COMP3009/ COMP4139 Machine Learning

Lab 6 - Tensorflow for Artificial Neural Networks

Dr Xin Chen, Autumn Semester, 2024

1. Introduction

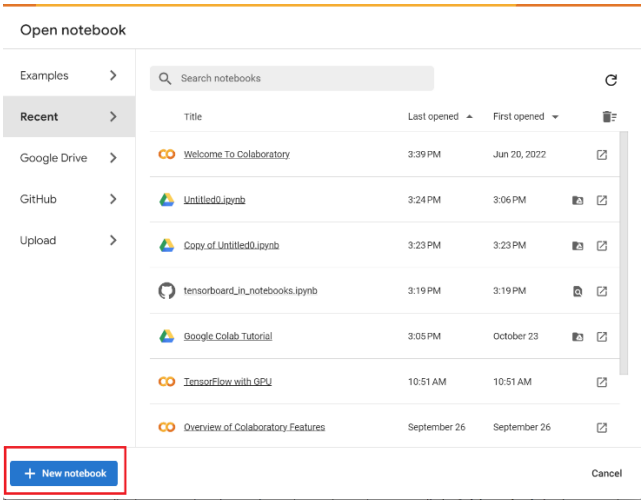
In all previous lab sessions, we learned how to use *Scikit_Learn* to perform data pre-processing/visualisation, feature selection, dimensionality reduction, modelling (e.g. linear regression, Support Vector Machines, Decision Trees, etc.) and evaluation. *Scikit_Learn* also includes the classical ANN methods (e.g. Multiple Layer Perceptron (MLP) Neural Network). It is perfectly fine to use *Scikit_Learn* libraries to complete assignment 2 without using deep learning packages. However, in the era of deep learning, *Tensorflow* (by Google) and *Pytorch* (by 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 become 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 can be learned from many online official tutorials suggested at the end of this document.

Training deep learning models often requires the use of Graphical Processing Units (GPU). Many companies provide online GPU service, e.g. Google and Amazon. In this lab session we will use Google Colab as the platform to run our code. It has a free service version that provides minimal GPU support which is more than enough for this tutorial and some basic deep learning models.

2. Google Colab

To get started, please have a look at Colab's official tutorial. The actual programming environment is here: https://colab.research.google.com/?utm_source=scs-index. Colab uses Google Drive to save and load data, hence please sign up for a Google account first. Then you can create a new notebook using the "New notebook" button in the bottom left as shown below. All necessary machine learning and deep learning (i.e. Tensorflow) packages should be already pre-installed in this environment.



3. Tutorial

This tutorial uses the same dataset (WDBC) that we used previously.

- Upload the "WDBC.csv" file to the **"content/sample_data/"** folder in the Colab environment. Then go through the code below by running them block by block and see the results. Use "+code" to add new block of code, similar to Jupyter Notebook.

- **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('/content/sample_data/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 using zero mean normalisation. Why do we need feature normalisation in ANN?**

```
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. Try to build different MLP structures.**

```
# 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.weights.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.weights.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*. Search for the meaning and usage of each function that you don't know.**

```
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, "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;
```

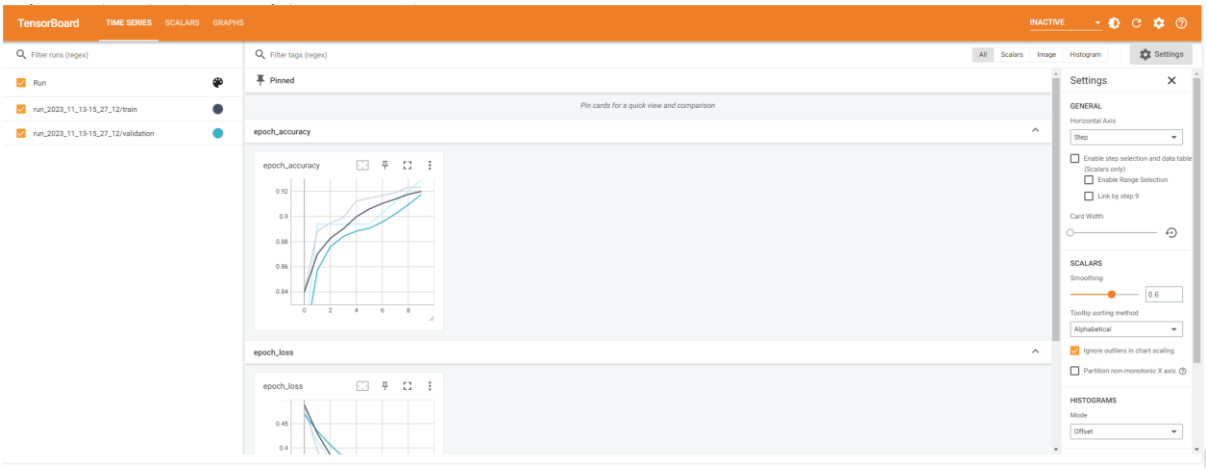
```
for train_index, test_index in kf.split(Xs_pca):

    print("fold",k)

    # initialise the weight for each fold
    model.load_weights('model.weights.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 prediction 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: {:5.2f}%".format(100 * acc))
```

- **Visualise the training process using *Tensorboard*. Play with Tensorboard by changing different numbers of epochs.**

```
# Load the TensorBoard notebook extension
%load_ext tensorboard

%tensorboard --logdir logs
```



4. Additional 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 how to perform early stopping for avoiding overfitting.
- Check how to enable GPU in Colab.