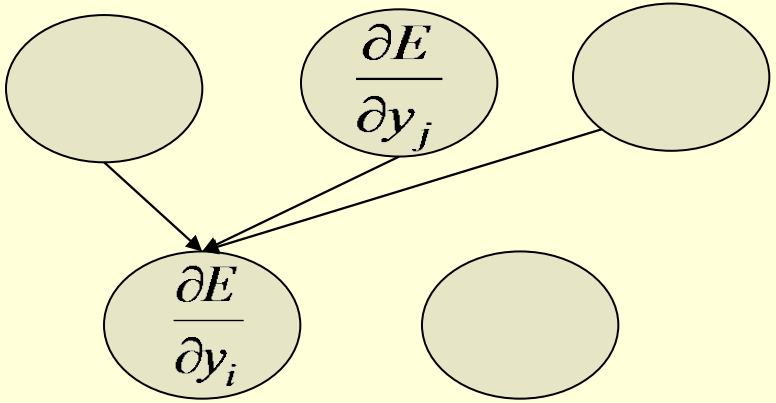
Slide by Geoff Hinton

Sketch of the backpropagation algorithm on a single case

- First convert the discrepancy between each output and its target value into an error derivative.
- Then compute error derivatives in each hidden layer from error derivatives in the layer above.
- Then use error derivatives *w.r.t.* activities to get error derivatives *w.r.t.* the incoming weights.

$$E = \frac{1}{2} \sum_{j \in output} (t_j - y_j)^2$$

$$\frac{\partial E}{\partial y_j} = -(t_j - y_j)$$



The derivatives of a logistic neuron

- The derivatives of the logit, z, with respect to the inputs and the weights are very simple:

$$z = b + \sum_i x_i w_i$$

$$\frac{\partial z}{\partial w_i} = x_i \qquad \frac{\partial z}{\partial x_i} = w_i$$

- The derivative of the output with respect to the logit is simple if you express it in terms of the output:

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

$$\frac{dy}{dz} = y\,(1 - y)$$

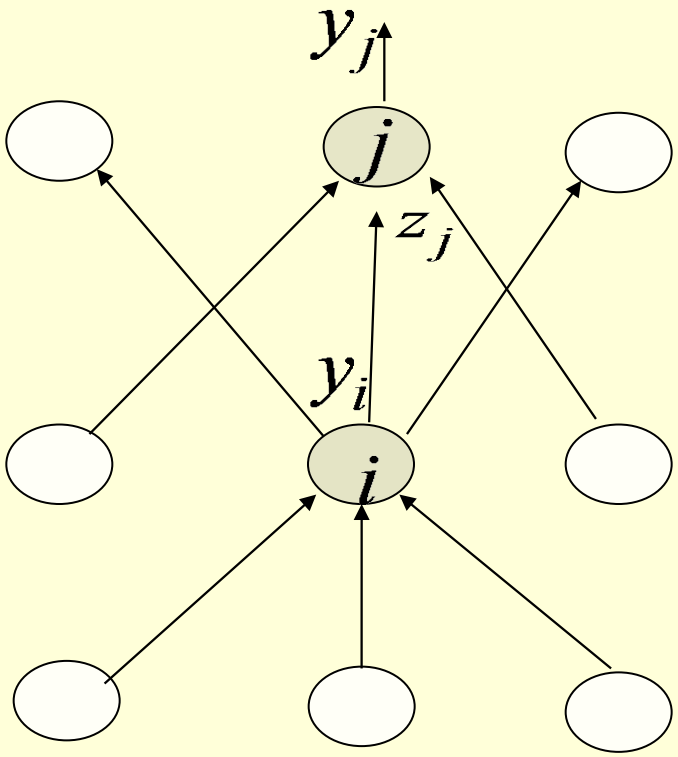
The derivatives of a logistic neuron

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

$$\frac{dy}{dz} = \frac{-1(-e^{-z})}{(1+e^{-z})^2} = \left(\frac{1}{1+e^{-z}}\right)\left(\frac{e^{-z}}{1+e^{-z}}\right) = y(1-y)$$

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

Backpropagating dE/dy



$$\frac{\partial E}{\partial z_j} = \frac{dy_j}{dz_j}\frac{\partial E}{\partial y_j} = y_j(1-y_j)\frac{\partial E}{\partial y_j}$$

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{dz_j}{dy_i}\frac{\partial E}{\partial z_j} = \sum_j w_{ij}\frac{\partial E}{\partial z_j}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}}\frac{\partial E}{\partial z_j} = y_i\frac{\partial E}{\partial z_j}$$