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

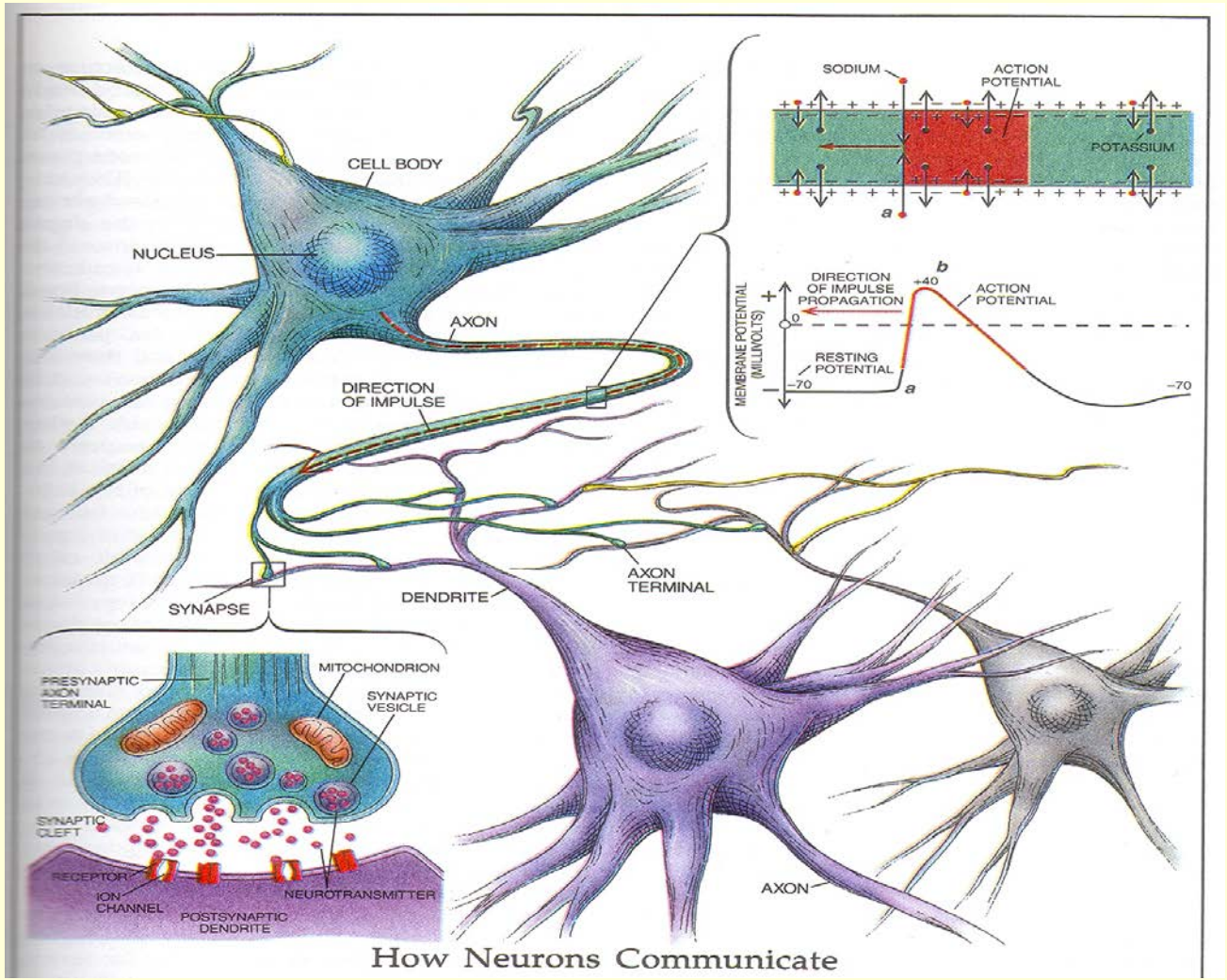
What's new:

- 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.

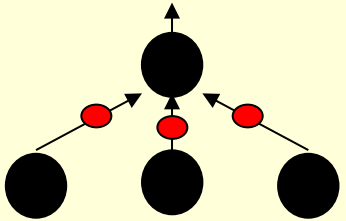
Slide by Geoff Hinton



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.

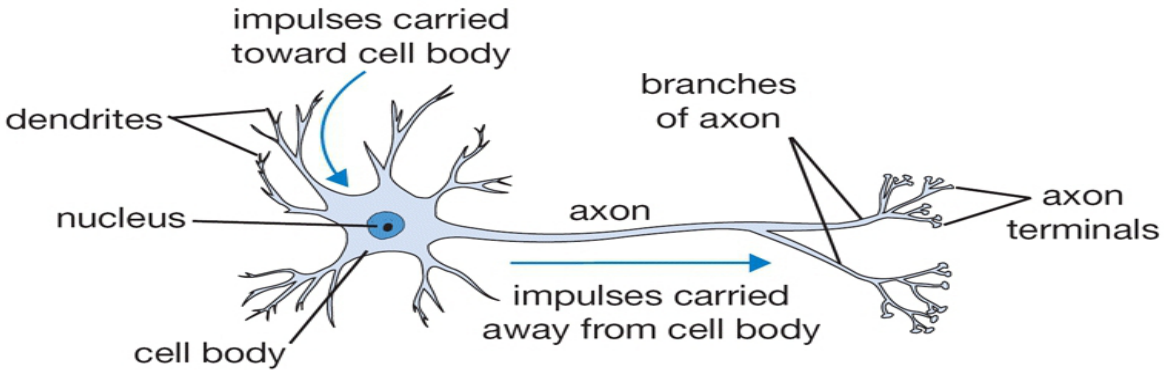


Slide by Geoff Hinton

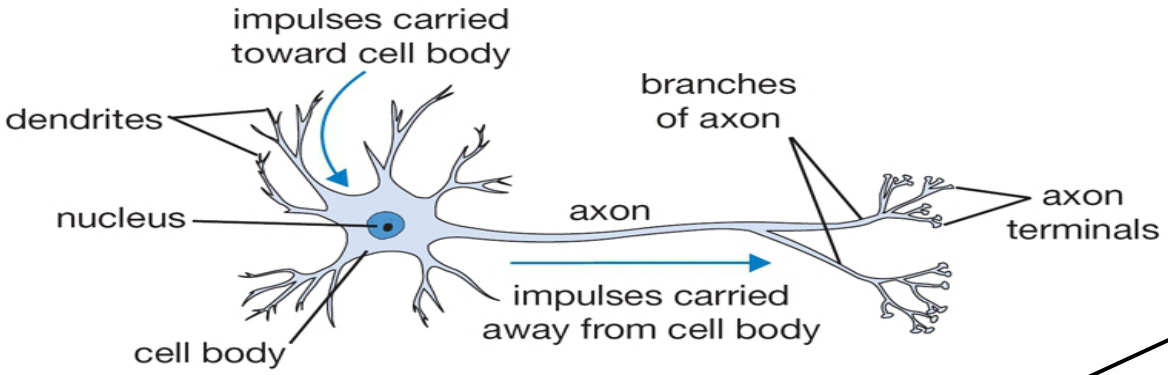
Modularity and the brain

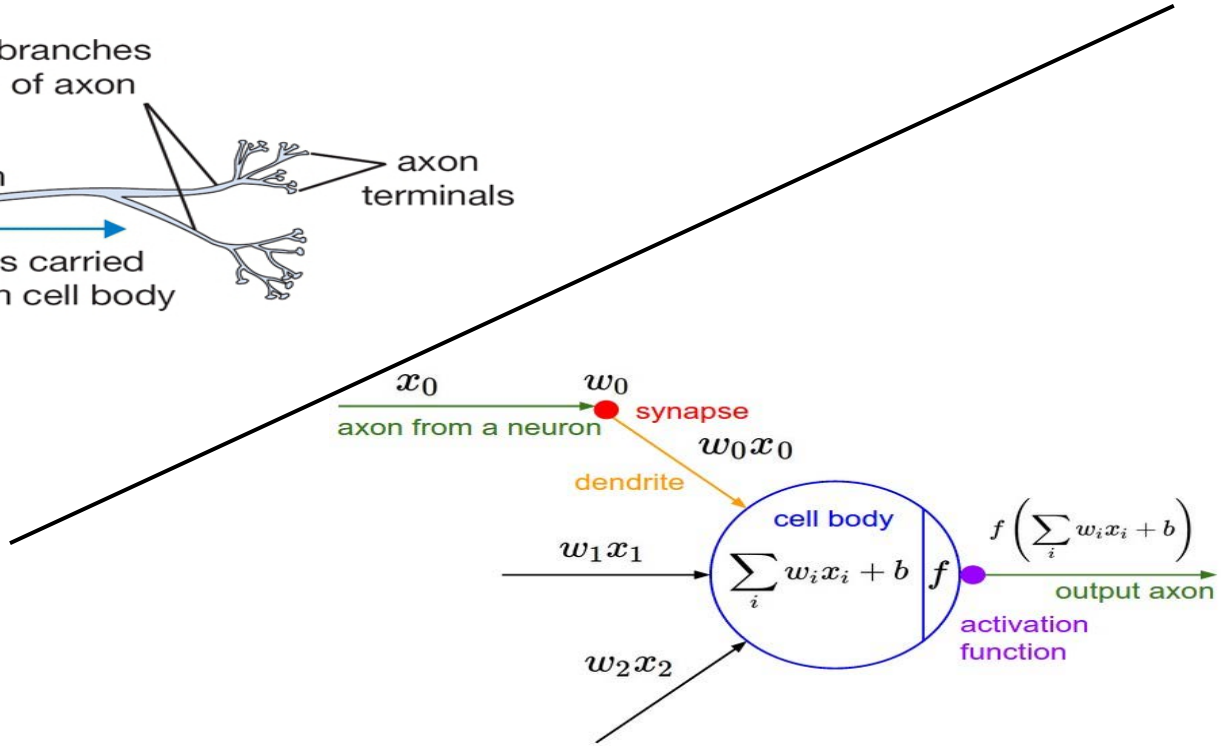
- Different bits of the cortex do different things.
  - Local damage to the brain has specific effects.
  - Specific tasks increase the blood flow to specific regions.
- But cortex looks pretty much the same all over.
  - Early brain damage makes functions relocate.
- Cortex is made of general purpose 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

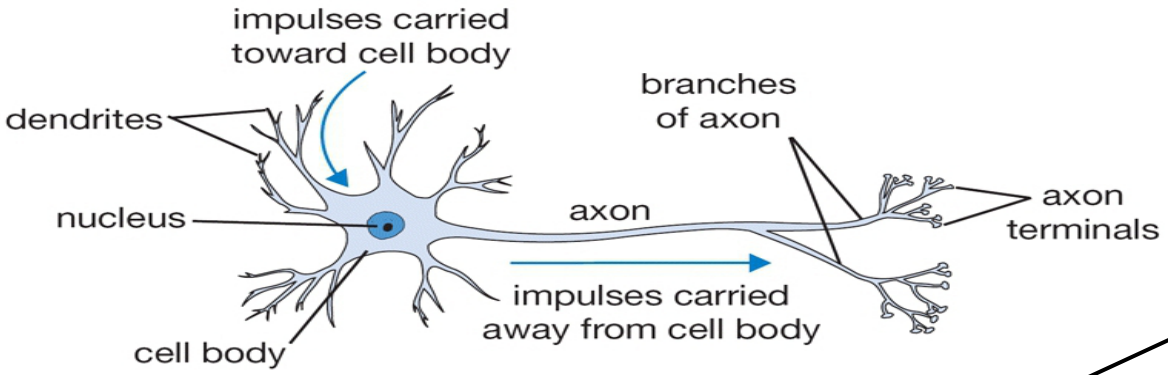


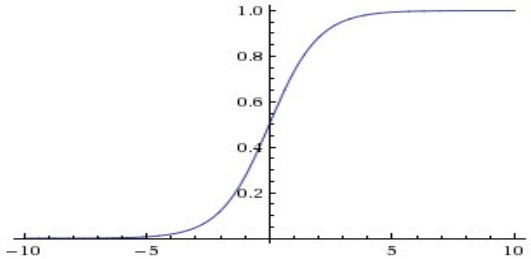
Source: Andrej Karpathy & Fei-Fei Li





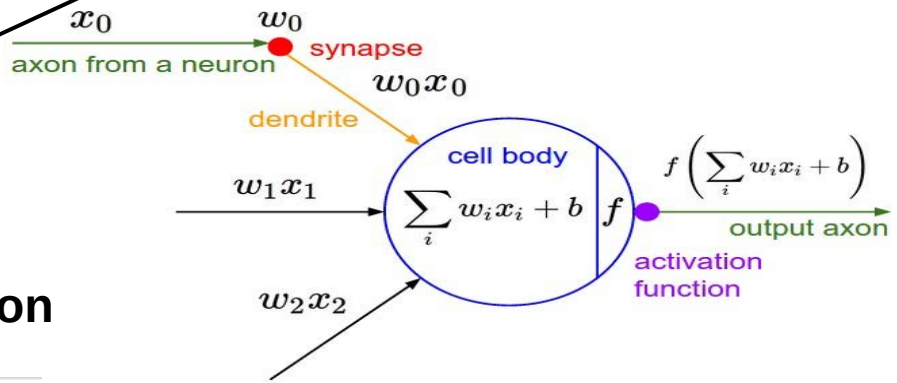
Source: Andrej Karpathy & Fei-Fei Li





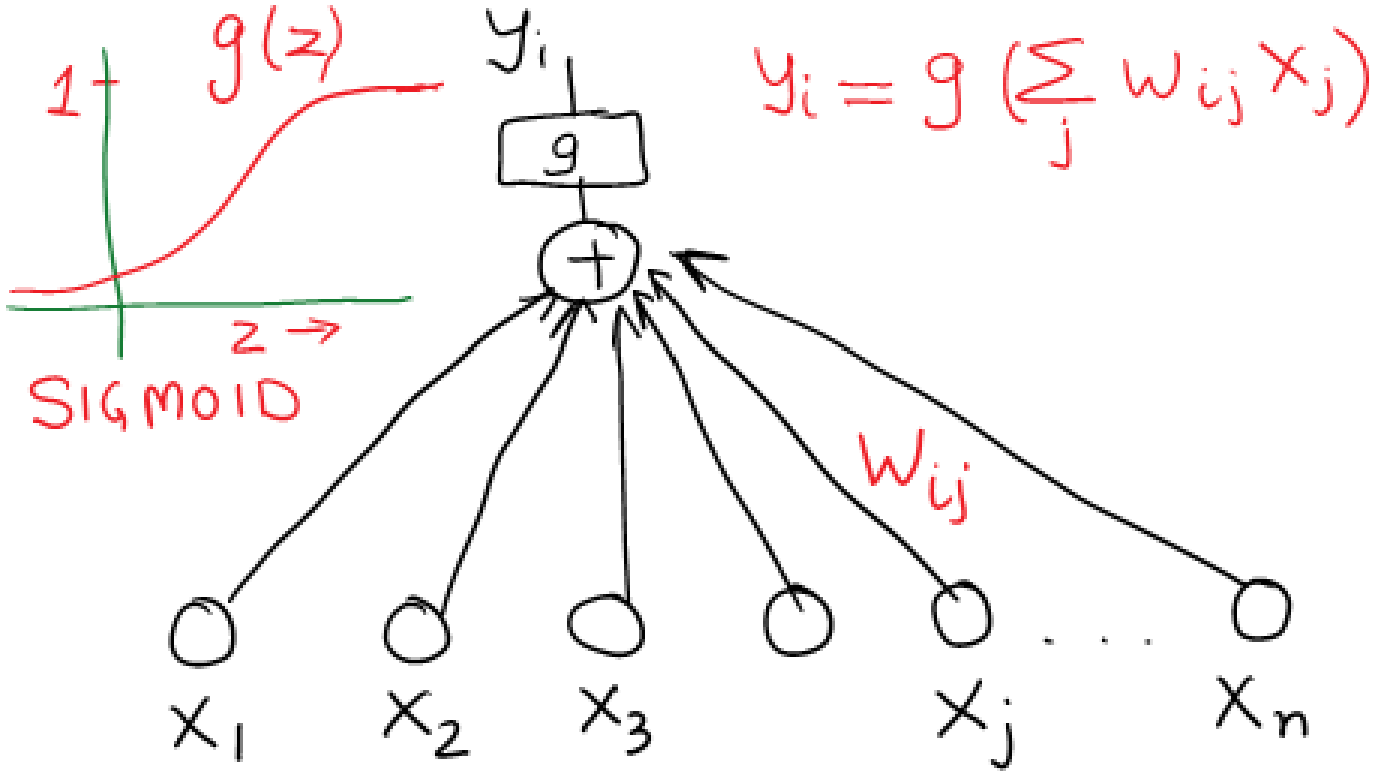
sigmoid activation function

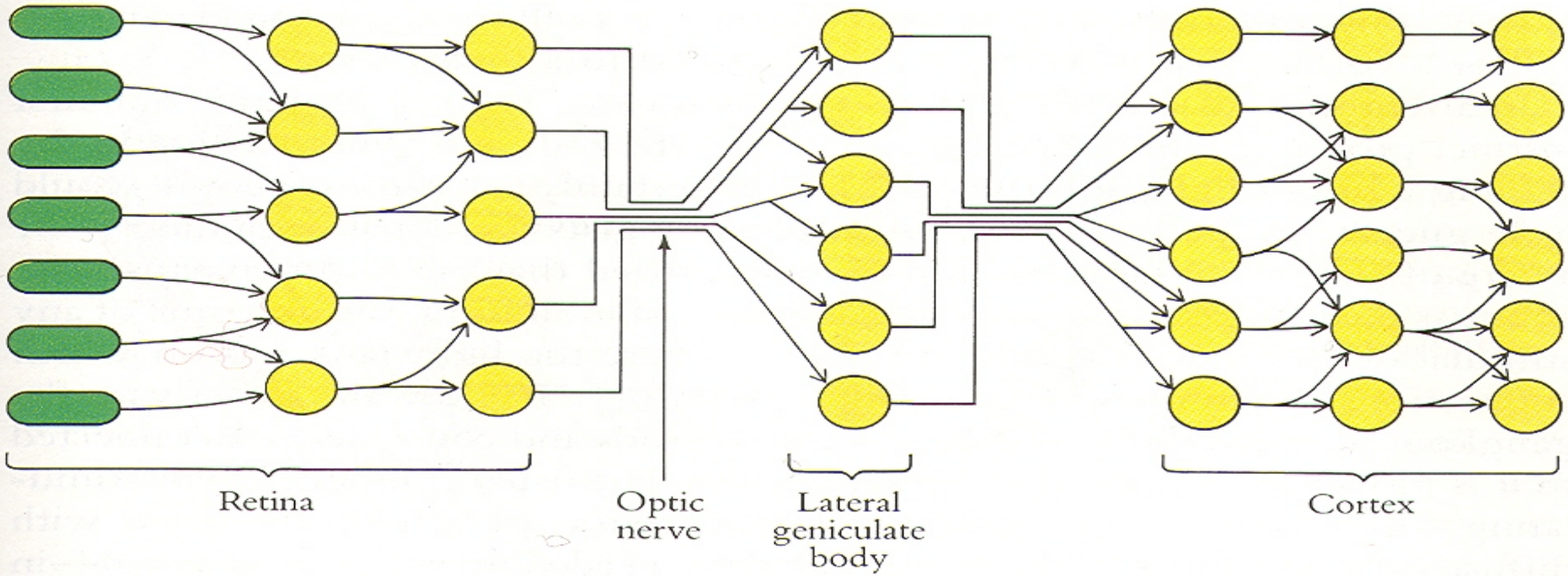
$$\frac{1}{1+e^{-x}}$$



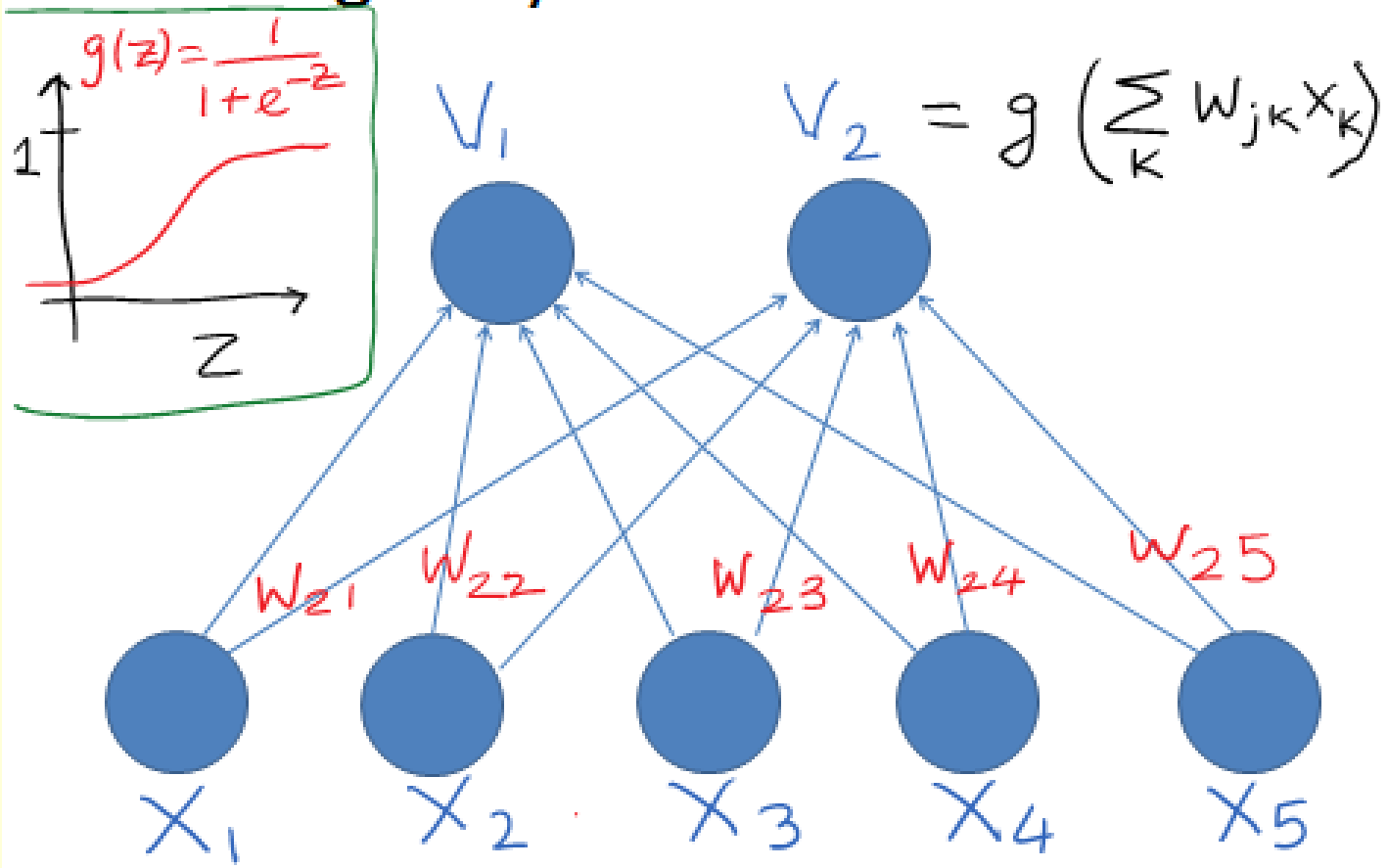
Source: Andrej Karpathy & Fei-Fei Li

Mathematical Abstraction



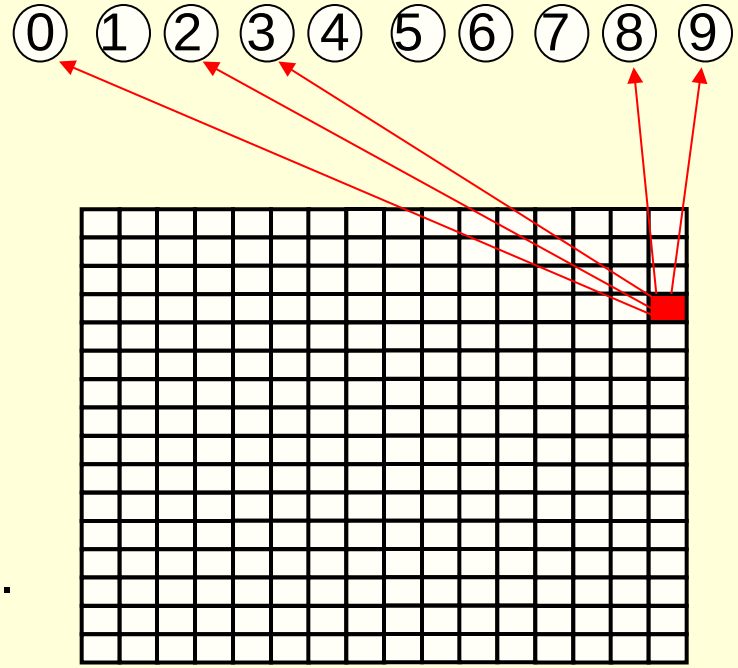

Retina
Optic nerve
Lateral geniculate body
Cortex

Single layer neural network



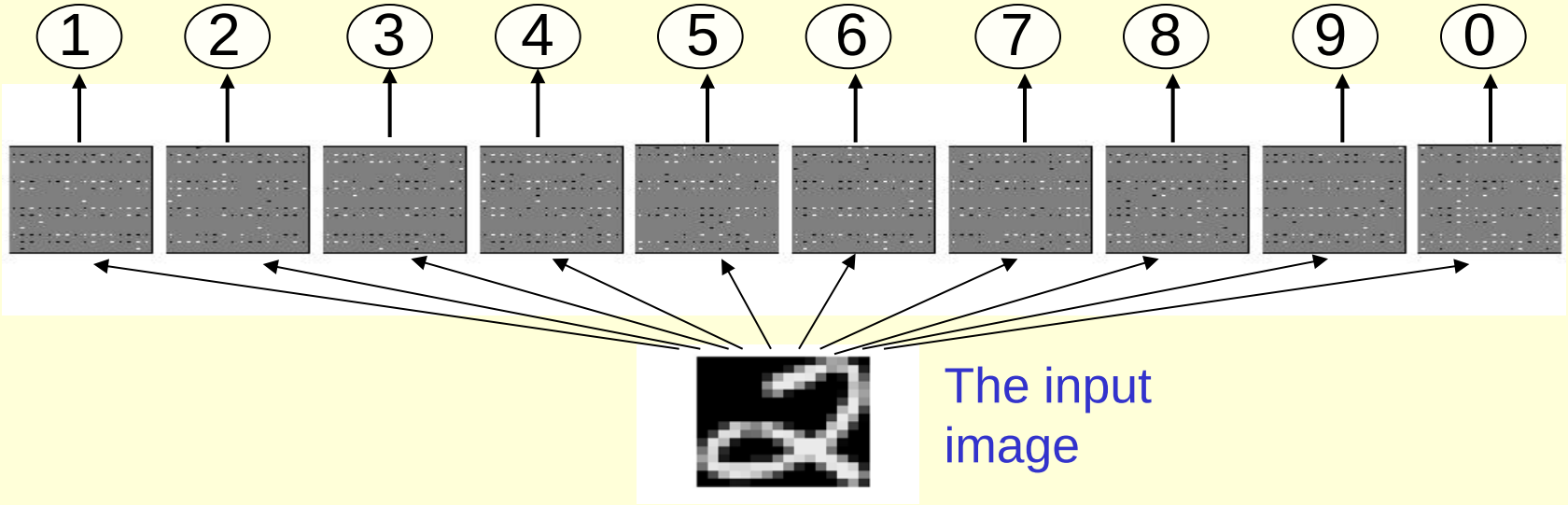
A very simple way to recognize handwritten shapes

- 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

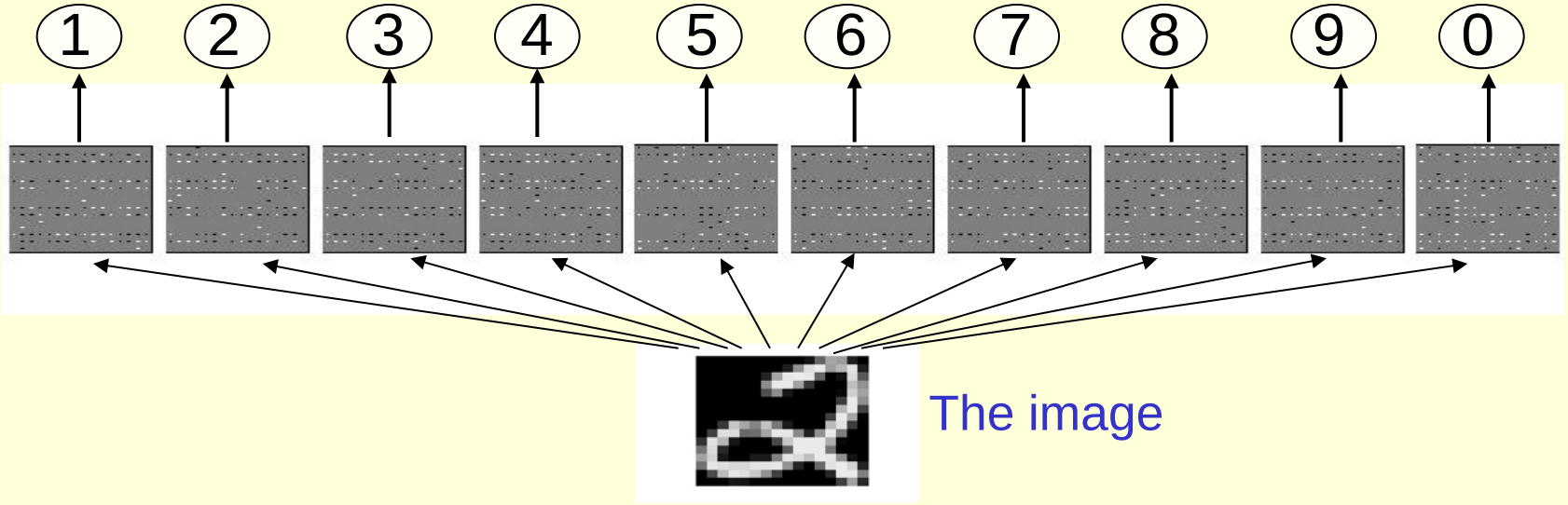


Give each output unit its own “map” of the input image and display the weight coming from each pixel in the location of that pixel in the map.

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

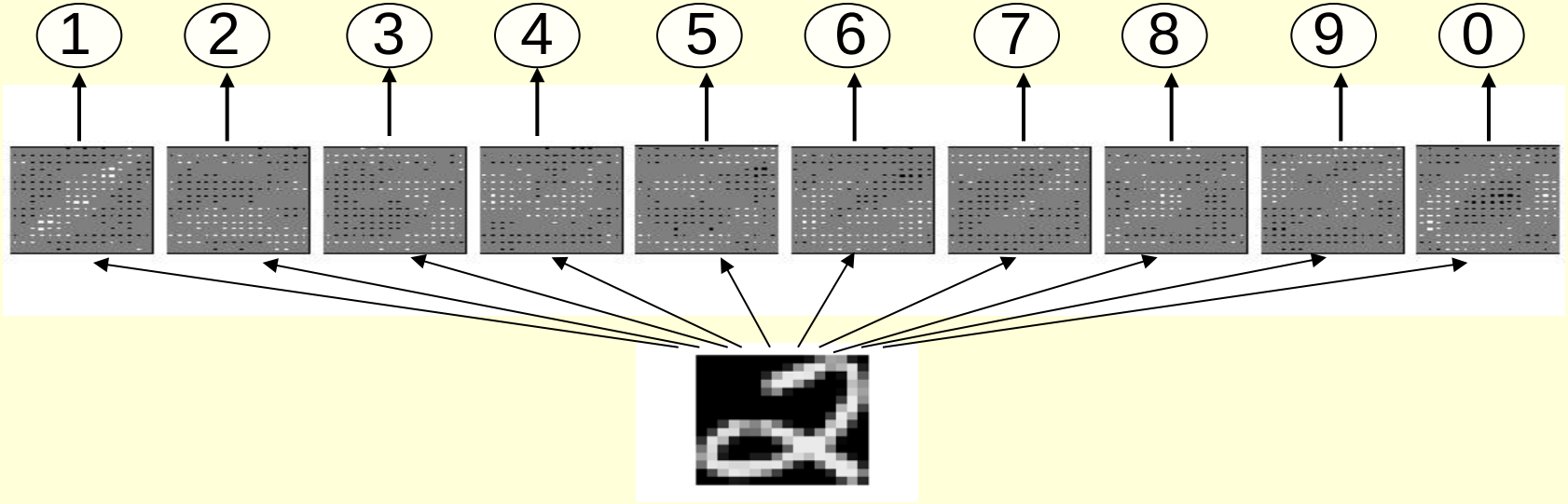
How to learn the weights



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

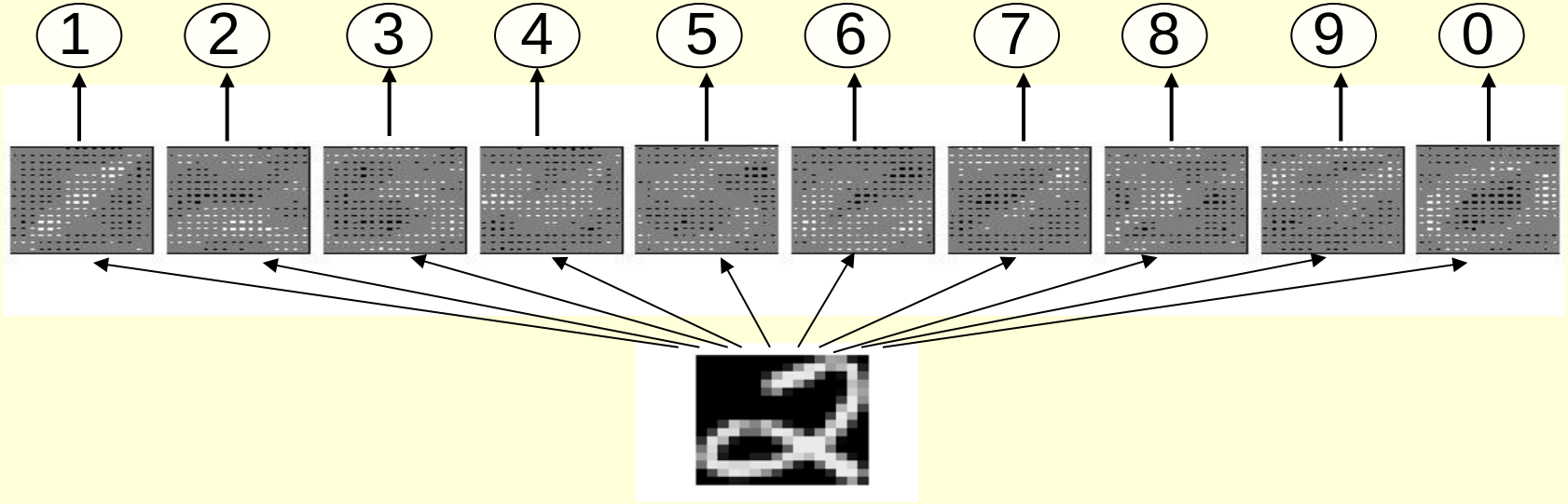
Then decrement the weights from active pixels to whatever class the network guesses.

Slide by Geoff Hinton



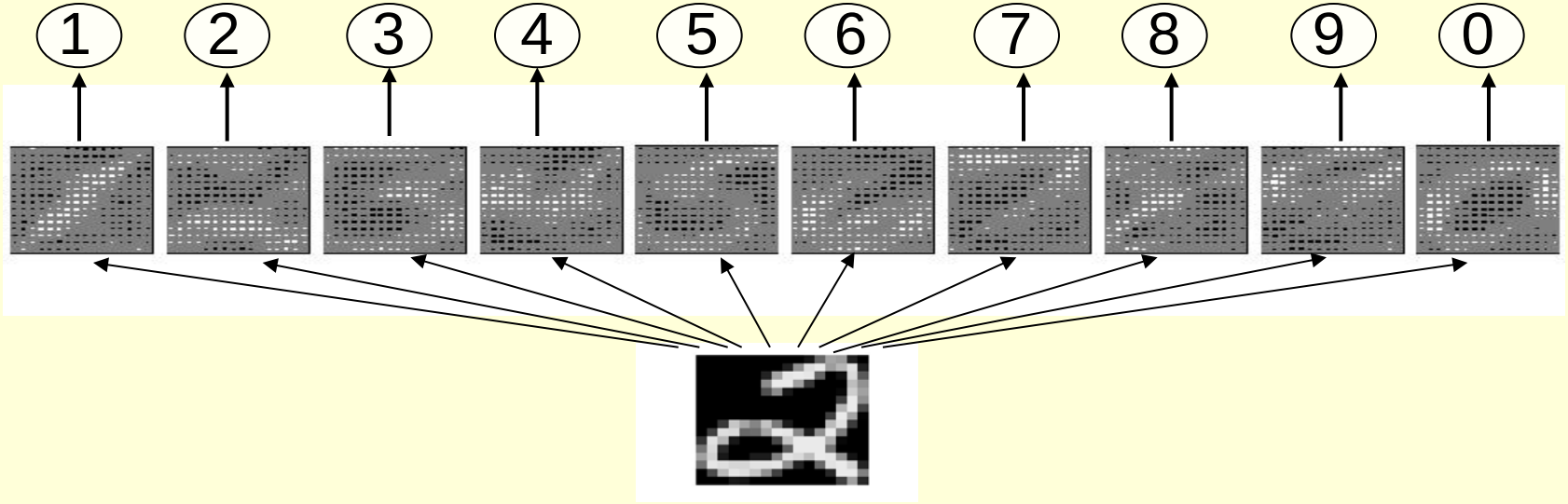
The image

Slide by Geoff Hinton



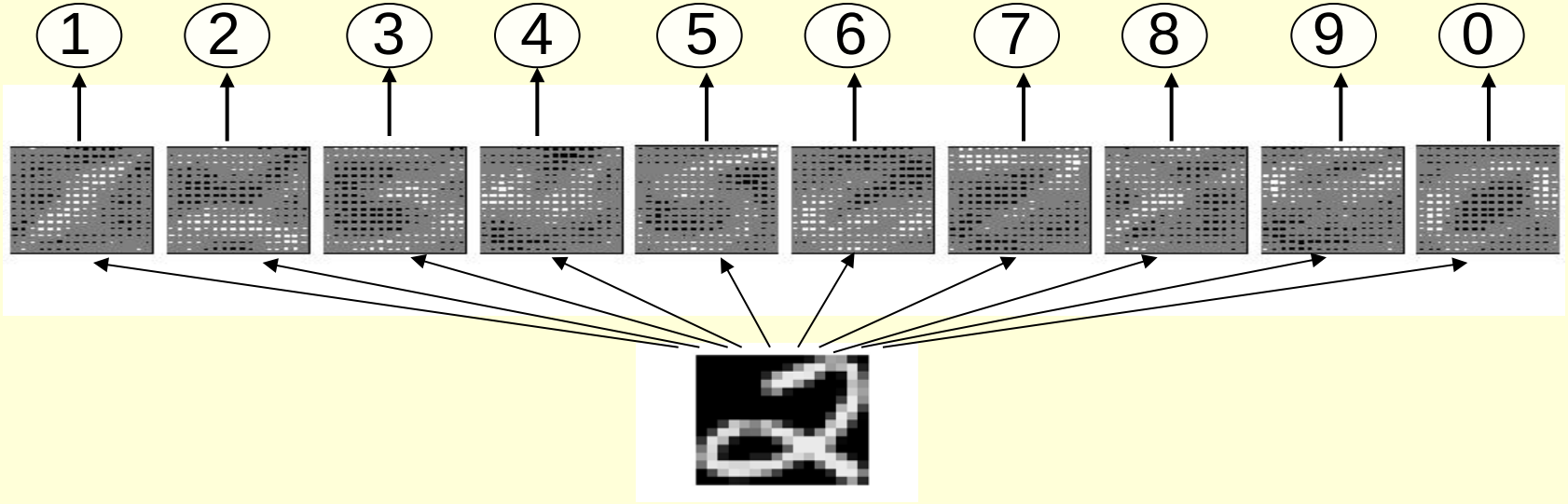
The image

Slide by Geoff Hinton



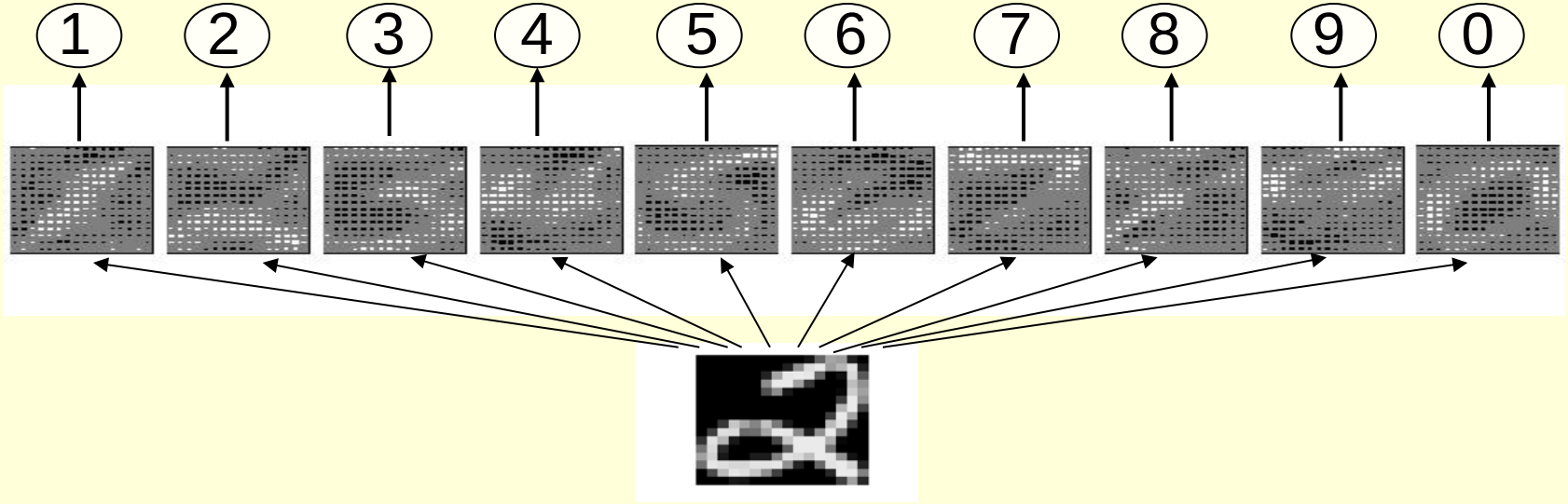
The image

Slide by Geoff Hinton



The image

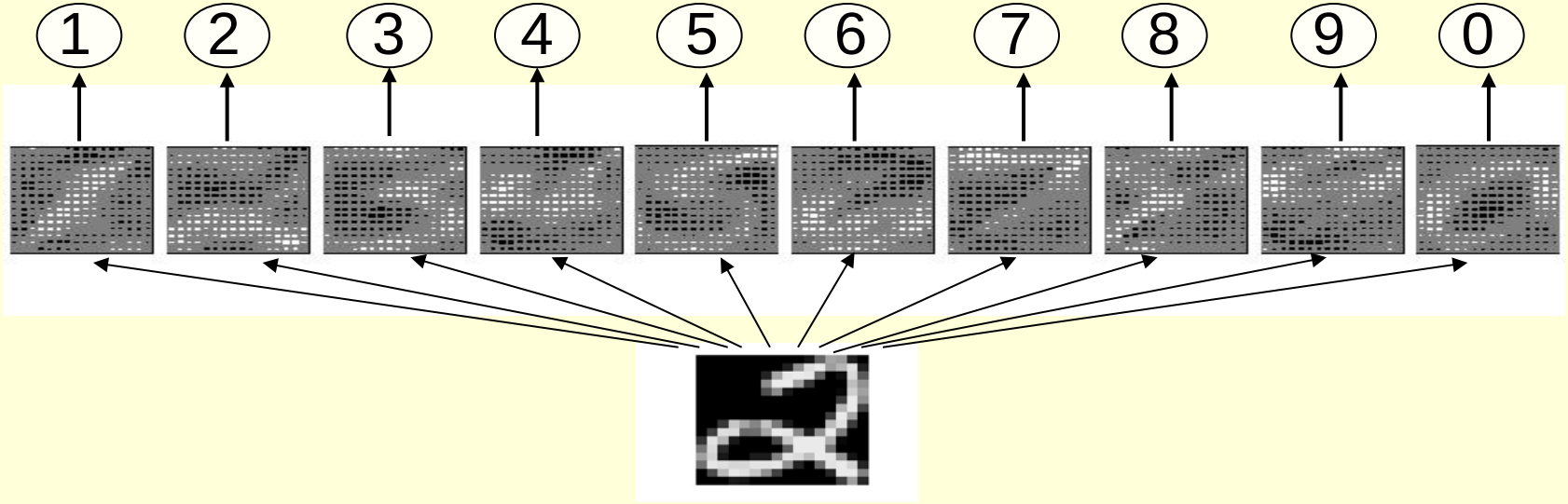
Slide by Geoff Hinton



The image

Slide by Geoff Hinton

The learned weights



The image

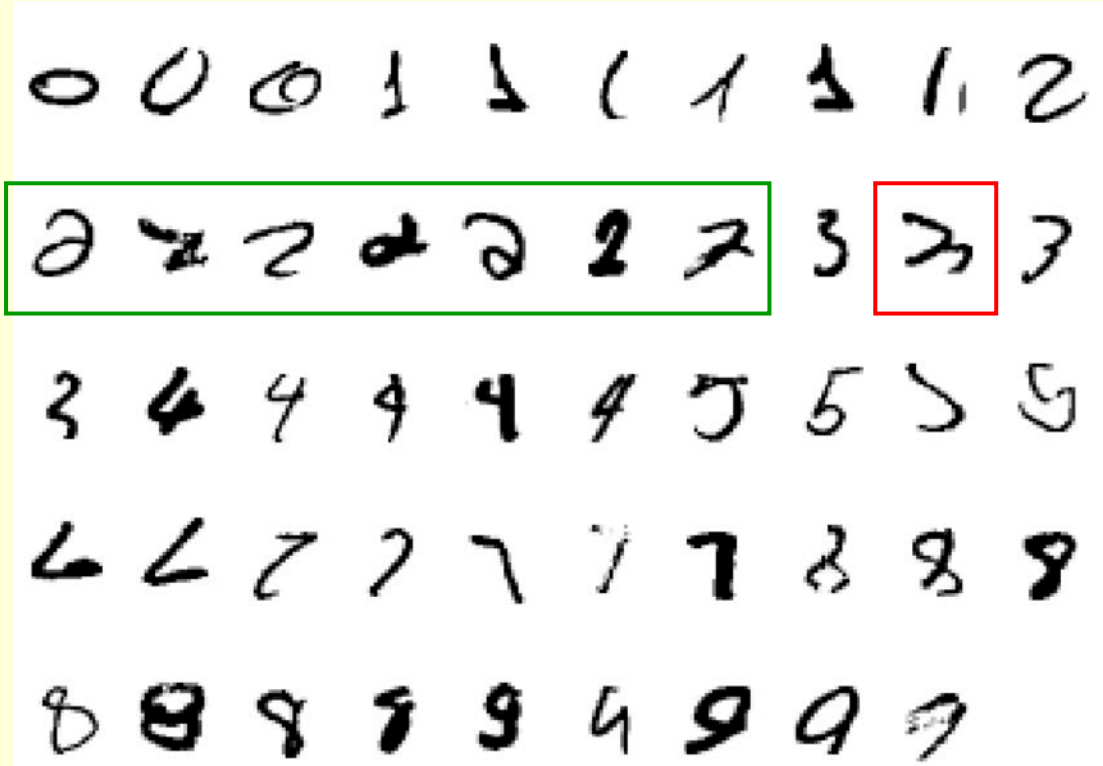
Slide by Geoff Hinton

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.

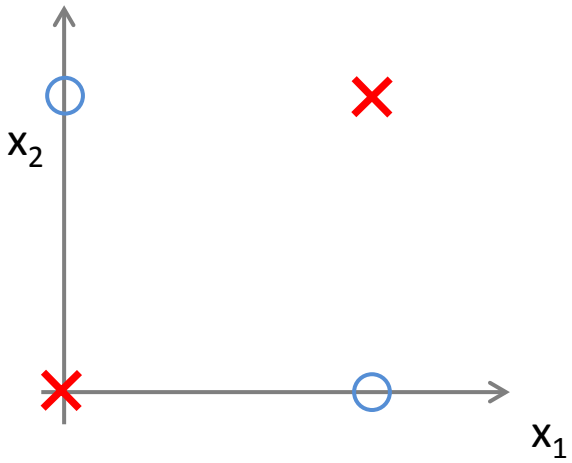
Slide by Geoff Hinton

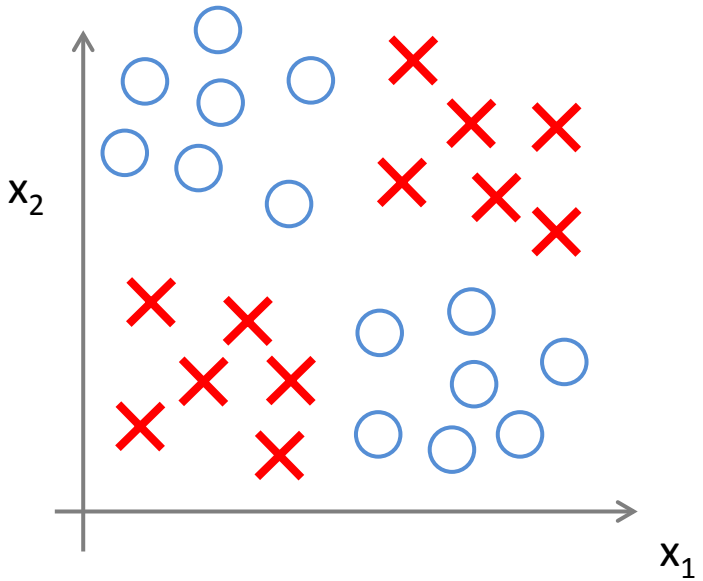
Examples of handwritten digits that can be recognized correctly the first time they are seen



Non-linear classification example: XOR/XNOR

x_1, x_2 are binary (0 or 1).





$y = x_1 \text{ XOR } x_2$

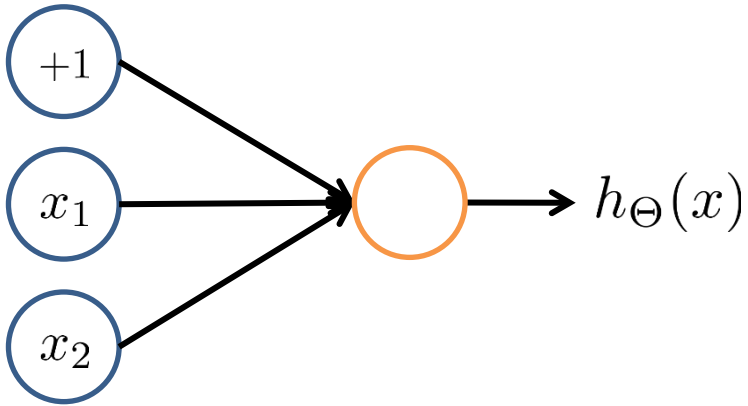
$x_1 \text{ XNOR } x_2$

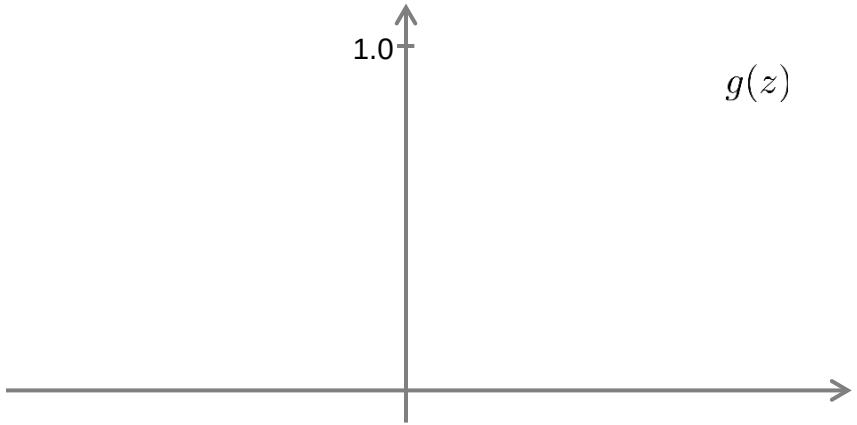
$\text{NOT } (x_1 \text{ XOR } x_2)$

Simple example: AND

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

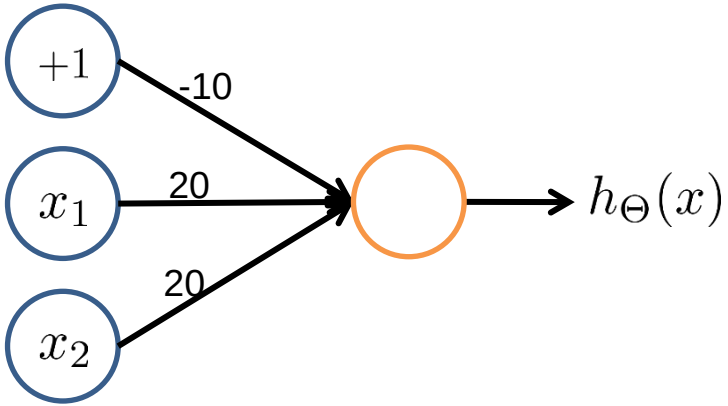
$y = x_1 \text{ AND } x_2$





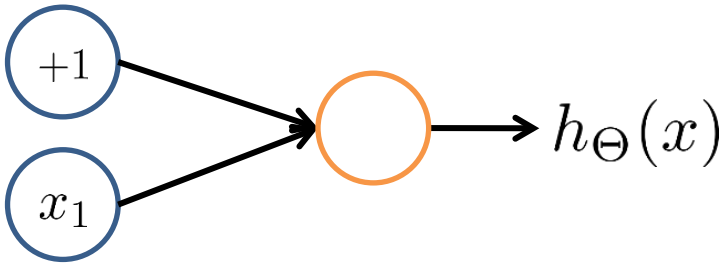
x_1	x_2	$h_\Theta(x)$
0	0	
0	1	
1	0	
1	1	

Example: OR function



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

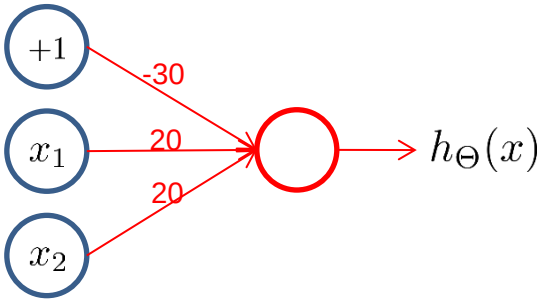
Negation:



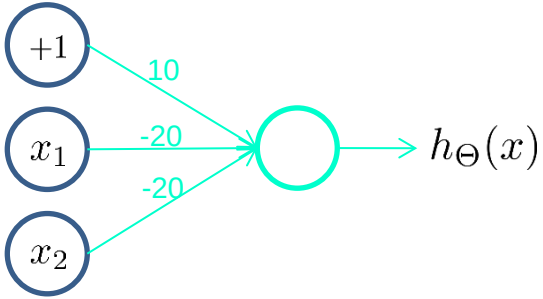
x_1	$h_\Theta(x)$
0	
1	

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

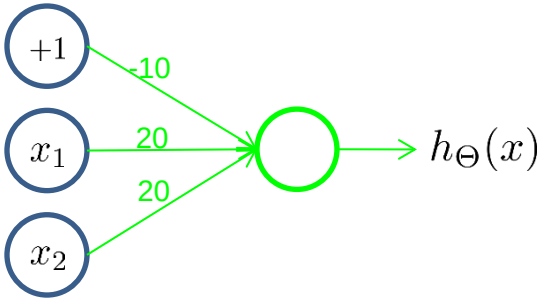
Putting it together: x_1 XNOR x_2



x_1 AND x_2



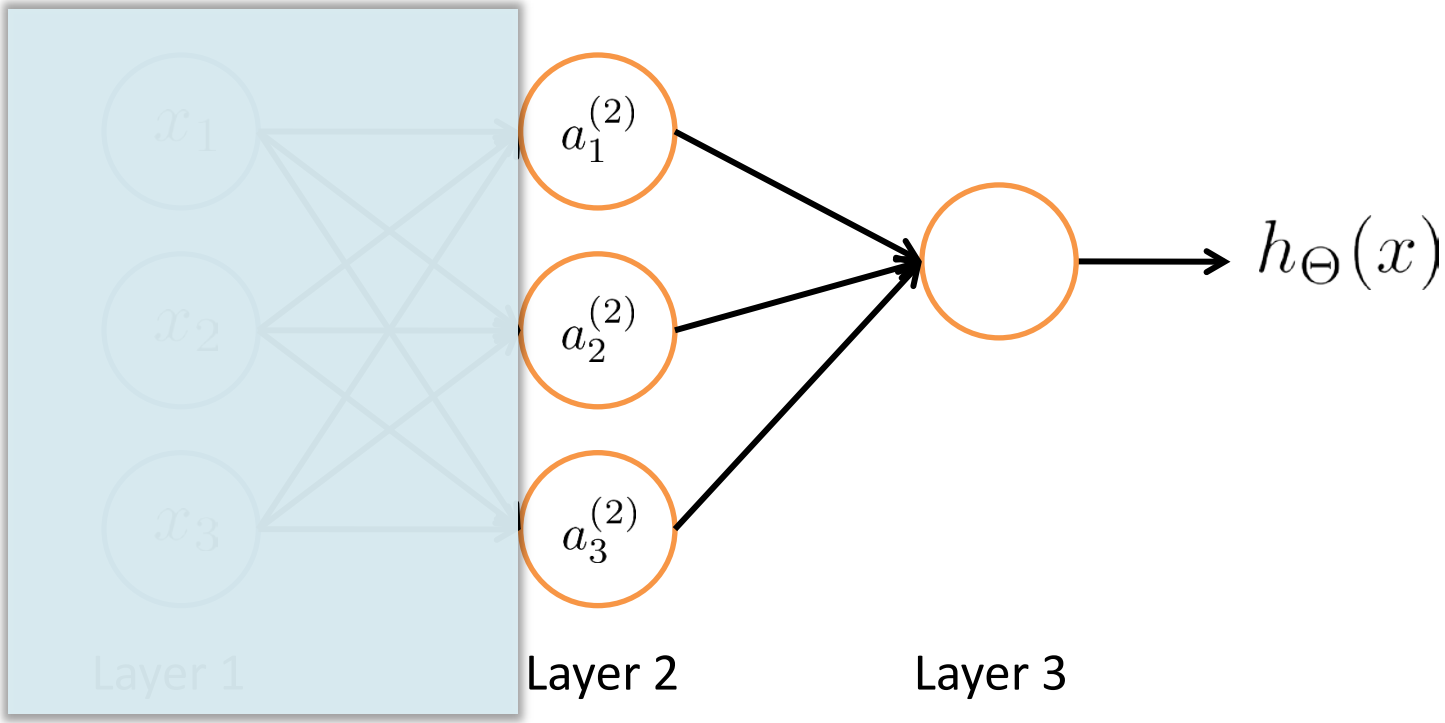
(NOT x_1) AND (NOT x_2)



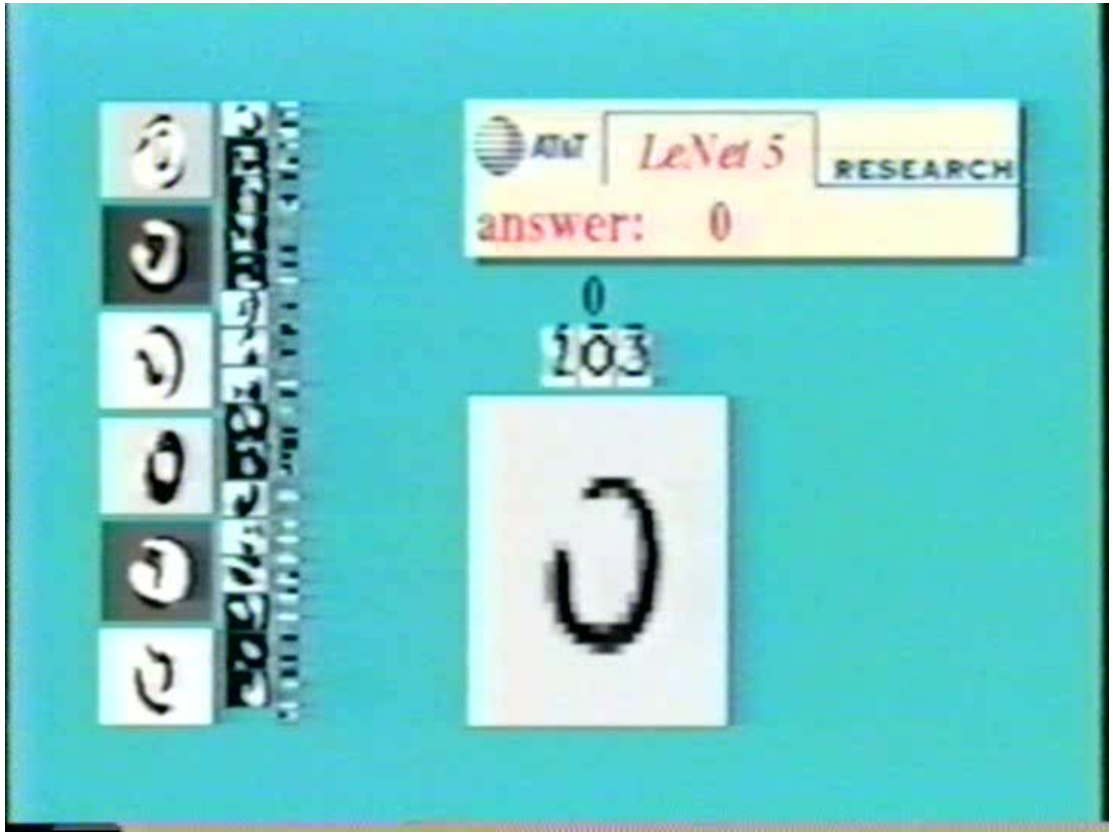
x_1 OR x_2

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

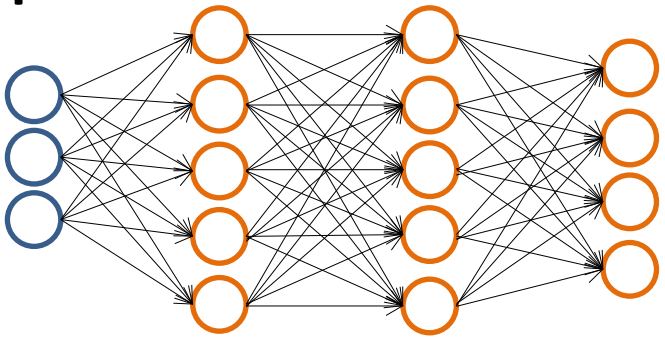


Handwritten digit classification



[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.

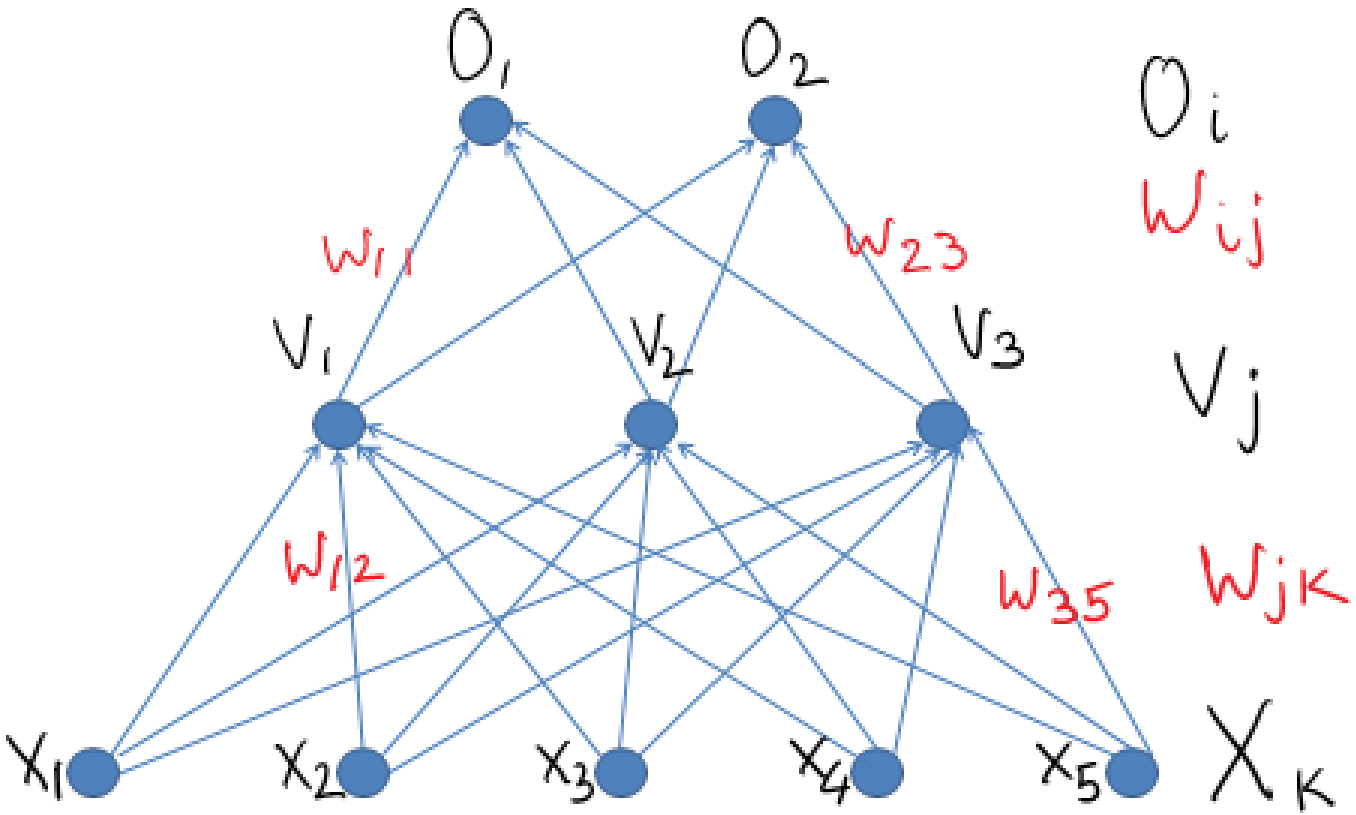
when pedestrian when car when motorcycle

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

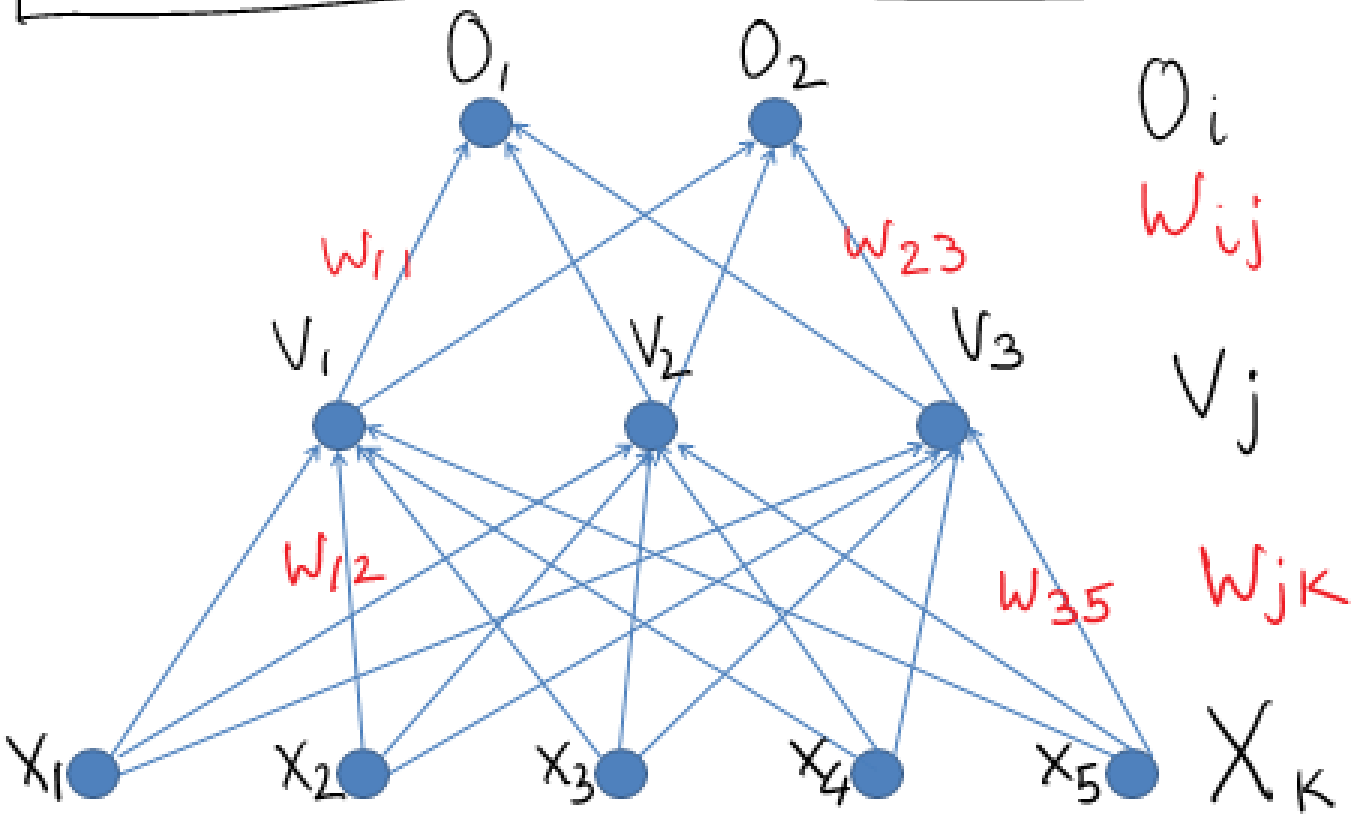
$y^{(i)}$ one of $\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$, $\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$, $\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$

pedestrian car motorcycle truck

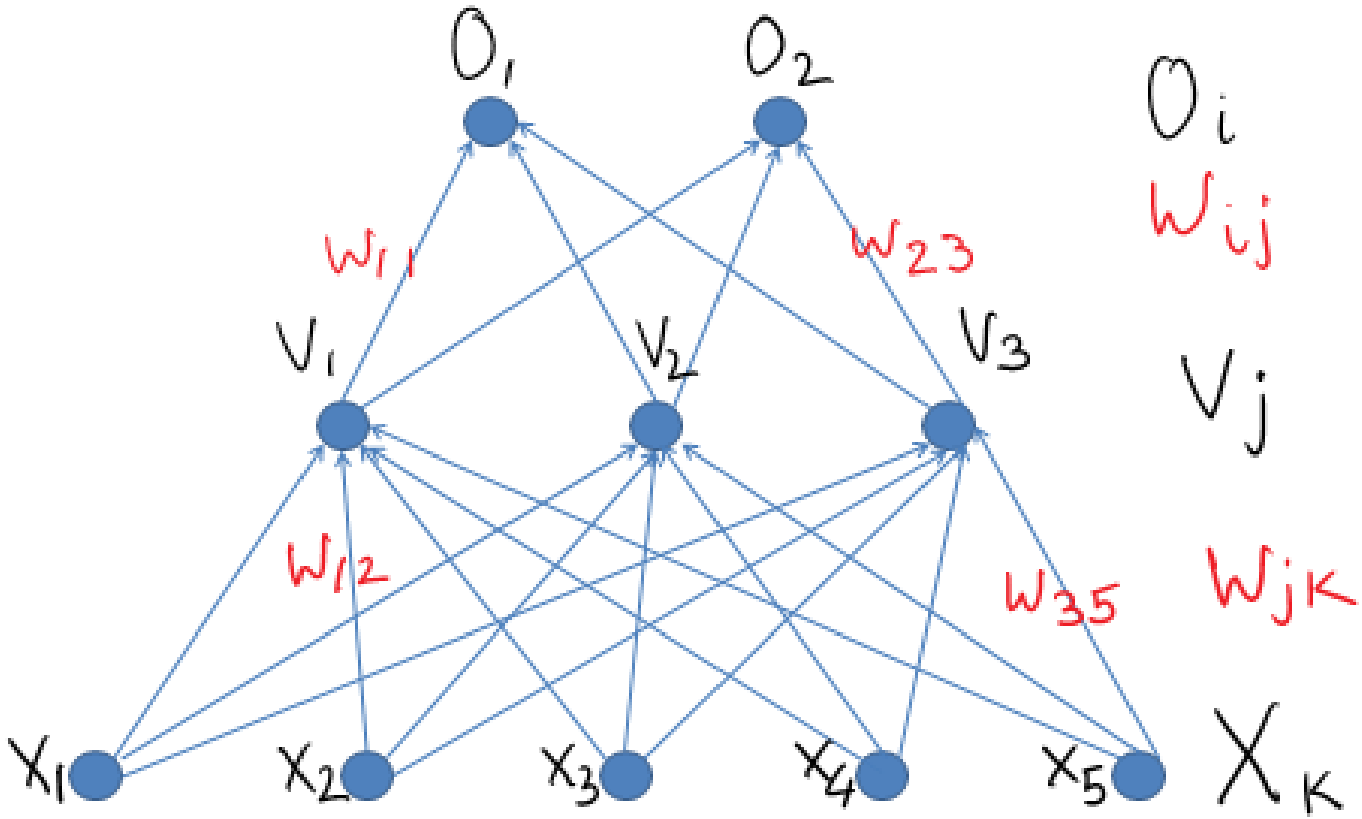
Two layer neural network



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



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



O_i

W_{ij}

V_j

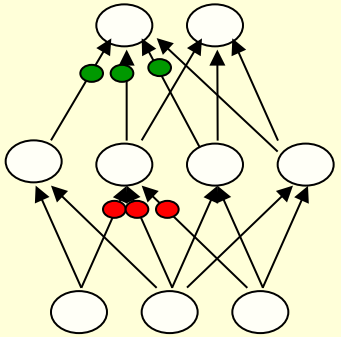
W_{jK}

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

Training a neural network

Gradient Descent

- **Numerical gradient**: easy to write ☺, slow ☹, approximate ☹
  - $O(N_w^2)$

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- **(native) analytic gradient**: exact ☺, compicated ☹, slow ☹
  - $O(N_w^2)$
- **back-propagation (cached analytic gradient)**: exact ☺, fast ☺, error-prone ☹
  - $O(N_w)$, similar to dynamic programming
  - glorified chain rule

In practice: Derive analytic gradient, check your implementation with numerical gradient

$$f(x,y) = xy \qquad \rightarrow \qquad \frac{\partial f}{\partial x} = y \qquad \frac{\partial f}{\partial y} = x$$

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$f(x+h) = f(x) + h \frac{df(x)}{dx}$$

$$f(x,y) = xy \qquad \rightarrow \qquad \frac{\partial f}{\partial x} = y \qquad \frac{\partial f}{\partial y} = x$$

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$f(x+h) = f(x) + h \frac{df(x)}{dx}$$

Example: x = 4, y = -3. => f(x,y) = -12

$$\frac{\partial f}{\partial x} = -3$$

$$\frac{\partial f}{\partial y} = 4$$

partial derivatives

$$\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}]$$

gradient

$$f(x,y) = xy \quad \rightarrow \quad \frac{\partial f}{\partial x} = y \qquad \frac{\partial f}{\partial y} = x$$

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$f(x+h) = f(x) + h \frac{df(x)}{dx}$$

Example: x = 4, y = -3. => f(x,y) = -12

$$\frac{\partial f}{\partial x} = -3 \qquad \frac{\partial f}{\partial y} = 4$$

partial derivatives

$$\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}]$$

gradient

Question: If I increase x by h, how would the output of f change?

Compound expressions: $f(x, y, z) = (x + y)z$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Compound expressions: $f(x, y, z) = (x + y)z$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

Compound expressions: $f(x, y, z) = (x + y)z$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

```
# set some inputs
x = -2; y = 5; z = -4

# perform the forward pass
q = x + y # q becomes 3
f = q * z # f becomes -12

# perform the backward pass (backpropagation) in reverse order:
# first backprop through f = q * z
dfdz = q # df/fz = q, so gradient on z becomes 3
dfdq = z # df/dq = z, so gradient on q becomes -4
# now backprop through q = x + y
dfdx = 1.0 * dfdq # dq/dx = 1. And the multiplication here is the chain rule!
dfdy = 1.0 * dfdq # dq/dy = 1
```

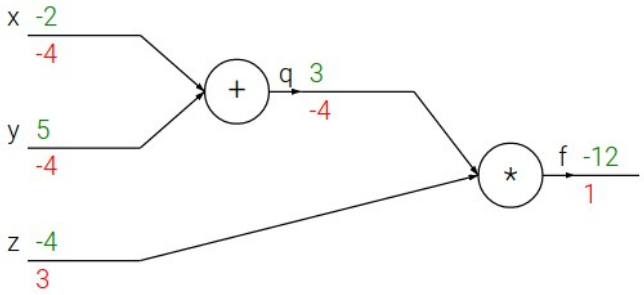
Compound expressions: $f(x, y, z) = (x + y)z$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

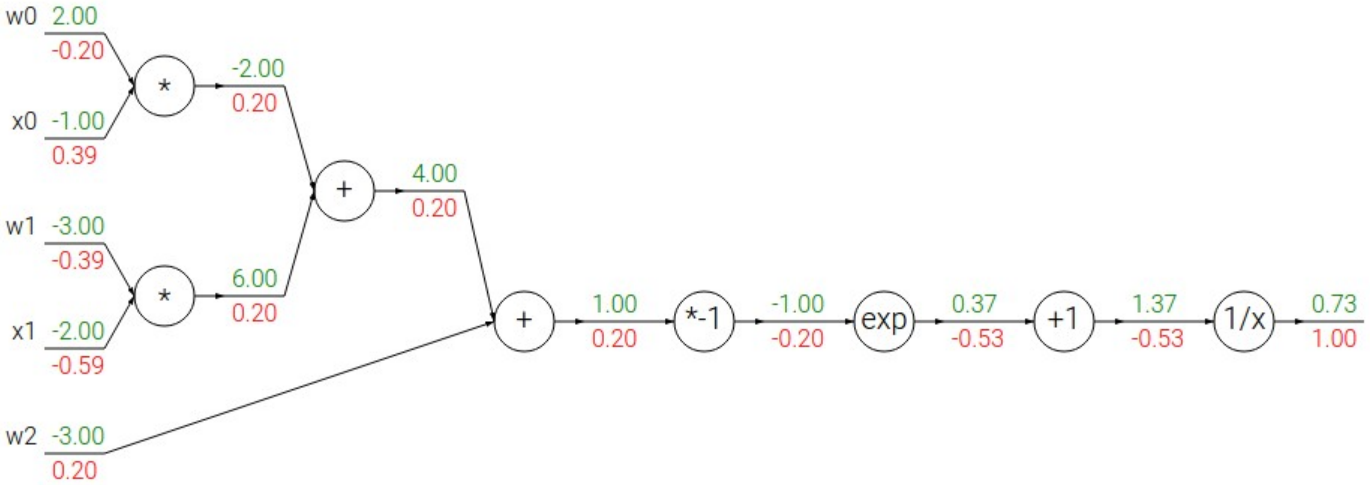



```
# set some inputs
x = -2; y = 5; z = -4

# perform the forward pass
q = x + y # q becomes 3
f = q * z # f becomes -12

# perform the backward pass (backpropagation) in reverse order:
# first backprop through f = q * z
dfdz = q # df/fz = q, so gradient on z becomes 3
dfdq = z # df/dq = z, so gradient on q becomes -4
# now backprop through q = x + y
dfdx = 1.0 * dfdq # dq/dx = 1. And the multiplication here is the chain rule!
dfdy = 1.0 * dfdq # dq/dy = 1
```

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



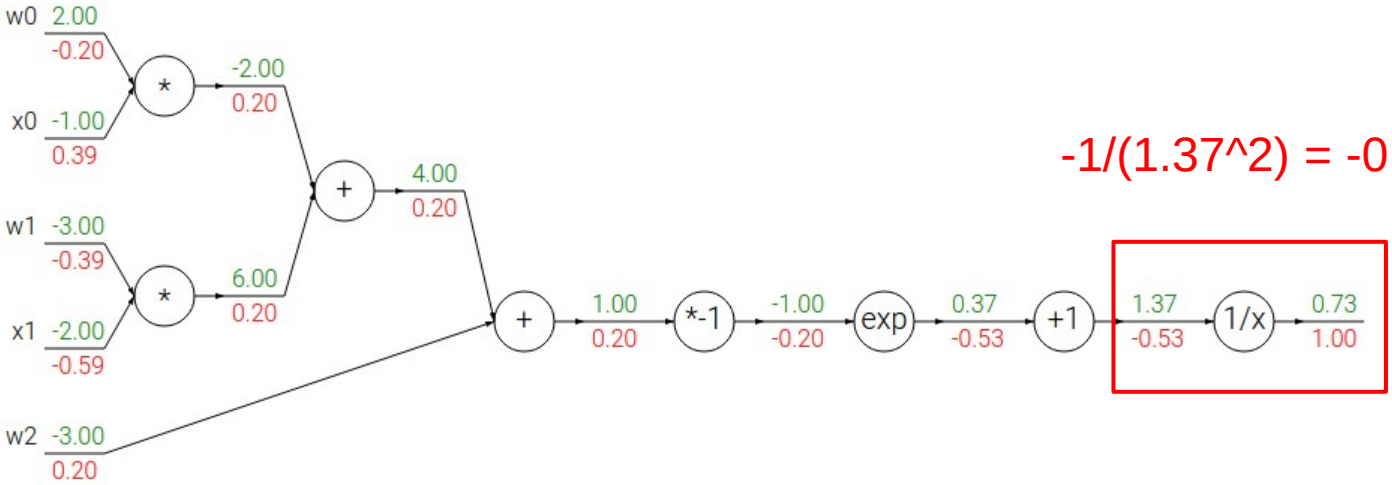
$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



-1/(1.37^2) = -0.53

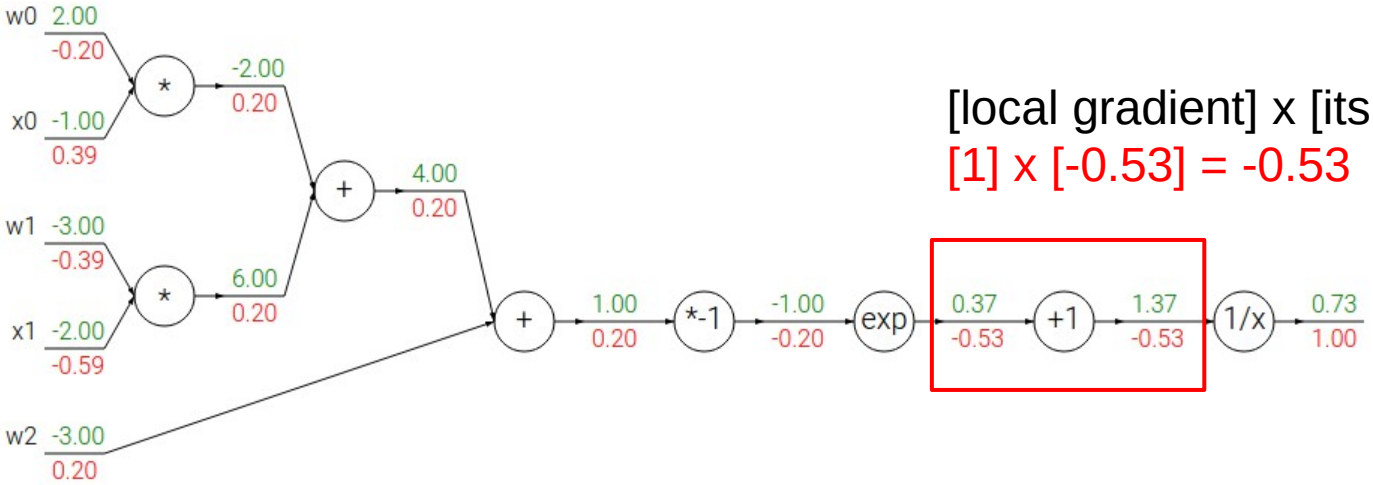
$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



[local gradient] x [its gradient]
[1] x [-0.53] = -0.53

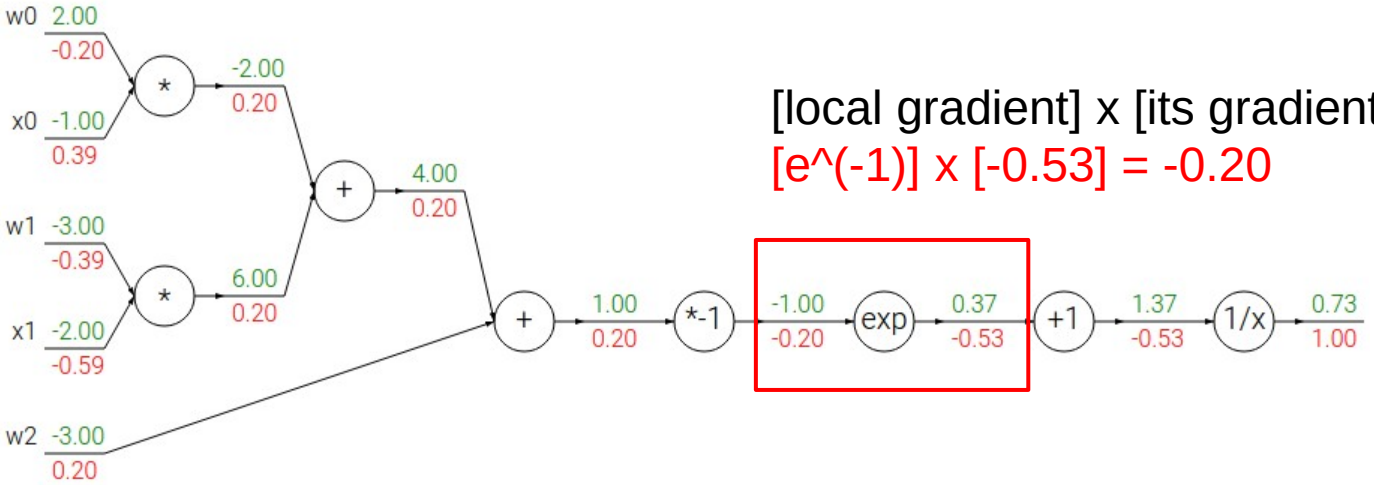
$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

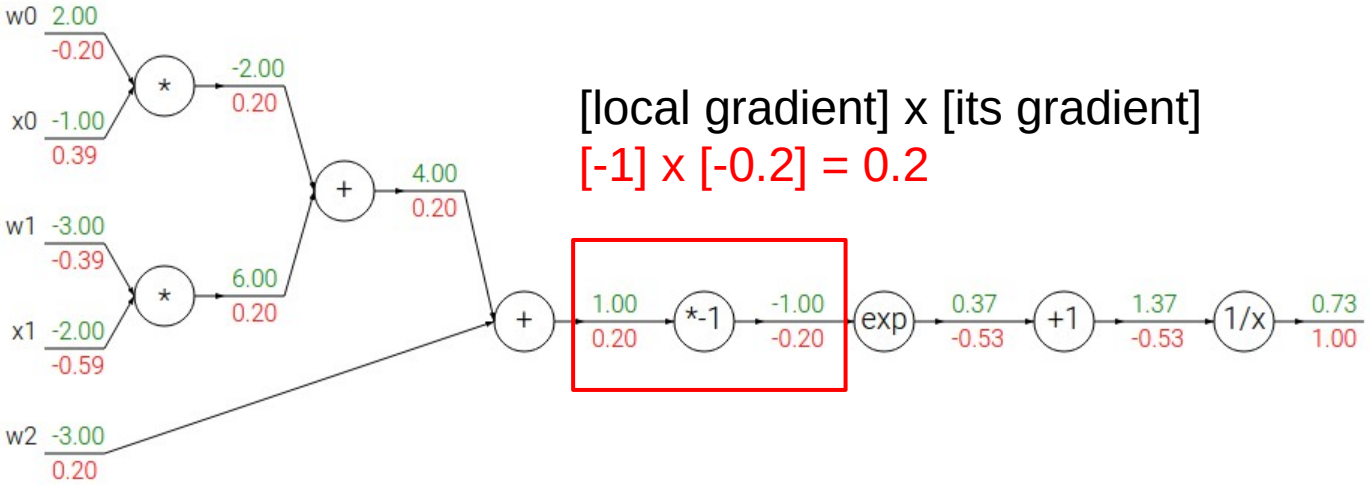


[local gradient] x [its gradient]
[e^(-1)] x [-0.53] = -0.20

$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x \qquad f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a \qquad f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



[local gradient] x [its gradient]

[-1] x [-0.2] = 0.2

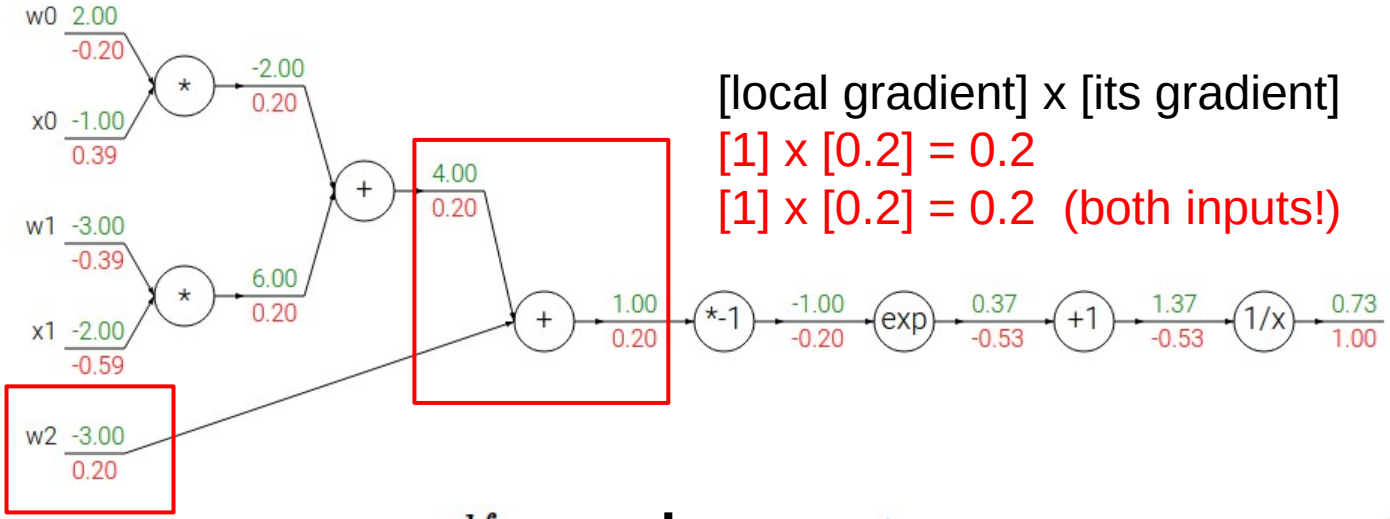
$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



[local gradient] x [its gradient]

[1] x [0.2] = 0.2

[1] x [0.2] = 0.2 (both inputs!)

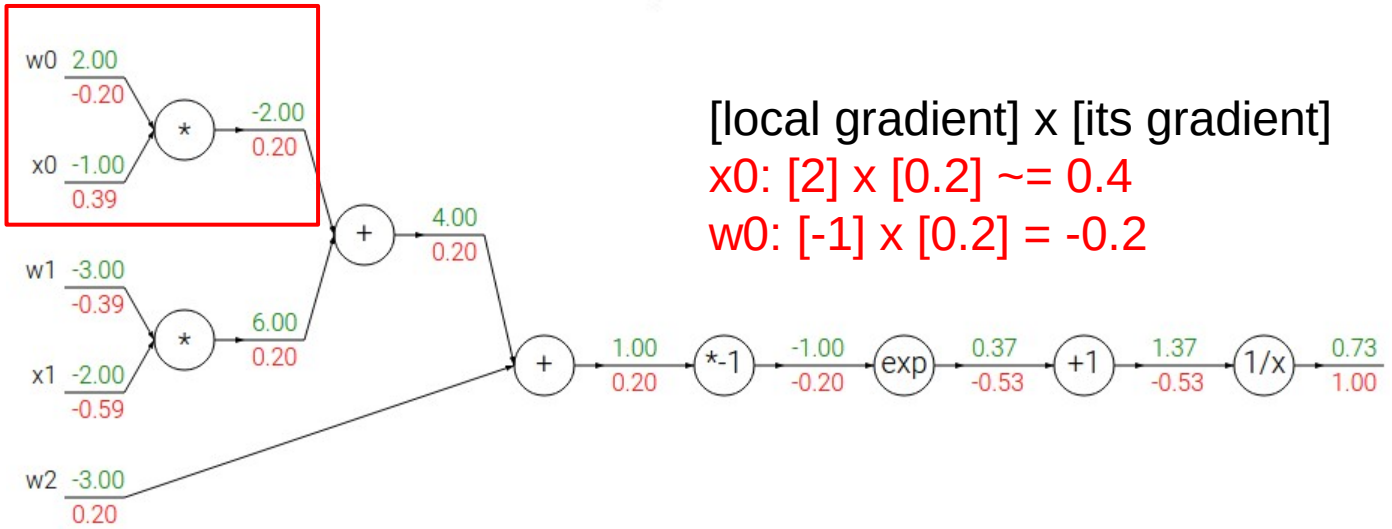
$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Another example: $$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



[local gradient] x [its gradient]
x0: [2] x [0.2] ~= 0.4
w0: [-1] x [0.2] = -0.2

$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

Every gate during backprop computes, for all its inputs:

[LOCAL GRADIENT] x [GATE GRADIENT]

Can be computed right away, even during forward pass

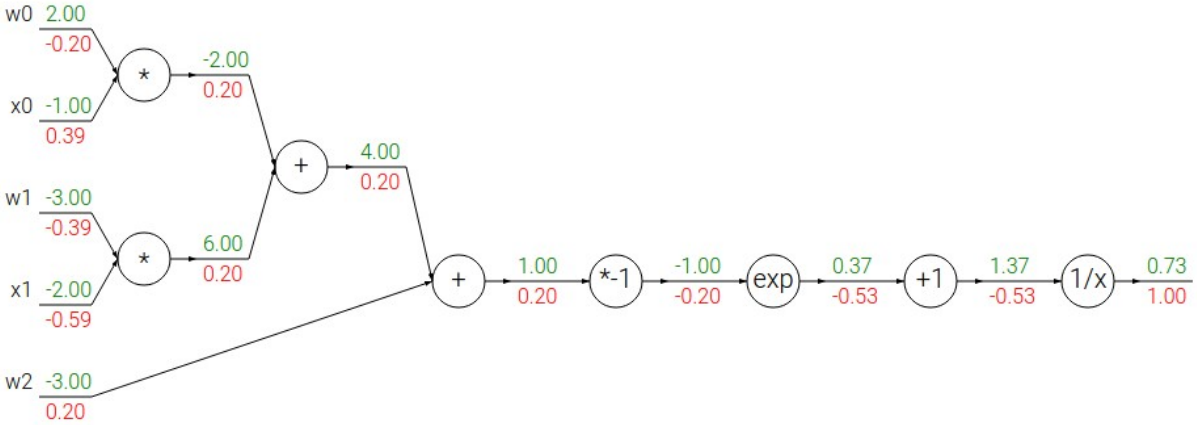
The gate receives this during backpropagation

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

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

sigmoid function

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$

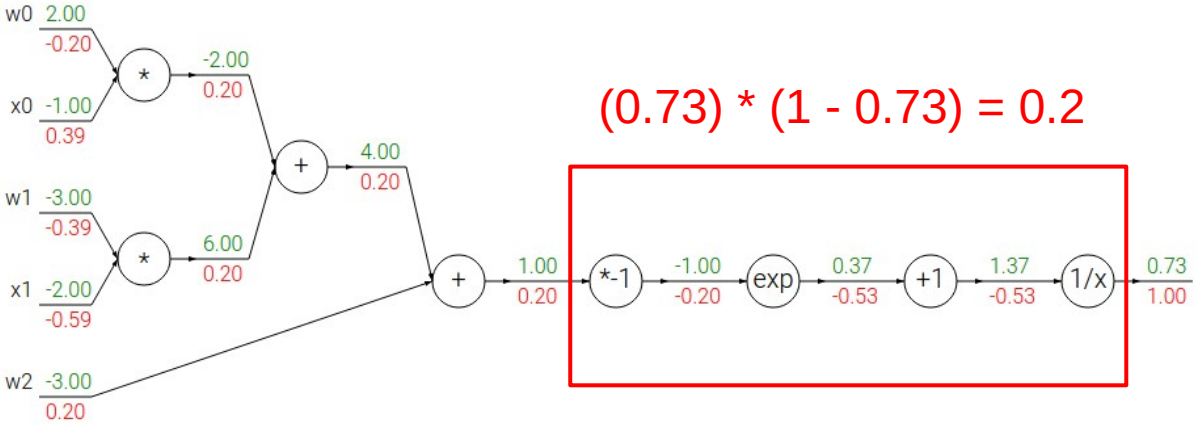


$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

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

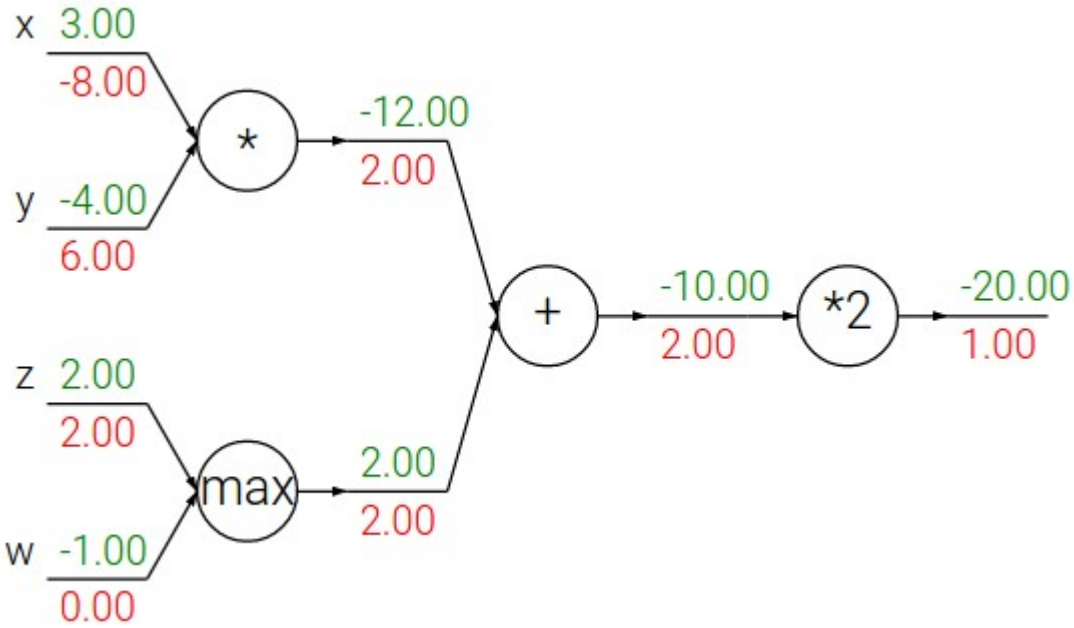
sigmoid function

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$



Patterns in backward flow

add gate: gradient distributor
max gate: gradient router
mul gate: gradient… “switcher”?



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.
  - 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.

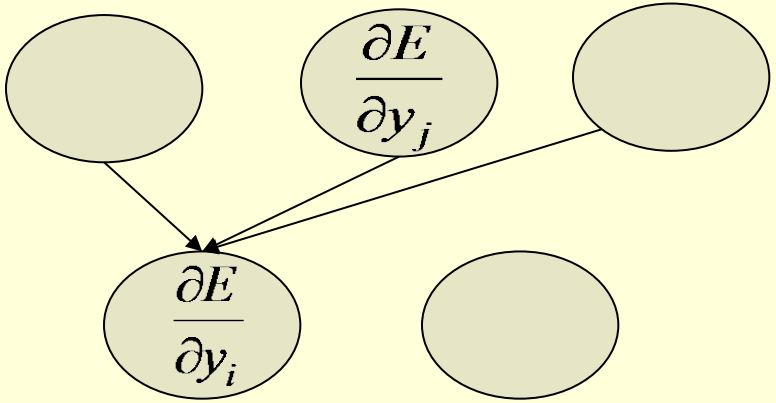
Slide by Geoff Hinton

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:

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

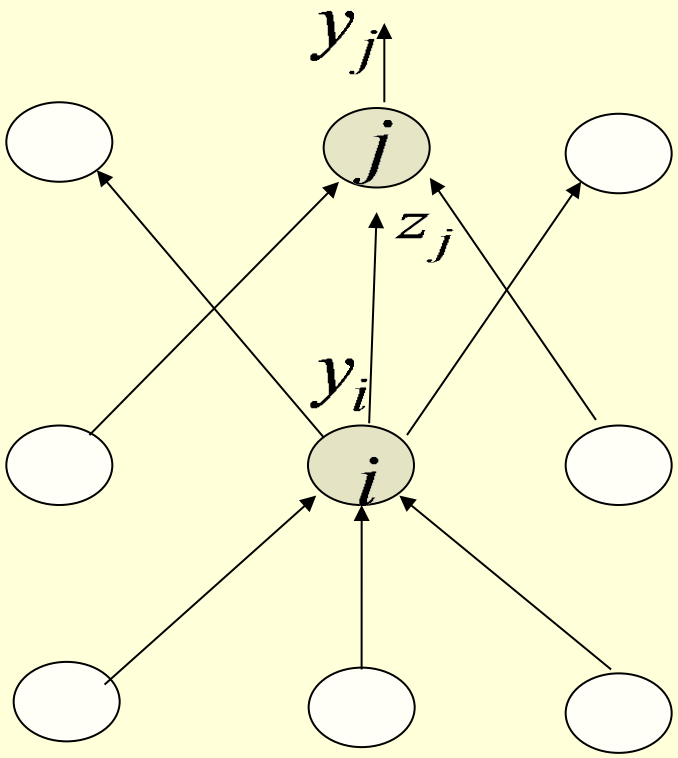
$$\frac{\partial z}{\partial w_i} = x_i \qquad \frac{\partial z}{\partial x_i} = w_i$$

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

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

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

Backpropagating dE/dy



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

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

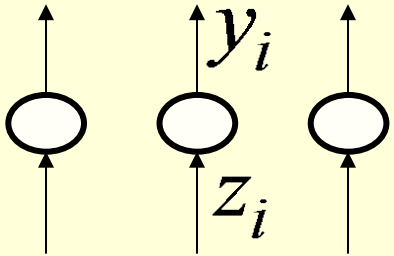
$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}}\frac{\partial E}{\partial z_j} = y_i\frac{\partial E}{\partial z_j}$$

Problems with squared error

- The squared error measure has some drawbacks:
  - If the desired output is 1 and the actual output is 0.00000001 there is almost no gradient for a logistic unit to fix up the error.
  - If we are trying to assign probabilities to mutually exclusive class labels, we know that the outputs should sum to 1, but we are depriving the network of this knowledge.
- Is there a different cost function that works better?
  - Yes: Force the outputs to represent a probability distribution across discrete alternatives.

Softmax

The output units in a softmax group use a non-local non-linearity:



softmax group

$$y_i = \frac{e^{z_i}}{\sum_{j \in group} e^{z_j}}$$

$$\frac{\partial y_i}{\partial z_i} = y_i (1 - y_i)$$

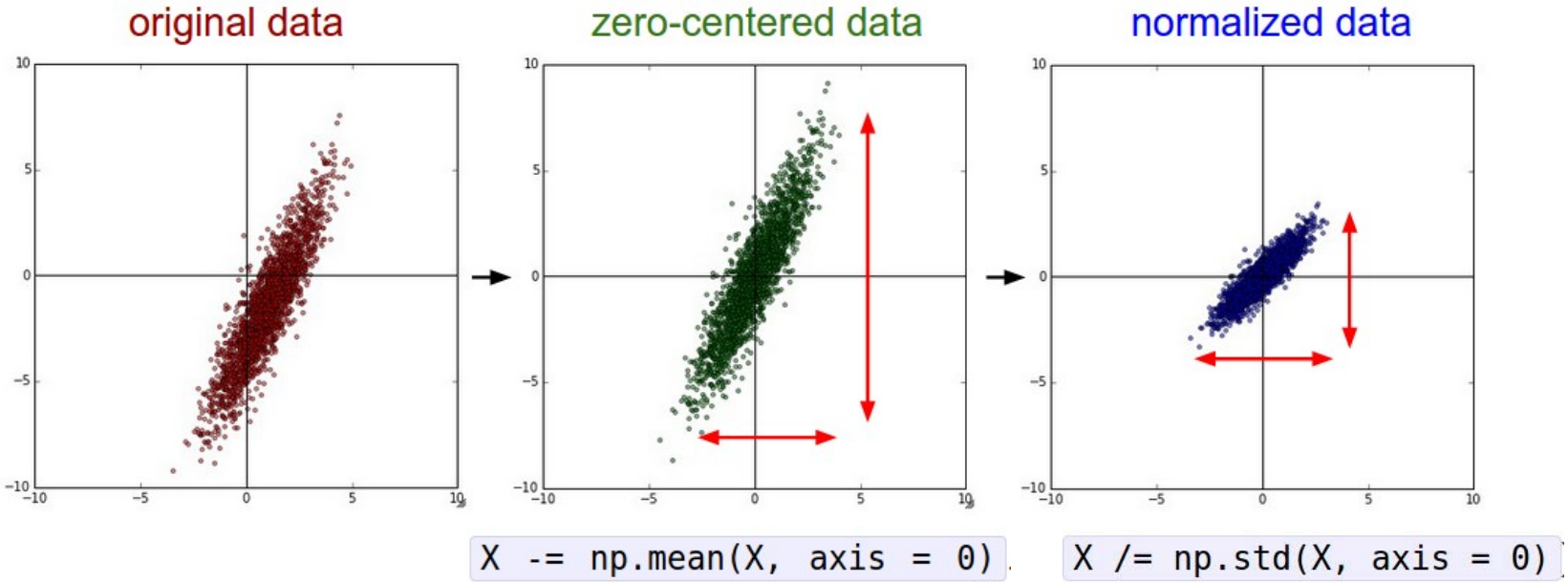
Cross-entropy: the right cost function to use with softmax

- The right cost function is the negative log probability of the right answer.
- C has a very big gradient when the target value is 1 and the output is almost zero.

$$C = -\sum_j t_j \log y_j$$

target value

Preprocessing the data



(Assume X [NxD] is data matrix,
each example in a row)

Initializing Weights

- set weights to small random numbers

```
W = 0.001* np.random.randn(D,H)
```

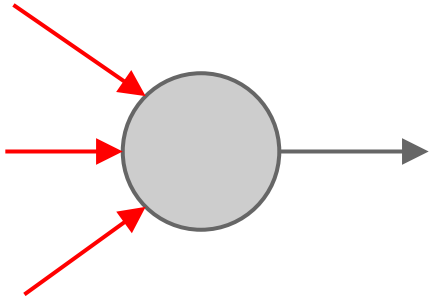
(Matrix of small numbers drawn randomly from a gaussian)

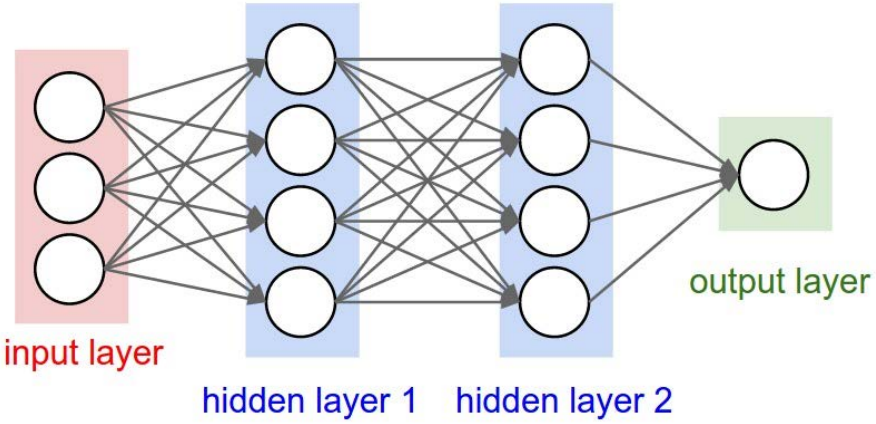
- set biases to zero

Regularization knobs

- L2 regularization $\frac{1}{2}\lambda w^2$
- L1 regularization $\lambda \mid w \mid$
- L1 + L2 can also be combined
- Max norm constraint

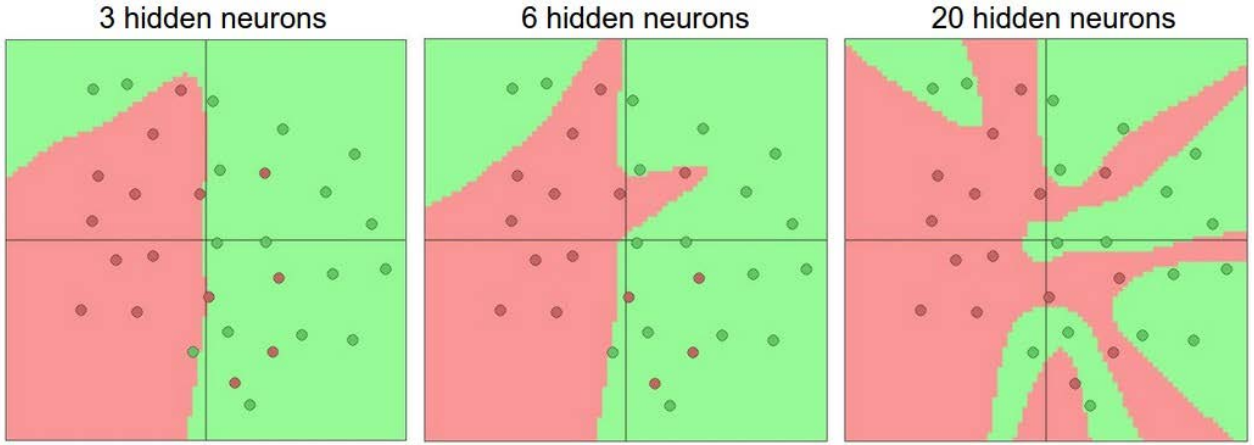
L1 is “sparsity inducing” (many weights become almost exactly zero)





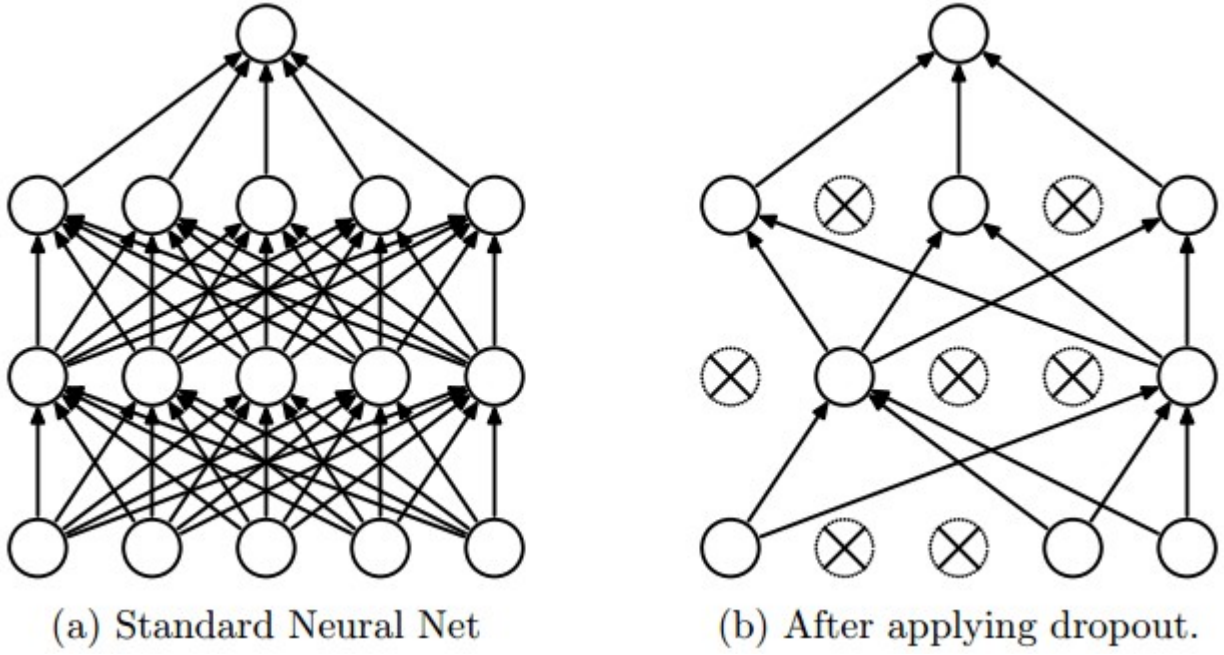
Seemingly unrelated: **Model Ensembles**

- One way to *always* improve final accuracy: take several trained models and average their predictions



Regularization: **Dropout**

"randomly set some neurons to zero"

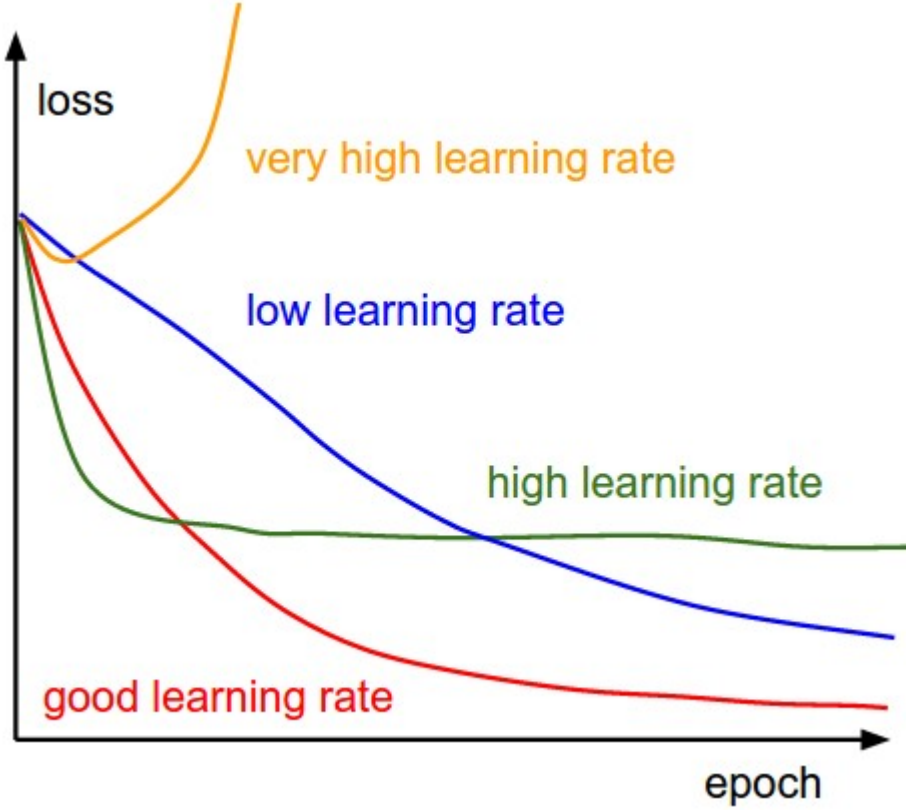


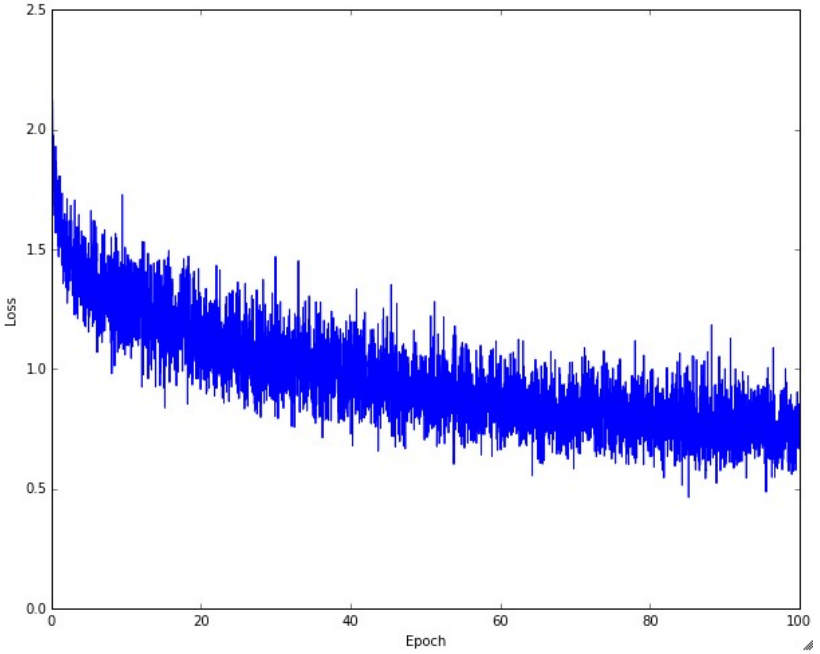
(a) Standard Neural Net

(b) After applying dropout.

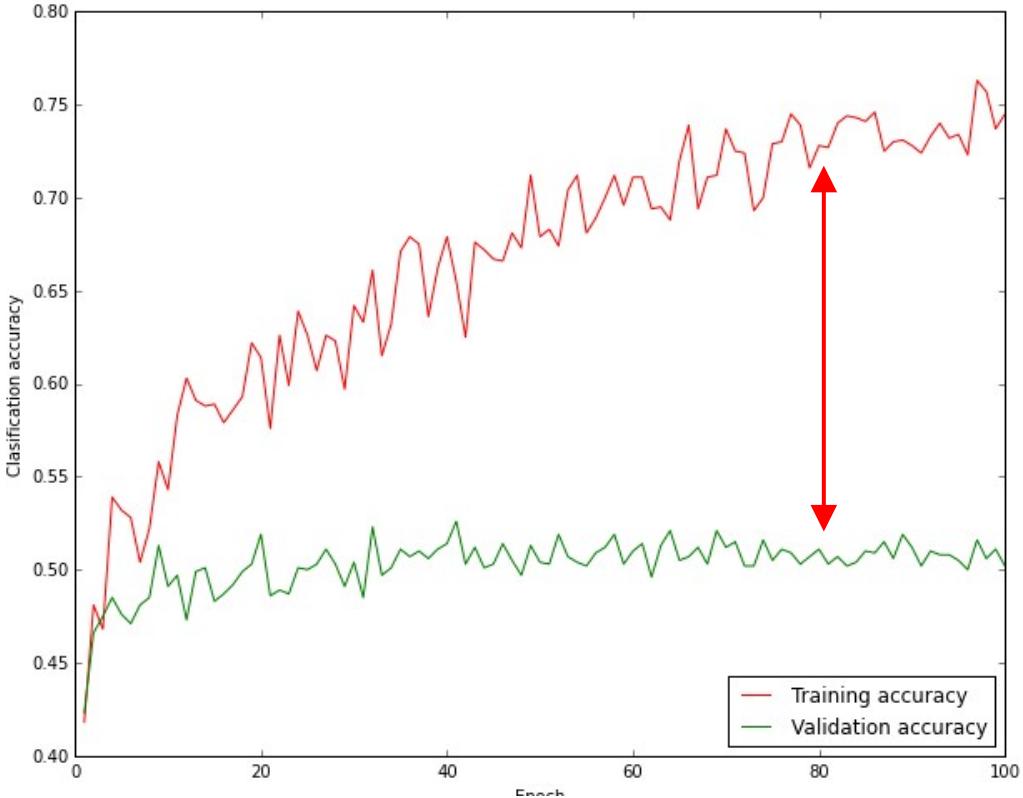
[Srivastava et al.]

Baby-sitting your network: monitoring loss





Baby-sitting your network: monitoring accuracy



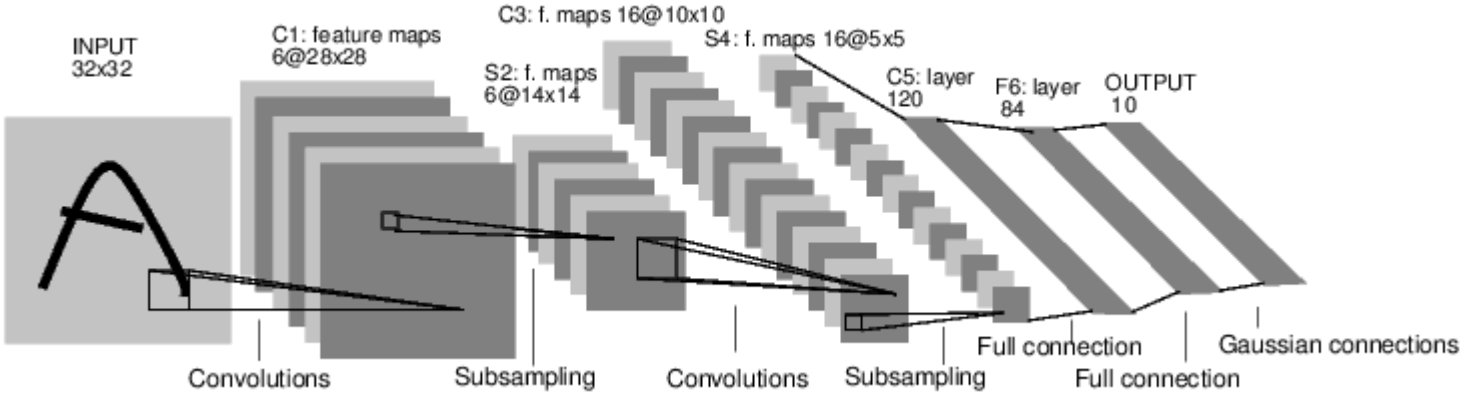
big gap = overfitting

=> increase regularization strength

no gap

=> increase model capacity

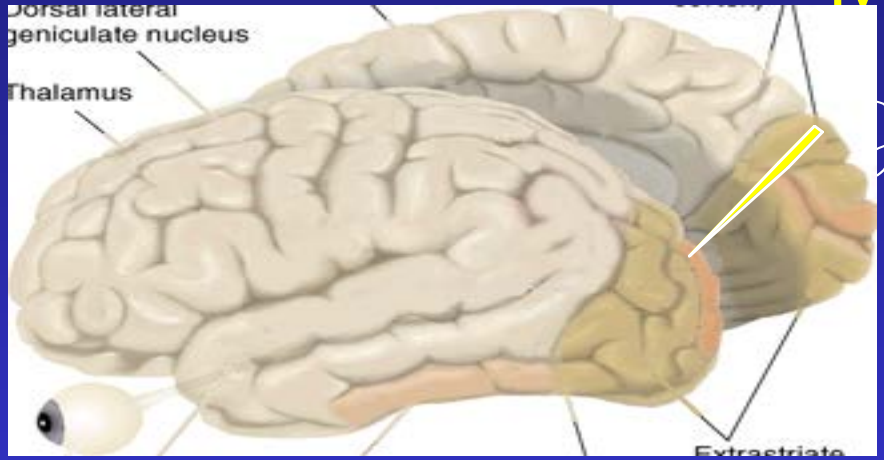
Convolutional Neural Networks (CNNs)



[LeNet-5, LeCun 1980]

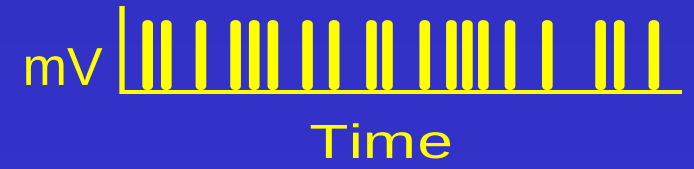
Single Cell Recording

Microelectrode

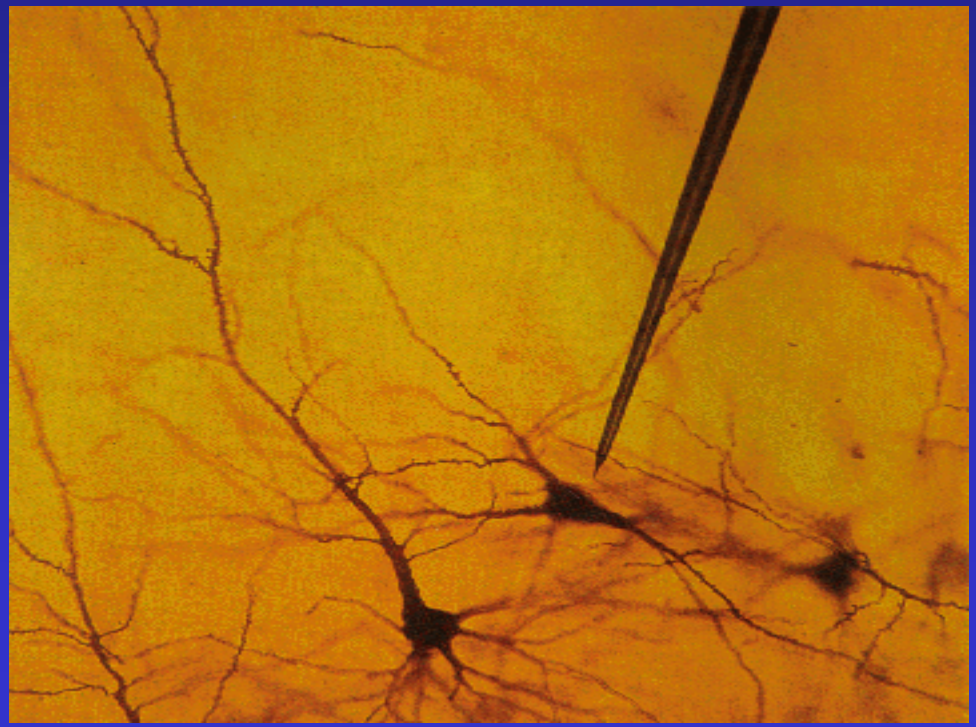


Amplifier

Electrical response
(action potentials)

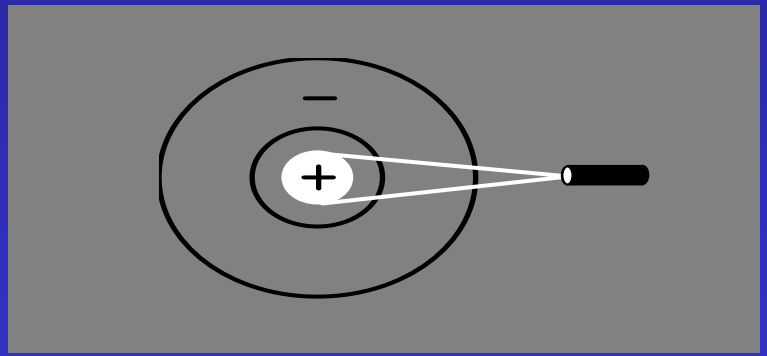


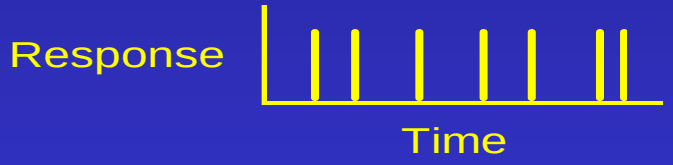
Single Cell Recording



Retinal Receptive Fields

Receptive field structure in ganglion cells:
On-center Off-surround



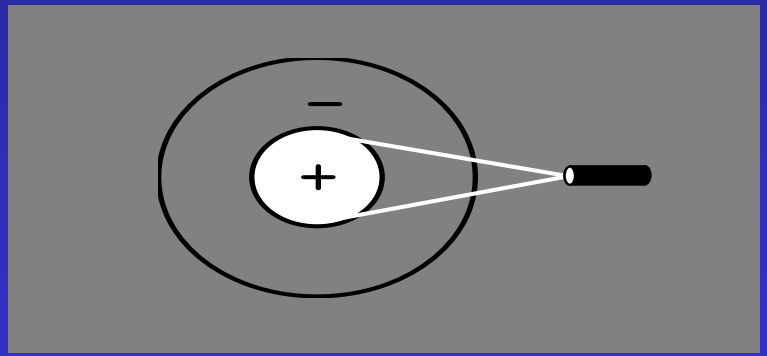


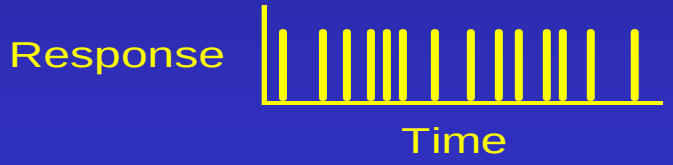
Stimulus condition

Electrical response

Retinal Receptive Fields

Receptive field structure in ganglion cells:
On-center Off-surround



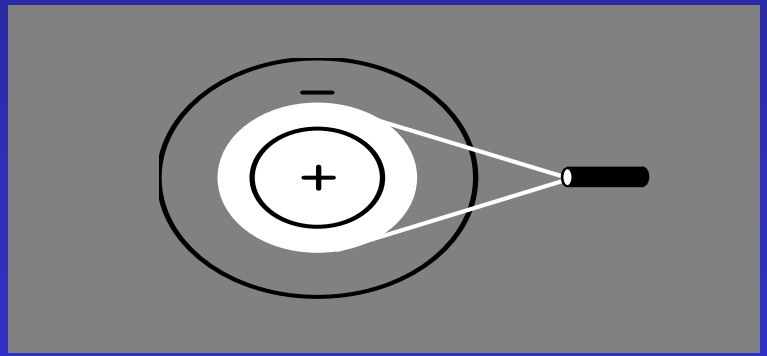


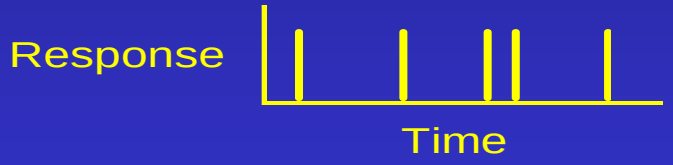
Stimulus condition

Electrical response

Retinal Receptive Fields

Receptive field structure in ganglion cells:
On-center Off-surround



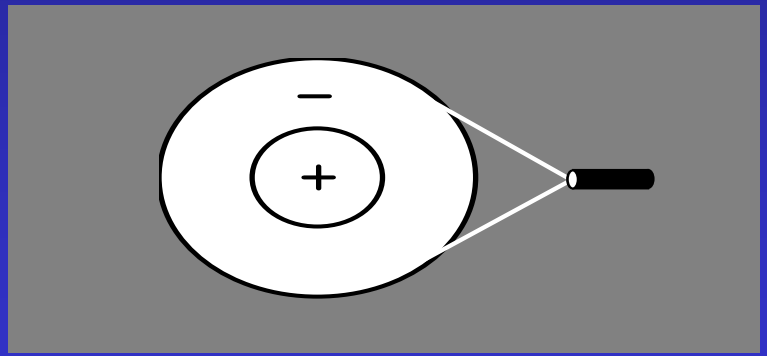


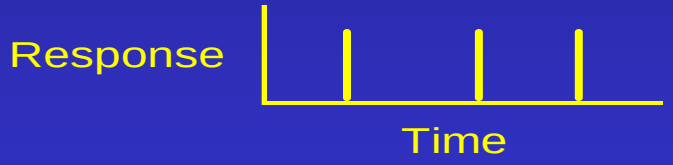
Stimulus condition

Electrical response

Retinal Receptive Fields

Receptive field structure in ganglion cells:
On-center Off-surround



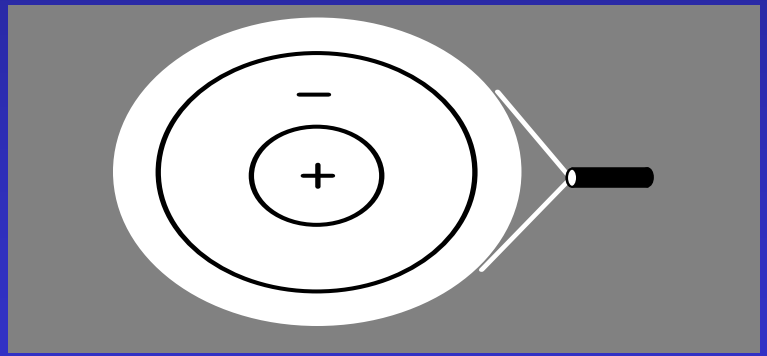


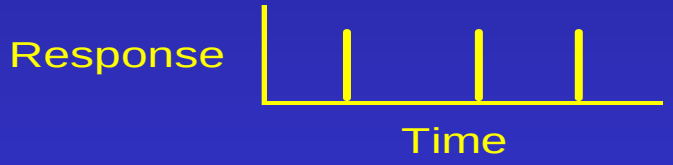
Stimulus condition

Electrical response

Retinal Receptive Fields

Receptive field structure in ganglion cells:
On-center Off-surround



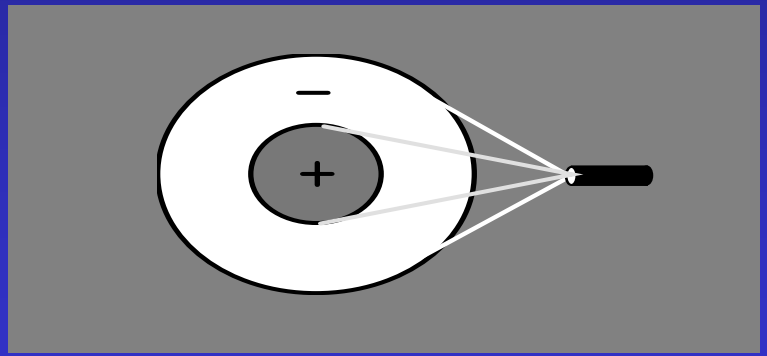


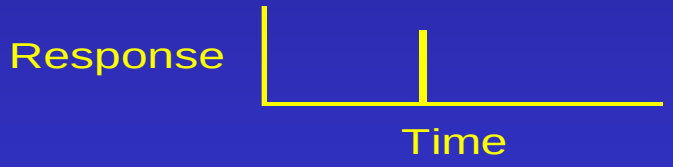
Stimulus condition

Electrical response

Retinal Receptive Fields

Receptive field structure in ganglion cells:
On-center Off-surround



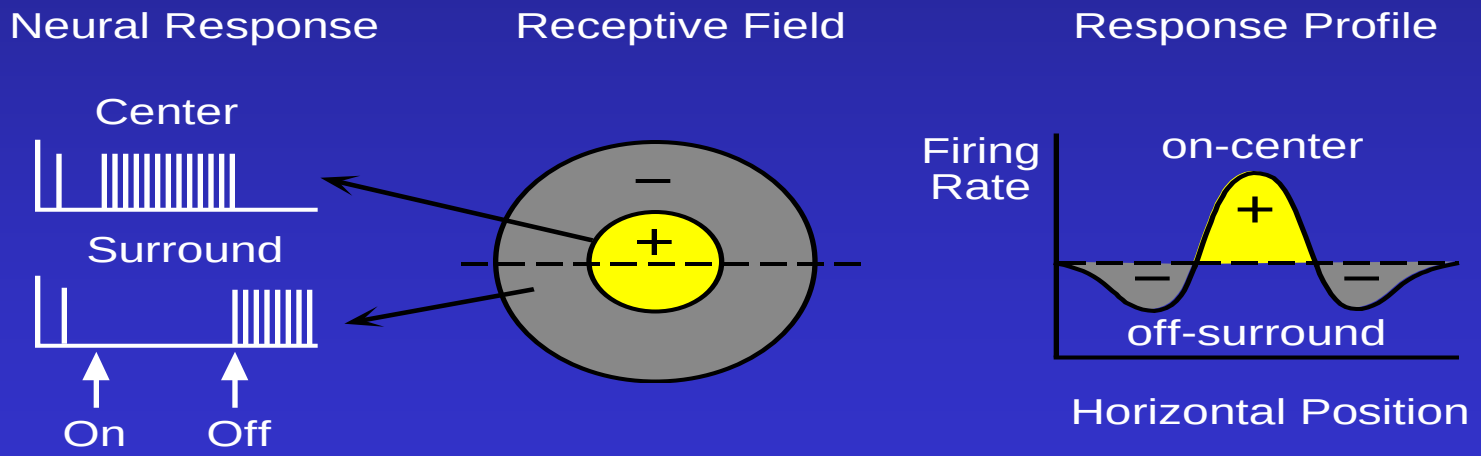


Stimulus condition

Electrical response

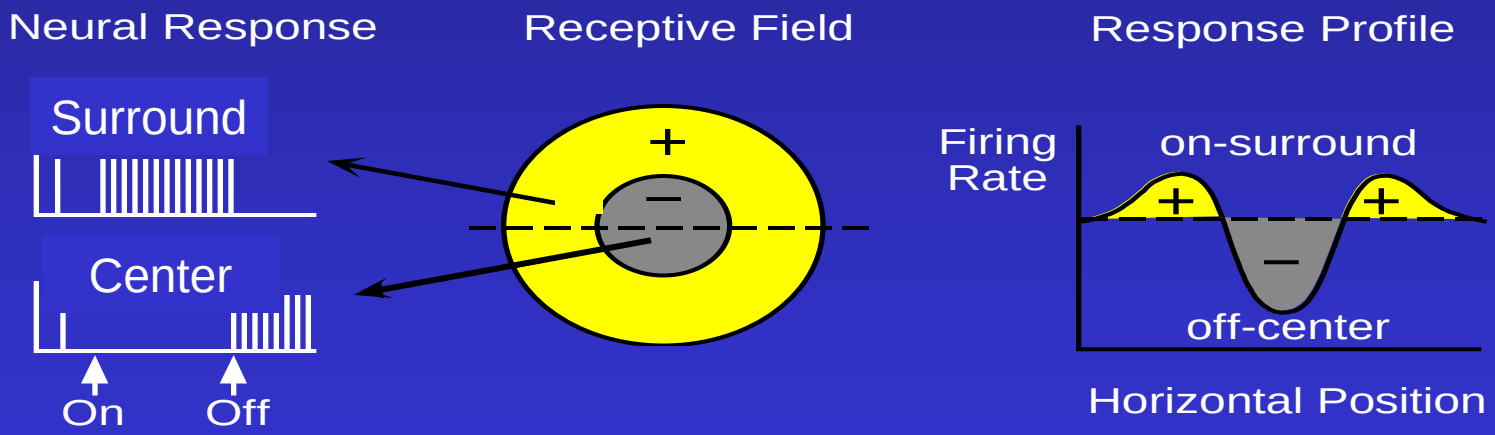
Retinal Receptive Fields

RF of On-center Off-surround cells

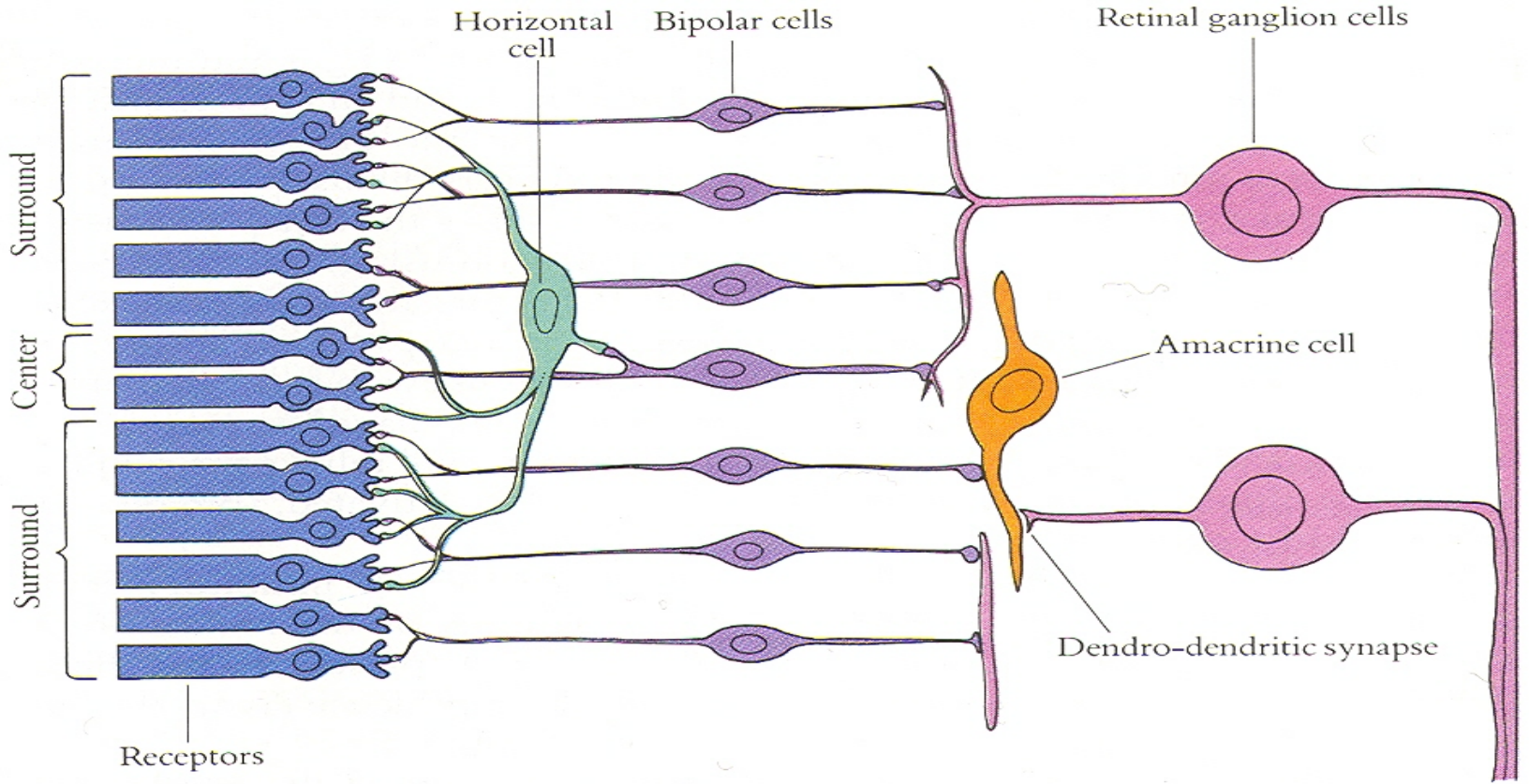


Retinal Receptive Fields

RF of Off-center On-surround cells

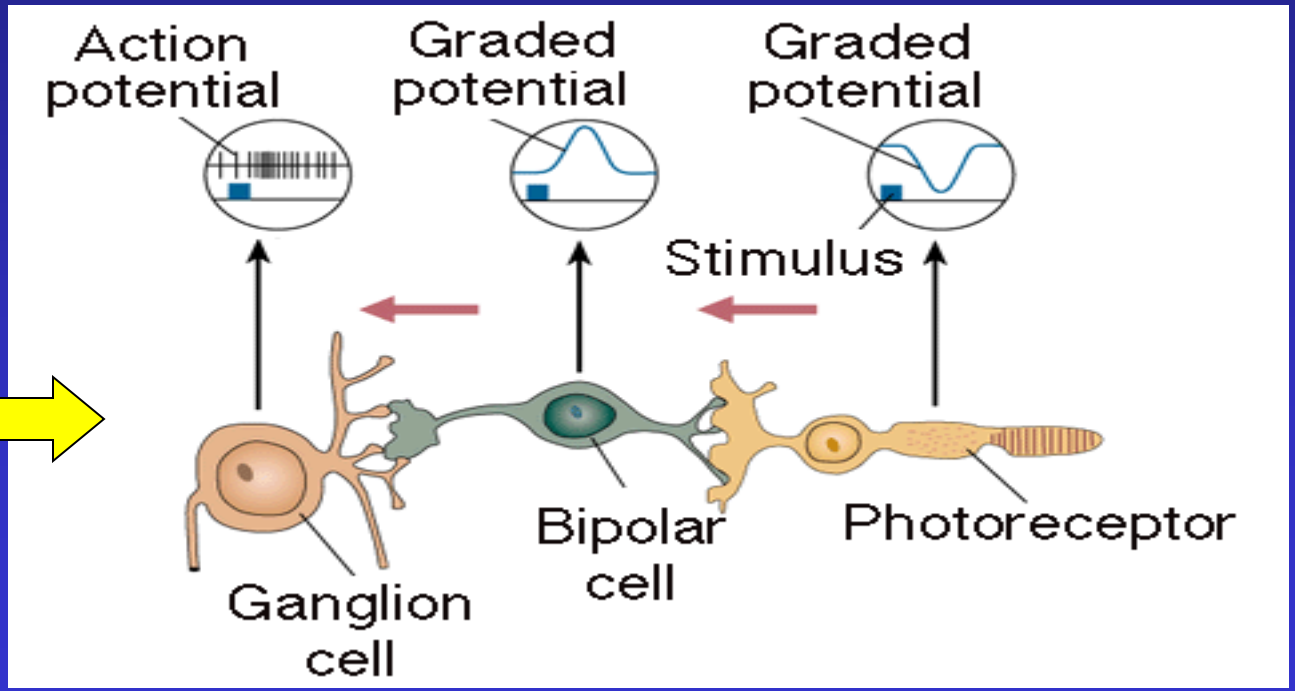


Retinal Receptive Fields



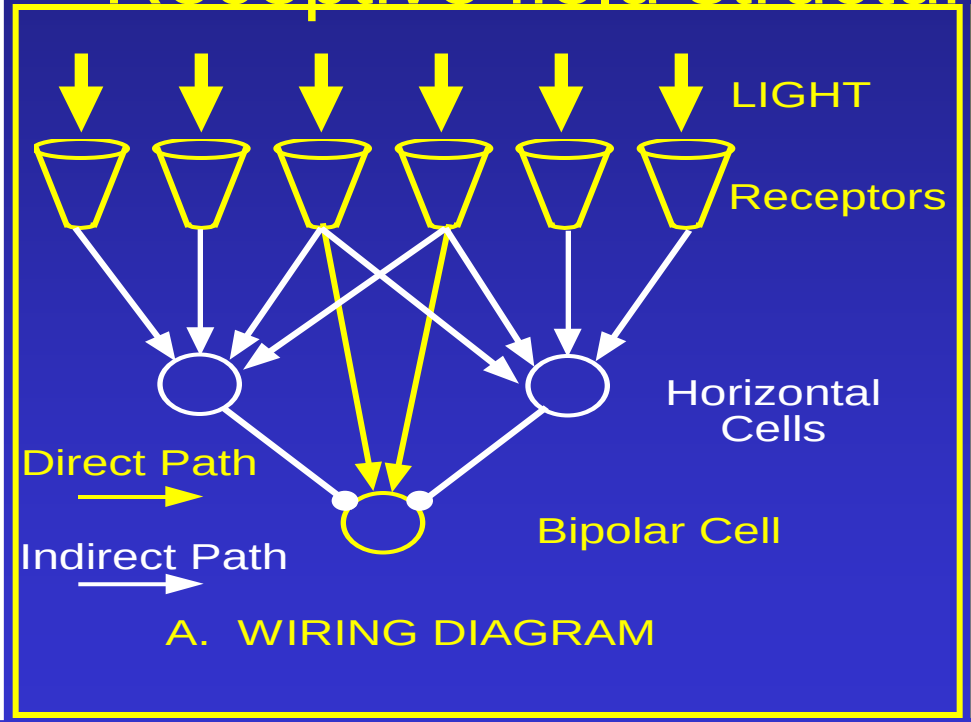
Retinal Receptive Fields

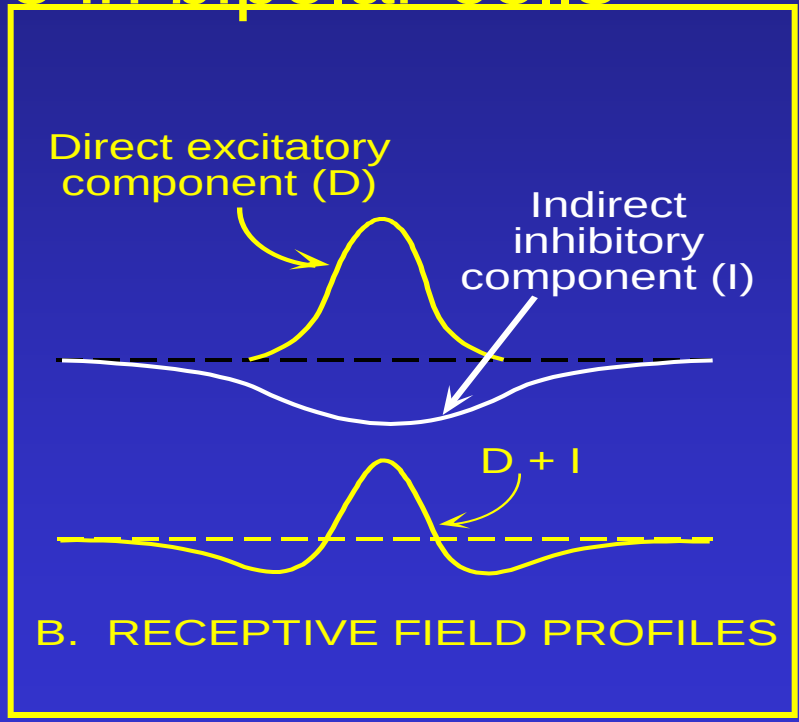
Receptive field structure in bipolar cells



Retinal Receptive Fields

Receptive field structure in bipolar cells

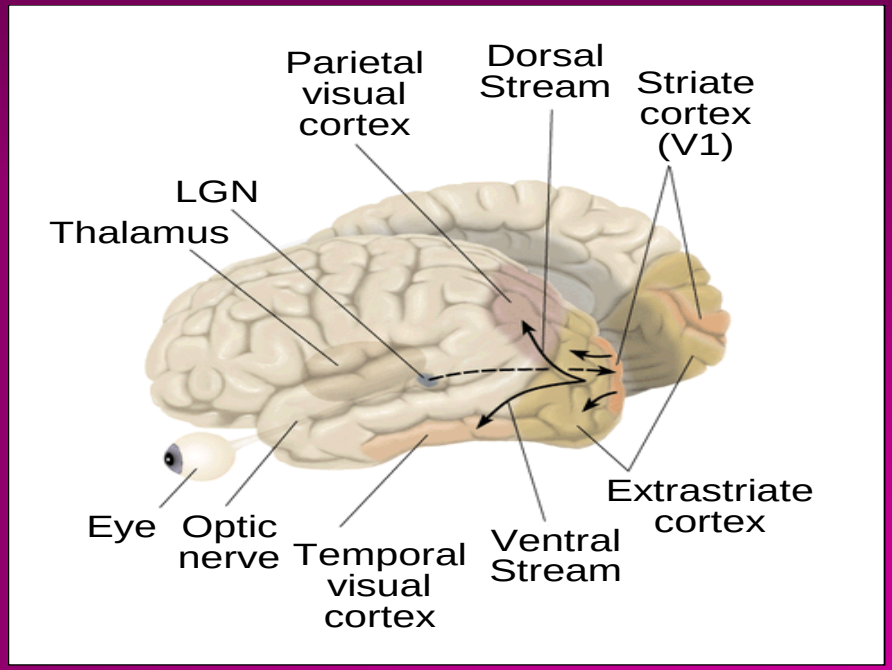




Visual Cortex

Cortical Area V1

aka:
Primary visual cortex
Striate cortex
Brodman's area 17



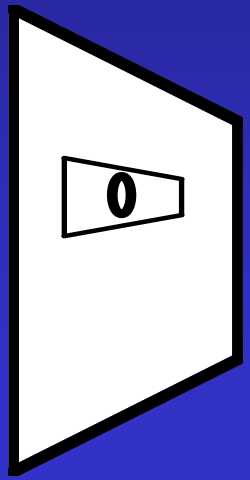
Cortical Receptive Fields

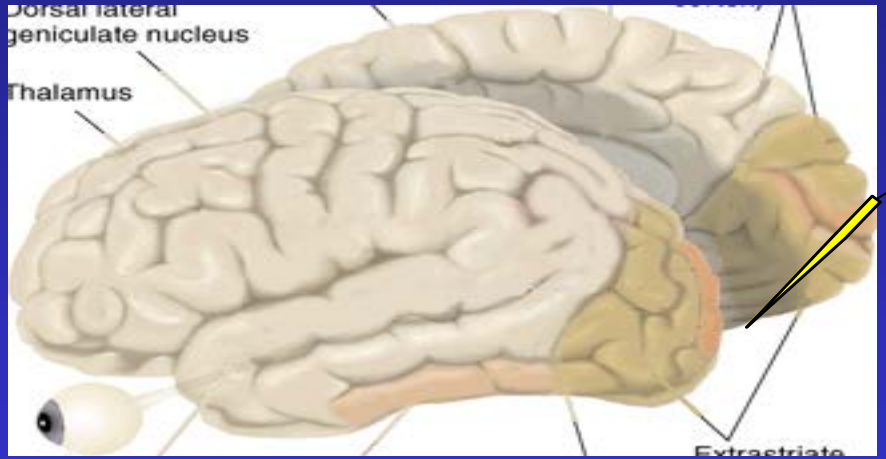
Single-cell recording from visual cortex

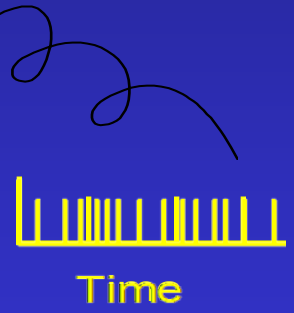
David Hubel & Thorston Wiesel

Cortical Receptive Fields

Single-cell recording from visual cortex







https://www.youtube.com/watch?v=IOHayh06LJ4

Cortical Receptive Fields

Three classes of cells in V1

Simple cells

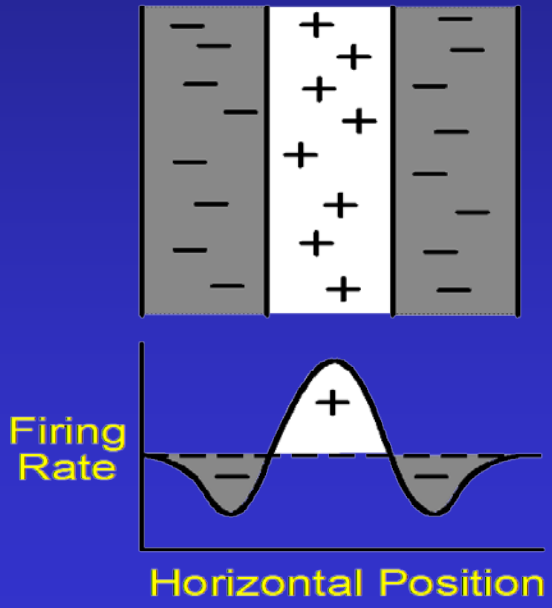
Complex cells

Hypercomplex cells

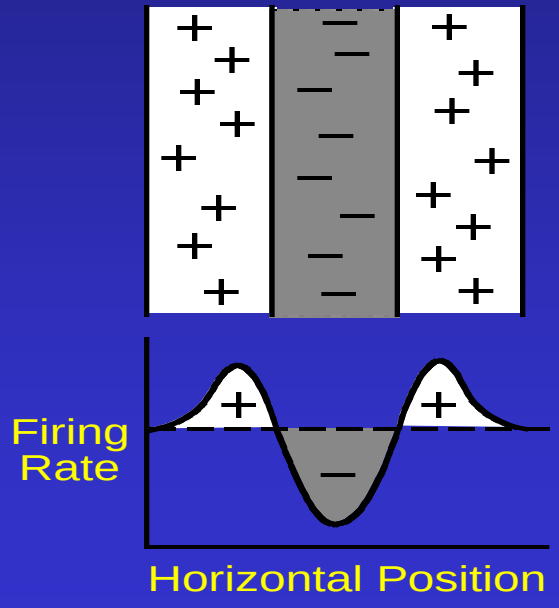
Cortical Receptive Fields

Simple Cells: “Line Detectors”

A. Light Line Detector



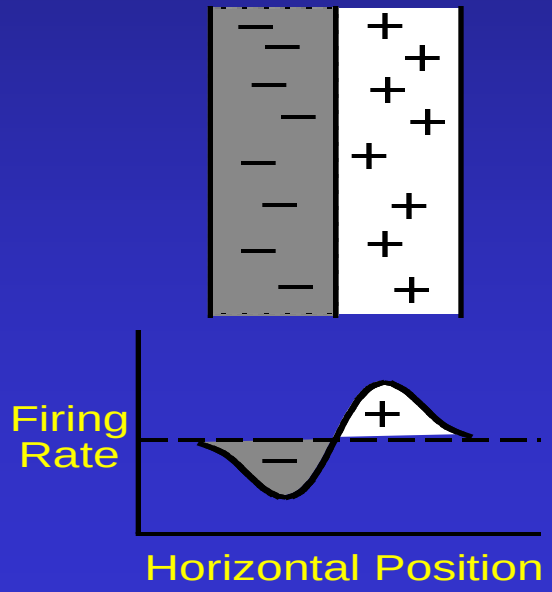
B. Dark Line Detector



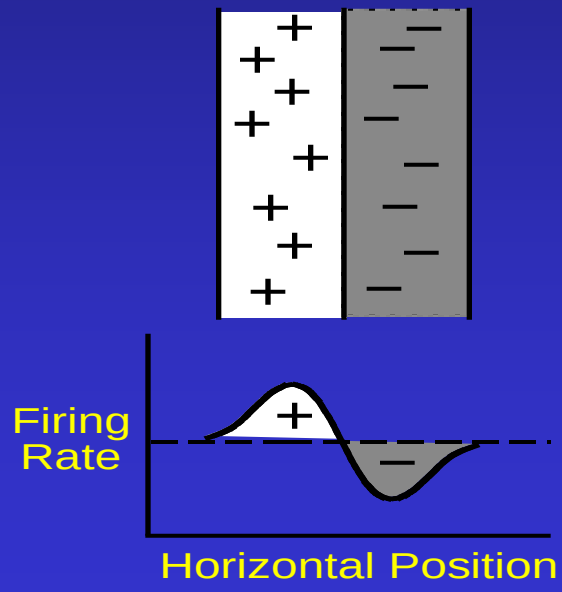
Cortical Receptive Fields

Simple Cells: “Edge Detectors”

C. Dark-to-light Edge Detector

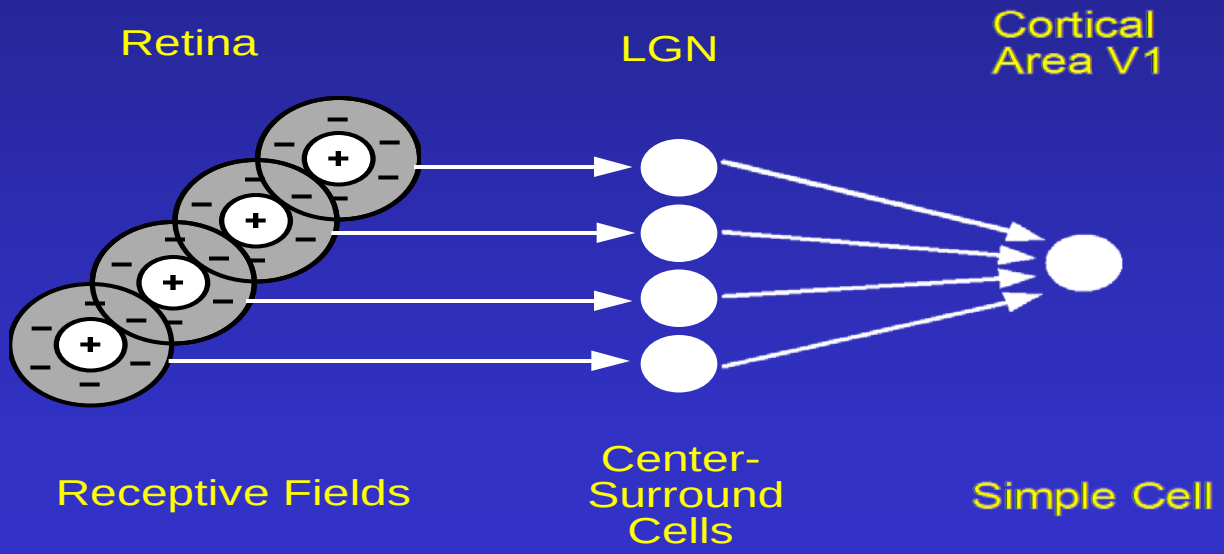


D. Light-to-dark Edge Detector



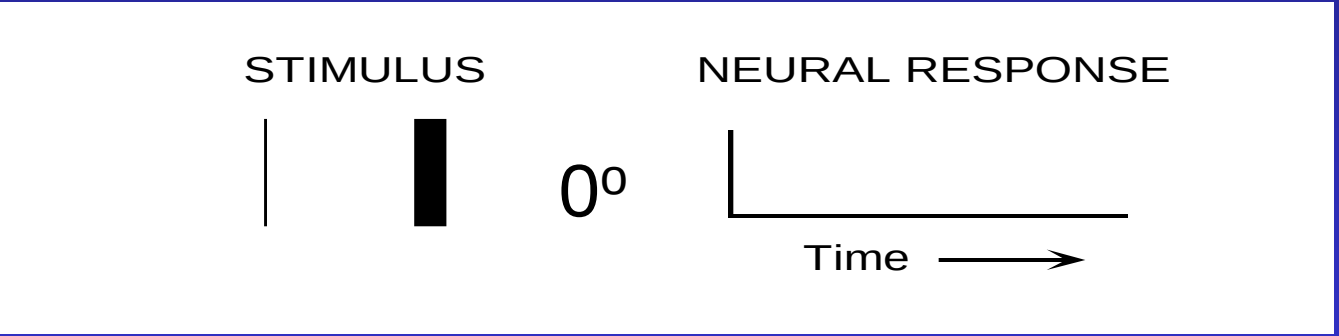
Cortical Receptive Fields

Constructing a line detector



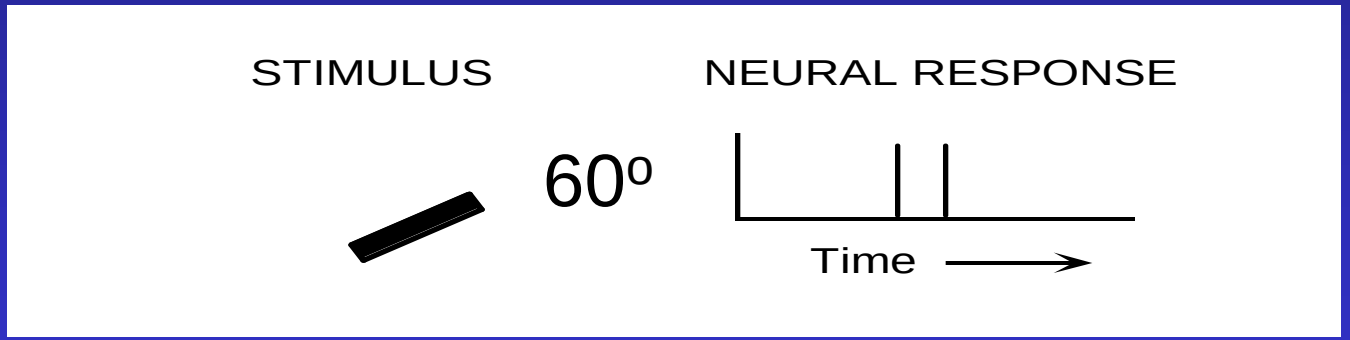
Cortical Receptive Fields

Complex Cells



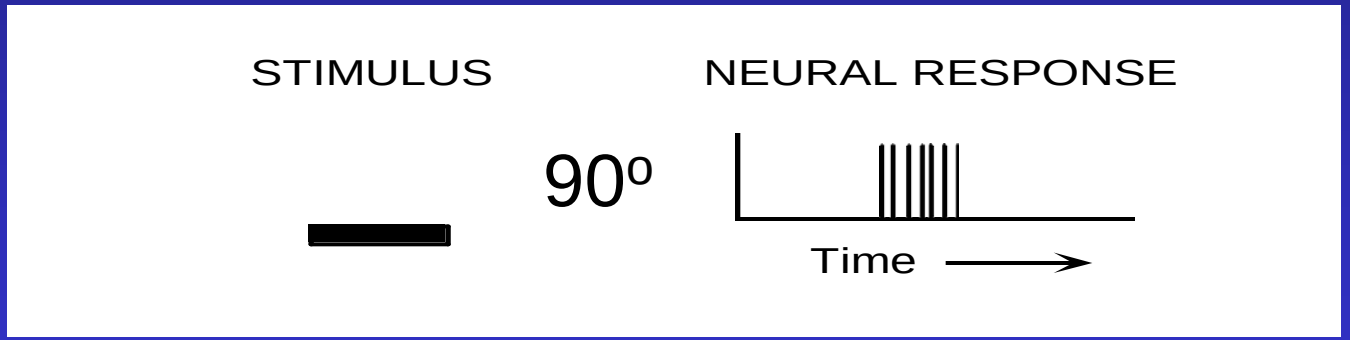
Cortical Receptive Fields

Complex Cells



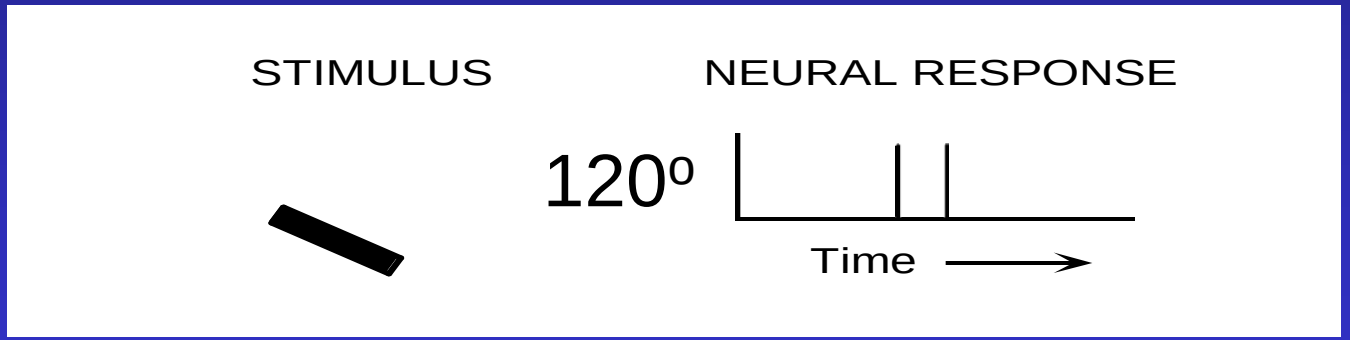
Cortical Receptive Fields

Complex Cells



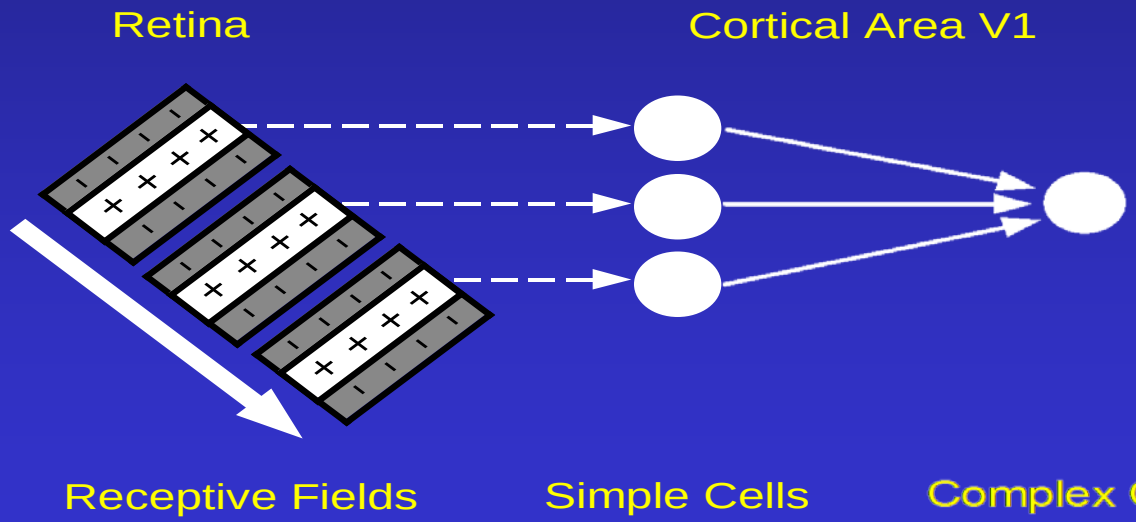
Cortical Receptive Fields

Complex Cells



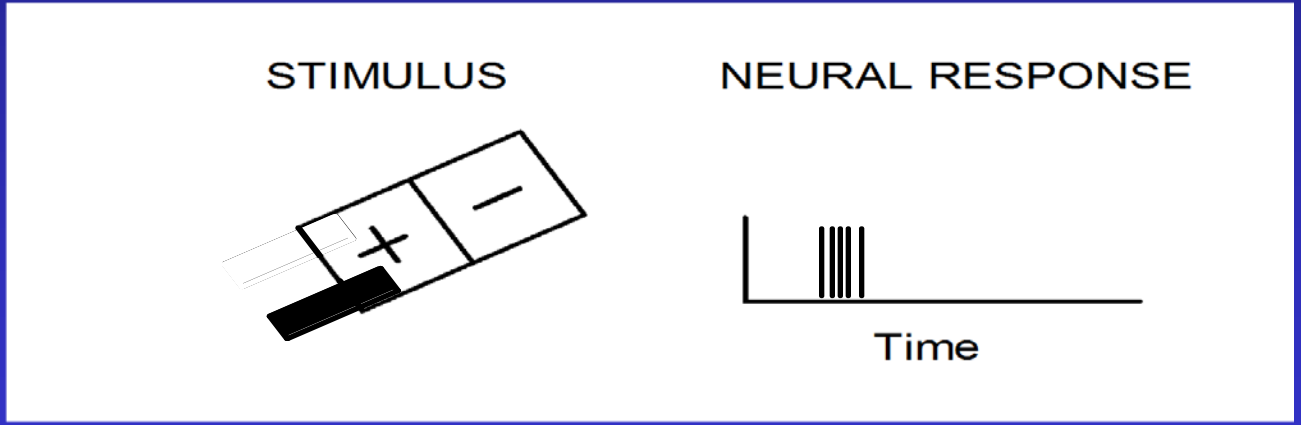
Cortical Receptive Fields

Constructing a Complex Cell



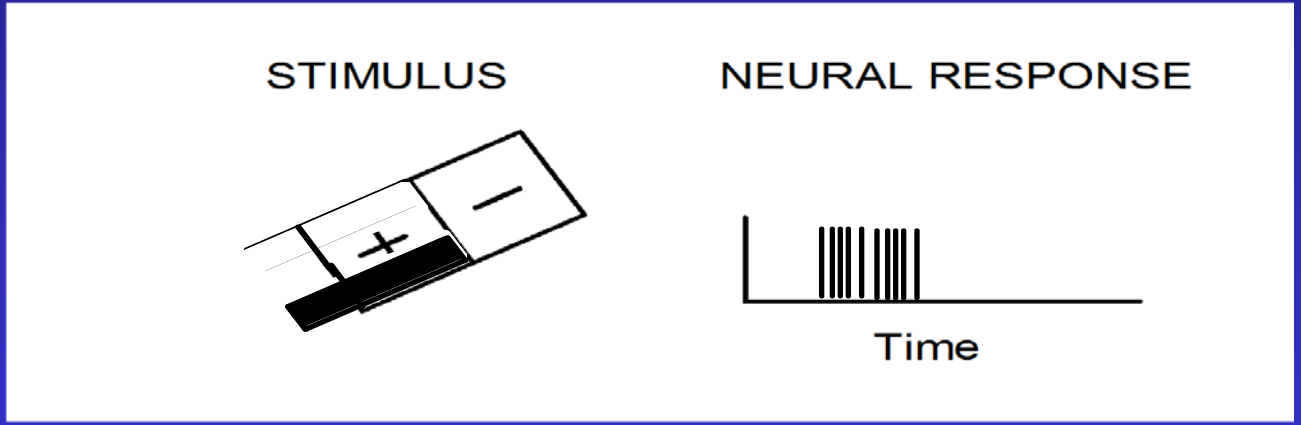
Cortical Receptive Fields

Hypercomplex Cells



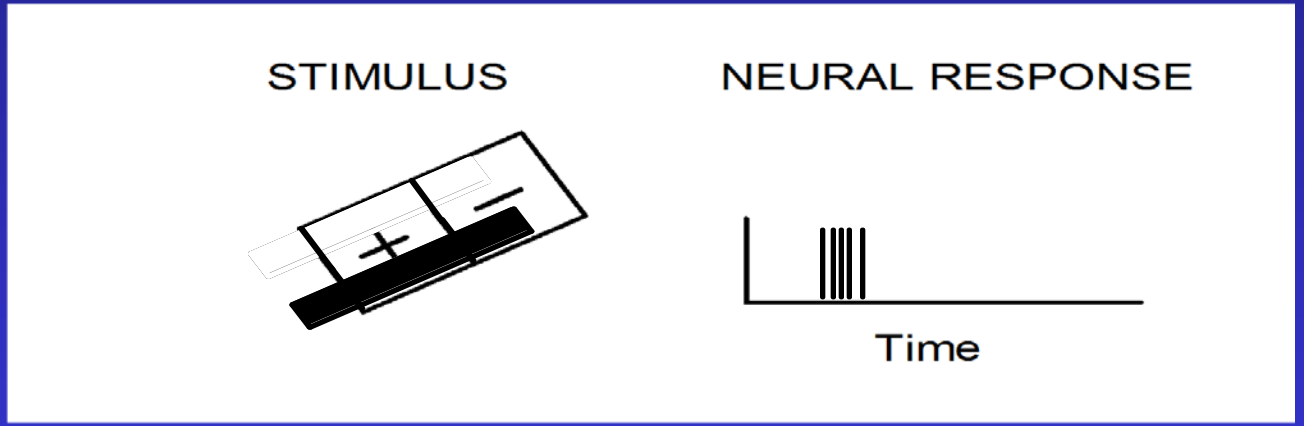
Cortical Receptive Fields

Hypercomplex Cells



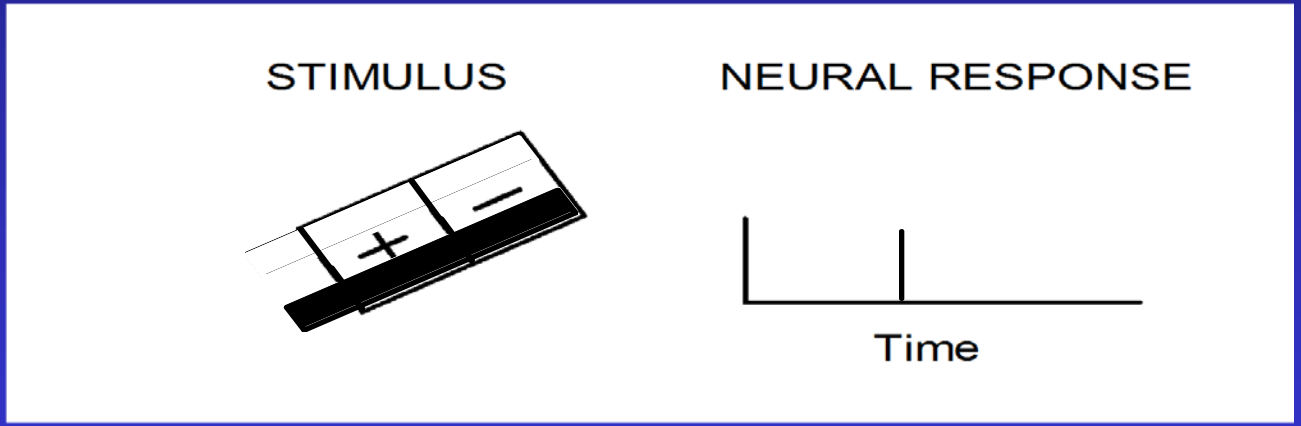
Cortical Receptive Fields

Hypercomplex Cells



Cortical Receptive Fields

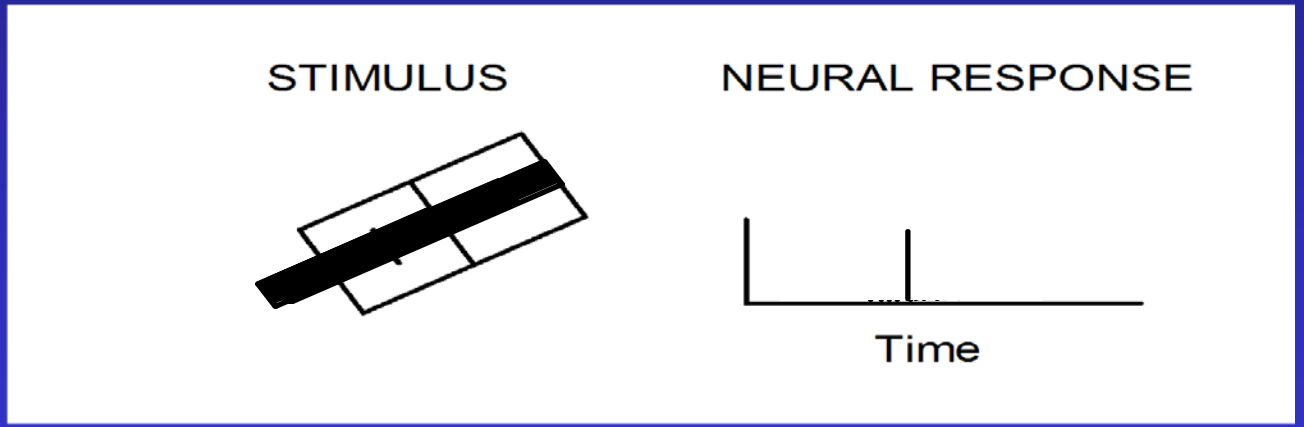
Hypercomplex Cells



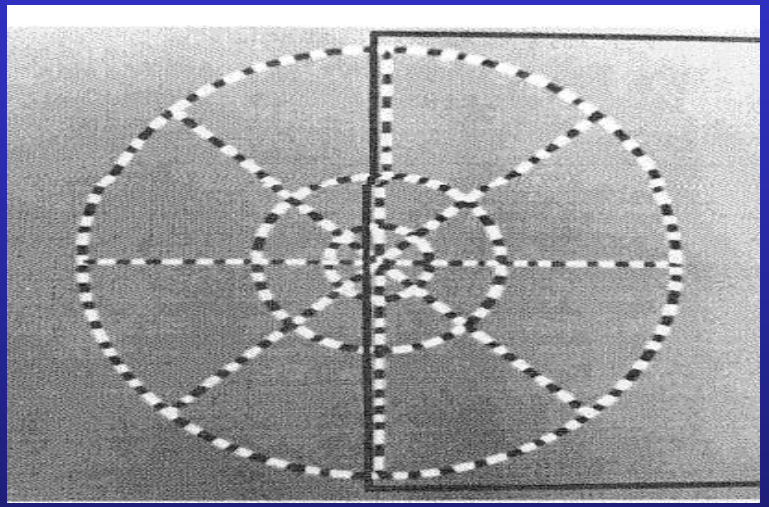
“End-stopped” Cells

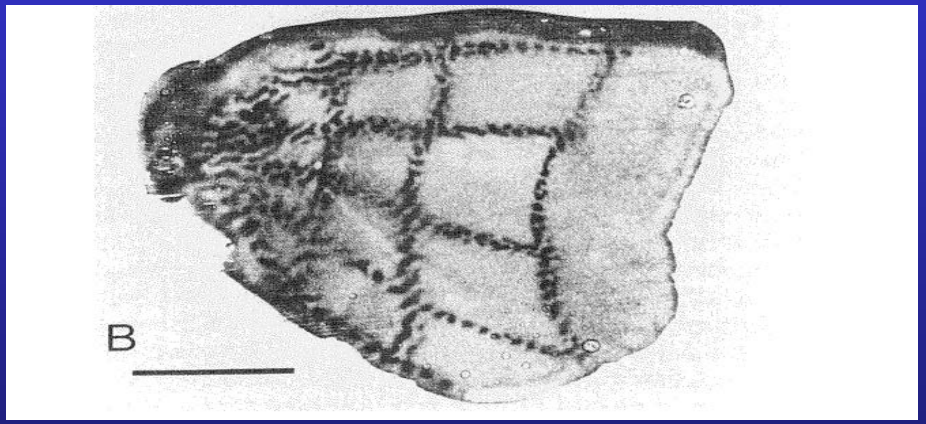
Cortical Receptive Fields

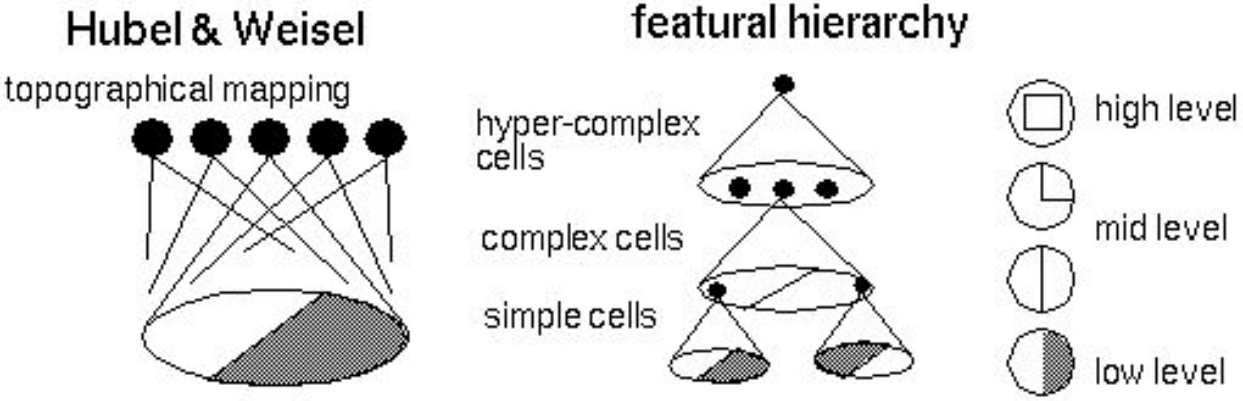
“End-stopped” Simple Cells



Mapping from Retina to V1

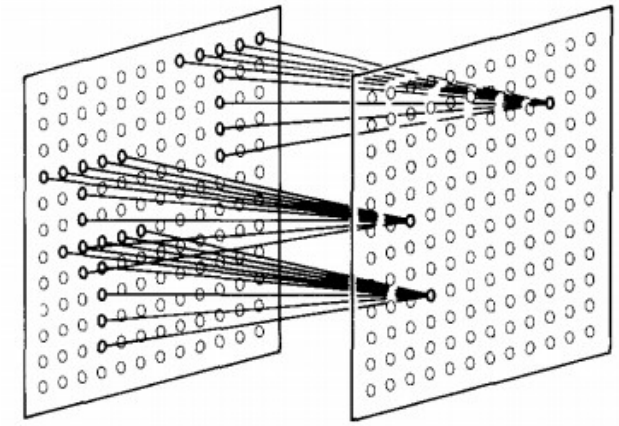




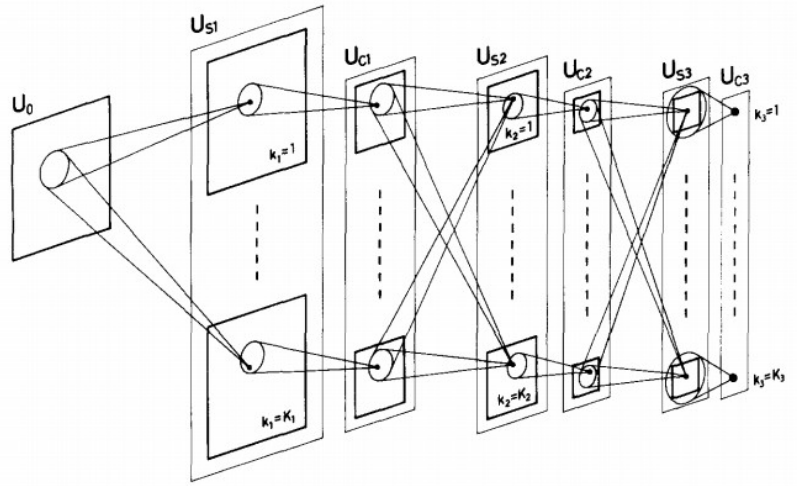

Hubel & Weisel
topographical mapping
featural hierarchy
hyper-complex cells
complex cells
simple cells
high level
mid level
low level

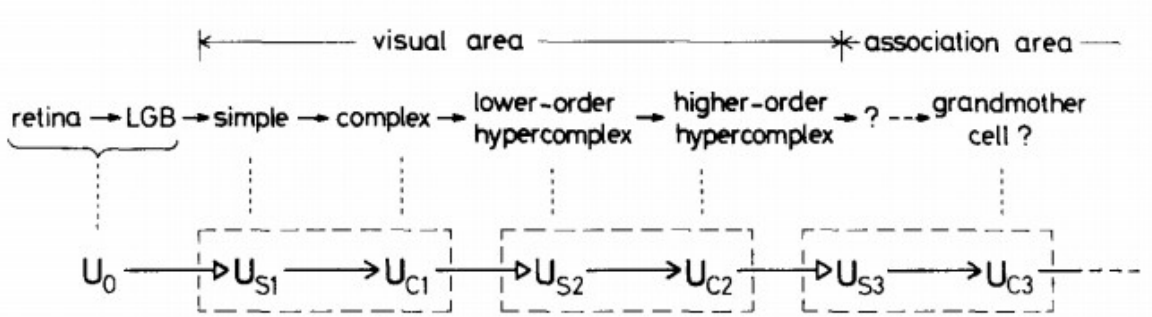
A bit of history:

Neurocognitron
[Fukushima 1980]



"sandwich" architecture (SCSCSC…)
simple cells: modifiable parameters
complex cells: perform pooling

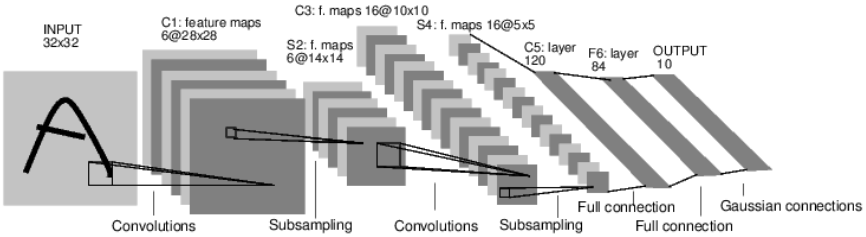




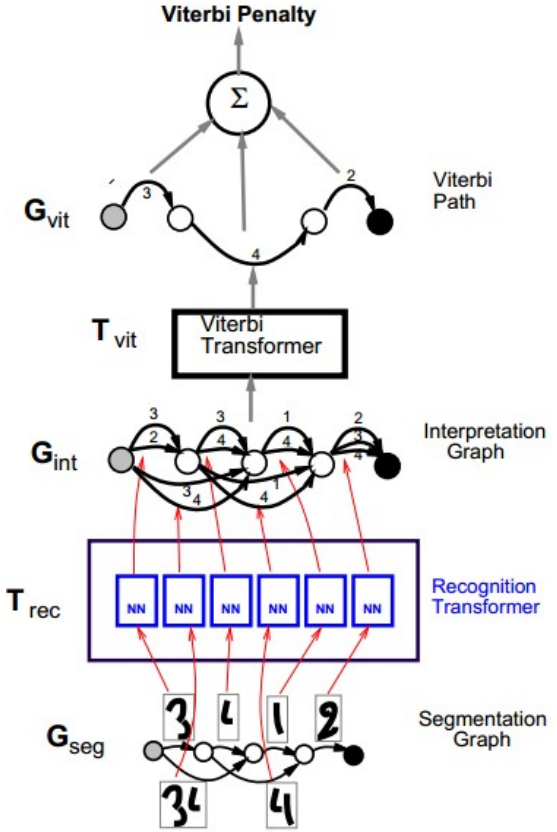
A bit of history:

Gradient-based learning applied to document recognition

[LeCun, Bottou, Bengio, Haffner 1998]

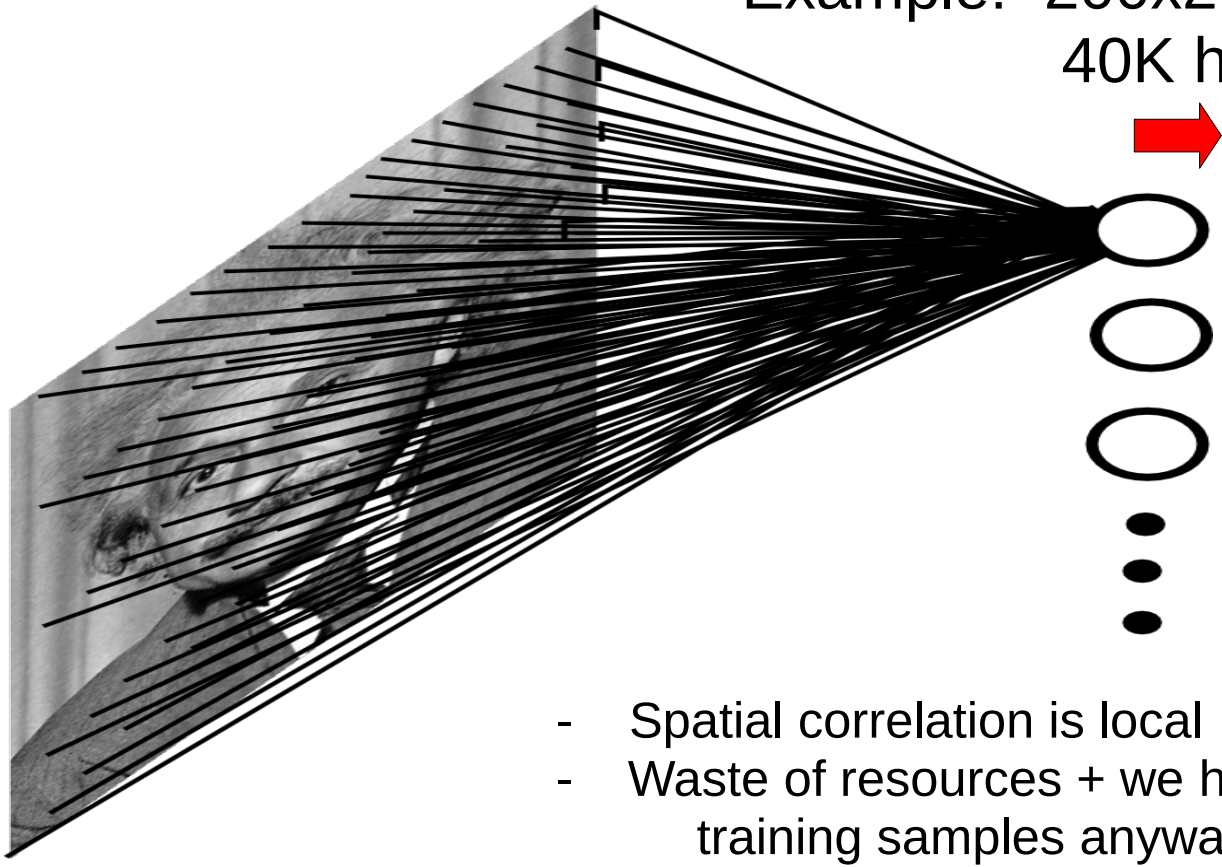


LeNet-5



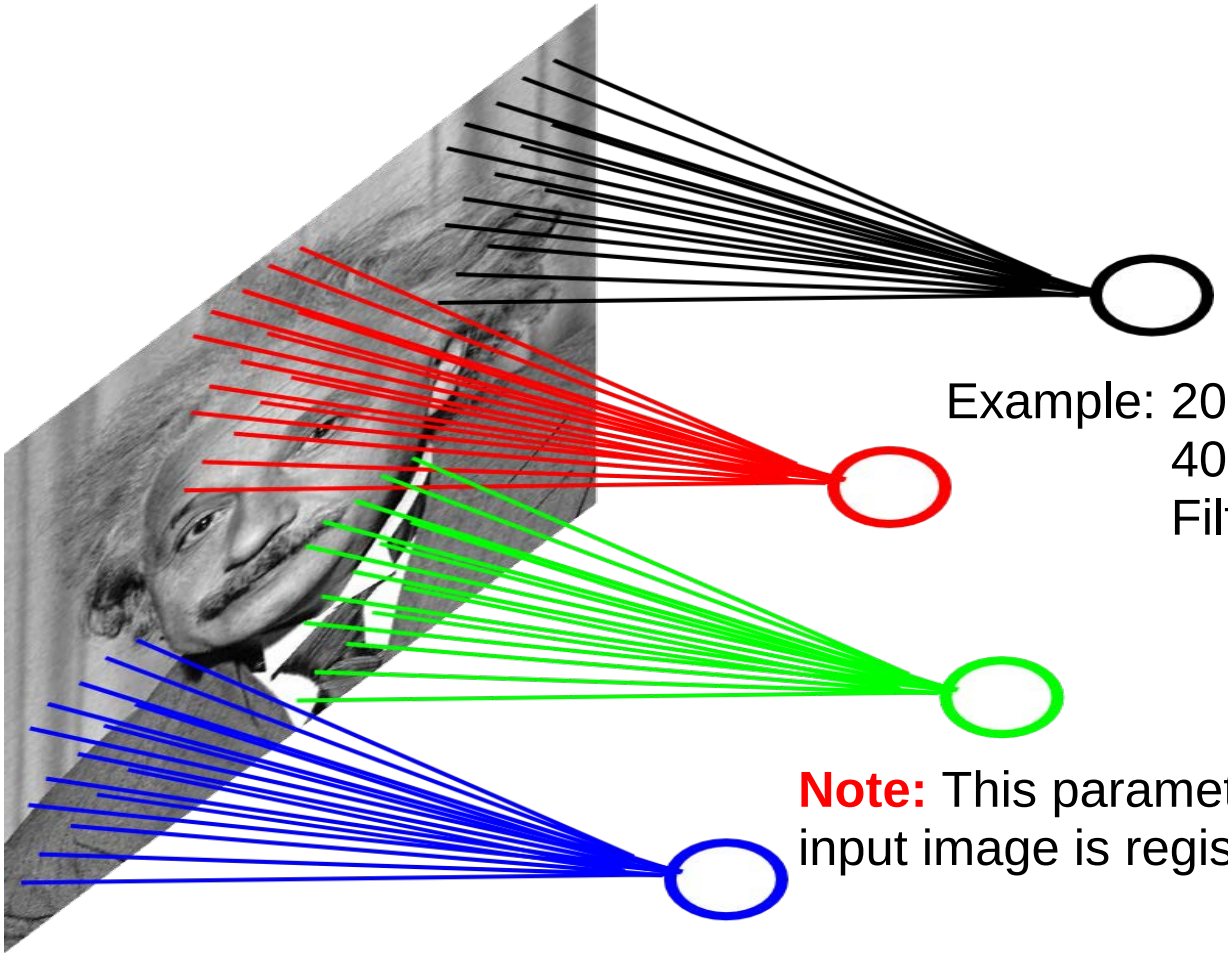
Fully Connected Layer

Example: 200x200 image
40K hidden units
~2B parameters!!!



- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..

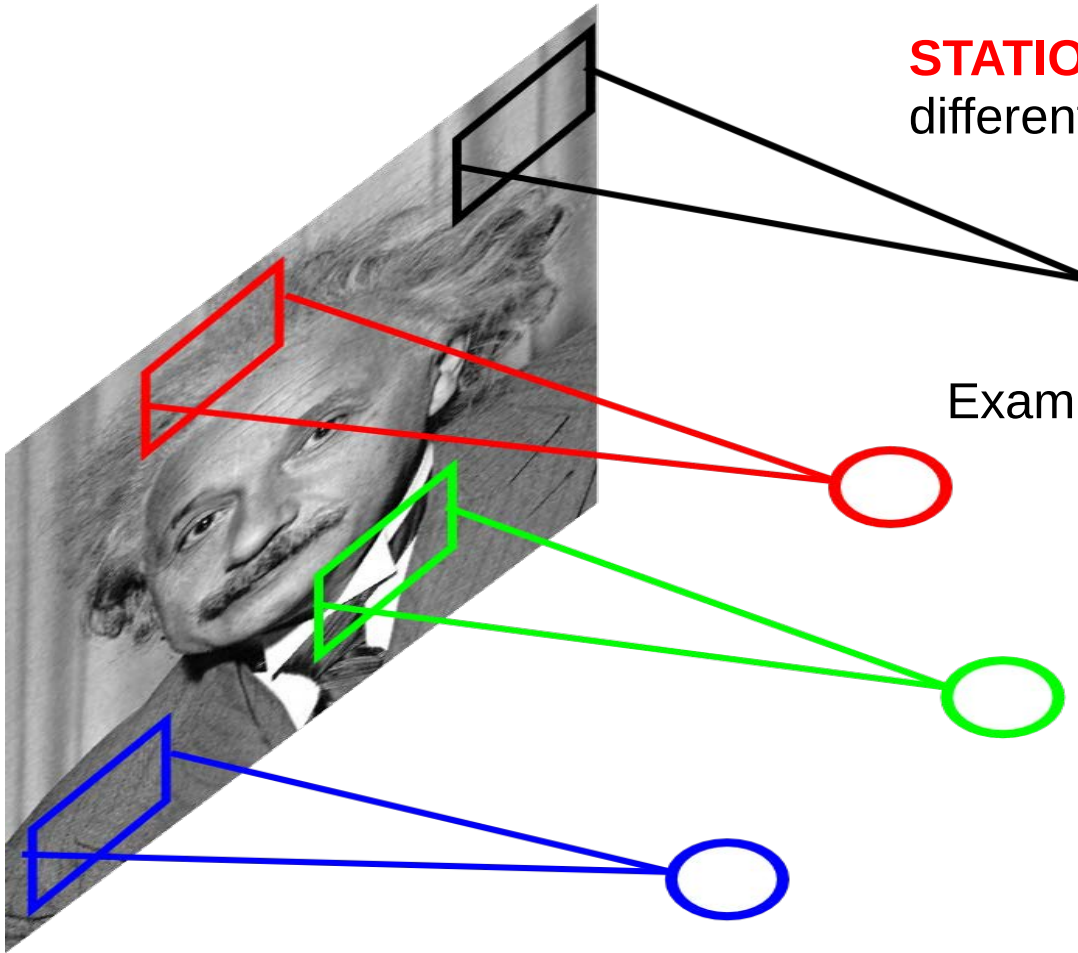
Locally Connected Layer



Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).

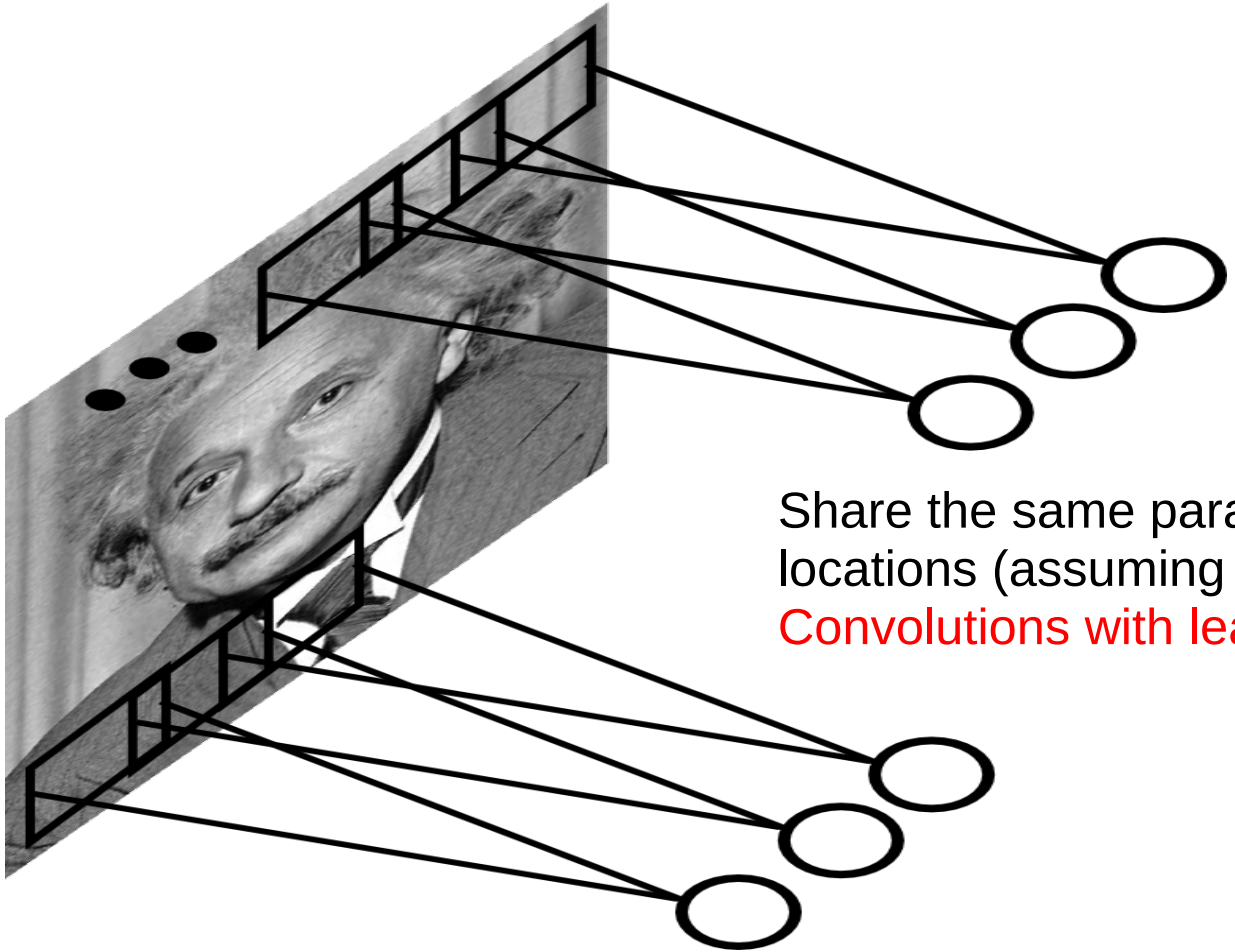
Locally Connected Layer



STATIONARITY? Statistics is similar at different locations

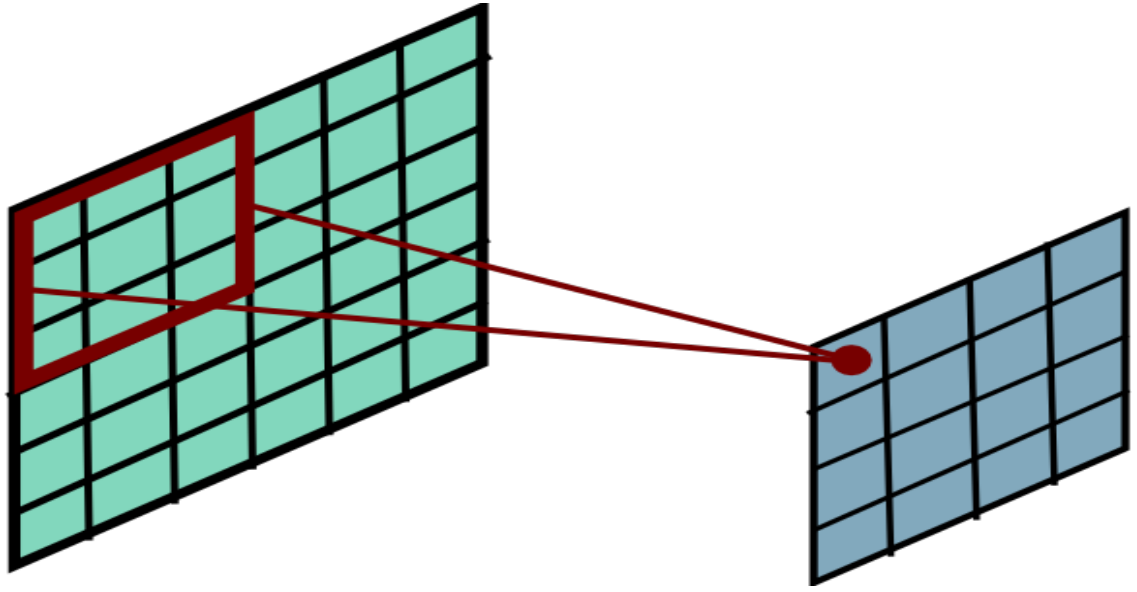
Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Convolutional Layer

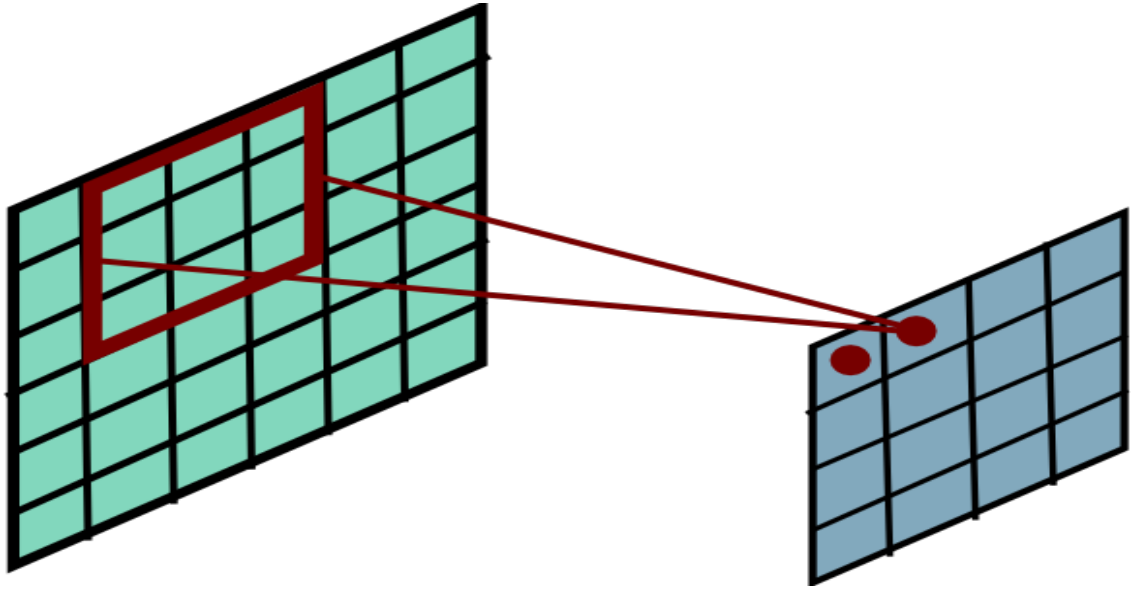


Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels

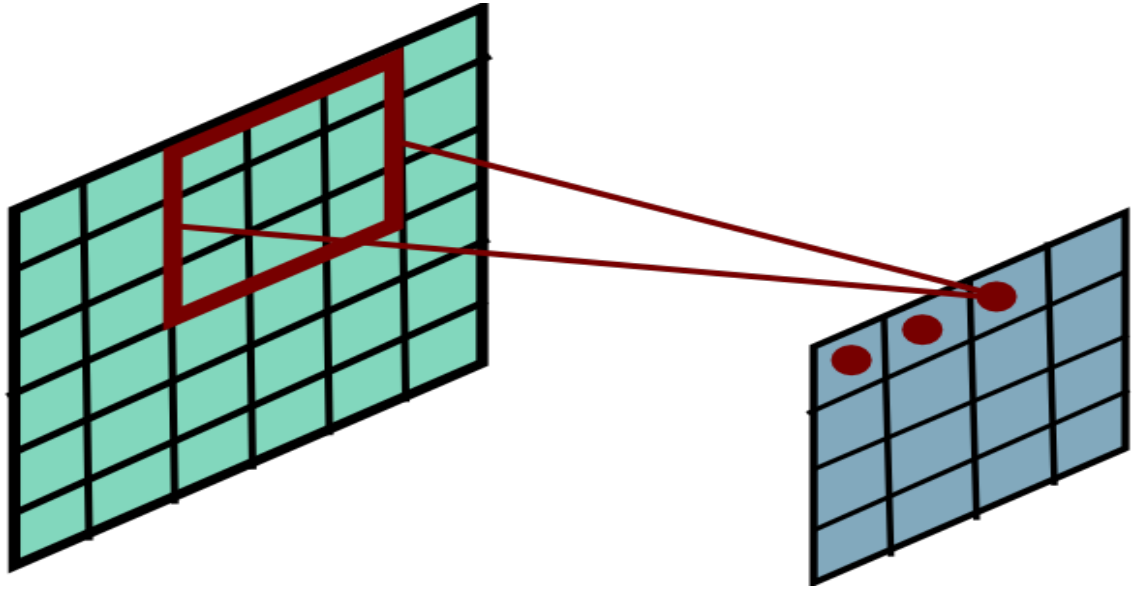
Convolutional Layer



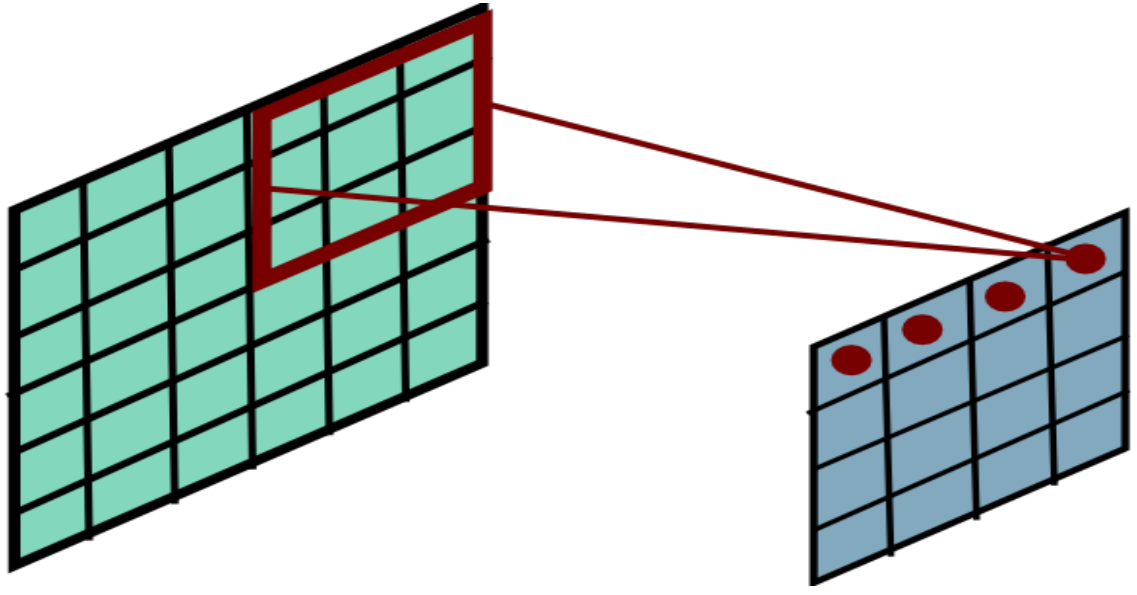
Convolutional Layer



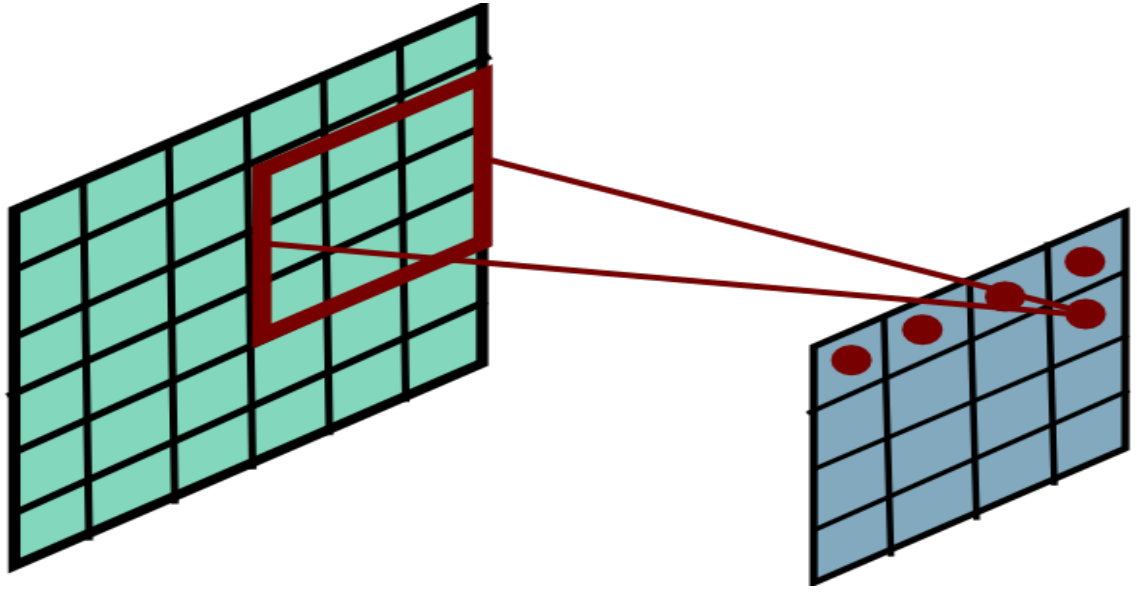
Convolutional Layer



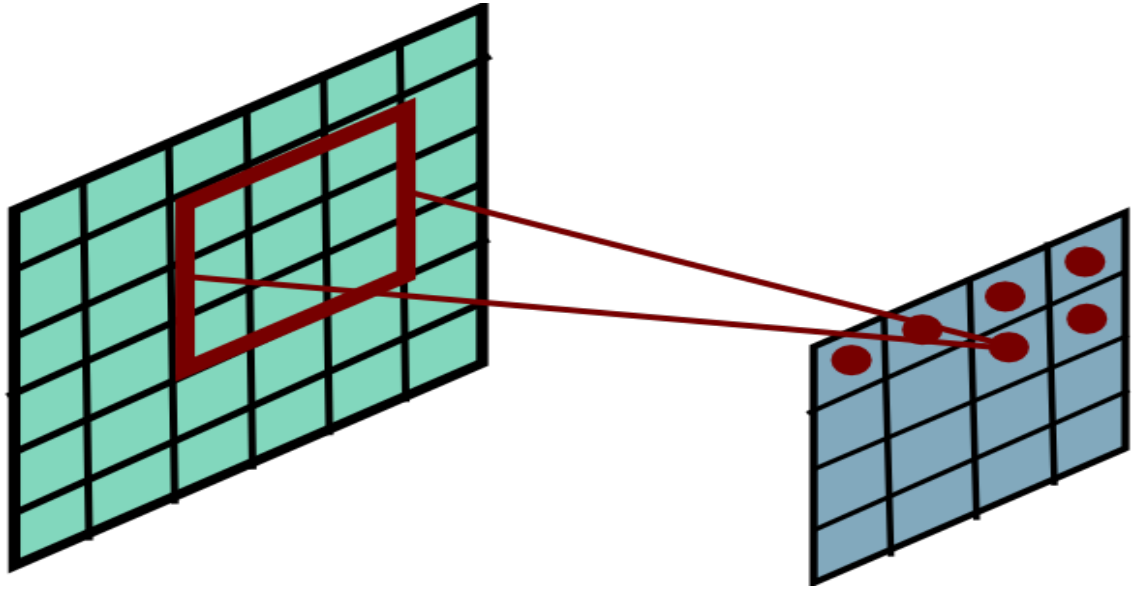
Convolutional Layer



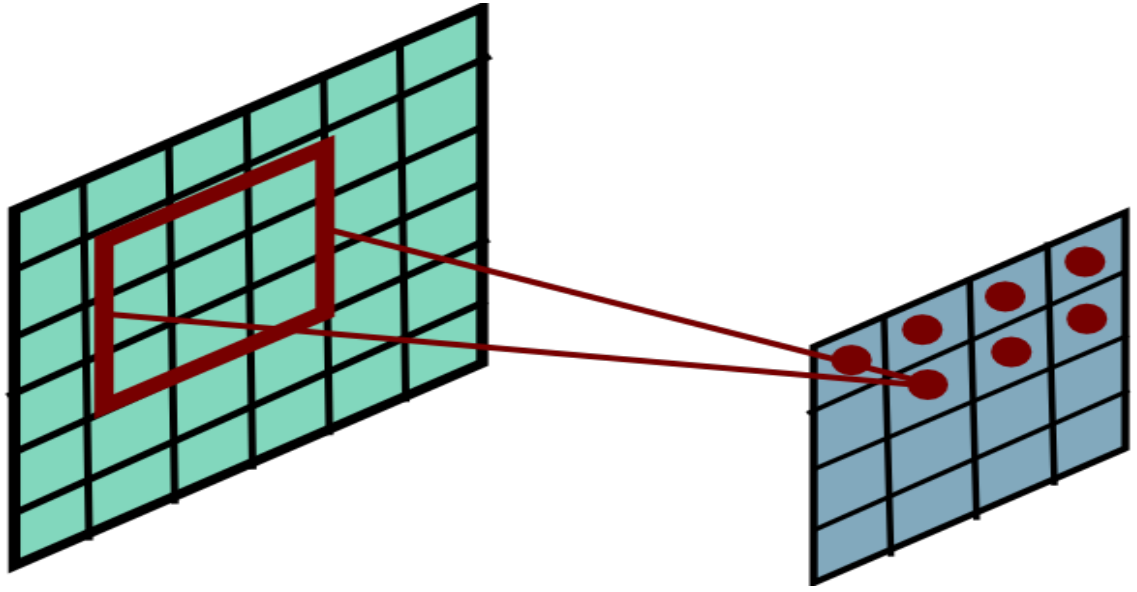
Convolutional Layer



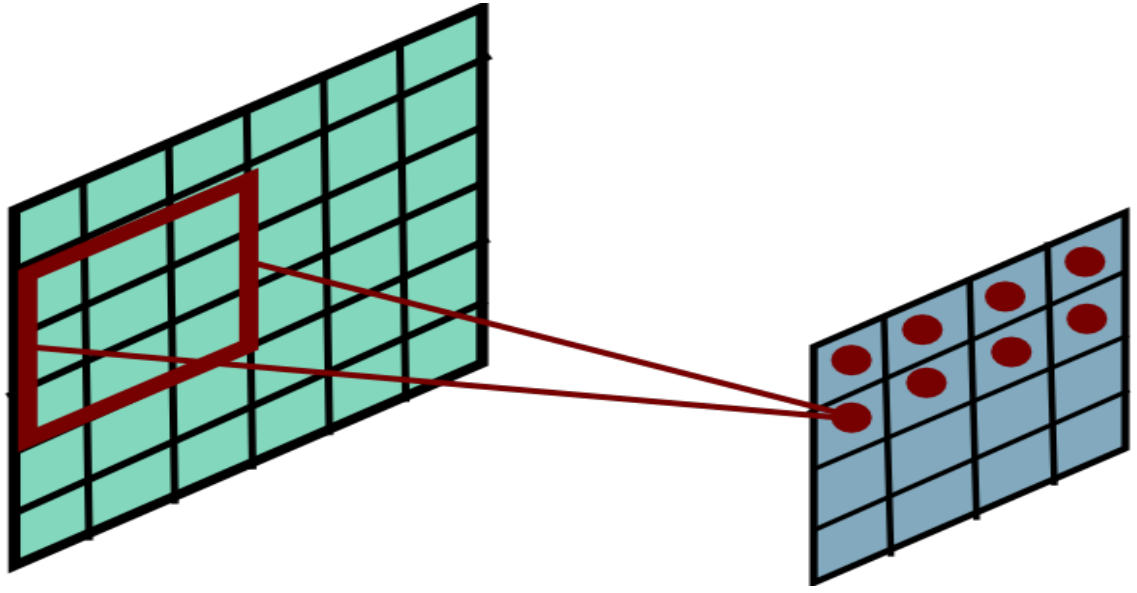
Convolutional Layer



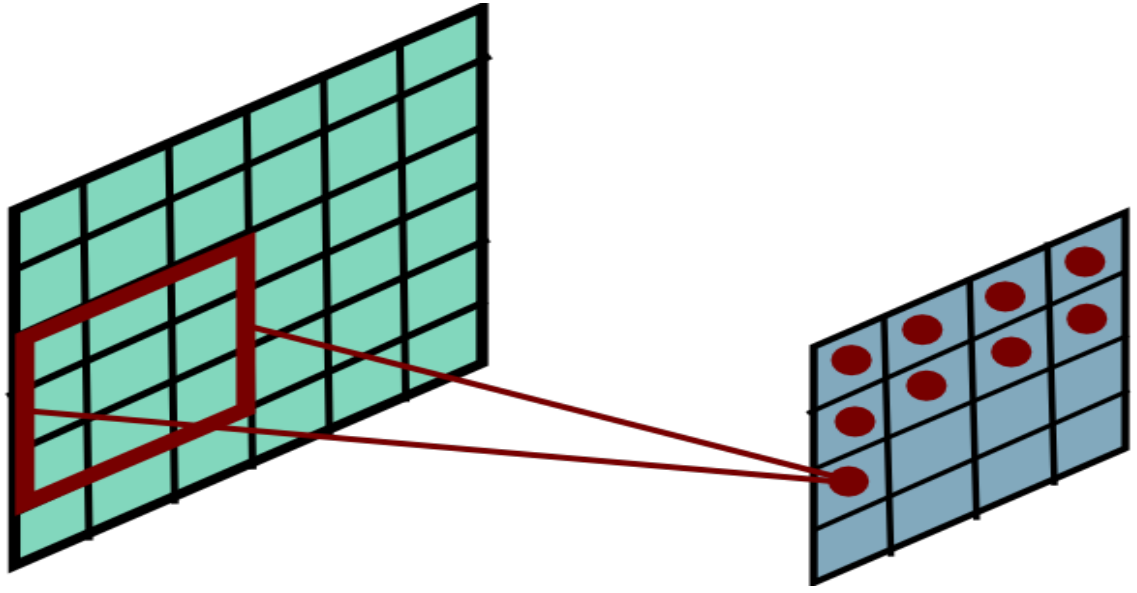
Convolutional Layer



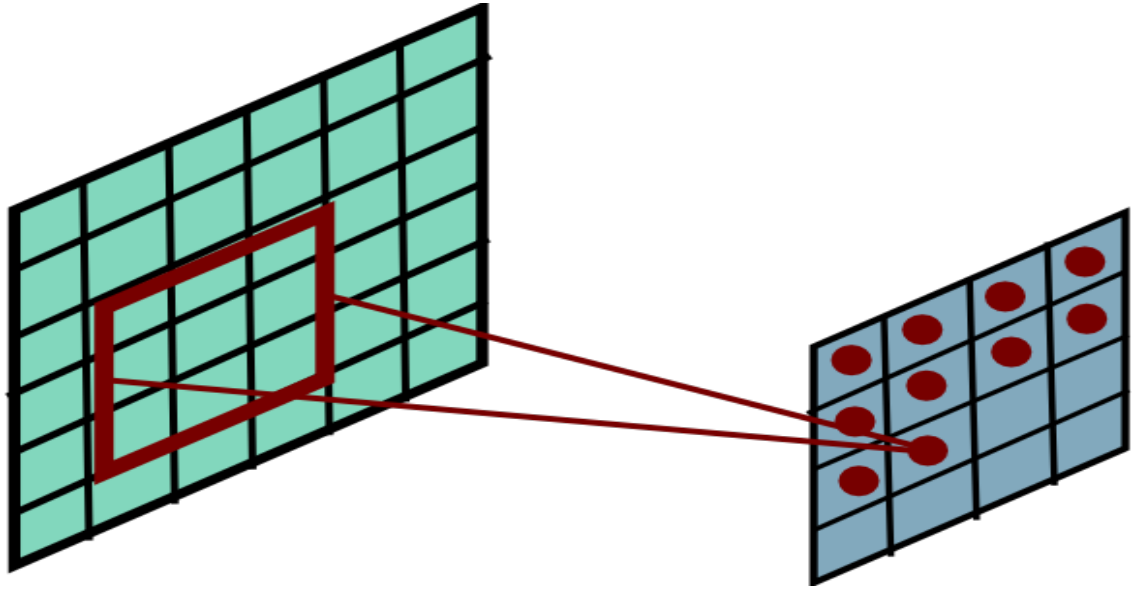
Convolutional Layer



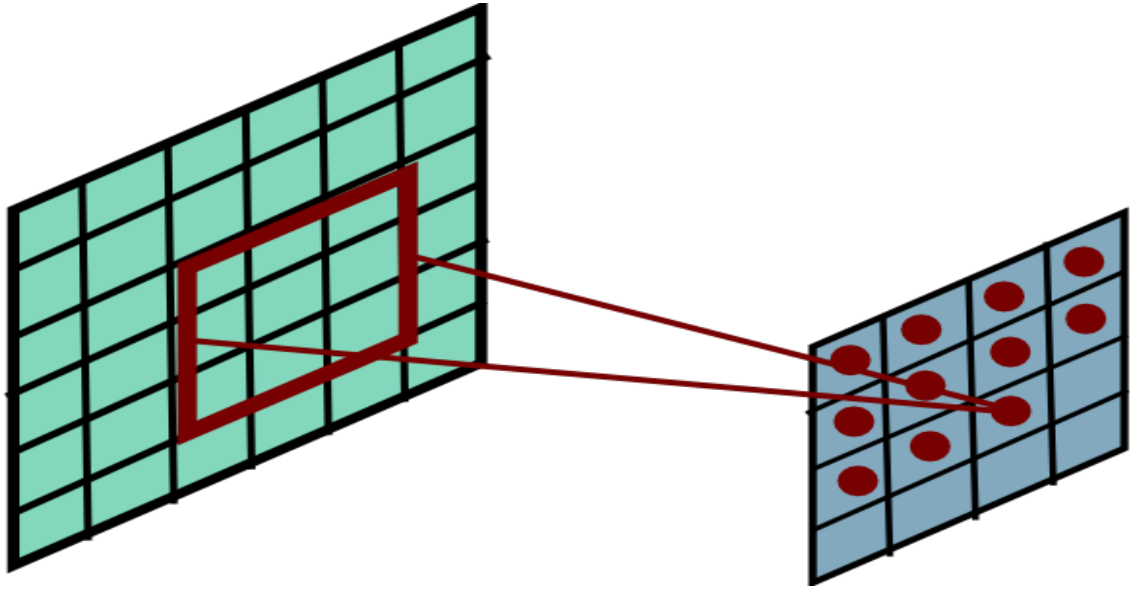
Convolutional Layer



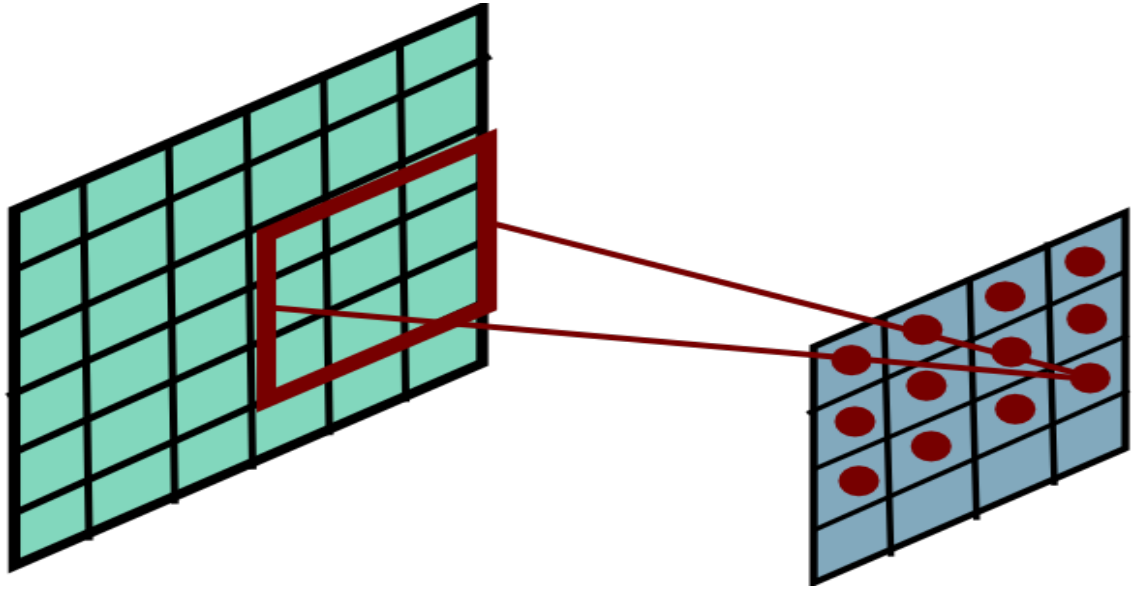
Convolutional Layer



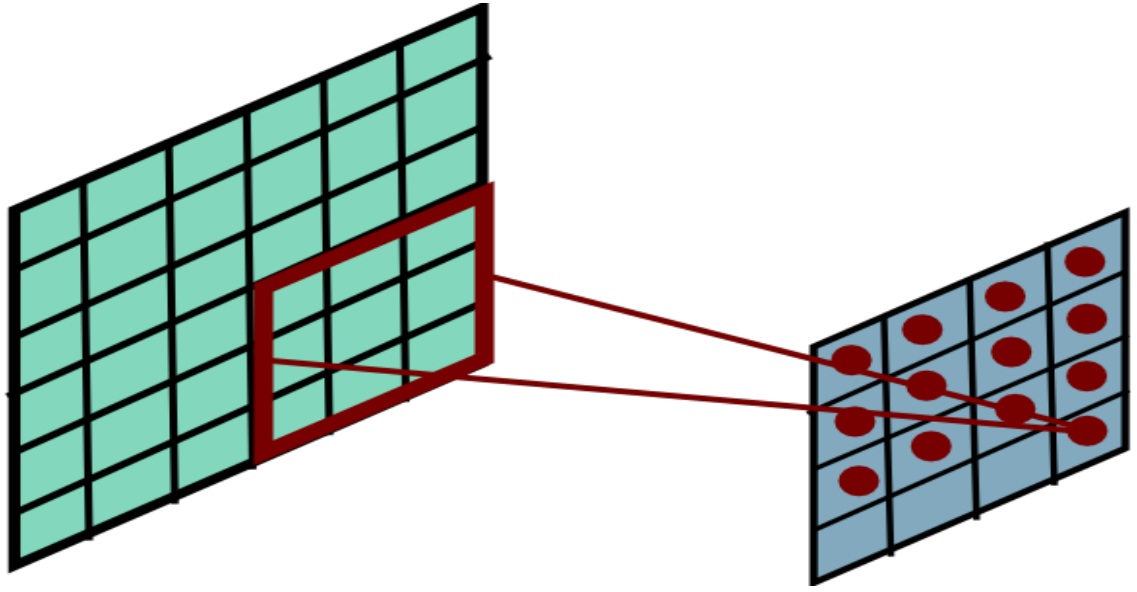
Convolutional Layer



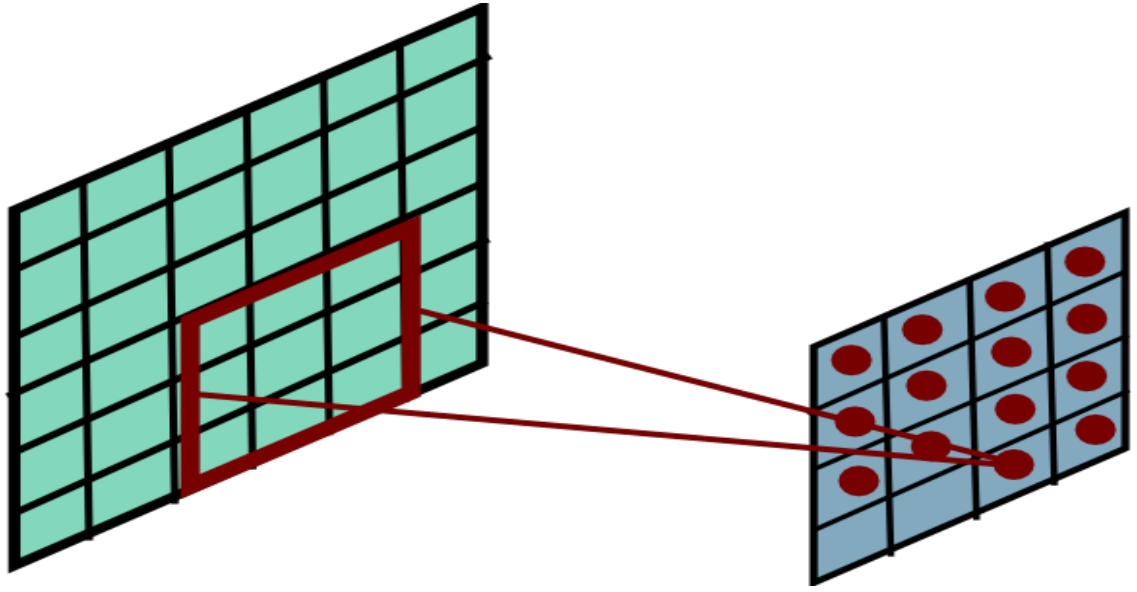
Convolutional Layer



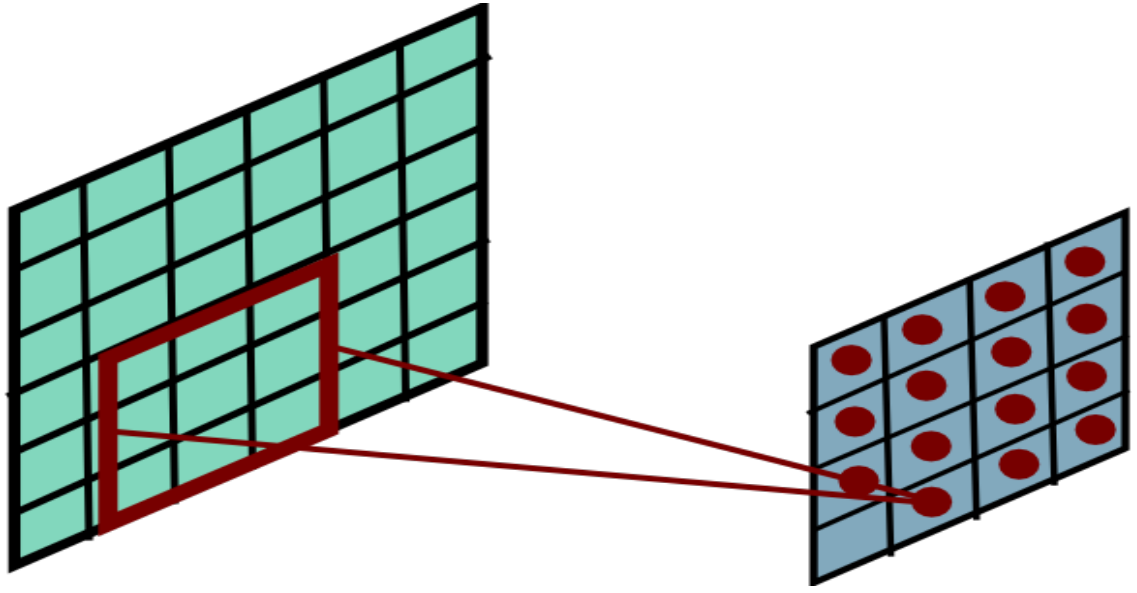
Convolutional Layer



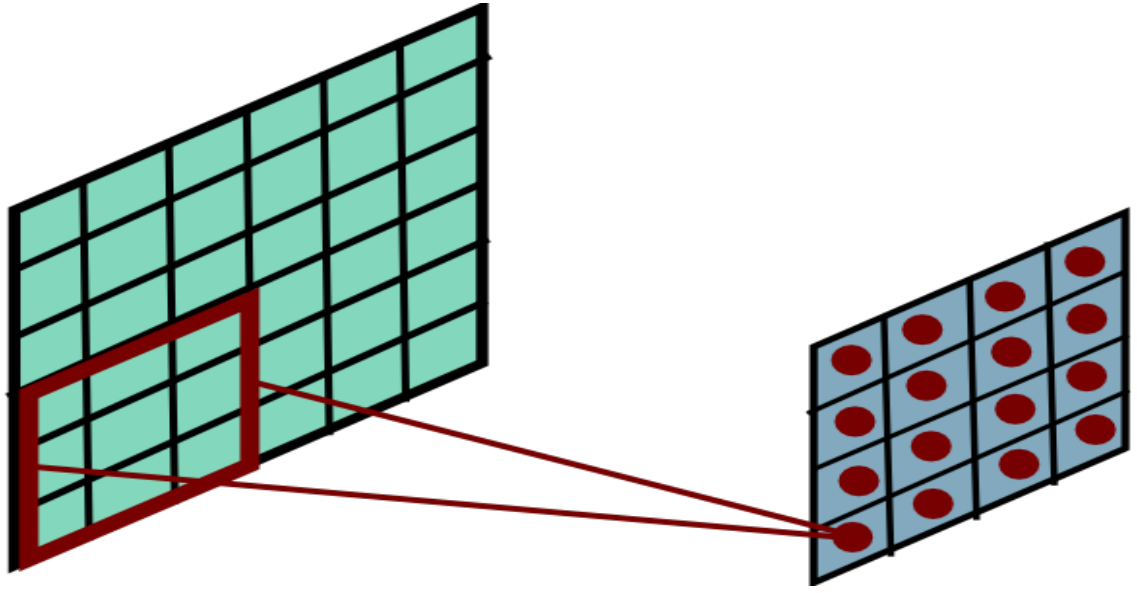
Convolutional Layer



Convolutional Layer

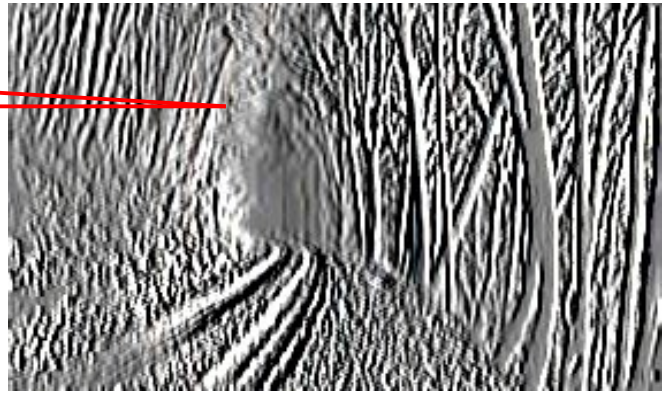


Convolutional Layer



Convolutional of Two Signals

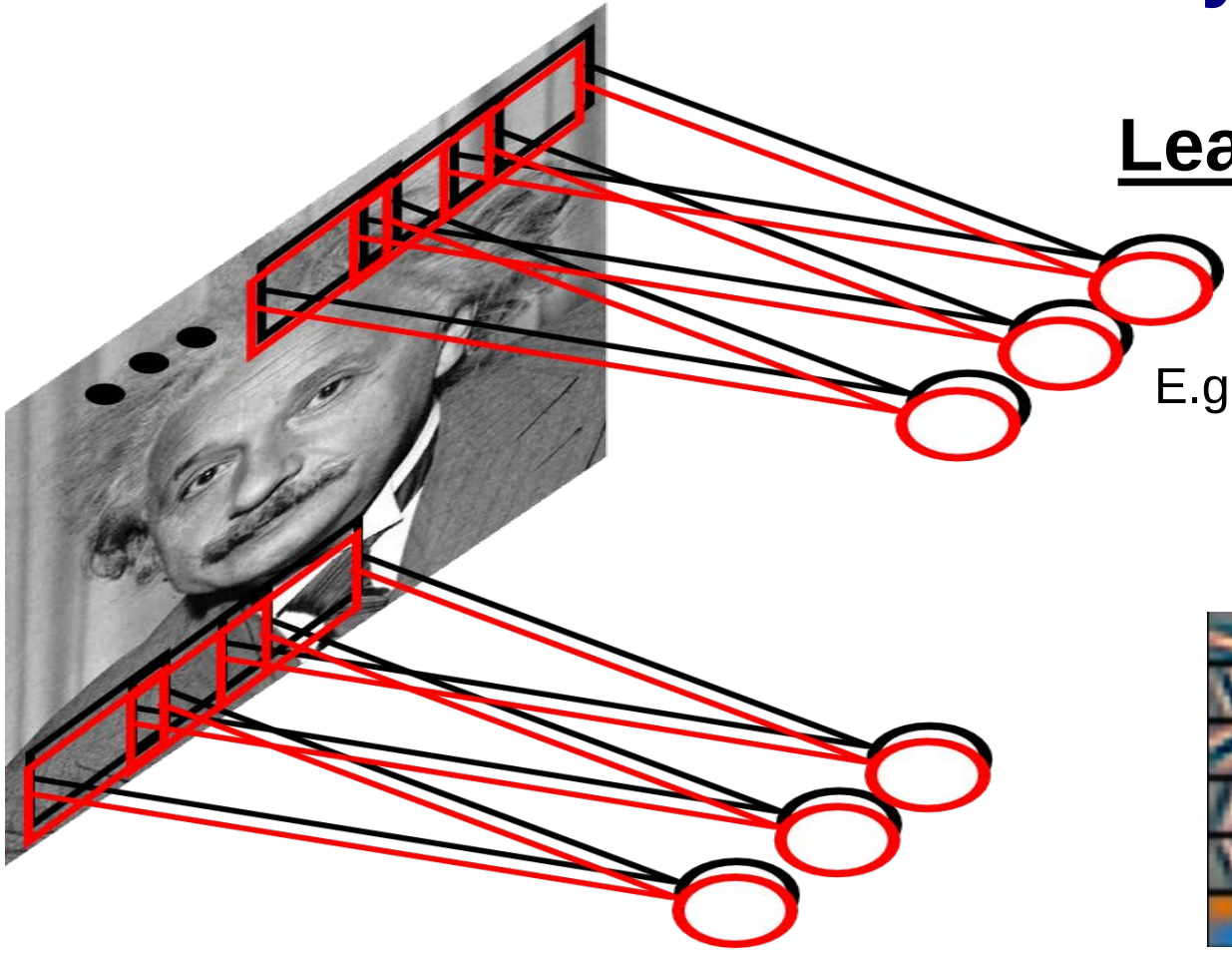
$$* \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} =$$



$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

elementwise multiplication and sum of a filter and the signal (image)

Convolutional Layer



Learn multiple filters.

E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters

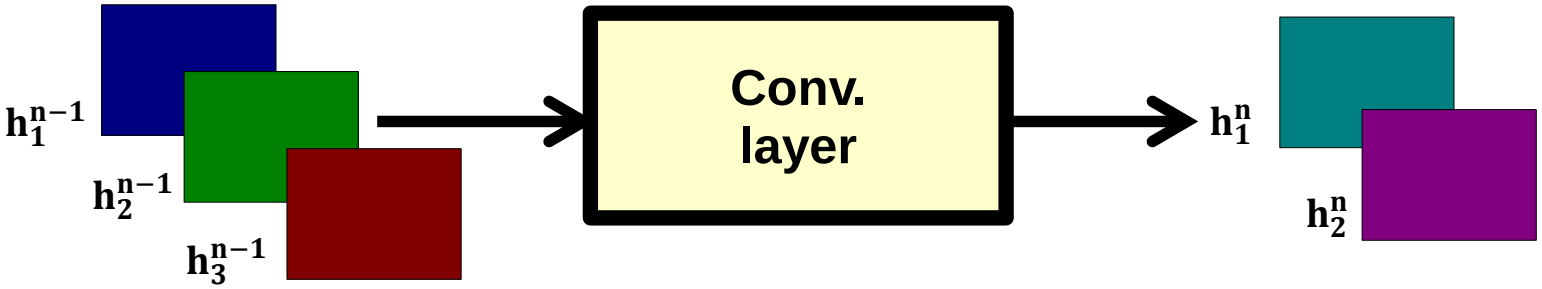
Convolutional Layer

$$h_j^n = max\left(0, \sum_{k=1}^{K} h_k^{n-1} * w_{kj}^n\right)$$

output feature map

input feature map

kernel



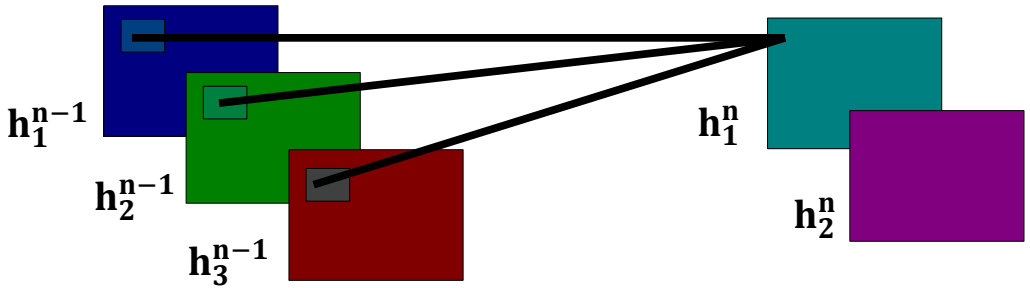
Convolutional Layer

$$h_j^n = max\left(0, \sum_{k=1}^{K} h_k^{n-1} * w_{kj}^n\right)$$

output feature map

input feature map

kernel



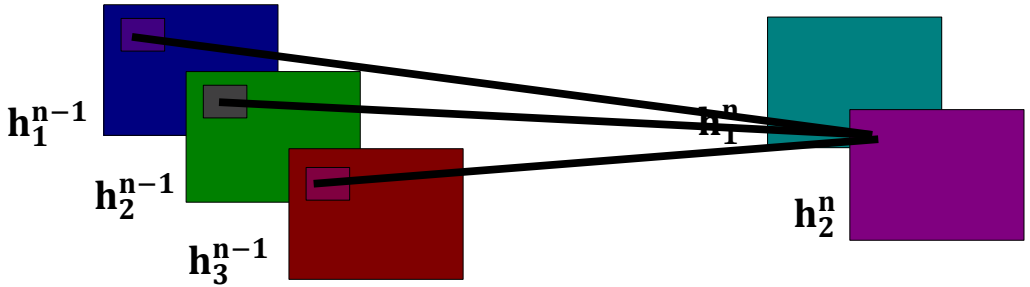
Convolutional Layer

$$h_j^n = max\left(0, \sum_{k=1}^{K} h_k^{n-1} * w_{kj}^n\right)$$

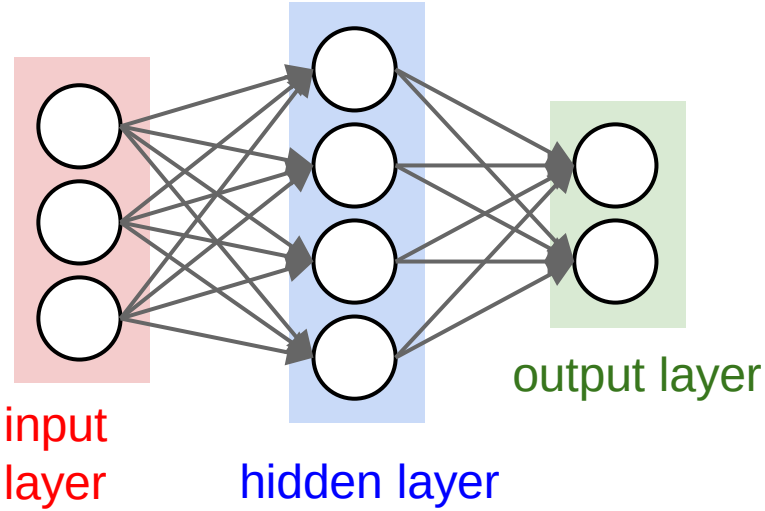
output feature map

input feature map

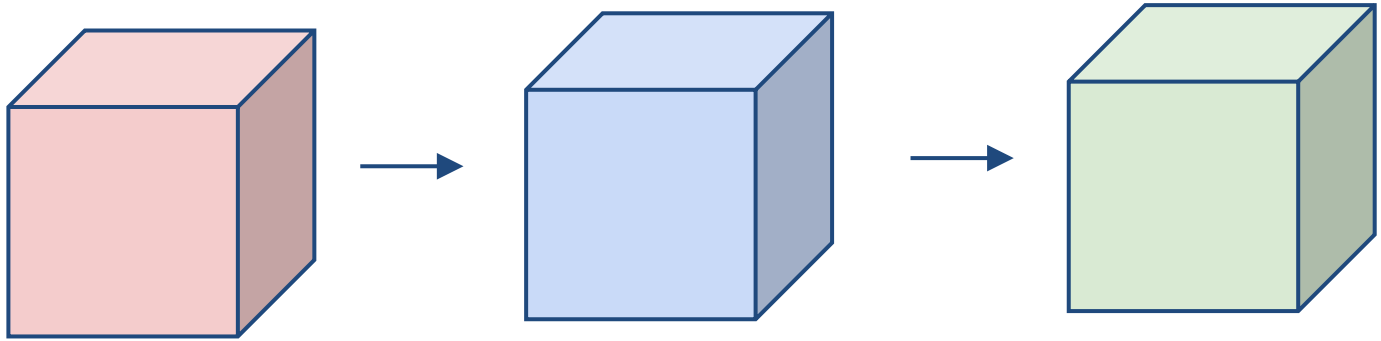
kernel



before:

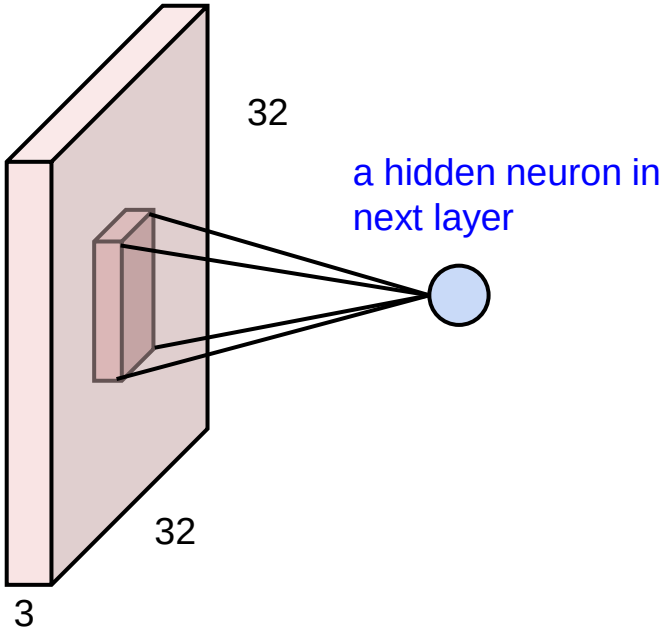


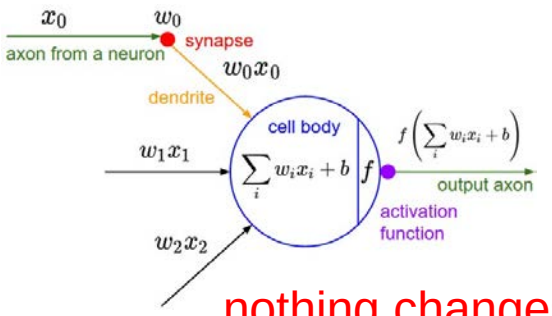
now:



Convolutional Neural Networks are just Neural Networks BUT:

1. **Local connectivity**

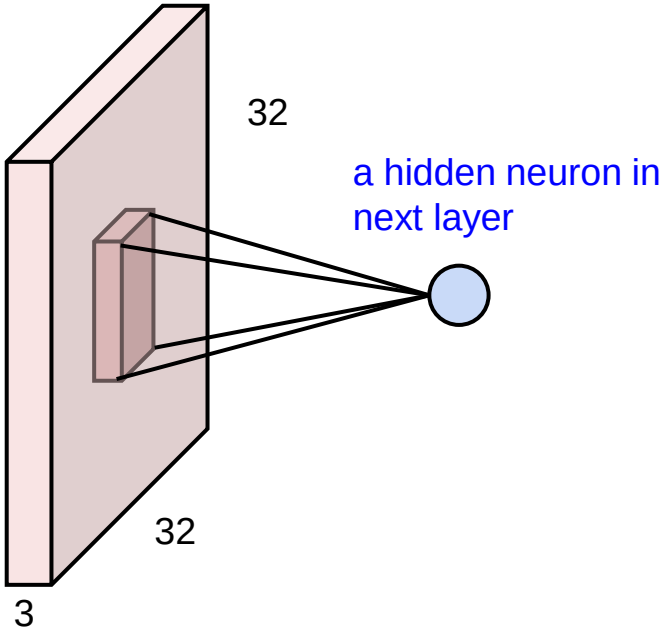


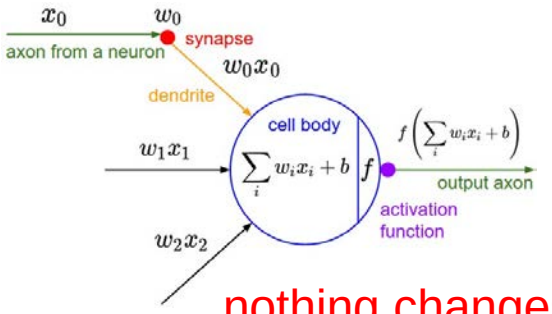


nothing changes

Convolutional Neural Networks are just Neural Networks BUT:

1. Local connectivity





nothing changes

image: 32x32x3 volume
before: full connectivity: 32x32x3 weights

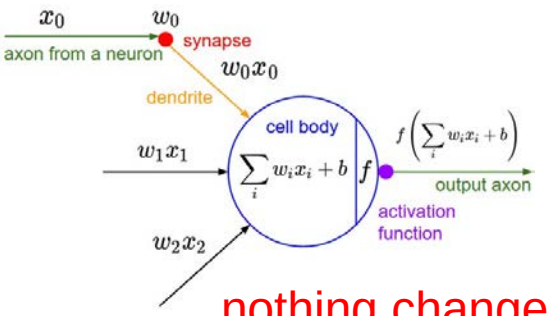
now: one neuron will connect to, e.g. 5x5x3 chunk and only have 5x5x3 weights.

note that connectivity is:

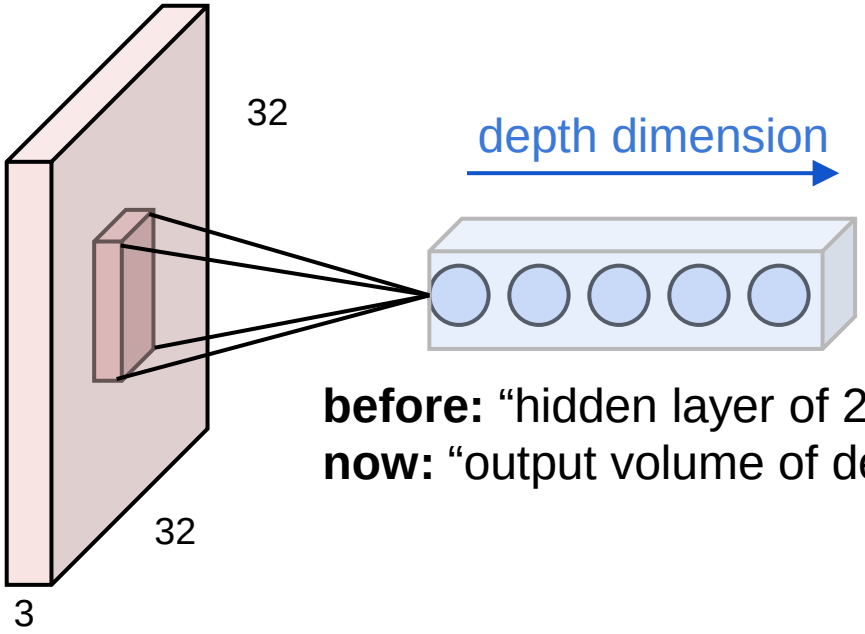
- local in space (5x5 inside 32x32)
- but full in depth (all 3 depth channels)

Convolutional Neural Networks are just Neural Networks BUT:

1. **Local connectivity**



nothing changes

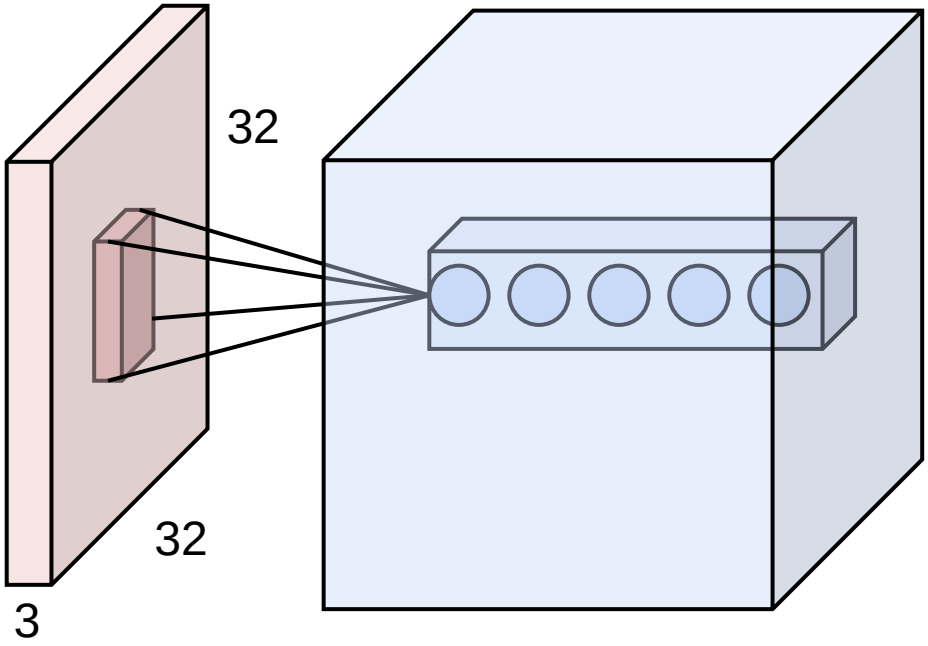


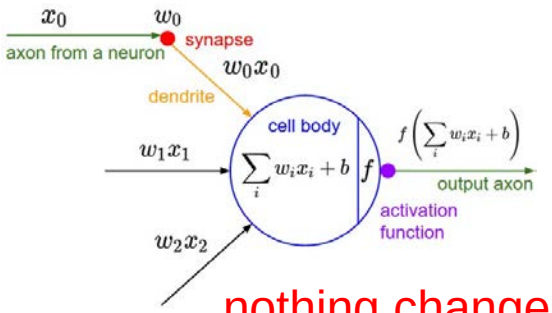
before: “hidden layer of 200 neurons”
now: “output volume of depth 200”

Multiple neurons all looking at the same region of the input volume, stacked along depth.

Convolutional Neural Networks are just Neural Networks BUT:

1. **Local connectivity**

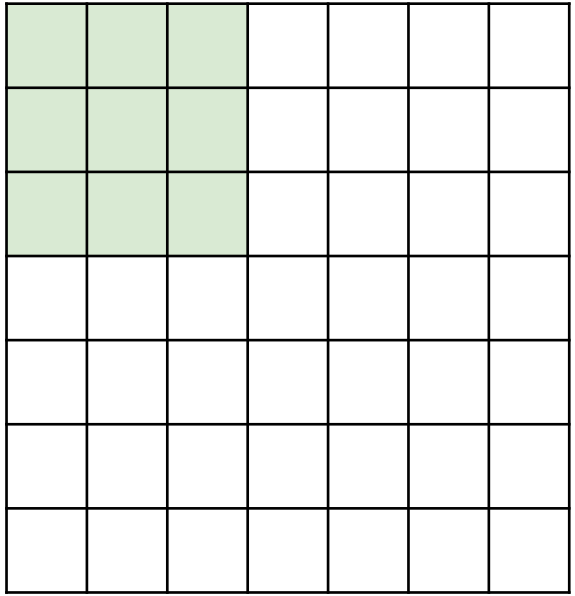




nothing changes

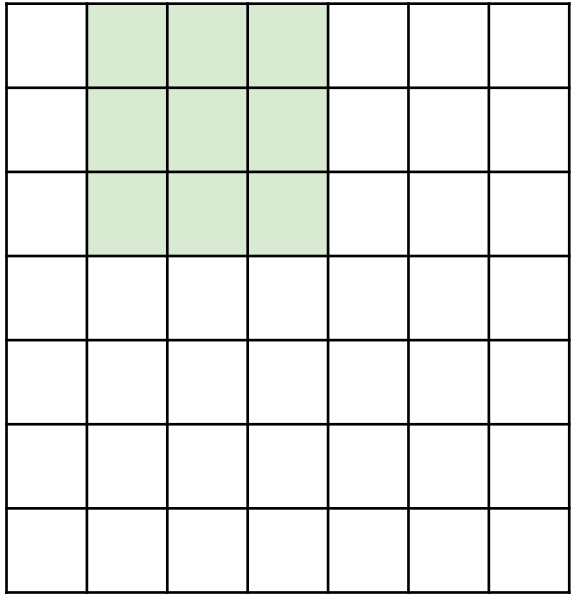
These form a single [1 x 1 x depth] “depth column” in the output volume

Replicate this column of hidden neurons across space, with some **stride.**



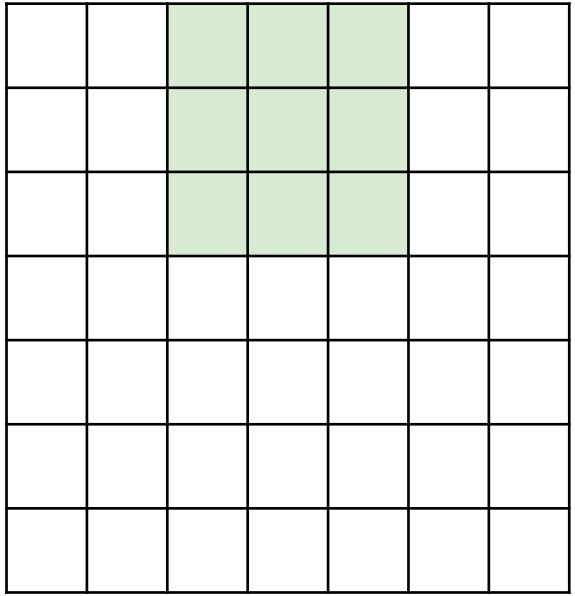
7x7 input
assume 3x3 connectivity, stride 1

Replicate this column of hidden neurons across space, with some **stride.**



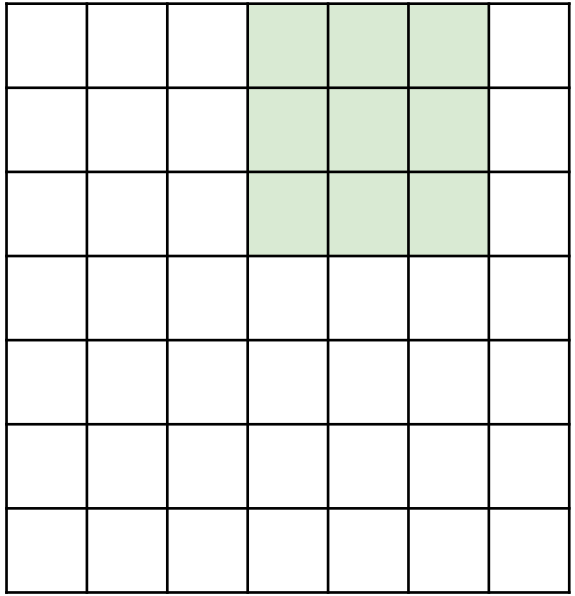
7x7 input
assume 3x3 connectivity, stride 1

Replicate this column of hidden neurons across space, with some **stride.**



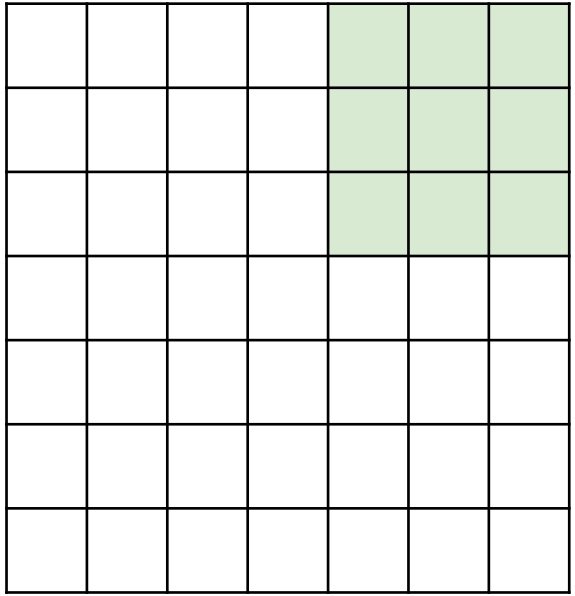
7x7 input
assume 3x3 connectivity, stride 1

Replicate this column of hidden neurons across space, with some **stride.**



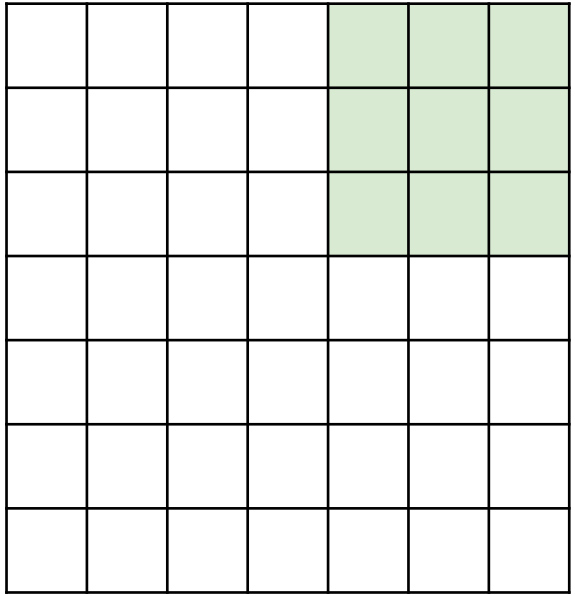
7x7 input
assume 3x3 connectivity, stride 1

Replicate this column of hidden neurons across space, with some **stride.**



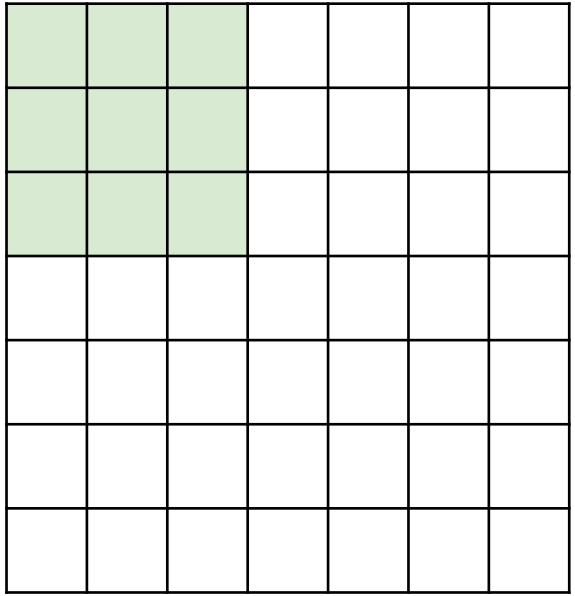
7x7 input
assume 3x3 connectivity, stride 1

Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

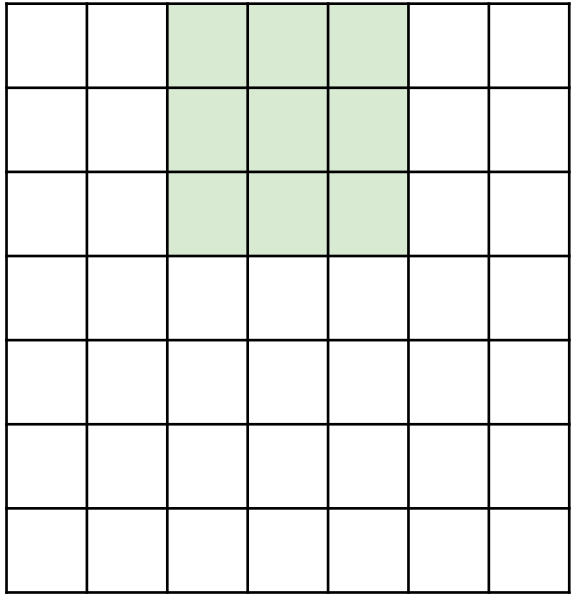
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?

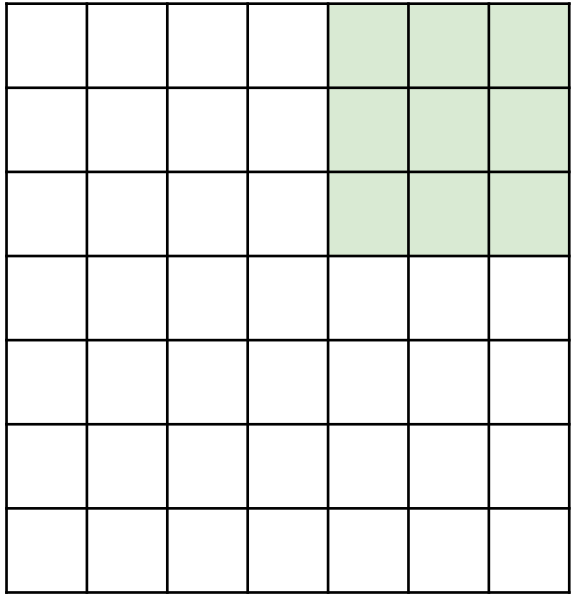
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?

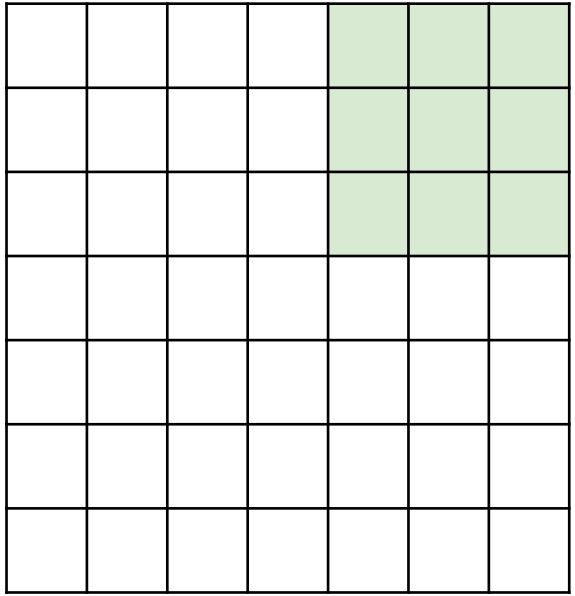
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?

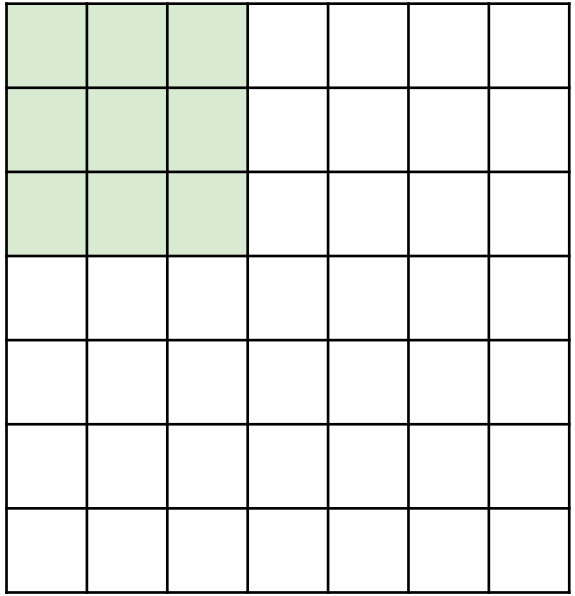
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?
=> **3x3 output**

Replicate this column of hidden neurons across space, with some **stride.**

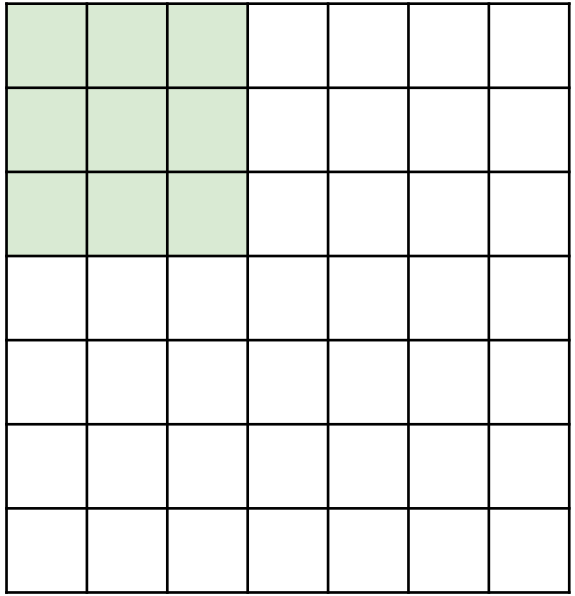


7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?
=> **3x3 output**

what about stride 3?

Replicate this column of hidden neurons across space, with some **stride.**



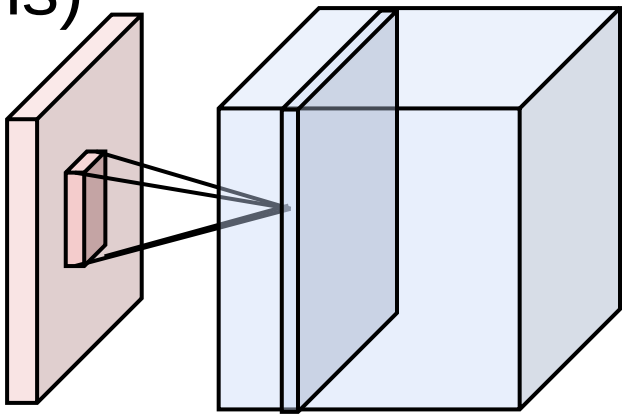
7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?
=> **3x3 output**

what about stride 3? Cannot.

Convolutional Layers

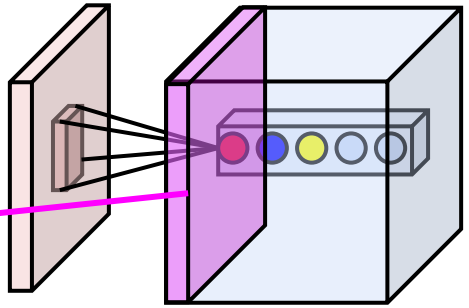
1. Connect neurons only to local receptive fields
2. Share weights: use the *same neuron weight parameters* for neurons in each “depth slice” (i.e. across spatial positions)



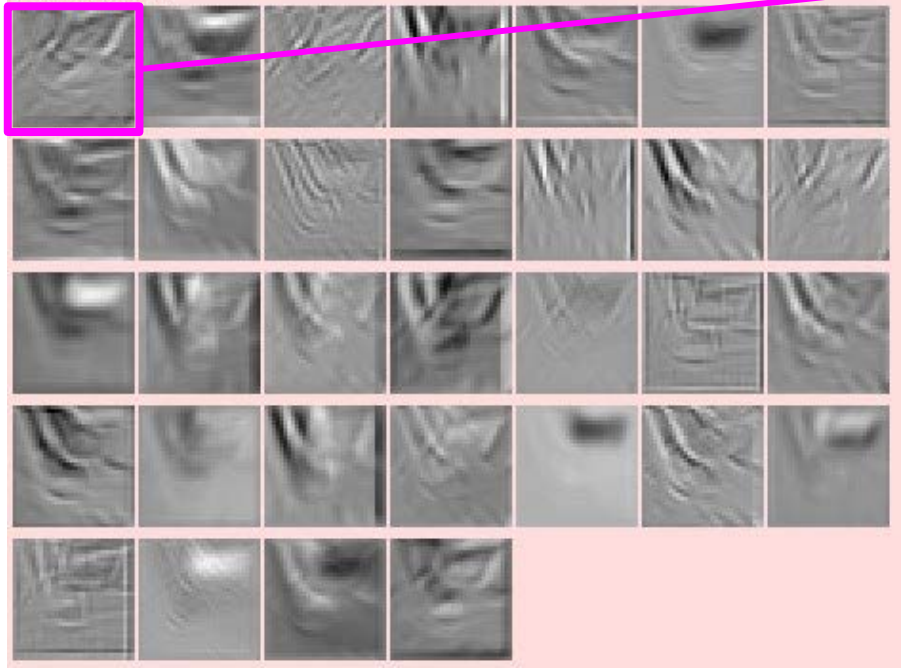
one activation map (a depth slice), computed with one set of weights

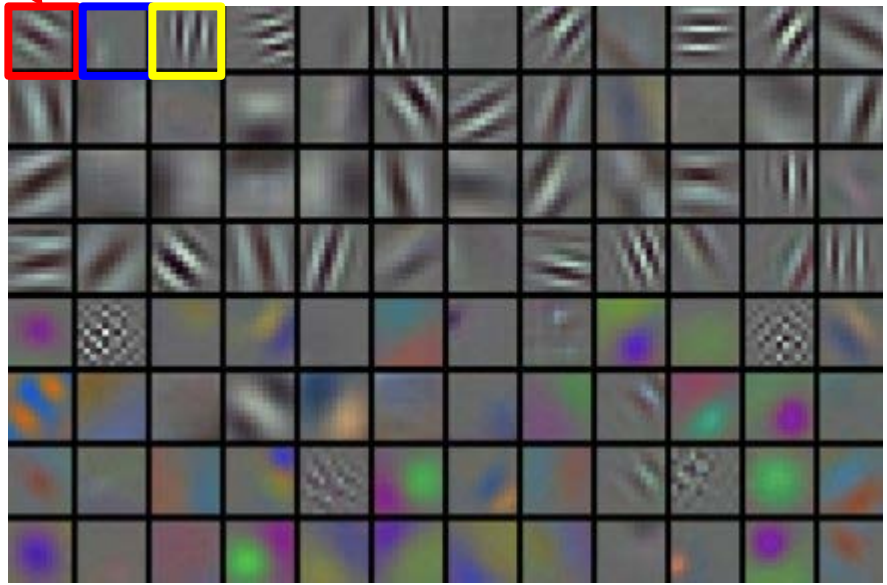
Activations:

convolving the first filter in the input gives the first slice of depth in output volume

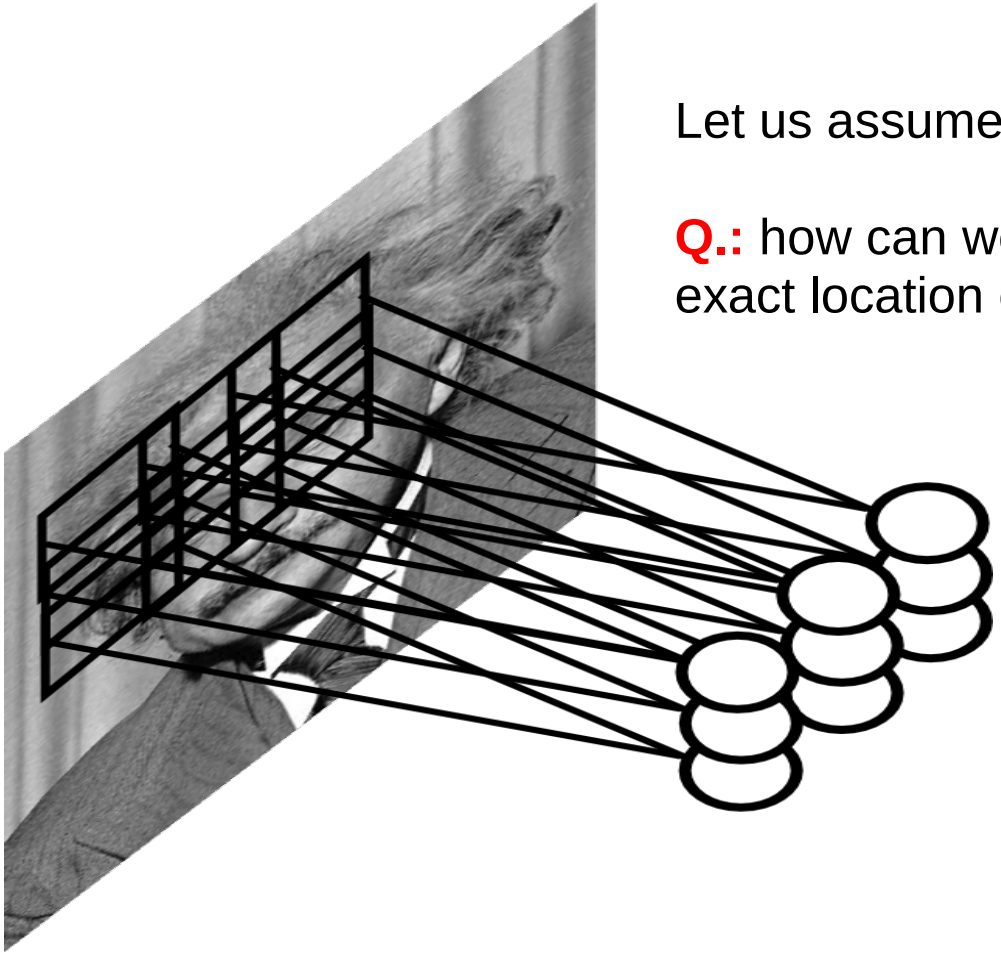


Activations:





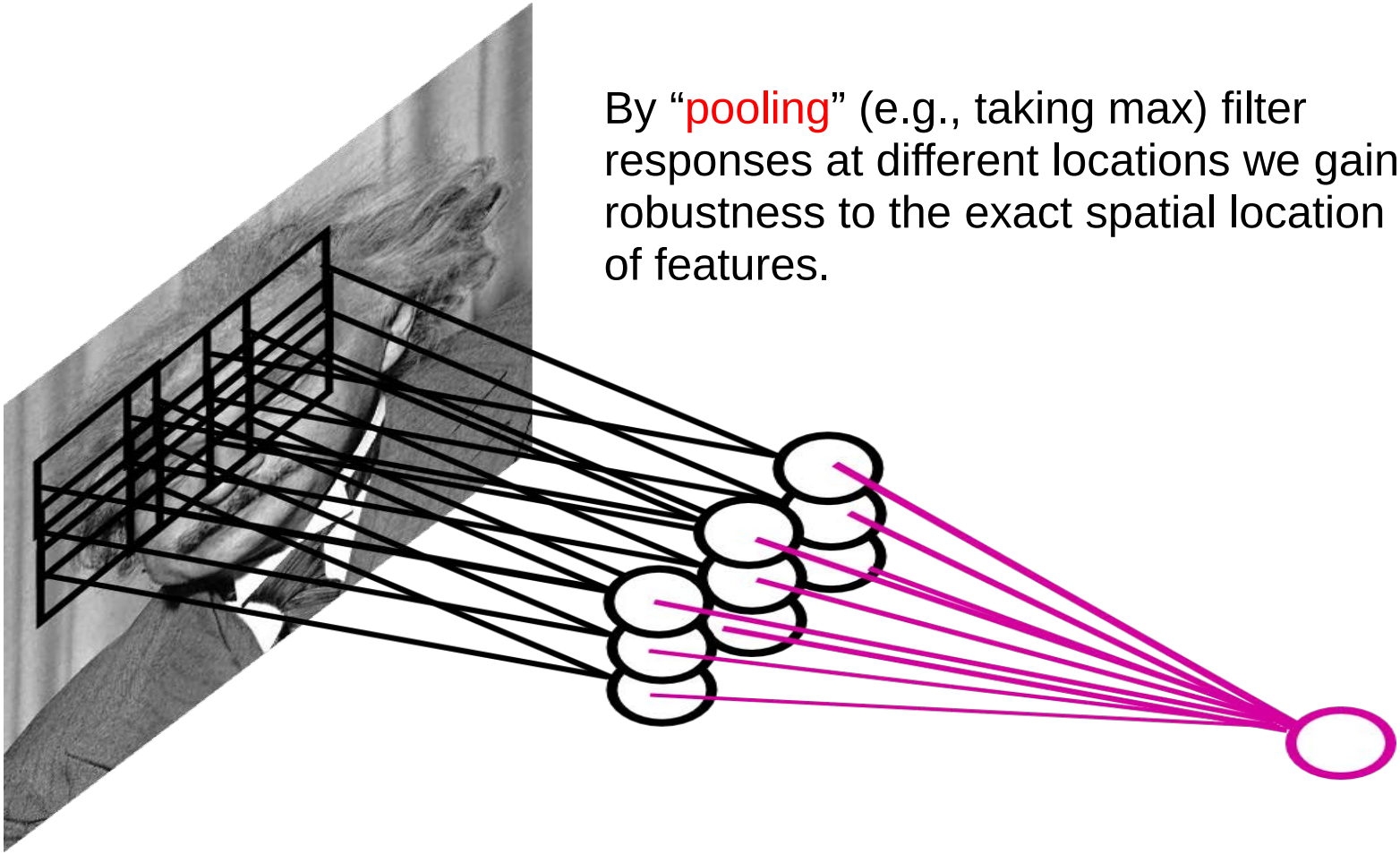
Pooling Layer



Let us assume filter is an “eye” detector.

Q.: how can we make the detection robust to the exact location of the eye?

Pooling Layer



By “pooling” (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.

Pooling Layer: Examples

Max-pooling:

$$h_j^n(x,y) = max_{\tilde{x}\in N(x),\tilde{y}\in N(y)} h_j^{n-1}\left(\tilde{x},\tilde{y}\right)$$

Average-pooling:

$$h_j^n(x,y) = 1/K \sum_{\tilde{x}\in N(x),\tilde{y}\in N(y)} h_j^{n-1}\left(\tilde{x},\tilde{y}\right)$$

L2-pooling:

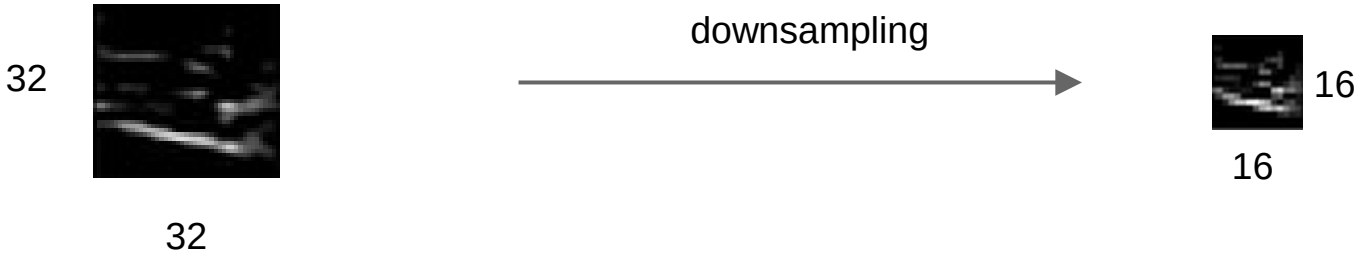
$$h_j^n(x,y) = \sqrt{\sum_{\tilde{x}\in N(x),\tilde{y}\in N(y)} h_j^{n-1}\left(\tilde{x},\tilde{y}\right)^2}$$

L2-pooling over features:

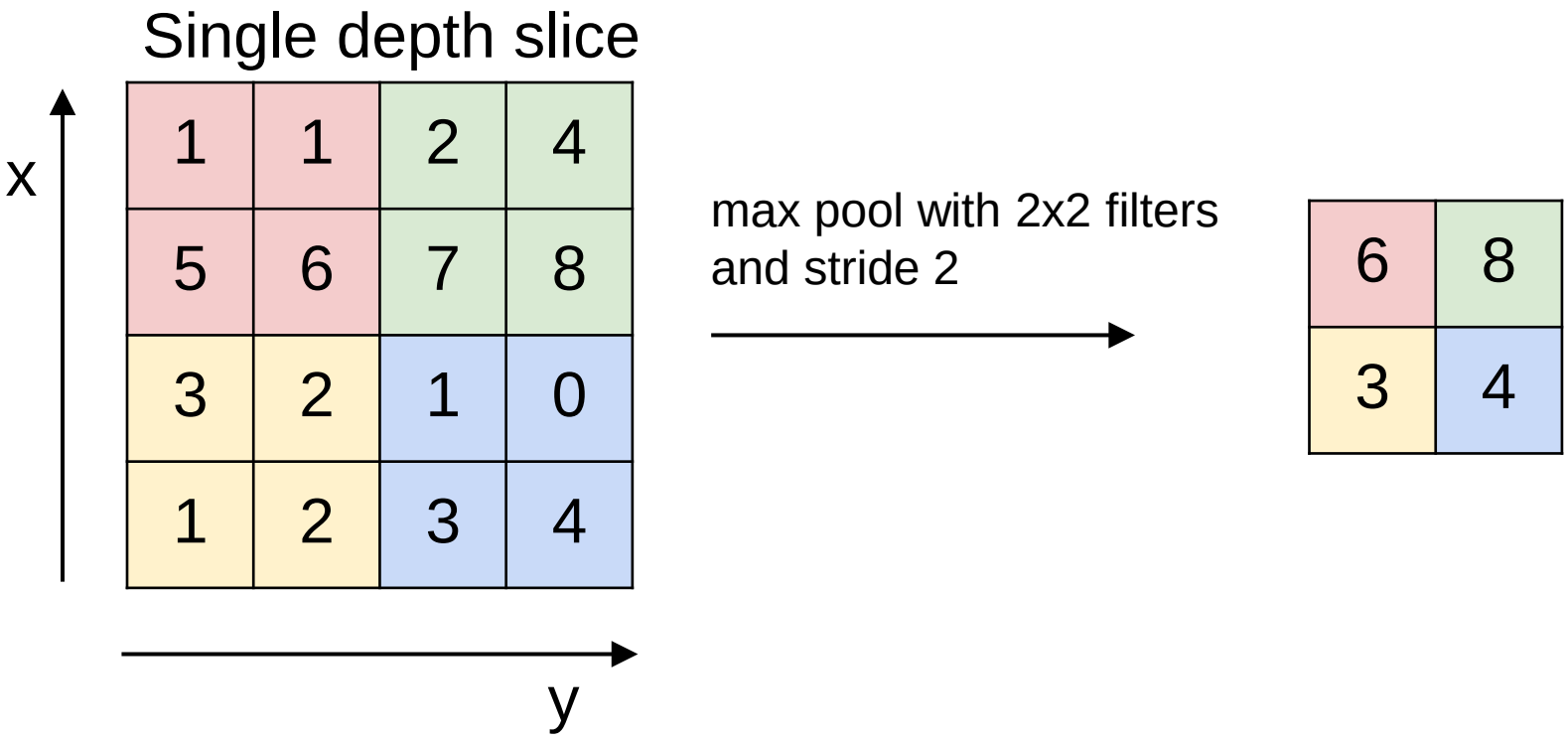
$$h_j^n(x,y) = \sqrt{\sum_{k\in N(j)} h_k^{n-1}(x,y)^2}$$

In ConvNet architectures, **Conv** layers are often followed by **Pool** layers

- convenience layer: makes the representations smaller and more manageable without losing too much information. Computes MAX operation (most common)

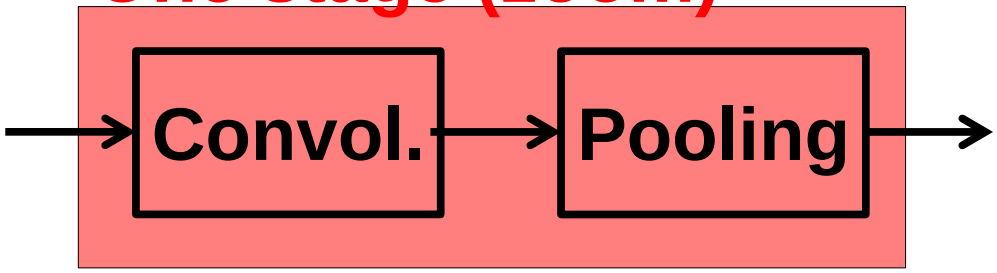


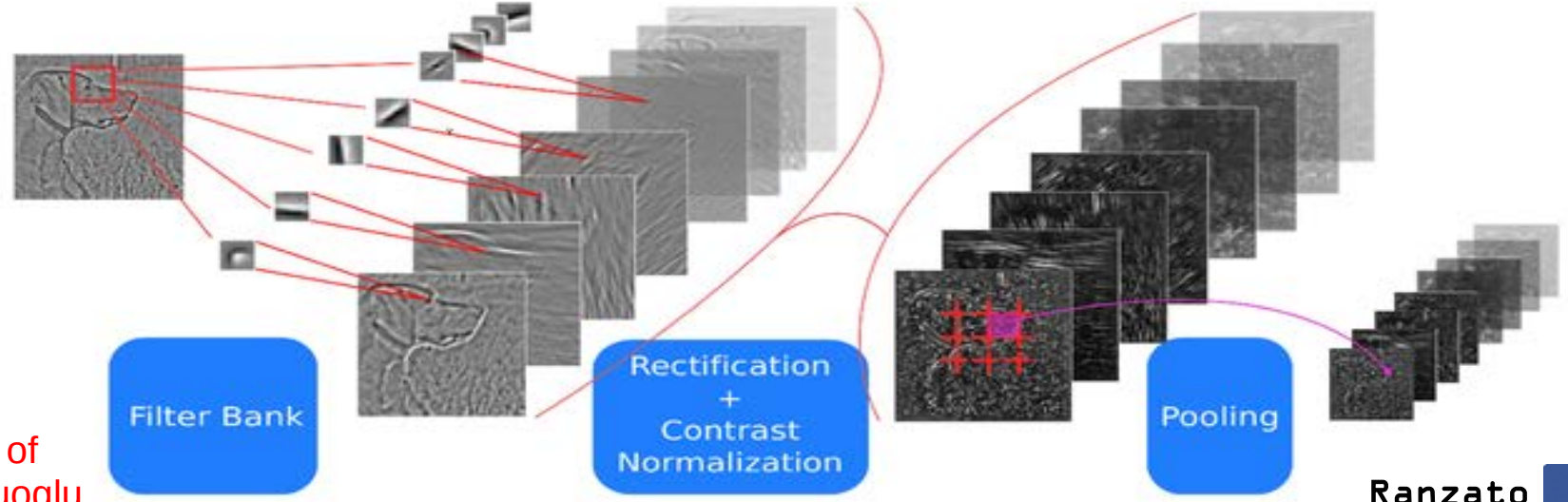
MAX POOLING


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

ConvNets: Typical Stage

One stage (zoom)

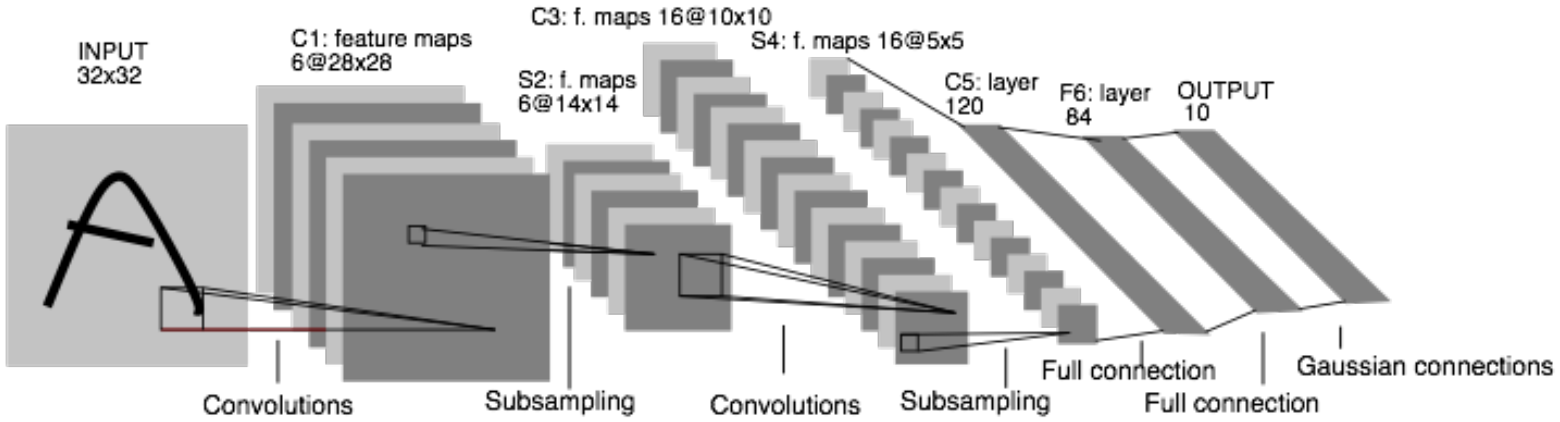




courtesy of
K. Kavukcuoglu

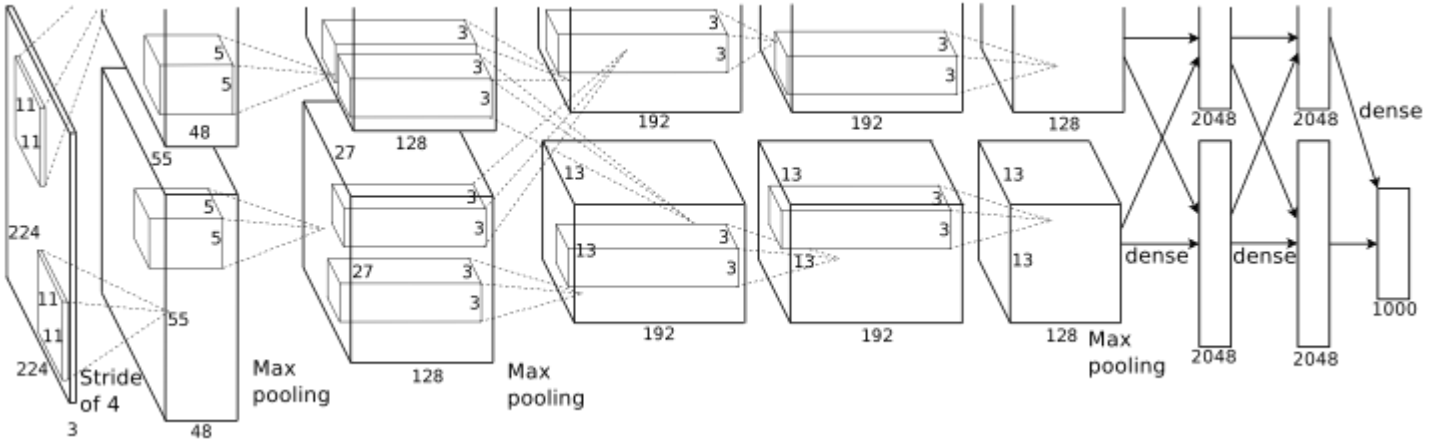
Ranzato

Convolutional Neural Nets (CNNs): 1989



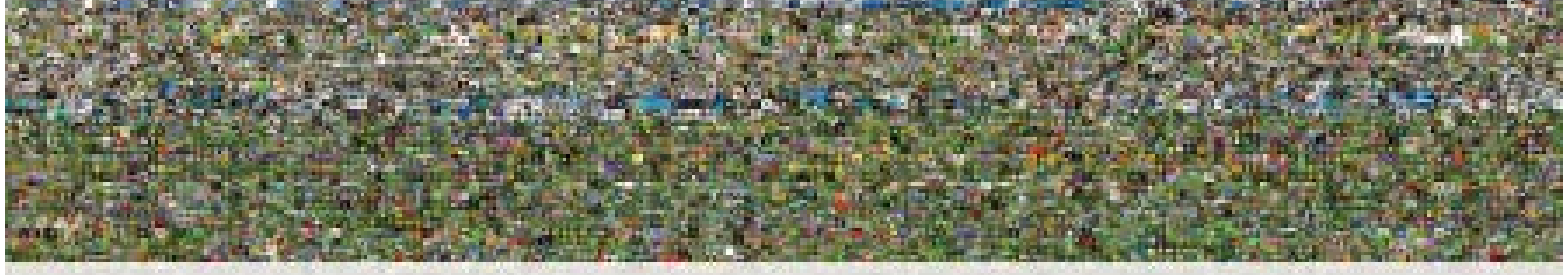
LeNet: a layered model composed of convolution and subsampling operations followed by a holistic representation and ultimately a classifier for handwritten digits. [LeNet]

Convolutional Nets: (Krizevsky et al, 2012)



AlexNet: a layered model composed of convolution, subsampling, and further operations followed by a holistic representation and all-in-all a landmark classifier on ILSVRC12. [AlexNet]

+ data
+ gpu
+ non-saturating nonlinearity
+ regularization

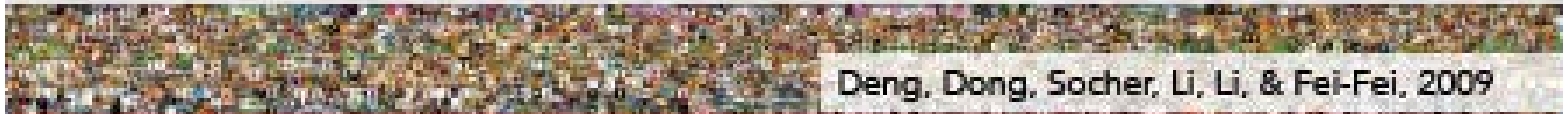


IMAGENET

www.image-net.org

22K categories and **14M** images

- Animals
  - Bird
  - Fish
  - Mammal
  - Invertebrate
- Plants
  - Tree
  - Flower
- Food
- Materials
- Structures
- Artifact
  - Tools
  - Appliances
  - Structures
- Person
- Scenes
  - Indoor
  - Geological Formations
- Sport Activities



Deng, Dong, Socher, Li, Li, & Fei-Fei, 2009

IMAGENET Large Scale Visual Recognition Challenge

The Image Classification Challenge:

1,000 object classes

1,431,167 images

Output:
Scale
T-shirt
Steel drum
Drumstick
Mud turtle

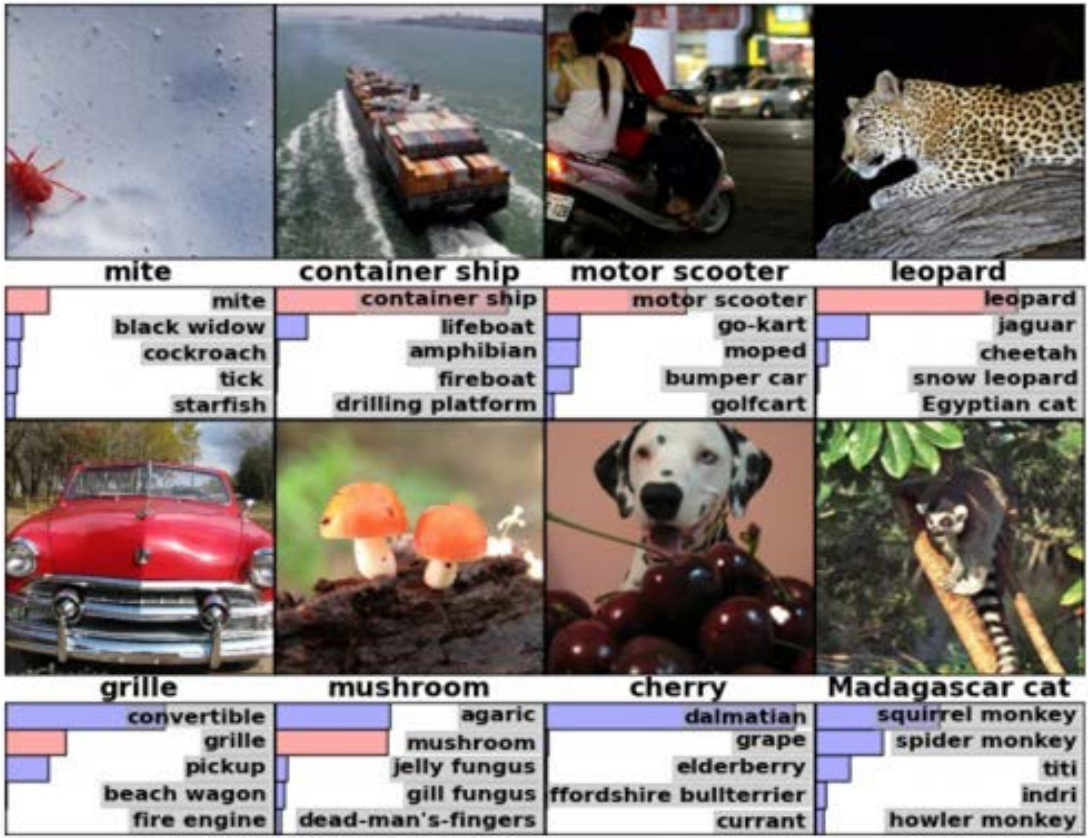
✔

Output:
Scale
T-shirt
Giant panda
Drumstick
Mud turtle

✗

Russakovsky et al. arXiv, 2014

ImageNet Challenge

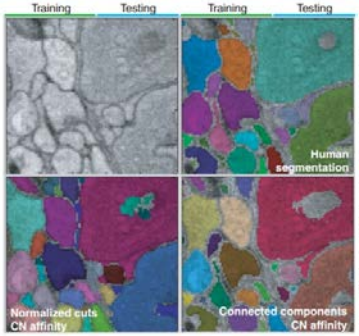


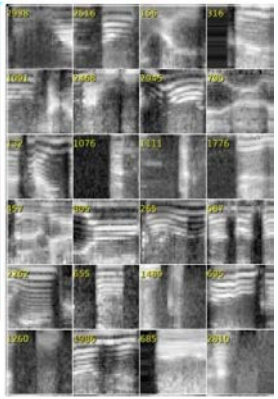
(show convnetjs example of training of CIFAR-10)

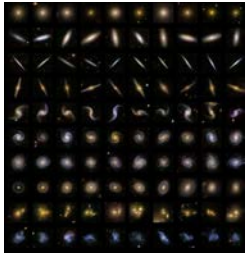
demo

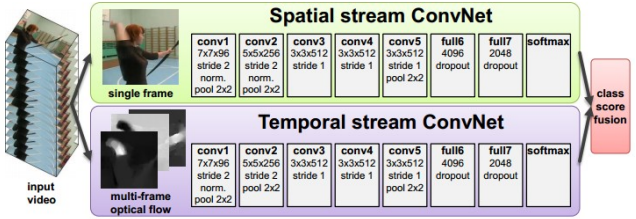
(show convnetjs example of training of CIFAR-10)

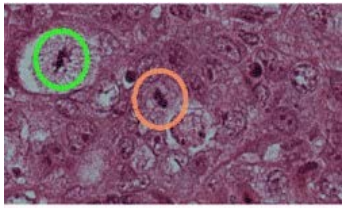
demo

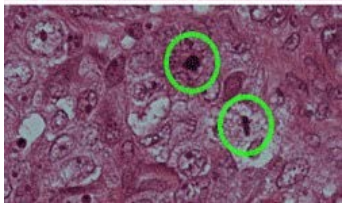

Training
Testing
Training
Testing
Human segmentation
Normalized cuts CN affinity
Connected components CN affinity

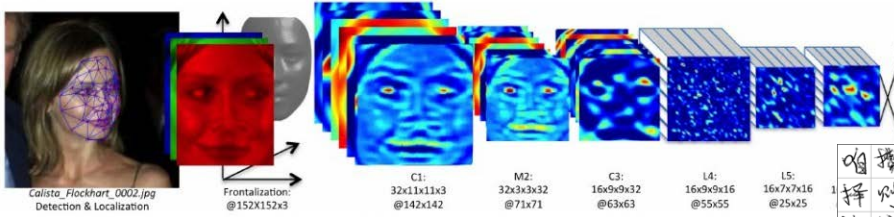




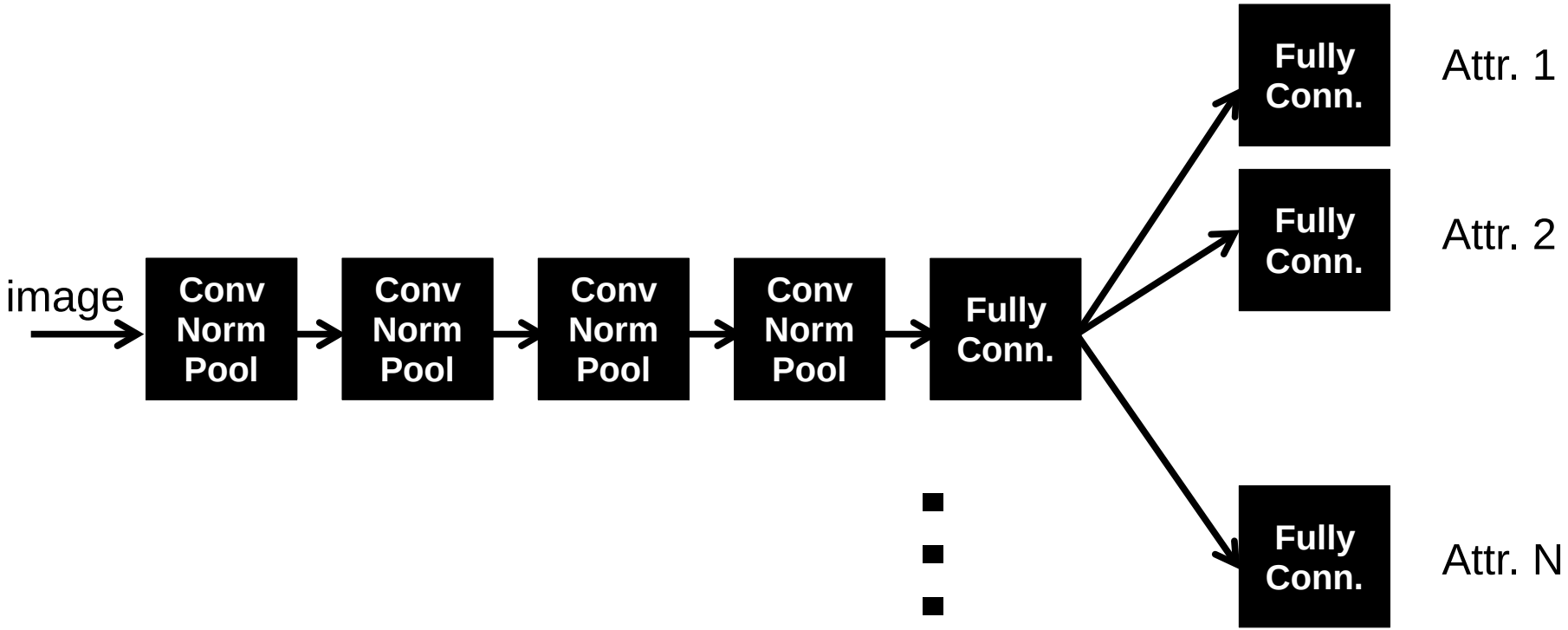

Spatial stream ConvNet
single frame
conv1 7x7x96 stride 2 norm. pool 2x2
conv2 5x5x256 stride 2 norm. pool 2x2
conv3 3x3x512 stride 1
conv4 3x3x512 stride 1
conv5 3x3x512 stride 1 pool 2x2
full6 4096 dropout
full7 2048 dropout
softmax
Temporal stream ConvNet
multi-frame optical flow
input video
class score fusion






Calista_Flockhart_0002.jpg
Detection & Localization
Frontalization: @152x152x3
C1: 32x11x11x3 @142x142
M2: 32x3x3x32 @71x71
C3: 16x9x9x32 @63x63
L4: 16x9x9x16 @55x55
L5: 16x7x7x16 @25x25
REPRESENTATION
SFC labels

Fancier Architectures: Multi-Task



Zhang et al. “PANDA..” CVPR 2014

Fancier Architectures: Multi-Task

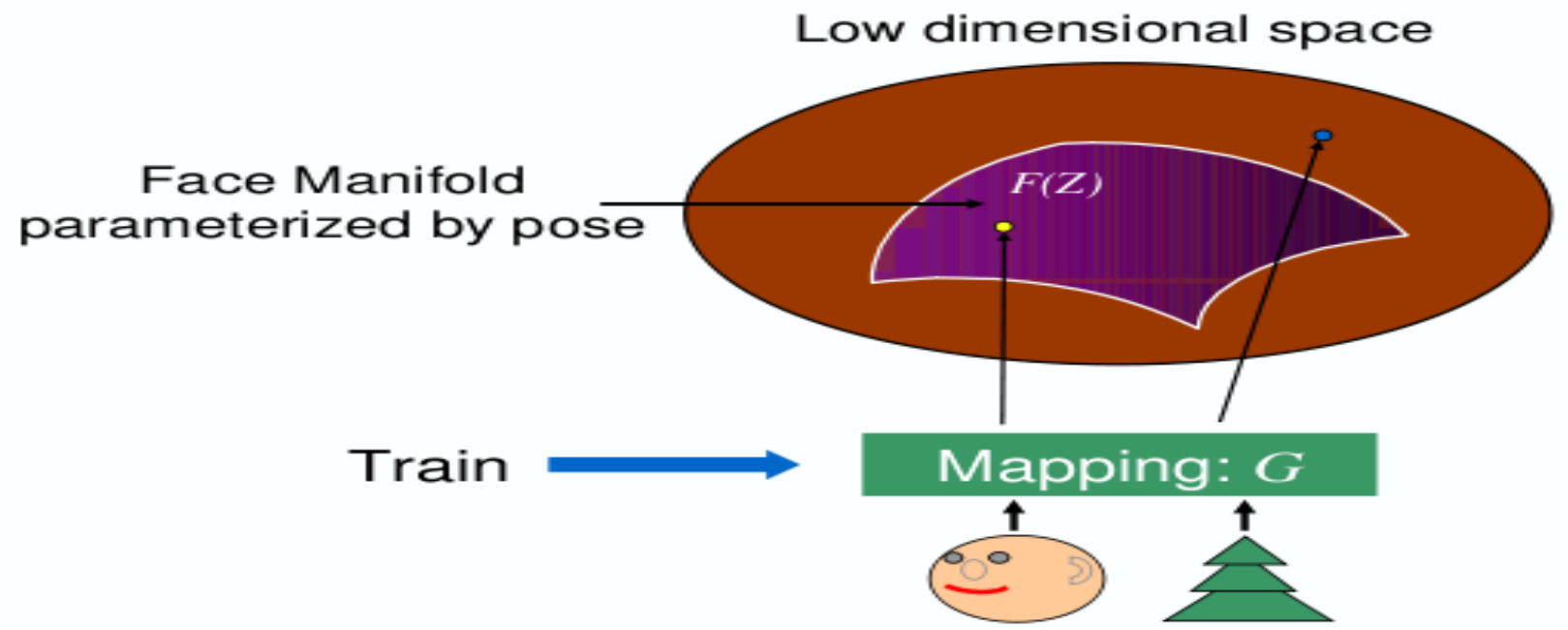
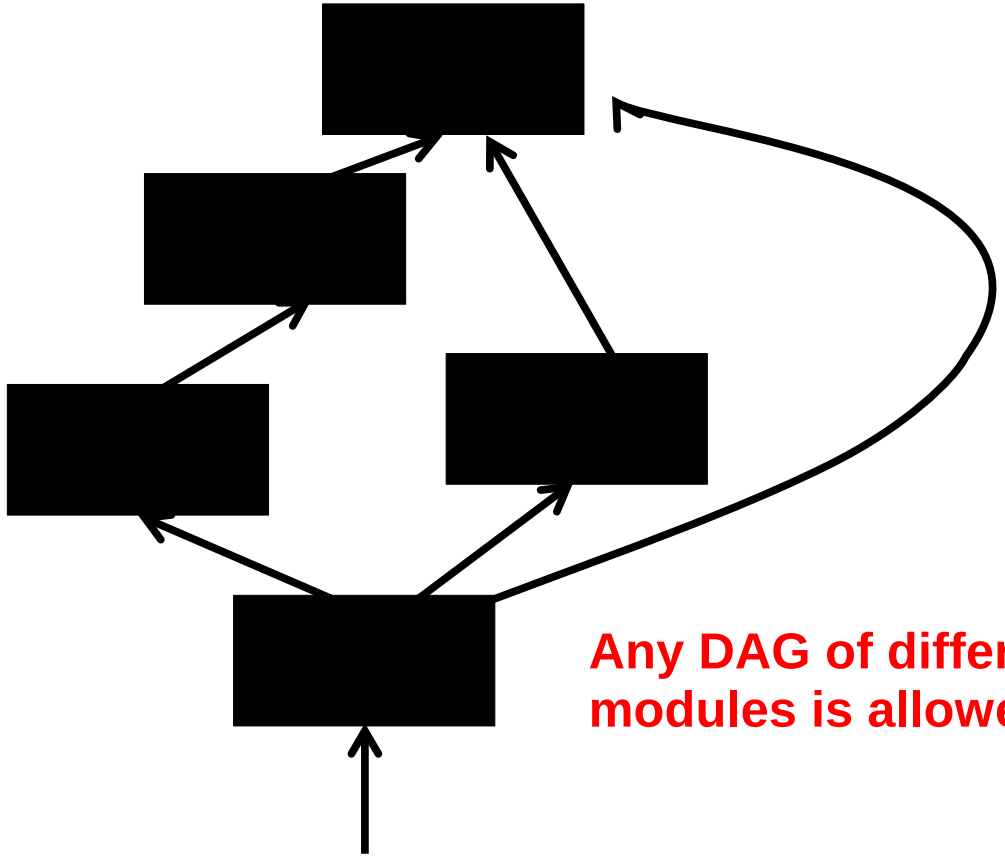


Figure 1: Manifold Mapping—Training

Osadchy et al. “Synergistic face detection and pose estimation..” JMLR 2007

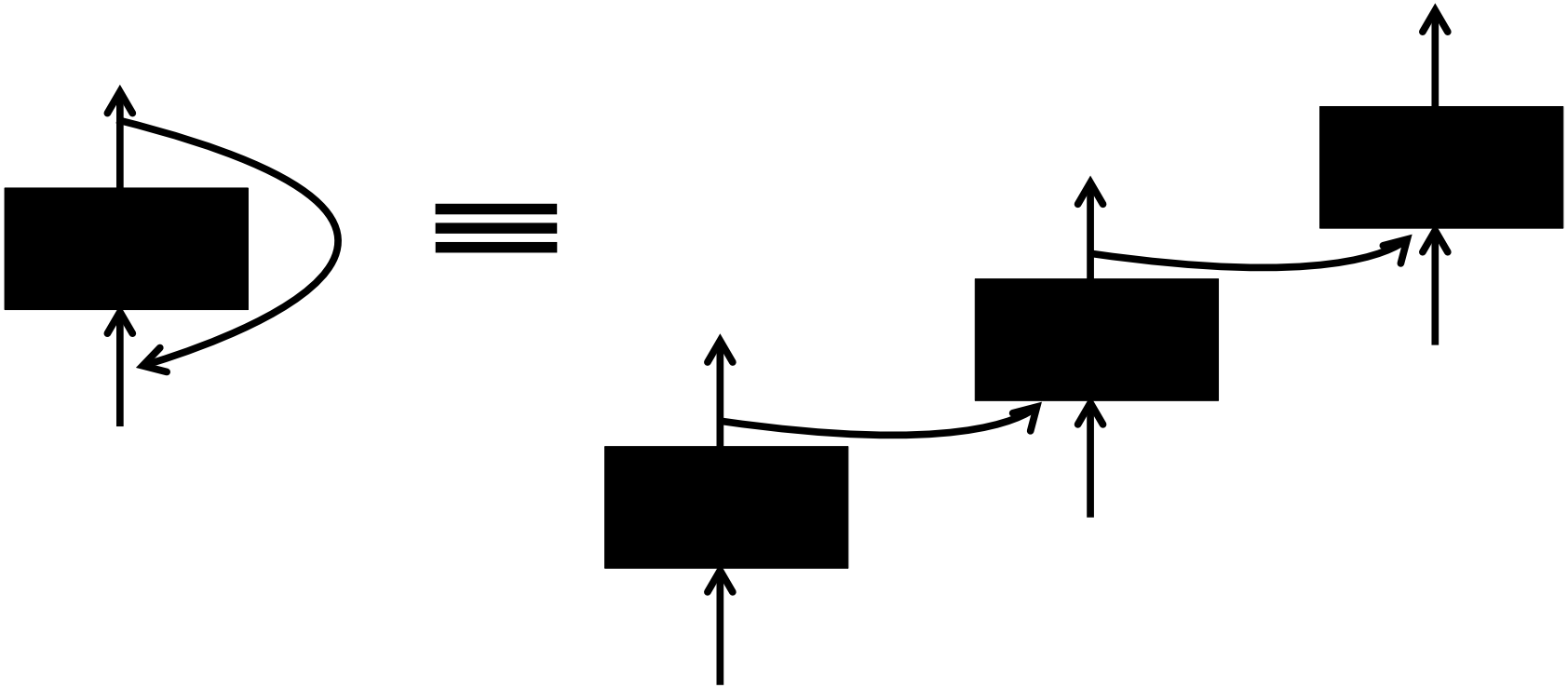
Fancier Architectures: Generic DAG



Any DAG of differentialble modules is allowed!

Fancier Architectures: Generic DAG

If there are cycles (RNN), one needs to un-roll it.



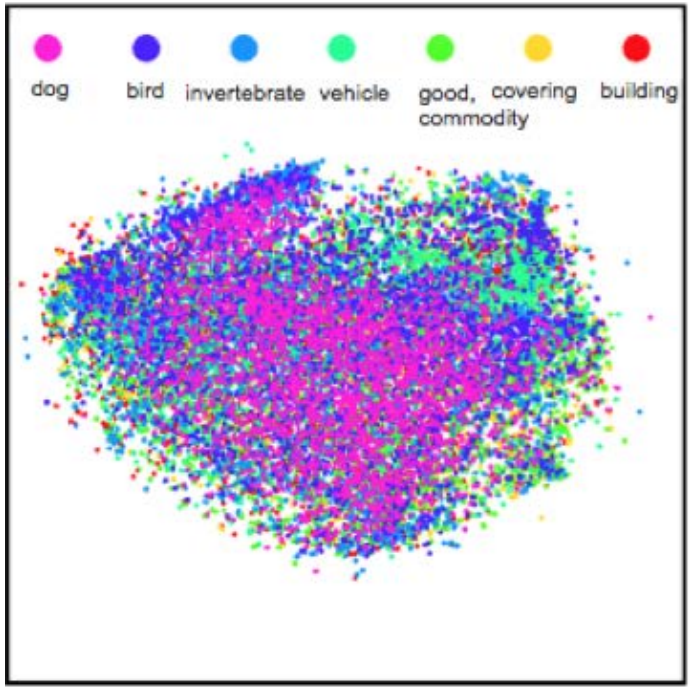
Pinheiro, Collobert “Recurrent CNN for scene labeling” ICML 2014
Graves “Offline Arabic handwriting recognition..” Springer 2012

“You need a lot of a data if you want to train/use CNNs”

Transfer Learning

“You need a lot of a data if you want to train/use CNNs”

The Unreasonable Effectiveness of Deep Features



Low-level: $Pool_1$

High-level: FC_6

Classes separate in the deep representations and transfer to many tasks. [DeCAF] [Zeiler-Fergus]

Can be used as a generic feature

(“CNN code” = 4096-D vector before classifier)

query image

nearest neighbors in the “code” space

Transfer Learning with CNNs

1. Train on Imagenet

Transfer Learning with CNNs

1. Train on Imagenet

2. If small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier

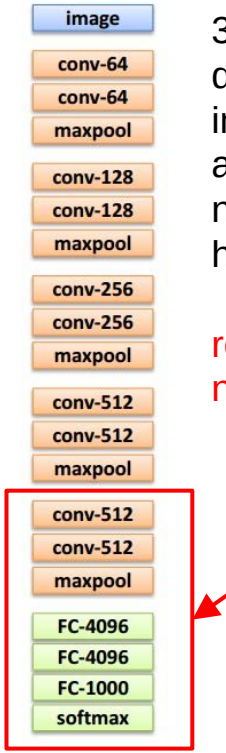
i.e. swap the Softmax layer at the end

Transfer Learning with CNNs

1. Train on Imagenet

2. If small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier

i.e. swap the Softmax layer at the end



3. If you have medium sized dataset, **"finetune"** instead: use the old weights as initialization, train the full network or only some of the higher layers

retrain bigger portion of the network, or even all of it.

Transfer Learning with CNNs

1. Train on Imagenet

2. If small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier

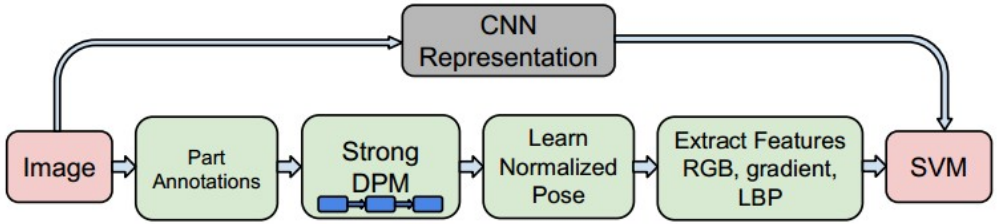
i.e. swap the Softmax layer at the end

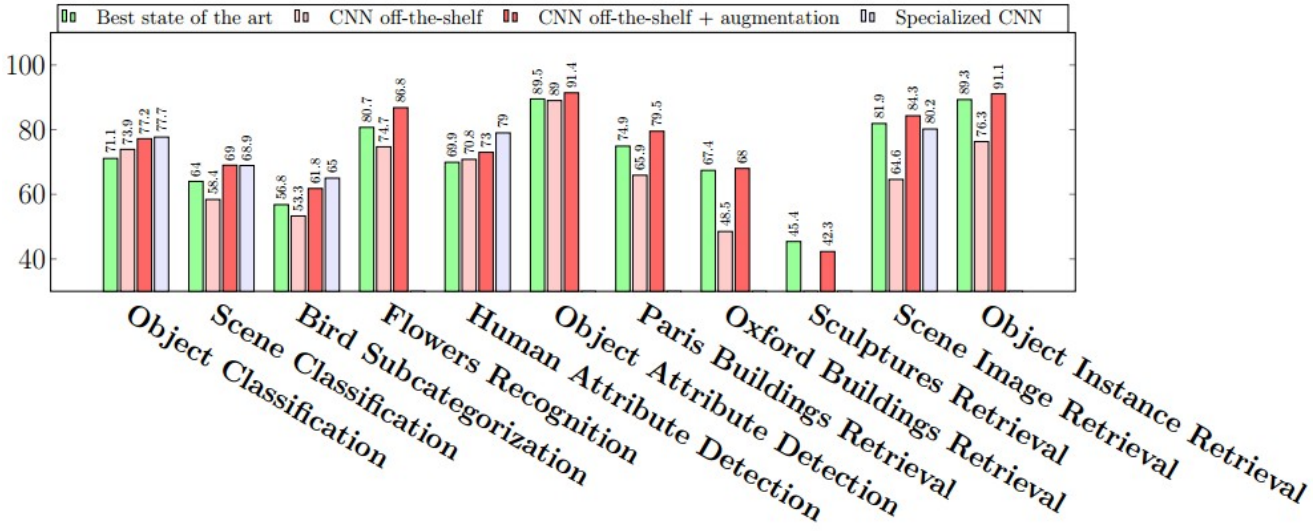
3. If you have medium sized dataset, **"finetune"** instead: use the old weights as initialization, train the full network or only some of the higher layers

retrain bigger portion of the network, or even all of it.

tip: use only ~1/10th of the original learning rate in finetuning to player, and ~1/100th on intermediate layers

CNN Features off-the-shelf: an Astounding Baseline for Recognition
[Razavian et al, 2014]





SOFTWARE

Torch7: learning library that supports neural net training

torch.ch
http://code.cogbits.com/wiki/doku.php (tutorial with demos by C. Farabet)
https://github.com/jhjin/overfeat-torch
https://github.com/facebook/fbcunn/tree/master/examples/imagenet

Python-based learning library (U. Montreal)

- http://deeplearning.net/software/theano/ (does automatic differentiation)

Efficient CUDA kernels for ConvNets (Krizhevsky)

– code.google.com/p/cuda-convnet

Caffe (Yangqing Jia)

– http://caffe.berkeleyvision.org