Python Notes

IF/ELSE

if statements end with :

if preceded by if instead of elseif

STRINGS

strings denoted by “ “

check index of string by variable\_name[:n]

concatenate strings using +

INDEXING

Starts at 0

[:n] all indices up to but not including the nth

[n:] all indices from n on

FOR LOOP

for i in range(n) #from i = 1 to n

OPERATORS

% is modulus

COOL INDEXING THING

for i, n in enumerate(range):

i keeps track of index

MATHEMATICS LIBRARY (numpy) from numpy import exp, array, dot, random

Exp – e^x

Array – creates matrix

.T transposes matrix

Dot – multiplies matrices

Random – gives random numbers

IMAGE READING LIBRARY (OpenCV) for reading in image files

Import cv2

Cv2.imread(file,flag)

File is the path to the file

Flag determines how you want to read it in, no flag = standard, 0 = grayscale

REFERENCING LIBRARY (glob) for referencing multiple files from a folder

Import glob

For file in glob.glob(“path\*.ext”)

Path is the path to a folder of files and .ext is the extension of the files

GOOGLE COLAB

Data Structures

from \_\_future\_\_ import print\_function

import pandas as pd

like a standard library for data structures

pd is the class name (can change to anything)

pd.Series([]) creates a list of objects indexing from 0

pd.DataFrame({}) creates a table of named lists

* names=pd.Series([‘John’,’Jacob’,’Shmidt’])

ages=pd.Series([‘1’,’2’,’3’])

pd.DataFrame({‘Name’: names, ‘Age’ :ages})

creates a table with 2 columns Name and Age using the series

pd.read\_csv(“filename.csv”,sep=”,”)

reads a file with given delimiter “,”

var.describe() gives info about the table var

var.head() outputs the first few rows of var

var.hist(‘Name of graph’) create a graph with the given name

print(type(var)) tells u the class of the object

var[‘Series name’] prints the entire given series

var[‘Series name’][index] gives u the specified value

var[‘New series’]=… adds new column to var

can index with a single number or 0:2 for the first 2 values

can perform basic arithmetic on series or…

Importing Files from Drive

Math

import numpy as np

Math library with class name np

np.log(var) takes the log of all the values in series var

Series.apply(lambda val: val>10)

Looks through series and creates new bool series with val>10 as the if statement

Series.apply(lambda name: name.startswith(‘First couple letters’))

Checks if values in series starts with these letters

Series.index gives beginning end and step indices

Var.reindex([2, 0, 1]) reorders the series(s)

var.reindex(np.random.permutation(var.index)) randomly reorders

reindex can use non valued indices, but will return nan values

nice for processing data that has gaps

TensorFlow <2.0 Garbage only here for reference

import tensorflow as tf

tf is class name

import matplotlib.pyplot as plt # Dataset visualization.

import numpy as np # Low-level numerical Python library.

import pandas as pd # Higher-level numerical Python library.

Session

with tf.Session() as sess:

initialization = tf.global\_variables\_initializer()

graphs and therefore tensorflow need a session to work in that holds their state, initialization should be enough but create multiple sessions for network that works with multiple systems simultaneously

var = tf.constant([num])

server = tf.train.Server.create\_local\_server() #creates single-process cluster

sess = tf.Session(server.target) # Create a session on the server.

sess.run(var)

further documentation

<https://github.com/tensorflow/examples/blob/master/community/en/docs/deploy/distributed.md>

Tensors

Scalar = single value

Vector = array

Matirx = 2d array

Operations

Create, destroy and manipulate

Var = tf.constant(num) creates a constant value

Var = tf.Variable([num]) creates a variable that can be reassigned

Var = var.assign([newNum]) redefines var

Functions like tf.add do what you would expect

Graph

Like the data structure

Nodes are operations, edges are tensors

g=tf.Graph()

ex of graph implemented as default to add 2 constants

from \_\_future\_\_ import print\_function

import tensorflow as tf

# Create a graph.

g = tf.Graph()

# Establish the graph as the "default" graph.

with g.as\_default():

# Assemble a graph consisting of the following three operations:

# \* Two tf.constant operations to create the operands.

# \* One tf.add operation to add the two operands.

x = tf.constant(8, name="x\_const")

y = tf.constant(5, name="y\_const")

my\_sum = tf.add(x, y, name="x\_y\_sum")

# names are unneeded

# Now create a session.

# The session will run the default graph.

with tf.Session() as sess:

print(my\_sum.eval())

.eval tells the print function to throw the value instead of the object (my\_sum)’s properties

Creating simple predictive system

Define input feature:

Input\_feature = var[[“Series”]]

Configure numerical column for input\_feature:

numeric\_columns=[tf.feature\_column.numeric\_column(“Series”)]

Define target:

Target = var[“Series”]

Make optimizer for training the model:

my\_optimizer=tf.train.GradientDescentOptimizer(learning\_rate=0.000001)

Clip gradients to prevent optimization failure:

my­\_optimizer=tf.contrib.estimator.clip\_gradients\_by\_norm(optimizer, 5)

Configure linear regression model with input and target:

Linear\_regressor=tf.estimator.LinearRegressor(feature\_columns=

numeric\_columns,optimizer=my\_optimizer)

Create input function for linear\_regressor (normally use batch\_size=32 or 64)

def my\_input\_fn(features, targets, batch\_size=1, shuffle=True, num\_epochs=None):

"""Trains a linear regression model of one feature.

Args:

features: pandas DataFrame of features

targets: pandas DataFrame of targets

batch\_size: Size of batches to be passed to the model

shuffle: True or False. Whether to shuffle the data.

num\_epochs: Number of epochs for which data should be repeated. None = repeat indefinitely

Returns:

Tuple of (features, labels) for next data batch

"""

# Convert pandas data into a dict of np arrays.

features = {key:np.array(value) for key,value in dict(features).items()}

# Construct a dataset, and configure batching/repeating.

ds = Dataset.from\_tensor\_slices((features,targets)) # warning: 2GB limit

ds = ds.batch(batch\_size).repeat(num\_epochs)

# Shuffle the data, if specified.

if shuffle:

ds = ds.shuffle(buffer\_size=10000)

# Return the next batch of data.

features, labels = ds.make\_one\_shot\_iterator().get\_next()

return features, labels

Train linear regressor with train() function with input function and iteration number as arguments (this is the model learning)

\_=linear\_regressor.train(input\_fn = lambda:my\_input\_fn(my\_feature, targets), steps=100)

Evaluate model for accuracy using rmse

# Create an input function for predictions.

# Note: Since we're making just one prediction for each example, we don't

# need to repeat or shuffle the data here.

prediction\_input\_fn =lambda: my\_input\_fn(my\_feature, targets, num\_epochs=1, shuffle=False)

# Call predict() on the linear\_regressor to make predictions.

predictions = linear\_regressor.predict(input\_fn=prediction\_input\_fn)

# Format predictions as a NumPy array, so we can calculate error metrics.

predictions = np.array([item['predictions'][0] for item in predictions])

# Print Mean Squared Error and Root Mean Squared Error.

mean\_squared\_error = metrics.mean\_squared\_error(predictions, targets)

root\_mean\_squared\_error = math.sqrt(mean\_squared\_error)

print("Mean Squared Error (on training data): %0.3f" % mean\_squared\_error)

print("Root Mean Squared Error (on training data): %0.3f" % root\_mean\_squared\_error)

Analyze data to see if this error is acceptable

min\_house\_value = california\_housing\_dataframe["median\_house\_value"].min()

max\_house\_value = california\_housing\_dataframe["median\_house\_value"].max()

min\_max\_difference = max\_house\_value - min\_house\_value

print("Min. Median House Value: %0.3f" % min\_house\_value)

print("Max. Median House Value: %0.3f" % max\_house\_value)

print("Difference between Min. and Max.: %0.3f" % min\_max\_difference)

print("Root Mean Squared Error: %0.3f" % root\_mean\_squared\_error)

TensorFlow 2.0 (the good one)

<https://keras.io/#getting-started-30-seconds-to-keras>

<https://elitedatascience.com/keras-tutorial-deep-learning-in-python>

<https://keras.io/optimizers/>

from \_\_future\_\_ import absolute\_import, division, print\_function, unicode\_literals

# TensorFlow and tf.keras

import tensorflow as tf

from tensorflow import keras

# Helper libraries

import numpy as np

import matplotlib.pyplot as plt

print(tf.\_\_version\_\_) #if version isn’t 2.0.0 put these few lines above the rest:

try:

# %tensorflow\_version only exists in Colab.

%tensorflow\_version 2.x

except Exception:

pass

Import data set: example fashion\_mnist

fashion\_mnist = keras.datasets.fashion\_mnist

(train\_images, train\_labels), (test\_images, test\_labels) = fashion\_mnist.load\_data()

With fashion\_mnist the clothing labels are designated by numbers, this is common but its important to keep straight what each number represents so make an array of labels that correspond accordingly

class\_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',

'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

.shape tells you the number and dimensions of images in a variable

len() tells you the dimension of the array

next preprocess the data:

plt.figure() #makes the figure

plt.imshow(train\_images[0]) #specify what the plot is of

plt.colorbar() #add a color bar (0-255 grayscale value)

plt.grid(False) #specify is you want the plot gridded or not

plt.show() #make it pretty by getting rid of unnecessary outputs

Neural networks work with coefficients 0-1 so scale values:

train\_images = train\_images / 255.0

test\_images = test\_images / 255.0

plt.figure(figsize=(10,10))

for i in range(25):

plt.subplot(5,5,i+1) #nrows, ncolumns describes the grid and index is what position in the grid plot takes

plt.xticks([]) #specify what to label on x axis

plt.yticks([]) #same for y axis

plt.imshow(train\_images[i], cmap=plt.cm.binary) #show iamges and convert to grayscale

plt.xlabel(class\_names[train\_labels[i]]) #names of each thing

plt.show()

Build the model

Set up layers

model = keras.Sequential([

keras.layers.Flatten(input\_shape=(28, 28)), #formats images from 2d array:28 by 28 to 1d array:28\*28=784

keras.layers.Dense(128, activation='relu'), #128 nodes (neurons) learning function

keras.layers.Dense(10, activation='softmax') #10 nodes because 10 options for classification

#returns array of 10 with probabilities for the image belonging to the class of that index (sum\_total=1)

])

Compile the model

model.compile(optimizer=’’, loss=’’, metrics=[‘’])

Optimizer: how the model is updated, uses input data and loss function

Loss: measure of accuracy during training (want to minimize output of this function)

Metrics: tells how good the model is

Ex. model.compile(optimizer=’adam’, loss=’ sparse\_categorical\_crossentropy’, metrics=[‘accuracy’])

<https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Optimizer> more info

Model is Done! Train it:

model.fit(input\_data, classification\_options, epochs) #epochsiterations

Ex. model.fit(train\_images, train\_labels, epochs=10)

Let’s see how good it is with new data

lossVar, accuracyVar = model.evaluate(test\_input, test\_class)

Ex. test\_loss, test\_acc =model.evaluate(test\_images, test\_labels)

print(‘\nTest accuracy:’, test\_acc) #print result of test

Hopefully the accuracy is pretty good, if not go fix some stuff and try again

Low test accuracy compared to training accuracy is due to overfitting (research if the gap is large to see how to minimize this)

Once model is passing tests you can use it to make predictions about new input data

predictVar=model.predict(test\_data)

predictVar[0] will print out the calculated probabilities for classifying the first piece of data

np.argmax(predictVar[0]) will print out the index in the classification array of which the predicted probability is the highest

set up analysis plotting (don’t worry about learning this, if needed plug in new variables to this exact code)

makes histograms show up blue if the prediction was correct and red if it was not

def plot\_image(i, predictions\_array, true\_label, img):

predictions\_array, true\_label, img = predictions\_array, true\_label[i], img[i]

plt.grid(False)

plt.xticks([])

plt.yticks([])

plt.imshow(img, cmap=plt.cm.binary)

predicted\_label = np.argmax(predictions\_array)

if predicted\_label == true\_label:

color = 'blue'

else:

color = 'red'

plt.xlabel("{} {:2.0f}% ({})".format(class\_names[predicted\_label],

100\*np.max(predictions\_array),

class\_names[true\_label]),

color=color)

def plot\_value\_array(i, predictions\_array, true\_label):

predictions\_array, true\_label = predictions\_array, true\_label[i]

plt.grid(False)

plt.xticks(range(10))

plt.yticks([])

thisplot = plt.bar(range(10), predictions\_array, color="#777777")

plt.ylim([0, 1])

predicted\_label = np.argmax(predictions\_array)

thisplot[predicted\_label].set\_color('red')

thisplot[true\_label].set\_color('blue')

Show image and histogram of what classifications the model believes each piece of data belongs in

num\_rows = 5

num\_cols = 3

num\_images = num\_rows\*num\_cols

plt.figure(figsize=(2\*2\*num\_cols, 2\*num\_rows))

for i in range(num\_images):

plt.subplot(num\_rows, 2\*num\_cols, 2\*i+1)

plot\_image(i, predictions[i], test\_labels, test\_images)

plt.subplot(num\_rows, 2\*num\_cols, 2\*i+2)

plot\_value\_array(i, predictions[i], test\_labels)

plt.tight\_layout()

plt.show()

Test model on single piece of new data:

Keras models work with lists so you need to make the data into a list of 1

data=np.expand\_dims(test\_data[0],0) #first arg is the data you’re reformatting, second arg is the dimension you’re adding

if test\_data is 28, 28 0 makes it 1, 28, 28 1: 28, 1, 28 2: 28, 28, 1

Make prediction

predict1=model.predict(data)

plot\_value\_array(1,predict1[0],test\_labels)

Make plot cooler by adding:

\_=plt.xticks(range(10), class\_names, rotation=45)

Range = number of ticks, class\_names = labels, rotation = label rotation

\_ = plt.yticks(range(4), (['0']))

\_ = plt.ylabel('Probability') # \_= stops functions from outputting

Find predicted output

class\_names[np.argmax(predict1[0])]

OPENCV COLAB IMPLEMENTATION

Reading in an image

cv2.imread(r’path)