IT8303
AI HUMAN INTERFACE

Lab 3
Deep Learning with Python

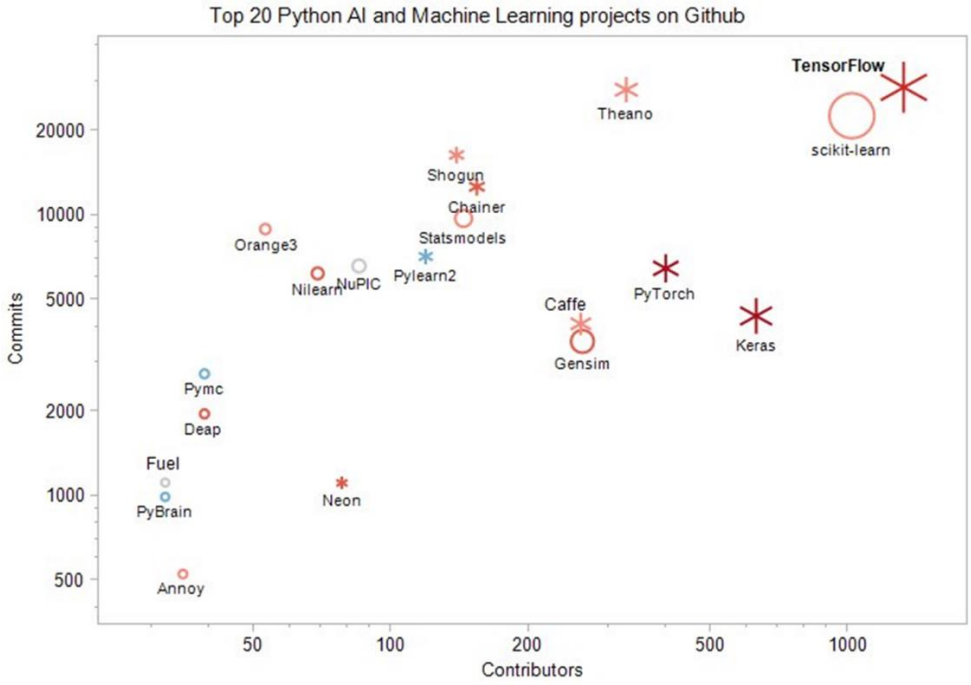
What you will learn / do in this lab

1. *Explore Deep Learning tools for Python*
2. *Conduct experiment using TensorFlow*
3. *Conduct experiment using Keras*

TABLE OF CONTENTS

1.
OVERVIEW

Python is a popular language for Deep Learning. Several Deep Learning libraries or frameworks. Amount the two most popular for python are TensorFlow and Keras.



INTRODUCTION TO TENSORFLOW

TensorFlow (http://www.tensorflow.org) is created as a mathematical library. However, it's data structures and functions, and efficient computation make it very suited for many machine learning algorithms and deep learning algorithms.

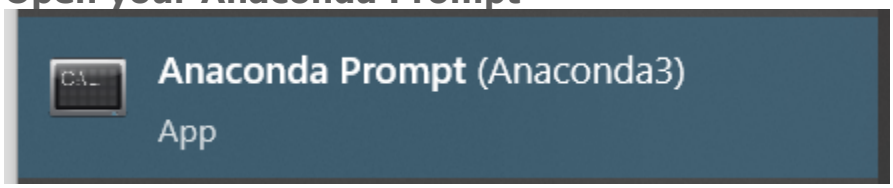
TensorFlow is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API.

TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well.

TensorFlow can run with only CPU (no graphics accelerator) but will run faster with Nvidia GPU installed.

Tensorflow Installation

Open your Anaconda Prompt



Type the following command:

```
pip install tensorflow==2.10.0
```

After installation, you can use the command below to check whether the installation is successful

```
pip show tensorflow
```

TensorFlow Hello World

Here is a simple "Hello World" program to just test the installation. You can run the code in your Python Jupyter Notebook.

```
# TensorFlow Hello World
import tensorflow as tf
print(tf.__version__)

a = tf.constant([10,11])
b = tf.constant(32)
c = a + b
print(c)
```

It shows how you can define constants and perform computation with those constants.

INTRODUCTION TO KERAS

Keras (http://keras.io) is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation.

In Tensorflow 2.0, Keras is integrated into TensorFlow as its high-level API library. The new package location is tf.keras

Keras Hello World

Here is a simple "Hello World" program to just test the installation

```
# Keras Hello World
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
print(tf.keras.__version__)

# create model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam',
              metrics=['accuracy'])
print(model.summary())
```

We can piece it all together by adding each layer. The first layer has 12 neurons and expects 8 input variables. The second hidden layer has 8 neurons and finally, the output layer has 1 neuron to predict the class.

2.
TENSORFLOW

In this section, we will be using the TensorFlow library to run some simple examples.

Computation in TensorFlow is described in terms of data flow and operations in the structure of a directed graph.

- *Nodes: Nodes perform computation and have zero or more inputs and outputs. Data that moves between nodes are known as tensors, which are multi-dimensional arrays of real values.*
- *Edges: The graph defines the flow of data, branching, looping and updates to state. Special edges can be used to synchronize behavior within the graph, for example waiting for computation on a number of inputs to complete.*
- *Operation: An operation is a named abstract computation which can take input attributes and produce output attributes. For example, you could define an add or multiply operation.*

LINEAR REGRESSSION WITH TENSORFLOW

This example shows how you can define variables (e.g. W and b) as well as variables that are the result of computation (y). W is the linear regression coefficient b1 and b is the constant.

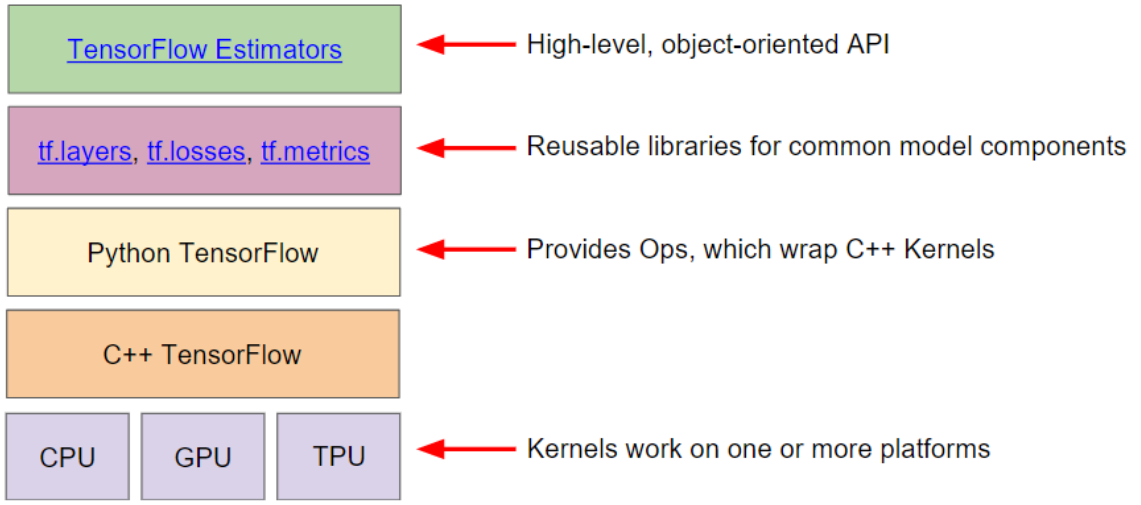
y = b1*x + … b

We get some sense of TensorFlow separates the definition and declaration of the computation from the execution in the session and the calls to run.

```
# Linear Regression using Tensorflow
import tensorflow as tf
import numpy as np

# Create 100 phony x, y data points in NumPy, y = x * 0.1 + 0.3
x_data = np.random.rand(100).astype(np.float32)
y_data = x_data * 0.1 + 0.3

# Try to find values for W and b that compute y_data = W * x_data + b
# (We know that W should be 0.1 and b 0.3, but Tensorflow will
# figure that out for us.)
W = tf.Variable(tf.random.uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))

# Minimize the mean squared errors.
def loss(W,b):
  # linear model y = W * x + b
  yhat = W * x_data + b
  # loss = mean(square(y_true - y_pred), axis=-1)
  return tf.keras.losses.mse(y_data,yhat)

# The GradientDescent in TF1 is now implemented as SGD in TF2
# The API of optimizers has been changed in TF2, the loss 1st argument
# is now callable
optimizer = tf.keras.optimizers.SGD(learning_rate=0.5)

# Minimize the loss function and print the loss
for j in range(100):
  optimizer.minimize(lambda: loss(W,b),var_list=[W, b])
  if j % 10 == 0: # print every 10 iterations
    print("Iteration={} W={} b={} loss={:.9f}".format(
        j, W.numpy(), b.numpy(), loss(W,b).numpy()))

# Print the trained parameters
print("Output: W={} b={}".format(W.numpy(), b.numpy()))
```

CLASSIFICATION WITH TENSORFLOW ESTIMATOR

TensorFlow Estimators are a high-level API for TensorFlow that allows you to create, train, and use deep learning models easily. With TF2, the role of Estimators has been superseded by the Keras API.



The following table summarizes the purposes of the different layers:

Toolkit(s)	**Description**
Estimator (tf.estimator)	High-level, OOP API.
tf.layers/tf.losses/tf.metrics	Libraries for common model components.
TensorFlow	Lower-level APIs

3.
KERAS

We will will use Keras to build an equivalent classifier for the iris data set as a comparison with the earlier TensorFlow example.

```
import tensorflow as tf
# Keras version of Iris classifier
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn import datasets
from sklearn import model_selection
from sklearn import preprocessing
import pandas as pd
import numpy as np
# loading and pre-processing of the data
# We use the 2 class version of iris data set
iris = pd.read_csv("Iris.csv")
x = np.array(iris.drop("Class",axis=1))
y = np.array(iris["Class"])
# Split dataset into train / test
x_train, x_test, y_train, y_test = model_selection.train_test_split(
      x, y, test_size=0.2, random_state=42)
# Scale data (training set) to 0 mean and unit standard deviation.
scaler = preprocessing.StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)
# create model
model = Sequential()
model.add(Dense(10, input_dim=4, activation='relu'))
model.add(Dense(20, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam',
              metrics=['accuracy'])

# training the model
model.fit(x_train, y_train, epochs=10, batch_size=10)
# eval model
scores = model.evaluate(x_test, y_test)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
# calculate predictions
predictions = model.predict(x_test)
# round predictions
rounded = [round(x[0]) for x in predictions]
print(rounded)
```