

# COMP3009 Machine Learning

## Labs – Introduction to *Tensorflow* for Artificial Neural Networks

Dr Xin Chen, Autumn Semester, 2022

### 1. Introduction

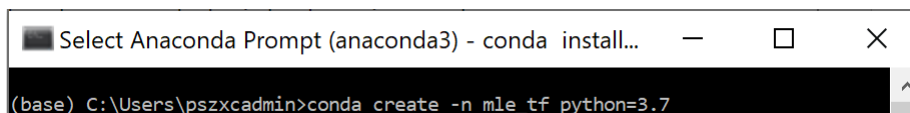
In tutorial 1, we learned how to use *Scikit\_Learn* to perform data pre-processing/visualisation, feature selection, dimensionality reduction, modelling (e.g. linear regression, Support Vector Machines, Decision Trees, etc.) and evaluation. *Scikit\_Learn* also includes classical ANN method (e.g. Multiple Layer Perceptron (MLP) Neural Network). **It is perfectly fine to use *Scikit\_Learn* libraries to complete the coursework.** However, in the era of deep learning, *Tensorflow* (Google) and *Pytorch* (Meta) are much more popular APIs for ANN model implementation, which provide more flexibilities in terms of network structure, loss function design, model training and optimisation.

This lab session aims to help you get familiar with *Tensorflow/Keras* API for implementing, training and evaluating Artificial Neural Networks. This is NOT a tutorial about building deep learning models, but an introduction of using *Tensorflow/Keras* to build a simple MLP neural network. Building deep learning models follow a similar process but more complicated model structures.

### 2. Software Installation

You should have already installed all the required packages in the first lab. If not, follow the steps below again.

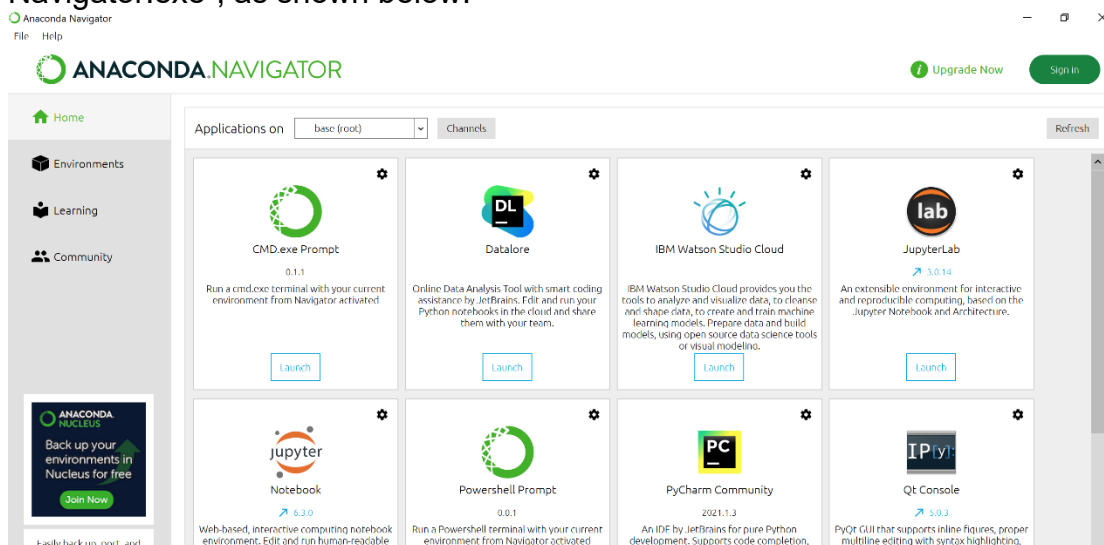
- To download Anaconda, a Python package and environment manager, go to: <https://www.anaconda.com/distribution/>. Download and install the binary file from this page. (You may already have this installed on the lab computers).
- After installing the Anaconda executable, open 'Anaconda Prompt' from the Windows (like the figure below).



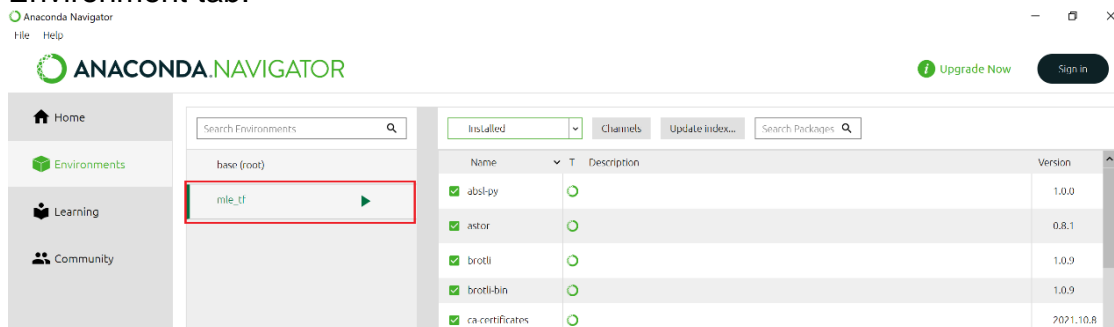
Execute the following commands to install python and other packages.

- To create a new Conda virtual environment called '**mle\_tf**' that uses **Python 3.7**:  
**conda create -n mle\_tf python=3.7**

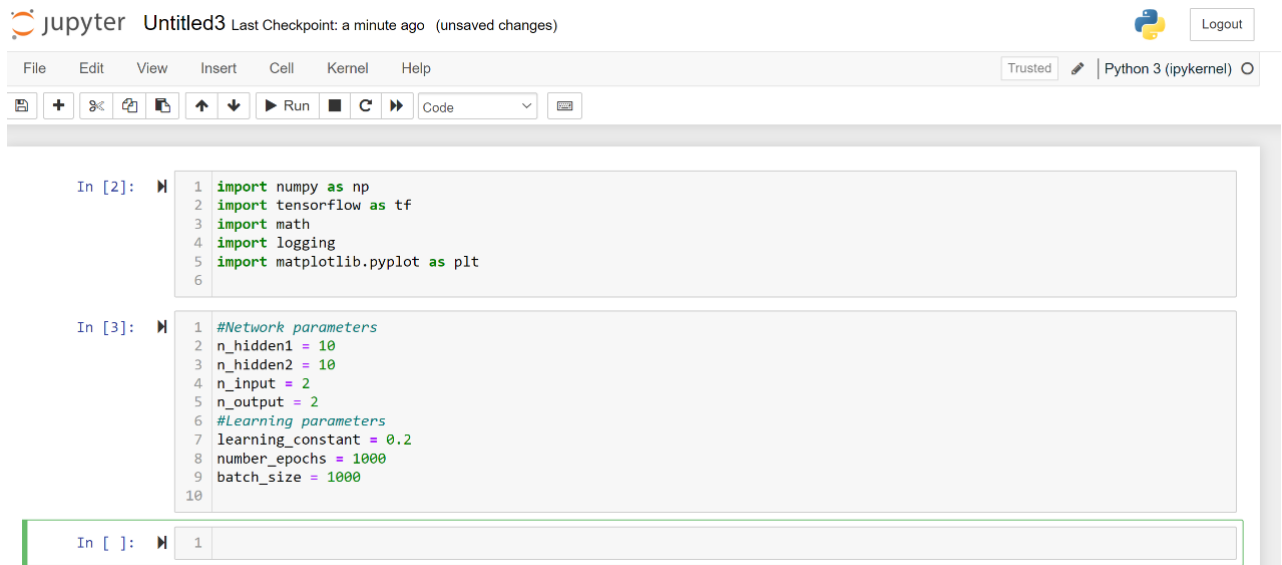
- To enter into the above created virtual environment:  
**conda activate mle\_tf**
- To install Numpy:  
**conda install -c conda-forge numpy**
- To install Tensorflow-cpu (**version 1.14.0**), this version will be automatically selected:  
**conda install -c conda-forge tensorflow**
- To install Matplotlib for data visualization:  
**conda install -c conda-forge matplotlib**
- To install Scipy:  
**conda install -c conda-forge scipy**
- To install Scikit\_Learn:  
**conda install -c conda-forge scikit\_learn**
- Once you installed the above packages you can now start the “Anaconda Navigator.exe”, as shown below:



- Make sure you select the virtual environment (mle\_tf) we just setup, in the Environment tab.



- Then you can start programming using one of the suggested IDE (e.g. PyCharm or Jupyter Notebook). Need to install the IDE in the Home tab and launch it.
- Use Jupyter Notebook as an example, if you type the code below and run it (as the figure below). If no error messages, it means you have correctly setup the required packages.



The screenshot shows a Jupyter Notebook window titled 'Untitled3'. The interface includes a top bar with the Jupyter logo, the title, and a 'Logout' button. Below this is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', and 'Help'. A toolbar contains icons for file operations, cell navigation, and execution. The notebook area displays two code cells. The first cell, labeled 'In [2]:', contains five lines of import statements. The second cell, labeled 'In [3]:', contains ten lines of parameter assignments for a neural network and learning process. The third cell is currently empty and labeled 'In [ ]:'.

```
In [2]: 1 import numpy as np
        2 import tensorflow as tf
        3 import math
        4 import logging
        5 import matplotlib.pyplot as plt
        6

In [3]: 1 #Network parameters
        2 n_hidden1 = 10
        3 n_hidden2 = 10
        4 n_input = 2
        5 n_output = 2
        6 #Learning parameters
        7 learning_constant = 0.2
        8 number_epochs = 1000
        9 batch_size = 1000
        10

In [ ]: 1
```

### 3. Tutorial

This tutorial will guide you to implement a machine learning solution using a real-world binary classification problem (breast cancer prediction) as an example.

- **Dataset:** The Breast Cancer dataset (WDBC) is available in machine learning repository maintained by the University of California, Irvine. The dataset contains 569 samples of malignant and benign tumour cells. The first two columns in the dataset store the unique ID numbers of the samples and the corresponding diagnosis (M=malignant, B=benign), respectively. The columns 3-32 contain 30 real-value features that have been computed from digitized images of the cell nuclei, which can be used to build a model to predict whether a tumour is benign or malignant.

**Please download the “WDBC.csv” file from Moodle and save it to the same directory as your Python file.** Then go through the code below by running them block by block and see the results.

- **Import all required libraries**

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import random
import seaborn as sns
import scipy
```

```

from scipy.stats import pearsonr
import sklearn
from sklearn import datasets, linear_model
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras

```

- **Load CSV file and identify the features and outcome**

```

# load csv file and display some rows
all_df=pd.read_csv('WDBC.csv', index_col=False)
all_df.head()

# ID column is not useful, drop it
all_df.drop('ID', axis=1, inplace=True)
all_df.head()

# Assign features to X
X = all_df.drop('Diagnosis', axis=1)

# Identify the outcome
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
all_df['Diagnosis'] = le.fit_transform(all_df['Diagnosis'])
all_df.head()

# assign numerical label to y
y = all_df['Diagnosis']

```

- **Normalise the features to use zero mean normalisation**

```

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
Xs = scaler.fit_transform(X)

```

- **Perform PCA and retain the first 3 principal components**

```

from sklearn.decomposition import PCA
feature_names = list(X.columns)
pca = PCA(n_components=10)
Xs_pca = pca.fit_transform(Xs)
Xs_pca=Xs_pca[:,0:3] #retain the first 3 PC

```

- **Build a Multiple Layer Perceptron Neural Network**

```

# The model is built using Sequential API in Keras.
# This model contains 3 input neurons, 10 neurons in hidden layer and 1
# output neuron for binary classification
# You may design your own network structure for the task you have

```

```
# The activation function will be different for regression (linear) or multi-class  
# classification (softmax)
```

```
model=keras.models.Sequential()  
model.add(keras.layers.Dense(10, input_dim=3,activation="relu"))  
model.add(keras.layers.Dense(1,activation='sigmoid'))  
model.summary()
```

- **Compile the model**

```
#After a model is created we need to compile the model to specify the loss  
# function and optimiser  
#If you use one-hot encoding for multi-class you need to use  
# 'categorical_crossentropy'  
# If you use class index e.g. from 0 to 3, you can use  
# 'sparse_categorical_crossentropy'
```

```
model.compile(loss="binary_crossentropy", optimizer="sgd",  
metrics=["accuracy"])
```

```
# save the initial weight for initilise new models in cross validation  
model.save_weights('model.h5')
```

- **A quick test of the model using 80% / 20% training and testing split.**

```
# Then we split the data to train and test 80%/20%  
Xs_train, Xs_test, y_train, y_test = train_test_split(Xs_pca, y, test_size=0.2,  
random_state=1, stratify=y)
```

```
# Now we can start the training  
# Tensorflow/Keras uses np array, so need to convert the data format
```

```
#make sure the weights are initialised  
model.load_weights('model.h5')
```

```
# Model learning  
history= model.fit(np.array(Xs_train), np.array(y_train), epochs=50,  
validation_data=(np.array(Xs_test), np.array(y_test)))
```

- **Visualise the training process, loss and accuracy**

```
import pandas as pd  
pd.DataFrame(history.history).plot(figsize=(8, 5))  
plt.grid(True)  
plt.gca().set_ylim(0, 1) # set the vertical range to [0-1]  
plt.show()
```

- **Perform a K-fold cross validation. Observe the training process of each fold. We will also visualise the training process using *Tensorboard* later.**

```

from sklearn.model_selection import KFold
import os

# root file for logging the learning process and can be visualised later in
# tensorboard
root_logdir = os.path.join(os.curdir, "my_logs")

def get_run_logdir():
    import time
    run_id = time.strftime("run_%Y_%m_%d-%H_%M_%S")
    return os.path.join(root_logdir, run_id)
run_logdir = get_run_logdir()

kf = KFold(n_splits=5)
k=1;

for train_index, test_index in kf.split(Xs_pca):

    print("fold",k)

    # initialise the weight for each fold
    model.load_weights('model.h5')

    # Split the data
    X_train, X_test = Xs_pca[train_index], Xs_pca[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # tensorboard for visualising the training process later
    tensorboard_cb = keras.callbacks.TensorBoard(run_logdir)

    # training and validation
    model.fit(np.array(X_train), np.array(y_train), epochs=10,
              validation_data=(np.array(X_test),
                               np.array(y_test)),callbacks=[tensorboard_cb])

    #save the model of each fold
    model.save(os.path.join('fold_{}_model.hdf5'.format(k)))

    # evaluate the accuracy of each fold
    scores = model.evaluate(np.array(X_test), np.array(y_test), verbose=0)
    print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
    k=k+1

```

- We can now use the model to perform predict of new data

```
#load one model to do prediction
model.load_weights('fold_5_model.hdf5')
```

```
# You can use "predict" to predict output in the range of [0 1]
y_pred=model.predict(np.array(X_test))
```

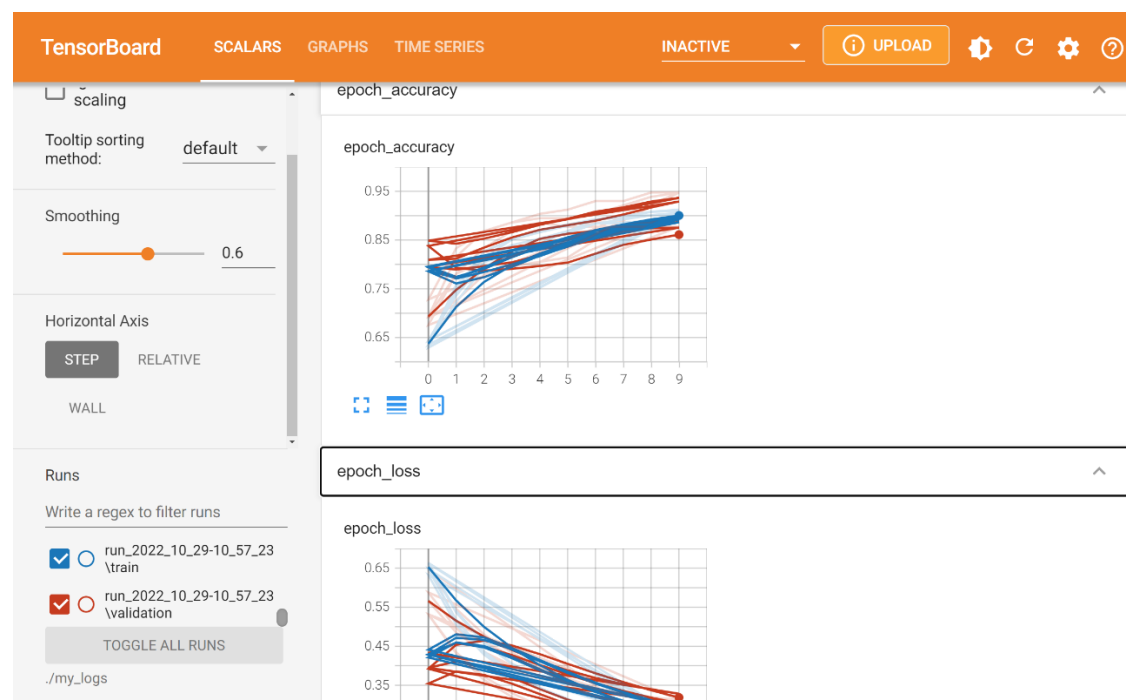
```
# Or use model.evaluate to get the accuracy if the true labels are known
# Here we use the test data of the last fold as an example,
# in practice this should be an independent test set
```

```
loss, acc = model.evaluate(np.array(X_test), np.array(y_test), verbose=2)
print("Restored model, accuracy: {:.5f}%".format(100 * acc))
```

- Visualise the training process using *Tensorboard*

We can also visualise the training process in *Tensorboard*. If you installed *TensorFlow* within a virtual environment (Anaconda), you should activate it in the prompt window: **conda activate [env name]**. Then type "**tensorboard --logdir=./my\_logs --port=6006**". Finally, open up a web browser to <http://localhost:6006>

In our case, the loss and accuracy of all the 5 folds are stored. You may play with the *Tensorboard* and explore the functions.



## 4. Optional Tasks

- Use other evaluation metrics to do some analysis. E.g. precision, recall, F1, ROC, etc.
- Tune the hyperparameters in MLP to achieve a better performance.
- Check how to use "keras.callbacks.ModelCheckpoint" to save the best performed model using validation set.
- Use "keras.callbacks.EarlyStopping" to find do early stopping for avoiding overfitting