A TensorFlow Tutorial

Email Classification with Logistic Regression

Josh Meyer [1] Michael Capizzi [2]

[1] University of Arizona

[2] PitchVantage

Table of contents

1. What's Going to Happen
2. TensorFlow Structures
3. The Flow of TensorFlow

Tonight we will...

- describe the basic TensorFlow structures
- build a working example of text classification
- point out places where *other* TensorFlow "built-ins" apply (optimizers, cost functions, etc)
- "hand-wave" liberally

  (when we don't want to get into it or don't know the answer)

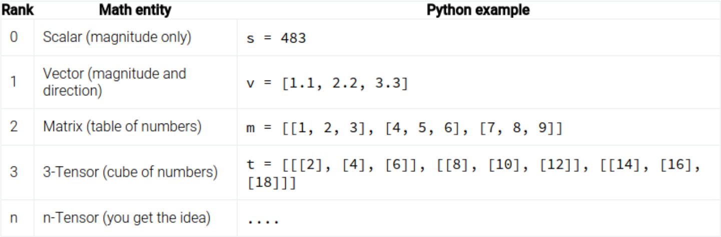
Tonight we will **not**...

- discuss details of NLP feature selection
- discuss details of Machine Learning

  (linear algebra, backpropogation, etc.)

TensorFlow Structures

tensor $=$ *n*-dimensional matrix



Rank	Math entity	Python example
0	Scalar (magnitude only)	`s = 483`
1	Vector (magnitude and direction)	`v = [1.1, 2.2, 3.3]`
2	Matrix (table of numbers)	`m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`
3	3-Tensor (cube of numbers)	`t = [[[2], [4], [6]], [[8], [10], [12]], [[14], [16], [18]]]`
n	n-Tensor (you get the idea)	`....`

TensorFlow Structures

- **constants**: *never* changes its value(s)

```
c = tf.constant(2.0, name="constantC") #can be int, float, or tensor
```

- **placeholders**: shell into which tensors can be iteratively inserted

```
X = tf.placeholder(tf.float32, [None, 200], name="input")
```

- **variables**: value(s) can be updated

```
weights = tf.Variable(tf.random_normal([1, 200], name="weights"))
```

- **operations**: computations that will act on tensors

```
apply_weights_OP = tf.matmul(X, weights, name="apply_weights")
add_bias_OP = tf.add(apply_weights_OP, bias, name="add_bias")
activation_OP = tf.nn.sigmoid(add_bias_OP, name="activation")
```

Let's get to the script!

Preamble

```
####################
#### PREAMBLE ####
####################

from __future__ import division
import tensorflow as tf
import numpy as np
import tarfile
import os
import matplotlib.pyplot as plt
import time
```

Import the Email Data

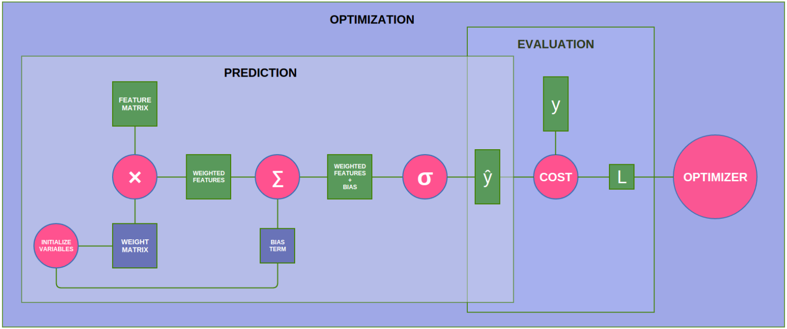
```
######################
### IMPORT DATA ###
######################

def csv_to_numpy_array(filePath, delimiter):
    return np.genfromtxt(filePath, delimiter=delimiter, dtype=None)

def import_data():
    if "data" not in os.listdir(os.getcwd()):
        # Untar directory of data if we haven't already
        tarObject = tarfile.open("data.tar.gz")
        tarObject.extractall()
        tarObject.close()
        print("Extracted tar to current directory")
    else:
        # we've already extracted the files
        pass

    print("loading training data")
    trainX = csv_to_numpy_array("data/trainX.csv", delimiter="\t")
    trainY = csv_to_numpy_array("data/trainY.csv", delimiter="\t")
    print("loading test data")
    testX = csv_to_numpy_array("data/testX.csv", delimiter="\t")
    testY = csv_to_numpy_array("data/testY.csv", delimiter="\t")
    return trainX,trainY,testX,testY

trainX,trainY,testX,testY = import_data()
```

Some Global Parameters

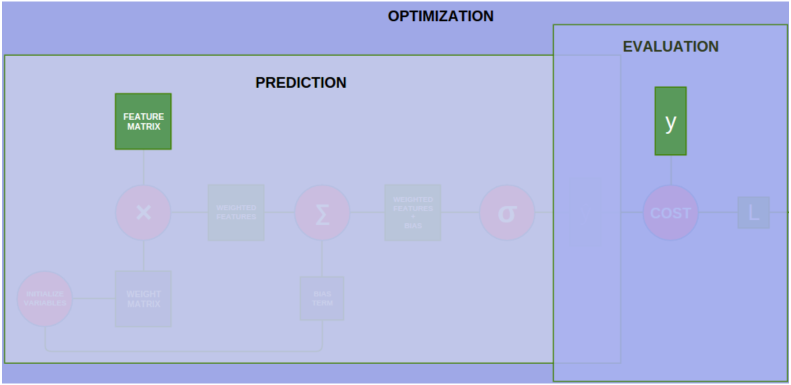
```
#################################
### GLOBAL PARAMETERS ###
#################################

# DATA SET PARAMETERS
# Get our dimensions for our different variables and placeholders:
# numFeatures = the number of words extracted from each email
numFeatures = trainX.shape[1]
# numLabels = number of classes we are predicting (here just 2: Ham or Spam)
numLabels = trainY.shape[1]

# TRAINING SESSION PARAMETERS
# number of times we iterate through training data
# tensorboard shows that accuracy plateaus at ~25k epochs
numEpochs = 27000
# a smarter learning rate for gradientOptimizer
learningRate = tf.train.exponential_decay(learning_rate=0.0008,
                                          global_step= 1,
                                          decay_steps=trainX.shape[0],
                                          decay_rate= 0.95,
                                          staircase=True)
```

The Computational Graph

The Full Computational Graph



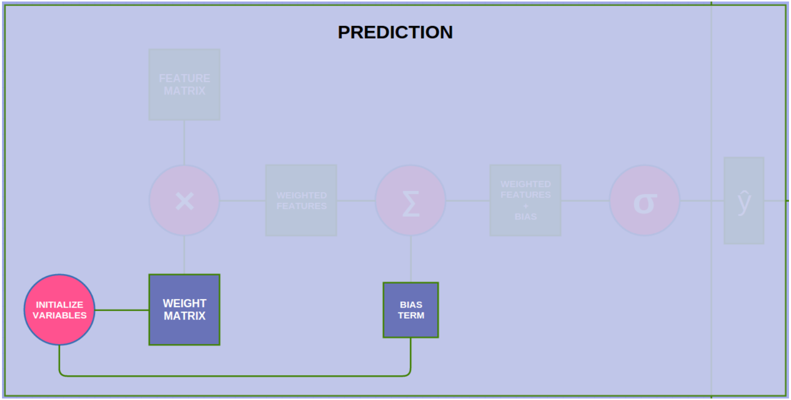
Define Feature and Label Placeholders



Define Feature and Label Placeholders

```
####################
### PLACEHOLDERS ###
####################

# X = X-matrix / feature-matrix / data-matrix... It's a tensor to hold our
# email data. 'None' here means that we can hold any number of emails
X = tf.placeholder(tf.float32, [None, numFeatures])

# yGold = Y-matrix / label-matrix / labels... This will be our correct answers
# matrix. Every row has either [1,0] for SPAM or [0,1] for HAM. 'None' here
# means that we can hold any number of emails
yGold = tf.placeholder(tf.float32, [None, numLabels])
```

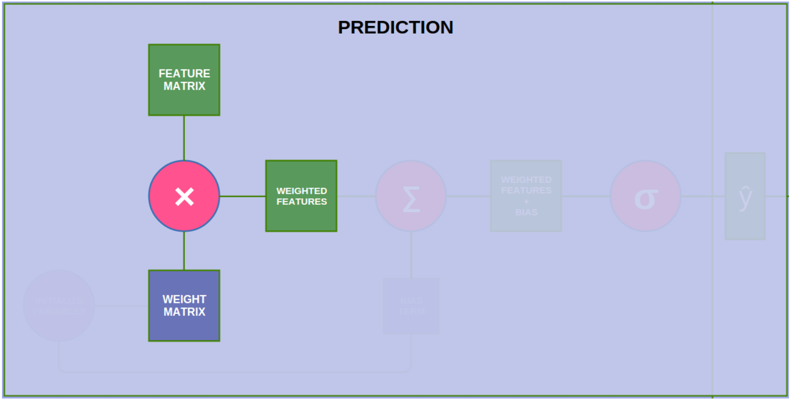
Initialize Weights & Bias Terms Op



Initialize Weights & Bias Terms Op

```
#################
### VARIABLES ###
#################

# all values are randomly assigned:
# sqrt(6 / (numInputNodes + numOutputNodes + 1))

weights = tf.Variable(tf.random_normal([numFeatures,numLabels],
          mean=0,
          stddev=(np.sqrt(6/numFeatures+numLabels+1)),
          name="weights"))

bias = tf.Variable(tf.random_normal([1,numLabels],
       mean=0,
       stddev=(np.sqrt(6/numFeatures+numLabels+1)),
       name="bias"))


# INITIALIZE our weights and biases
init_OP = tf.initialize_all_variables()
```

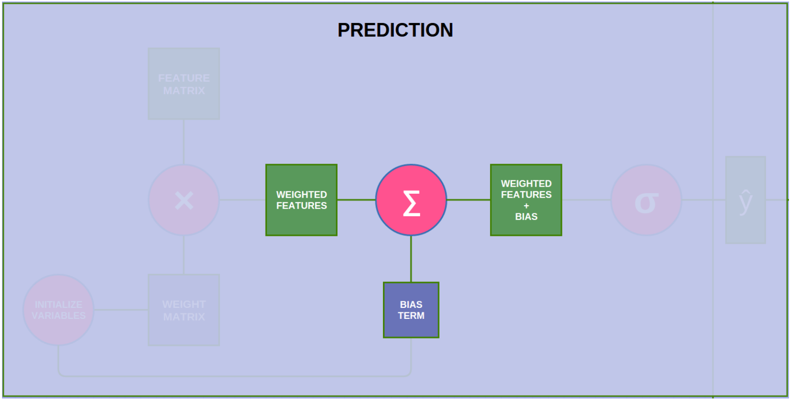
Apply Weights to Features Op



Apply Weights to Features Op

```
apply_weights_OP = tf.matmul(X, weights, name="apply_weights")
```

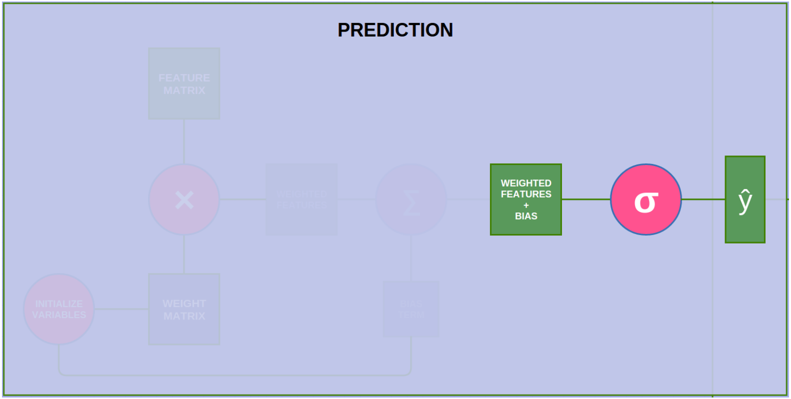
Add Bias to Weighted Features Op



Add Bias to Weighted Features Op

```
add_bias_OP = tf.add(apply_weights_OP, bias, name="add_bias")
```

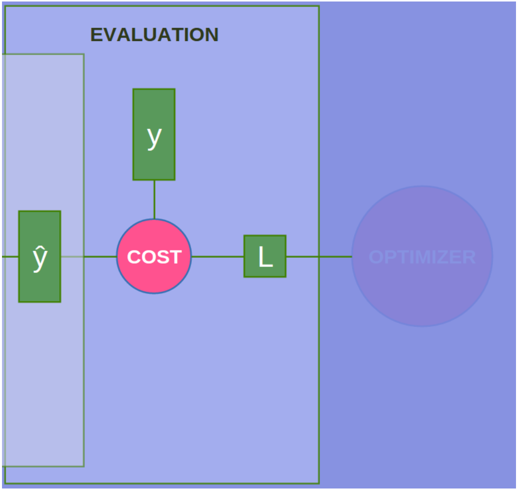
Activation Op



Activation Op

```
activation_OP = tf.nn.sigmoid(add_bias_OP, name="activation")
```

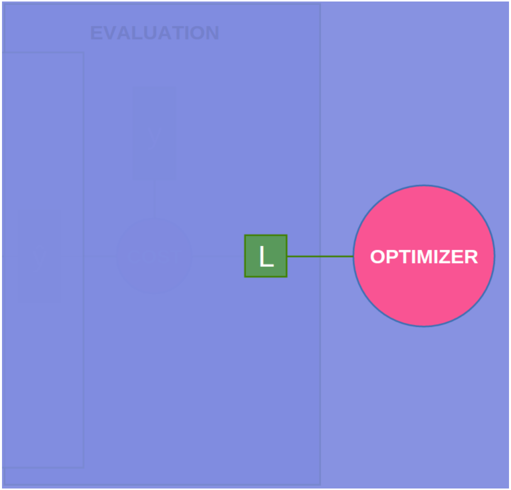
Evaluation Op: Mean Squared Error



Evaluation Op: Mean Squared Error

```
#####################
### EVALUATION OP ###
#####################

# COST FUNCTION i.e. MEAN SQUARED ERROR
cost_OP = tf.nn.l2_loss(activation_OP-yGold, name="squared_error_cost")
```

Optimization Op: Gradient Descent



Optimization Op: Gradient Descent

```
#######################
### OPTIMIZATION OP ###
#######################

# OPTIMIZATION ALGORITHM i.e. GRADIENT DESCENT
training_OP = tf.train.GradientDescentOptimizer(learningRate).minimize(cost_OP)
```

Run the Graph

```
#####################
### RUN THE GRAPH ###
#####################

# Create a tensorflow session
sess = tf.Session()
# Initialize all tensorflow variables
sess.run(init_OP)

## Ops for vizualization
# argmax(activation_OP, 1) gives the label our model thought was most likely
# argmax(yGold, 1) is the correct label
correct_predictions_OP=tf.equal(tf.argmax(activation_OP,1),tf.argmax(yGold,1))
# False is 0 and True is 1, what was our average?
accuracy_OP = tf.reduce_mean(tf.cast(correct_predictions_OP, "float"))
# Summary op for regression output
activation_summary_OP = tf.histogram_summary("output", activation_OP)
# Summary op for accuracy
accuracy_summary_OP = tf.scalar_summary("accuracy", accuracy_OP)
# Summary op for cost
cost_summary_OP = tf.scalar_summary("cost", cost_OP)
# Summary ops to check how variables (W, b) are updating after each iteration
weightSummary = tf.histogram_summary("weights", weights.eval(session=sess))
biasSummary = tf.histogram_summary("biases", bias.eval(session=sess))
# Merge all summaries
all_summary_OPS = tf.merge_all_summaries()
# Summary writer
writer = tf.train.SummaryWriter("summary_logs", sess.graph_def)
```

Still 'Running the Graph'

```
# Initialize reporting variables
cost = 0
diff = 1
# Training epochs
for i in range(numEpochs):
    if i ¿ 1 and diff ¡ .0001:
        print("change in cost %g; convergence."%diff)
        break
    else:
        # Run training step
        step = sess.run(training_OP, feed_dict={X: trainX, yGold: trainY})
        # Report occasional stats
        if i % 10 == 0:
            # Add epoch to epoch_values
            epoch_values.append(i)
            # Generate accuracy stats on test data
            summary_results, train_accuracy, newCost = sess.run(
                [all_summary_OPS, accuracy_OP, cost_OP],
                feed_dict={X: trainX, yGold: trainY}
            )
            # Add accuracy to live graphing variable
            accuracy_values.append(train_accuracy)
            # Add cost to live graphing variable
            cost_values.append(newCost)
            # Write summary stats to writer
            writer.add_summary(summary_results, i)
            # Re—assign values for variables
            diff = abs(newCost - cost)
            cost = newCost
```

Still 'Still Running the Graph'

```
        #generate print statements
        print("step %d, training accuracy %g"%(i, train_accuracy))
        print("step %d, cost %g"%(i, newCost))
        print("step %d, change in cost %g"%(i, diff))

        # Plot progress to our two subplots
        accuracyLine, = ax1.plot(epoch_values, accuracy_values)
        costLine, = ax2.plot(epoch_values, cost_values)
        fig.canvas.draw()
        time.sleep(1)


# How well do we perform on held-out test data?
print("final accuracy on test set: %s" %str(sess.run(accuracy_OP,
                                     feed_dict={X: testX,
                                           yGold: testY})))
```

Reuse, Recycle

```
##############################
### SAVE TRAINED VARIABLES ###
##############################

# Create Saver
saver = tf.train.Saver()
# Save variables to .ckpt file
# saver.save(sess, "trained_variables.ckpt")

# Close tensorflow session
sess.close()
```

That's all Folks!