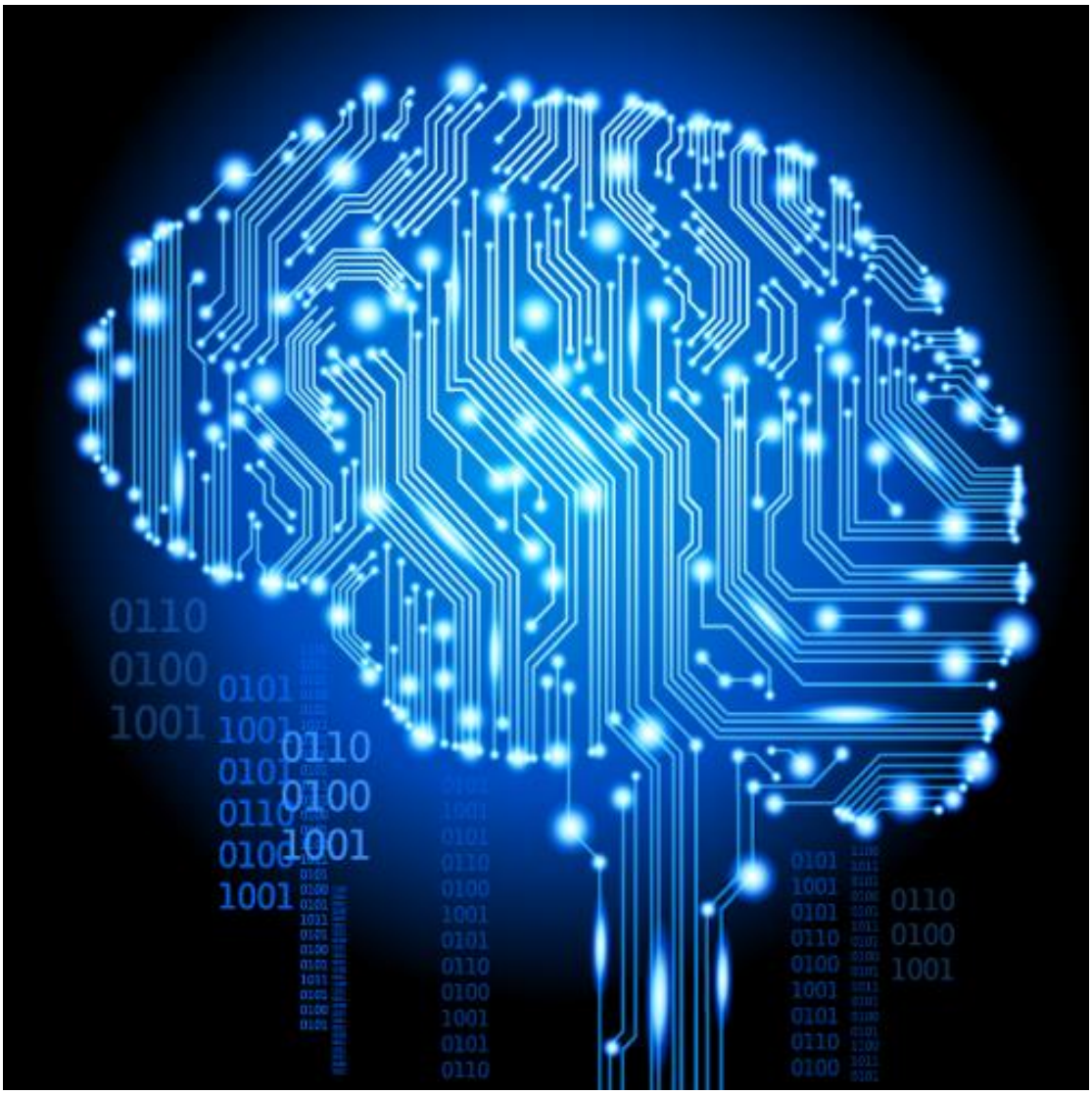
COMP3009 Machine Learning

Computer Based Coursework Manual – Autumn 2021

Assessed coursework 3: Artificial Neural Networks



Assignment 3: Artificial Neural Networks

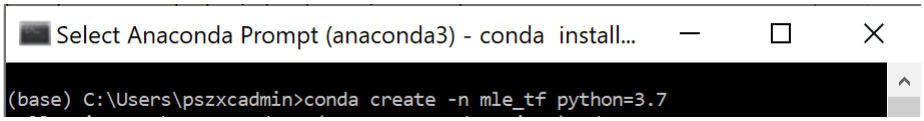
Introduction

This is the third assessed lab on the topic of Artificial Neural Networks. As a group, you will implement it using TensorFlow in Python programming language. You are required to implement, training and testing of ANN models for both regression and classification problems. You will need to use the same datasets that you used for the Decision Trees assignment. The submission deadline is **9th December, 2021 at 4pm**.

Instructions to install Python and Tensorflow

- 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.

- 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

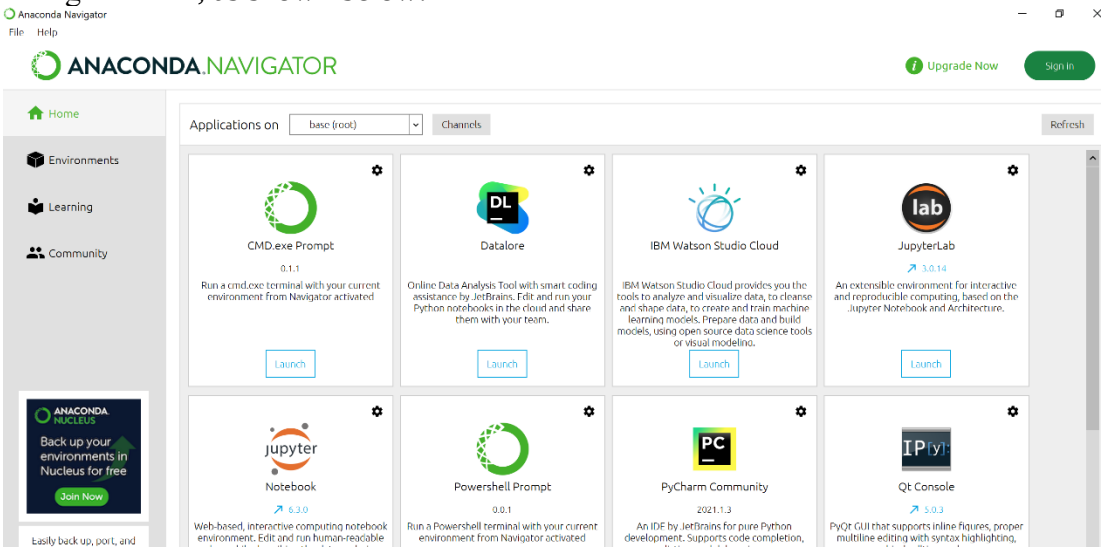
- To install Numpy:
  conda install -c conda-forge numpy

- To install Tensforlow-cpu (**version 1.14.0**), this version will be automatically selected:
  conda install -c conda-forge tensorflow

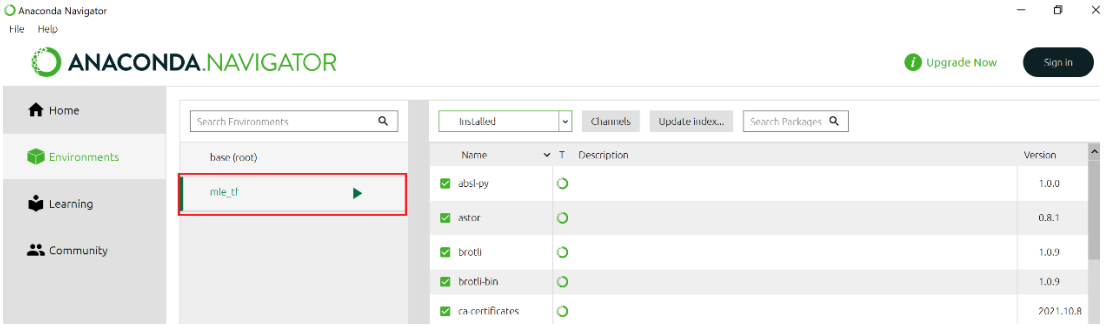
- To install Matplotlib for data visualization:
  conda install -c conda-forge matplotlib

- To install Scipy:
  conda install -c conda-forge scipy

- Once you installed the above packages you can now start the "Anaconda Navigator.exe", as shown below:

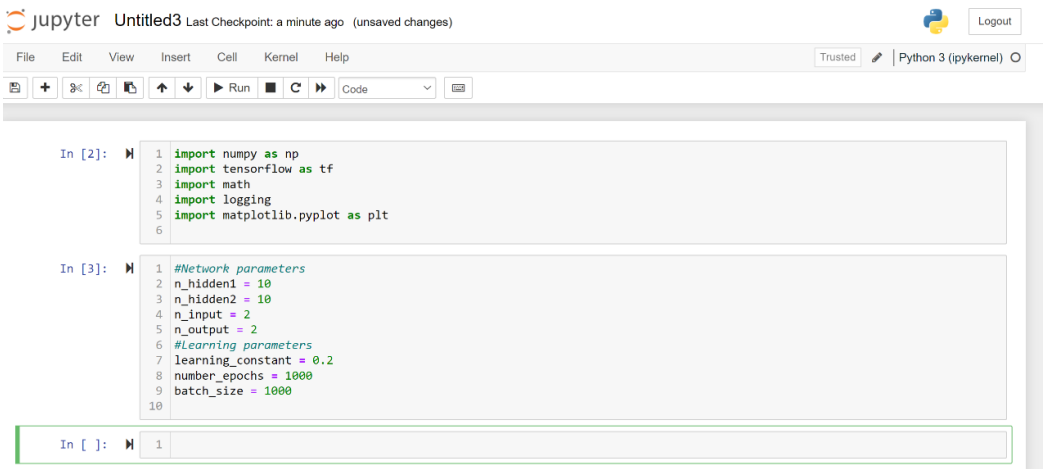


- Make sure you select the virtual environment (mle_tf) we just setup, in the Environment tab.



- 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.



Artificial Neural Networks

An Artificial Neural Network (ANN) or simply Neural Network is a network of interconnected neurons (see Fig. 1 for an example on the domain of facial expression analysis) that is able to learn non-linear real-valued, discrete-valued or vector-valued functions from examples. They are inspired by the observation that biological learning systems are composed of very complex webs of neurons. Every neuron has a number of real-valued inputs, which can be the outputs of other neurons, and uses a mathematical function called the transfer function to compute one real-valued output.

The way the various neurons are connected is called the network topology. The neurons are interconnected with each other by synapses, the weights of which define the output of the neuron. In the training phase, it is these weights that need to be learned. Together with the topology and the type of neurons the weights completely define the ANN.

The procedure used to update the weights in order to minimize the mistakes made by the classifier is called the training rule, or learning law. In the course lectures the *backpropagation* method is described.

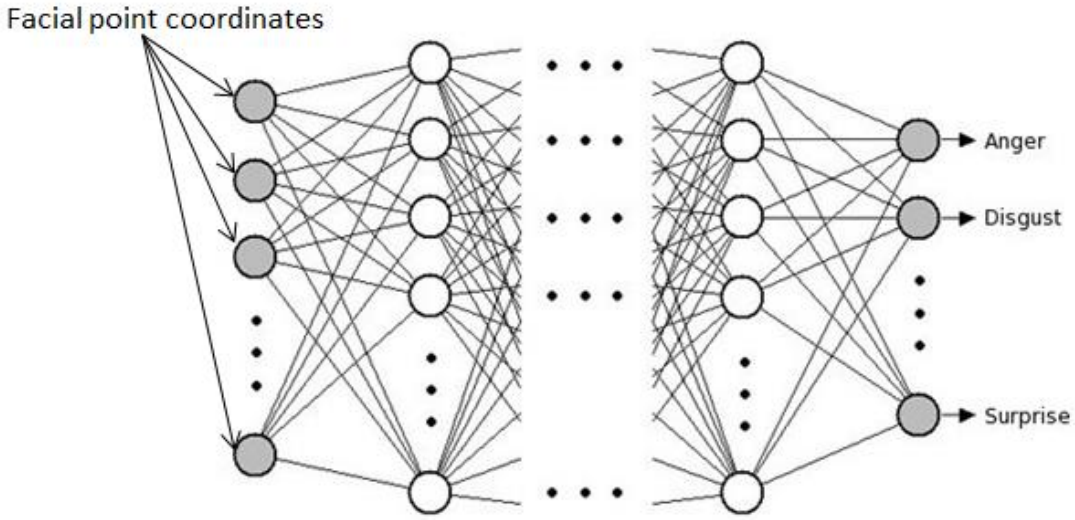


Figure 1.Artificial Neural Network with facial point coordinates as inputs and emotions as outputs

ANN in TF implementation example

The neural network structure must be implemented and trained using the TensorFlow (TF) package available for Python programming language. **Please use Python 3.7 and TF version 1.14 for this coursework (don't need to use the latest version for this coursework).** For a basic introduction about what is TensorFlow and how parallel processing in this structure works you may refer to https://www.tensorflow.org/. and the provided document named "TensorFlow Introduction".

As an example, a multi-layer perceptron is implemented under this package for a simple pattern recognition application. **Note that some of the functions used below might be deprecated due to version change. Please either use alternative functions or Google for a solution.** This artificial neural network which is considered in this case consists of two hidden layer and an output layer with two outputs representing the classes corresponding to the dataset. The number of samples considered are 1000 with 600 being chosen for training and the rest for

testing data. The activation functions considered are sigmoid function for the hidden layer and a linear layer for the output layer. The code is as follows.
The following few lines define the packages

```
import numpy as np
import tensorflow as tf
import math
import logging
logging.basicConfig(level=logging.DEBUG)
import matplotlib.pyplot as plt
```

The parameters used for the neural network are defined as follows.

```
#Network parameters
n_hidden1 = 10
n_hidden2 = 10
n_input = 2
n_output = 2
#Learning parameters
learning_constant = 0.2
number_epochs = 1000
batch_size = 1000
```

The weights, biases and input output settings of the neural network are defined as follows.

```
#Defining the input and the output
X = tf.placeholder("float", [None, n_input])
Y = tf.placeholder("float", [None, n_output])

#DEFINING WEIGHTS AND BIASES

#Biases first hidden layer
b1 = tf.Variable(tf.random_normal([n_hidden1]))
#Biases second hidden layer
b2 = tf.Variable(tf.random_normal([n_hidden2]))
#Biases output layer
b3 = tf.Variable(tf.random_normal([n_output]))

#Weights connecting input layer with first hidden layer
w1 = tf.Variable(tf.random_normal([n_input, n_hidden1]))
#Weights connecting first hidden layer with second hidden layer
w2 = tf.Variable(tf.random_normal([n_hidden1, n_hidden2]))
#Weights connecting second hidden layer with output layer
w3 = tf.Variable(tf.random_normal([n_hidden2, n_output]))
```

The following function defines the neural network structure and calculates its output when its inputs are given.

```
def multilayer_perceptron(input_d):
    #Task of neurons of first hidden layer
    layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(input_d, w1), b1))
    #Task of neurons of second hidden layer
    layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1, w2), b2))
    #Task of neurons of output layer
    out_layer = tf.add(tf.matmul(layer_2, w3),b3)

    return out_layer
```

The neural network is defined as follows.

```
#Create model
neural_network = multilayer_perceptron(X)
```

The loss function and the optimization algorithm are defined as follows.

```
#Define loss and optimizer
loss_op = tf.reduce_mean(tf.math.squared_difference(neural_network,Y))

#loss_op =
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=neural_netw
ork,labels=Y))
optimizer =
tf.train.GradientDescentOptimizer(learning_constant).minimize(loss_op)
```

The few next lines initialize the variables.

```
#Initializing the variables
init = tf.global_variables_initializer()
```

The batch input and output data are defined as follows.

```
batch_x1=np.loadtxt('x1.txt')
batch_x2=np.loadtxt('x2.txt')
batch_y1=np.loadtxt('y1.txt')
batch_y2=np.loadtxt('y2.txt')

label=batch_y2#+1e-50-1e-50

batch_x=np.column_stack((batch_x1, batch_x2))
batch_y=np.column_stack((batch_y1, batch_y2))

batch_x_train=batch_x[:,0:599]
batch_y_train=batch_y[:,0:599]

batch_x_test=batch_x[:,600:1000]
batch_y_test=batch_y[:,600:1000]

label_train=label[0:599]

label_test=label[600:1000]
```

You have to replace the filenames of the data to load to suit your data set. The following few lines prepare a TensorFlow session and perform the optimization. Finally, the results of `print(accuracy1.eval({X: batch_x}))` is the accuracy of the classification.

```
with tf.Session() as sess:
    sess.run(init)
    #Training epoch
    for epoch in range(number_epochs):

        sess.run(optimizer, feed_dict={X: batch_x_train, Y:
batch_y_train})
        #Display the epoch
        if epoch % 100 == 0:
            print("Epoch:", '%d' % (epoch))

    # Test model
```

```
    pred = (neural_network)  # Apply softmax to logits
    accuracy=tf.keras.losses.MSE(pred,Y)
    print("Accuracy:", accuracy.eval({X: batch_x_train, Y:
batch_y_train}))
    #tf.keras.evaluate(pred,batch_x)

    print("Prediction:", pred.eval({X: batch_x_train}))
    output=neural_network.eval({X: batch_x_train})
    plt.plot(batch_y_train[0:10], 'ro', output[0:10], 'bo')
    plt.ylabel('some numbers')
    plt.show()

    estimated_class=tf.argmax(pred, 1)#+1e-50-1e-50
    correct_prediction1 =  tf.equal(tf.argmax(pred, 1),label)
    accuracy1 = tf.reduce_mean(tf.cast(correct_prediction1, tf.float32))

    print(accuracy1.eval({X: batch_x}))
```

Create/load data

It is required to load data in Python to process them. For the Neural Networks implementation, the features should be arranged such that every row represents one example, and every column one attribute (feature).

You will have to write a data-loader for your two datasets. This will take some effort, so please assign some of your team to work on one dataset and some others to work on the second dataset.

Split your data into a training, validation, and test set.

Include the data loading code in your submission.

Create network using the sample file

Use the Python code from the example above to initialise your own neural network structure. Some modifications can be made to achieve a network structure that you expect to work better for your data. For instance, the number of the layers can be changed by adding another weight, bias and activation function which can act on the output of the second layer. The number of neurons can be modified using `n_hidden1, n_hidden2.` The number of outputs and inputs for the neural network can be changed using `n_output` and `n_input`, respectively. The parameters `learning_constant` and `number_epochs` can affect the learning process as well and can be modified to improve the performance. There exist other methods to evaluate the error of neural network for instance consider:

- `tf.keras.losses.MeanAbsoluteError`
- `tf.keras.losses.MeanAbsolutePercentageError`
- `tf.keras.losses.MeanSquaredLogarithmicError`

Have a go at experimenting with some of the training parameters, and report on how this changes the performance on your test set.

In particular, include in your report what are the influences of the learning rate and the number of epochs used? You can report on this using a single train, validation, and test split.

Evaluation

Now that you have a basic understanding of Neural Networks train a network using the two datasets from the previous assignment. You can use line:

```
output=neural_network.eval({X: batch_x_train})
```

to evaluate the performance of the neural network. Next, evaluate the neural networks using 10-fold cross validation. Make a 10-fold cross-validation evaluation of the neural networks using the parameters and learning rule that is optimal according to your previous experiments. Evaluate the two problems (regression and classification) using appropriate evaluation metrics.

Deliverables

For the completion of this assessed pair of labs, the following has to be handed in:

1. Python material, containing:
   - The resulting Artificial Neural Networks (one for each of the two ML problems), created by training on the whole dataset. Save the code which includes the insert data evaluation process and the figure comparing the obtained results and the expected ones.

2. Report of up to 1,000 words containing:
   - Explanation of the parameters chosen (e.g. topology, learning rule, learning rate, nr. of epochs, sizes,) and reports on the difficulties encountered.
   - Classification results per cross-validation fold for classification, and Root Means Square Error for the regression problem.
   - Average cross-validation accuracy results, that include:
     - Average accuracy for the classification problems.
     - Average Root Mean Square Error for the regression problem
   - Explain what action(s) you took to ensure generalisation of the network and overcome the problem of overfitting.
   - Discussion about the pros and cons of SVM, decision trees and ANN based on your results.

Marking Criteria

Clarity of report: 10%
Code quality: 10%
Implementation correctness: 20%
Parameter setting: 15%
Method evaluation: 15%
Explanation of action taken to avoid model overfitting: 10%
Discussion about SVM, decision trees and ANN: 20%