Deep Learning with TensorFlow

Himanshu Mittal
himanshu.mittal224@gmail.com

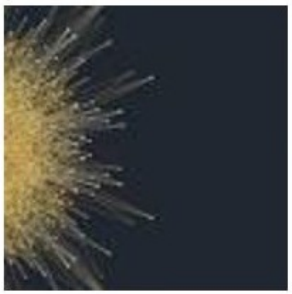
Department of Computer Science
Jaypee Institute of Information Technology, Noida

March 24, 2018

Outline

Outline

2016-17: Year of Deep Learning



2016: The Year That Deep Learning Took Over the Internet
WIRED - Dec 25, 2016
The project is still in the early stages, but it hints at the widespread impact of **deep learning** over past year. In 2016, this very old but newly ...

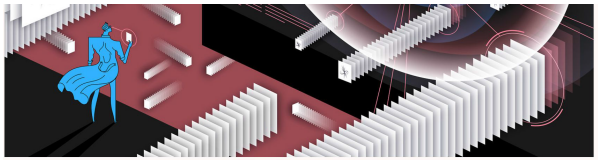


Illustration by Sandbox Studio, Chicago with Ana Kova

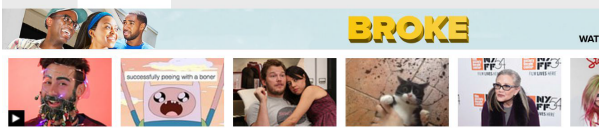
Deep learning takes on physics

12/06/16 | By Molly Olmstead

Can the same type of technology Facebook uses to recognize faces also recognize particles?

BuzzFeedNEWS

News Videos Quizzes Tasty DIY More Get Our Ne



This Is Why A Computer Winning At Go Is Such A Big Deal

People didn't think this would happen for at least 10 years; it's a sign of how far artificial intelligence has come.

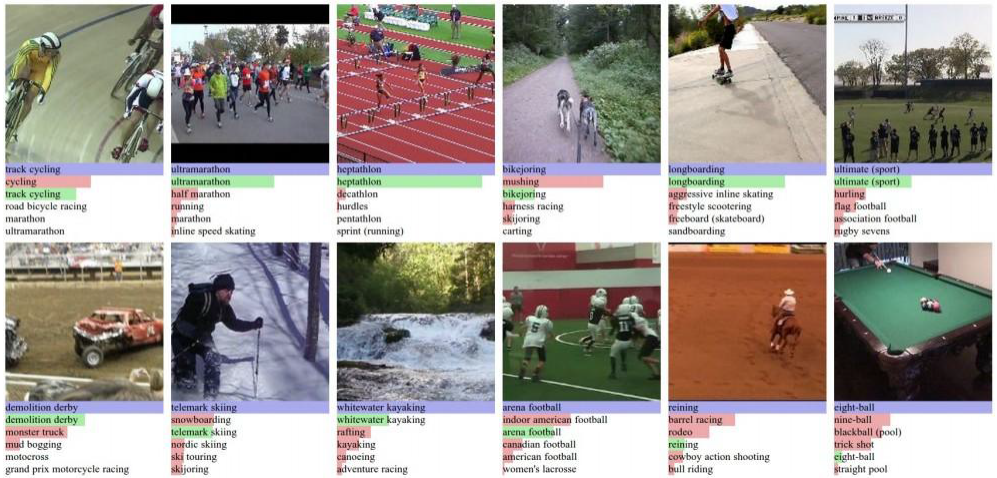
posted on Mar. 14, 2016, at 8:46 a.m.

A Primer On Deep Learning
Forbes - 18-Dec-2017
Fortunately, after a half-century of research, that gap between our expectations and reality is finally closing, thanks to **deep learning**, a more advanced type of machine learning that's capable of generating human-like insight. Here, I will give an introduction to **deep learning** and explore the potential of this ...

Deep Learning Success

- Image Classification
- Machine Translation
- Speech Recognition
- Game Playing

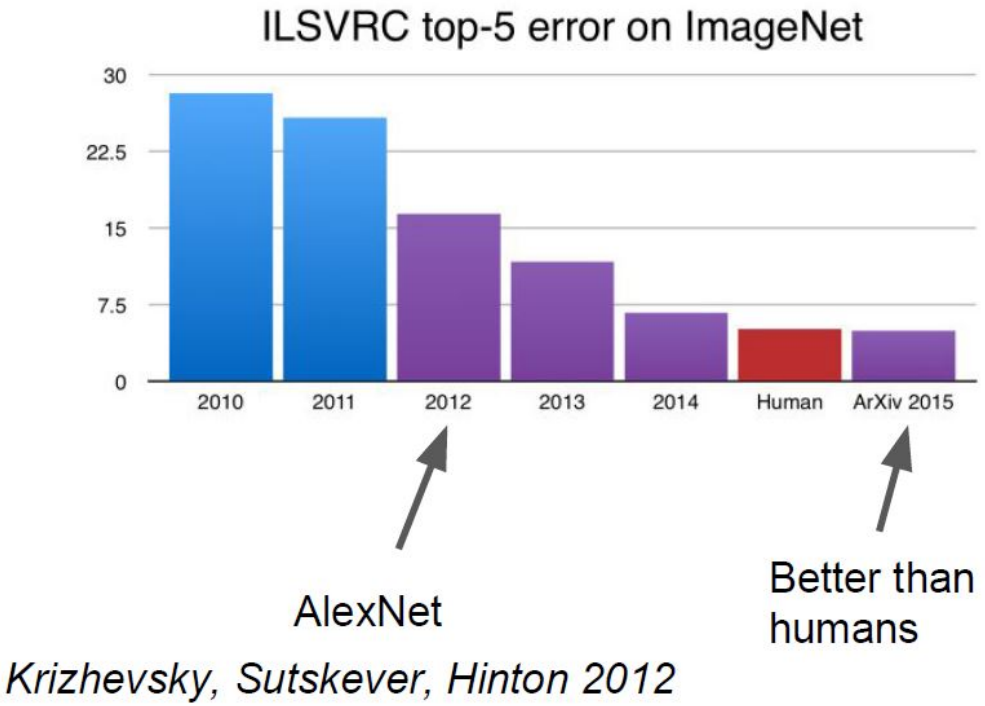
. . . and many, many more



Deep Learning Success

- Image Classification
- Machine Translation
- Speech Recognition
- Game Playing

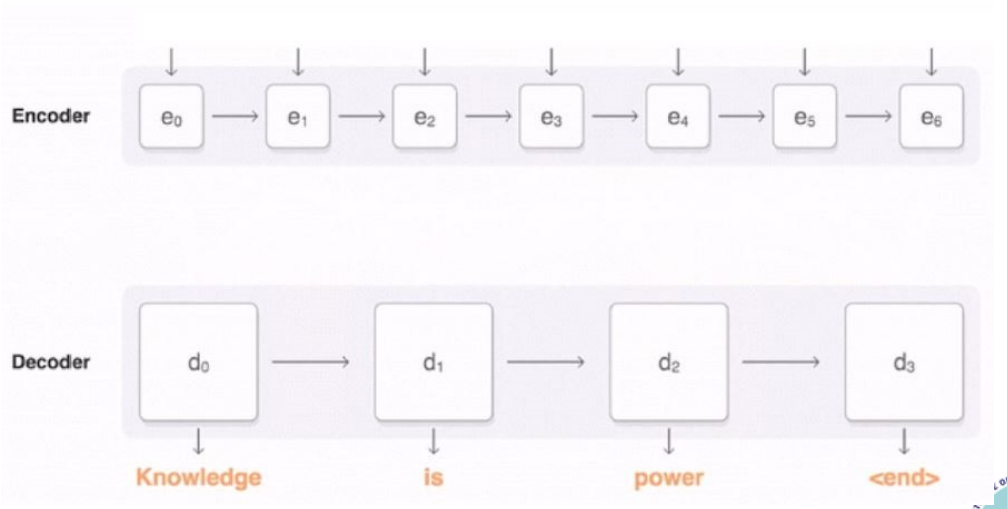
. . . and many, many more



Deep Learning Success

- Image Classification
- Machine Translation
- Speech Recognition
- Game Playing

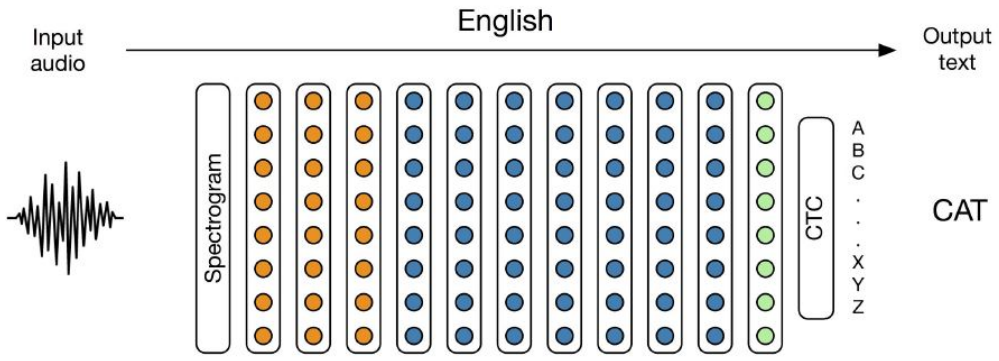
. . . and many, many more



Deep Learning Success

- Image Classification
- Machine Translation
- Speech Recognition
- Game Playing

. . . and many, many more



Deep Learning Success

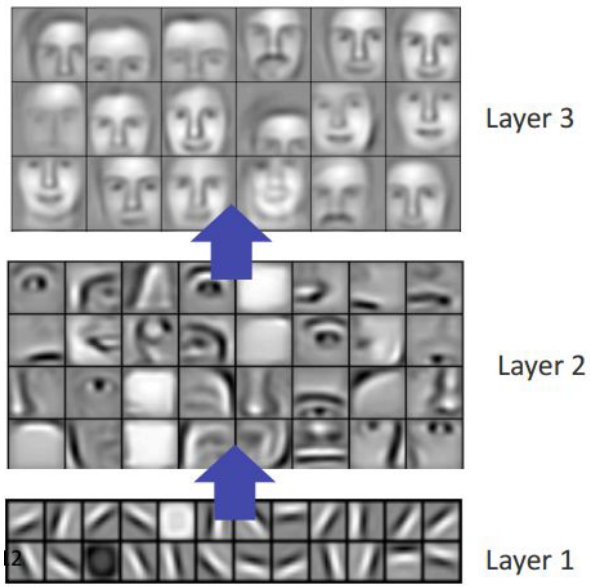
- Image Classification
- Machine Translation
- Speech Recognition
- Game Playing

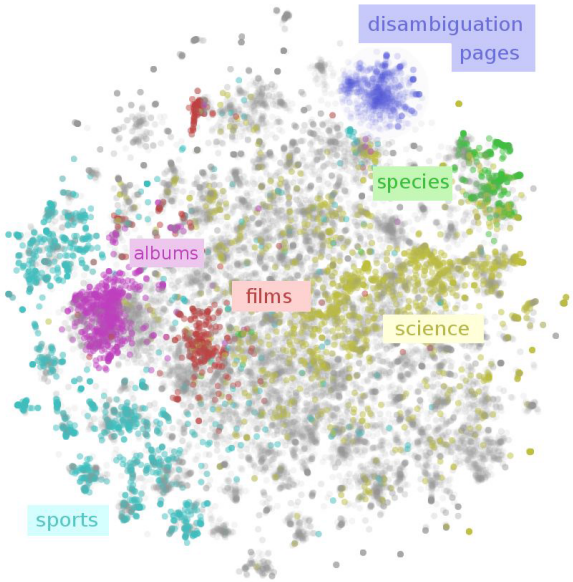
. . . and many, many more

Outline

Why Deep Learning?

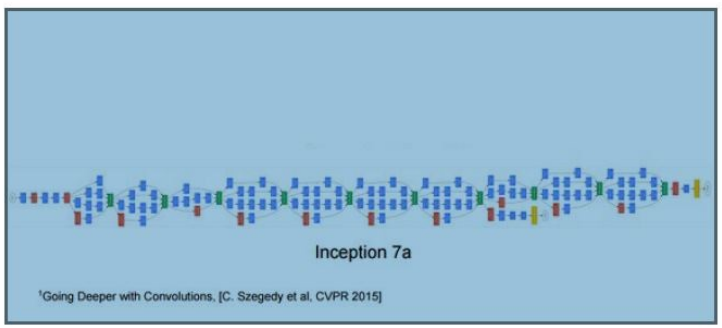
- Hand-Engineered Features vs. Learned features



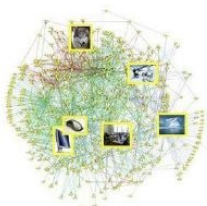


Why Now?

- Large Datasets
- GPU Hardware Advances + Price Decreases
- Improved Techniques



WIKIPEDIA



IMAGENET

The Big Players

Startups


vicarious
enlitic
clarifai
SKYMIND
SIGNALSENSE
ersatz labs
nnaisense
cortica
In Every Image
Numenta
sentient
OpenAI
MetaMind
DEEPMIND
AlchemyAPI
An IBM Company
wit.ai
DNNresearch

Outline

TensorFlow

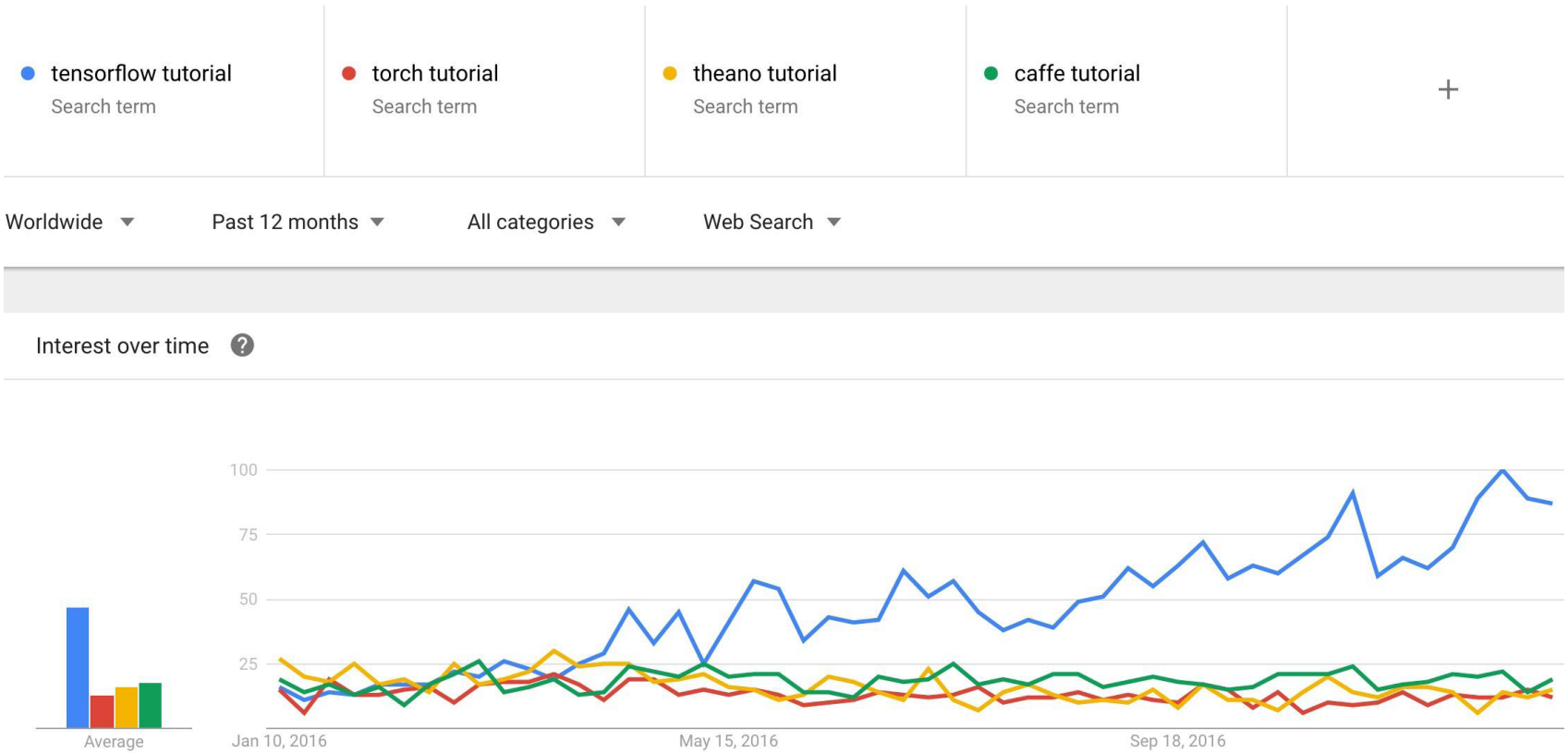
- Open source software library for numerical computation using data flow graphs
- Originally developed by Google Brain Team to conduct machine learning and deep neural networks research
- General enough to be applicable in a wide variety of other domains as well
- TensorFlow provides an extensive suite of functions and classes that allow users to build various models from scratch

Why TensorFlow?

- Python API
- Portability: deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API
- Auto-differentiation (no more taking derivatives by hand)
- Awesome projects already using TensorFlow
- Companies using Tensorflow
  - Google
  - OpenAI
  - DeepMind
  - Snapchat
  - Uber
  - Airbus
  - eBay
  - Dropbox
  - A bunch of startups

Why TensorFlow?

- TF is not the only deep learning library



Outline

TensorFlow Example

$$a = (b + c) * (c + 2) \tag{1}$$

- This function can be broken down into following operations:

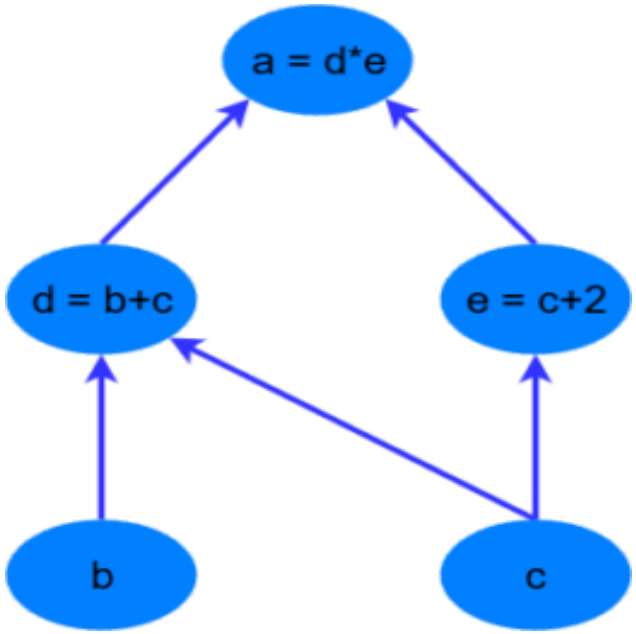
$$d = b + c \tag{2}$$

$$e = c + 2 \tag{3}$$

$$a = d * e \tag{4}$$

- Powerful idea in expressing the equation this way:
  - Two of the computations d and e can be performed in parallel
  - This can give us significant gains in computational times
  - These gains are a must for big data applications and deep learning

Data Flow Graphs



- Data flows between different nodes in the graph are tensors which are multi-dimensional data arrays.

What's a tensor?

- An n-dimensional array
  - 0-d tensor: scalar (number)
  - 1-d tensor: vector
  - 2-d tensor: matrix
  - and so on

Simple TensorFlow example: Declaration Phase

$$a = (b + c) * (c + 2) \tag{5}$$

Variable and Constant Declaration Code

```
import tensorflow as tf

# first, create a TensorFlow constant
const = tf.constant(2.0, name="const")

# create TensorFlow variables
b = tf.Variable(2.0, name='b')
c = tf.Variable(1.0, name='c')
```

Simple TensorFlow Example Contd..

TensorFlow Operations Code

```
# now create some operations
d = tf.add(b, c, name='d')
e = tf.add(c, const, name='e')
a = tf.multiply(d, e, name='a')
```

Initialization Code

- Setup an object to initialise the variables and the graph structure.

```
# setup the variable initialisation
init_op = tf.global_variables_initializer()
```

Simple TensorFlow Example Contd..

Session Code

- To run the operations between the variables, we need to start a TensorFlow session. The TensorFlow session is an object where all operations are run.

```
# start the session
with tf.Session() as sess:
    # initialise the variables
    sess.run(init_op)
    # compute the output of the graph
    a_out = sess.run(a)
    print("Variable a is {}".format(a_out))
```

TensorFlow Placeholder

Placeholder

- To get the value of the array b during the tf.Session(), the variables are declared using the tf.placeholder.

```
# create TensorFlow variables
b = tf.placeholder(tf.float32, name='b')
```

- The value may be passed as:
  a_out = sess.run(a, feed_dict={b:5})

Visualize Graph with TensorBoard

```
import tensorflow as tf
a = tf.constant(2)
b = tf.constant(3)
x = tf.add(a, b)
with tf.Session() as sess:
#Create the summary writer after graph definition and before
running your session
# add this line to use TensorBoard.
   writer = tf.summary.FileWriter('C:\\Graph')
#Path where you want to keep your event files
   x_out= sess.run(x)
   print(x_out)
   writer.add_graph(sess.graph)
   writer.close() # close the writer when you're done using it
```

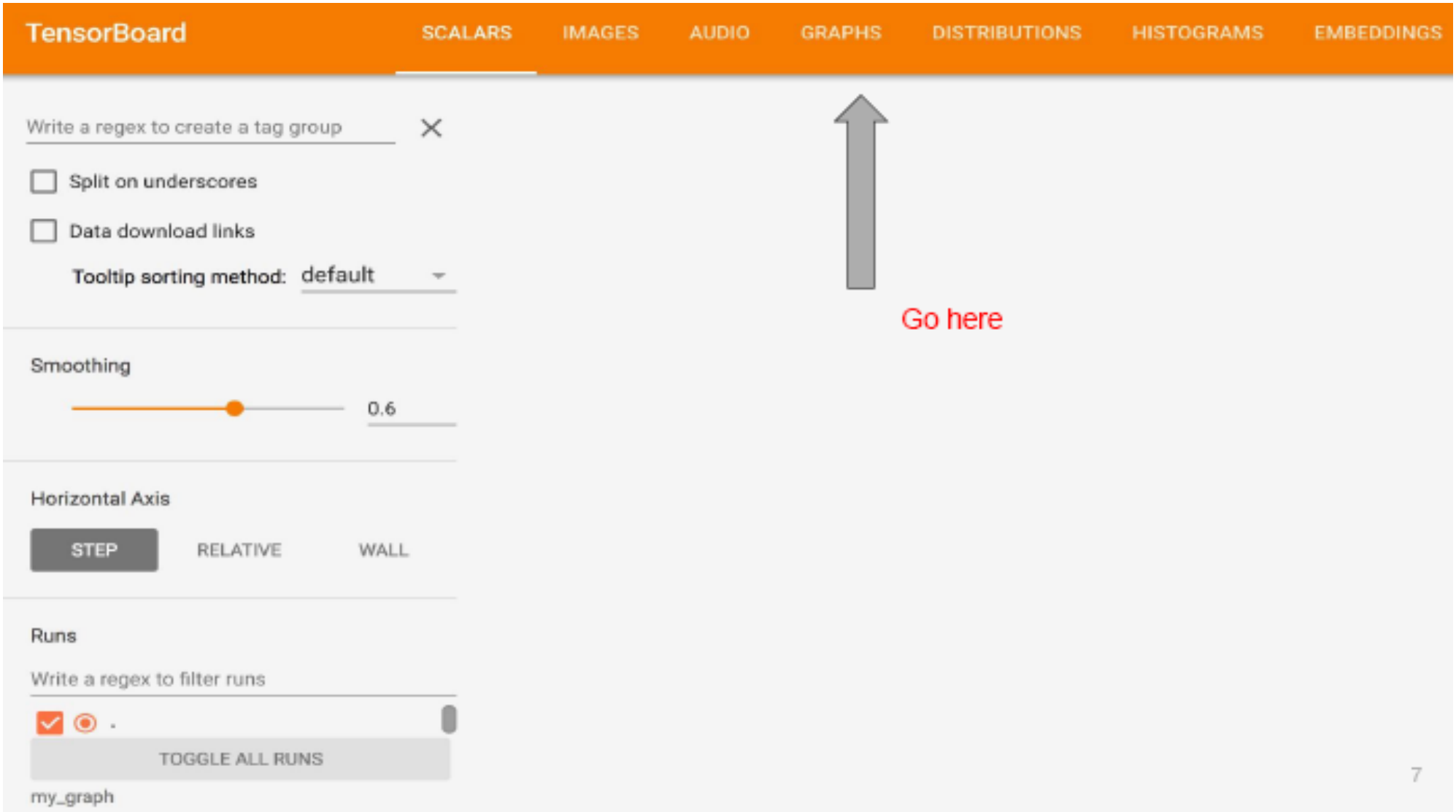
Run it

Go to terminal, run:

$ python [yourprogram].py

$ tensorboard - -logdir C:\Graph

Then open your browser and go to: http://localhost:6006/

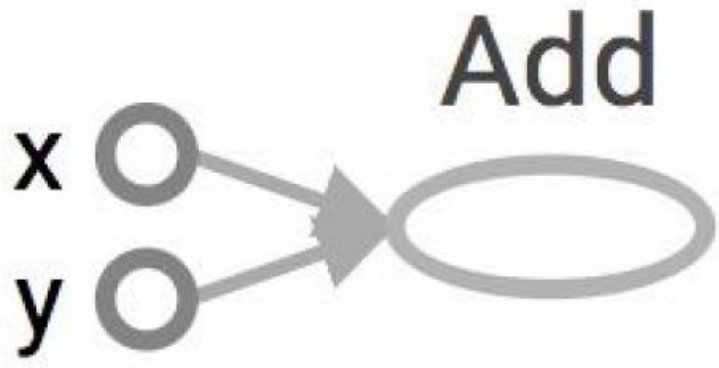

TensorBoard
SCALARS
IMAGES
AUDIO
GRAPHS
DISTRIBUTIONS
HISTOGRAMS
EMBEDDINGS
Write a regex to create a tag group
Split on underscores
Data download links
Tooltip sorting method: default
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
TOGGLE ALL RUNS
my_graph
Go here
7

More about Data Flow Graphs

```
import tensorflow as tf
a = tf.add(3, 5)
```

- Why x, y?
- TF automatically names the nodes when you don't explicitly name them
  - x=3
  - y=5

Visualized by TensorBoard

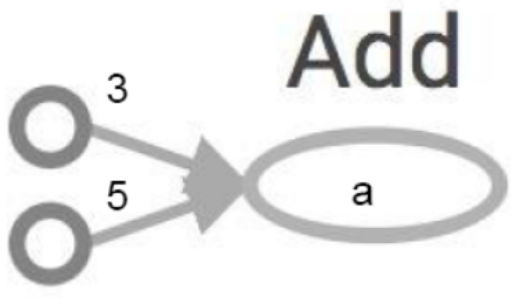


More about Data Flow Graphs

```
import tensorflow as tf
a = tf.add(3, 5)
```

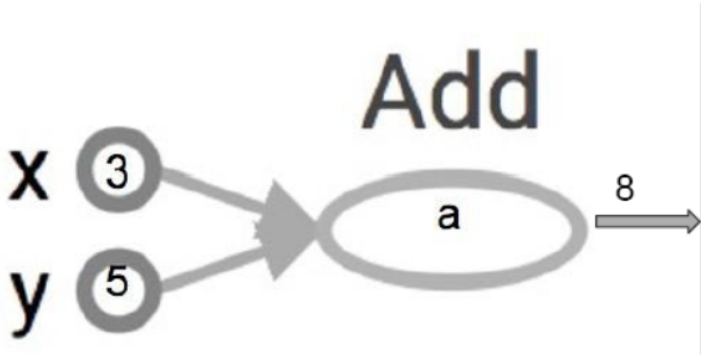
- Nodes: operators, variables, and constants
- Edges: tensors

Interpreted?



How to get the value of a?

- Create a session, assign it to variable sess so we can call it later
- Within the session, evaluate the graph to fetch the value of a

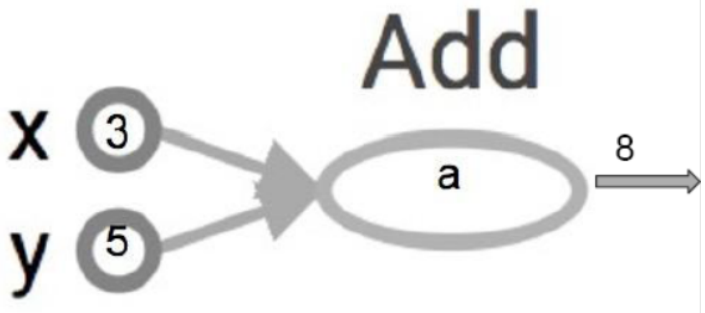



```
import tensorflow as tf
a = tf.add(3, 5)
sess = tf.Session()
print sess.run(a)      >> 8
sess.close()
```

How to get the value of a?

- Create a session, assign it to variable sess so we can call it later
- Within the session, evaluate the graph to fetch the value of a

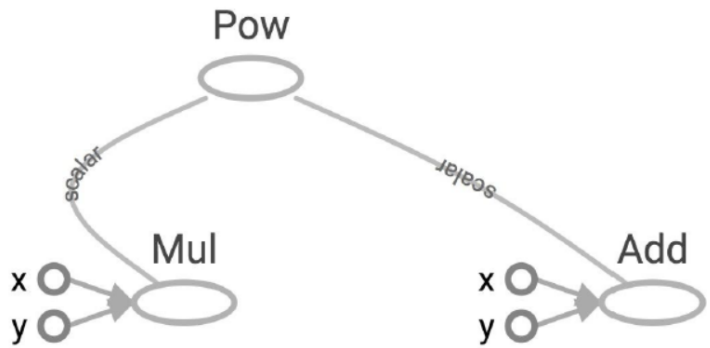
```
import tensorflow as tf
a = tf.add(3, 5)
sess = tf.Session()
with tf.Session() as sess:
    print sess.run(a)
sess.close()
```



More Graphs

```
import tensorflow as tf
x = 2
y = 3
op1 = tf.add(x, y)
op2 = tf.multiply(x, y)
op3 = tf.pow(op2, op1)
with tf.Session() as sess:
    op3 = sess.run(op3)
```

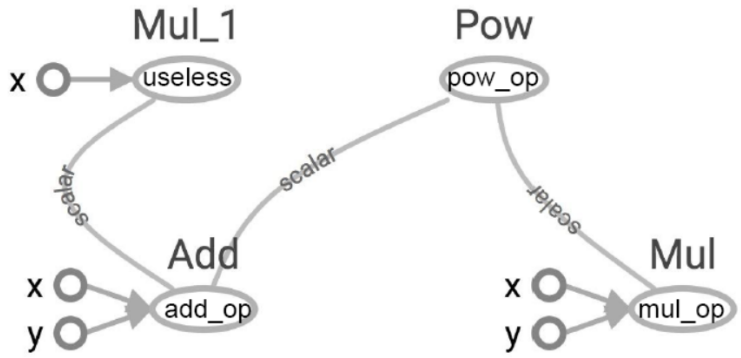
Visualized by TensorBoard



Sub-Graphs

```
import tensorflow as tf
x = 2
y = 3
add_op = tf.add(x, y)
mul_op = tf.multiply(x, y)
useless = tf.multiply(x, add_op)
pow_op = tf.pow(add_op, mul_op)
with tf.Session() as sess:
    z = sess.run(pow_op)
```

Visualized by TensorBoard

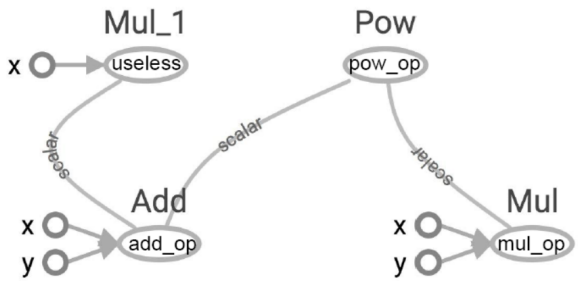


Because we only want the value of pow_op and pow_op doesn't depend on useless, session won't compute value of useless

—–> save computation

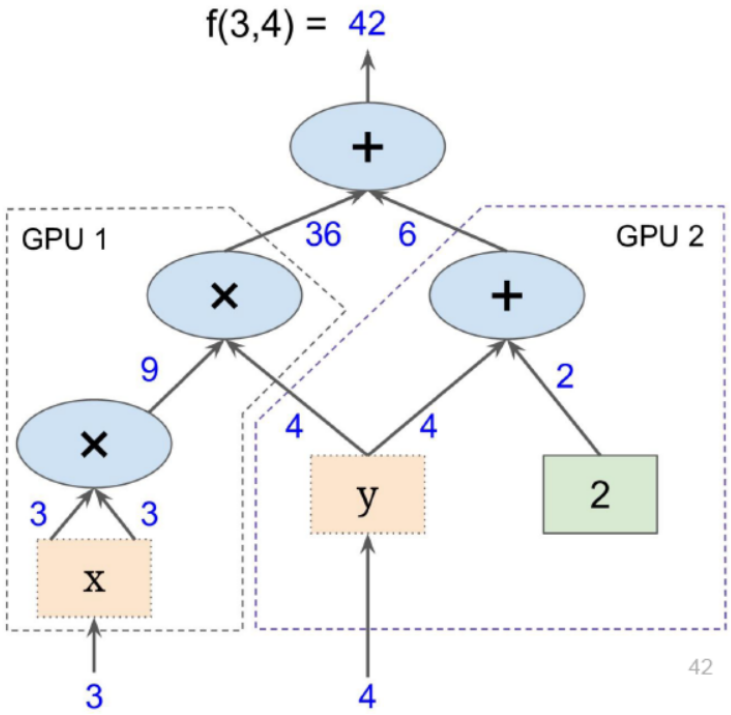
Sub-Graphs

```
import tensorflow as tf
x = 2
y = 3
add_op = tf.add(x, y)
mul_op = tf.multiply(x, y)
useless = tf.multiply(x, add_op)
pow_op = tf.pow(add_op, mul_op)
with tf.Session() as sess:
  z, not_useless = sess.run([pow_op, useless])
```



Sub-Graphs

- Possible to break graphs into several chunks and run them parallelly across multiple CPUs, GPUs, or devices



More Constants

- a = tf.constant([2, 2], name="a") <— 1d Tensor
- b = tf.constant([[0, 1], [2, 3]], name="b") <— 2d Tensor
- Tensors filled with a specific value
  - tf.zeros([2, 3], tf.int32) ==> [[0, 0, 0], [0, 0, 0]]
  - Let input_tensor is [0, 1], [2, 3], [4, 5]]

    tf.zeros_like(input_tensor) ==> [[0, 0], [0, 0], [0, 0]]
  - tf.fill([2, 3], 8) ==> [[8, 8, 8], [8, 8, 8]]

More Constants

- Constants as sequences
  - tf.linspace(start, stop, num)

    tf.linspace(10.0, 13.0, 4) ==> [10.0 11.0 12.0 13.0]
  - tf.range(start, limit=None, delta=1, dtype=None, name='range')

    Let 'start' is 3, 'limit' is 18, 'delta' is 3

    tf.range(start, limit, delta) ==> [3, 6, 9, 12, 15]

    Let 'limit' is 5

    tf.range(limit) ==> [0, 1, 2, 3, 4]

Tensor objects are not iterable

for _ in tf.range(4): ==># TypeError

TensorFlow Data Types

Data type	Python type	Description
DT_FLOAT	tf.float32	32 bits floating point.
DT_DOUBLE	tf.float64	64 bits floating point.
DT_INT8	tf.int8	8 bits signed integer.
DT_INT16	tf.int16	16 bits signed integer.
DT_INT32	tf.int32	32 bits signed integer.
DT_INT64	tf.int64	64 bits signed integer.
DT_UINT8	tf.uint8	8 bits unsigned integer.
DT_UINT16	tf.uint16	16 bits unsigned integer.
DT_STRING	tf.string	Variable length byte arrays. Each element of a Tensor is a byte array.
DT_BOOL	tf.bool	Boolean.
DT_COMPLEX64	tf.complex64	Complex number made of two 32 bits floating points: real and imaginary parts.
DT_COMPLEX128	tf.complex128	Complex number made of two 64 bits floating points: real and imaginary parts.

Outline

Linear Regression & Gradient Descent

- Notation:
  - m : Number of training examples
  - x : Input variables (Features)
  - y: Output variables (Targets)
  - (x,y): Training Example (Represents 1 row on the table)
  - $(x^{(i)}, y^{(i)})$: i^{th} training example (Represent's i^{th} row on the table)
  - n : Number of features (Dimensionality of the input)

Representation of the Hypothesis (Function):

- In this case, y is represented as a linear combination of the inputs (x)
- Which leads to:

$$h(\theta) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 \cdots + \theta_n x_n$$

where, θ_i are the parameters (also called weights).

- For convenience and ease of representation: $x_0 = 1$
- So that, the above equation becomes:

$$h(x) = \sum_{i=0}^{n} (\theta^T x)$$

- The objective now, is to 'learn' the parameters θ
- So that $h(x)$ becomes as close to 'y' at least for the training set.

Linear Regression & Gradient Descent

- Define a function that measures for each value of the theta's, how close the $h(x^{(i)})$ are to the corresponding $y^{(i)}$
- The 'cost function':

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_{(\theta)}(x^{(i)}) - y^{(i)})^2$$

- Choose the parameters so as to minimize $J(\theta)$

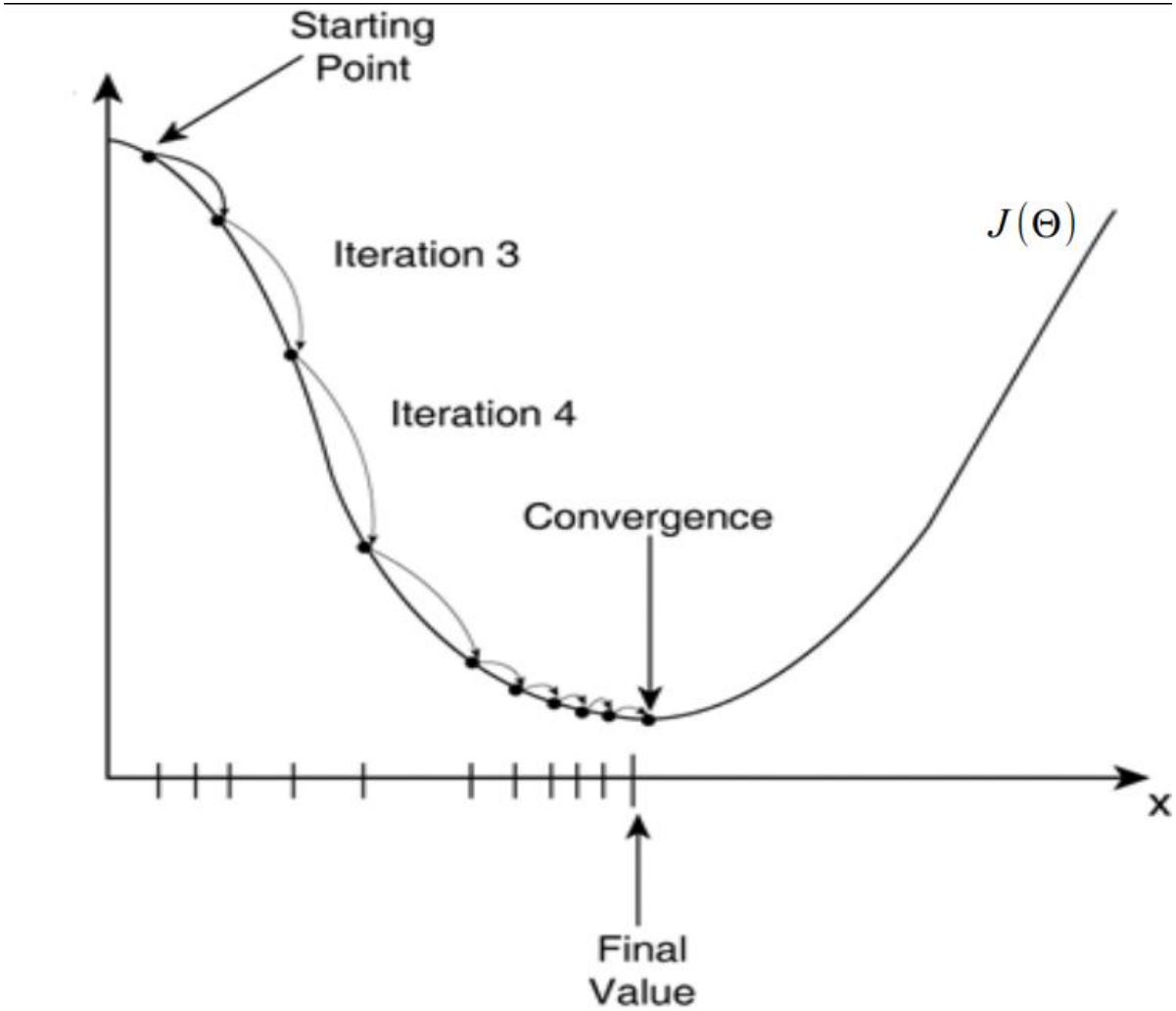
Linear Regression & Gradient Descent

- To minimize, Gradient Descent algorithm may be applied
- Gradient Descent starts off with some initial θ, and continually performs the following update:

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

- This update is simultaneously performed for all values of $j = 0, 1, \ldots, n$
- α is called the learning rate

Linear Regression & Gradient Descent



TensorFlow Code for Linear Regression & Gradient Descent

Import necessary libraries

```
import tensorflow as tf

import numpy

import matplotlib.pyplot as plt

rng = numpy.random
```

TensorFlow Code for Linear Regression & Gradient Descent

Parameters

learning_rate = 0.01

training_epochs = 1000

display_step = 50

Training Data

```
train_X = numpy.asarray([3.3,4.4,5.5,6.71,6.93,4.168,9.779,6.182,
                         7.59,2.167,7.042,10.791,5.313,7.997,5.654,9.27,3.1])

train_Y = numpy.asarray([1.7,2.76,2.09,3.19,1.694,1.573,3.366,
                         2.596,2.53,1.221,2.827,3.465,1.65,2.904,2.42,2.94,1.3])

n_samples = train_X.shape[0]
```

TensorFlow Code for Linear Regression & Gradient Descent

tf Graph Input

```
X = tf.placeholder("float")
Y = tf.placeholder("float")
```

Set model weights

```
W = tf.Variable(rng.randn(), name="weight")
b = tf.Variable(rng.randn(), name="bias")
```

TensorFlow Code for Linear Regression & Gradient Descent

Construct a linear model

```
pred = tf.add(tf.multiply(X, W), b)
```

Mean squared error

```
cost = tf.reduce_sum(tf.pow(pred-Y, 2))/(2*n_samples)
```

Gradient descent

```
optimizer =
tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

Initialize the variables

```
init = tf.global_variables_initializer()
```

Start training

```
with tf.Session() as sess:
    sess.run(init) # Run the initializer
# Fit all training data
    for epoch in range(training_epochs):
        for (x, y) in zip(train_X, train_Y):
            sess.run(optimizer, feed_dict={X: x, Y: y})
# Display logs per epoch step
        if (epoch+1) % display_step == 0:
            c = sess.run(cost, feed_dict={X: train_X, Y:train_Y})
            print("Epoch:", '%04d' % (epoch+1), "cost=",
                  "{:.9f}".format(c),"W=", sess.run(W), "b=",
                  sess.run(b))
    print("Optimization Finished!")
    training_cost = sess.run(cost, feed_dict=X: train_X, Y: train_Y)
    print("Training cost=", training_cost, "W=", sess.run(W),
          "b=", sess.run(b), '\n')
```

Start training cont...

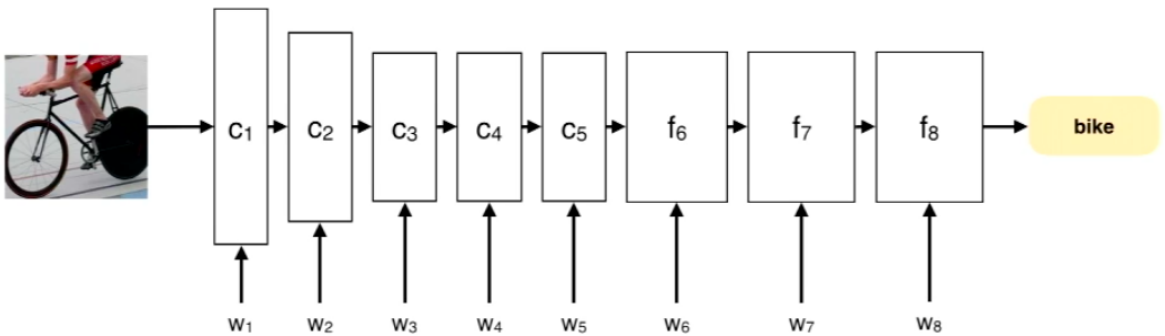
```
# Graphic display
plt.plot(train_X, train_Y, 'ro', label='Original data')
plt.plot(train_X, sess.run(W) * train_X + sess.run(b), label='Fitted
        line')
plt.legend()
plt.show()
```

Testing example

```
test_X = numpy.asarray([6.83, 4.668, 8.9, 7.91, 5.7, 8.7, 3.1, 2.1])
test_Y = numpy.asarray([1.84, 2.273, 3.2, 2.831, 2.92, 3.24, 1.35,
1.03])
print("Testing... (Mean square loss Comparison)")
testing_cost = sess.run(
    tf.reduce_sum(tf.pow(pred - Y, 2)) / (2 *test_X.shape[0]),
    feed_dict=X: test_X, Y: test_Y) # same function as cost above
print("Testing cost=", testing_cost)
print("Absolute mean square loss difference:", abs(
    training_cost - testing_cost))
plt.plot(test_X, test_Y, 'bo', label='Testing data')
plt.plot(train_X, sess.run(W) * train_X + sess.run(b), label='Fitted
line')
plt.legend()
plt.show()
```

Outline

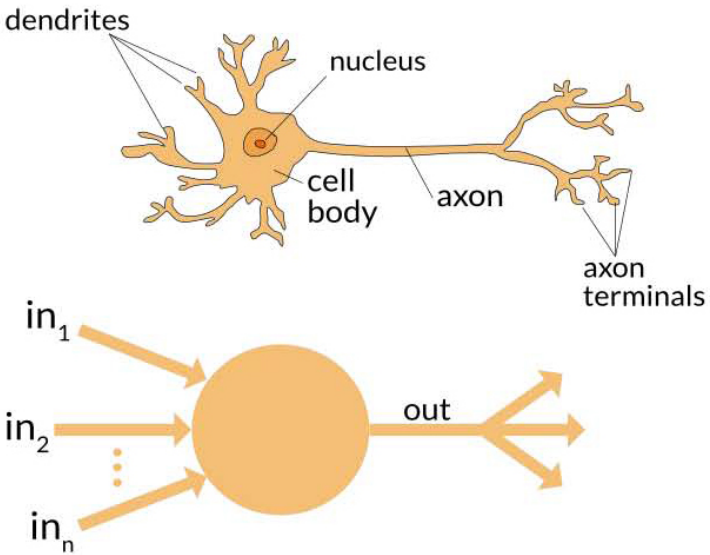
Deep Network: a magic box



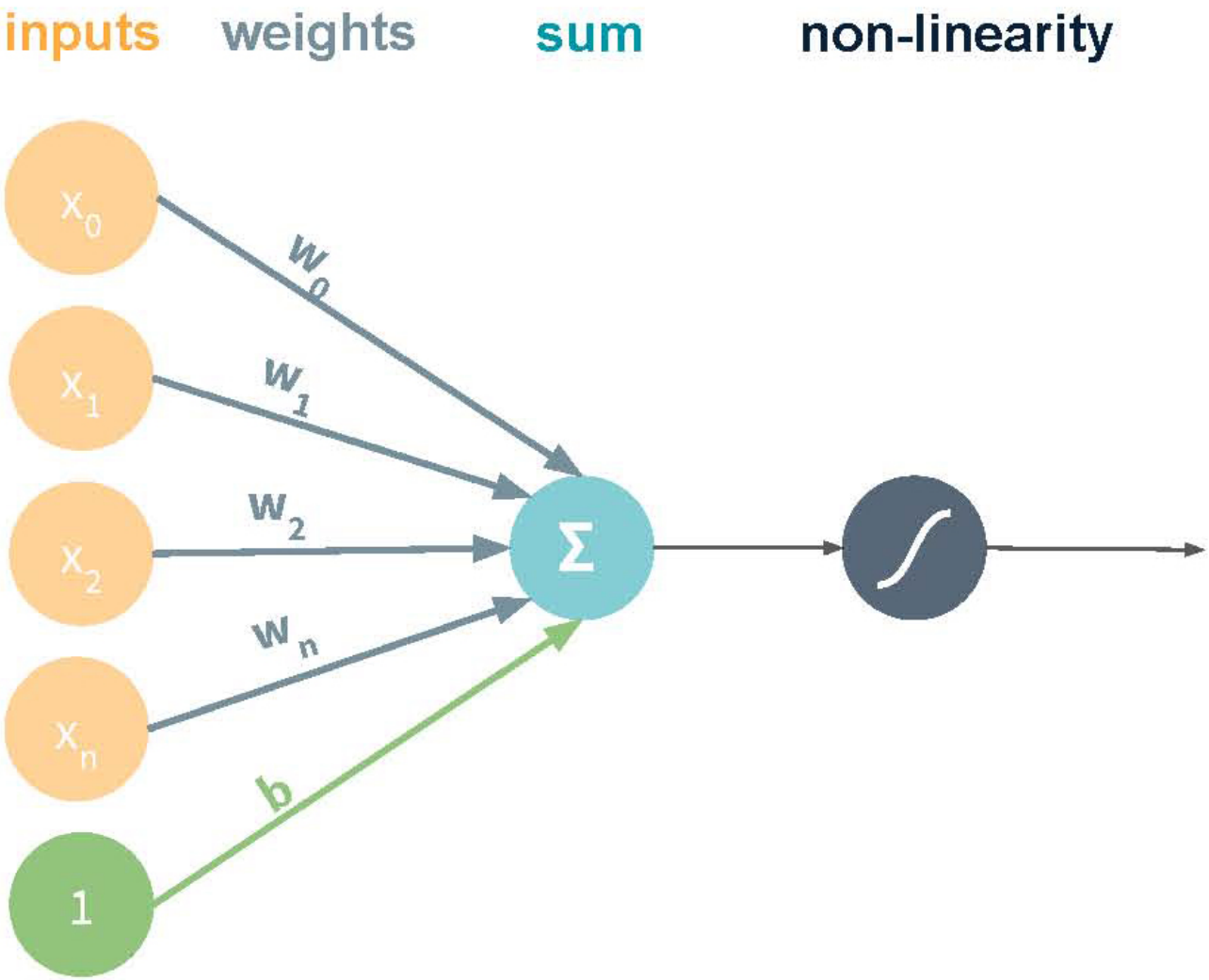
Outline

The Perceptron

- Invented in 1954 by Frank Rosenblatt
- Inspired by neurobiology

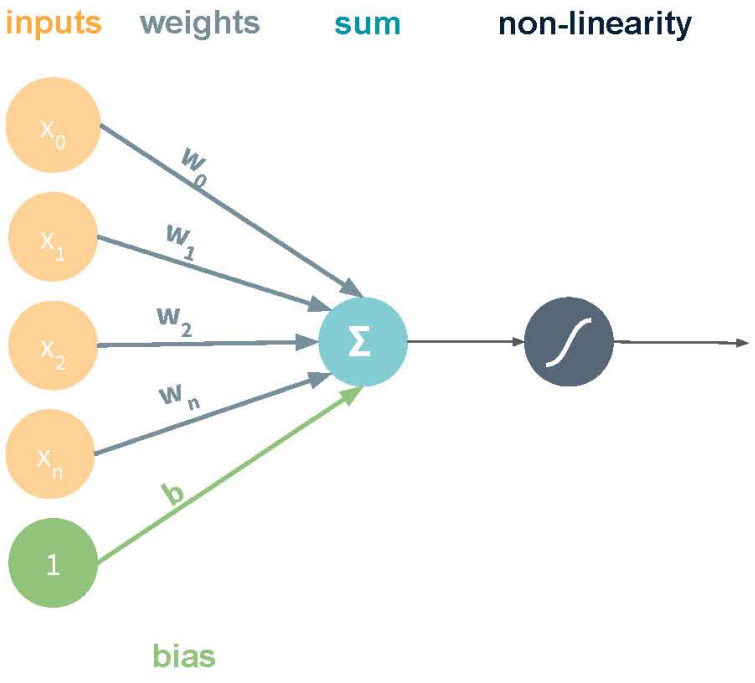


The Perceptron



Perceptron Forward Pass

$$output = g((\sum_{i=0}^{N} x_i * w_i) + b)$$

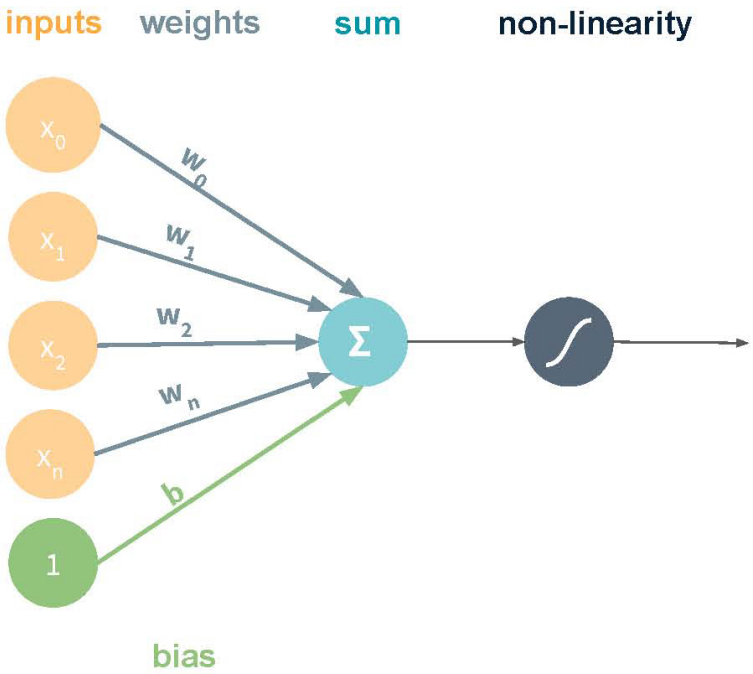


Perceptron Forward Pass

$$output = g(XW + b)$$

$$X = x_0, x_1, \ldots, x_n$$

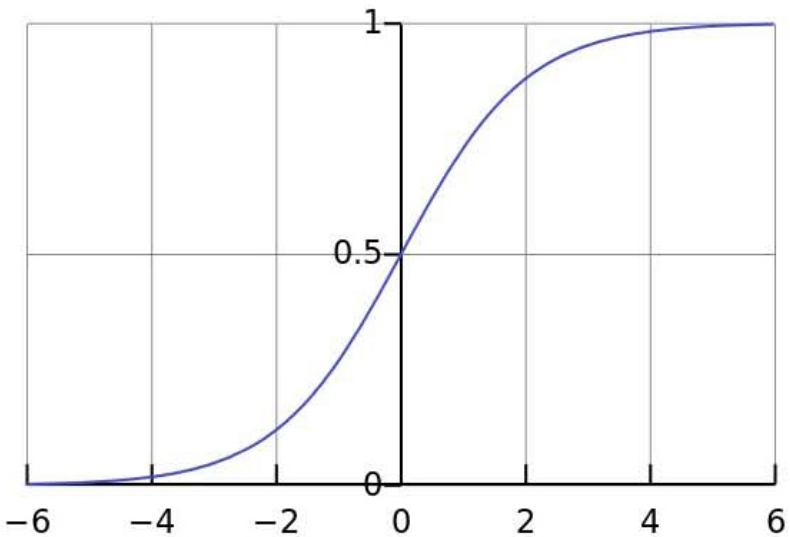
$$W = w_0, w_1, \ldots, w_n$$

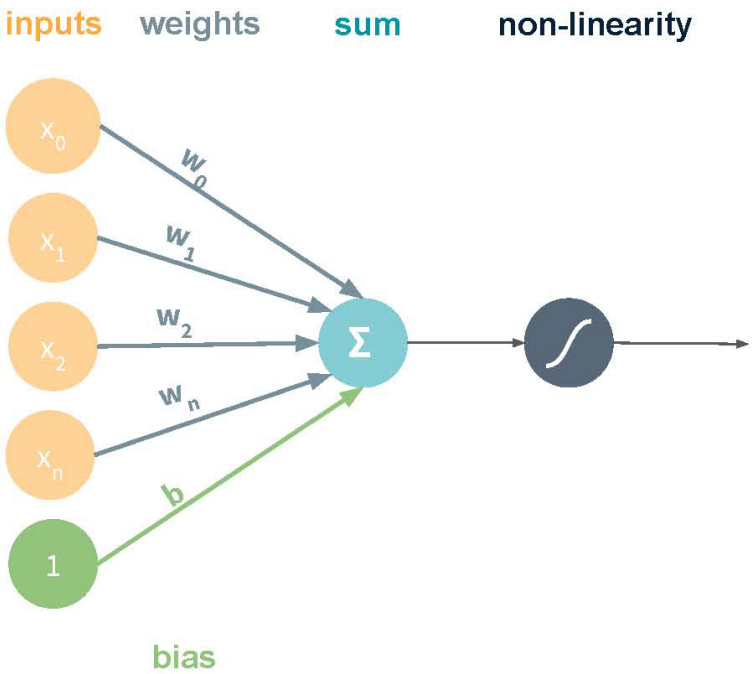


Sigmoid Activation

$$output = g(XW + b)$$

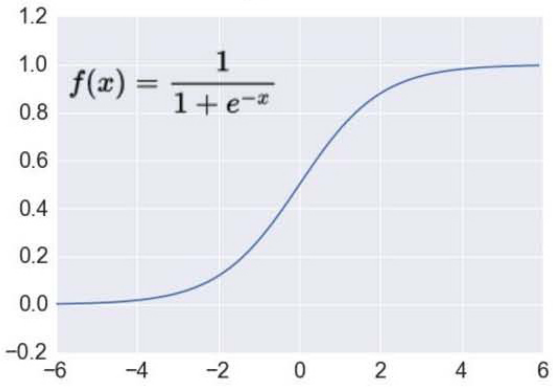
$$g(a) = \sigma(a) = \frac{1}{1 + e^{-a}}$$



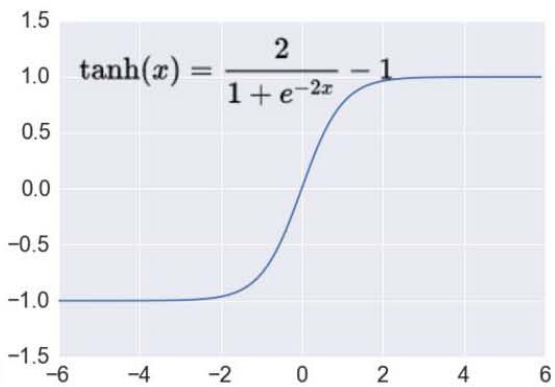


Common Activation Functions

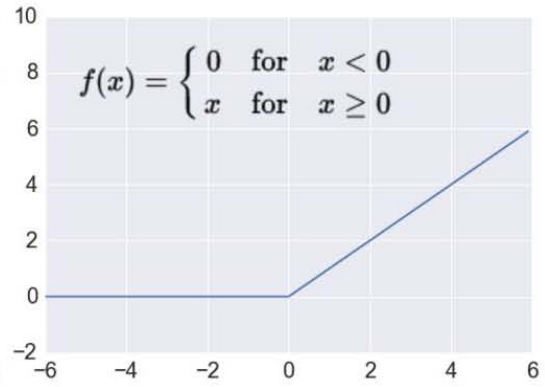
Sigmoid



TanH

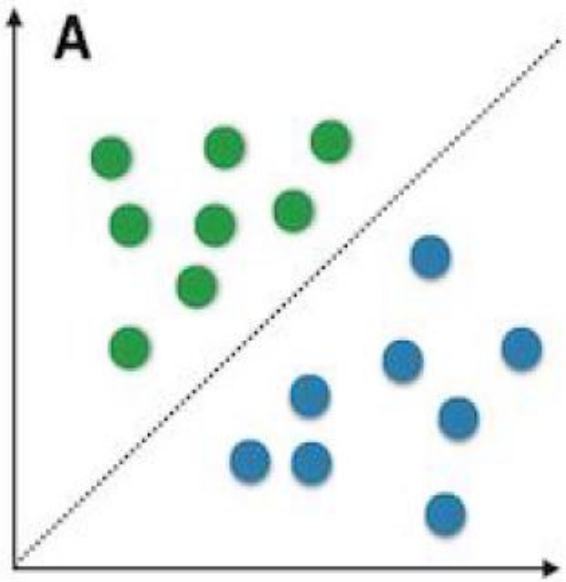


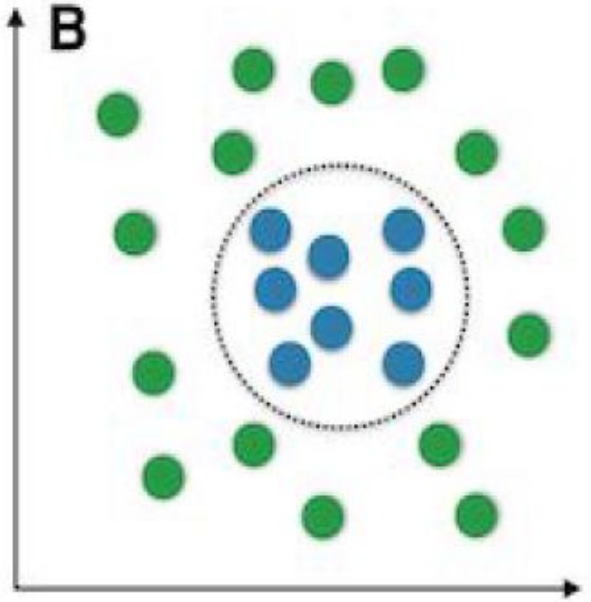
Rectified Linear Unit (ReLU)



Importance of Activation Functions

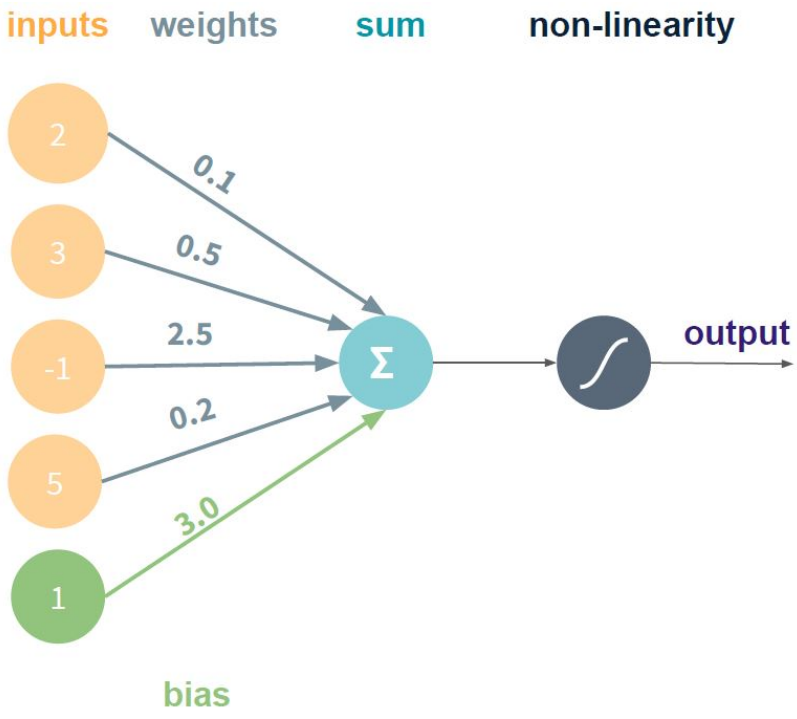
- Activation functions add non-linearity to our networks function
- Most real-world problems + data are non-linear





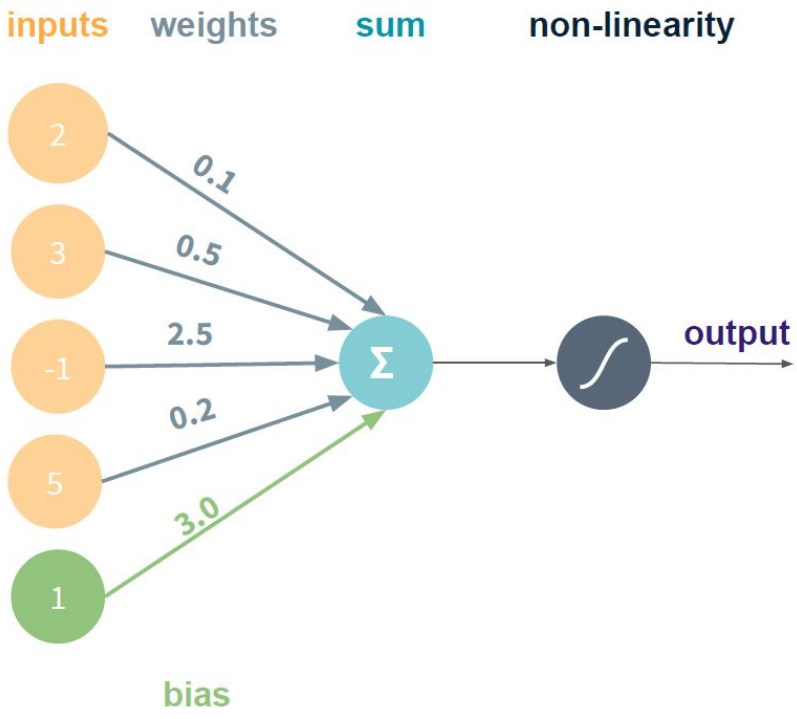
Perceptron Forward Pass

$$output = g(XW + b)$$



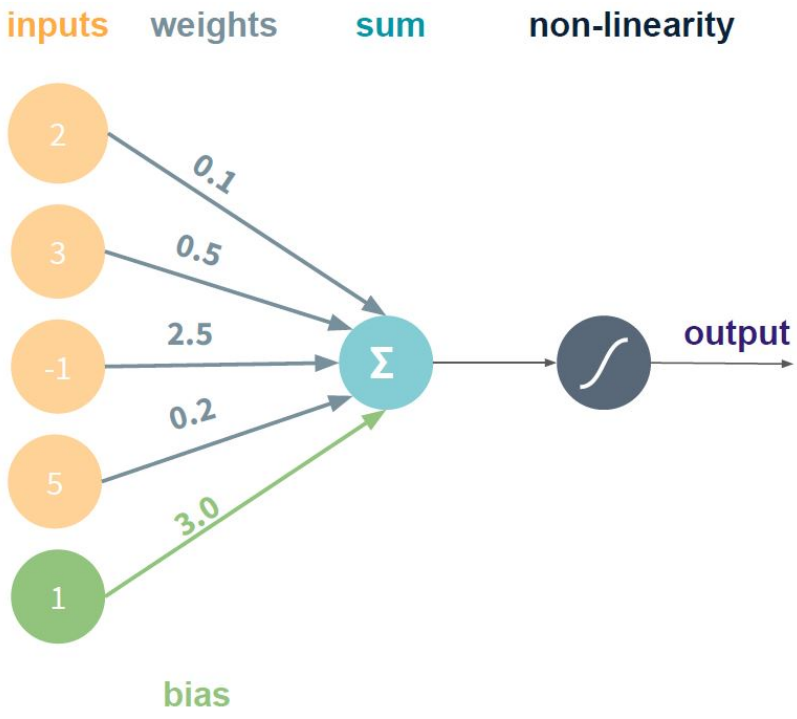
Perceptron Forward Pass

$$
\begin{aligned}
output = g(& \\
& (2 * 0.1) + \\
& (3 * 0.5) + \\
& (-1 * 2.5) + \\
& (5 * 0.2) + \\
& (1 * 3.0) \\
&)
\end{aligned}
$$



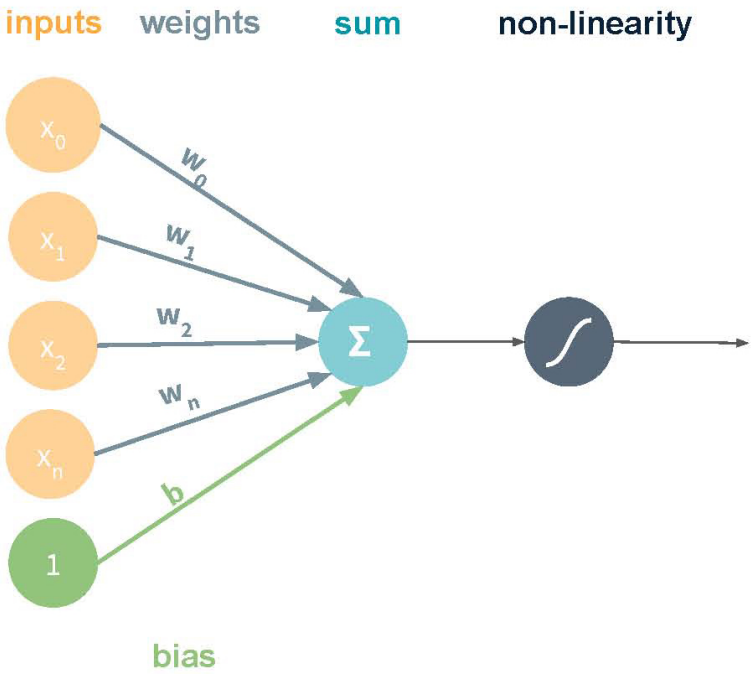
Perceptron Forward Pass

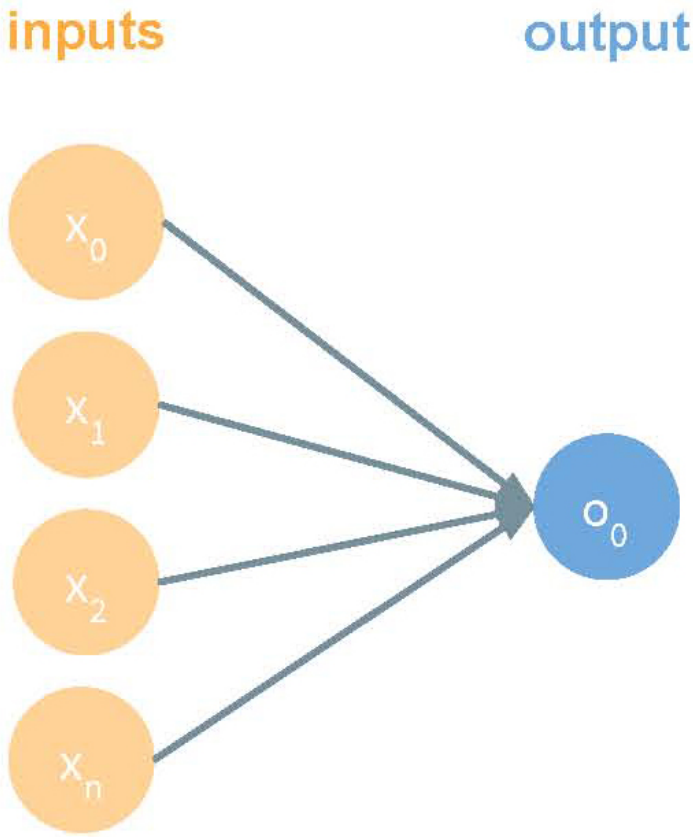
$$output = g(3.2) = \sigma(3.2)$$
$$= \frac{1}{(1 + e^{-3.2})} = 0.96$$



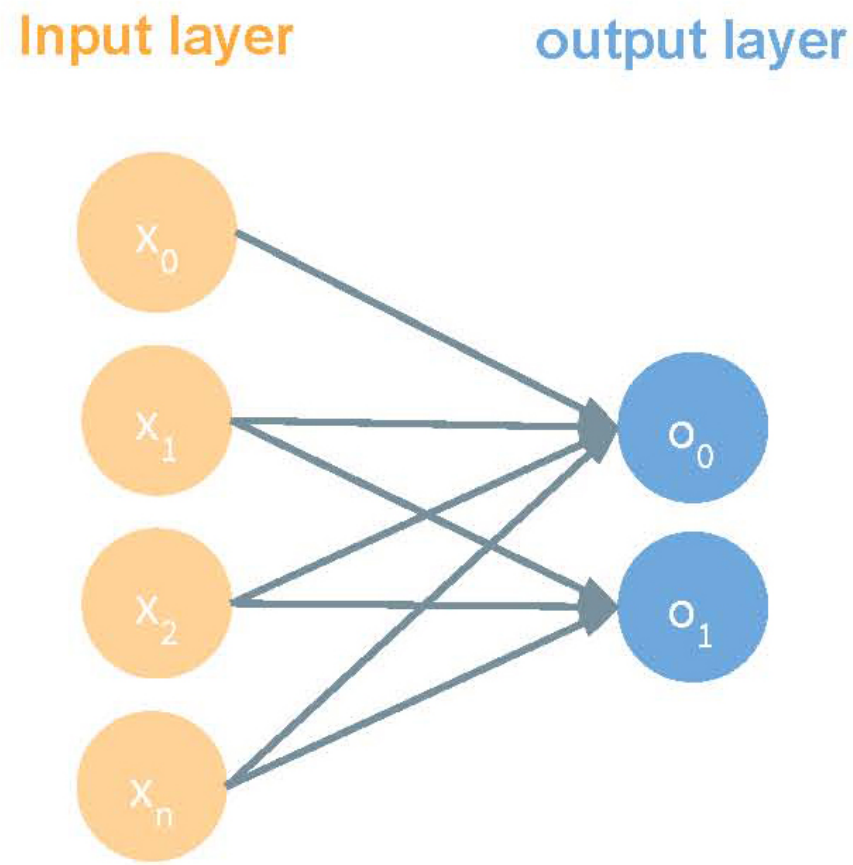
How do we build neural networks with perceptrons?

Perceptron Diagram Simplified

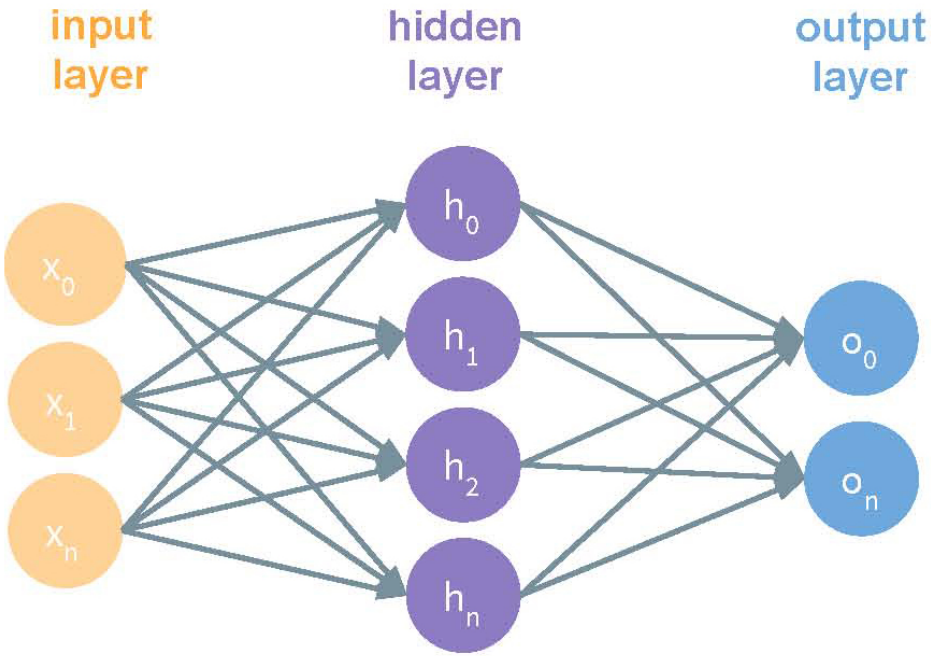




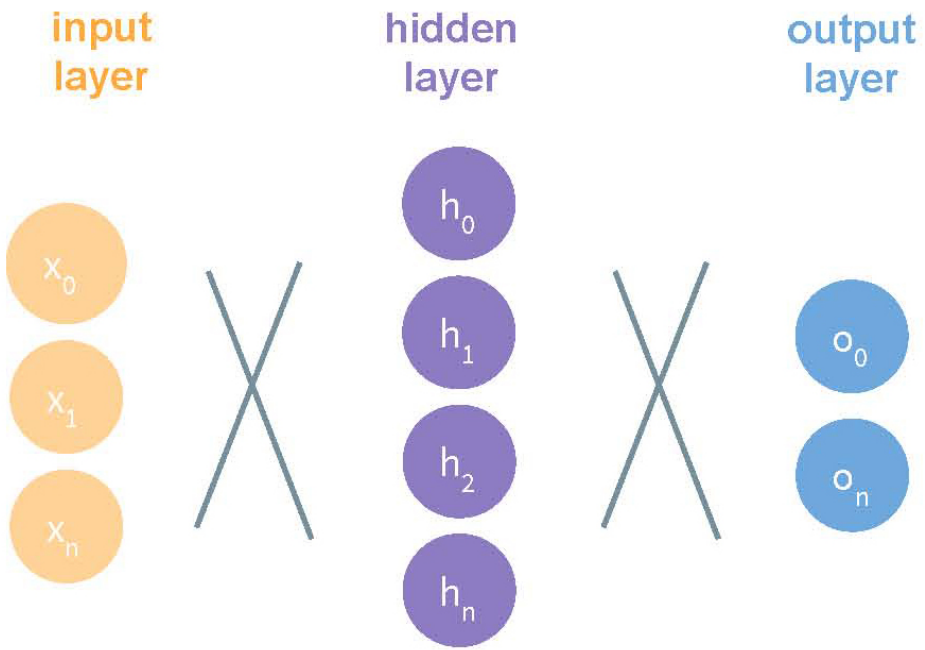
Multi-Output Perceptron



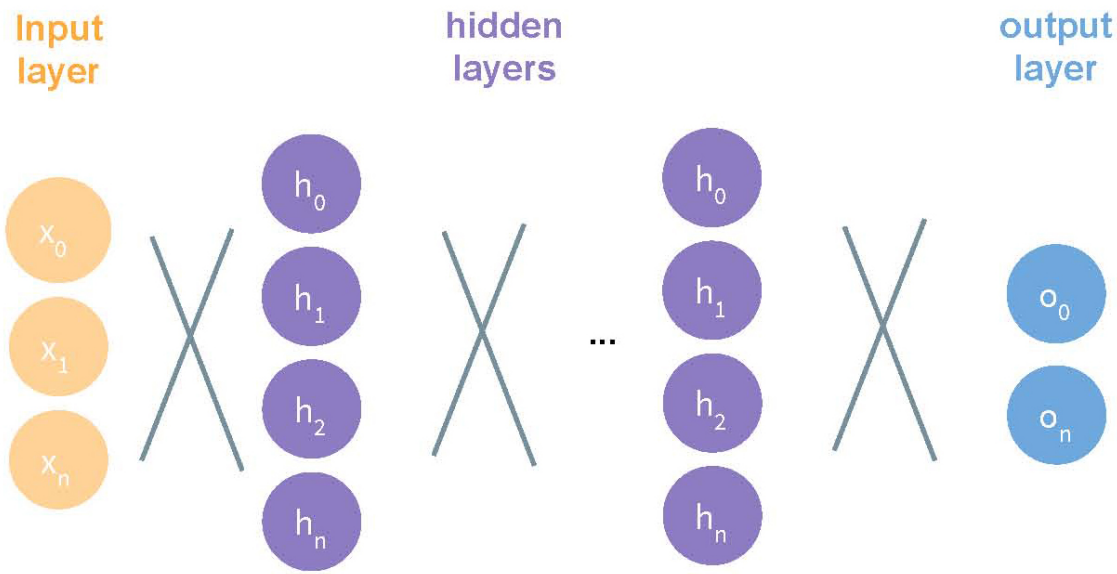
Multi-Layer Perceptron (MLP)



Multi-Layer Perceptron (MLP)



Deep Neural Network



Applying Neural Networks

Example Problem: Will my Flight be Delayed?

Example Problem: Will my Flight be Delayed?

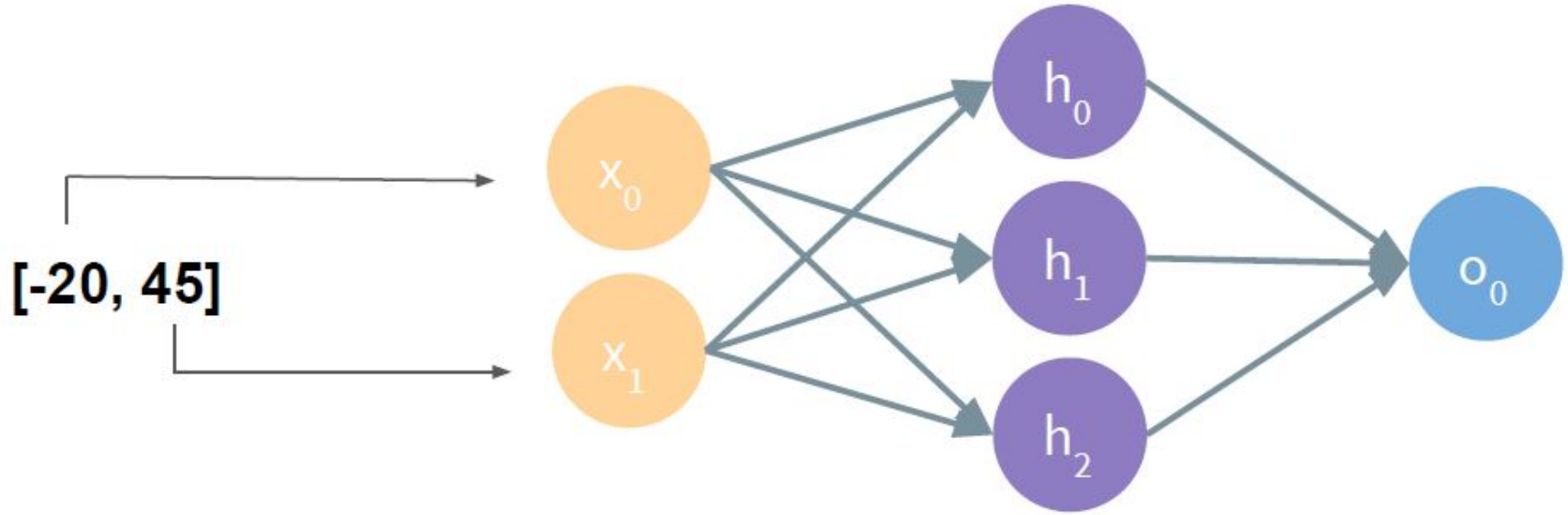
Temperature: -20 F

Wind Speed: 45 mph

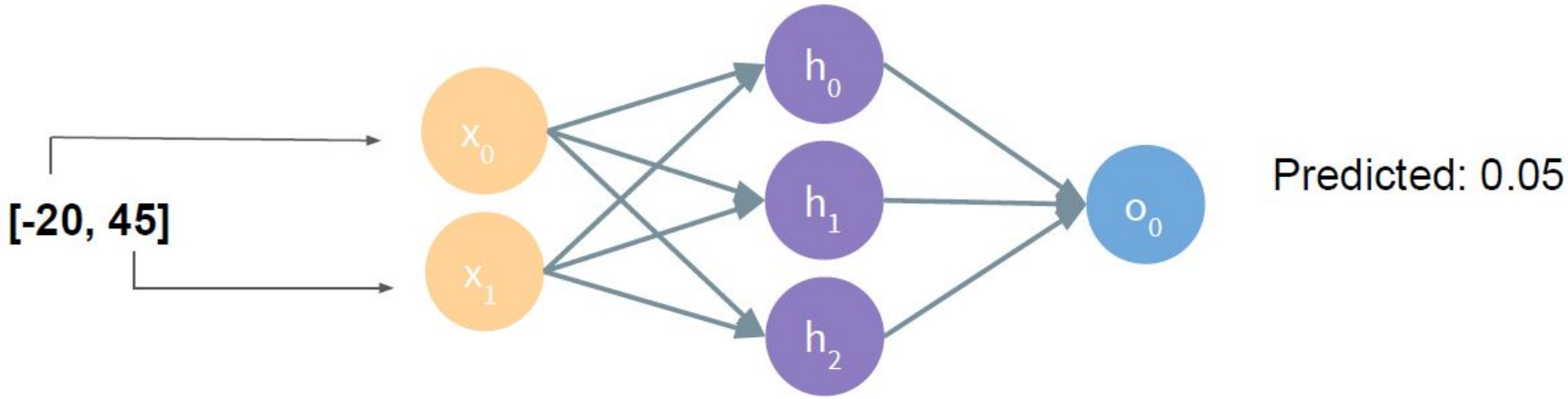
Example Problem: Will my Flight be Delayed?

[-20, 45]

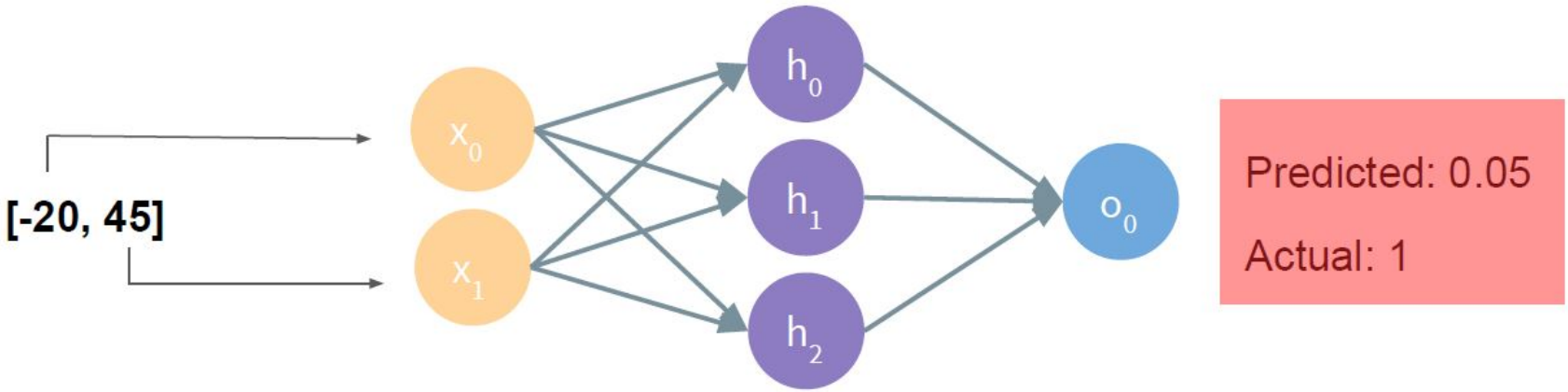
Example Problem: Will my Flight be Delayed?



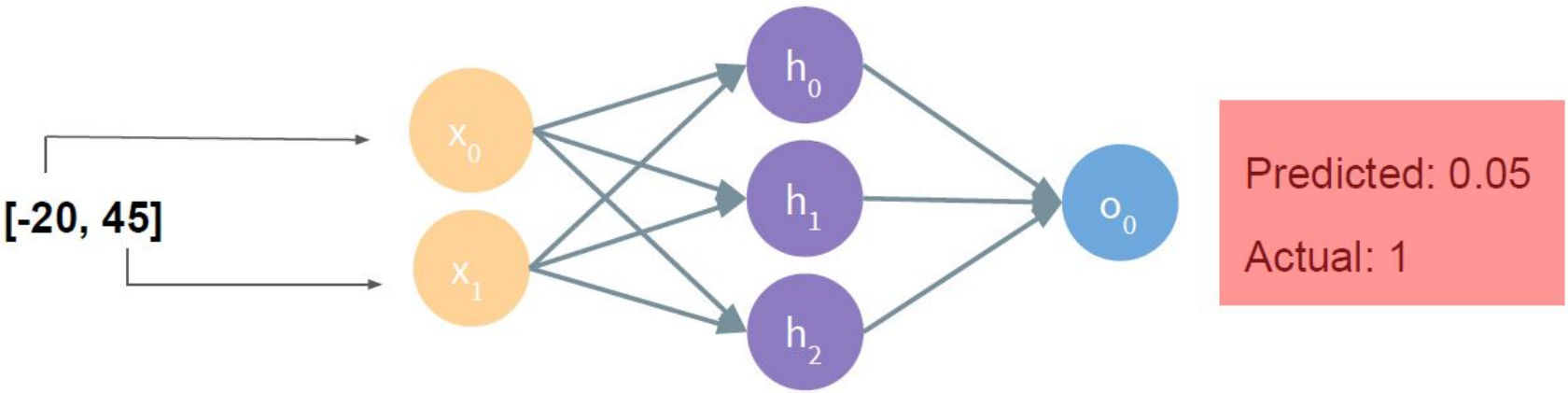
Example Problem: Will my Flight be Delayed?



Example Problem: Will my Flight be Delayed?



Quantifying Loss

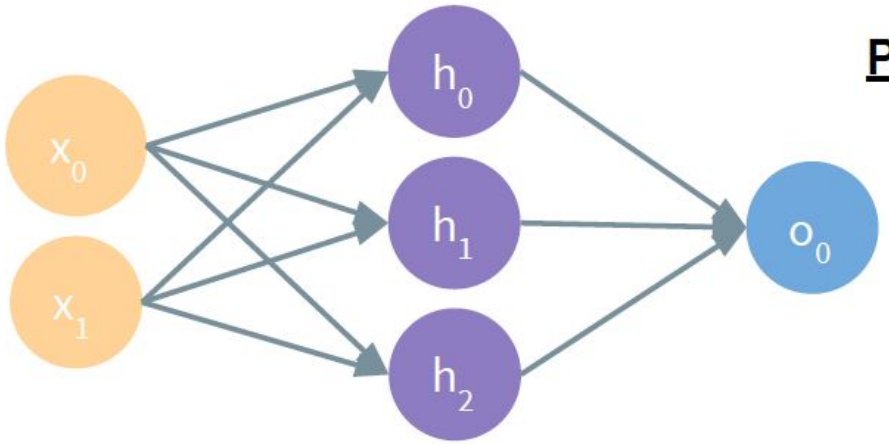


$$loss(f(x^{(i)};\theta),\ y^{(i)})$$

Total Loss

Input

[
[-20, 45],
[80, 0],
[4, 15],
[45, 60],
]



Predicted	Actual
[	[
0.05	1
0.02	0
0.96	1
0.35	1
]	]

$$Total\ \ loss := J(\theta) = \frac{1}{N}\sum_{i}^{N} loss(f(x^{(i)};\theta),\ y^{(i)})$$

Training Neural Networks

Training Neural Networks: Objective

$$arg_\theta min \frac{1}{N} \sum_{i}^{N} loss(f(x^{(i)}; \theta), y^{(i)})$$

Training Neural Networks: Objective

$$arg_\theta \min \frac{1}{N} \sum_{i}^{N} \text{loss}(f(x^{(i)};\theta), y^{(i)})$$

$$J(\theta)$$

loss function

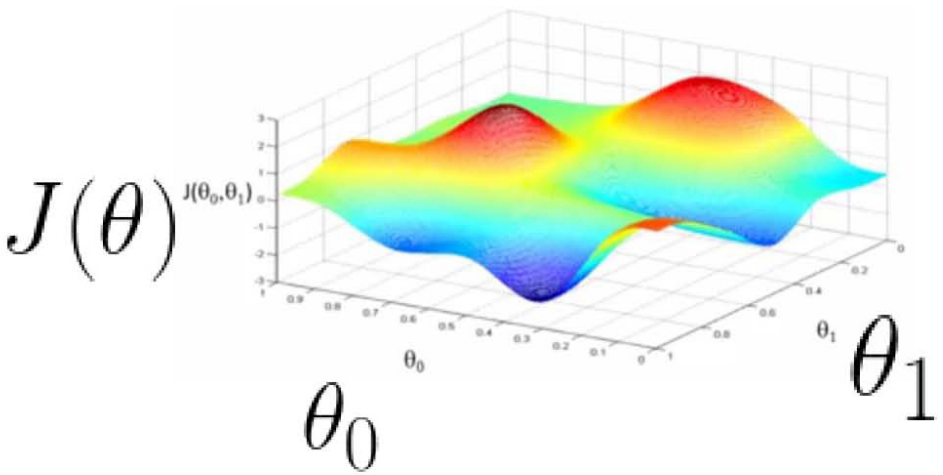
Training Neural Networks: Objective

$$arg_\theta \min \underbrace{\frac{1}{N}\sum_{i}^{N} \text{loss}(f(x^{(i)};\theta), y^{(i)})}$$

$$J(\theta) \leftarrow \theta = W_1, W_2 ... W_n$$

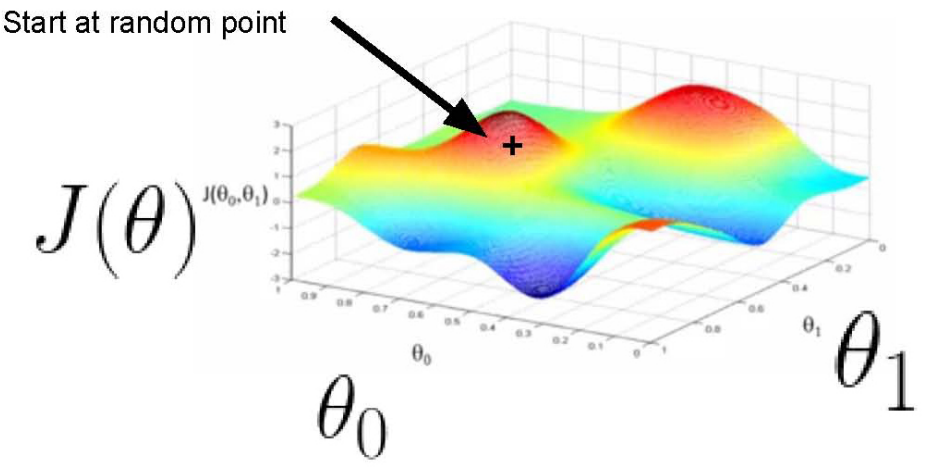
Training Neural Networks: Objective

- Loss is a function of the models parameters



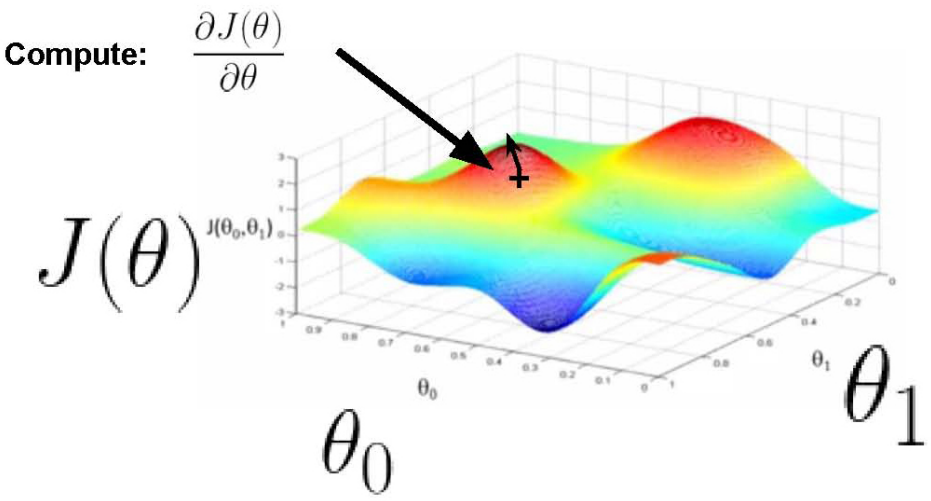
Training Neural Networks: Objective

- How to minimize loss?



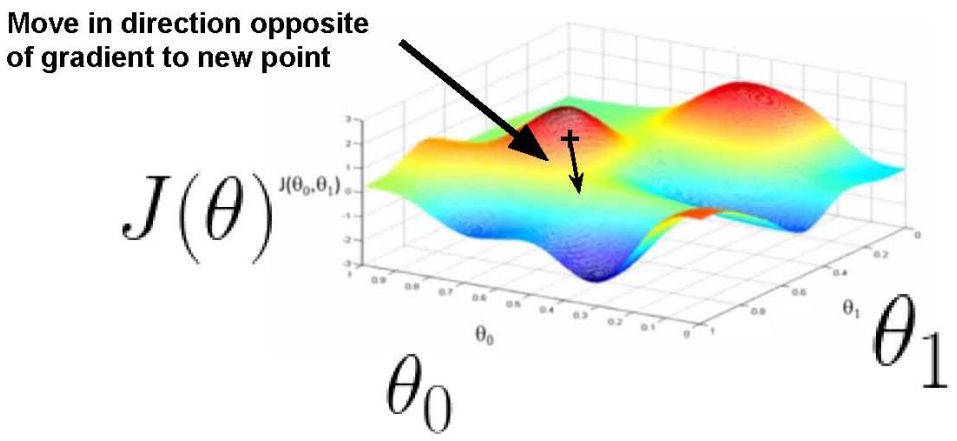
Training Neural Networks: Objective

- How to minimize loss?



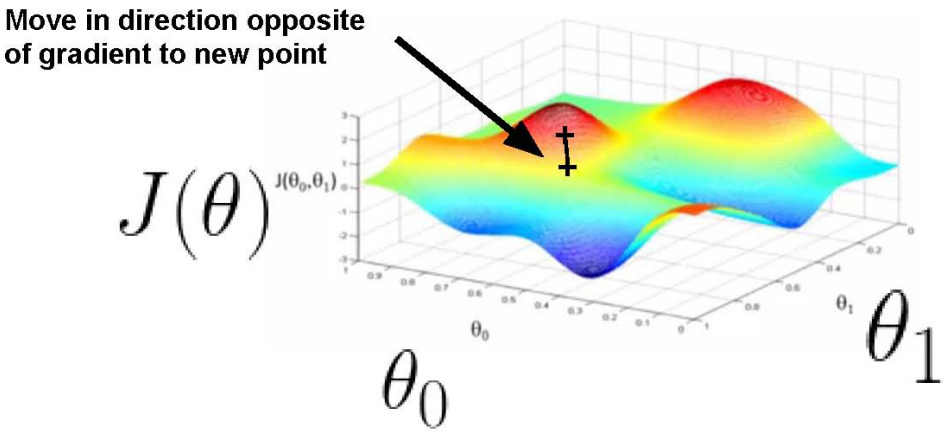
Training Neural Networks: Objective

- How to minimize loss?



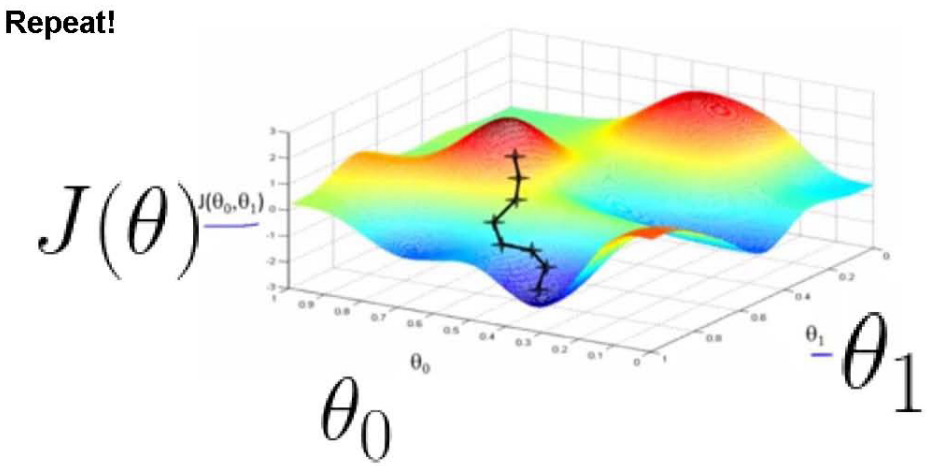
Training Neural Networks: Objective

- How to minimize loss?



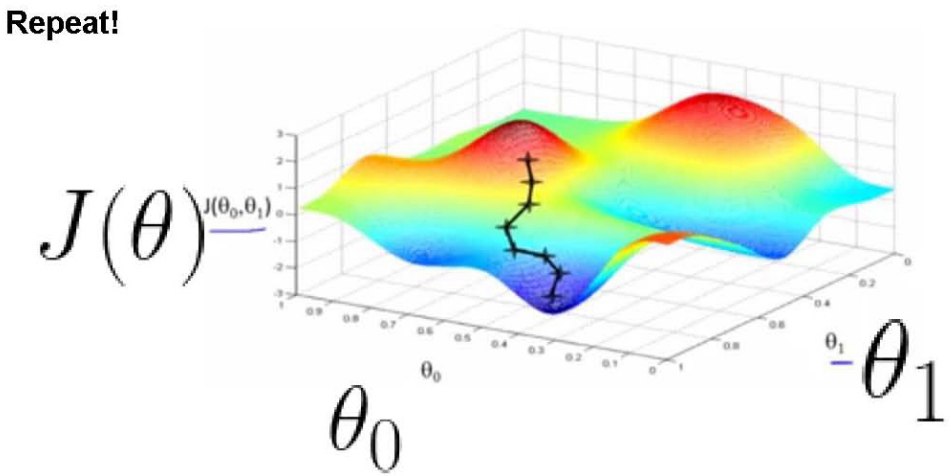
Training Neural Networks: Objective

- How to minimize loss?



Training Neural Networks: Objective

- This is called Stochastic Gradient Descent (SGD)



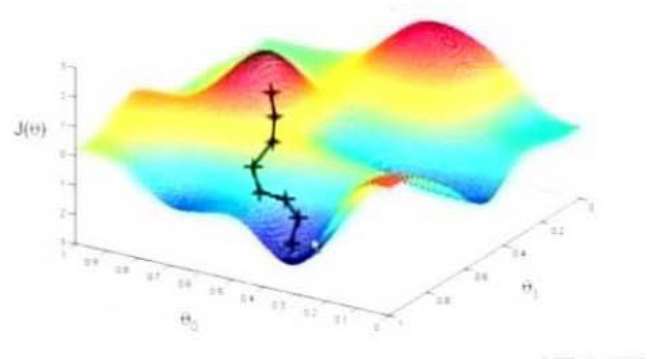
Stochastic Gradient Descent (SGD)

- Initialize θ randomly
- For N Epochs
  - For each training example (x, y):
    - Compute Loss Gradient:

$$\frac{\partial J(\theta)}{\partial \theta}$$

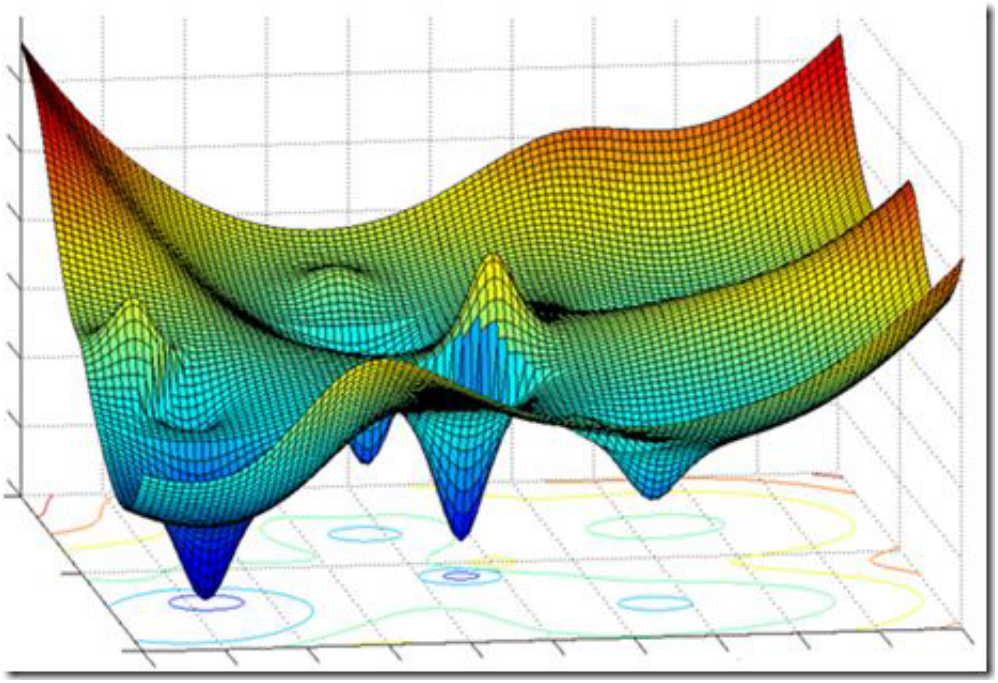
    - Update θ with update rule:

$$\theta := \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$$



Training Neural Networks In Practice

Loss function can be difficult to optimize



Loss function can be difficult to optimize

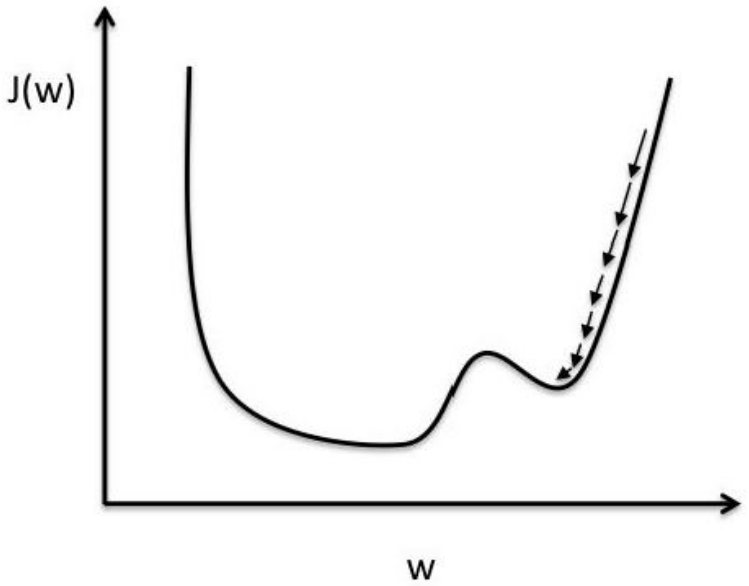
How to Choose Learning Rate?

Update Rule:

$$\theta := \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$$

Learning Rate & Optimization

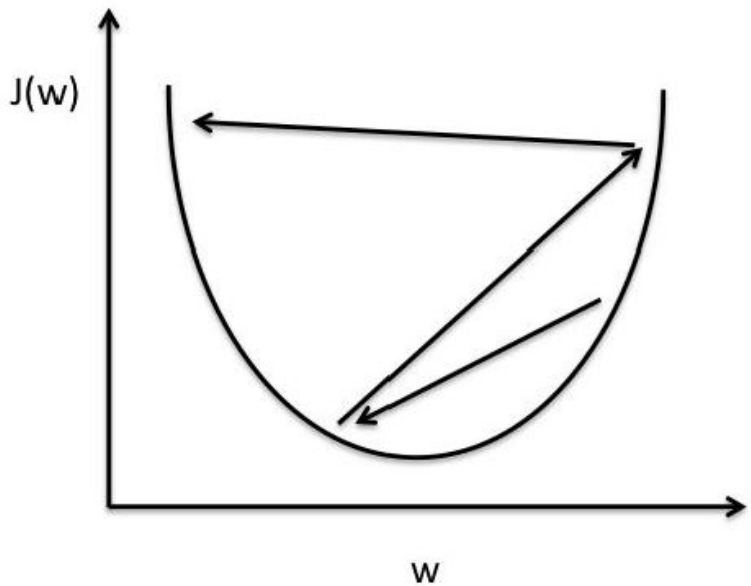
- Small Learning Rate



Small learning rate: Many iterations until convergence and trapping in local minima.

Learning Rate & Optimization

- Large Learning Rate



Large learning rate: Overshooting.

How to deal with this?

- Try lots of different learning rates to see what is just right

How to deal with this?

- Try lots of different learning rates to see what is just right
- Do something smarter

How to deal with this?

- Try lots of different learning rates to see what is just right
- **Do something smarter : Adaptive Learning Rate**
  - ADAM
  - Momentum
  - NAG
  - Adagrad
  - Adadelta
  - RMSProp
- For details: check out http://sebastianruder.com/optimizing-gradient-descent/

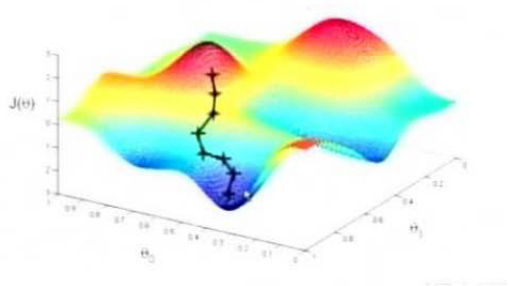
Training Neural Networks in Practice 2: MiniBatches

Why is it Stochastic Gradient Descent?

- Initialize θ randomly
- For N Epochs
  - For each training example (x, y):
    - Compute Loss Gradient: $\frac{\partial J(\theta)}{\partial \theta}$
    - Update θ with update rule:

$$\theta := \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$$

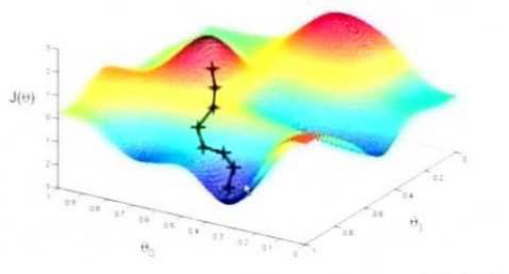
Only an estimate of true gradient!



Minibatches Reduce Gradient Variance

- Initialize θ randomly
- For N Epochs
  - For each training **batch {(x0, y0), ... , (x_B, y_B)}:**
    - Compute Loss Gradient: $\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{B}\sum_{i}^{B}\frac{\partial J_i(\theta)}{\partial \theta}$

More accurate estimate!

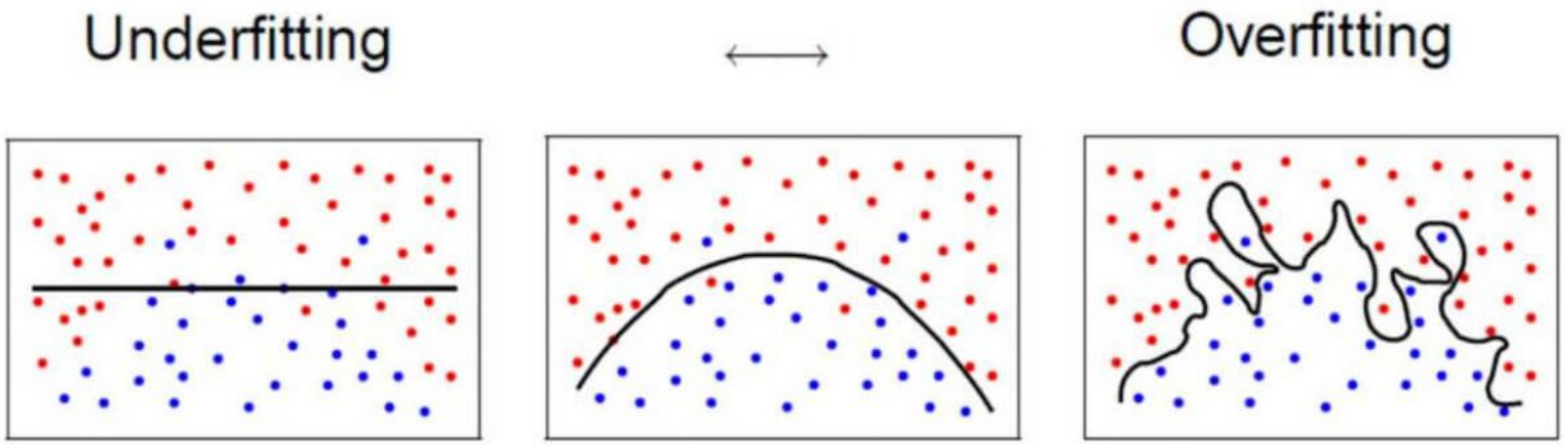


- Update θ with update rule:

$$\theta := \theta - \eta\frac{\partial J(\theta)}{\partial \theta}$$

Training Neural Networks In Practice 3: Fighting Overfitting

The Problem of Overfitting

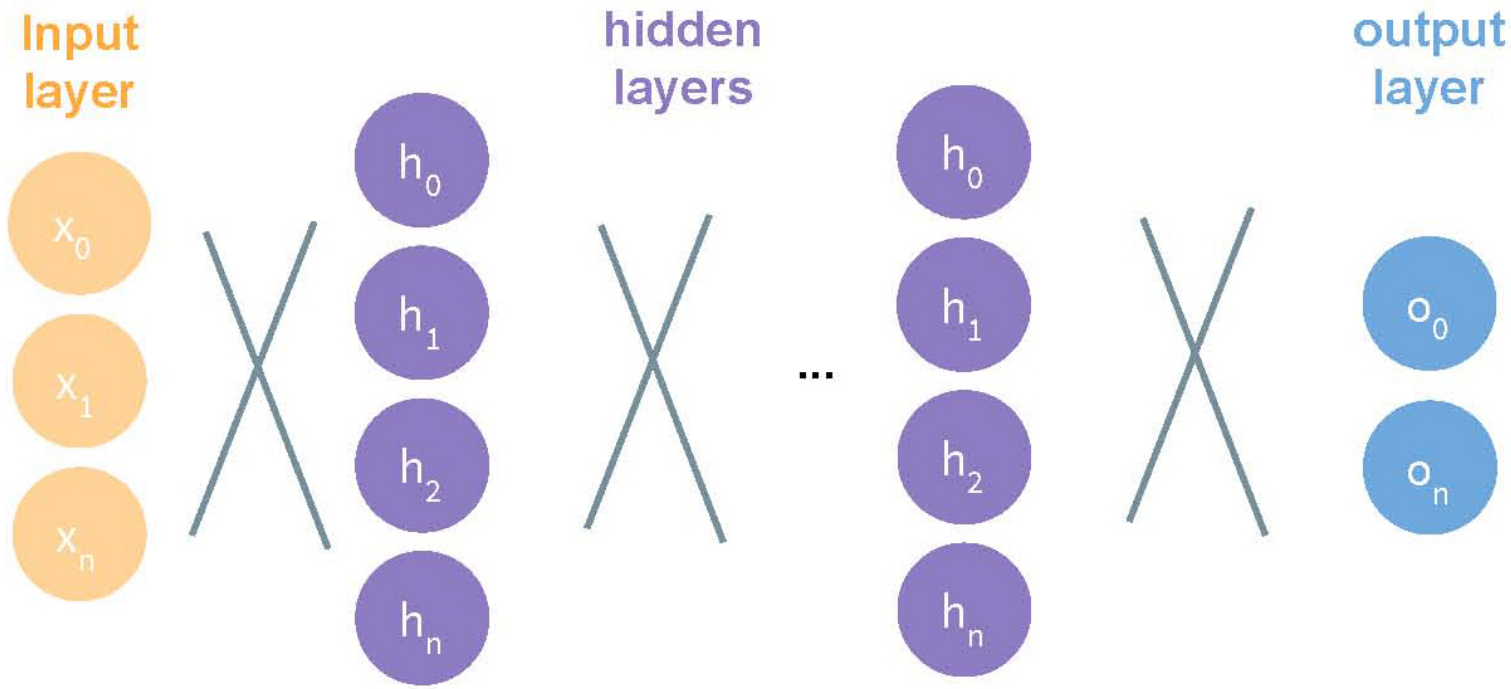


Popular Regularization Techniques

- Dropout
- Early Stopping

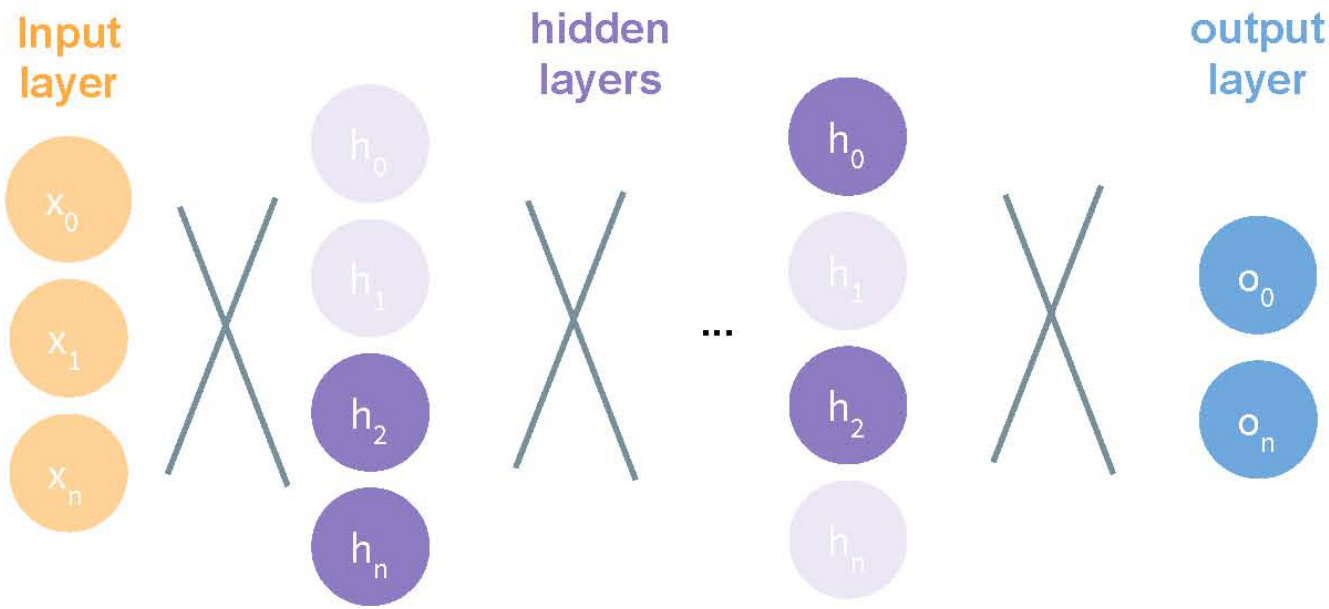
Regularization I: Dropout

- During training, randomly set some activations to 0



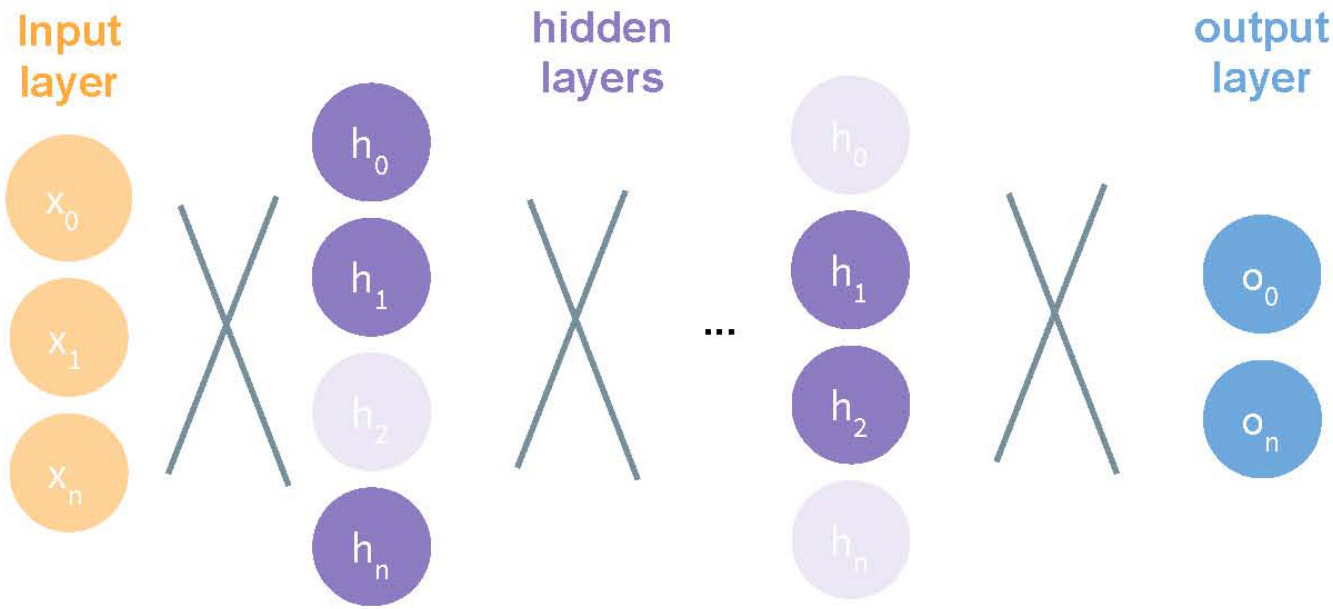
Regularization I: Dropout

- During training, randomly set some activations to 0
  - Typically drop 50% of activations in layer
  - Forces network to not rely on any 1 node



Regularization I: Dropout

- During training, randomly set some activations to 0
  - Typically drop 50% of activations in layer
  - Forces network to not rely on any 1 node



Regularization II: Early Stopping

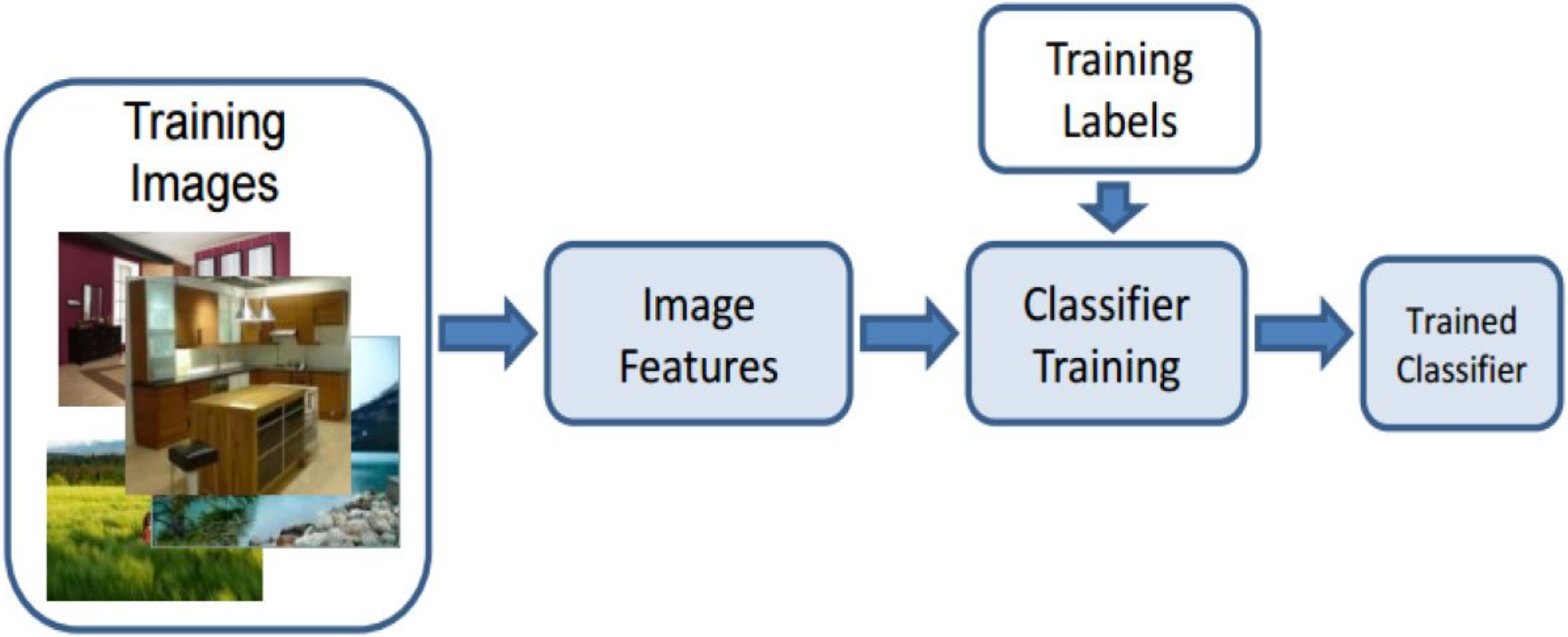
- Dont give the network time to overfit
- ...
- Epoch 15: Train: 85% Validation: 80%
- Epoch 16: Train: 87% Validation: 82%
- Epoch 17: Train: 90% Validation: 85%
- Epoch 18: Train: 95% Validation: 83%
- Epoch 19: Train: 97% Validation: 78%
- Epoch 20: Train: 98% Validation: 75%

Regularization II: Early Stopping

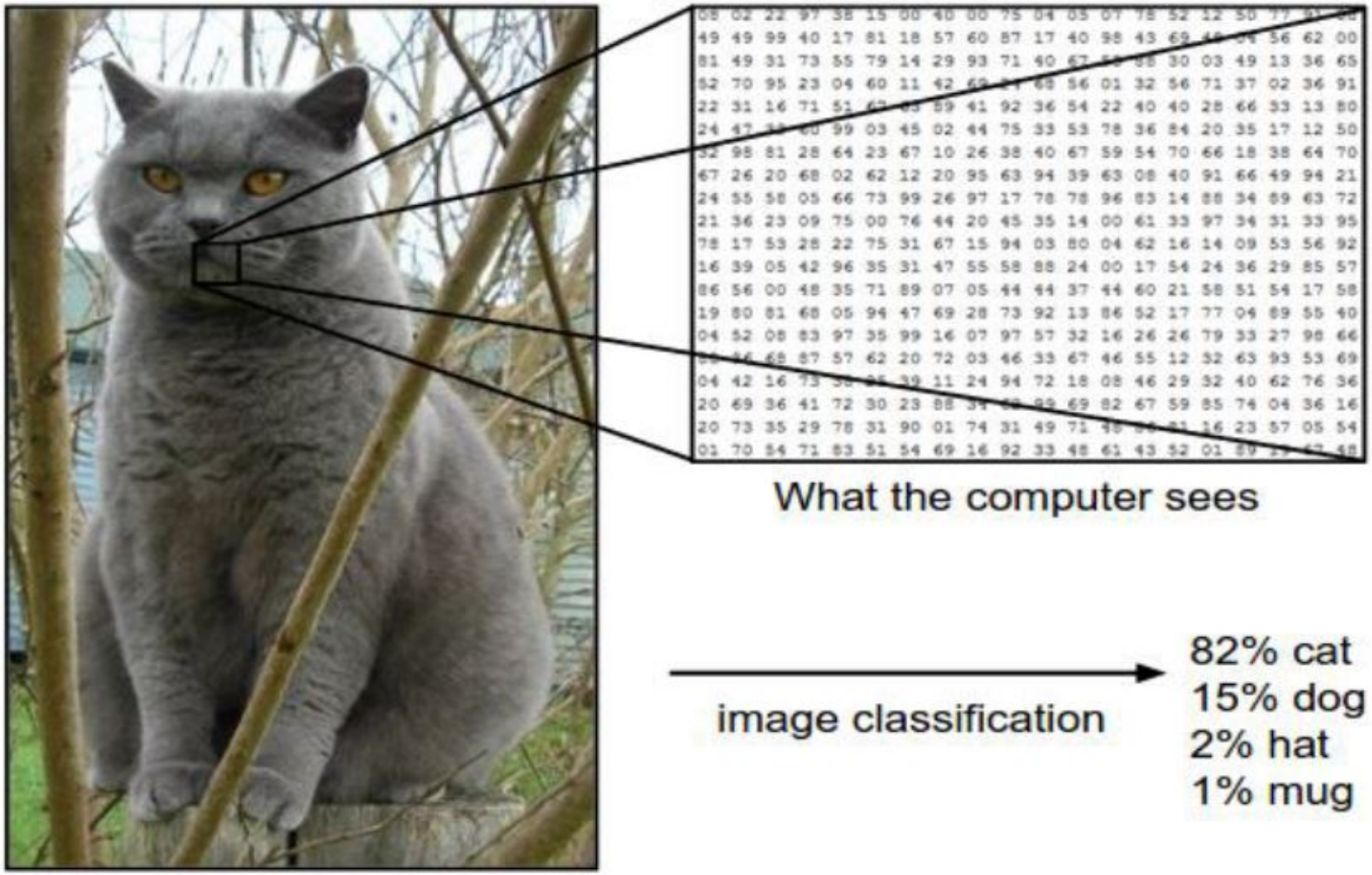
- Dont give the network time to overfit
- ...
- Epoch 15: Train: 85% Validation: 80%
- Epoch 16: Train: 87% Validation: 82%
- Epoch 17: **Train: 90% Validation: 85%** <—-Stop Here
- Epoch 18: Train: 95% Validation: 83%
- Epoch 19: Train: 97% Validation: 78%
- Epoch 20: Train: 98% Validation: 75%

Outline

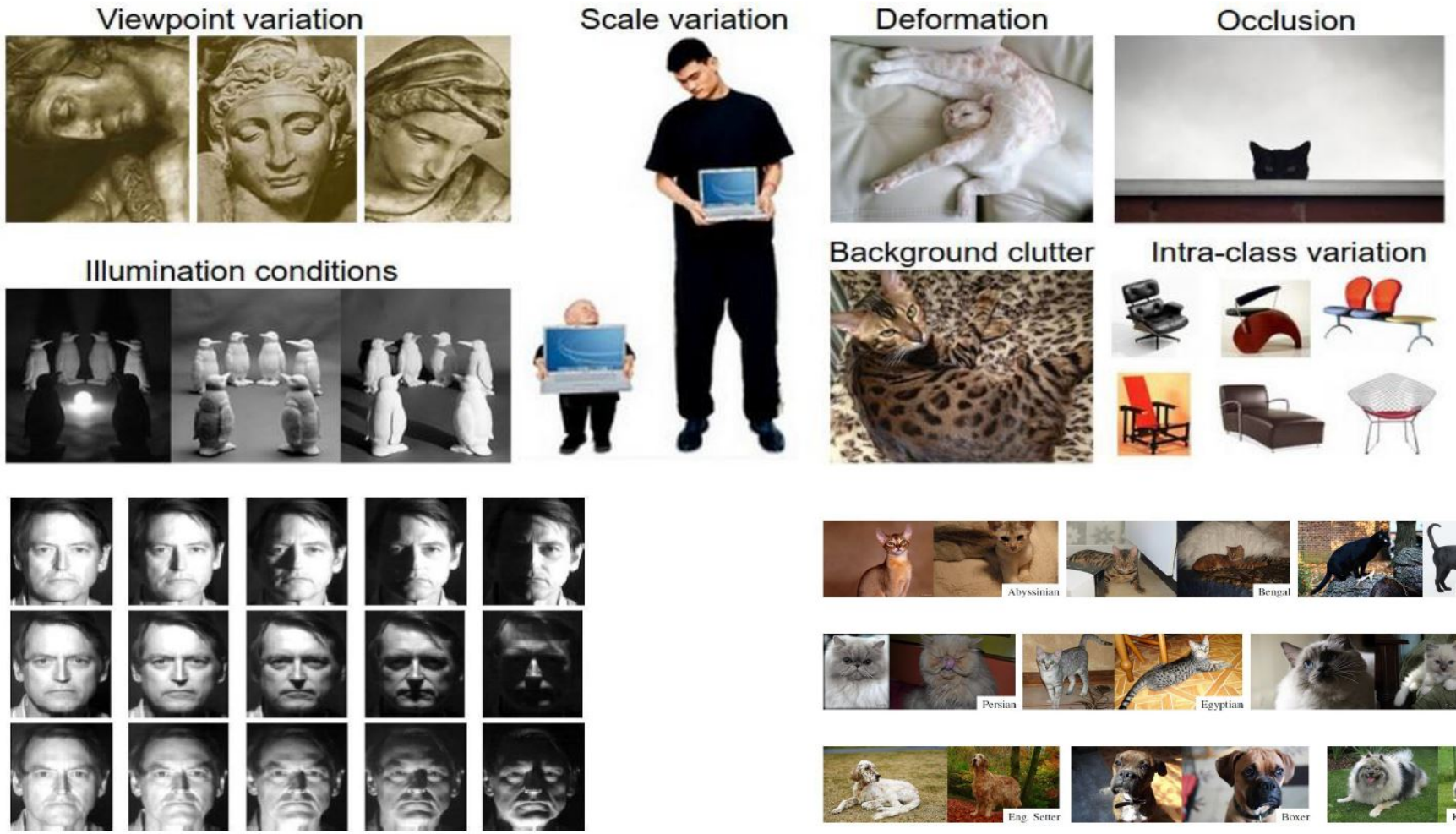
Computer Vision is Machine Learning



Images are Numbers

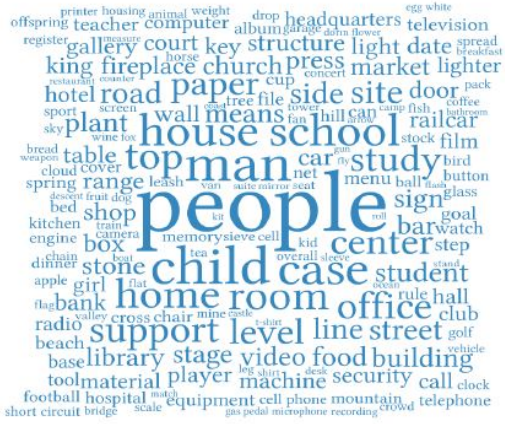


Computer Vision is Hard

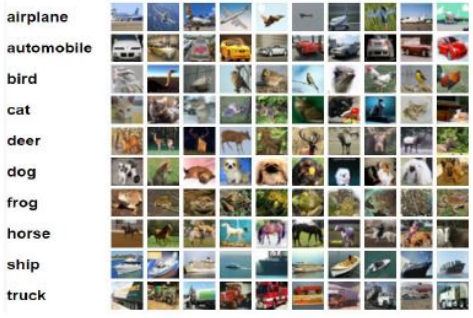


Famous Computer Vision Datasets

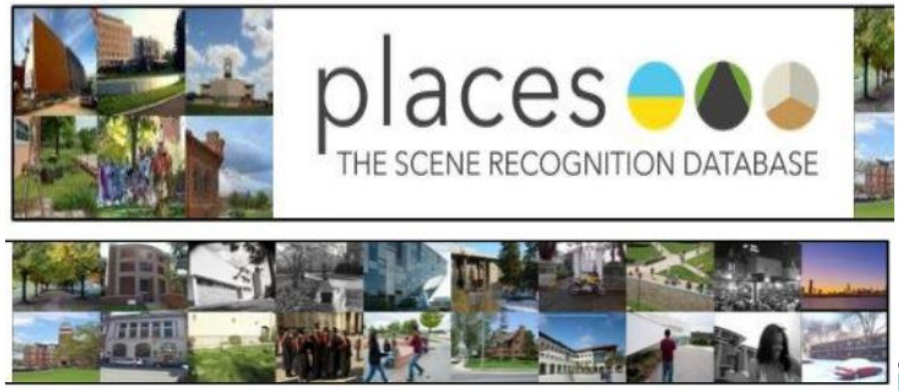
MNIST: handwritten digits



ImageNet: WordNet hierarchy



CIFAR-10(0): tiny images

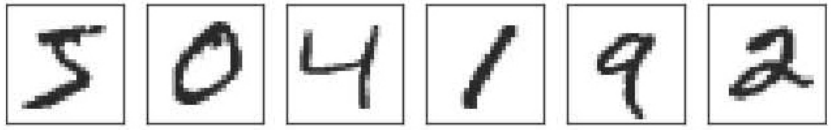


Places: natural scenes

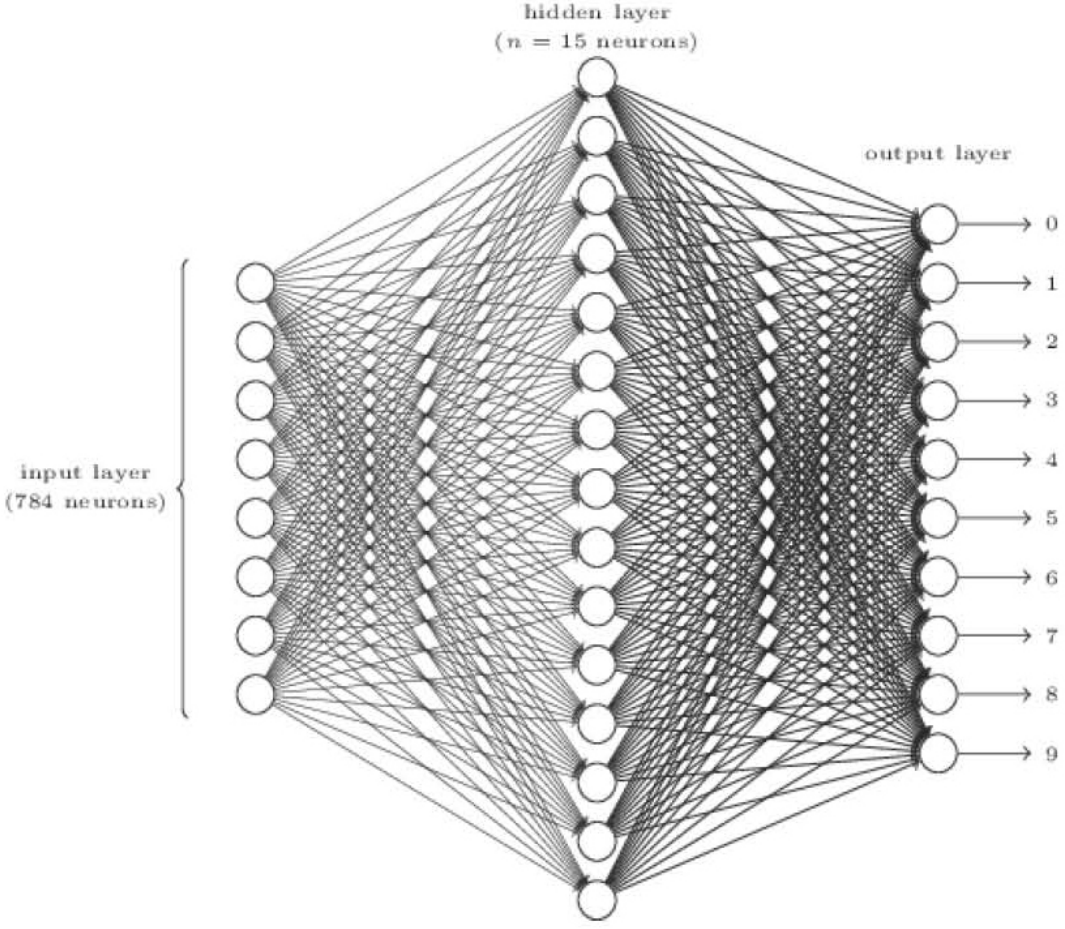
Outline

Classify an Image of a Number

Input:
(28x28)



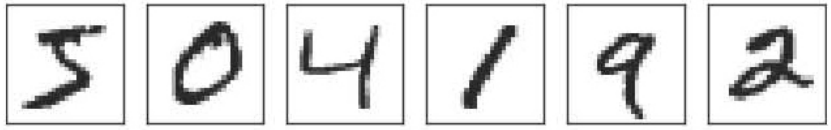
Network:



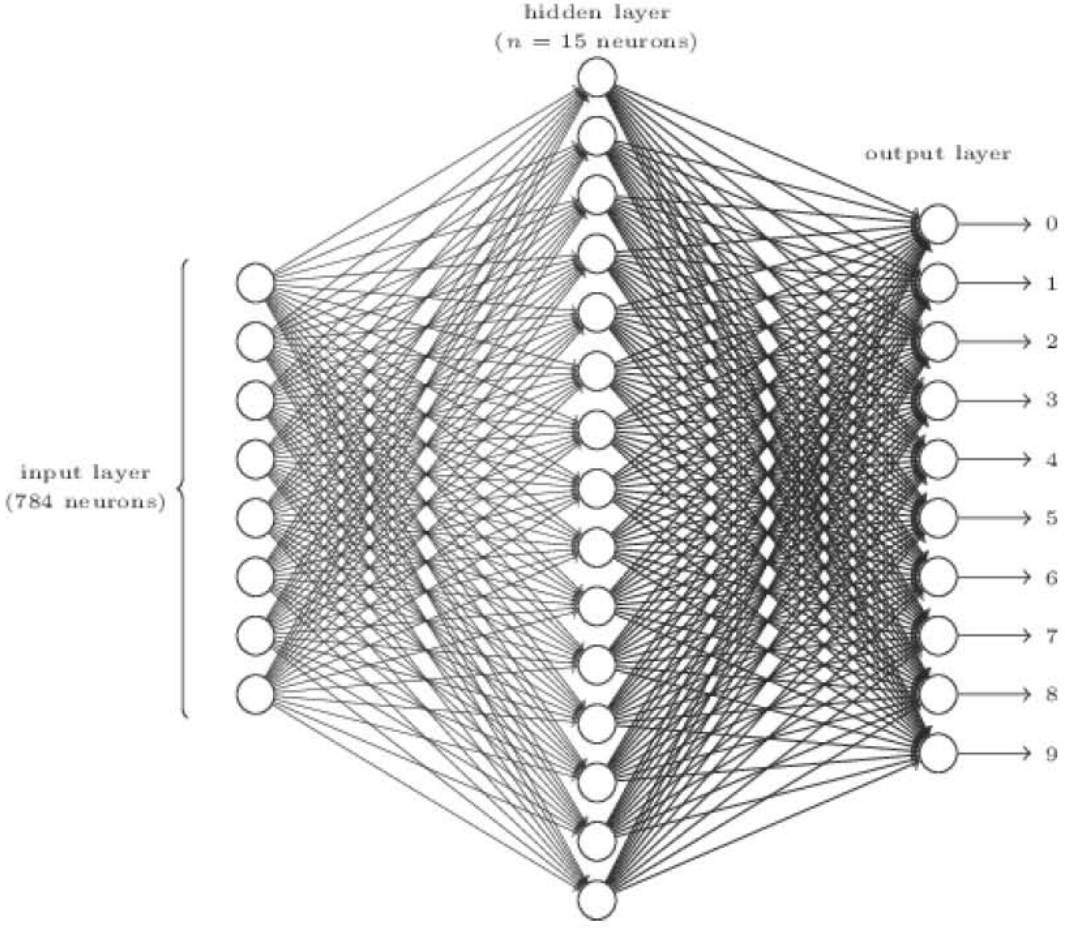
Outline

Classify an Image of a Number

Input:
(28x28)



Network:



Neural Network on MNIST Dataset

- Creating a simple three layer neural network by using the MNIST dataset that the TensorFlow package provides
- This MNIST dataset is a set of 28×28 pixel grayscale images which represent hand-written digits
- It has 55,000 training rows, 10,000 testing rows and 5,000 validation rows

Load Dataset

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

Neural Network on MNIST Dataset

Variable Set up

```
# Python optimisation variables
learning_rate = 0.5
epochs = 10
batch_size = 100

# declare the training data placeholders
# input x - for 28 x 28 pixels = 784
x = tf.placeholder(tf.float32, [None, 784])
# now declare the output data placeholder - 10 digits
y = tf.placeholder(tf.float32, [None, 10])
```

Neural Network on MNIST Dataset

- There are always L-1 number of weights/bias tensors, where L is the number of layers.

Weight and Bias Code

```
# now declare the weights connecting the input to the hidden layer
W1 = tf.Variable(tf.random_normal([784, 300], stddev=0.03), name='W1')
b1 = tf.Variable(tf.random_normal([300]), name='b1')
# and the weights connecting the hidden layer to the output layer
W2 = tf.Variable(tf.random_normal([300, 10], stddev=0.03), name='W2')
b2 = tf.Variable(tf.random_normal([10]), name='b2')
```

Neural Network on MNIST Dataset

- Setup node inputs and activation functions of the hidden layer nodes

Hidden Layer Node Setup

```
# calculate the output of the hidden layer
hidden_out = tf.add(tf.matmul(x, W1), b1)
hidden_out = tf.nn.relu(hidden_out)
```

Neural Network on MNIST Dataset

- Setup the output layer, $y\_$

Output Layer Node Setup

```
# now calculate the hidden layer output - in this case, let's use a softmax activated
# output layer
y_ = tf.nn.softmax(tf.add(tf.matmul(hidden_out, W2), b2))
```

Neural Network on MNIST Dataset

- Optimisation function
  - Include a cost or loss function for the optimization/ backpropagation to work on

cross entropy cost function

$$j = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n} y_j^{(i)} log(y_{j\_}^{(i)}) + (1 - y_j^{(i)}) log(1 - y_{j\_}^{(i)})$$

Output Layer Node Setup

```
y_clipped = tf.clip_by_value(y_, 1e-10, 0.9999999)
cross_entropy = -tf.reduce_mean(tf.reduce_sum(y * tf.log(y_clipped)
                    + (1 - y) * tf.log(1 - y_clipped), axis=1))
```

Neural Network on MNIST Dataset

Optimizer Algorithm Setup

```
# add an optimiser
optimiser = tf.train.GradientDescentOptimizer(
    learning_rate=learning_rate).minimize(cross_entropy)
```

- This function will then perform the gradient descent and the backpropagation for you.

Neural Network on MNIST Dataset

- Setup the variable initialisation operation and an operation to measure the accuracy of our predictions

Accuracy Code

```
# finally setup the initialisation operator
init_op = tf.global_variables_initializer()

# define an accuracy assessment operation
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

Neural Network on MNIST Dataset

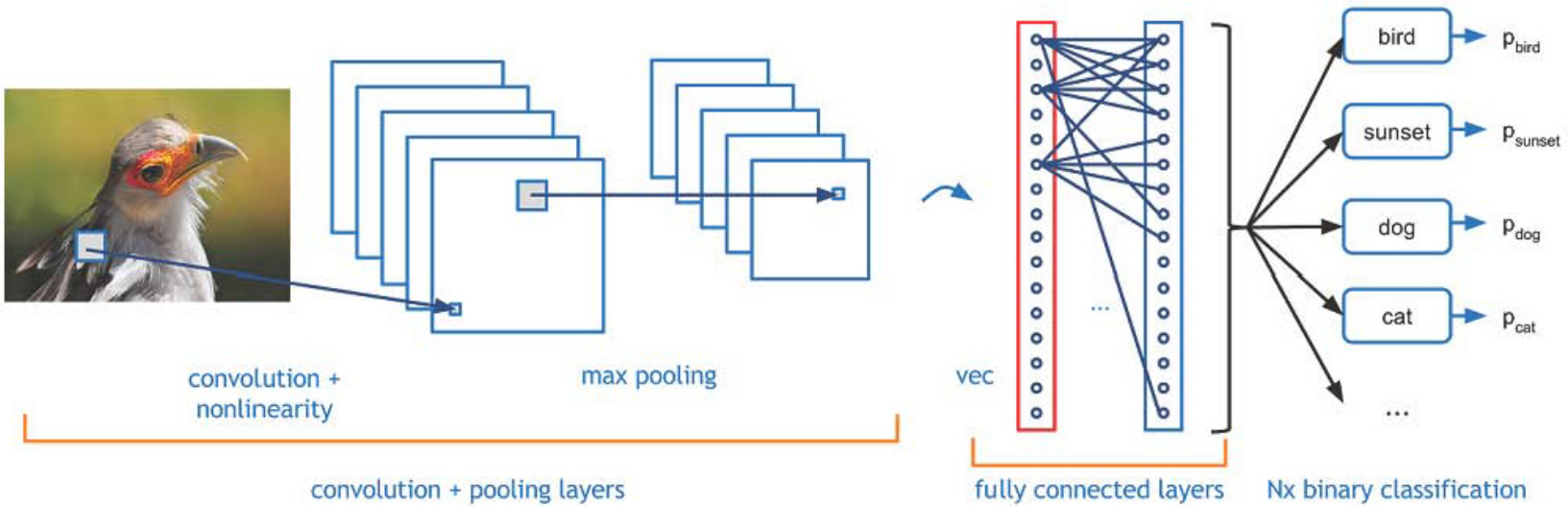
Setup Code for the Training Process

```
# start the session
with tf.Session() as sess:
    # initialise the variables
    sess.run(init_op)
    total_batch = int(len(mnist.train.labels) / batch_size)
    for epoch in range(epochs):
        avg_cost = 0
        for i in range(total_batch):
            batch_x, batch_y = mnist.train.next_batch(batch_size=batch_size)
            _, c = sess.run([optimiser, cross_entropy],
                            feed_dict={x: batch_x, y: batch_y})
            avg_cost += c / total_batch
        print("Epoch:", (epoch + 1), "cost =", "{:.3f}".format(avg_cost))
    print(sess.run(accuracy, feed_dict={x: mnist.test.images, y: mnist.test.labels}))
```

Outline

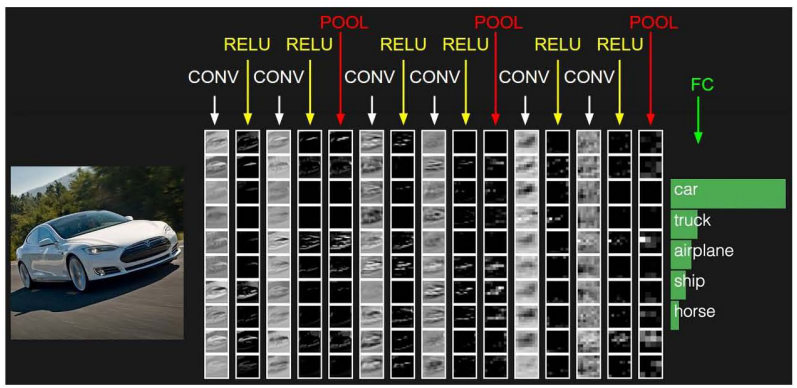
Convolutional Neural Networks (CNN)

- Learn a complex representation of visual data using vast amounts of data
- Inspired by the human visual system
- Learn multiple layers of transformations, which are applied on top of each other to extract a progressively more sophisticated representation of the input

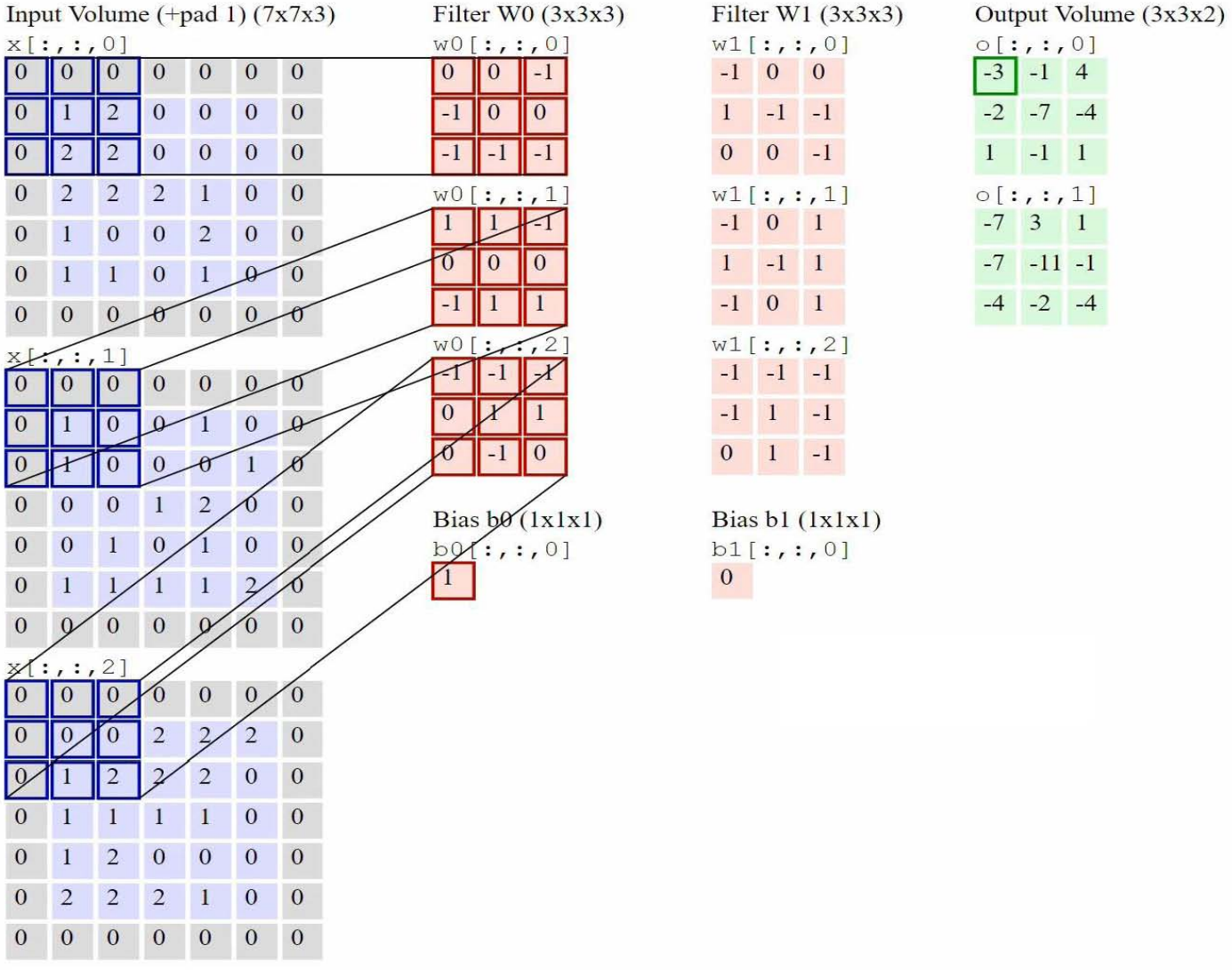


Convolutional Neural Networks: Layers

- **INPUT:** [32×32×3] will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R, G, B
- **CONV Layer:** will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32×32×12] if we decided to use 12 filters
- **RELU Layer:** will apply an elementwise activation function, such as the $max(0, x)$ thresholding at zero. This leaves the size of the volume unchanged ([32×32×12])
- **POOL Layer:** will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as [16×16×12]
- **Fully-Connected (FC) Layer:** will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume

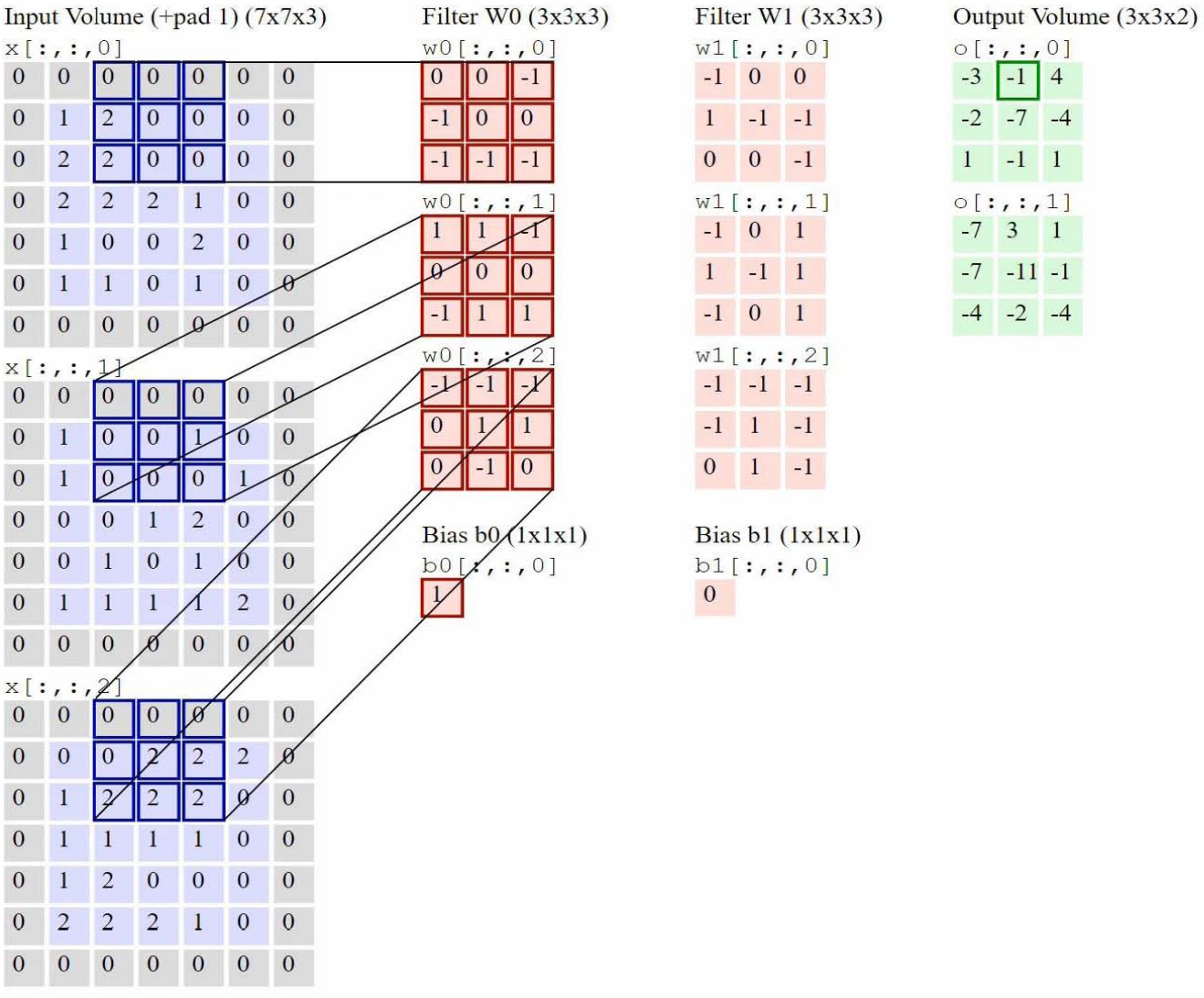


Convolutional Layer



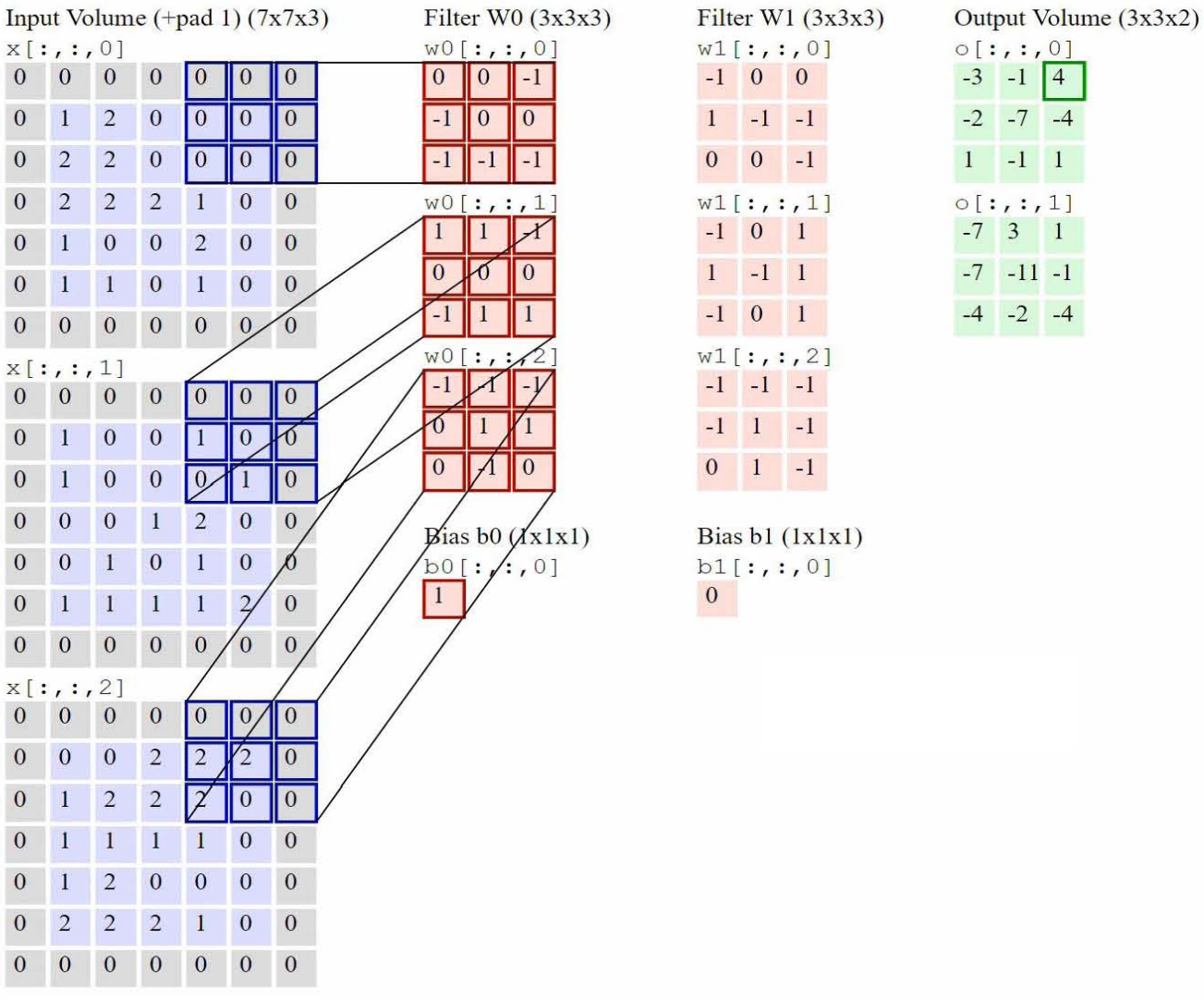
Source: MIT 6.S191

Convolutional Layer



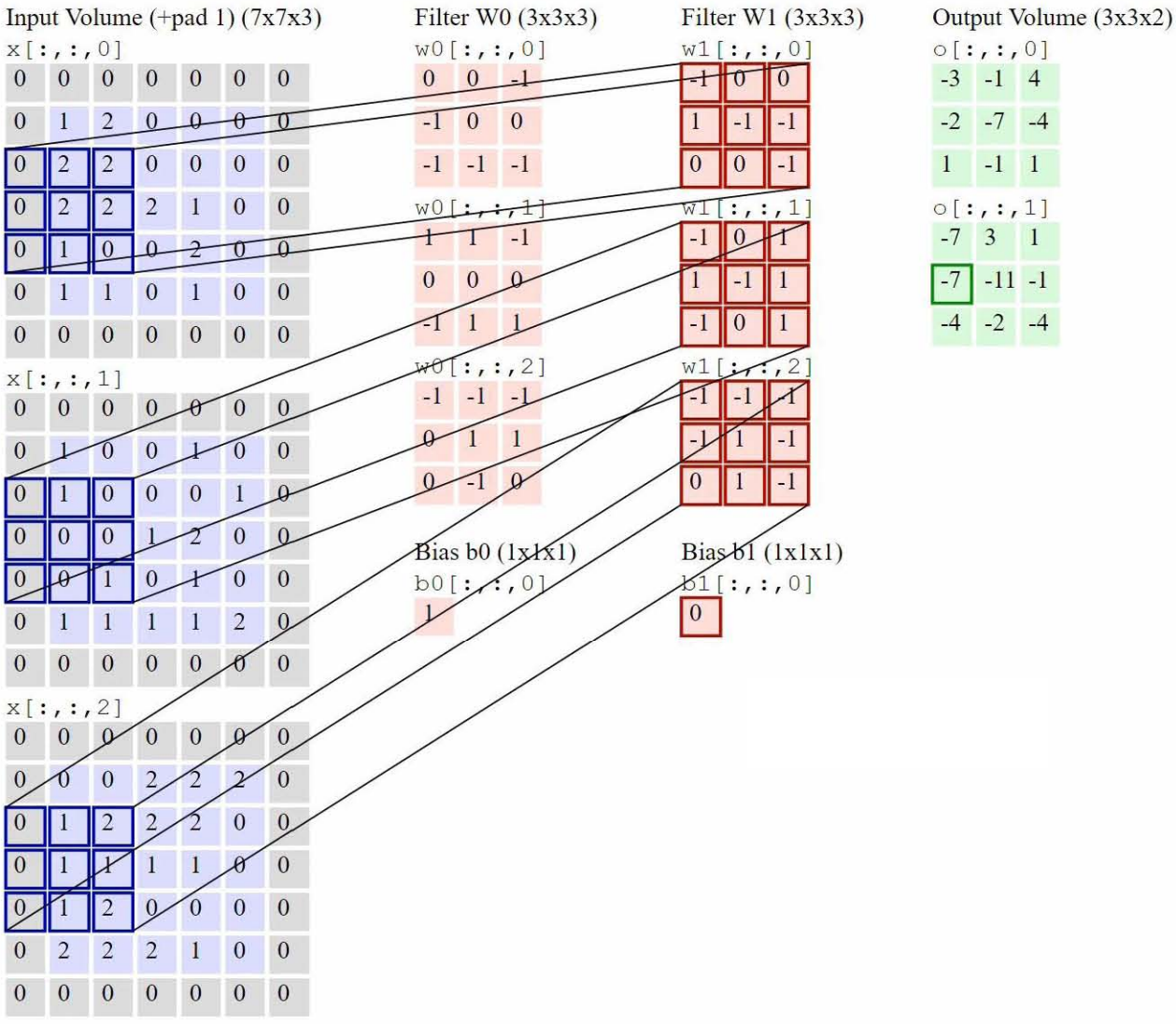
Source: MIT 6.S191

Convolutional Layer



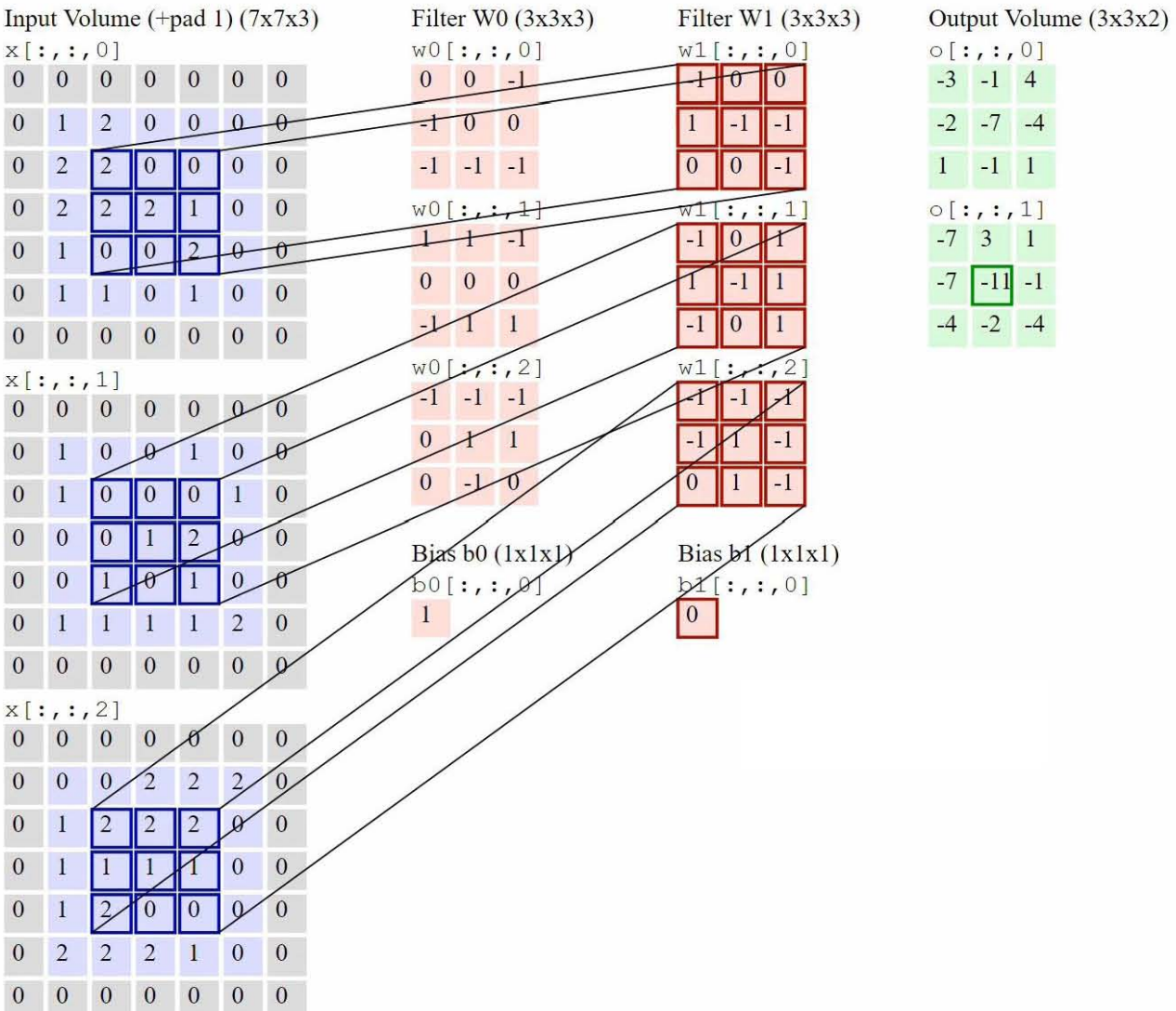
Source: MIT 6.S191

Convolutional Layer



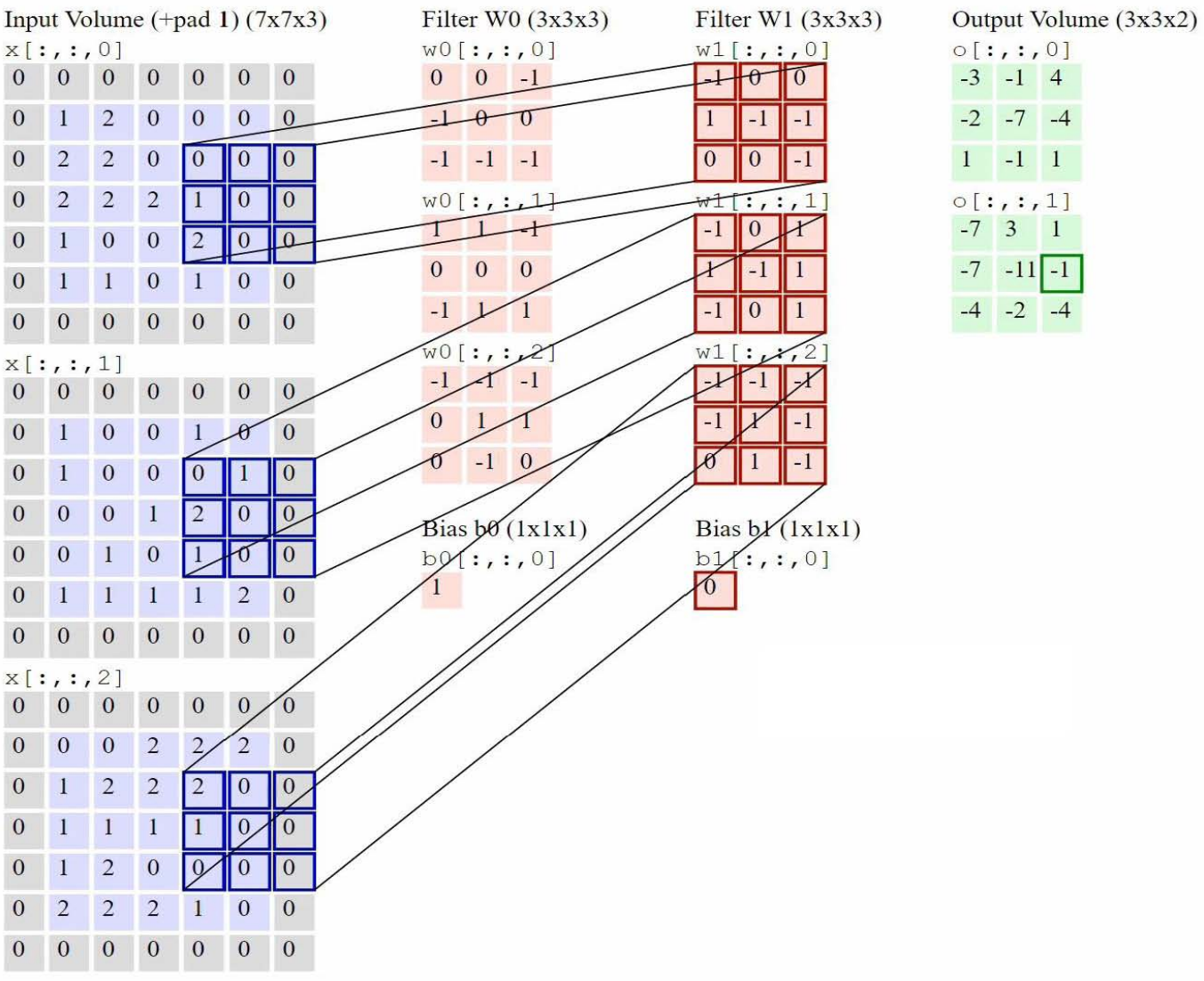
Source: MIT 6.S191

Convolutional Layer



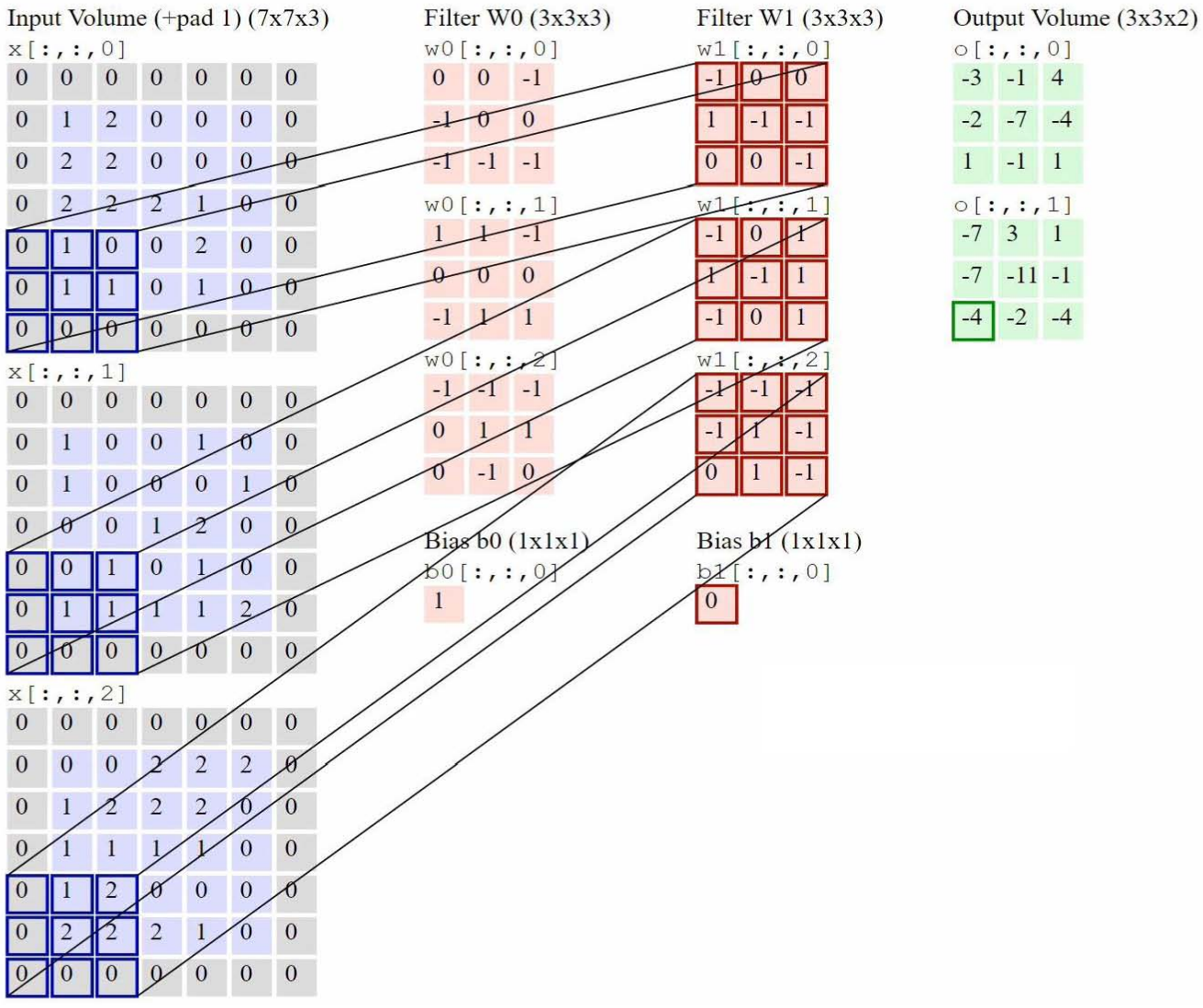
Source: MIT 6.S191

Convolutional Layer



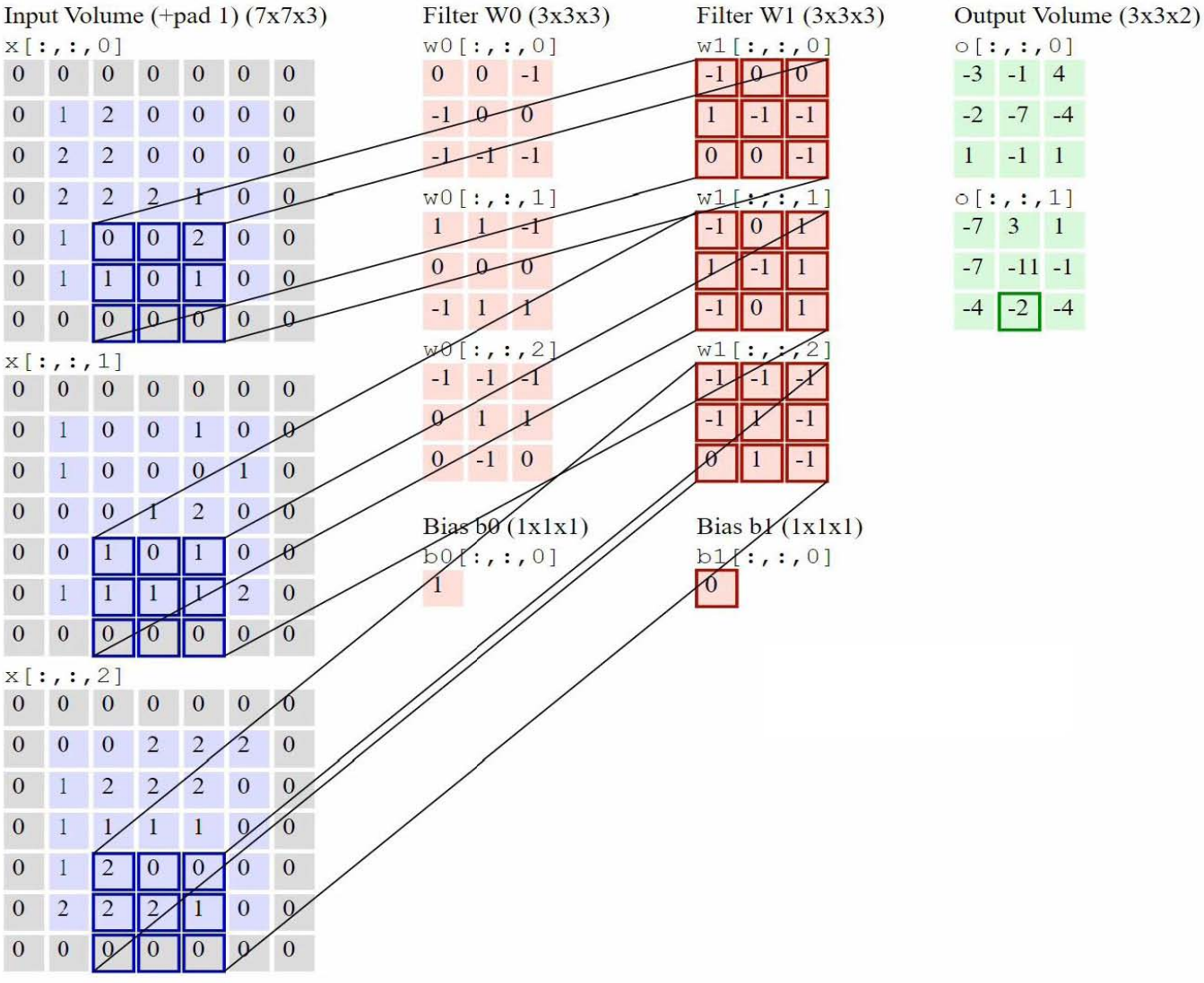
Source: MIT 6.S191

Convolutional Layer



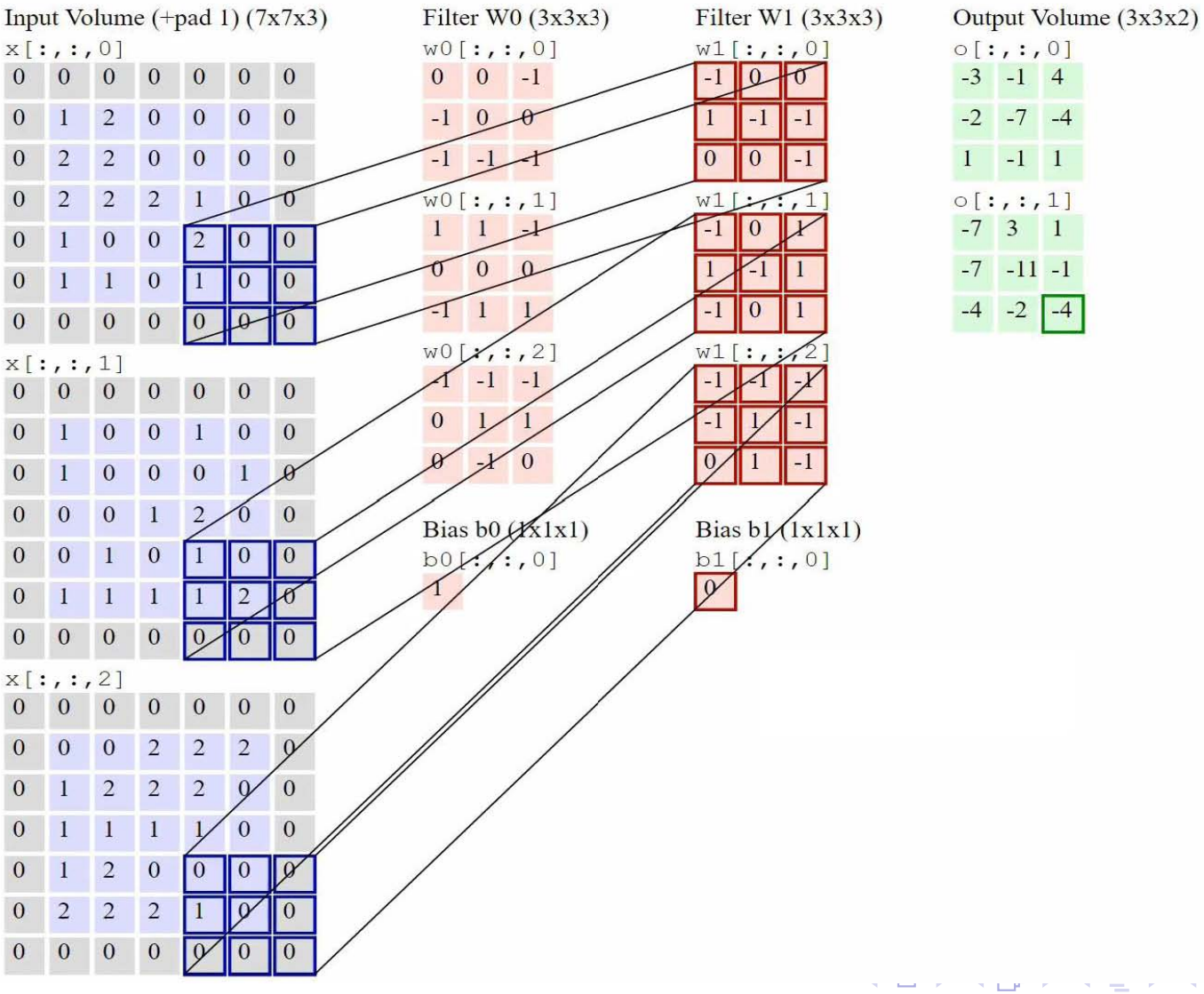
Source: MIT 6.S191

Convolutional Layer



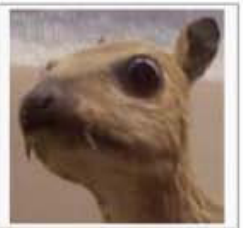
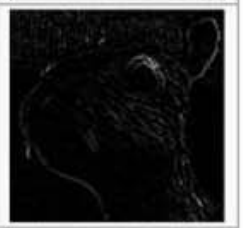
Source: MIT 6.S191

Convolutional Layer

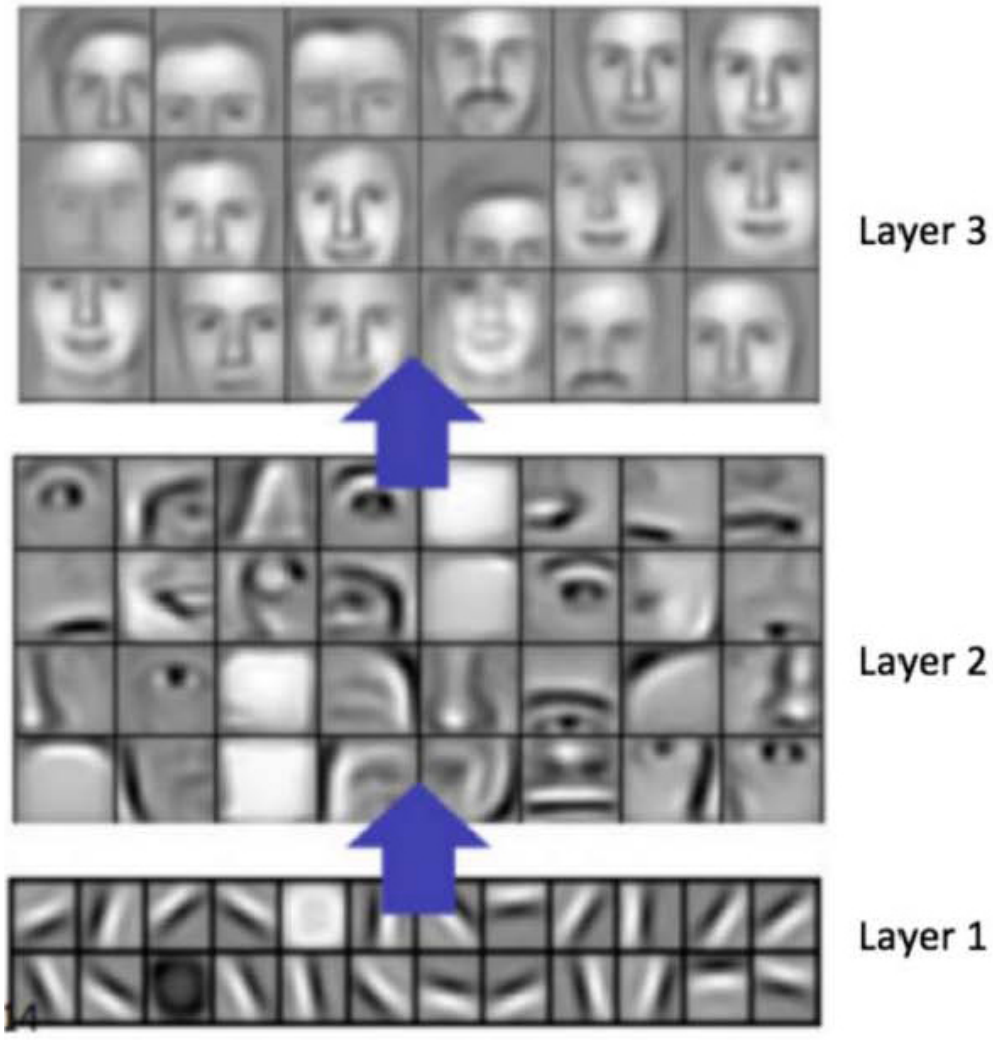


Source: MIT 6.S191

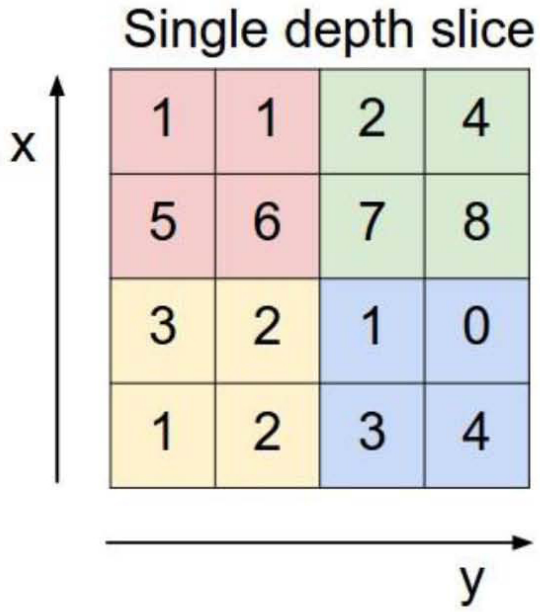
Convolution

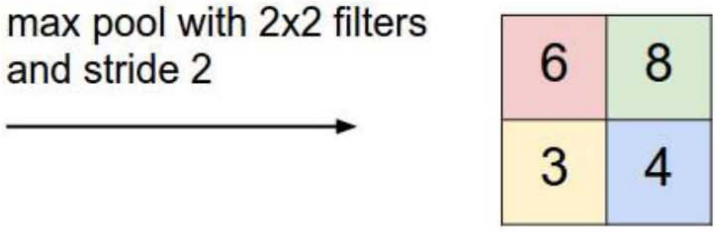
Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

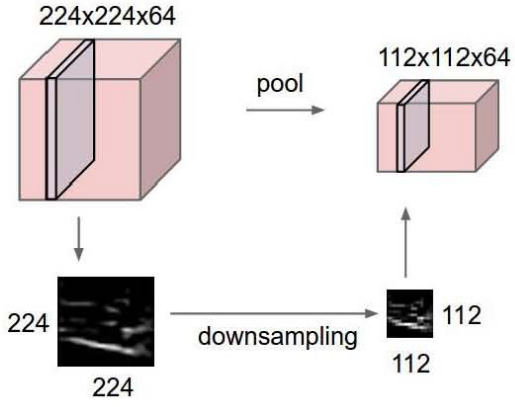
Convolution:Representation Learning



Pooling Layer



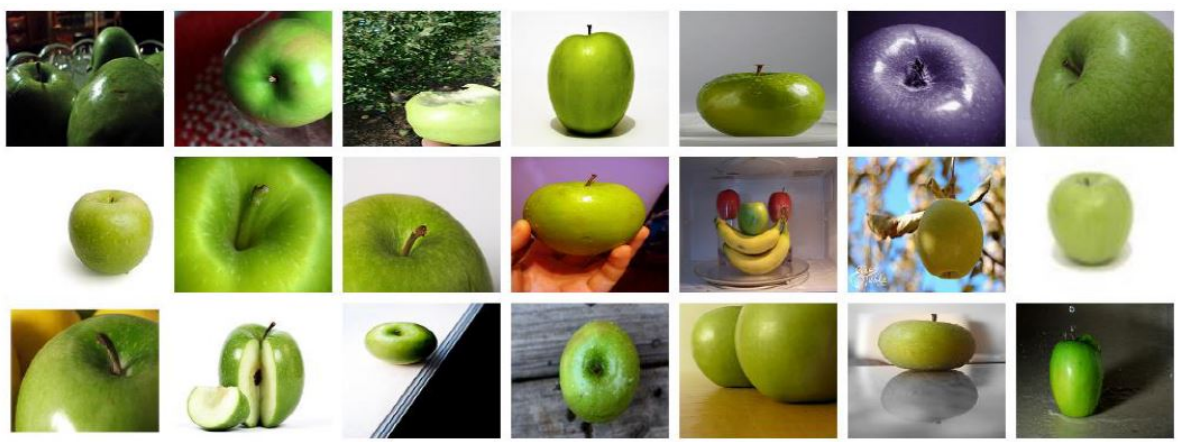




Outline

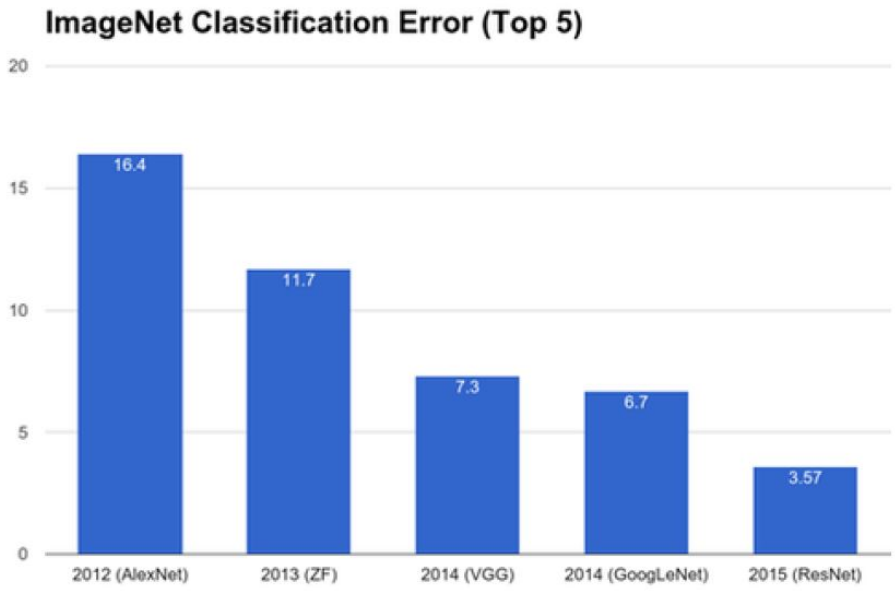
Object Recognition
Case Study: ImageNet

- **ImageNet:** dataset of 14+ million images (21,841 categories)
  - Links to images not images
- Lets take the high level category of **fruit** as an example:
  - Total 188,000 images of fruit
  - There are 1206 Granny Smith apples:



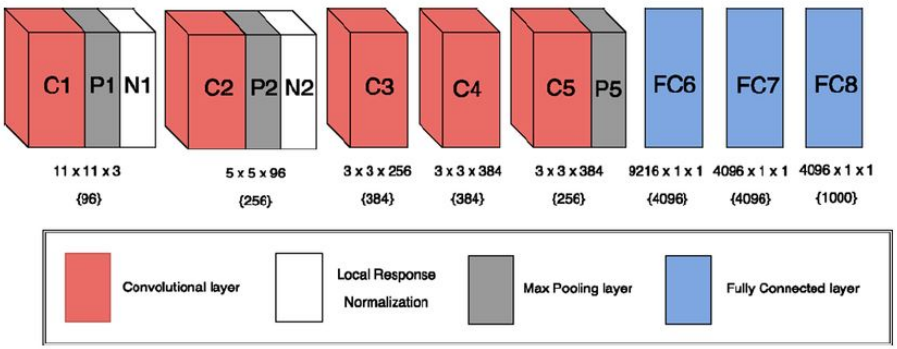
ImageNet

- **Competition:** ImageNet Large Scale Visual Recognition Challenge (ILSVRC)
- Networks
  - AlexNet (2012)
  - ZFNet (2013)
  - VGGNet (2014)
  - GoogLeNet (2014)
  - ResNet (2015)
  - CUImage (2016)



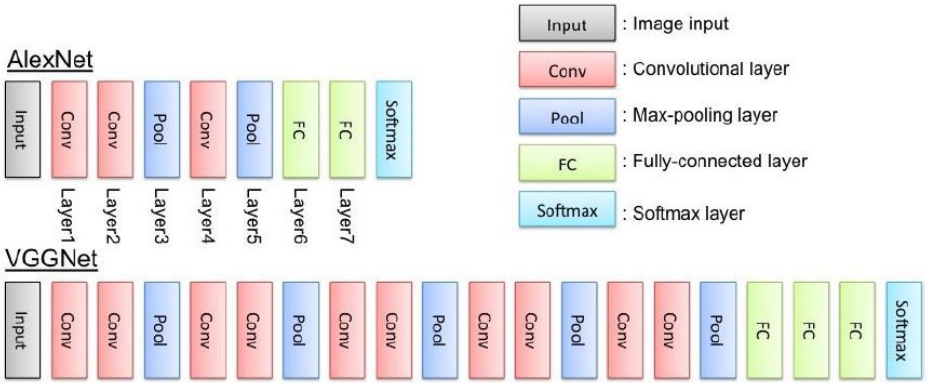
- **ResNet (2015): 6.7% to 3.57%**
  - More layers = better performance
  - 152 layers
- **CUImage (2016): 3.57% to 2.99%**
  - Ensemble of 6 models

- **AlexNet (2012): First CNN (15.4%)**
  - 8 layers
  - 61 million parameters
- **ZFNet (2013): 15.4% to 11.2%**
  - 8 layers
  - More filters. Denser stride.
- **VGGNet (2014): 11.2% to 7.3%**
  - Beautifully uniform:
    3x3 conv, stride 1, pad 1, 2x2 max pool
  - 16 layers
  - 138 million parameters
- **GoogLeNet (2014): 11.2% to 6.7%**
  - Inception modules
  - 22 layers
  - 5 million parameters
    (throw away fully connected layers)

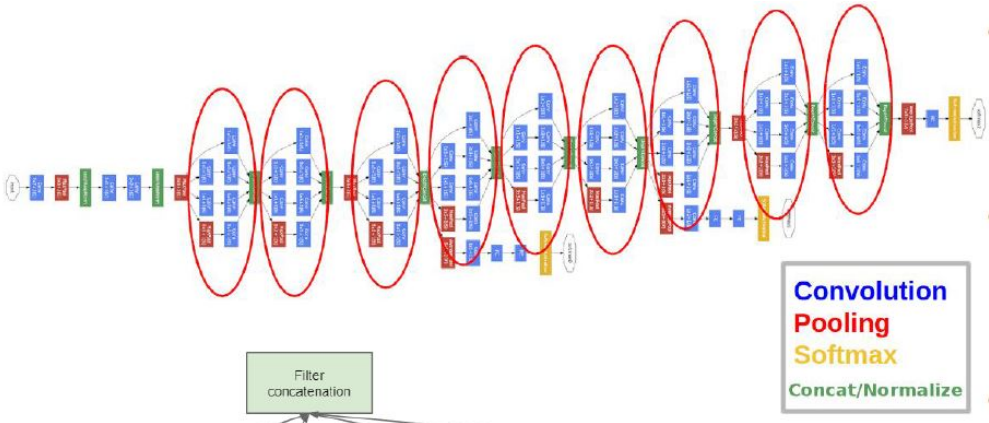


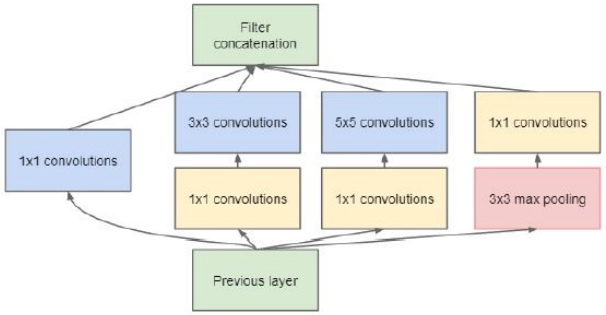
- **AlexNet (2012): First CNN (15.4%)**
  - 8 layers
  - 61 million parameters
- **ZFNet (2013): 15.4% to 11.2%**
  - 8 layers
  - More filters. Denser stride.
- **VGGNet (2014): 11.2% to 7.3%**
  - Beautifully uniform:
    3x3 conv, stride 1, pad 1, 2x2 max pool
  - 16 layers
  - 138 million parameters
- **GoogLeNet (2014): 11.2% to 6.7%**
  - Inception modules
  - 22 layers
  - 5 million parameters
    (throw away fully connected layers)

- **ResNet (2015): 6.7% to 3.57%**
  - More layers = better performance
  - 152 layers
- **CUImage (2016): 3.57% to 2.99%**
  - Ensemble of 6 models



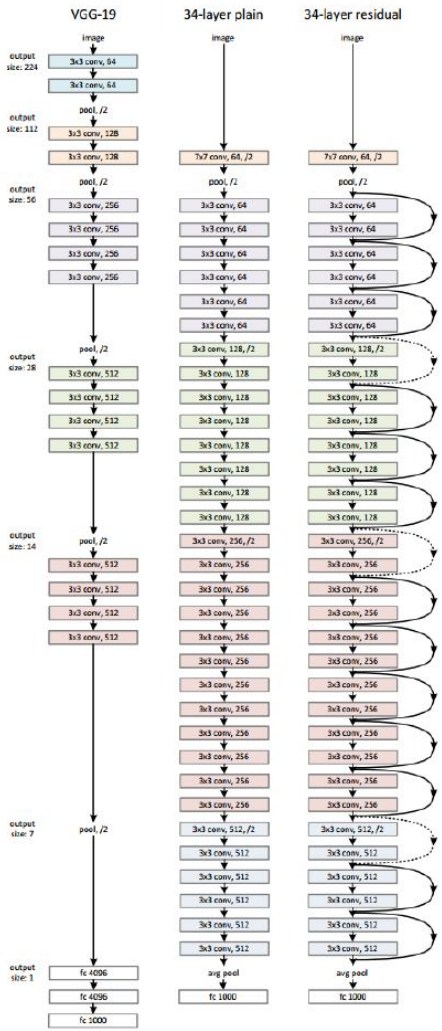
- **AlexNet (2012): First CNN (15.4%)**
  - 8 layers
  - 61 million parameters
- **ZFNet (2013): 15.4% to 11.2%**
  - 8 layers
  - More filters. Denser stride.
- **VGGNet (2014): 11.2% to 7.3%**
  - Beautifully uniform:
    3x3 conv, stride 1, pad 1, 2x2 max pool
  - 16 layers
  - 138 million parameters
- **GoogLeNet (2014): 11.2% to 6.7%**
  - Inception modules
  - 22 layers
  - 5 million parameters
    (throw away fully connected layers)
- **ResNet (2015): 6.7% to 3.57%**
  - More layers = better performance
  - 152 layers
- **CUImage (2016): 3.57% to 2.99%**
  - Ensemble of 6 models

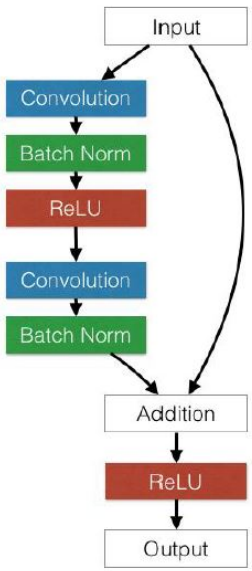




- **ResNet (2015): 6.7% to 3.57%**
  - More layers = better performance
  - 152 layers
- **CUImage (2016): 3.57% to 2.99%**
  - Ensemble of 6 models

- **AlexNet (2012): First CNN (15.4%)**
  - 8 layers
  - 61 million parameters
- **ZFNet (2013): 15.4% to 11.2%**
  - 8 layers
  - More filters. Denser stride.
- **VGGNet (2014): 11.2% to 7.3%**
  - Beautifully uniform:
    3x3 conv, stride 1, pad 1, 2x2 max pool
  - 16 layers
  - 138 million parameters
- **GoogLeNet (2014): 11.2% to 6.7%**
  - Inception modules
  - 22 layers
  - 5 million parameters
    (throw away fully connected layers)



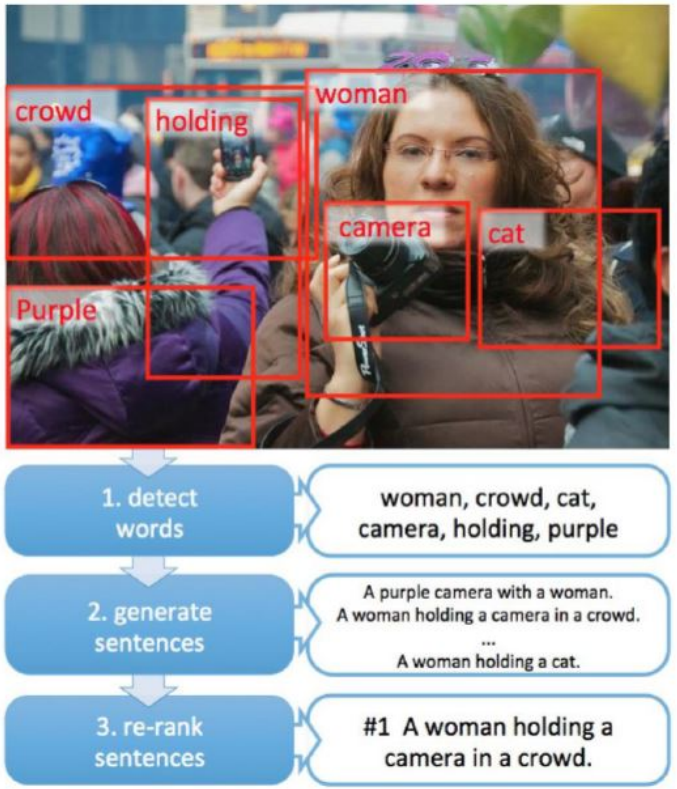


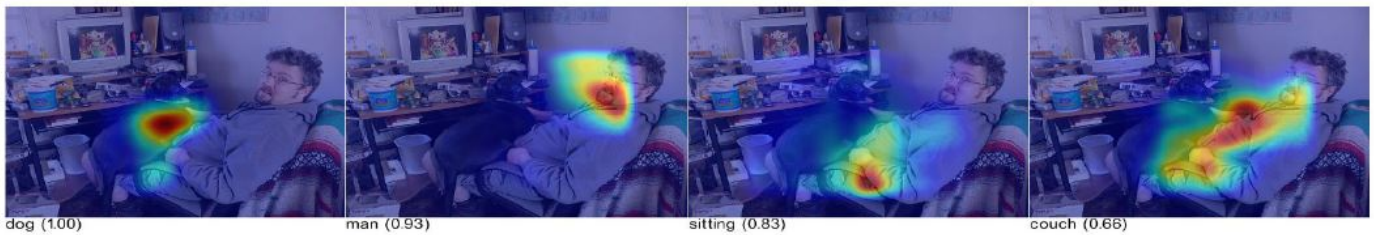
- **VGGNet (2014): 11.2% to 7.3%**
  - Beautifully uniform:
    3x3 conv, stride 1, pad 1, 2x2 max pool
  - 16 layers
  - 138 million parameters
- **GoogLeNet (2014): 11.2% to 6.7%**
  - Inception modules
  - 22 layers
  - 5 million parameters
    (throw away fully connected layers)
- **ResNet (2015): 6.7% to 3.57%**
  - More layers = better performance
  - 152 layers
- **CUImage (2016): 3.57% to 2.99%**
  - Ensemble of 6 models

Source: MIT 6.S1

Outline

Image Caption Generation

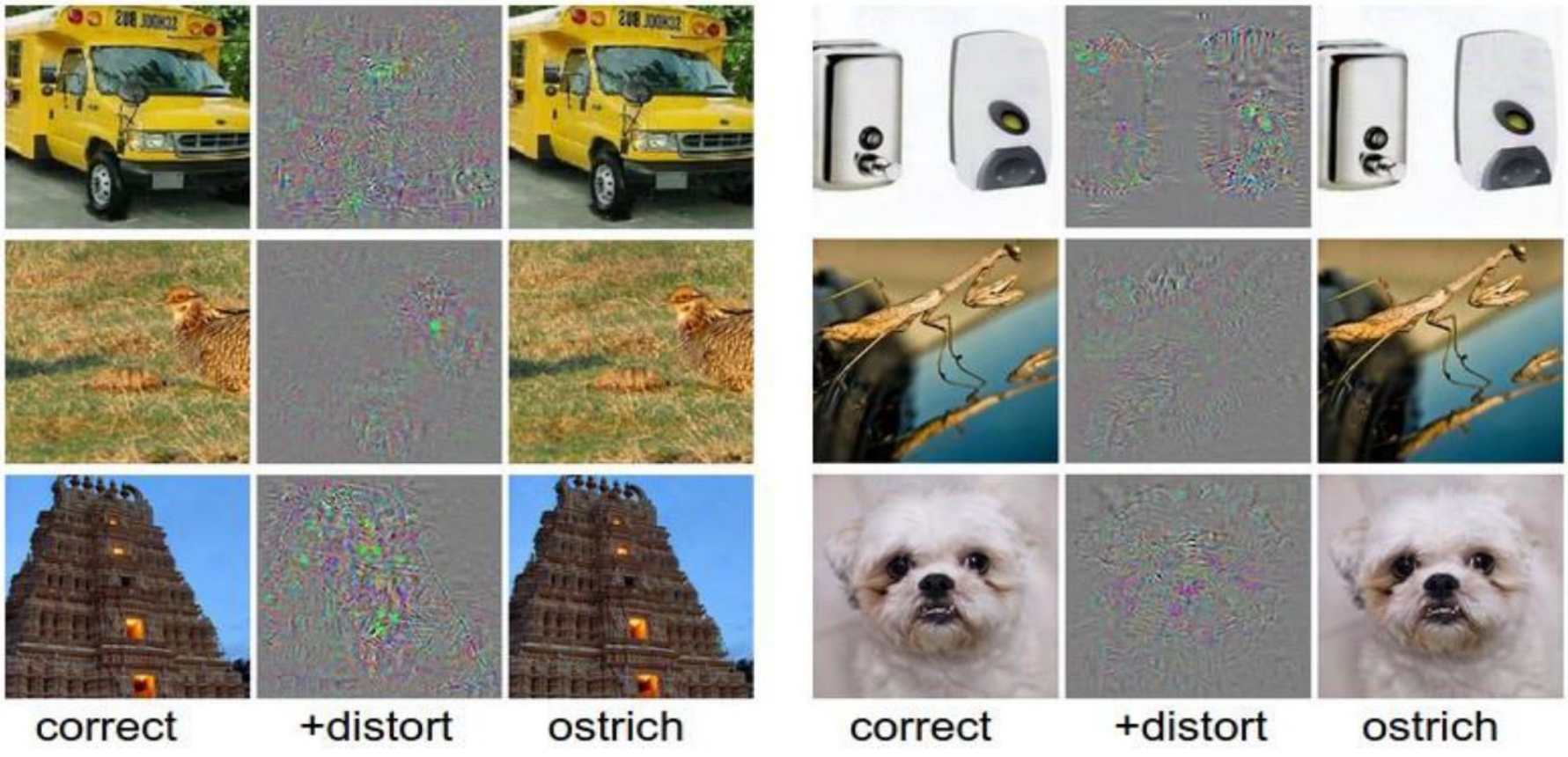




Source:

Outline

Fooled by a Little Distortion



Szegedy et al. "Intriguing properties of neural networks." 2013.

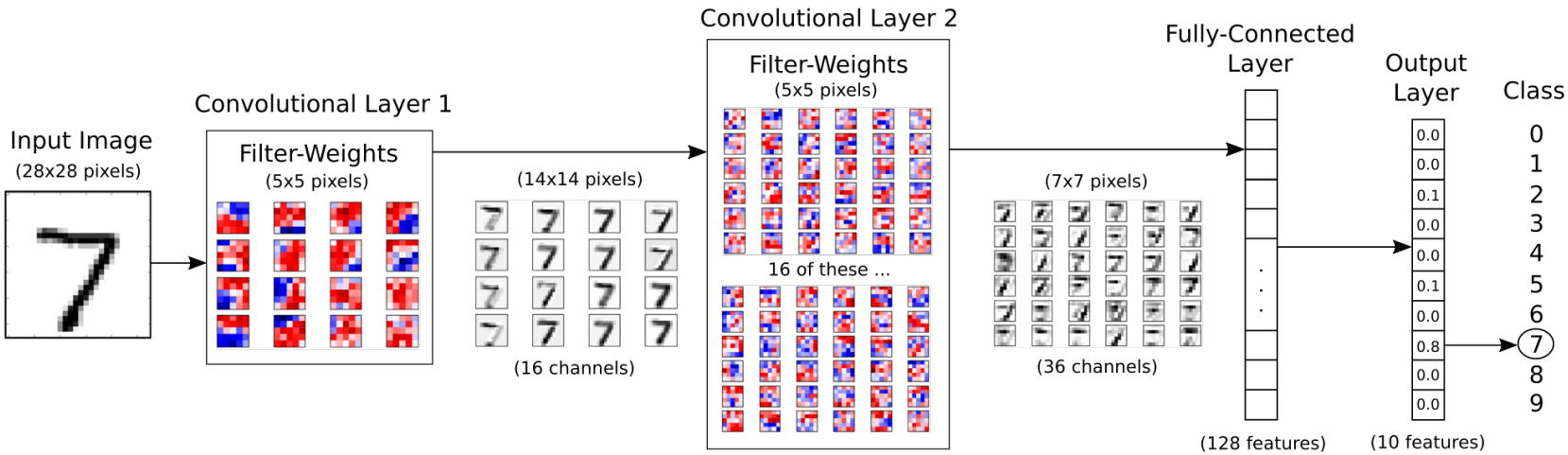
Object Category Recognition

Object Category Recognition

Object Category Recognition

Outline

CNN Architecture for Image Classification



CNN on MNIST Dataset

Load Dataset

```
import tensorflow as tf
import matplotlib.pyplot as plt
#Load dataset
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("/tmp/data/", one_hot = True)
```

Display of an Image

```
# Get the first images from the test-set.
images = mnist.test.images[0:9]
plt.imshow(images[0].reshape(28,28), cmap='binary')
plt.show()
```

CNN on MNIST Dataset

Variable Declaration

```
x = tf.placeholder(tf.float32, shape=[None, 784])
y_ = tf.placeholder(tf.float32, shape=[None, 10])

def weight_variable(shape):
  initial = tf.truncated_normal(shape, stddev=0.1)
  return tf.Variable(initial)

def bias_variable(shape):
  initial = tf.constant(0.1, shape=shape)
  return tf.Variable(initial)
```

CNN on MNIST Dataset

TensorFlow Operation for Convolution and Pooling Layer

- Strides are set to 1 in all dimensions
- The first and last stride must always be 1, because the first is for the image-number and the last is for the input-channel
- Strides=[1, 2, 2, 1] would mean that the filter is moved 2 pixels across the x- and y-axis of the image
- The padding is set to 'SAME' which means the input image is padded with zeroes so the size of the output is the same

```
def conv2d(x, W):
  return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):
  return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                        strides=[1, 2, 2, 1], padding='SAME')
```

CNN on MNIST Dataset

First Convolution and Pooling Layer

```
#Its weight tensor will have a shape of [5, 5, 1, 32].
#The first two dimensions are the patch size, the next
#is the number of input channels,
#and the last is the number of output channels.
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])

#First dim means for all. the second and third dimensions
#corresponding to image width and height,
#and the final dimension corresponding to the number of
#color channels.
x_image = tf.reshape(x, [-1, 28, 28, 1])

h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)
```

CNN on MNIST Dataset

Second Convolution and Pooling Layer

```
W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])

h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)
```

CNN on MNIST Dataset

Fully Connected, Pooling, and Dropout Layer

```
W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])

h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])

y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2
```

CNN on MNIST Dataset

Cross Entropy

```
cross_entropy = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits(labels=y_,
                                            logits=y_conv))
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
correct_prediction = tf.equal(tf.argmax(y_conv, 1),
                              tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction,
                                  tf.float32))
```

CNN on MNIST Dataset

Session Code

```
with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())
  for i in range(2000):
    batch = mnist.train.next_batch(50)
    if i % 100 == 0:
      train_accuracy = accuracy.eval(feed_dict={
          x: batch[0], y_: batch[1], keep_prob: 1.0})
      print('step %d, training accuracy %g' % (i,
                                                train_accuracy))
    train_step.run(feed_dict={x: batch[0], y_: batch[1],
                              keep_prob: 0.5})

  print('test accuracy %g' % accuracy.eval(feed_dict={
      x: mnist.test.images, y_: mnist.test.labels,
      keep_prob: 1.0}))
```

Summary

!For More Details!

https://www.tensorflow.org

Thank You