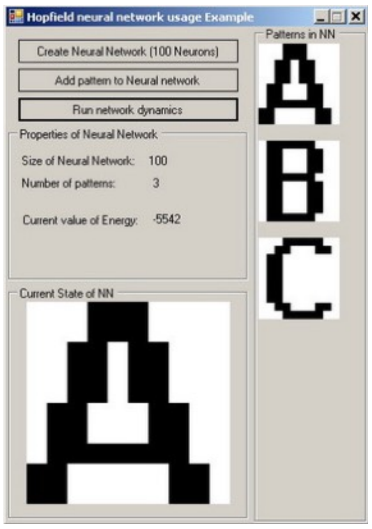
EE4305 Project AY16/17 S1

Literature Survey on Neural Networks

Neural networks are statistical-based learning models that are modelled after biological neural networks and can be used to predict output values that depend on a huge number of inputs. Essentially, they are a set of interconnected hidden layers of neurons between the input and output layer. Each neuron within a layer is connected to those in another layer via weighted edges. The weights of each edge are learned by training the model with the training data.

Over the years, neural networks have been used widely in many types of applications. For example, they have been successfully used in automatic document classification, pattern recognitions, stock markets predictions, medical diagnosis and even many complicated computer visions applications such as objects identification and optical character recognition shown below.


Hopfield neural network usage Example
Create Neural Network (100 Neurons)
Add pattern to Neural network
Run network dynamics
Properties of Neural Network
Current State of NN
Patterns in NN

Size of Neural Network: 100
Number of patterns: 3
Current value of Energy: -5542

Description of Assignment and Solution

There are usually two types of tasks that neural networks can solve namely classification and regression problems. For this assignment, we are given both a classification problem involving HAPT features and a regression problem involving Student Performance.

First of all, classification is a problem whereby we are interested to find out which category a new row of test data belong to. It is essentially a supervised learning method as we train the model by using a training dataset that has already been correctly tagged. After the model has been successfully trained, we can then apply the model to the new test dataset to predict classifications for the new input data.

On the other hand, regression is a problem where we try to approximate the relationships among variables. In essence, regression helps us to discover how a dependent variable will change when the independent variables are varied one by one.

For this assignment, I have chosen to implement my solutions for both problems using Python 2.7 and some additional Python packages related to Neural Networks. These packages are listed below:

1. **Keras** **-** A high-level neural network and deep-learning library that supports fast creation and training of of a NN model.

2. **TensorFlow** - An open-source library developed by Google for numerical computation of data flow graphs. Keras runs on top of TensorFlow (OSX). Alternatively, Theano can be used for Windows users.

3. **Pandas** - An open-source Python library providing data structures and data manipulation functions which I used to parse and manipulate the given .txt and .xls input files in the dataset.

4. **Scikit-learn** - A Python library for Machine Learning. It is used internally by Keras and requires NumPy, SciPy and Matplotlib.

Most of these dependencies can be easily installed via the pip install tool for Python.

Problem One Description - HAPT

The first dataset given contains multiple files with information about Human Activity and Posture Transition (HAPT) features. The main dataset haptAttr.txt is a text file with 8000 samples of 561 attributes. The objective for this problem was to design a neural network classifier that can associate features to one of 12 target classes, making this a multi-class classification problem.

Classification Algorithm

To solve this multi-class classification problem, I wrote a Python script named classify.py that uses Keras, Pandas and sklearn to create the neural network model and perform prediction. The classification algorithm can be summarised as follows:

1. Read in the input data from haptAttr.txt containing 8000 rows with 561 inputs columns using pandas into the input variable.

2. Read in the output data from haptLabel.txt containing one output column using pandas into the output variable.

3. Encode the output values using one hot encoding via Keras's `np_utils.to_categorical( )` method. This is done to create a vector of 1x12 (since there are 12 output categories) to uniquely distinguish each output category from the other categories and can help computation to be faster.

4. Split the data set of 8000 rows randomly into 70% training data and 30% test data using sklearn's `train_test_split( )` function. The rationale for selection of these sizes are described in a later section.

5. Build and train a baseline neural network model using 70% of the given dataset (train data) using Keras's `KerasClassifier`. In my solution, I created a neural network model with 2 hidden layers. The structure of the network model looks like: *561 inputs -> [4 hidden nodes] -> [50 hidden nodes] -> 12 outputs*

6. Use the remaining 30% of the given dataset (test data) to predict the outcome.

Selection of Inputs and Outputs of the Multi-Layer Perceptron (MLP)

The structure of my network model looks like:

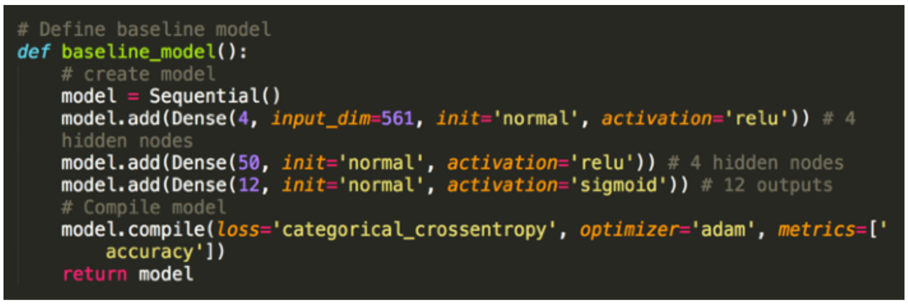
561 input nodes -> [4 hidden nodes] -> [50 hidden nodes] -> 12 output nodes

The input layer has 561 nodes since there are 561 input attributes. Similarly, the output layer has 12 nodes since there are 12 possible output categories.

Training of MLP

The exact number of hidden layers and the type of activation function in each hidden layer is problem-dependent and generally found through trial and error. Generally speaking, we need a neural network that can capture the size and structure of the problem.

Through trial and error, I have decided to use 2 *fully-connected* hidden layers with 4 and 50 nodes respectively, which I defined using the Dense class in Keras. The type of activation function in the hidden layers is the rectifier function while the type of activation function on the output layer is sigmoid to make sure that the output of the network output is between 0 and 1 and may be used as predicted probabilities for the output categories since the output layer will create 12 output values, one for each class. The output category with the biggest value will be the predicted class by the model.



```
# Define baseline model
def baseline_model():
    # create model
    model = Sequential()
    model.add(Dense(4, input_dim=561, init='normal', activation='relu')) # 4
    hidden nodes
    model.add(Dense(50, init='normal', activation='relu')) # 4 hidden nodes
    model.add(Dense(12, init='normal', activation='sigmoid')) # 12 outputs
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['
        accuracy'])
    return model
```

Selection of training data size

There are 8000 rows of HAPT data given that has to be split into the train data and test data. In order to select the best training data size, I varied the training data size that I fed into the model. The following table shows the results:

Training Data Size (%)	Testing Data Size (%)	Correctly Predicted
80	20	0.957916666667
70	**30**	**0.964166666667**
60	40	0.9178125
50	50	0.937
40	60	0.930208333333
30	70	0.918928571429

Overall, the training data size of 70% and testing data size of 30% gave the best accuracy. Hence, the 8000 rows are split into 70% training data and 30% test data randomly using sklearn's `train_test_split( )` function to train the model and be used for prediction.

Performance of MLP

After training the network with 70% training data and using it to predict the remaining 30% data, the following results were achieved: loss: 0.0597 - acc: 0.9750. Furthermore, out of the 30% tested data (randomly chosen 2400 rows out of the 8000 rows), the model was able to correctly classify approximately 95.54% of the test data into the correct class.

Problem Two Description

The second problem that we are given is a regression problem. The dataset describes 30 input data attributes including students grades, demographic, social and other school related attributes. We are concerned with modelling student performance in secondary education to predict the students' performance. Hence, this is a regression predictive modelling problem.

In the dataset, there are a total of 350 samples with 30 inputs attributes and 3 grades. Input and output attributes are not all numerical as the attributes for students' data contains both strings and integers. Hence, several columns of the dataset has to been converted into numerical format for this regression problem.

The first part of this problem involves using the first 30 attributes to build a regression model to predict G3. The second part involves using all the 32 attributes (with the addition of G1 and G2) to predict G3. According to the Answer 5 in the given FAQ, it is possible to attain a Mean Squared Error (MSE) lesser than 5. This is a nice target that I aimed for using my neural network model.

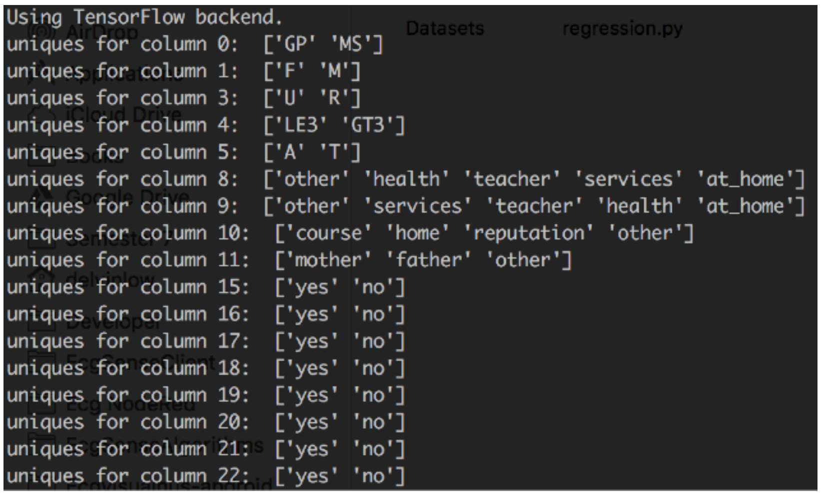
Regression Algorithm

To solve this regression problem, I wrote a Python script named regression.py that uses Keras, Pandas and sklearn to create the neural network model and estimate regression. The classification algorithm can be summarised as follows:

1. Read in the input data from students.csv containing 350 rows with 30/32 inputs columns using pandas into the input variable.

2. Read in the output data G3 from students.csv containing one output column in column 33 using pandas into the output variable.

3. Maps categorical data to numerical form using pandas factorize() function. The mapping is shown below. The index of the category in the list is the unique value it is mapped to.



```
Using TensorFlow backend.
uniques for column 0:  ['GP' 'MS']
uniques for column 1:  ['F' 'M']
uniques for column 3:  ['U' 'R']
uniques for column 4:  ['LE3' 'GT3']
uniques for column 5:  ['A' 'T']
uniques for column 8:  ['other' 'health' 'teacher' 'services' 'at_home']
uniques for column 9:  ['other' 'services' 'teacher' 'health' 'at_home']
uniques for column 10:  ['course' 'home' 'reputation' 'other']
uniques for column 11:  ['mother' 'father' 'other']
uniques for column 15:  ['yes' 'no']
uniques for column 16:  ['yes' 'no']
uniques for column 17:  ['yes' 'no']
uniques for column 18:  ['yes' 'no']
uniques for column 19:  ['yes' 'no']
uniques for column 20:  ['yes' 'no']
uniques for column 21:  ['yes' 'no']
uniques for column 22:  ['yes' 'no']
```

4. Use the data set of 350 rows to build and train a baseline neural network model as a regression estimator using Keras's `KerasRegressor`. In my solution, I created a neural network model with 2 hidden layers. The structure of the network model looks like: *30/32 inputs -> [10 hidden nodes] -> [10 hidden nodes] -> 1 output*

5. Running the script shows an estimate of the model's performance on the problem for unseen data on students' performance. The final result indicates the MSE including the mean and standard deviation through the ten folds of the cross validation evaluation.

Selection of Inputs and Outputs of the Multi-Layer Perceptron (MLP)

The structure of my regression model looks like:

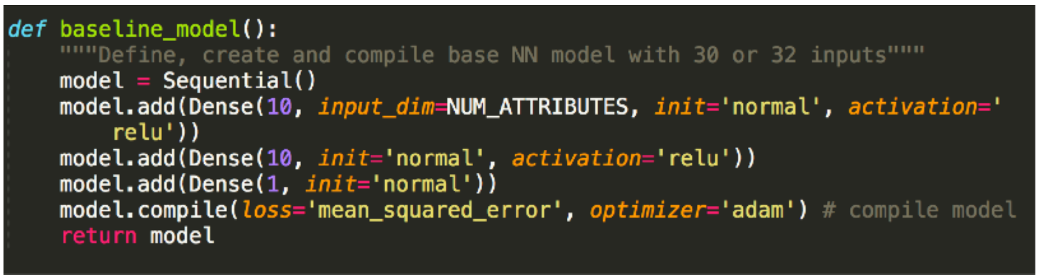
30/32 input nodes -> [10 hidden nodes] -> [10 hidden nodes] -> 1 output node

The input layer has 30/32 nodes since there are two parts to the problems which can either use only the first 30 input attributes or 32 input attributes that include G1 and G2 scores. Similarly, the output layer has 1 node since this is a regression problem.

Training of MLP

The exact number of hidden layers and the activation function of each hidden layer is problem-dependent and generally found through trial and error. Generally speaking, we need a neural network that can capture the size and structure of the problem. Through trial and error, I have decided to use 2 *fully-connected* hidden layers with 10 nodes each, which I defined using the Dense class in Keras.

The final activation functions of both layers are relu (rectifier activation function) but others like sigmoid and tanh had also been tried and work too. Since this is a regression problem, I did not use any activation function for the output layer in order to directly predict numerical values without any transformation.



```
def baseline_model():
    """Define, create and compile base NN model with 30 or 32 inputs"""
    model = Sequential()
    model.add(Dense(10, input_dim=NUM_ATTRIBUTES, init='normal', activation='
        relu'))
    model.add(Dense(10, init='normal', activation='relu'))
    model.add(Dense(1, init='normal'))
    model.compile(loss='mean_squared_error', optimizer='adam') # compile model
    return model
```

When compiling the model, I chose to use the ADAM optimizer algorithm in Keras to optimise the mean squared error loss function. This MSE is also used to assess the performance of the metric in the section below.

Performance of MLP

There are two parts to this problem. For part 1, I only used the first 30 attributes to build a regression model to predict G3. For part 2, I used all the 32 attributes (including ‘G1’ and ‘G2’) to predict G3.

The prediction of G3 using only the first 30 input attributes gave an average Mean Squared Error of 10.07 with a standard deviation of 2.56. On the other hand, the prediction of G3 using all 32 input attributes including G1 and G2 gave an average Mean Squared Error of 3.76 with a standard deviation of 1.82. This agrees with the Answer 5 in the given FAQ that it is possible to attain a Mean Squared Error (MSE) lesser than 5.

This difference in performance between these 2 parts is expected since the G1 and G2 are also student scores like G3. Hence, G1 and G2 are likely more related to G3 than all of the other attributes and when G1 and G2 used in the regression model to predict G3 (student performance), the neural network model is able to perform better and give a much smaller MSE.

Conclusion

In conclusion, the neural networks created have been able to successfully solve both the classification problem for the HAPT features and regression problem for the Student Performance pretty accurately. Overall, neural networks certainly seems like an useful and very interesting type of tool that us Engineers and Programmers can add to our arsenal of knowledge.

However, while neural networks have a proven record to solve certain problems like these two problems in the assignment and several others mentioned in the literature review, I believe they might not be suitable for every single problem given that they are too much of a black box. Basically, this makes neural networks difficult to train, tune and to understand exactly how they are solving a problem. Hence, we should also take precaution when choosing to use neural networks and also explore other methods too when dealing with related problems.