
Deep Learning and Applications

Today's Outline

- Python Walkthrough
- Feedforward Neural Nets
- Convolutional Neural Nets
 - Convolution
 - Pooling

Python Walkthrough

Python Setup (I)

- Necessary for the programming portions of the assignments
- More precisely, use Ipython (ipython.org)



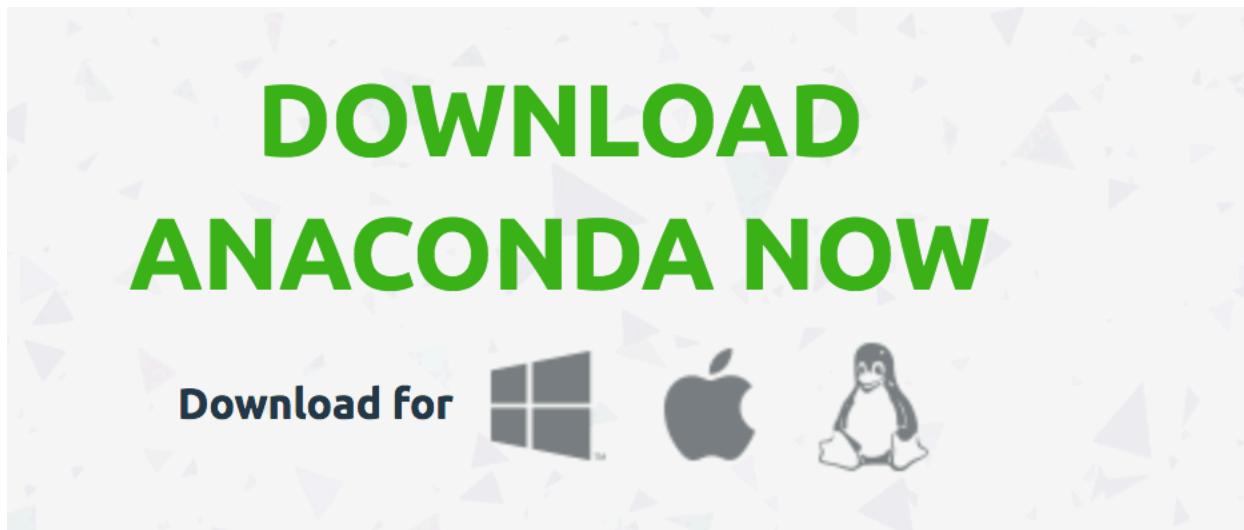
[Install](#) · [Documentation](#) · [Project](#) · [Jupyter](#) · [News](#) · [Cite](#) · [Donate](#) · [Books](#)

IPython provides a rich architecture for interactive computing with:

- A powerful interactive shell.
- A kernel for [Jupyter](#).
- Support for interactive data visualization and use of [GUI toolkits](#).
- Flexible, [embeddable](#) interpreters to load into your own projects.
- Easy to use, high performance tools for [parallel computing](#).

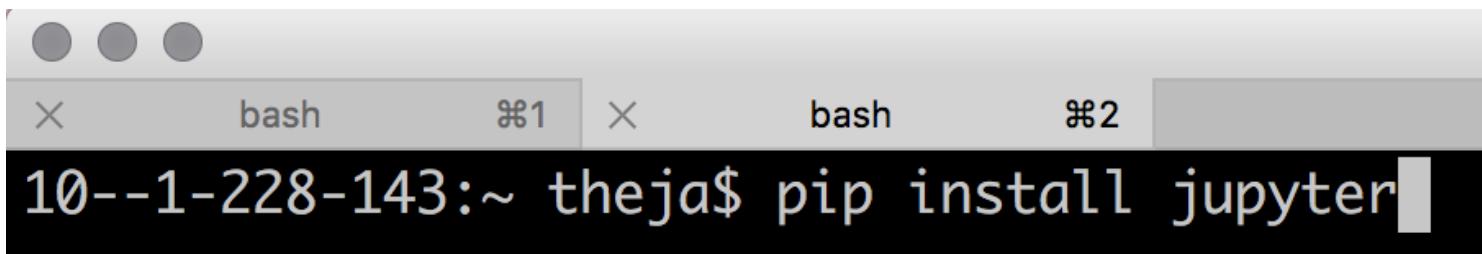
Python Setup (II)

- Install Python
 - Use Anaconda
(<https://www.continuum.io/downloads>)
 - Python 3



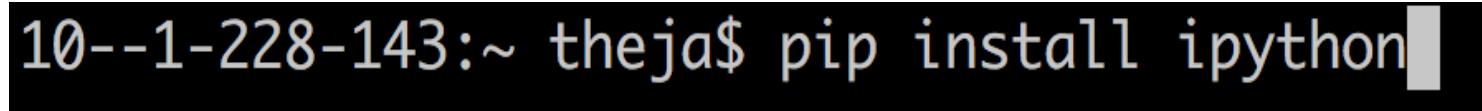
Python Setup (III)

- Install Ipython/Jupyter
 - If you installed the Anaconda distribution, you are all set
 - Else use the command on the command-line



```
10--1-228-143:~ theja$ pip install jupyter
```

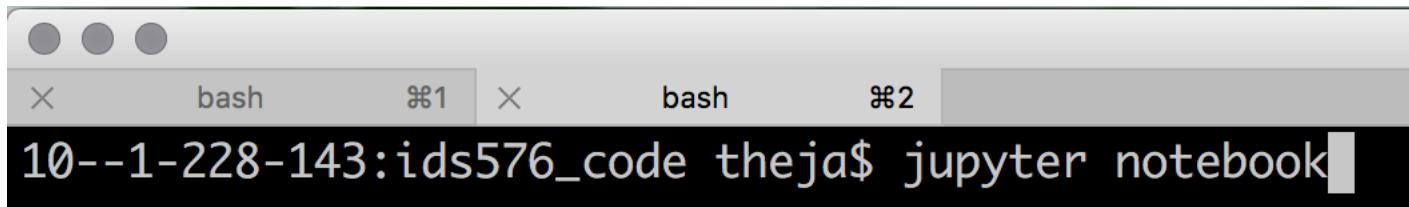
or



```
10--1-228-143:~ theja$ pip install ipython
```

Python Setup (IV)

- Run Jupyter (or ipython)



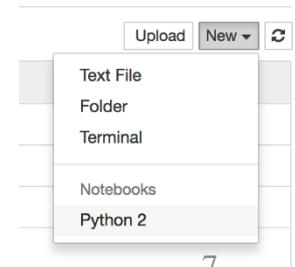
```
10--1-228-143:ids576_code theja$ jupyter notebook
```

A screenshot of a terminal window with two tabs labeled 'bash'. The current tab is 'bash' and contains the command 'jupyter notebook' which has been partially typed. The background shows a blurred view of a presentation slide.

- Your browser will open a page like this



- Start a new notebook (see button on the right)



Python Setup (V)

A screenshot of the Jupyter Notebook interface. The title bar says "jupyter Untitled". The toolbar includes standard file operations like File, Edit, View, Insert, Cell, Kernel, Help, and a CellToolbar. A Python 2 kernel icon is selected. The main area shows a code cell with the label "In []:". Inside the cell, the following Python code is written:

```
In [ ]: x = 1  
y = 2  
print(x+y)
```

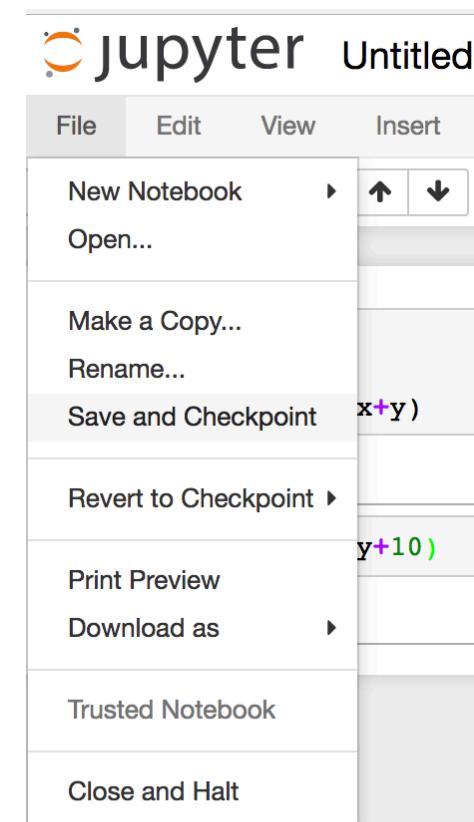
The word "code" is written in parentheses next to the cell content. A blue arrow points from the text "cells" to the right edge of the code cell.

A screenshot of the Jupyter Notebook interface showing the result of running a cell. The title bar says "jupyter Untitled". The toolbar is identical to the first screenshot. The main area shows the same code cell as before, but now it has been run. The output "3" is displayed below the cell. A new code cell is visible at the bottom with the label "In []:". A blue arrow points from the text "Press shift+enter, or ctrl+enter" to the bottom cell.

Python Setup (VI)

- Global variables are shared between cells
- Cells are typically run from top to bottom

```
In [1]: x = 1  
y = 2  
print(x+y)  
  
3  
  
In [2]: print(y+10)  
  
12
```



- Save changes using the save button

Python Review

- General purpose programming language
- 2 vs 3 (3 is backward incompatible)
- Very similar to Matlab (and better) for scientific computing
- It is dynamically typed

Python Review: Data Types

```
In [1]: x = 3
         y = 3.0
         z = 2
         print(x)
         print(y)
         print type(x)
         print type(y)
         print(x/z)
         print(y/z)
```

```
3
3.0
<type 'int'>
<type 'float'>
1
1.5
```

Python Review: Data Types

```
|: x +=1 #This is a comment. No unary operators (x++ will not work)
print(x)
y **=2
print y
```

```
4
9.0
```

```
|: a,b = True,False
mystring = 'ids676'
print a,b,mystring,'. In upper case: ' + mystring.upper()
```

```
True False ids676 . In upper case: IDS676
```

Python Review: List and Tuple

Dictionary, List, Tuple, Set

```
: mylist = ['i','d','s']
mytuple = (5,7,6)
print mylist, mytuple
```

```
['i', 'd', 's'] (5, 7, 6)
```

```
: mylist[0] = 'c'
mylist[1] = 'b'
mylist[2] = 'a'
mylist.append(5)
mylist.extend([7,6])
print mylist
```

```
['c', 'b', 'a', 5, 7, 6]
```

Python Review: Dictionary & Set

```
mylist[:2] = 'a','a'  
print mylist  
print set(mylist) #a set object will have unique elements
```

```
['a', 'a', 'a', 5, 7, 6]  
set(['a', 5, 6, 7])
```

```
course = {} #An empty dictionary/hash-map  
course[mytuple] = 'Advanced Prediction Models'  
course['572'] = 'Data Mining'  
print course
```

```
{(5, 7, 6): 'Advanced Prediction Models', '572': 'Data Mining'}
```

Python Review: Naïve for-loop

```
for x in mylist: #A for loop  
    print x
```

```
a  
a  
a  
5  
7  
6
```

Python Review: Function

Functions

```
import math, numpy
def softmax(z):
    return (1.0/(1+math.e**(-z)))
print softmax(-20)
print softmax(numpy.asarray([-1,0,1]))
```

```
2.06115361819e-09
[ 0.26894142  0.5                 0.73105858]
```

Python Review: Numpy

Numpy

```
a = numpy.array([-1,0,1])
print a,type(a),a.shape,a.dtype
b = numpy.array([[1.0,2,3],[1,2,3]])
print b, type(b), b.shape,b.dtype
```

```
[-1  0  1] <type 'numpy.ndarray'> (3,) int64
[[ 1.  2.  3.]
 [ 1.  2.  3.]] <type 'numpy.ndarray'> (2, 3) float64
```

```
c1 = b[1:,:0:2]#note the slice indexing
print c1,c1.shape
c2= b[1,0:2] #note the integer indexing
print c2,c2.shape
```

```
[[ 1.  2.]] (1, 2)
[ 1.  2.] (2,)
```

Python Review: Numpy

```
print b>2, b[b>2]
```

```
[[False False  True]
 [False False  True]] [ 3.  3.]
```

```
x = numpy.array([[1,2],[3,4]])
y = numpy.array([[1,1],[1,1]])
z = numpy.array([1,1])
print x*y #elementwise product
print x.dot(z) #matrix vector product
```

```
[[1 2]
 [3 4]]
[3 7]
```

```
print x.sum(), x.T
```

```
10 [[1 3]
 [2 4]]
```

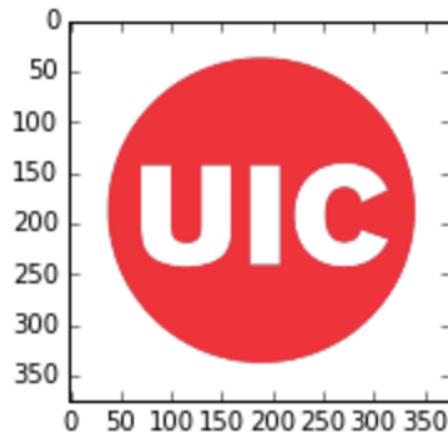
Python Review: Scipy Images

Scipy images

```
from scipy.misc import imread, imresize
%matplotlib inline
import matplotlib.pyplot as plt

img = imread('uic-logo-circle-red.jpg')

# Show the original image
plt.subplot(1, 2, 1)
plt.imshow(numpy.uint8(img))
plt.show()
```



Additional resources:

1. <http://cs231n.github.io/python-numpy-tutorial/>
2. <http://docs.scipy.org/doc/scipy/reference/index.html>

Questions?

Today's Outline

- Python Walkthrough
- Feedforward Neural Nets
- Convolutional Neural Nets
 - Convolution
 - Pooling

Feedforward Neural Network

- Linear model $f(x, W, b) = Wx + b$
- A feedforward neural network model will include nonlinearities
- Two layer model
 - $f(x, W_1, b_1, W_2, b_2) = W_2 \max(0, W_1 x + b_1) + b_2$
 - Say x is d dimensional
 - W_1 is $d \times q$ dimensional
 - W_2 is $q \times p$ dimensional
 - Then the number of hidden nodes is q
 - The number of labels is p
 - The notion of layer is for vectorizing/is conceptual

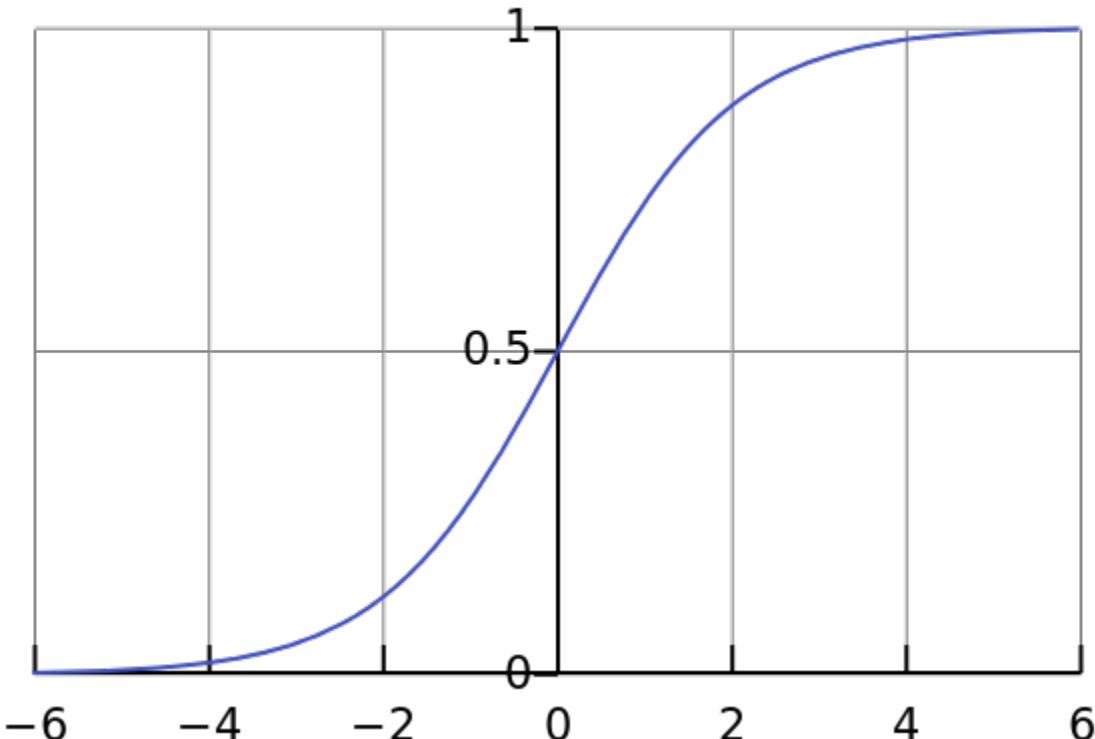
Nonlinearities (I)

Name	Formula	Year
none	$y = x$	-
sigmoid	$y = \frac{1}{1+e^{-x}}$	1986
tanh	$y = \frac{e^{2x}-1}{e^{2x}+1}$	1986
ReLU	$y = \max(x, 0)$	2010
(centered) SoftPlus	$y = \ln(e^x + 1) - \ln 2$	2011
LReLU	$y = \max(x, \alpha x), \alpha \approx 0.01$	2011
maxout	$y = \max(W_1x + b_1, W_2x + b_2)$	2013
APL	$y = \max(x, 0) + \sum_{s=1}^S a_i^s \max(0, -x + b_i^s)$	2014
VLReLU	$y = \max(x, \alpha x), \alpha \in 0.1, 0.5$	2014
RReLU	$y = \max(x, \alpha x), \alpha = \text{random}(0.1, 0.5)$	2015
PReLU	$y = \max(x, \alpha x), \alpha \text{ is learnable}$	2015
ELU	$y = x, \text{ if } x \geq 0, \text{ else } \alpha(e^x - 1)$	2015

- How to pick the nonlinearity/activation function?

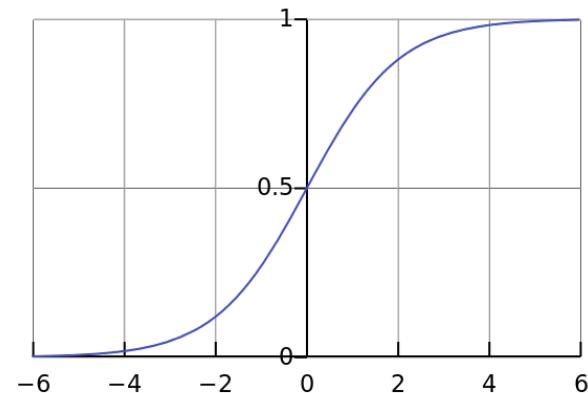
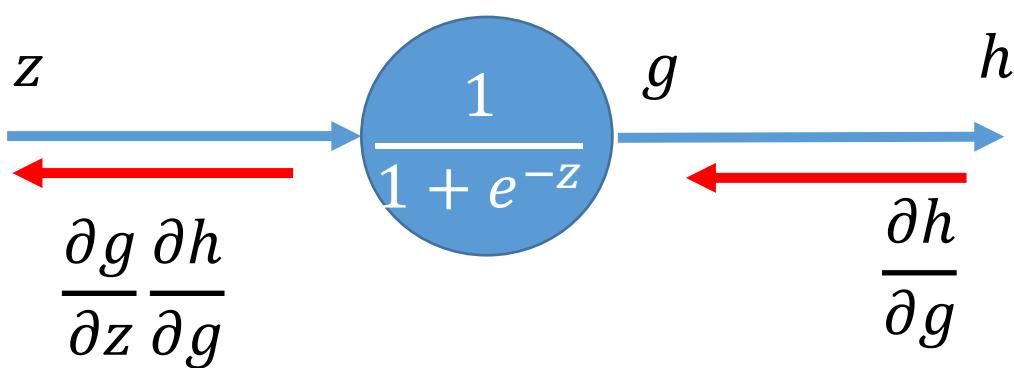
Nonlinearities (II)

- Sigmoid
 - Is a map whose range is $[0,1]$



Nonlinearities (III)

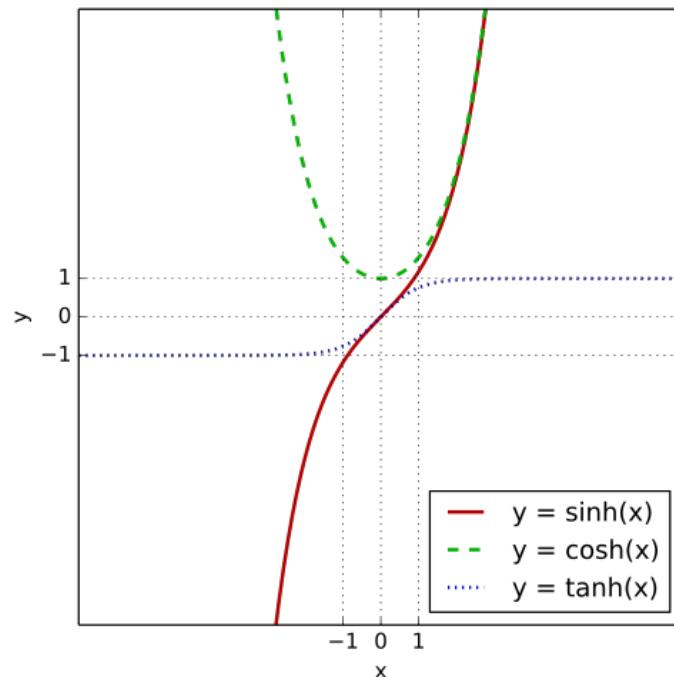
- Saturated node/neuron makes gradients vanish



- Not zero-centered
 - Empirically may lead to slower convergence

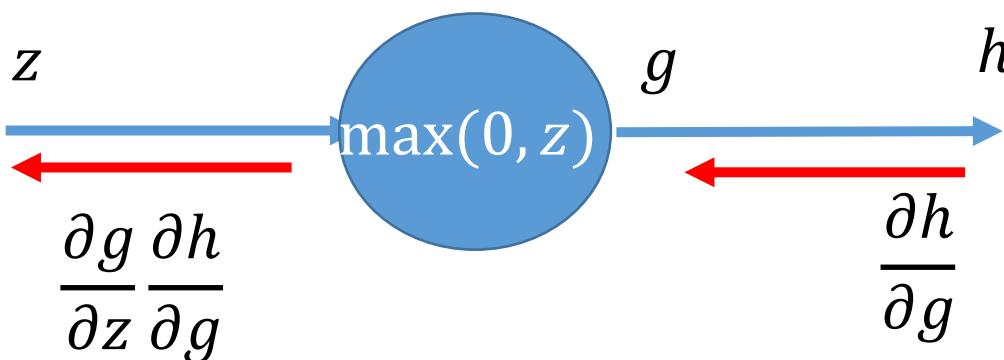
Nonlinearities (IV)

- `tanh()` addresses the zero-centering problem. So will typically give better results
- Still gradients vanish

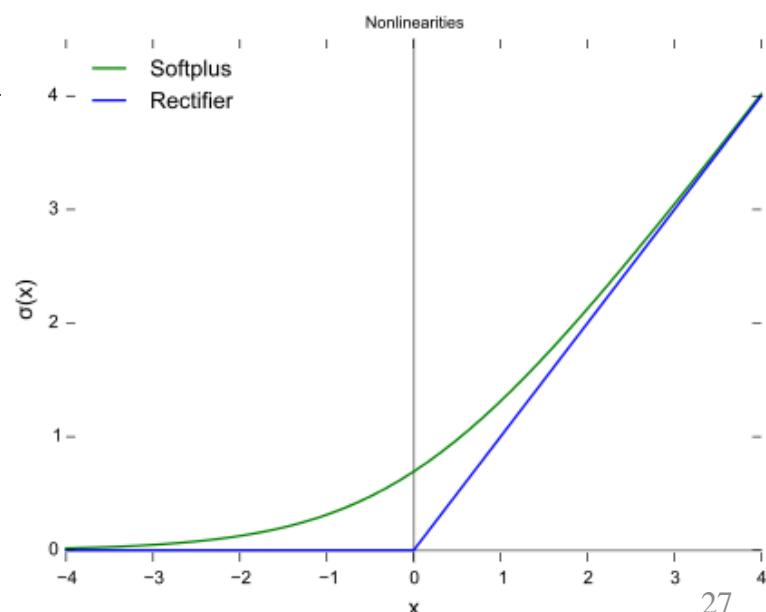


Nonlinearities (∇)

- ReLU (2012 Krizhevsky et al.)
- No vanishing gradient on the positive side
- Empirically observed to be very good
- Initialization/high learning rate may lead to permanently dead ReLUs (diagnosable)



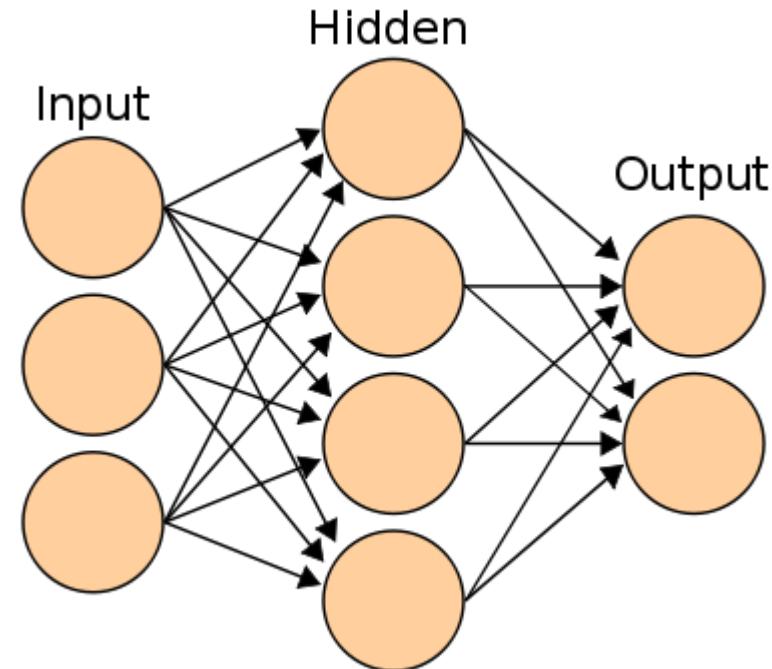
Is a gradient gate!



¹Figure: CC0, <https://en.wikipedia.org/w/index.php?curid=48817276>

Feedforward Neural Net

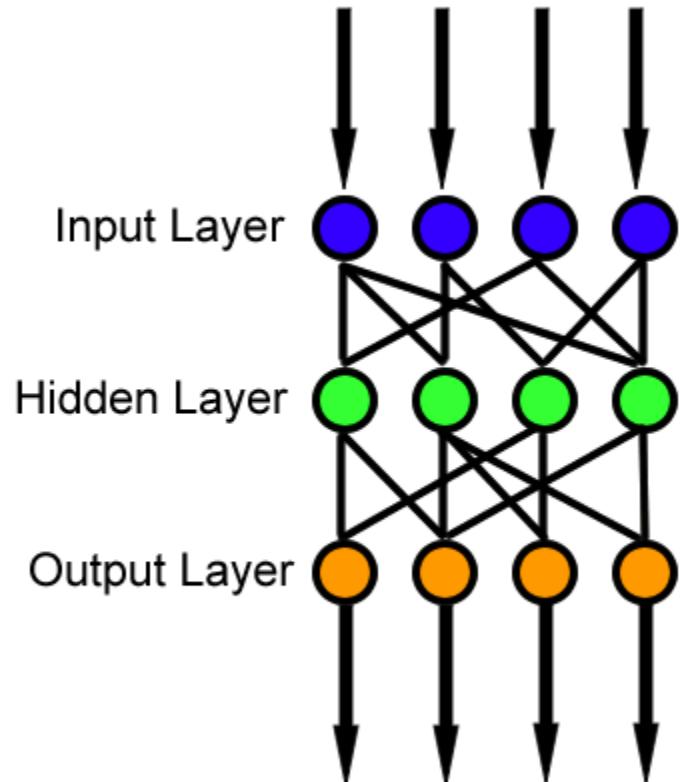
- Lets focus on a 2-layer net
- Layers
 - Input
 - Hidden
 - Output
- Node
- Nonlinearity
 - Activation



$$f(x, W_1, b_1, W_2, b_2) = W_2 \max(0, W_1 x + b_1) + b_2$$

Feedforward Net: Two Layer Model

- Number of layers is the number of W, b pairs
- Some questions to think about:
 - How to pick the number of layers?
 - How to pick the number of hidden units in each layer?



Feedforward Net and Backprop

- Choose a mini-batch (sample) of size B
- Forward propagate through the computation graph
 - Compute losses $L_{i_1}, L_{i_2}, \dots L_{i_B}$ and $R(W_1, b_1, W_2, b_2)$
 - Get loss L for the batch
- Backprop to compute gradients with respect to W_1, b_1, W_2 and b_2
- Update parameters W_1, b_1, W_2 and b_2
 - In the direction of the negative gradient

Feedforward Net in Python

```
# Feedforward neural net model

# Start with an initial set of parameters randomly
h = 100 # size of hidden layer
W = 0.01 * np.random.randn(D,h)
b = np.zeros((1,h))
W2 = 0.01 * np.random.randn(h,K)
b2 = np.zeros((1,K))

# Initial values from hyperparameter
reg = 1e-3 # regularization strength

#For simplicity, we will not optimize this using grid search here.
```

Feedforward Net in Python

```
#Perform batch SGD using manual backprop

#For simplicity we will take the batch size to be the same as number of examples
num_examples = X.shape[0]

#Initial value for the Gradient Descent Parameter
step_size = 1e-0 #Also called learning rate

#For simplicity, we will not hand tune this algorithm parameter as well.

# gradient descent loop
for i in xrange(10000):

    # evaluate class scores, [N x K]
    hidden_layer = np.maximum(0, np.dot(X, W) + b) # note, ReLU activation
    scores = np.dot(hidden_layer, W2) + b2

    # compute the class probabilities
    exp_scores = np.exp(scores)
    probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True) # [N x K]

    # compute the loss: average cross-entropy loss and regularization
    correct_logprobs = -np.log(probs[range(num_examples),y])
    data_loss = np.sum(correct_logprobs)/num_examples
    reg_loss = 0.5*reg*np.sum(W*W) + 0.5*reg*np.sum(W2*W2)
    loss = data_loss + reg_loss
    if i % 1000 == 0:
        print "iteration %d: loss %f" % (i, loss)
```

Feedforward Net in Python

```
# compute the gradient on scores
dscores = probs
dscores[range(num_examples),y] -= 1
dscores /= num_examples

# backpropate the gradient to the parameters
# first backprop into parameters W2 and b2
dW2 = np.dot(hidden_layer.T, dscores)
db2 = np.sum(dscores, axis=0, keepdims=True)
# next backprop into hidden layer
dhidden = np.dot(dscores, W2.T)
# backprop the ReLU non-linearity
dhidden[hidden_layer <= 0] = 0
# finally into W,b
dW = np.dot(X.T, dhidden)
db = np.sum(dhidden, axis=0, keepdims=True)

# add regularization gradient contribution
dW2 += reg * W2
dW += reg * W

# perform a parameter update
W += -step_size * dW
b += -step_size * db
W2 += -step_size * dW2
b2 += -step_size * db2
```

Feedforward Net in Python

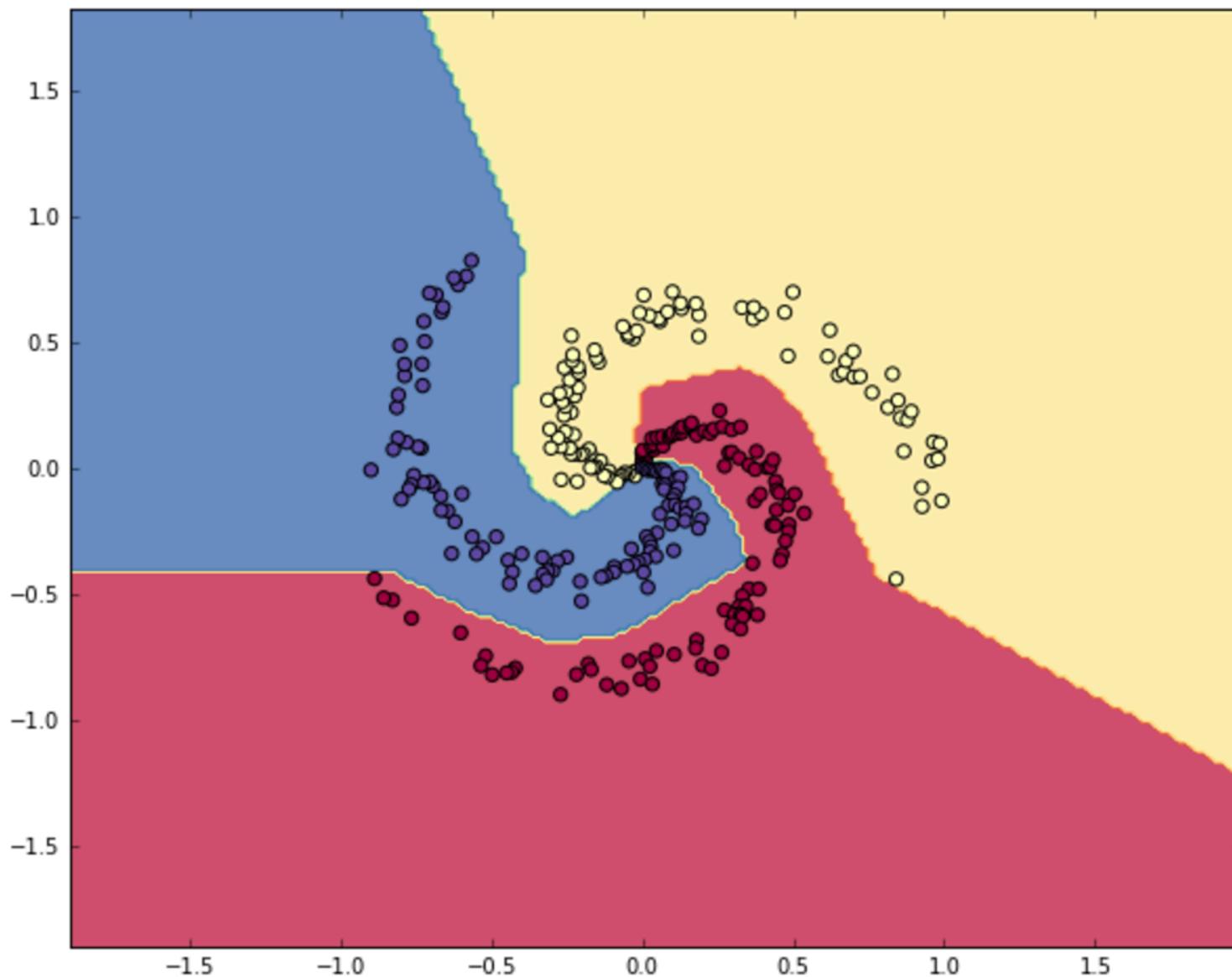
Post Training

```
# Post-training: evaluate test set accuracy

#For simplicity, we will use training data as proxy for test. Do not do this.
X_test = X
y_test = y

hidden_layer = np.maximum(0, np.dot(X_test, W) + b)
scores = np.dot(hidden_layer, W2) + b2
predicted_class = np.argmax(scores, axis=1)
print 'test accuracy: %.2f' % (np.mean(predicted_class == y_test))
```

Feedforward Net in Python



FNN in the Browser

- See playground.tensorflow.org

Questions?

Today's Outline

- Python Walkthrough
- Feedforward Neural Nets
- Convolutional Neural Nets
 - Convolution
 - Pooling

Convolutional Neural Network

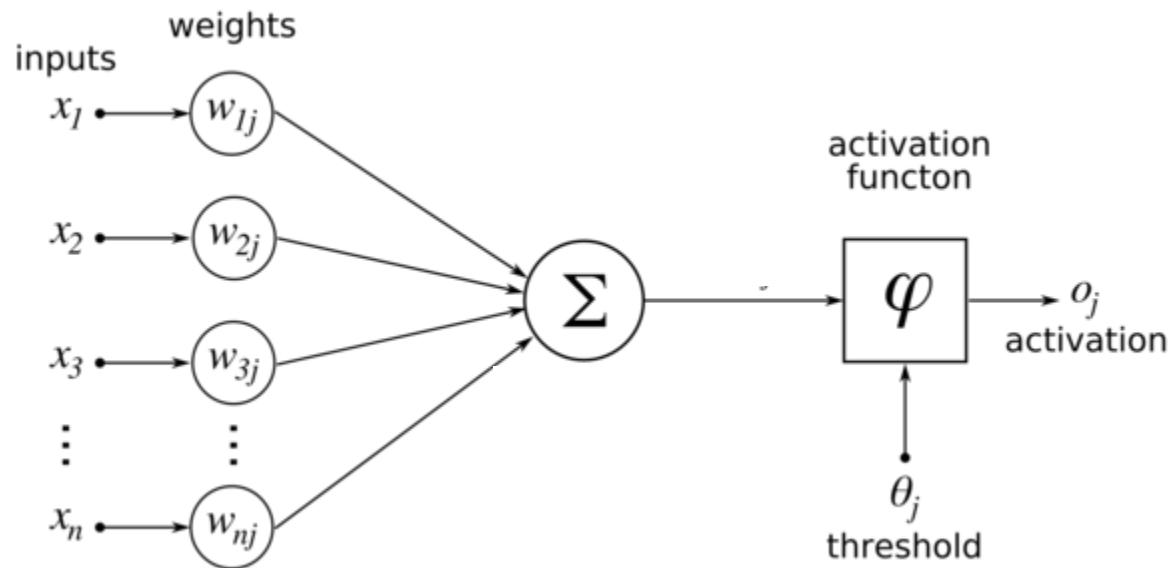
Similar to Feedforward NN

- Similar to feedforward neural networks
- Each neuron/node is associated with weights and a bias
- Node receives input
 - Performs dot product of vectors
 - Applies non-linearity
- The difference:
 - Number of parameters is reduced!

How? That is the content of this lecture!

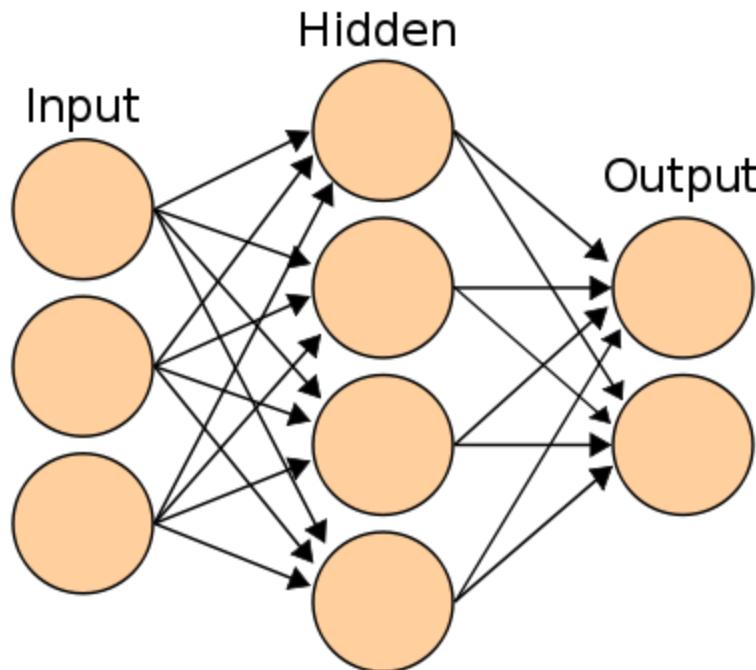
Similar to Feedforward NN

- Recall a Feedforward net:
 - Get a vector x_i and transform it to a score vector by passing through a sequence of hidden layers
 - Each hidden layer has neurons
 - Each neuron is fully connected to previous layer



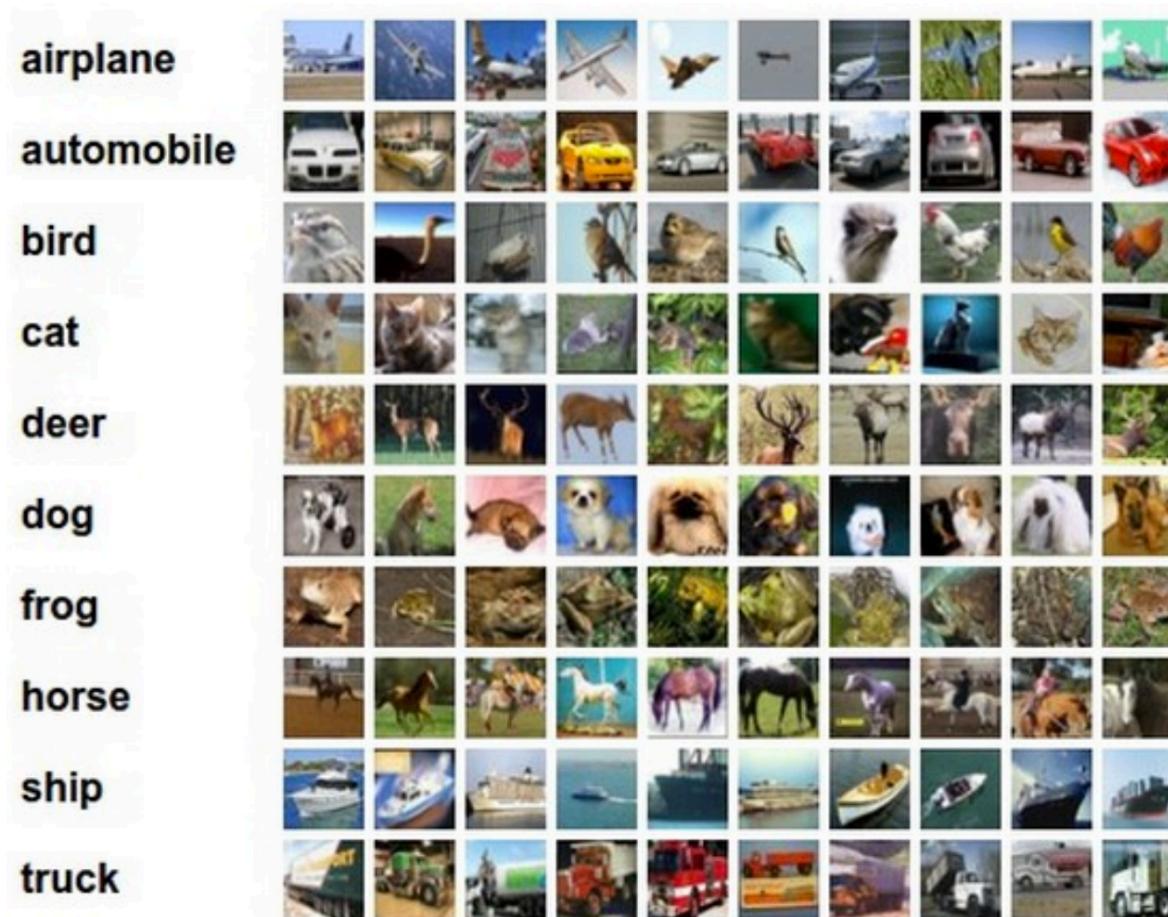
Towards CNNs (I)

- Feedforward net:
 - Can you visualize the connections for an arbitrary neuron here?



Towards CNNs (II)

- Consider the CIFAR-10 Dataset. Images are 32*32*3 in size



Towards CNNs (III)

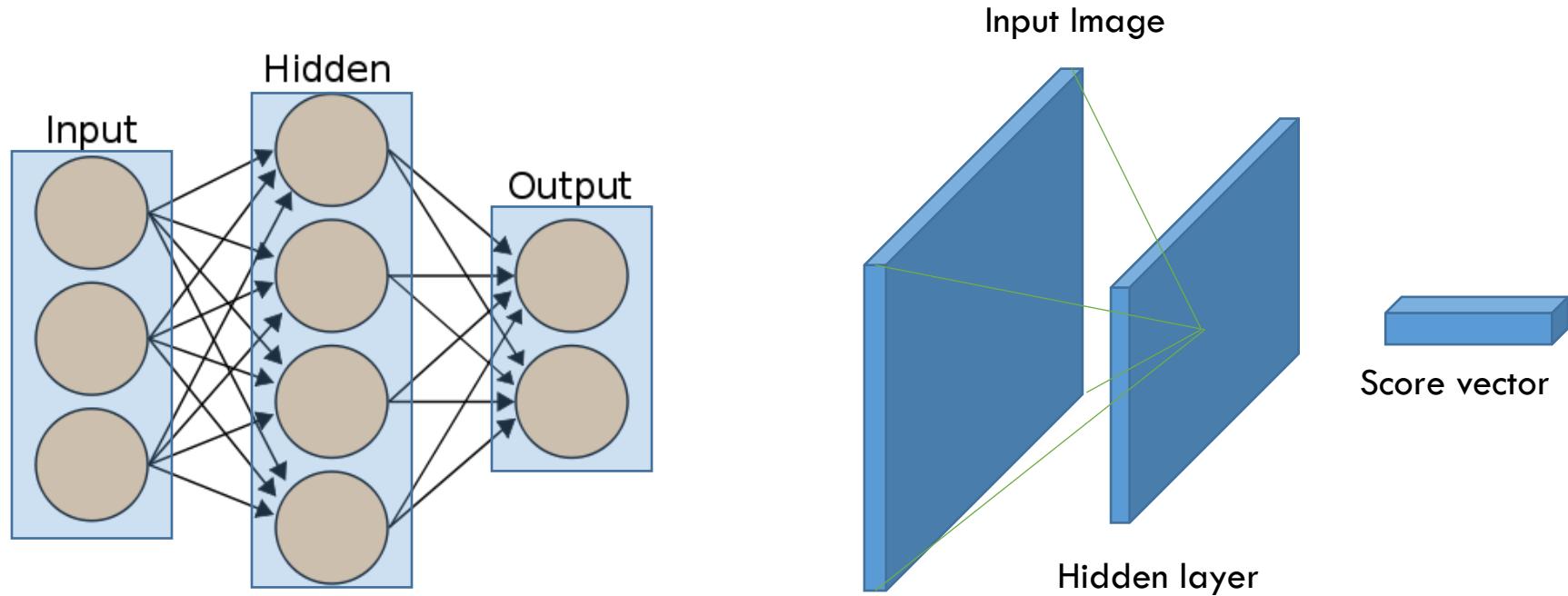
- First fully connected feedforward neuron would have $32*32*3$ weights associated with it (+1 bias parameter)
- What if the images were $1280*800*3$?
- Clearly, we also need many neurons in each hidden layer. This leads to explosion in the total number of parameters (or the dimension of W 's and b 's)

CNN Architecture

- We will look at it from layers point of view
- The new idea is that layers have **width** and **depth**!
 - (In contrast, Feedforward NN layers only had height)
 - (depth here does NOT correspond to number of layers of a network)

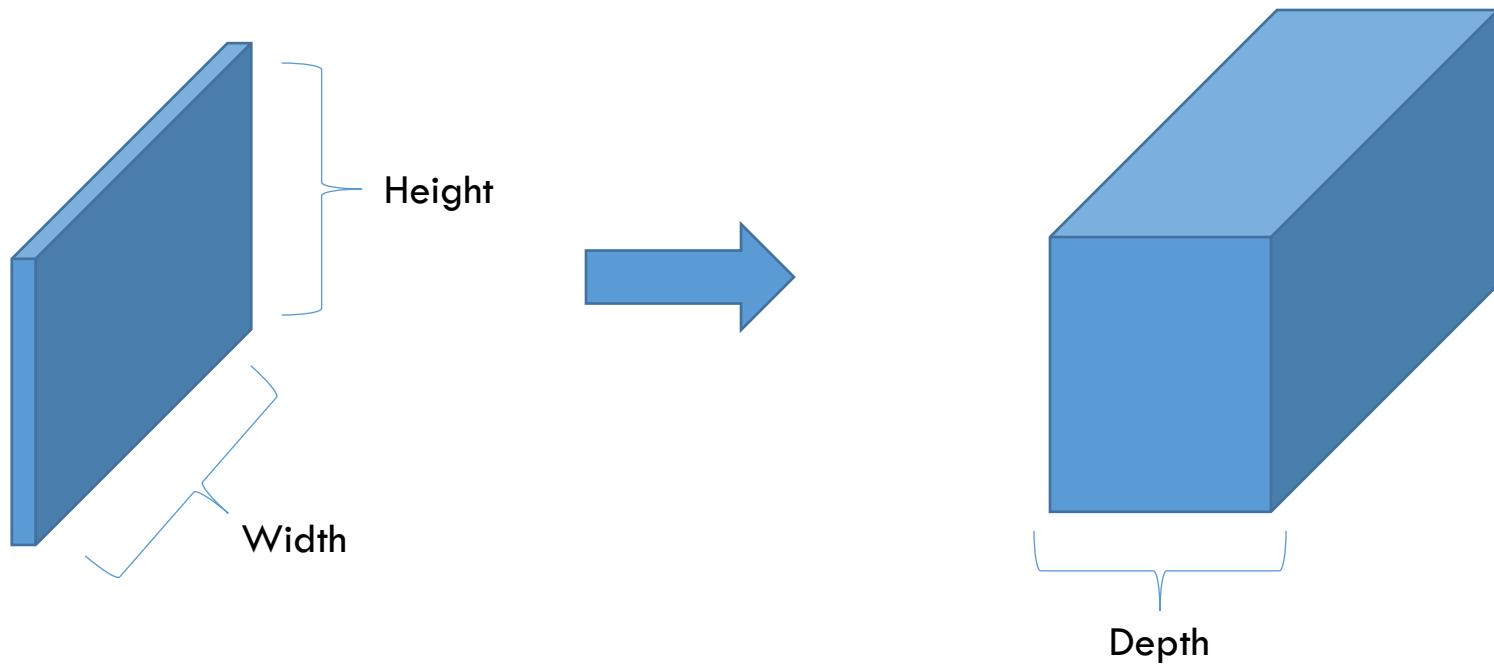
CNN Architecture

- View FFN layers as having width and height



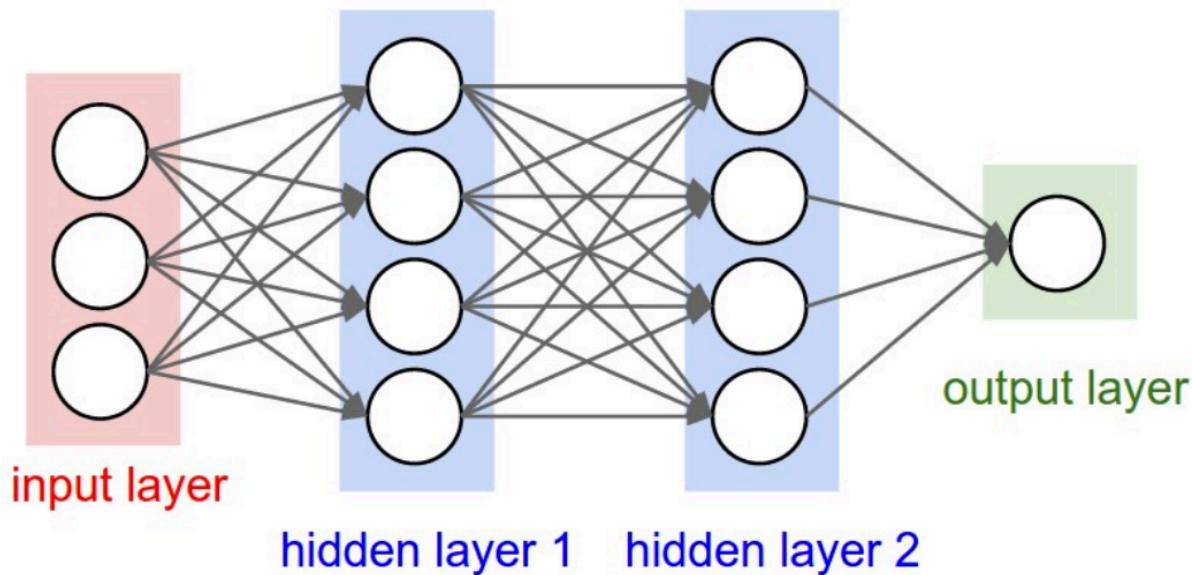
CNN Architecture

- The new idea is that CNN layers have **depth!**
 - (depth here does NOT correspond to number of layers of a network)



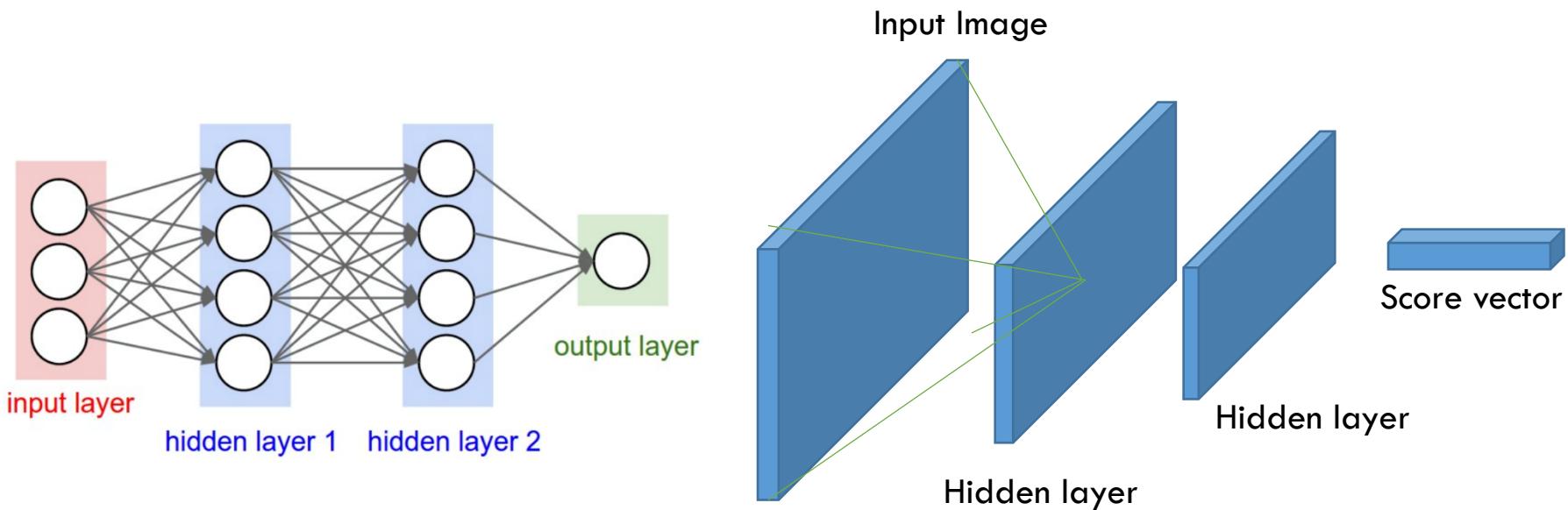
3D Volumes of Neurons

- Input has dimension $32*32*3$ (for CIFAR-10 dataset)
- Final output has dimension $1*1*10$ (10 classes)
- Previously,



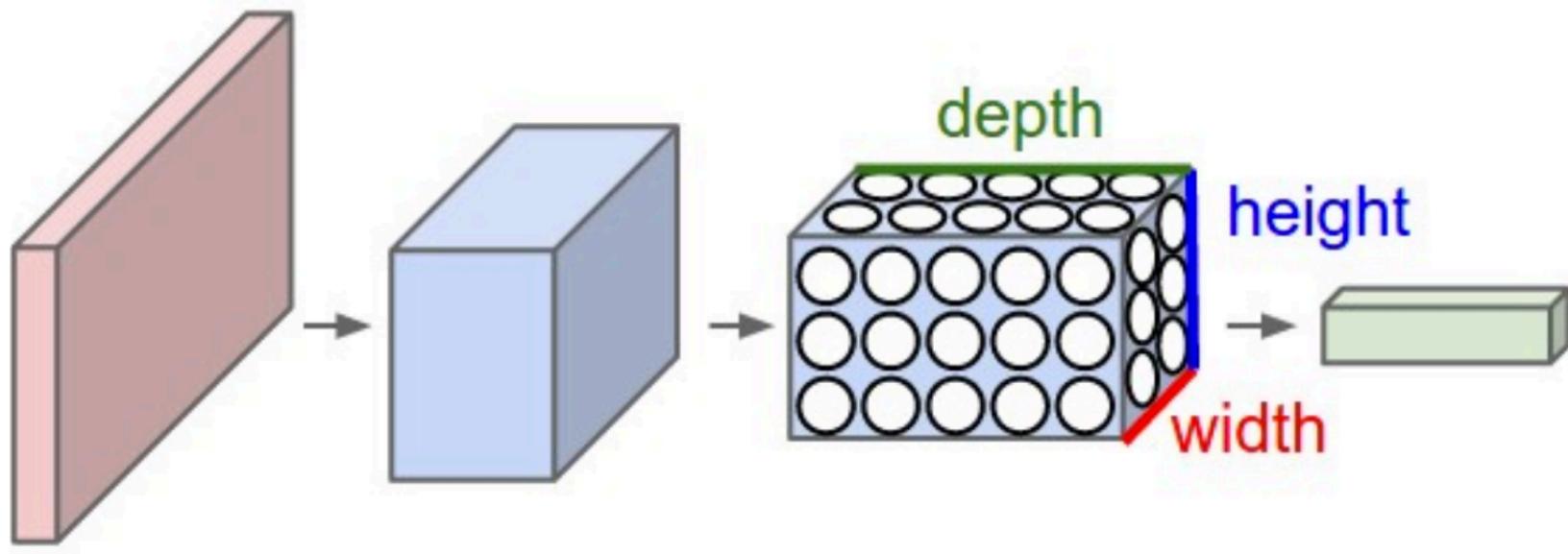
3D Volumes of Neurons

- Input has dimension $32*32*3$ (for CIFAR-10 dataset)
- Final output has dimension $1*1*10$ (10 classes)
- So assuming 2 hidden layers, previously we had,



3D Volumes of Neurons

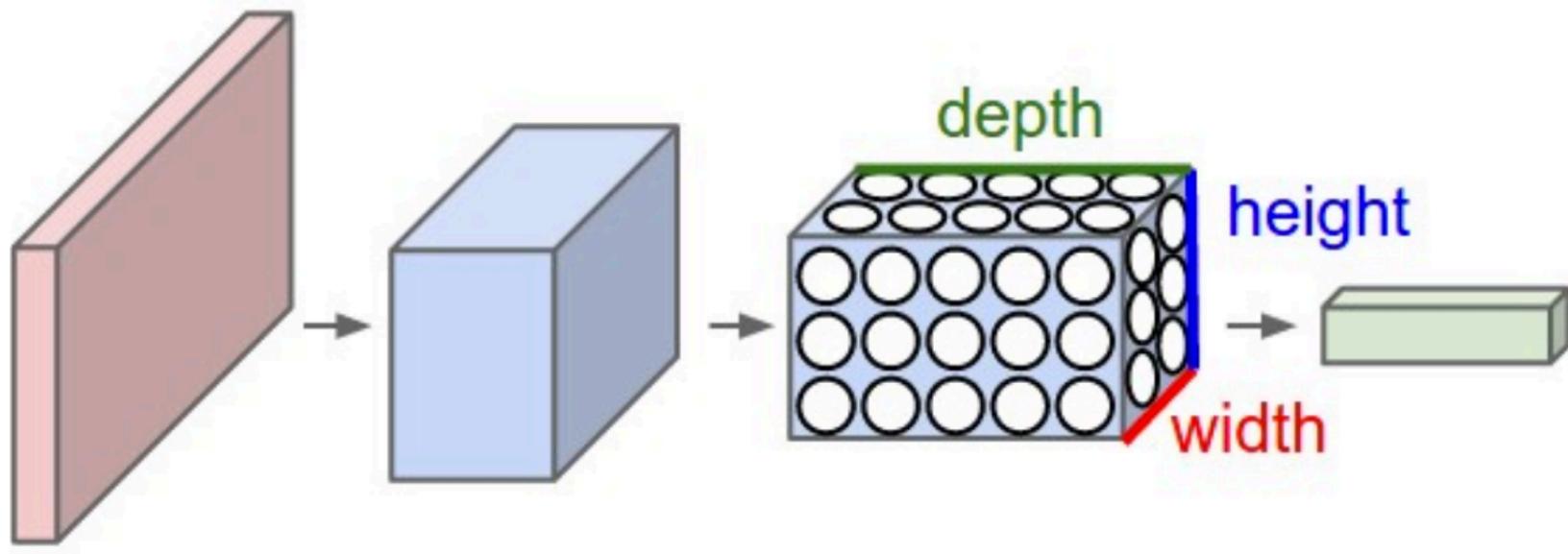
- Now,



- Each layer simply does this: transforms an input tensor (3D volume) to an output tensor using some function

3D Volumes of Neurons

- Now,



- Each layer simply does this: transforms an input tensor (3D volume) to an output tensor using some function

CNN Layers

- Three types
 - Convolutional Layer (CONV)
 - Pooling Layer (POOL)
 - Fully Connected Layer (same as Feedforward neural network, i.e., $1*1*\#$ Neurons is the layer's output tensor)
- Stack these in various ways

CNN Example Architecture

- Say our classification dataset is CIFAR-10
- Let the architecture be as follows:
 - INPUT -> CONV -> POOL -> FC
- INPUT:
 - This layer is nothing but $32 \times 32 \times 3$ in dimension (width*height*3 color channels)

CNN Example Architecture

- Say our classification dataset is CIFAR-10
- Let the architecture be as follows:
 - INPUT -> CONV -> POOL -> FC
- INPUT:
 - This layer is nothing but $32 \times 32 \times 3$ in dimension (width*height*3 color channels)
- CONV:
 - Neurons compute like regular feedforward neurons (sum the product of inputs with weights and add bias).
 - May output a different shaped tensor, say with dimension $32 \times 32 \times 12$

CNN Example Architecture

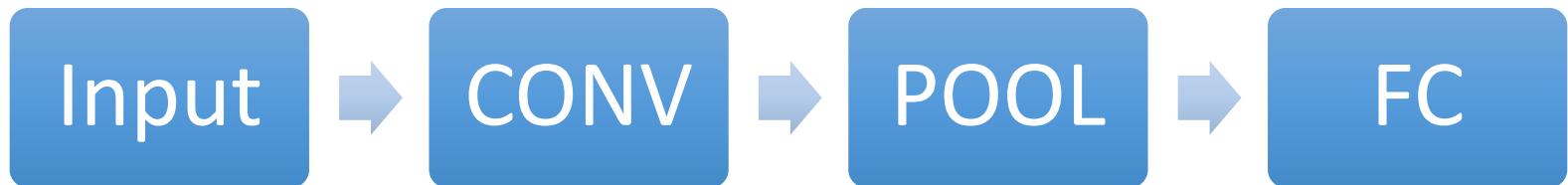
- POOL:
 - Performs a down-sampling in the spatial dimension
 - Outputs a tensor with the depth dimension the same as input
 - If input is $32*32*12$, then output could be $16*16*12$

CNN Example Architecture

- POOL:
 - Performs a down-sampling in the spatial dimension
 - Outputs a tensor with the depth dimension the same as input
 - If input is $32*32*12$, then output could be $16*16*12$
- FC:
 - This is the fully connected layer. Input can be any tensor (say $16*16*12$) but the output will have only one effective dimension ($1*1*10$ since this is the last layer and CIFAR-10 has 10 classes)

CNN Example Architecture

- So we went from pixels (32*32 RGB images) to scores (10 in number)



- Some layers have parameters (CONV and FC), other layers do not (POOL)
- Optimization of these parameters still for achieving scores consistent with image labels

The Convolution Layer (CONV)

- Layer's parameters correspond to a **set** of filters
- What is a filter?
 - A **linear function parameterized by a tensor**
 - Outputs a scalar
 - The parameter tensor is **learned** during training
- Example
 - First layer filter may be of dimension 3*3*3
 - 3 pixels wide
 - 3 pixels high
 - 3 unit **filter-depth** for three color channels
- We slide (convolve) the filter **across the width and height** of the input volume and compute the scalar output to be passed into the nonlinearity

CONV: Sliding/Convolving

- We slide (convolve) the filter across the width and height of the input volume and compute the scalar output to be passed into the nonlinearity

1 <small>x1</small>	1 <small>x0</small>	1 <small>x1</small>	0	0
0 <small>x0</small>	1 <small>x1</small>	1 <small>x0</small>	1	0
0 <small>x1</small>	0 <small>x0</small>	1 <small>x1</small>	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved Feature

Also see <http://setosa.io/ev/image-kernels/>

¹Figure: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

The Convolution Layer (CONV)

- Three things to notice
 - Filters are small along width and height
 - Same **filter-depth** as the input tensor (3D volume)
 - If the input is $x * y * z$, then filter could be $3 * 3 * z$
 - As we slide, we produce a **2D** activation map

The Convolution Layer (CONV)

- Three things to notice
 - Filters are small along width and height
 - Same **filter-depth** as the input tensor (3D volume)
 - If the input is $x * y * z$, then filter could be $3 * 3 * z$
 - As we slide, we produce a **2D** activation map
- Filters (i.e., filter parameters) will be learned during training that ‘detect’ certain visual features
 - Example:
 - Oriented edges, colors, etc. at the first layer
 - Specific patterns in higher layers

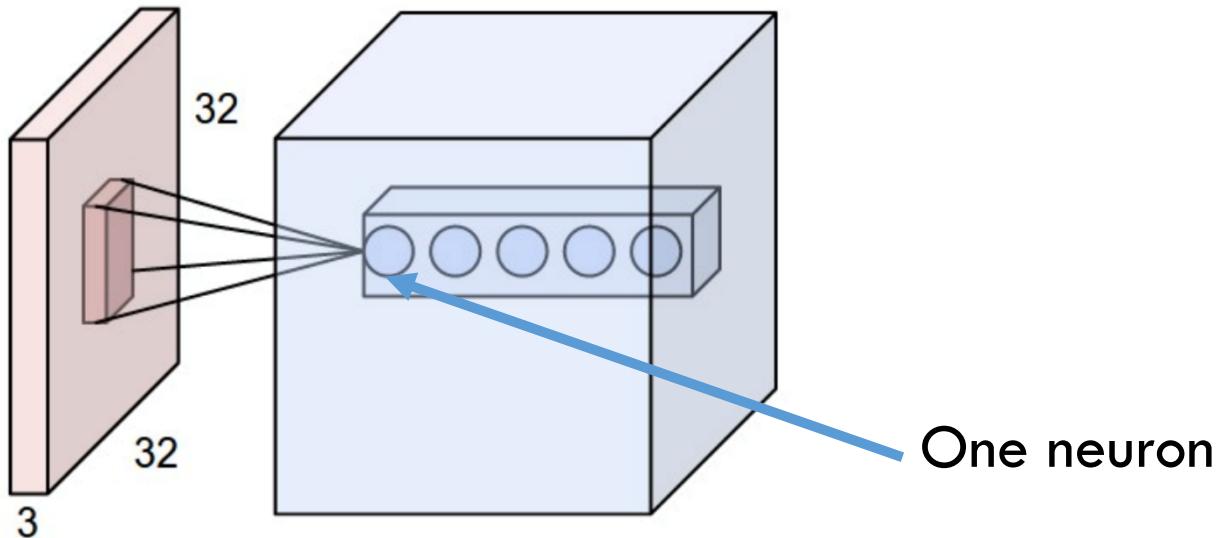
CONV: Filters

- Before we look at the patterns ...
- Lets now look at the neurons themselves
 - How are they connected?
 - How are they arranged?
 - How can we get reduced parameters?

CONV: Local Connectivity

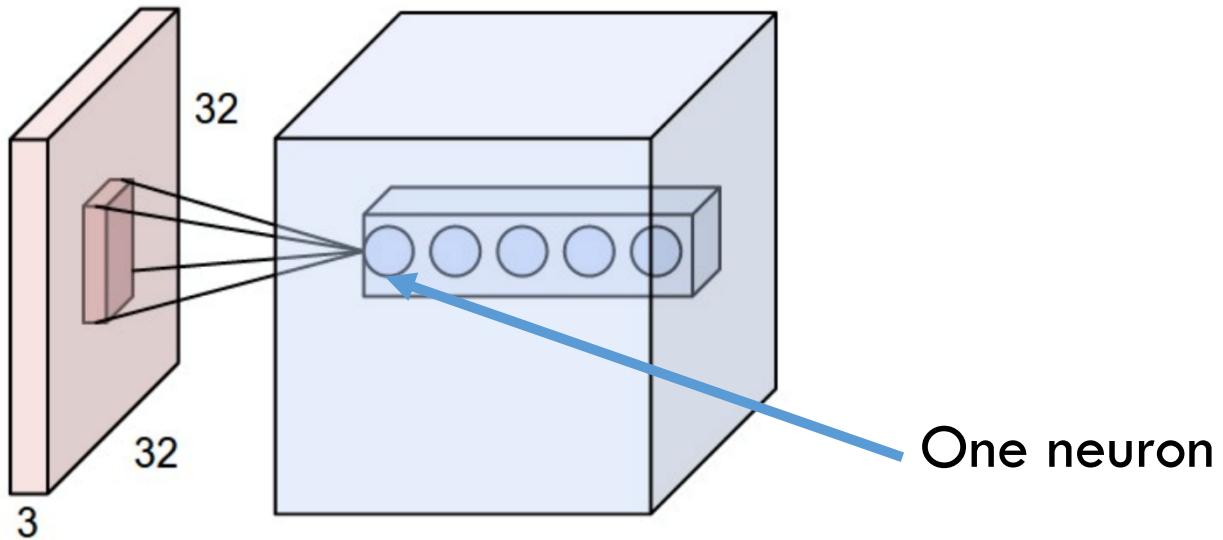
- Connect each neuron to a local (spatial) region of the input tensor
- Spatial extent of this connectivity is called **receptive field**
- Depth connectivity is the same as input depth

CONV: Local Connectivity



- Example: If input tensor is $32 \times 32 \times 3$ and filter is $3 \times 3 \times 3$ then
 - the number of weight parameters is 27, and
 - there is 1 bias parameter

CONV: Local Connectivity



- All 5 neurons are looking at the same spatial region
- Each neuron belongs to a different filter

CONV: Spatial Arrangement

- Back to layer point of view
- Size of output **tensor** depends on three numbers:
 - **Layer Depth**
 - Corresponds to the number of filters
 - **Stride** (how much the filter is moved spatial)
 - Example: If stride is 1, then filter is moved 1 pixel at a time
 - **Zero-padding**
 - Deals with boundaries (is usually 1 or 2)

CONV: Stride/Zero-pad

Stride = 1, Zero-padding = 0

The diagram illustrates a convolution operation. A blue arrow points from the text "Stride = 1, Zero-padding = 0" to a 5x5 input image matrix. The image matrix has alternating yellow and green cells. The first row contains values 1, 1, 1, 0, 0. The second row contains values 0, 1, 1, 1, 0. The third row contains values 0, 0, 1, 1, 1. The fourth row contains values 0, 0, 1, 1, 0. The fifth row contains values 0, 1, 1, 0, 0. Red annotations show the result of element-wise multiplication between the input and a 3x3 kernel. The first row of the kernel is [1, 1, 1] and the second row is [0, 1, 1]. The resulting 3x3 matrix is highlighted in pink and contains a single value '4' at position (1,1).

1 x1	1 x0	1 x1	0	0
0 x0	1 x1	1 x0	1	0
0 x1	0 x0	1 x1	1	1
0	0	1	1	0
0	1	1	0	0

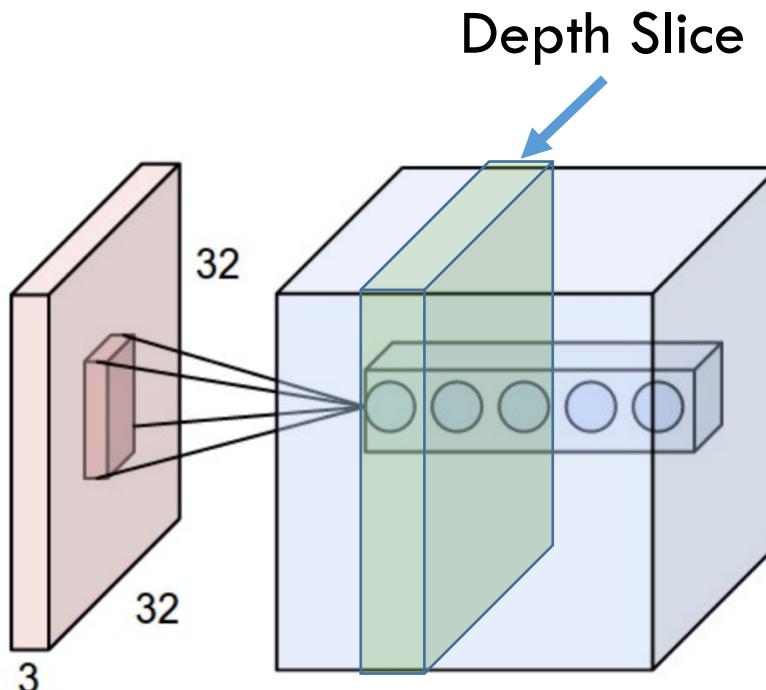
Image

4		

Convolved
Feature

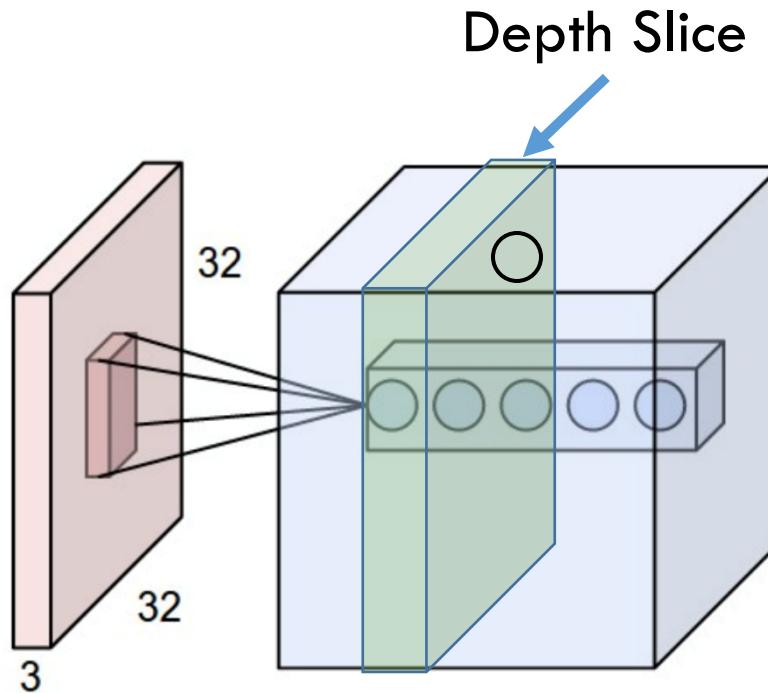
CONV: Parameter Sharing

- Key assumption: If a filter is useful for one region, it should also be useful for another region
- Denote a single 2D slice of depth of a layer as **depth slice**



CONV: Parameter Sharing

- Then, all neurons in each depth slice use the same weight and bias parameters!



CONV: Parameter Sharing

- Number of parameters is reduced!
- Example:
 - Say the number of filters is M (= Layer Depth)
 - Then, this layer will have $M * (3 * 3 * 3 + 1)$ parameters
- Gradients will get added up across neurons of a depth slice

CONV: Parameter Sharing

- AlexNet's first layer has $11 \times 11 \times 3$ sized filters 96 in number. The filter weights are plotted below:



- Intuition: If capturing an edge is important, then important everywhere

Example: CONV Layer Computation

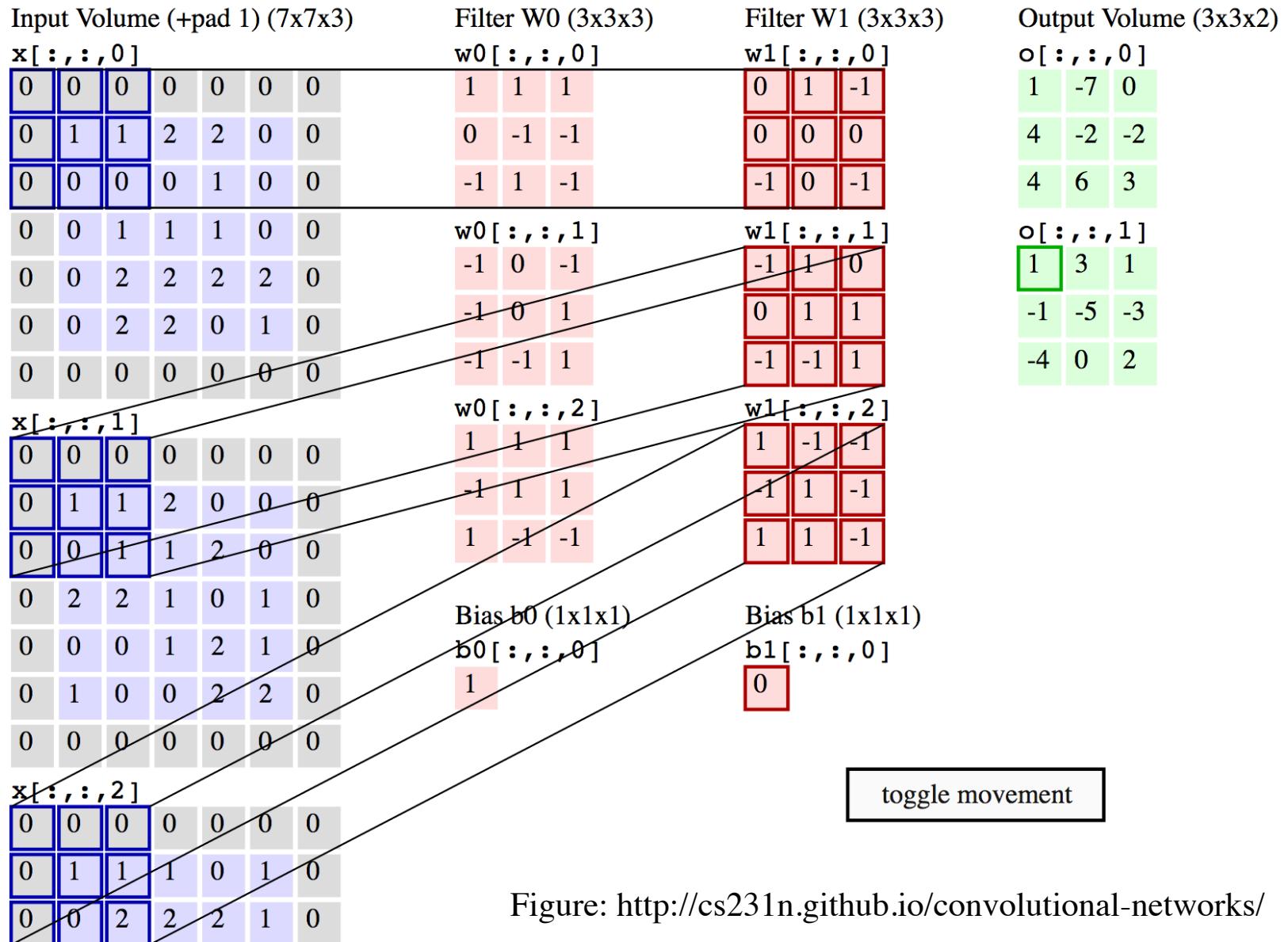


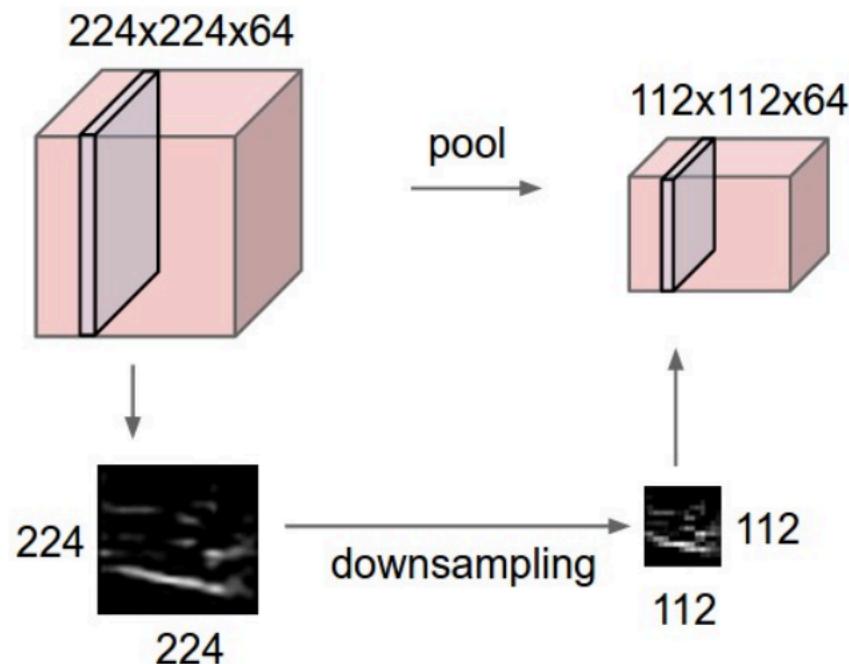
Figure: <http://cs231n.github.io/convolutional-networks/>

The Pooling Layer: POOL

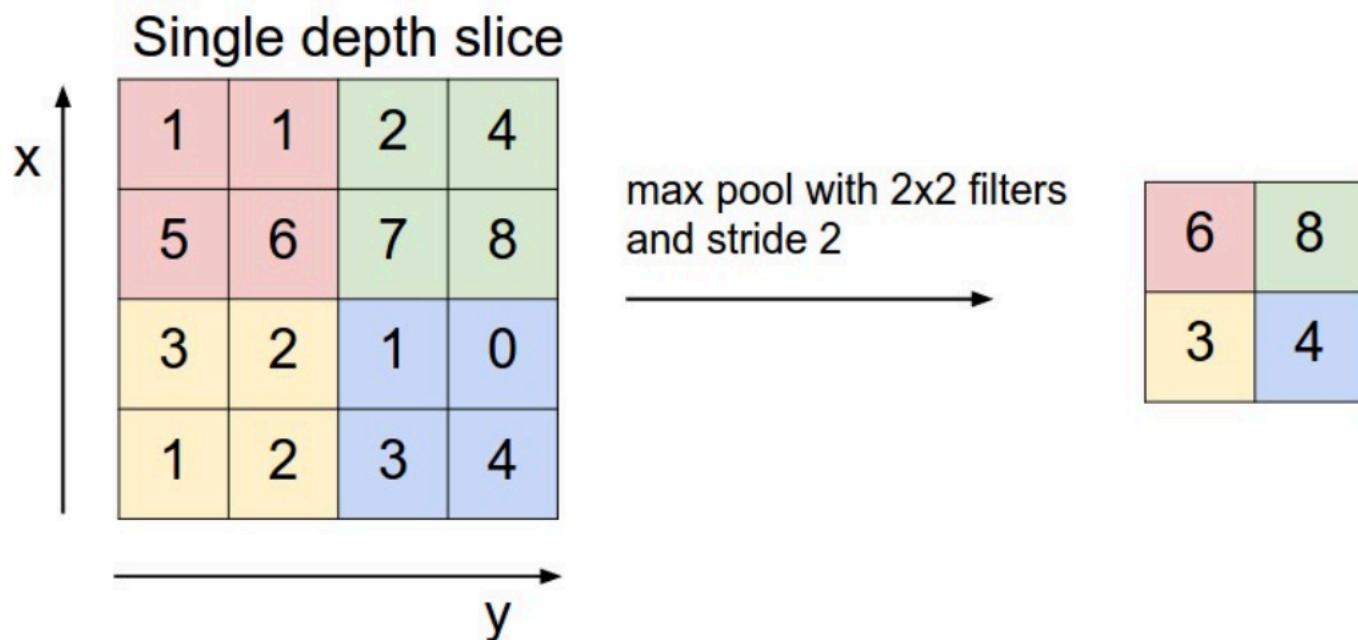
- Vastly more simpler than CONV
- Reduce the **spatial** size by using a MAX or similar operation
- Operate independently for each depth slice

POOL: Example

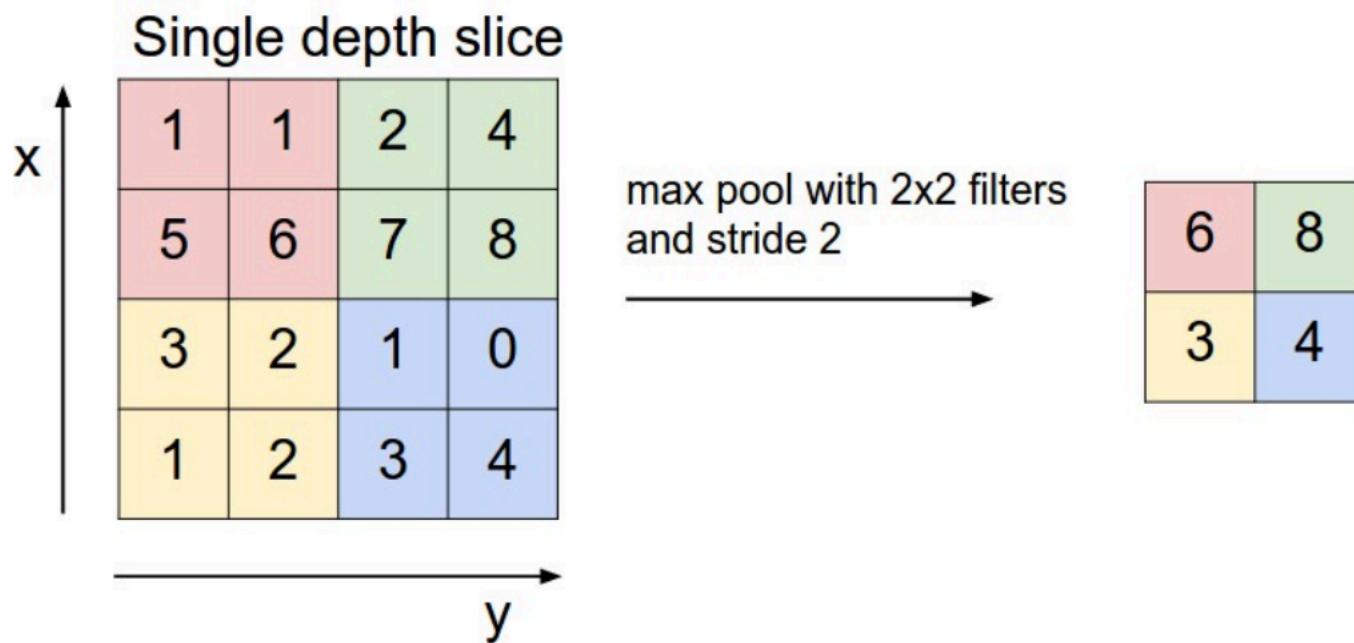
- Input depth is retained



POOL: Example



POOL: Example



- Recent research is showing that you may not need a pooling layer

Fully Connected Layer: FC

- Essentially a fully connected layer
- Already seen while discussing feedforward neural networks

CNN in the Browser

- Dataset: CIFAR-10
- <http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html>

Summary

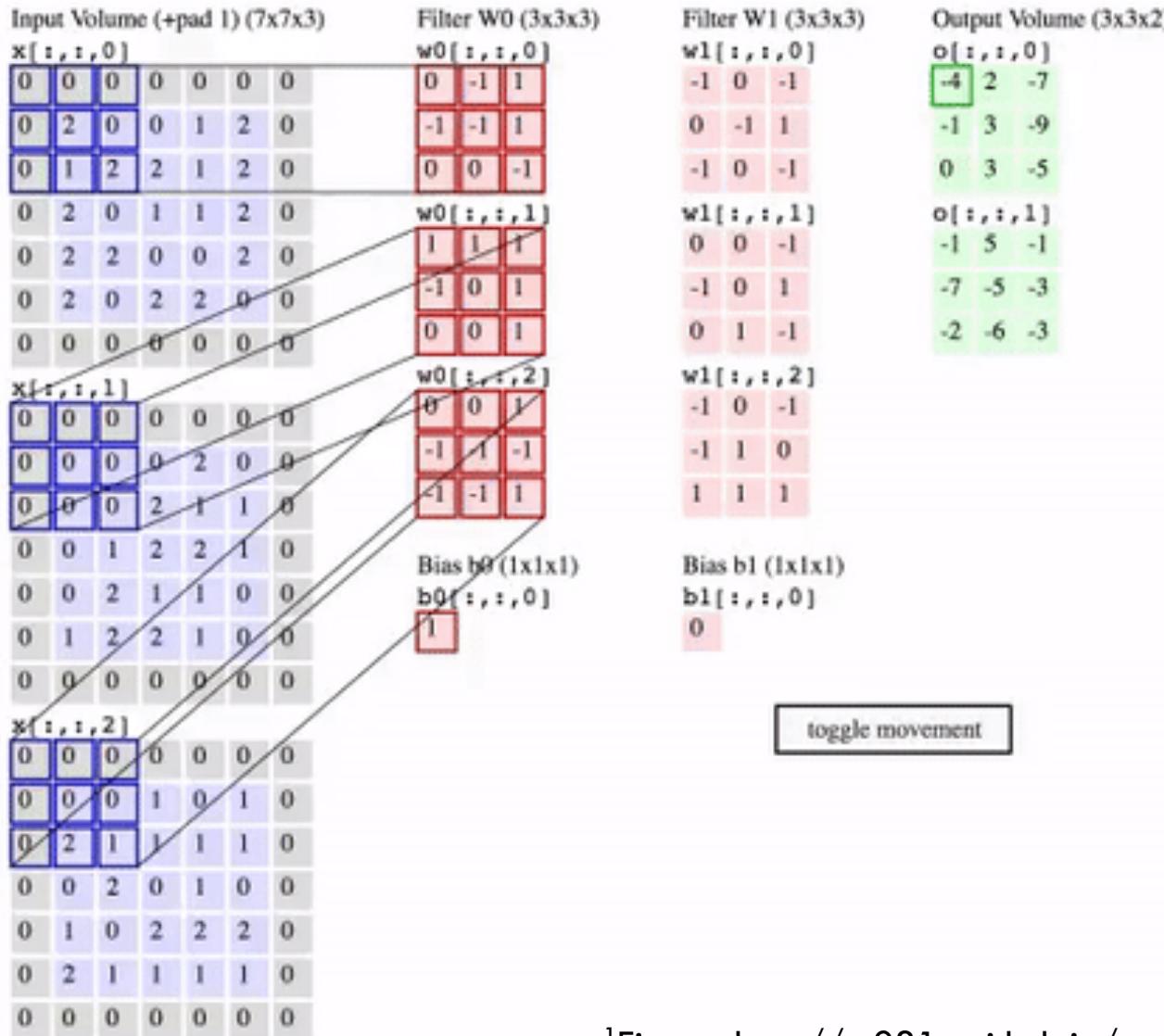
- Feedforward neural nets can do better than linear classifiers (saw this for a low-dimensional small synthetic example)
- CNN have been very effective in image related applications.
- Exploit specific properties of images
 - Hierarchy of features
 - Locality
 - Spatial invariance
- Lots of **design choices** that have been empirically validated and are intuitive. Still, there is room for improvement.

Appendix

Naming: Why ‘Neural’

- Historical
- Let $f(x) = w \cdot x + b$
- Perceptron from 1957: $h(x) = \begin{cases} 0, & f(x) < 0 \\ 1, & \text{otherwise} \end{cases}$
- Update rule was $w_{k+1} = w_k + \alpha(y - h(x))x$ similar to gradient update rules we see today
- Passing the score through a sigmoid was likened to how a neuron fires
 - Firing rate = $\frac{1}{1+e^{-yf(x)}}$

Naming: Why ‘Convolution’



The name ‘convolution’ comes from the convolution operation in signal processing that is essentially a matrix matrix product.

Naming: Why ‘Convolution’

