

Convolutional Neural Nets

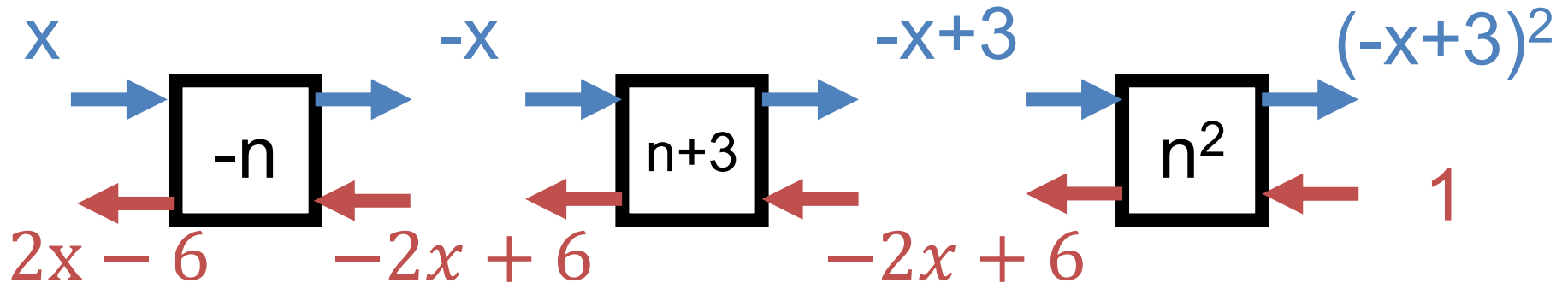
EECS 442 – David Fouhey

Fall 2019, University of Michigan

http://web.eecs.umich.edu/~fouhey/teaching/EECS442_F19/

Previously – Backpropagation

$$f(x) = (-x + 3)^2$$



Forward pass: compute function

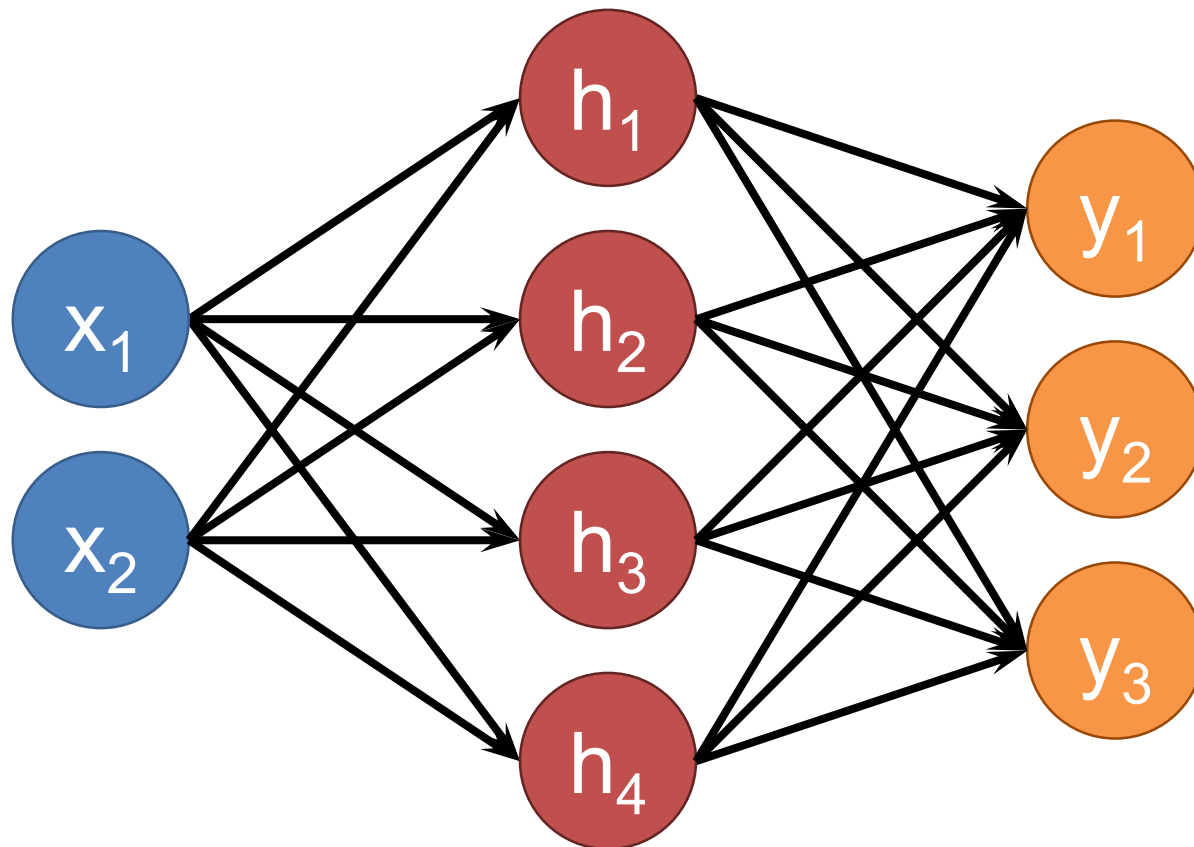
Backward pass: compute derivative of all parts of the function

Setting Up A Neural Net

Input

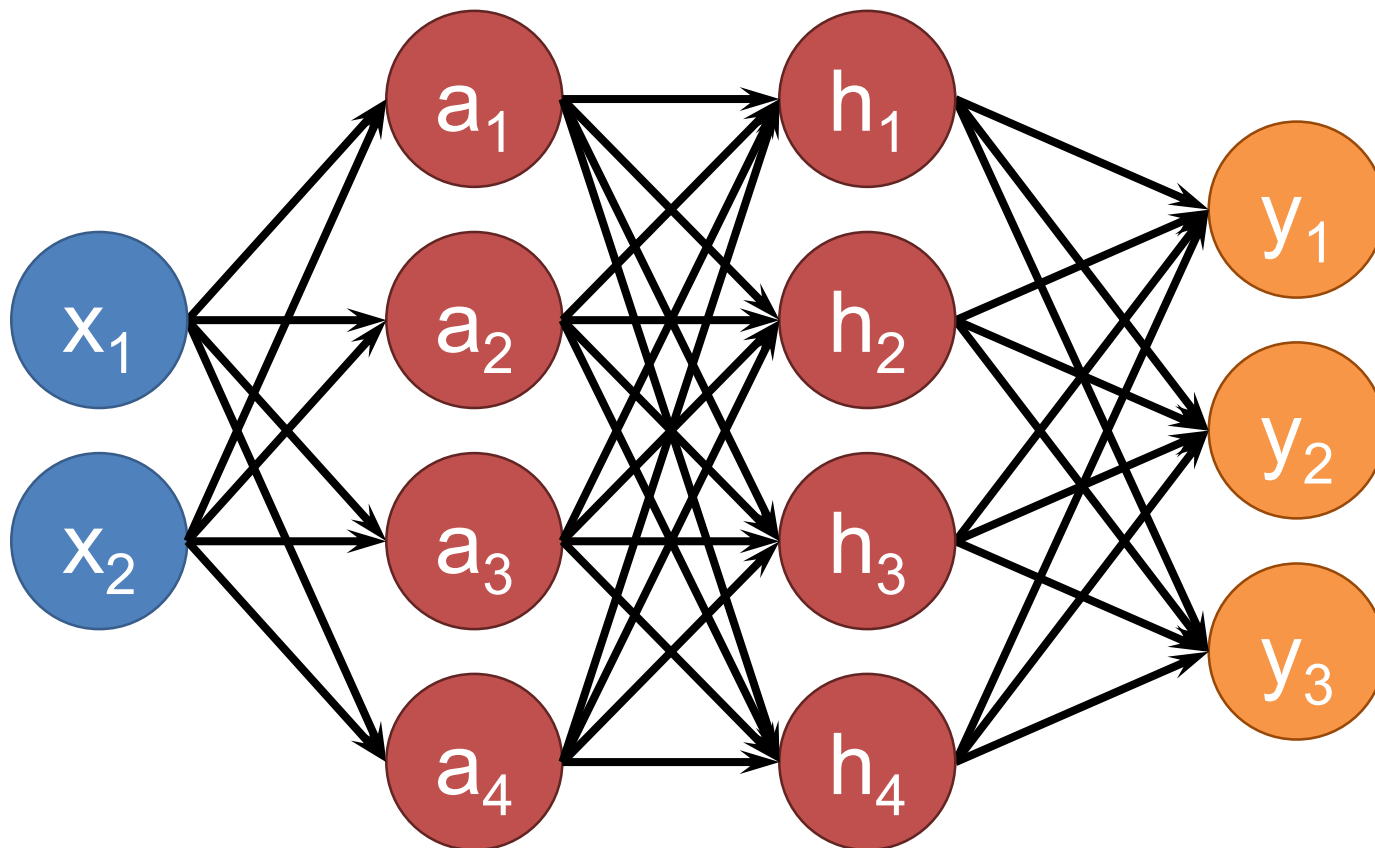
Hidden

Output

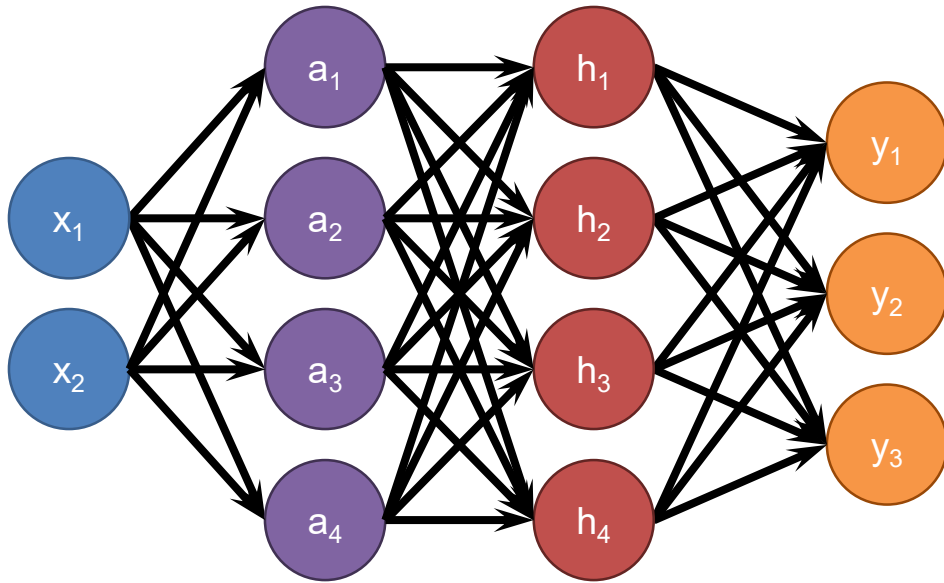


Setting Up A Neural Net

Input Hidden 1 Hidden 2 Output

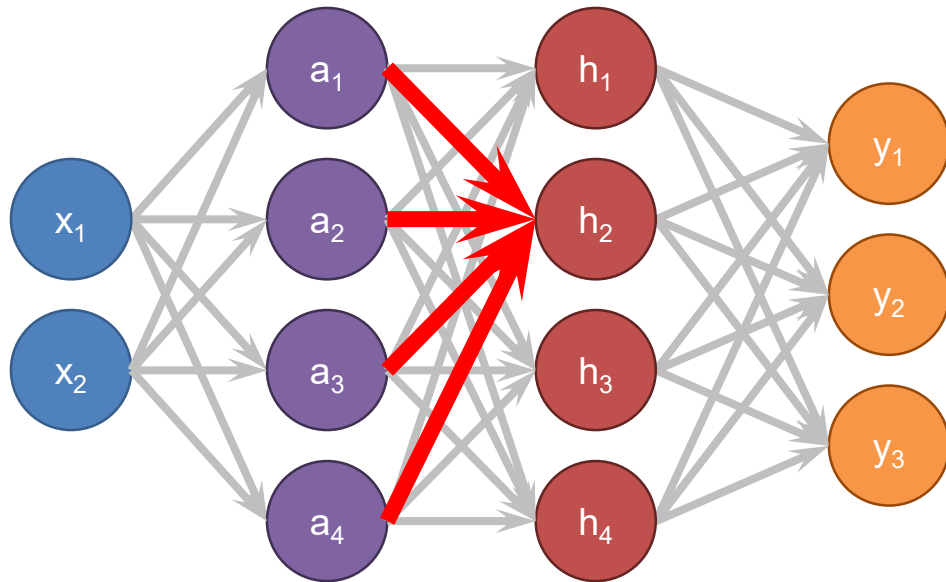


Fully Connected Network



Each neuron connects to each neuron in the previous layer

Fully Connected Network



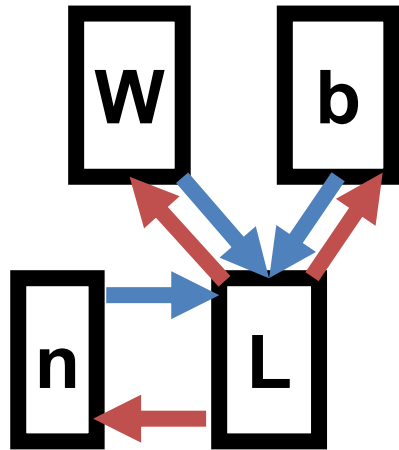
\mathbf{a} All layer a values
 \mathbf{w}_i, b_i Neuron i weights, bias
 f Activation function

$$\begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \end{bmatrix} = f \left(\begin{bmatrix} \leftarrow w_1^T \rightarrow \\ \leftarrow w_2^T \rightarrow \\ \leftarrow w_3^T \rightarrow \\ \leftarrow w_4^T \rightarrow \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \right)$$

Fully Connected Network

Define New Block: “Linear Layer”

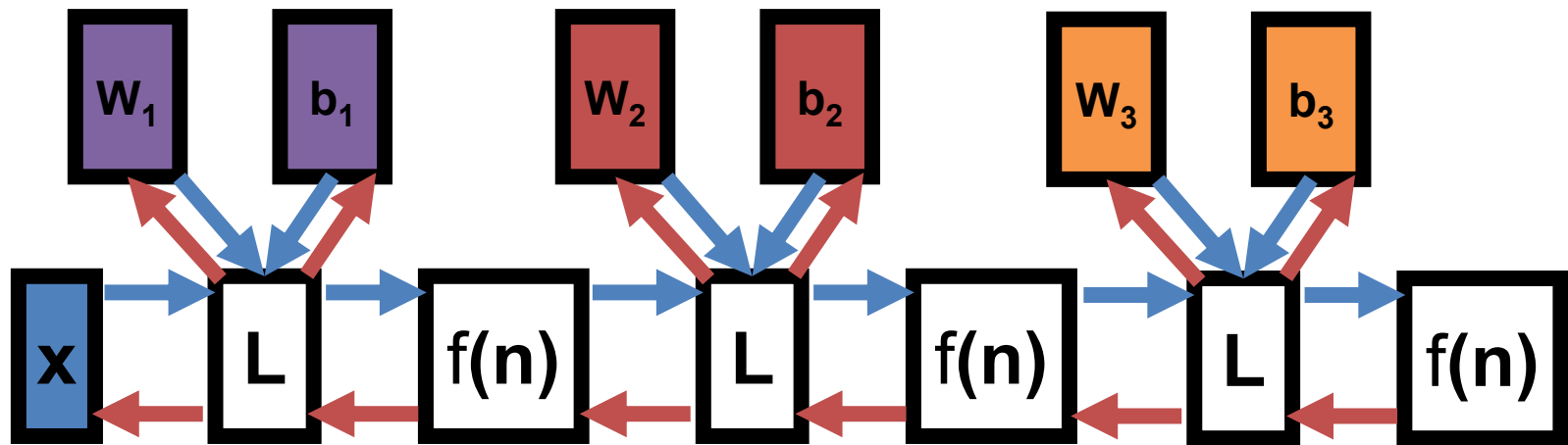
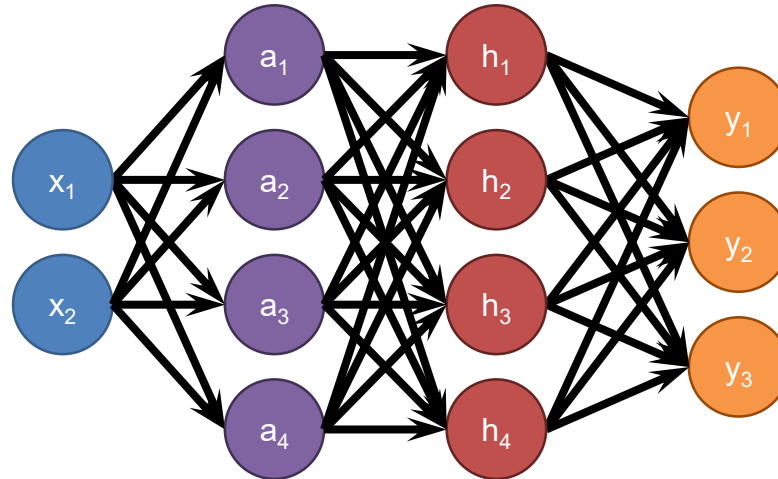
(Ok technically it's Affine)



$$L(\mathbf{n}) = \mathbf{W}\mathbf{n} + \mathbf{b}$$

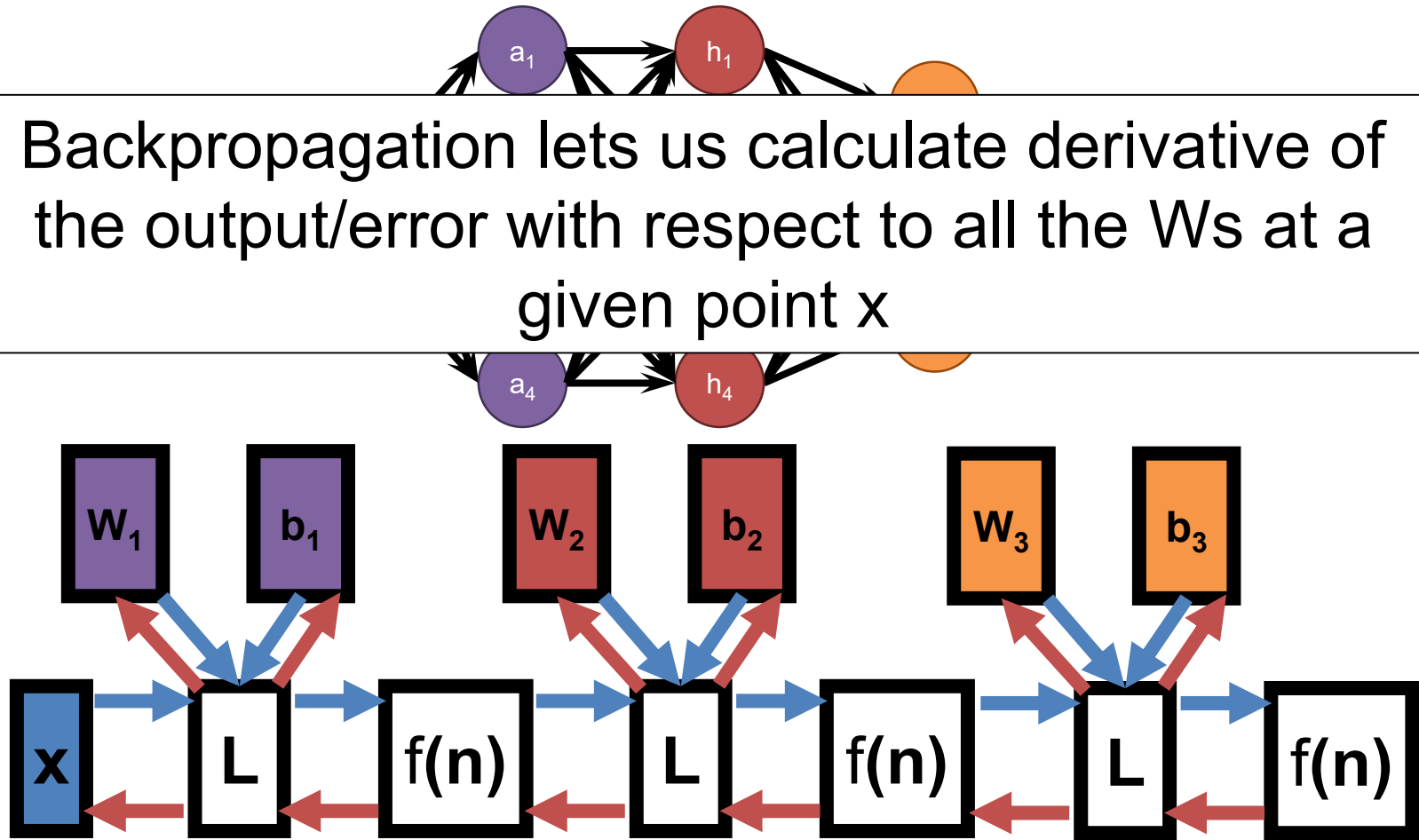
Can get gradient with respect to all the inputs
(do on your own; useful trick: have to be able to do matrix multiply)

Fully Connected Network



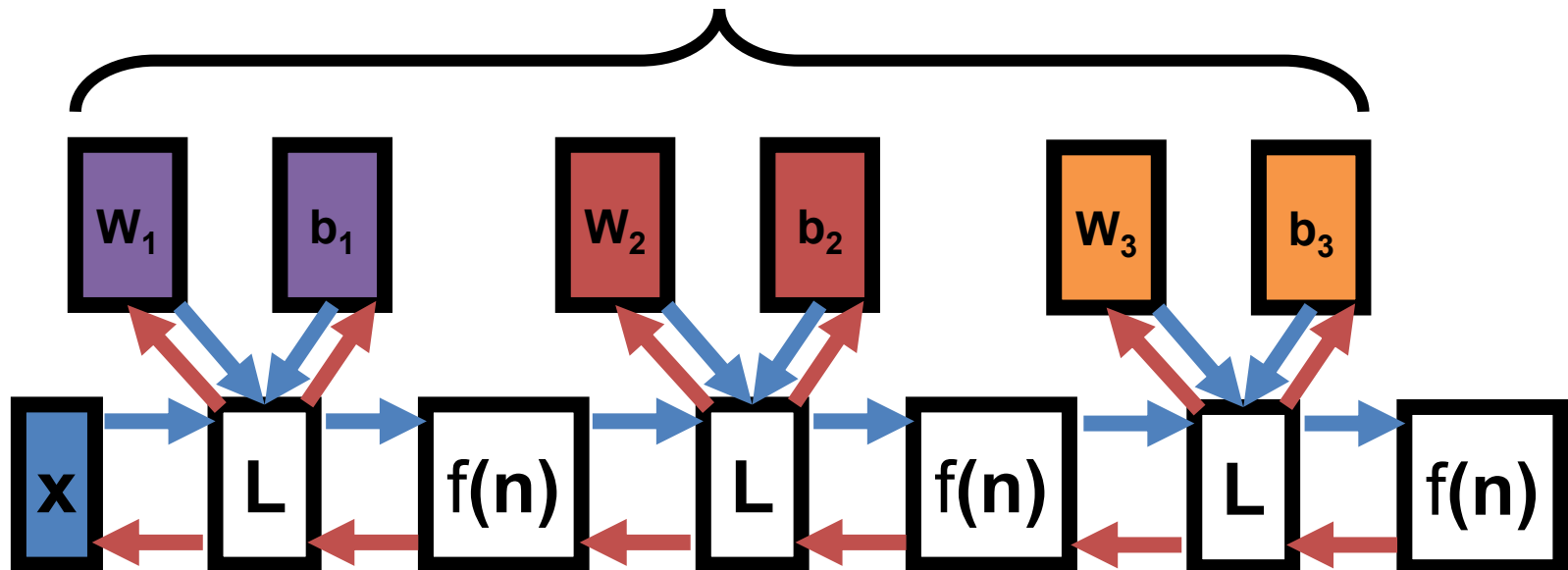
Fully Connected Network

Backpropagation lets us calculate derivative of the output/error with respect to all the W s at a given point x



Putting It All Together – 1

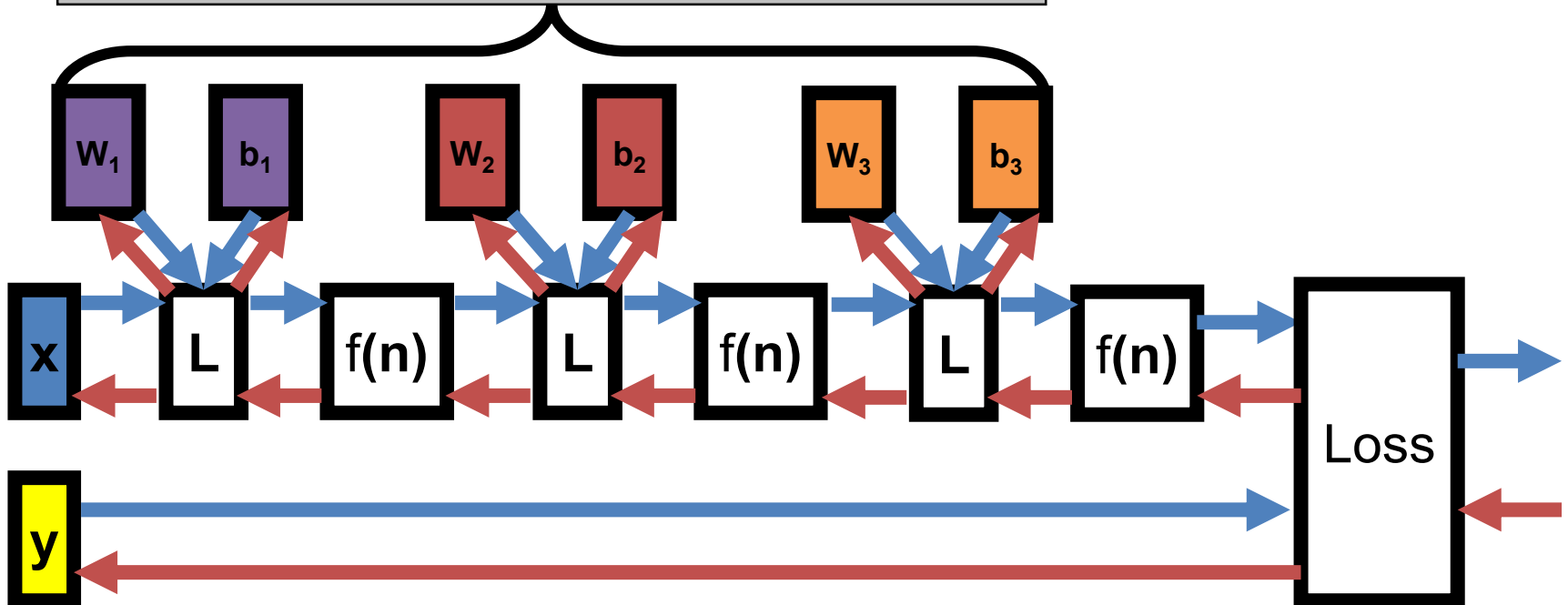
Function: $NN(x; W_i, b_i)$
Parameterized by $W = \{W_i, b_i\}$



Putting It All Together

Function: $\text{Loss}(\text{NN}(x; W_i, b_i), y)$

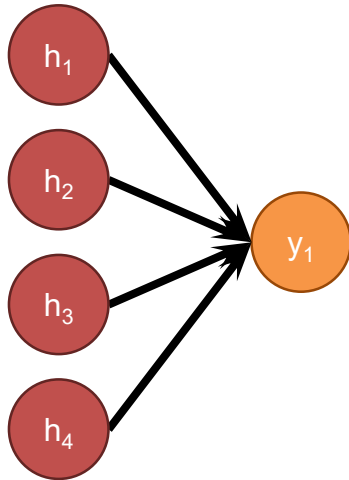
Function: $\text{NN}(x; W_i, b_i)$



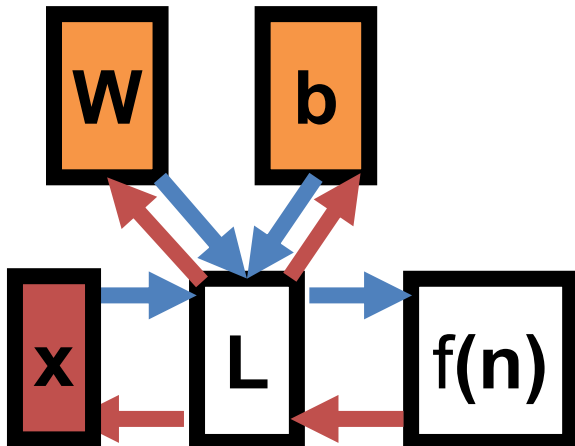
Putting It All Together

```
W = initializeWeights()
for i in range(numIterations):
    #sample a batch
    batch = random.subset(0,#datapoints,K)
    batchX, batchY = dataX[batch], dataY[batch]
    #compute gradient with batch
    gradW = backprop(Loss(NN(batchX,W),batchY))
    #update W with gradient step
    W += -stepsize*gradW
return W
```

What Can We Represent?

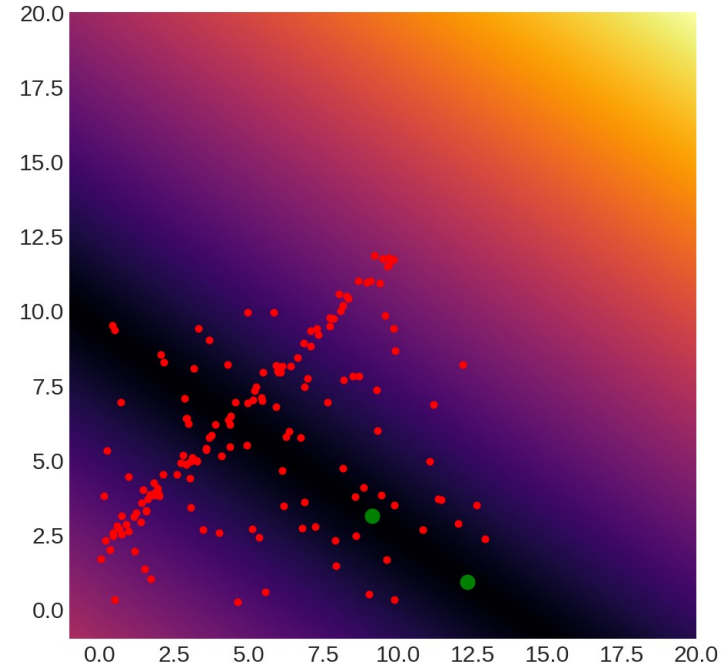
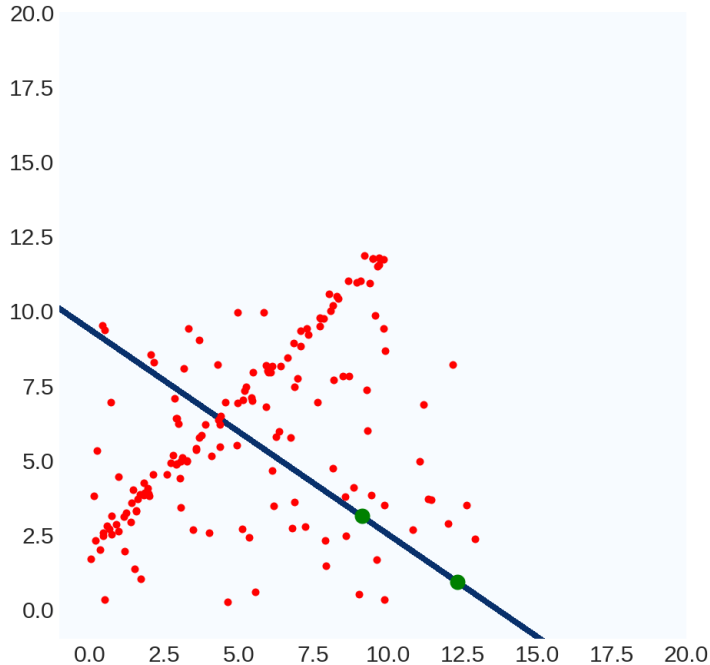


$$L(\mathbf{n}) = \mathbf{W}\mathbf{n} + \mathbf{b}$$

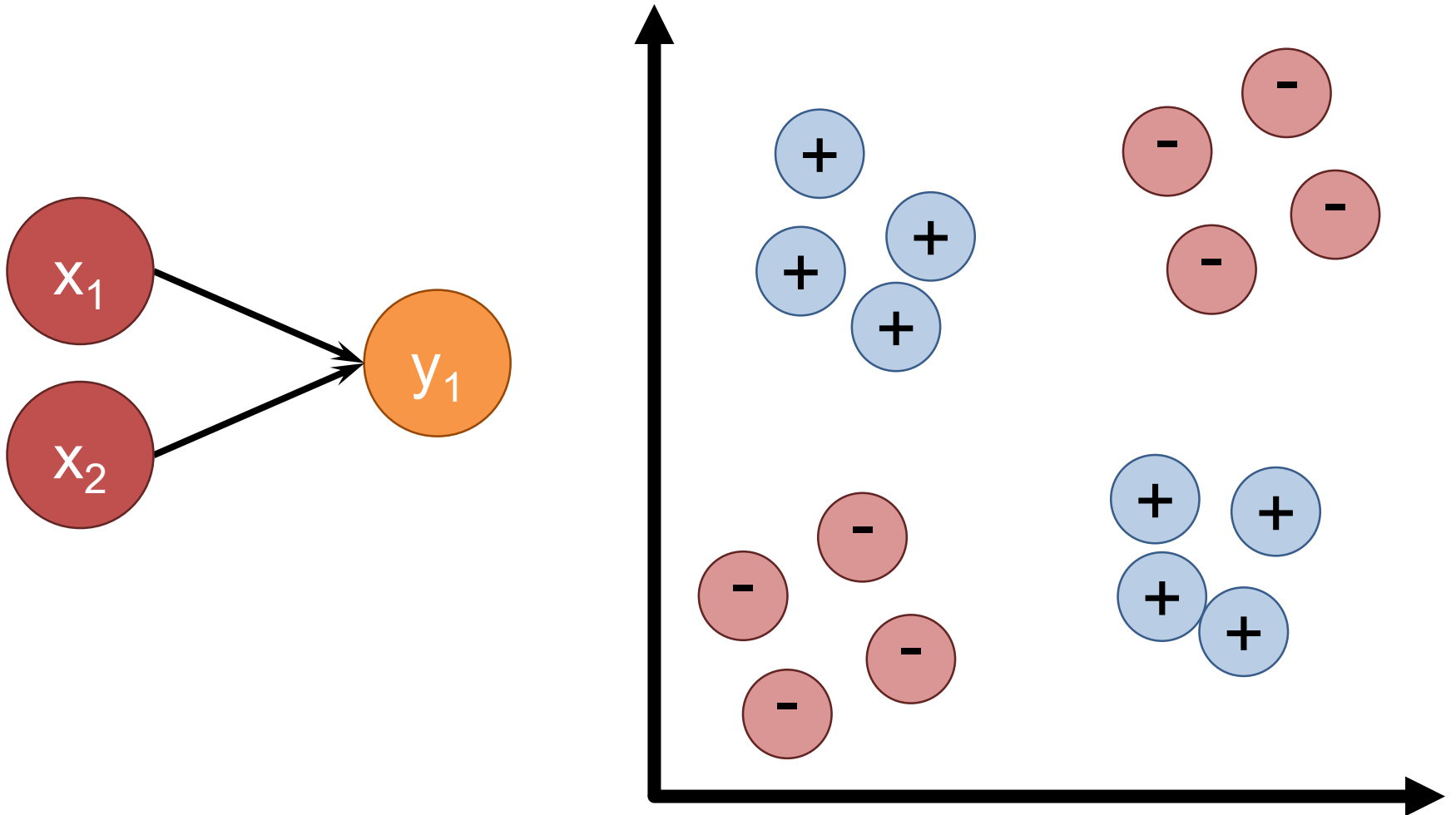


What Can We Represent

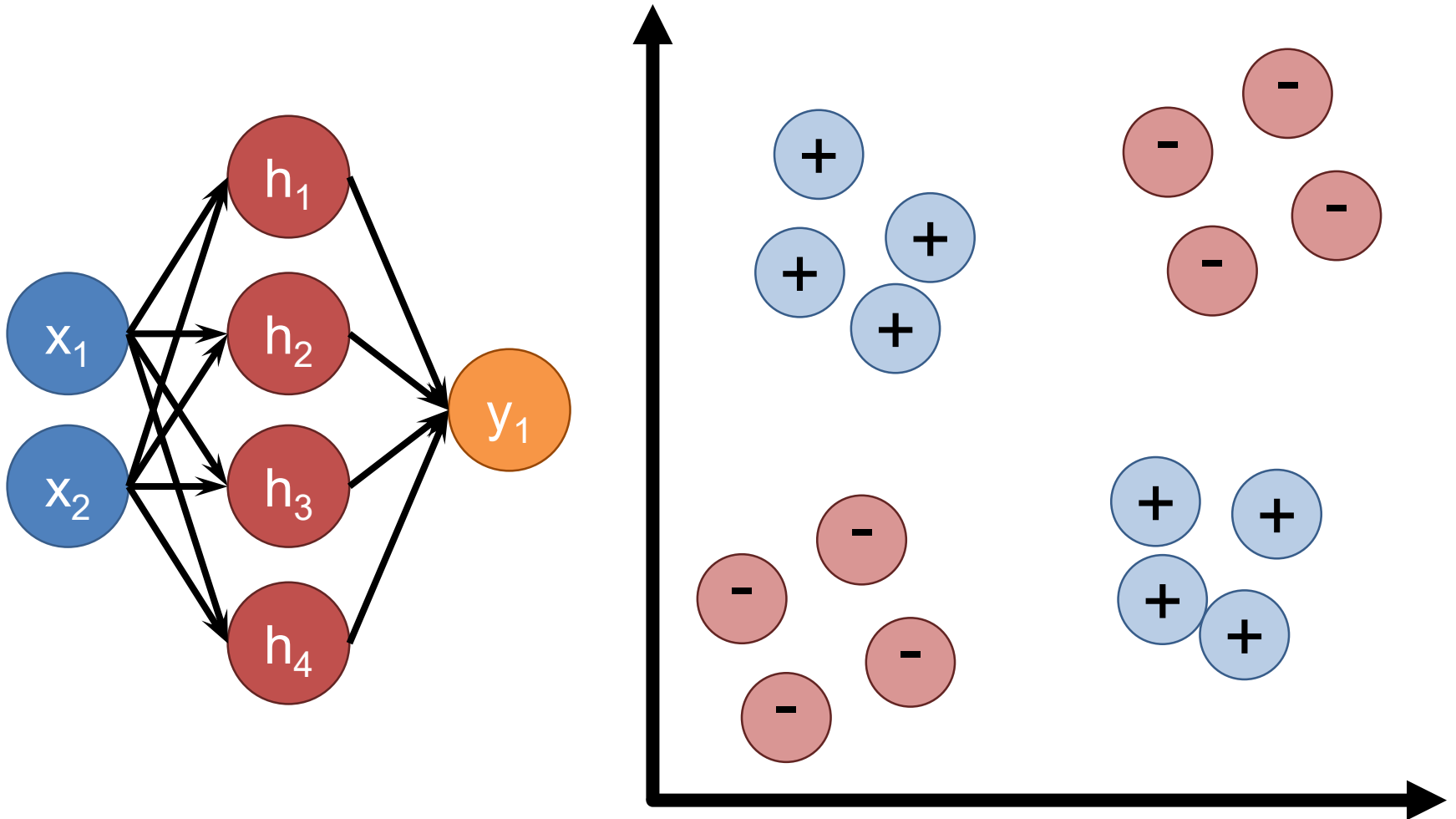
- Recall: $ax+by+z$ is
 - proportional to **signed** distance to line
 - equal to signed distance if you set it right
- Generalization to N-D: hyperplane $\mathbf{w}^T \mathbf{x} + b$



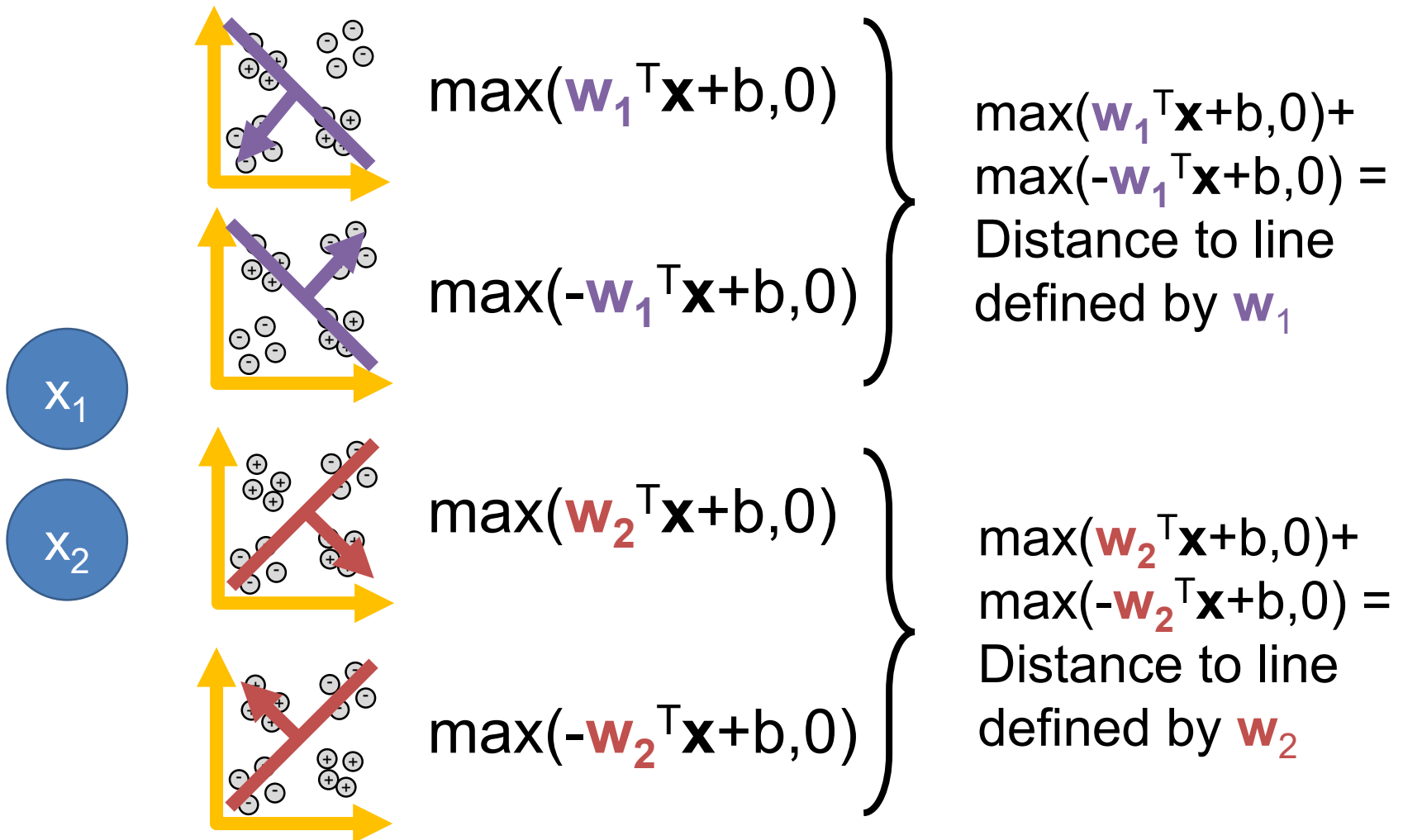
Can We Train a Network To Do It?



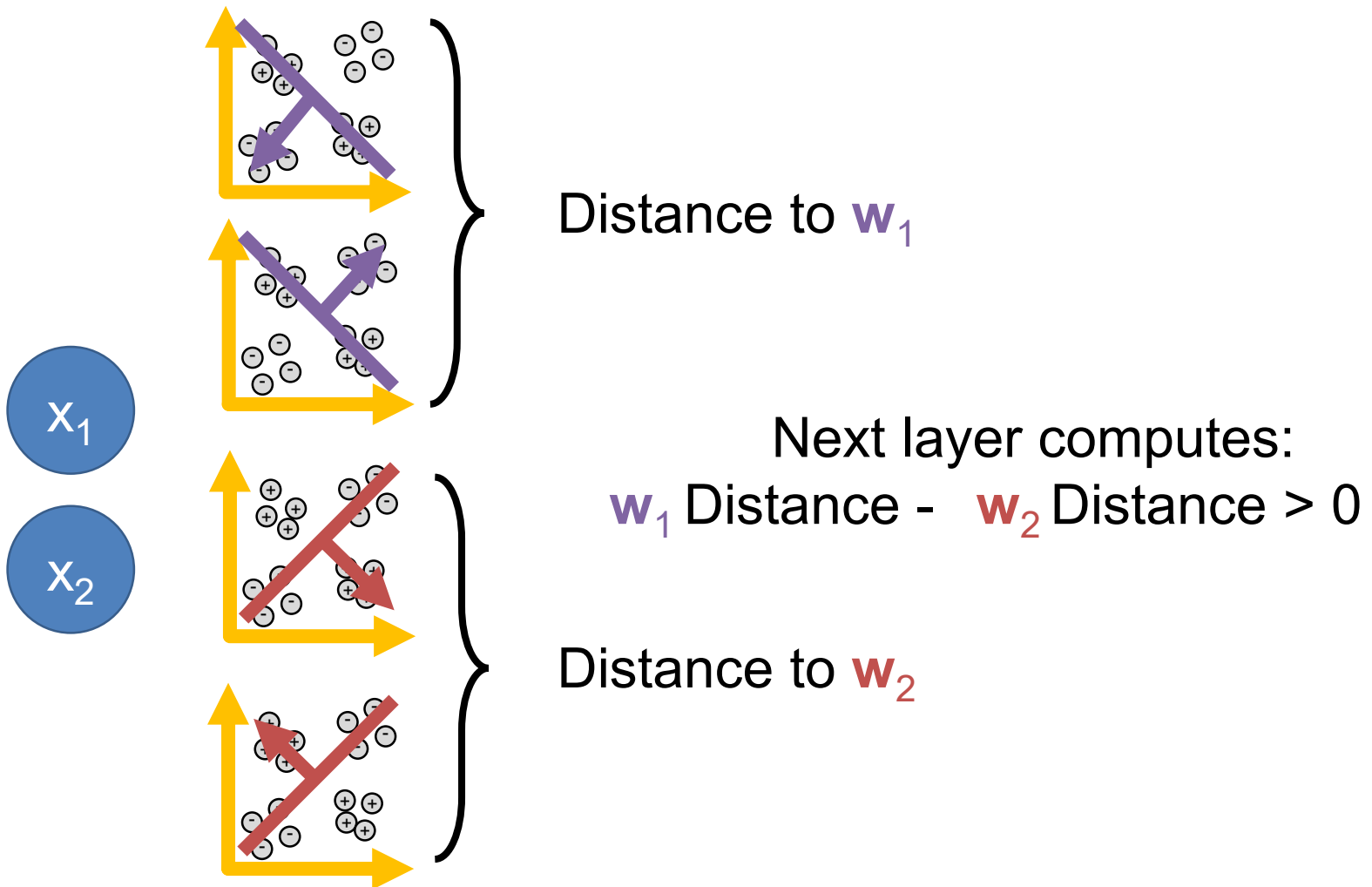
Can We Train a Network To Do It?



Can We Train a Network To Do It?



Can We Train a Network To Do It?



Can We Train a Network To Do It?

Result: feedforward neural networks with a finite number of neurons in a hidden layer can approximate any reasonable* function

Cybenko (1989) for neural networks with sigmoids; Hornik (1991) more generally

In practice, doesn't give a practical guarantee.
Why?

*Continuous, with bounded domain.

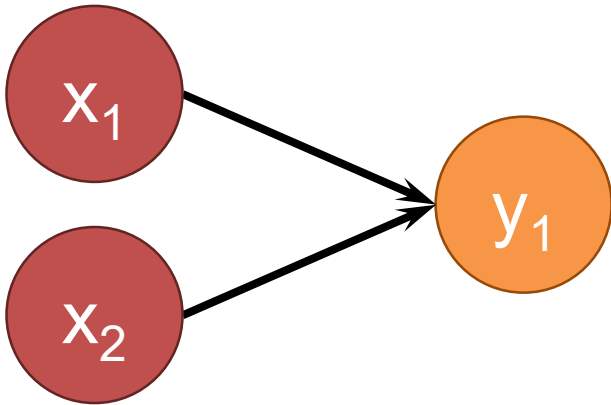
Developing Intuitions

There is no royal road to geometry. – Euclid

- Best way: play with data, be skeptical of everything you do, be skeptical of everything you are told
- Remember: this is linear algebra, not magic
- Common technique: How would you set the weights by hand if you were forced to be a deep net

Parameters

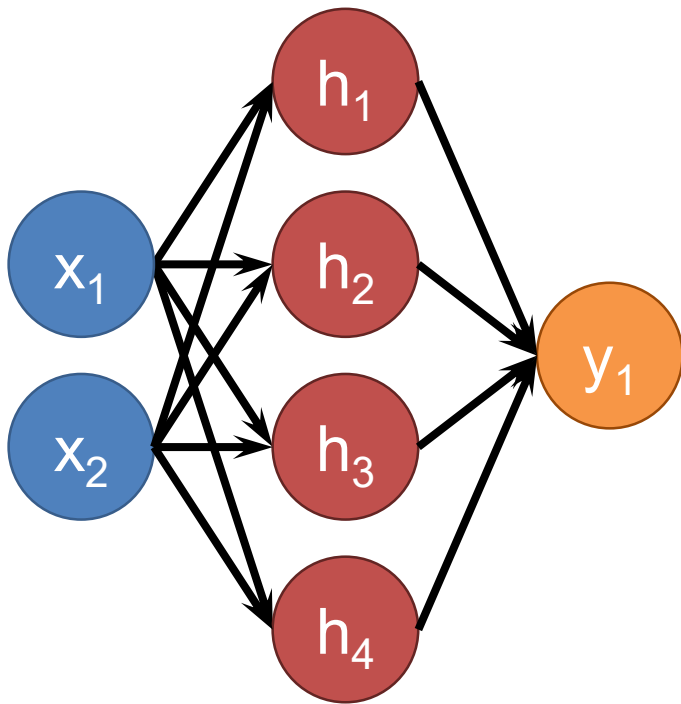
How many parameters does this network have?



Weights: 1x2
Parameters: 3 (bias!)

Parameters

How many parameters does this network have?

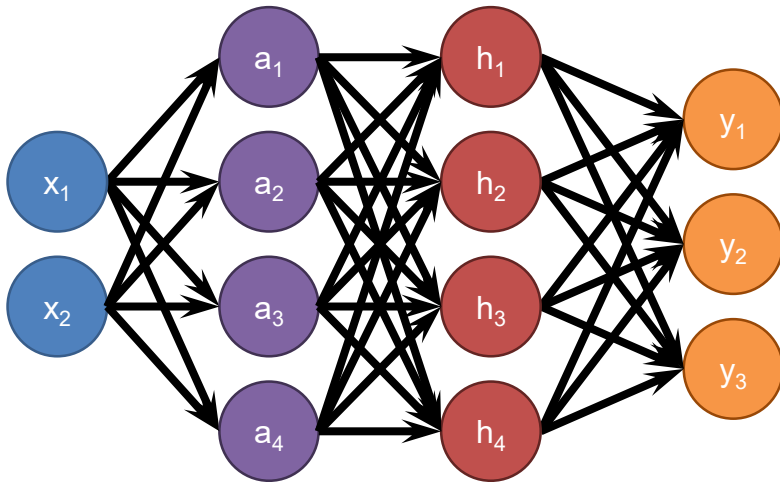


Weights: $1 \times 4 + 4 \times 2 = 12$

Parameters: $12 + 5 = 17$

Parameters

How many parameters does this network have?



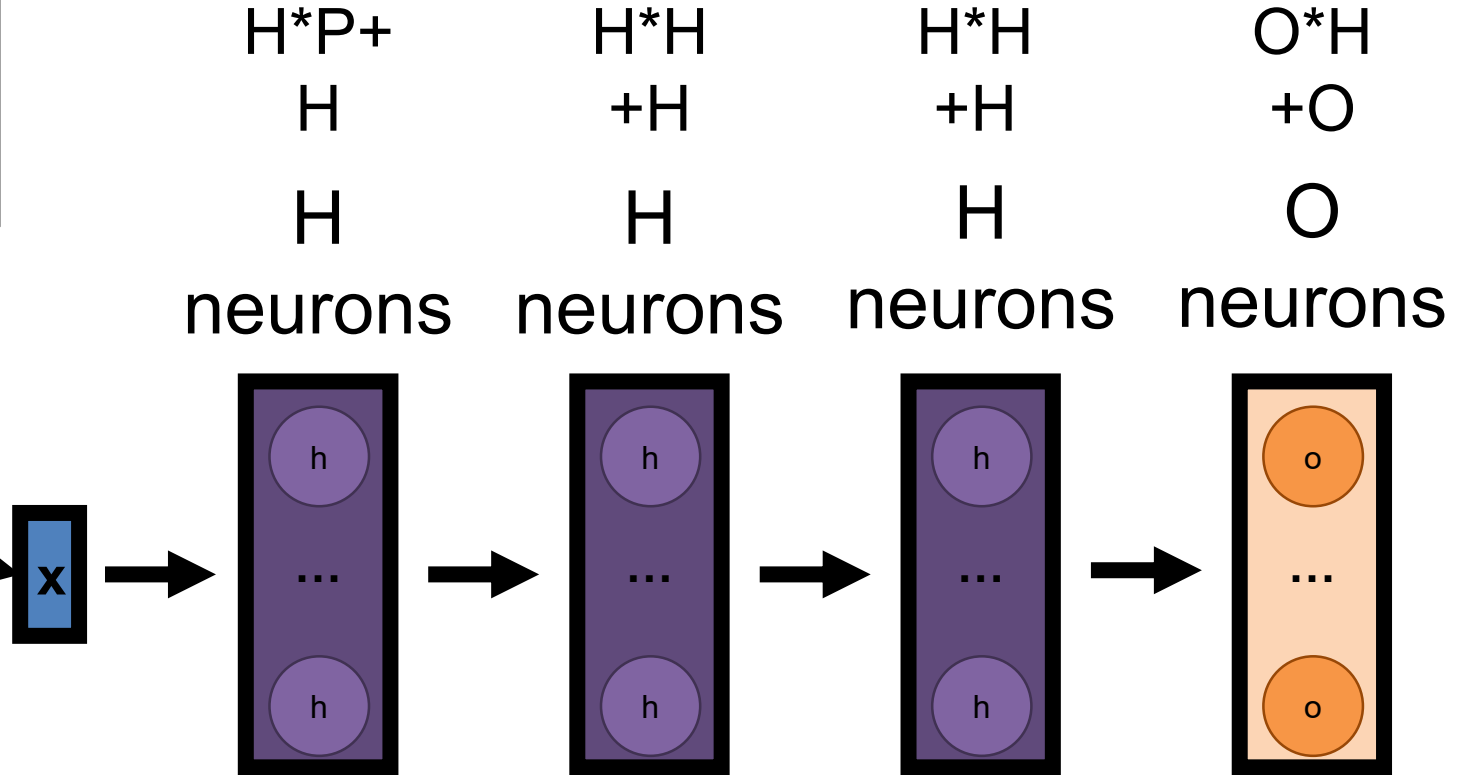
Weights: $3 \times 4 + 4 \times 4 + 4 \times 2 = 36$

Parameters: $36 + 11 = 47$

Parameters

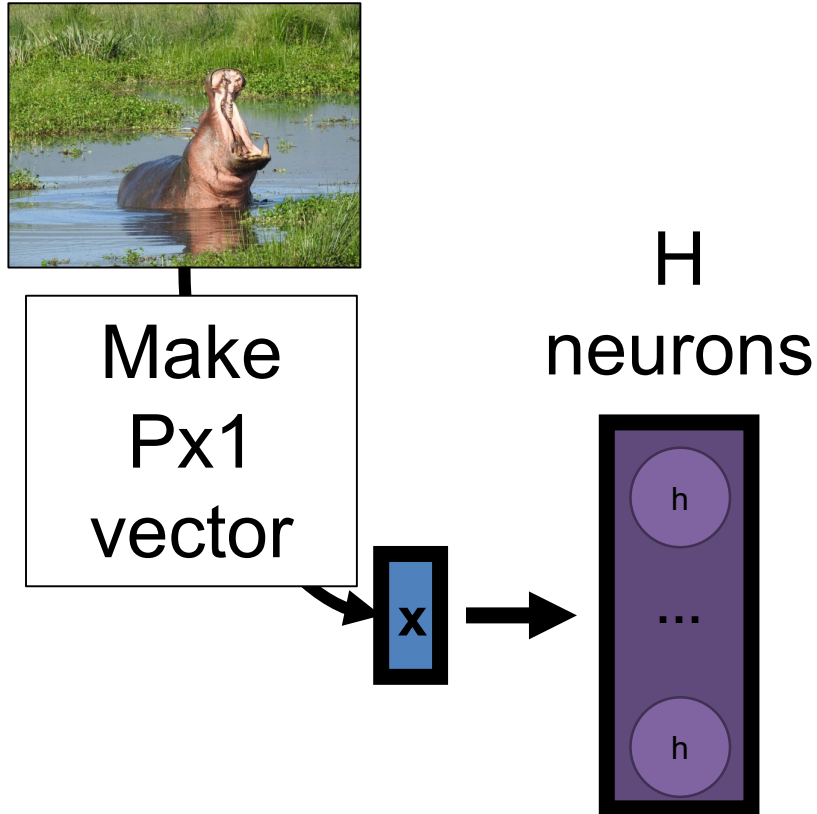


Make
 $P \times 1$
vector



P : 285x350 picture (**terrible!**), H : 1000, O : 3
102 million parameters (400MB)

Parameters

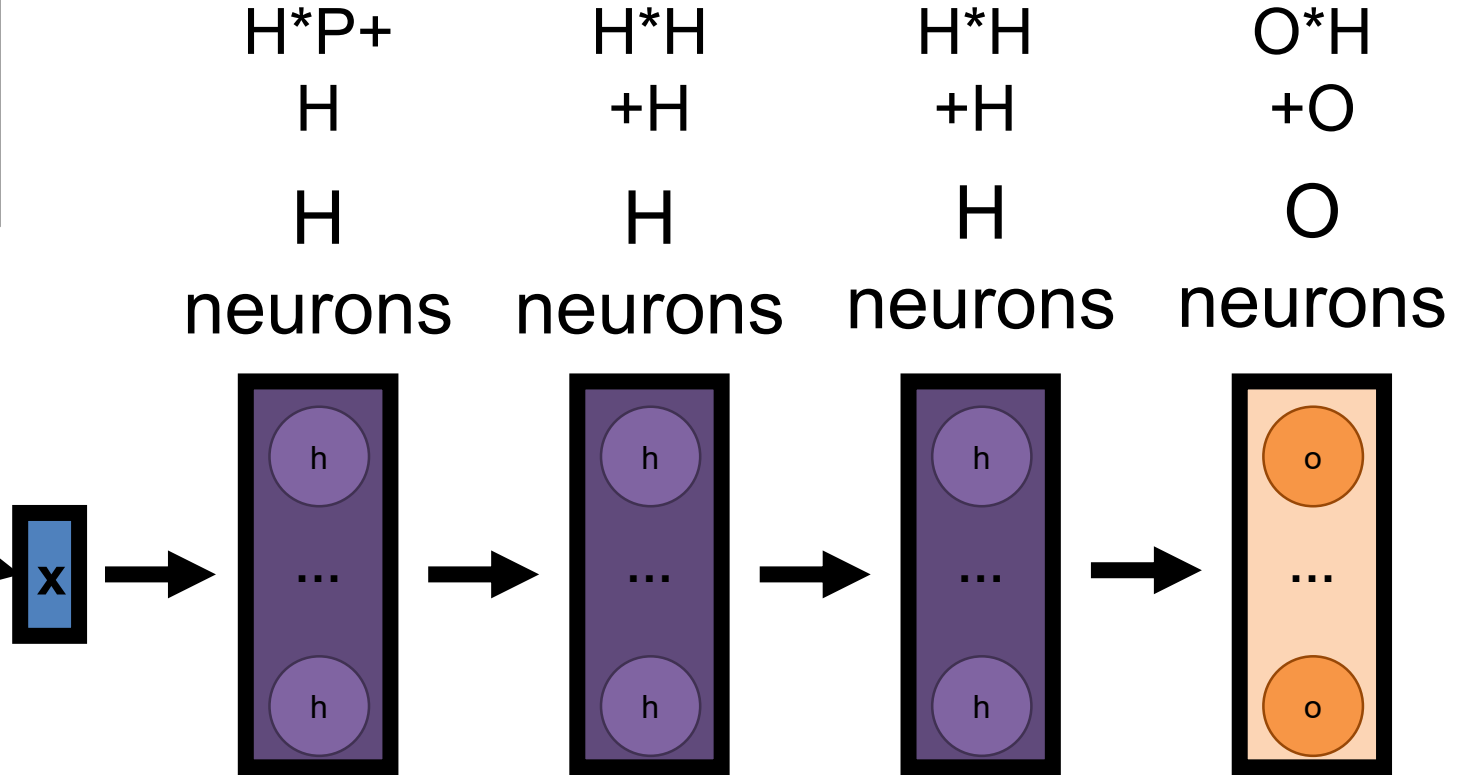


- First layer converts **all** visual information into a single N dimensional vector.
- Suppose you want a neuron to represent dx/dy at each pixel. **How many neurons do you need?**
- **$2P!$**

Parameters



Make
 $P \times 1$
vector



$P: 285 \times 350, H: 2P, O: 3$

100 billion parameters (400GB)

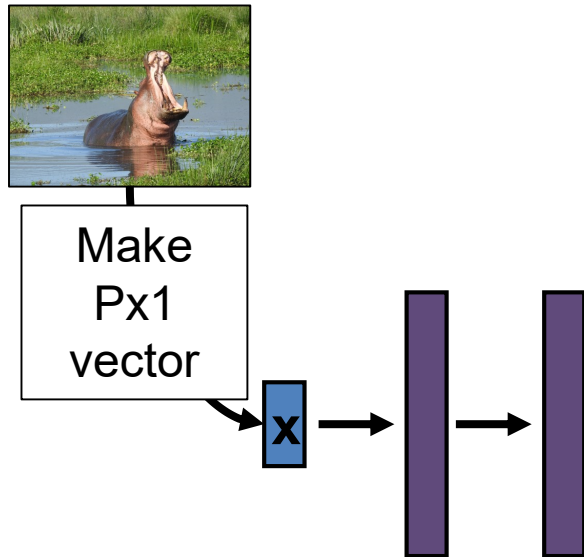
Convnets

Keep Spatial Resolution Around

Neural net:

Data: vector $F \times 1$

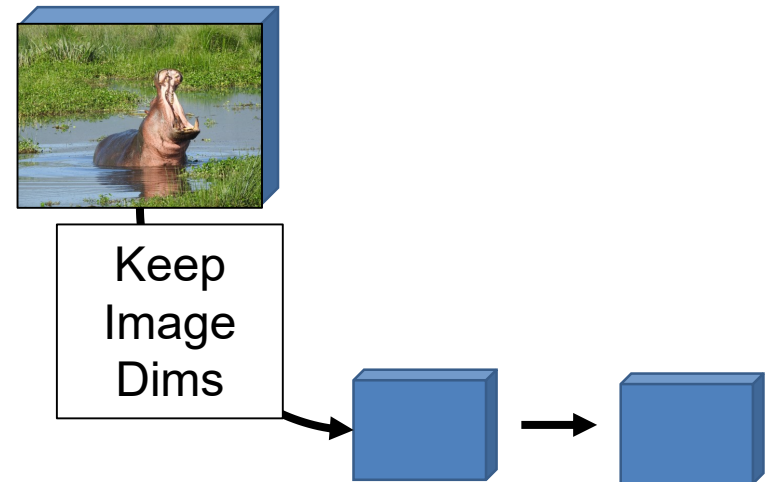
Transform: matrix-multiply



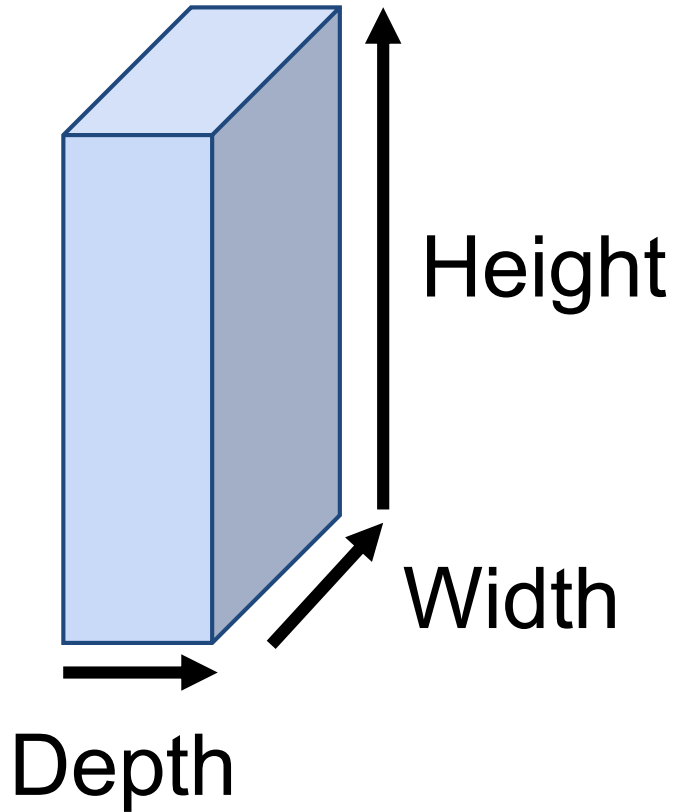
Convnet:

Data: image $H \times W \times F$

Transform: convolution



Convnet



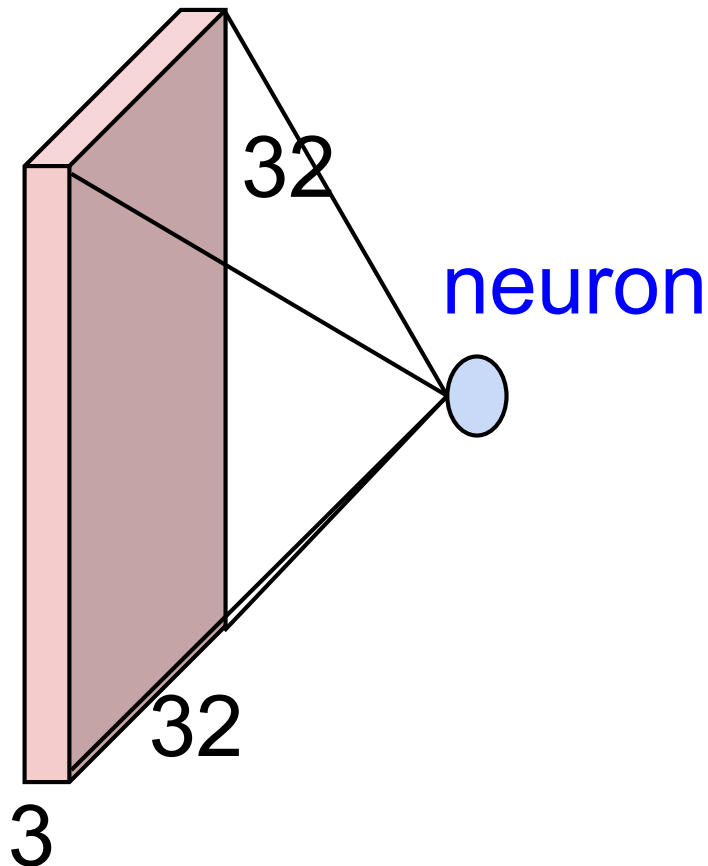
Height: 300
Width: 500
Depth: 3



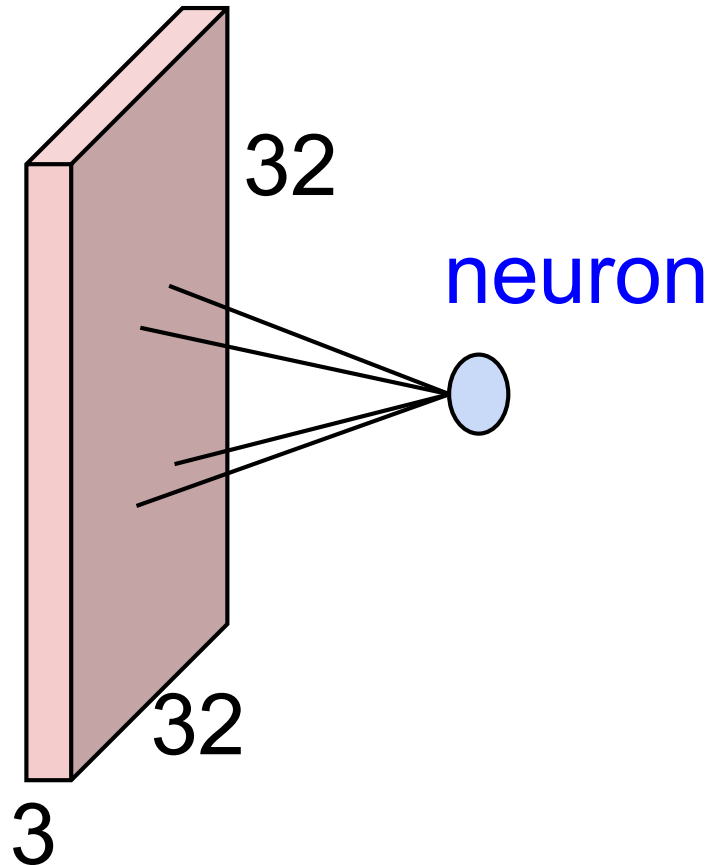
Height: 32
Width: 32
Depth: 3

Convnet

Fully connected:
Connects to everything

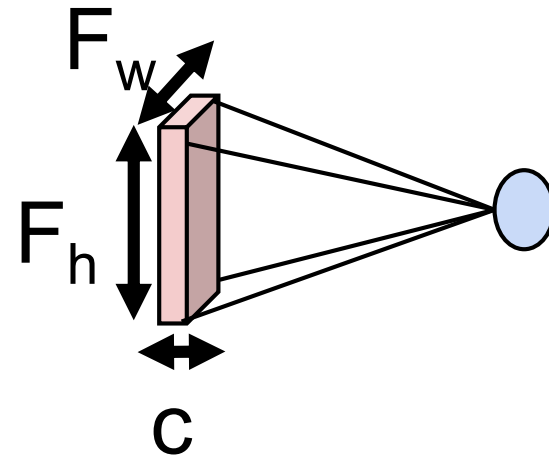
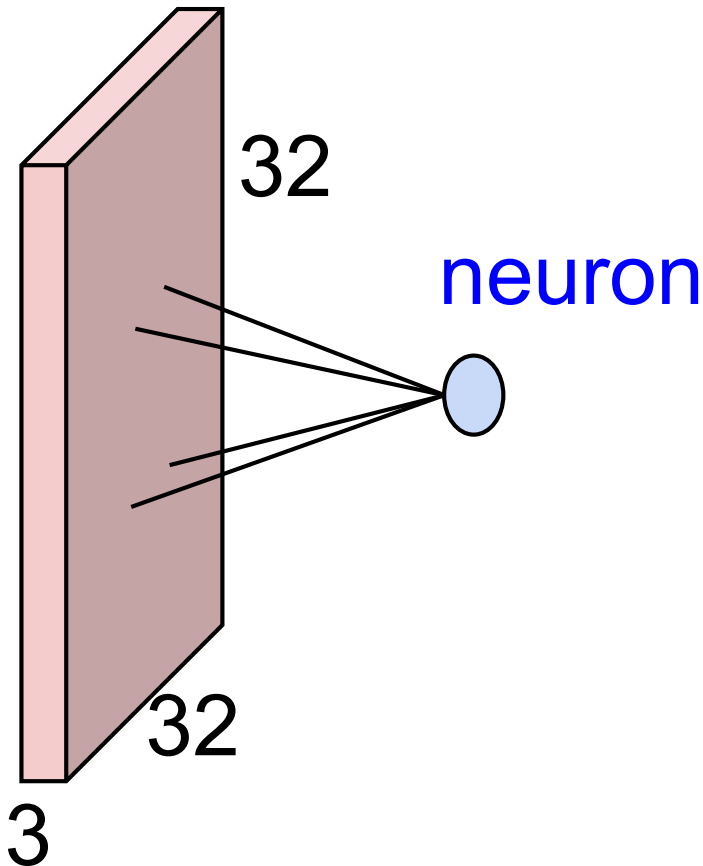


Convnet:
Connects locally



Convnet

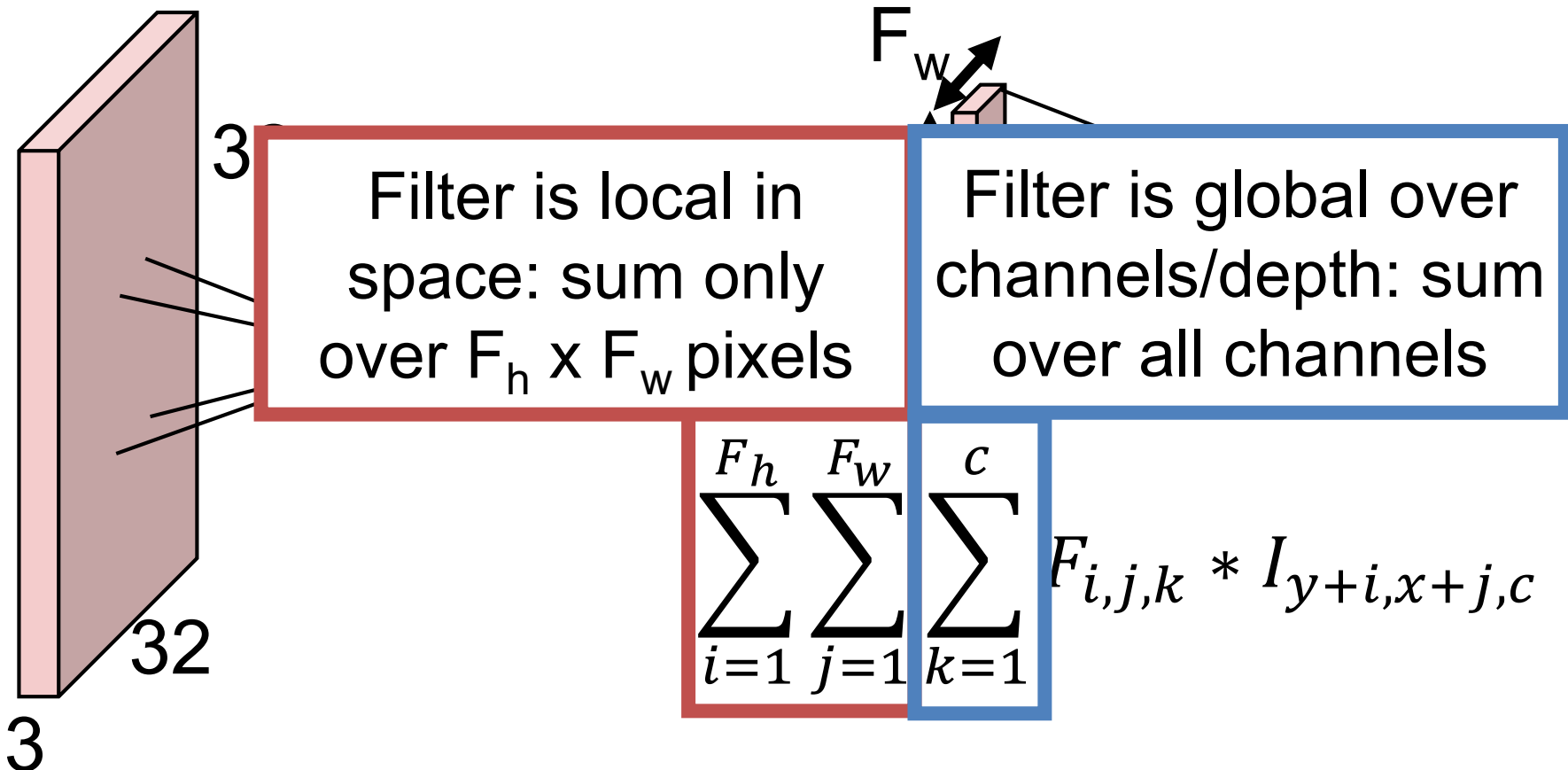
Neuron is the same: weighted linear average



$$\sum_{i=1}^{F_h} \sum_{j=1}^{F_w} \sum_{k=1}^c F_{i,j,k} * I_{y+i,x+j,c}$$

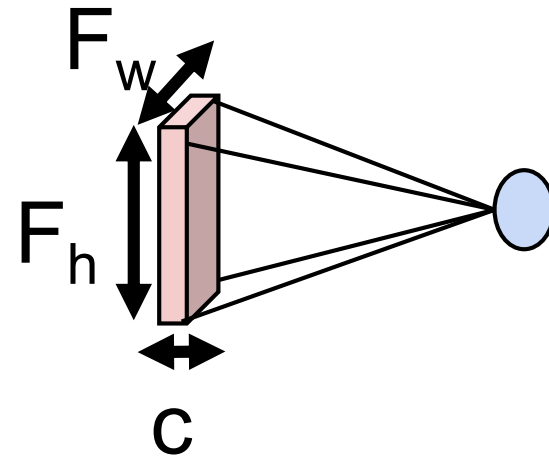
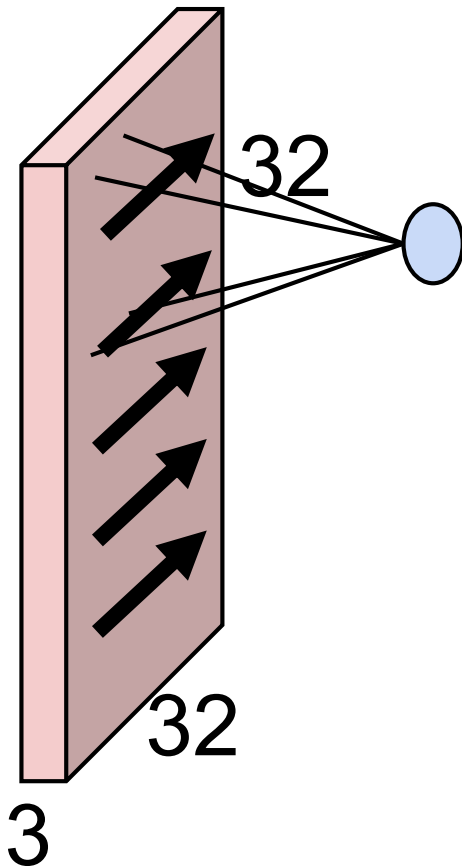
Convnet

Neuron is the same: weighted linear average



Convnet

Get spatial output by sliding filter over image



$$\sum_{i=1}^{F_h} \sum_{j=1}^{F_w} \sum_{k=1}^c F_{i,j,k} * I_{y+i,x+j,c}$$

Differences From Lecture 4 Filtering

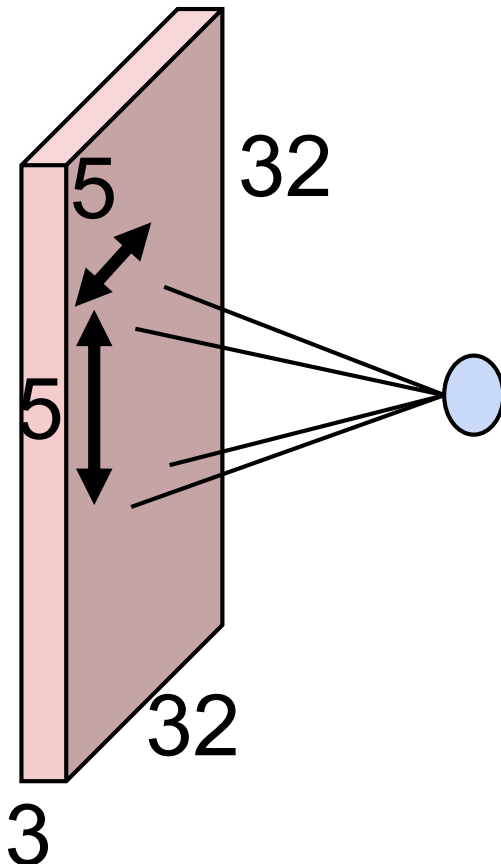
- (a) #input channels can be greater than one
- (b) forget you learned the difference between convolution and cross-correlation

I11	F11	F12	F13	I15	I16
I21	F21	F22	F23	I25	I26
I31	F31	F32	F33	I35	I36
I41	I42	I43	I44	I45	I46
I51	I52	I53	I54	I55	I56

$$\begin{aligned} & \text{Output}[1,2] \\ &= I[1,2]*F[1,1] + I[1,3]*F[1,2] \\ & \quad + \dots + I[3,4]*F[3,3] \end{aligned}$$

Convnet

How big is the output?



Height? $32-5+1=28$

Width? $32-5+1=28$

Channels? 1

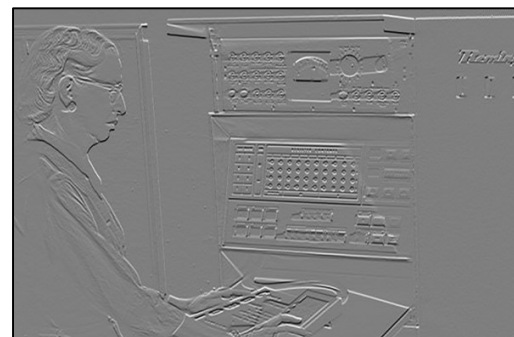
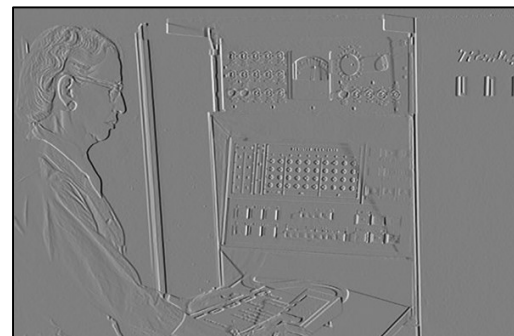
One filter not very
useful by itself

Multiple Filters

You've already seen this before

Input:
400x600x1

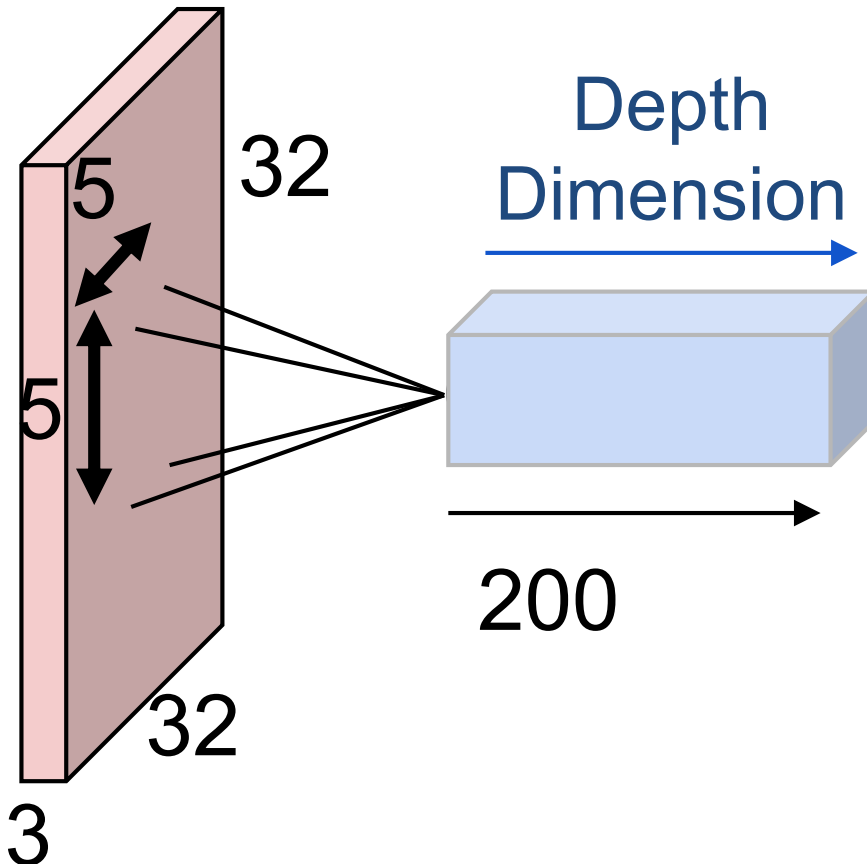
Output:
400x600x2



Convnet

Multiple out channels via multiple filters.

How big is the output?



Height? $32-5+1=28$

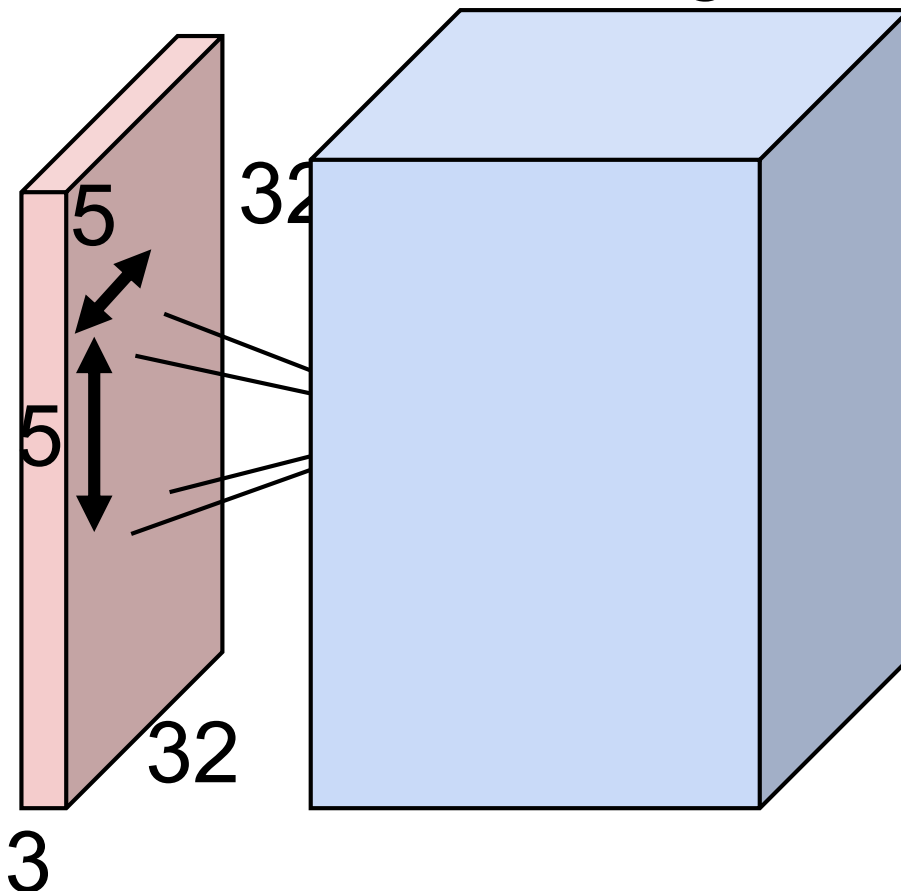
Width? $32-5+1=28$

Channels? 200

Convnet

Multiple out channels via multiple filters.

How big is the output?



Height? $32 - 5 + 1 = 28$

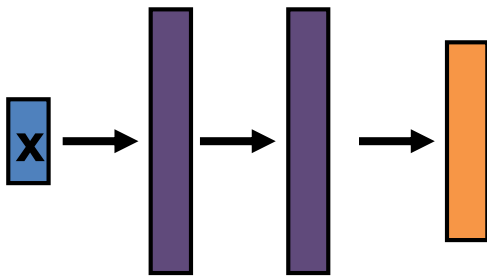
Width? $32 - 5 + 1 = 28$

Channels? 200

Convnet, Summarized

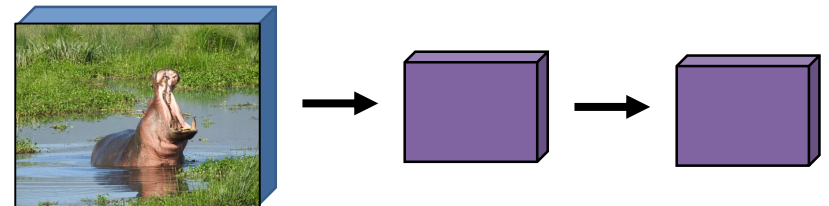
Neural net:

series of matrix-multiplies
parameterized by \mathbf{W}, \mathbf{b} +
nonlinearity/activation
Fit by gradient descent



Convnet:

series of convolutions
parameterized by \mathbf{F}, \mathbf{b} +
nonlinearity/activation
Fit by gradient descent



One Additional Subtlety – Stride

Warmup: how big is the output spatially?

F11	F12	F13	I14	I15	I16	I17
F21	F22	F23	I24	I25	I26	I27
F31	F32	F33	I34	I35	I36	I37
I41	I42	I43	I44	I45	I46	I47
I51	I52	I53	I54	I55	I56	I57
I61	I62	I63	I64	I65	I66	I67
I71	I72	I73	I74	I75	I76	I77

Normal (Stride 1):
5x5 output

One Additional Subtlety – Stride

Stride: skip a few (here 2)

F11	F12	F13	I14	I15	I16	I17
F21	F22	F23	I24	I25	I26	I27
F31	F32	F33	I34	I35	I36	I37
I41	I42	I43	I44	I45	I46	I47
I51	I52	I53	I54	I55	I56	I57
I61	I62	I63	I64	I65	I66	I67
I71	I72	I73	I74	I75	I76	I77

Normal (Stride 1):
5x5 output

One Additional Subtlety – Stride

Stride: skip a few (here 2)

I11	I12	F11	F12	F13	I16	I17
I21	I22	F21	F22	F23	I26	I27
I31	I32	F31	F32	F33	I36	I37
I41	I42	I43	I44	I45	I46	I47
I51	I52	I53	I54	I55	I56	I57
I61	I62	I63	I64	I65	I66	I67
I71	I72	I73	I74	I75	I76	I77

Normal (Stride 1):
5x5 output

One Additional Subtlety – Stride

Stride: skip a few (here 2)

I11	I12	I13	I14	F11	F12	F13
I21	I22	I23	I24	F21	F22	F23
I31	I32	I33	I34	F31	F32	F33
I41	I42	I43	I44	I45	I46	I47
I51	I52	I53	I54	I55	I56	I57
I61	I62	I63	I64	I65	I66	I67
I71	I72	I73	I74	I75	I76	I77

Normal (Stride 1):
5x5 output

Stride 2 convolution:
3x3 output

One Additional Subtlety – Stride

What about stride 3?

F11	F12	F13	I14	I15	I16	I17
F21	F22	F23	I24	I25	I26	I27
F31	F32	F33	I34	I35	I36	I37
I41	I42	I43	I44	I45	I46	I47
I51	I52	I53	I54	I55	I56	I57
I61	I62	I63	I64	I65	I66	I67
I71	I72	I73	I74	I75	I76	I77

Normal (Stride 1):
5x5 output

Stride 2 convolution:
3x3 output

One Additional Subtlety – Stride

What about stride 3?

I11	I12	I13	F11	F12	F13	I17
I21	I22	I23	F21	F22	F23	I27
I31	I32	I33	F31	F32	F33	I37
I41	I42	I43	I44	I45	I46	I47
I51	I52	I53	I54	I55	I56	I57
I61	I62	I63	I64	I65	I66	I67
I71	I72	I73	I74	I75	I76	I77

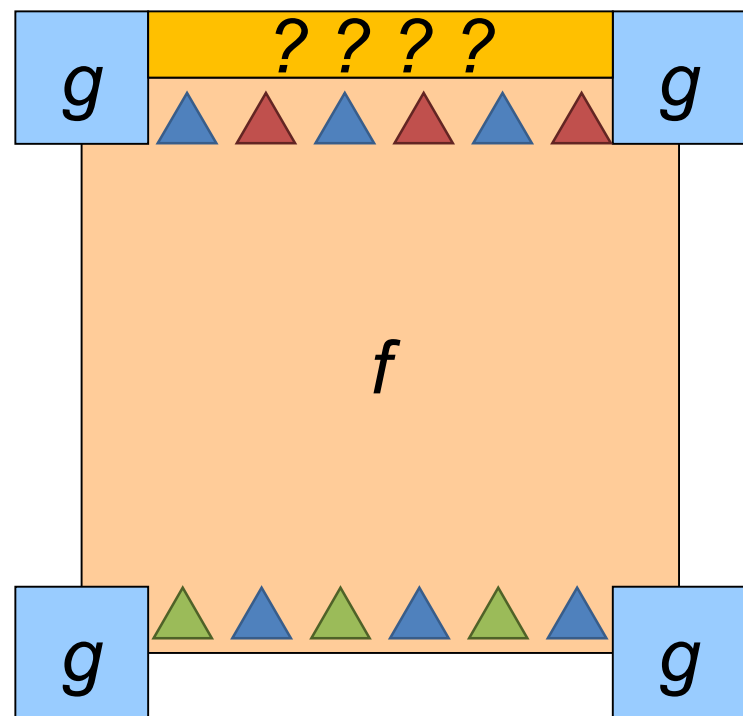
Normal (Stride 1):
5x5 output

Stride 2 convolution:
3x3 output

Stride 3 convolution:
Doesn't work!

One Additional Subtlety

Zero padding is extremely common, although other forms of padding do happen



Symm: fold sides over



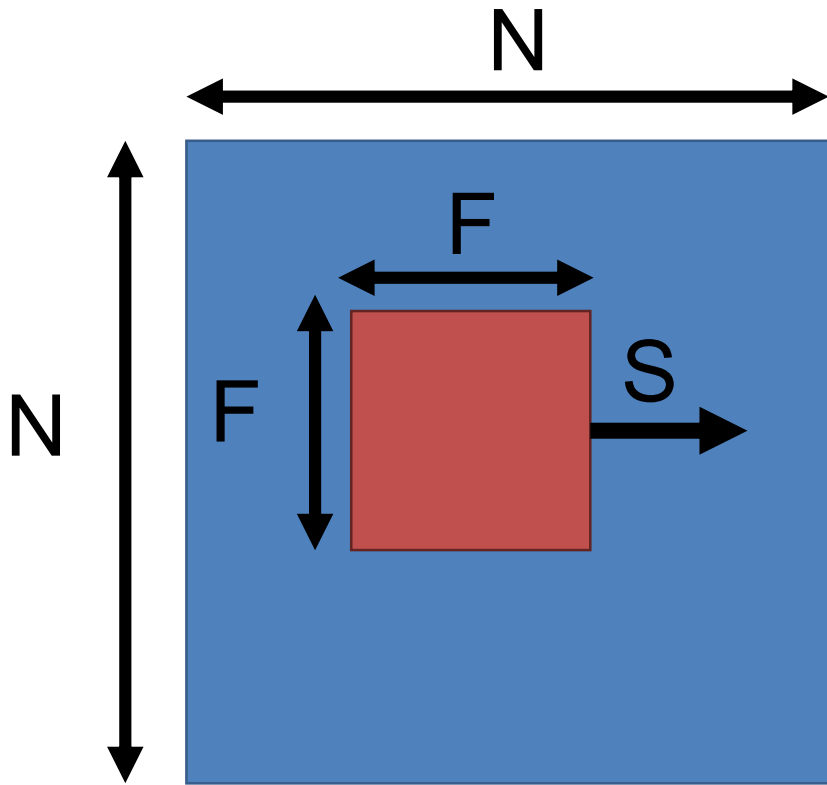
Circular/Wrap: wrap around



pad/fill: add value, often 0



In General



Output Size

$$\frac{(N - F)}{S} + 1$$

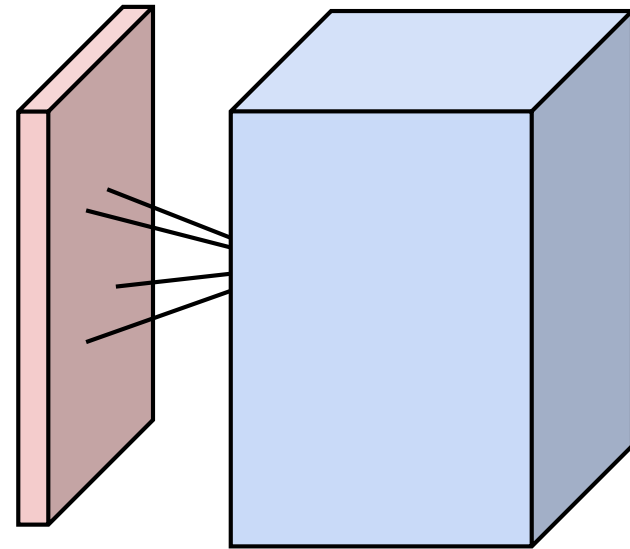
More Examples

Input volume: **32x32x3**

Receptive fields: **5x5**, **stride 1**

Number of neurons: **5**

$$\frac{(N - F)}{s} + 1$$



Output volume size?

More Examples

Input volume: **32x32x3**

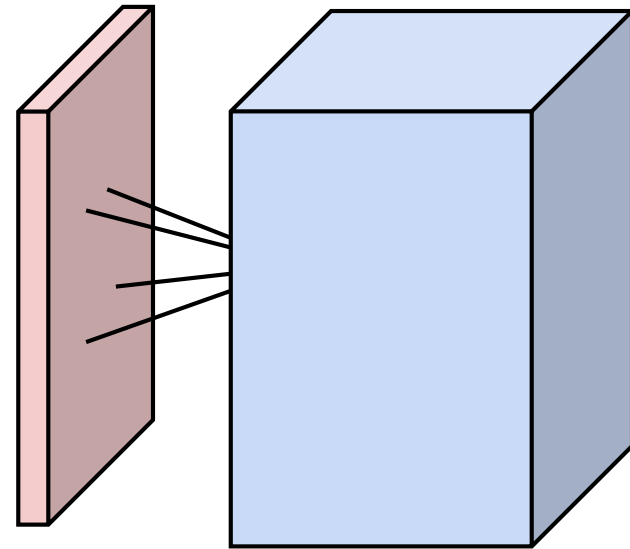
Receptive fields: **5x5**, **stride 1**

Number of neurons: **5**

$$\frac{(N - F)}{s} + 1$$

Output volume: $(32 - 5) / 1 + 1 = 28$, so: **28x28x5**

Number of Parameters?



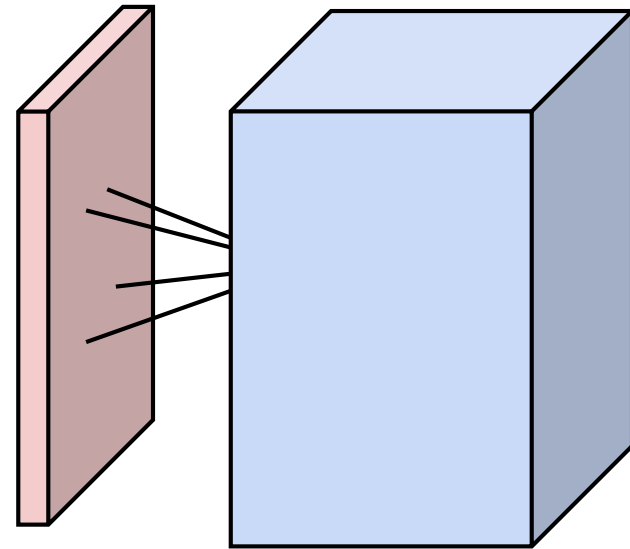
More Examples

Input volume: **32x32x3**

Receptive fields: **5x5**, **stride 1**

Number of neurons: **5**

$$\frac{(N - F)}{s} + 1$$



Output volume: $(32 - 5) / 1 + 1 = 28$, so: **28x28x5**

How many parameters? **5x5x3x5 + 5 = 380**

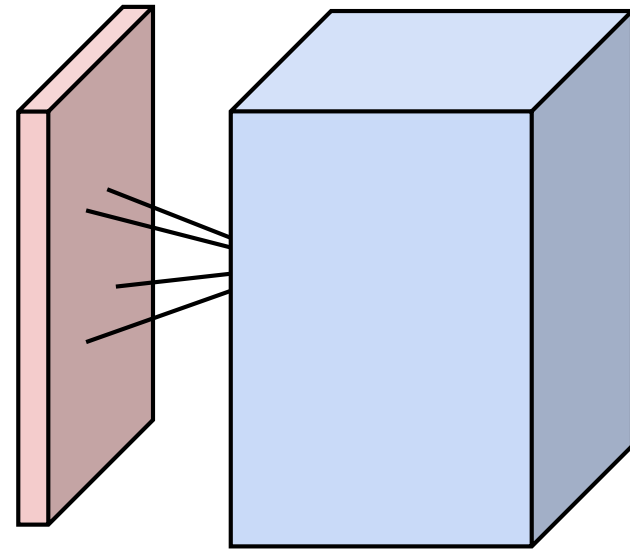
More Examples

Input volume: **32x32x3**

Receptive fields: **5x5**, **stride 3**

Number of neurons: **5**

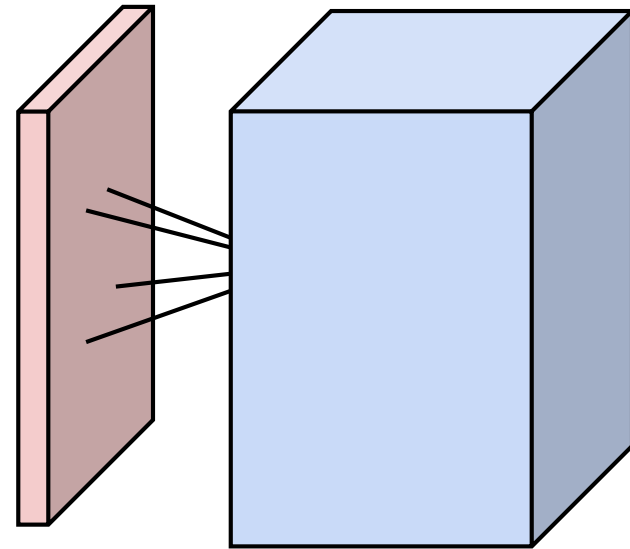
$$\frac{(N - F)}{s} + 1$$



Output volume size?

More Examples

Input volume: **32x32x3**
Receptive fields: **5x5**, **stride 3**
Number of neurons: **5**



Output volume: $(32 - 5) / 3 + 1 = 10$, so: **10x10x5**

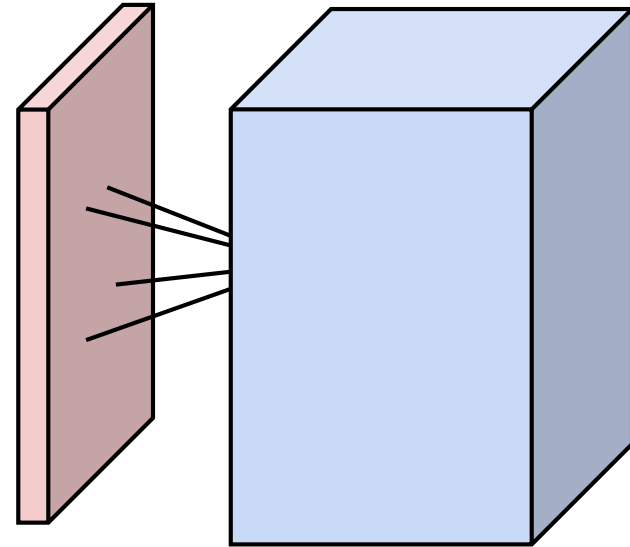
Number of Parameters?

More Examples

Input volume: **32x32x3**

Receptive fields: **5x5**, **stride 3**

Number of neurons: **5**



Output volume: $(32 - 5) / 3 + 1 = 10$, so: **10x10x5**

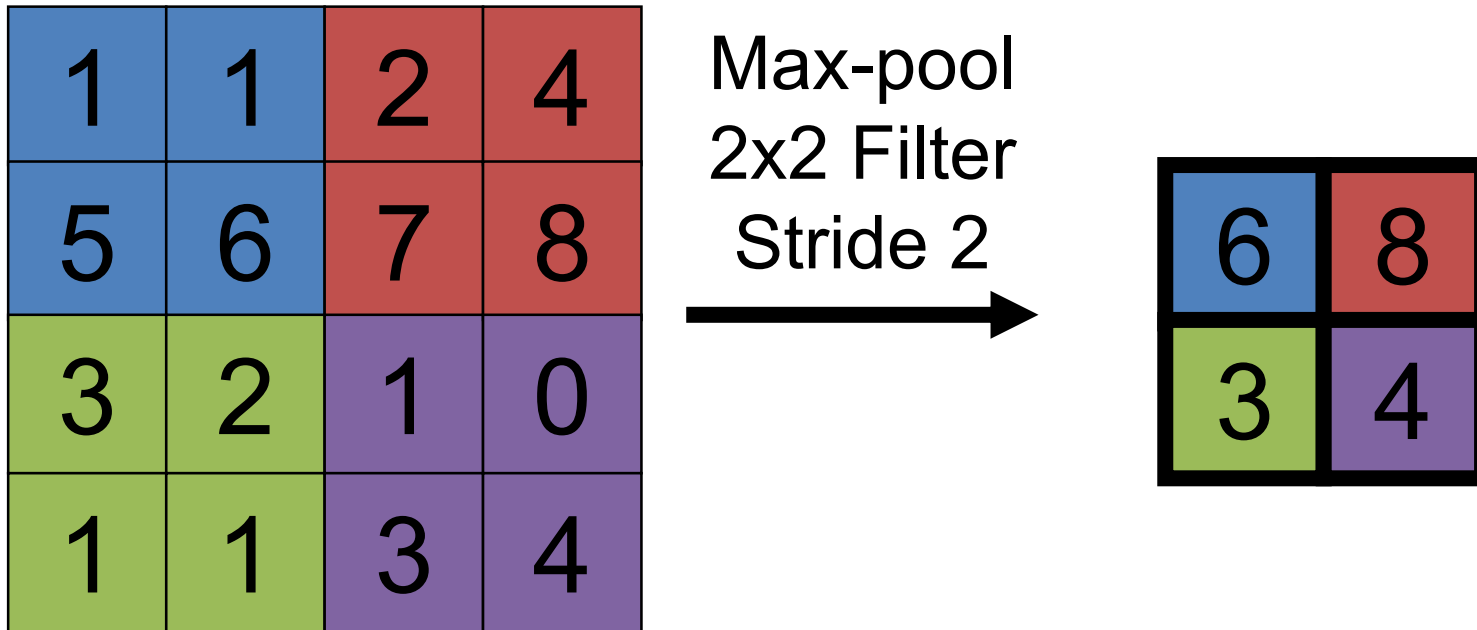
How many parameters? $5 \times 5 \times 3 \times 5 + 5 = 380$. **Same!**

Thought Problem

- How do you write a normal neural network as a convnet?

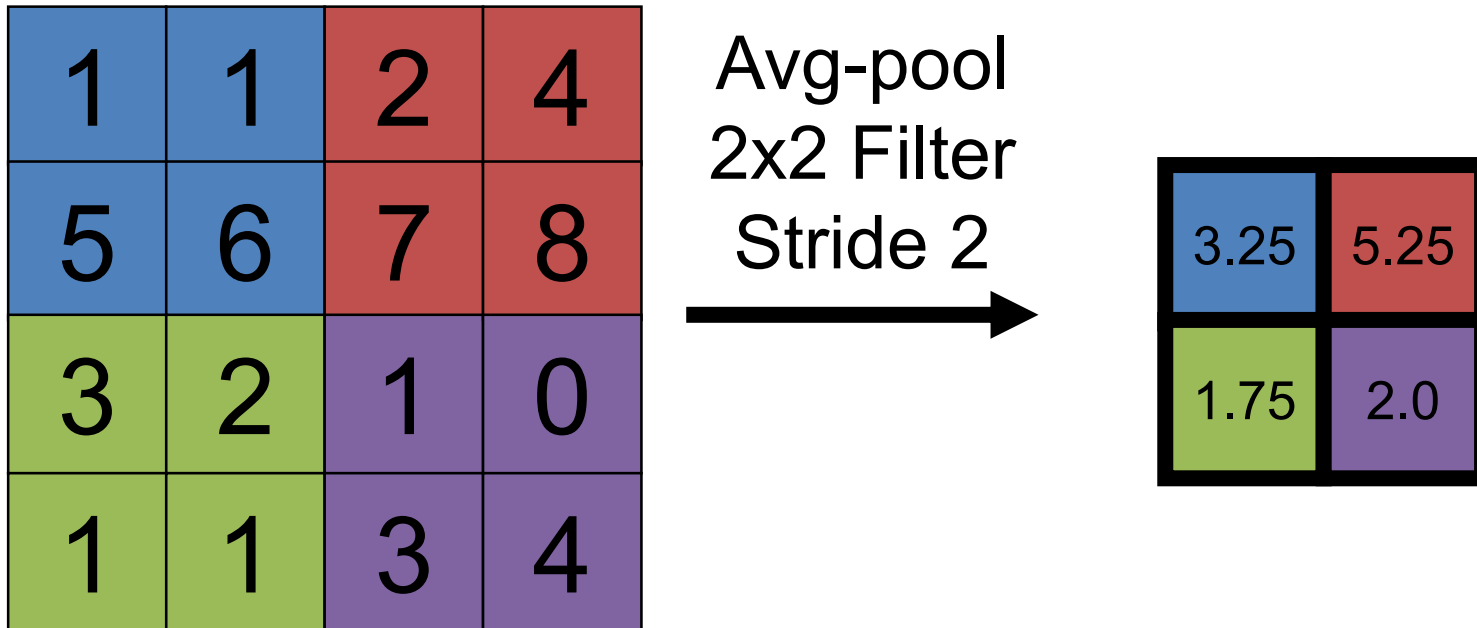
Other Layers – Pooling

Idea: just want spatial resolution of activations
/ images smaller; applied per-channel



Other Layers – Pooling

Idea: just want spatial resolution of activations
/ images smaller; applied per-channel



Other Layers – Pooling

Idea: just want spatial resolution of activations
/ images smaller; applied per-channel

			I14	I15	I16	I17
			I24	I25	I26	I27
			I34	I35	I36	I37
I41	I42	I43	I44	I45	I46	I47
I51	I52	I53	I54	I55	I56	I57
I61	I62	I63	I64	I65	I66	I67
I71	I72	I73	I74	I75	I76	I77

Max-pool
3x3 Filter
Stride 2



O11

O11 = maximum
value in blue box

Other Layers – Pooling

Idea: just want spatial resolution of activations / images smaller; applied per-channel

I11	I12				I16	I17
I21	I22				I26	I27
I31	I32				I36	I37
I41	I42	I43	I44	I45	I46	I47
I51	I52	I53	I54	I55	I56	I57
I61	I62	I63	I64	I65	I66	I67
I71	I72	I73	I74	I75	I76	I77

Max-pool
3x3 Filter
Stride 2



O11	O12
-----	-----

O12 = maximum
value in blue box

Other Layers – Pooling

Idea: just want spatial resolution of activations
/ images smaller; applied per-channel

I11	I12	I13	I14			
I21	I22	I23	I24			
I31	I32	I33	I34			
I41	I42	I43	I44	I45	I46	I47
I51	I52	I53	I54	I55	I56	I57
I61	I62	I63	I64	I65	I66	I67
I71	I72	I73	I74	I75	I76	I77

Max-pool
3x3 Filter
Stride 2

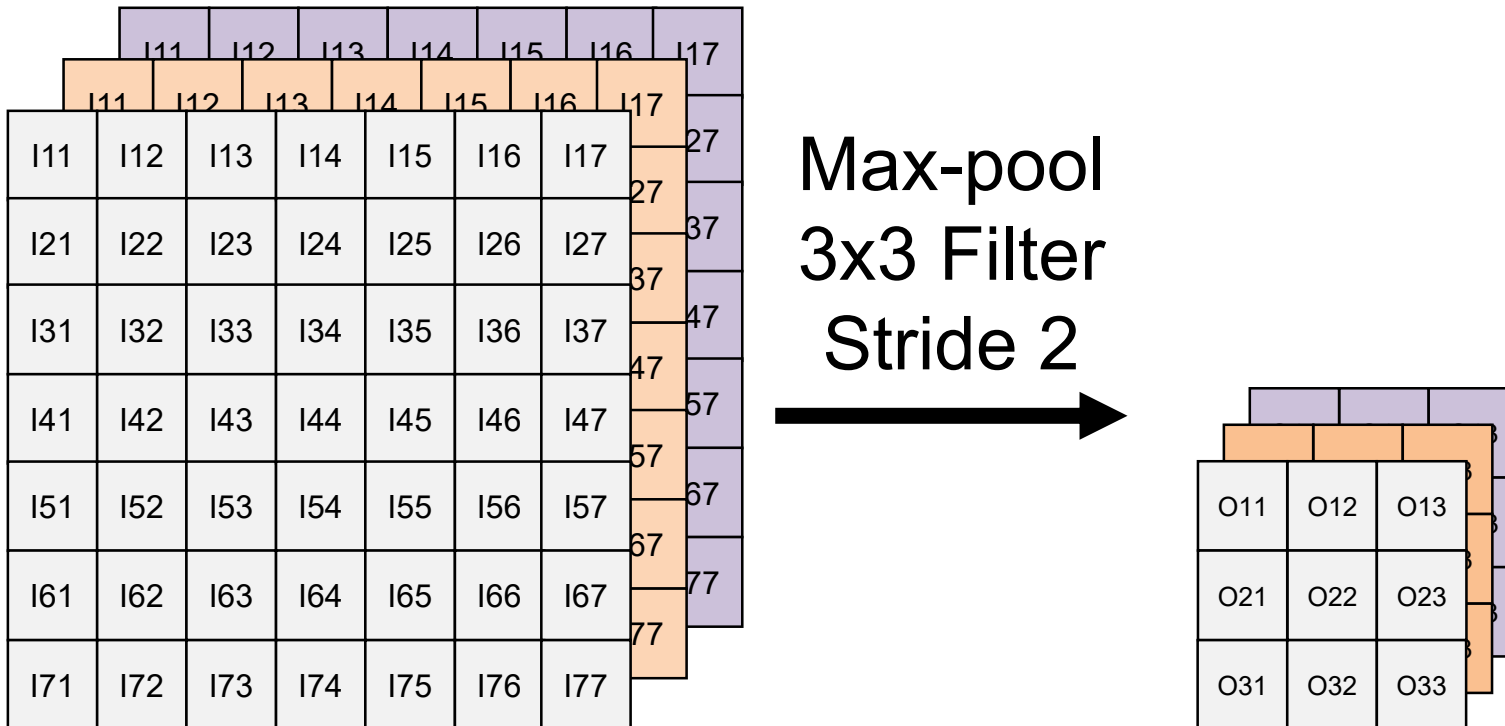


O11	O12	O13
-----	-----	-----

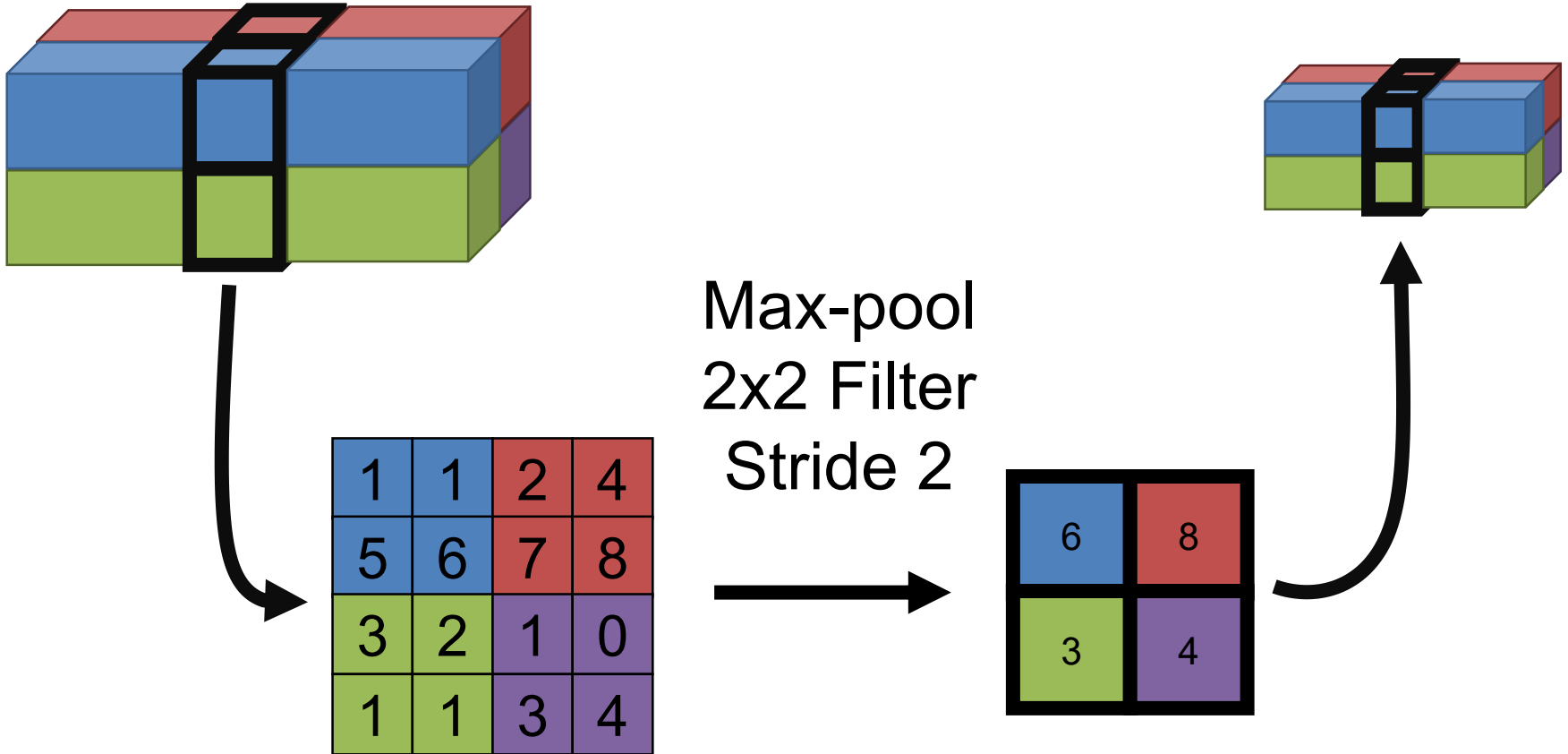
O13 = maximum
value in blue box

Other Layers – Pooling

Idea: just want spatial resolution of activations / images smaller; applied per-channel



Squeezing a Loaf of Bread



Example Network

Suppose we want to convert a 32x32x3 image into a 10x1 vector of classification results



Example Network

input: [32x32x3]

CONV with 10 3x3 filters, stride 1, pad 1:

gives: [32x32x10]

new parameters: $(3*3*3)*10 + 10 = 280$

RELU

CONV with 10 3x3 filters, stride 1, pad 1:

gives: [32x32x10]

new parameters: $(3*3*10)*10 + 10 = 910$

RELU

POOL with 2x2 filters, stride 2:

gives: [16x16x10]

parameters: 0

Example Network

Previous output: [16x16x10]

CONV with 10 3x3 filters, stride 1:

gives: [16x16x10]

new parameters: $(3*3*10)*10 + 10 = 910$

RELU

CONV with 10 3x3 filters, stride 1:

gives: [16x16x10]

new parameters: $(3*3*10)*10 + 10 = 910$

RELU

POOL with 2x2 filters, stride 2:

gives: [8x8x10]

parameters: 0

Example Network

Conv, Relu, Conv, Relu, Pool continues until it's [4x4x10]

Fully-Connected FC layer to 10 neurons
(which are our class scores)

Number of parameters:

$$10 * 4 * 4 * 10 + 10 = 1610$$

done!

An Alternate Conclusion

Conv, Relu, Conv, Relu, Pool continues until it's [4x4x10]

Average POOL 4x4x10 to 10 neurons

Fully-Connected FC layer to 10 neurons

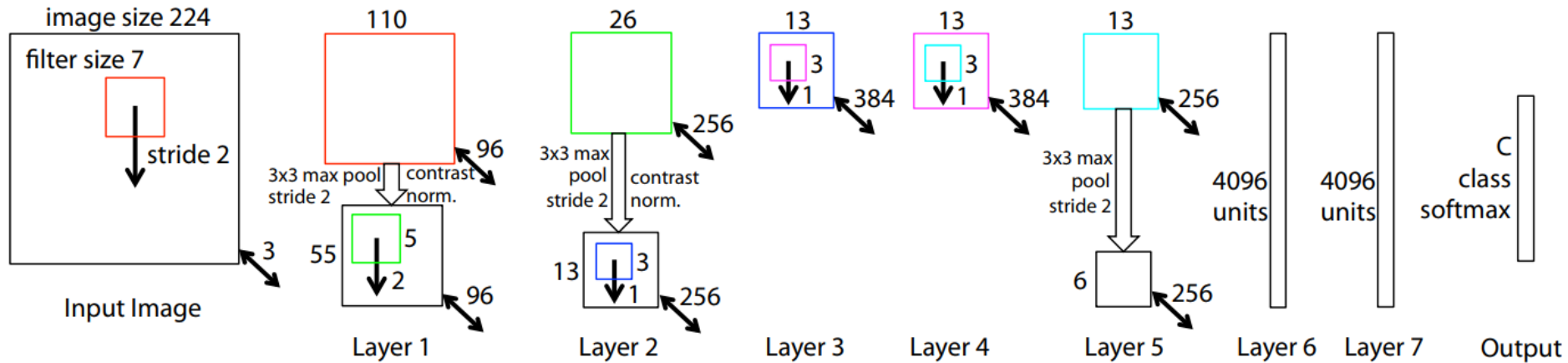
(which are our class scores)

Number of parameters:

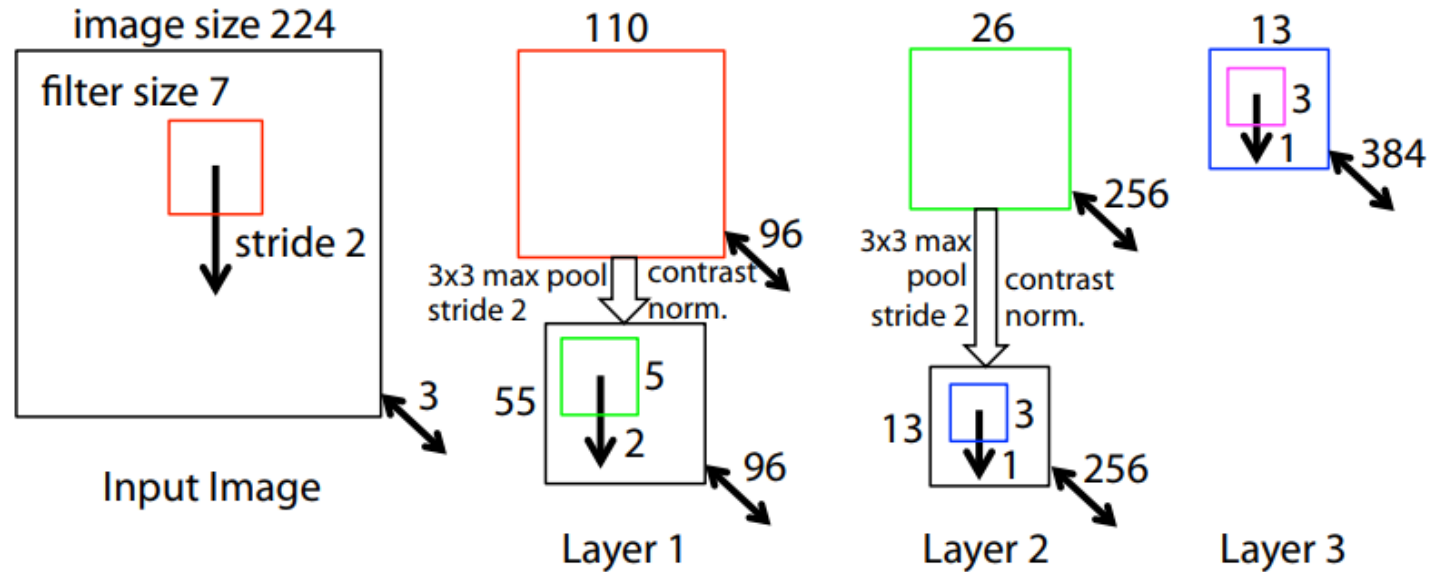
$$10 * 10 + 10 = 110$$

done!

Example Network



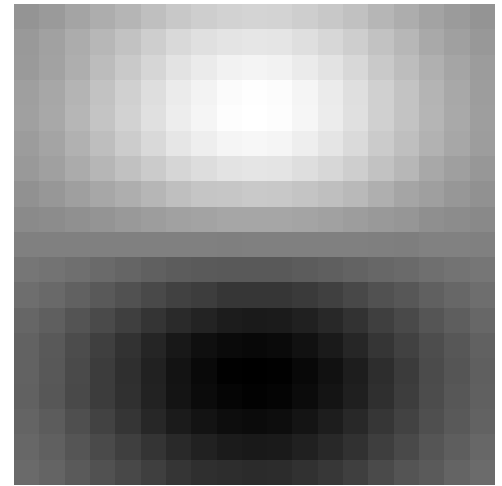
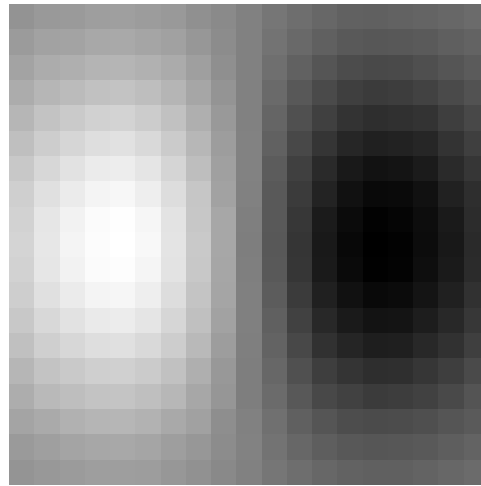
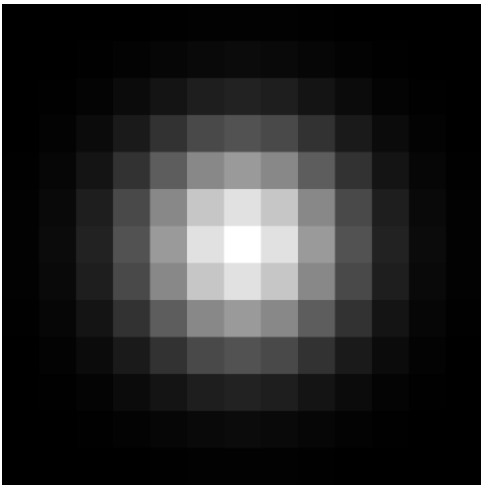
Example Network



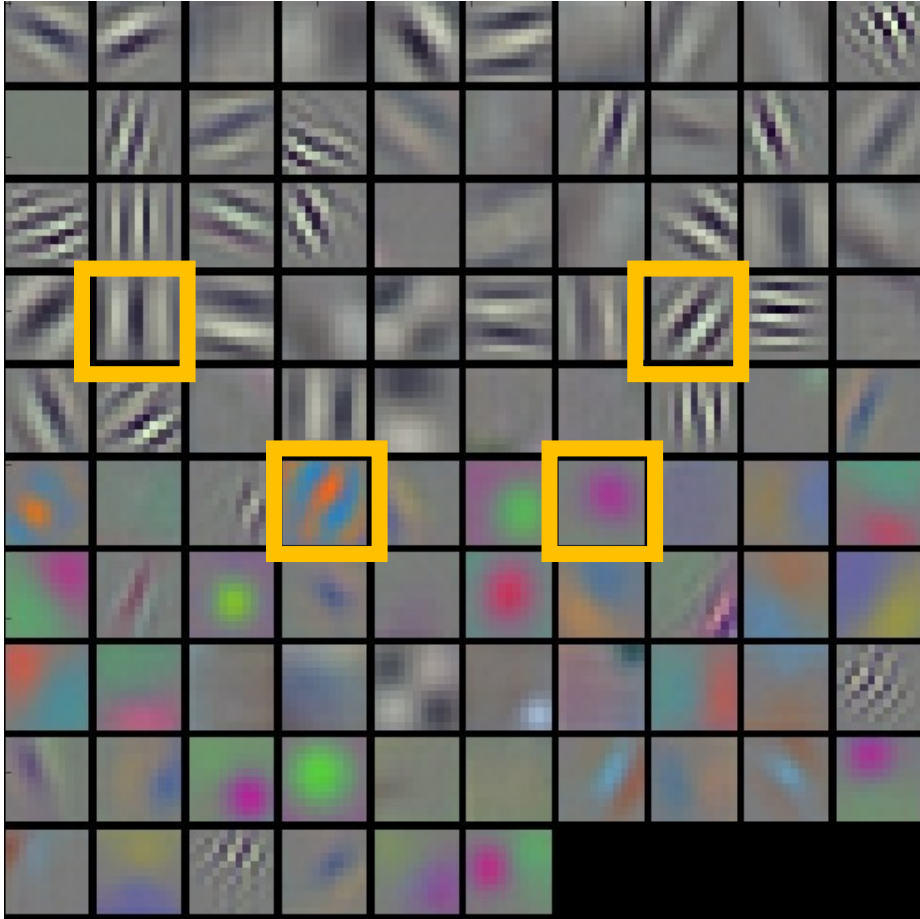
- (1) filter image with 96 7x7 filters
- (2) ReLU
- (3) 3x3 max pool with stride 2 (*and contrast normalization – now ignored*)

What Do The Filters Represent?

Recall: filters are images and we can look at them



What Do The Filters Represent?



First layer filters of a network trained to distinguish 1000 categories of objects

Remember these filters go over color.

For the interested:

[Gabor filter](#)

