NNFL (BITS F312) Assignment 3

2016B5A30572H
K Pranath Reddy

Question 1

The dataset in ‘data_for_cnn.mat’ consists of 1000 ECG signals and each row corresponds to one ECG signal. The class label for each ECG signal is given in ‘class_label. mat’ file. Implement the 1D convolutional neural network with BPCNN as the learning algorithm for the evaluation of optimal weight matrices in FC layers and optimal kernels or filters in convolution layer. The network consists of one convolutional layer, one pooling layer and two fully connected (FC) layers. The network flow is given by

Input-Convolution Layer-Pooling layer- Convolution Layer-Pooling layer -FC1-FC2-FC3-Output

Consider the square loss function as cost function in the output layer. You can select number of hidden neurons by your own choice. In the pooling layer, you can use average pooling with down- sampling factor as 2. (For implementation of the BPCNN algorithm, please refer to the class notes or slides).

(For MATLAB, you can use the inbuilt functions from deep learning toolbox) (You can use Python with Keras, and tensorflow at the backend.)

Solution :

Code :

```
'''
*** CNN ***
Binary Classification

Author :
Pranath Reddy
2016B5A30572H
'''

from mat4py import loadmat
import numpy as np
import pandas as pd
from keras import Sequential
from keras import optimizers
```

```python
from keras.layers import Dense,Conv1D,AveragePooling1D,Flatten
from sklearn.model_selection import train_test_split
from sklearn import preprocessing

x = loadmat('data_for_cnn.mat')
x = pd.DataFrame(x)
x = np.asarray(x)

x_temp = []
for i in range(len(x)):
   x_temp.append(x[i][0])
x_temp = np.asarray(x_temp)
x = x_temp
x = preprocessing.normalize(x)
x = x.reshape(x.shape[0],x.shape[1],1)

y = loadmat('class_label.mat')
y = pd.DataFrame(y)
y = np.asarray(y)

y_temp = []
for i in range(len(y)):
   y_temp.append(y[i][0][0])
y_temp = np.asarray(y_temp)
y = y_temp

x_tr, x_ts, y_tr, y_ts = train_test_split(x, y, test_size=0.3)
x_tr = x_tr.reshape(700,1000,1)
x_ts = x_ts.reshape(300,1000,1)

# Hyperparameters
learning_rate = 0.01
epochs = 1000

# Input-Convolution Layer-Pooling layer- Convolution Layer-Pooling layer
-FC1-FC2-FC3-Output
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=(5), input_shape=(1000,1), strides=2,
padding='valid', activation='relu'))
```

```
model.add(AveragePooling1D(pool_size=2,strides=2,padding='same'))
model.add(Conv1D(filters=32, kernel_size=(5), strides=2, padding='valid',
activation='relu'))
model.add(AveragePooling1D(pool_size=2, strides=2, padding='same'))
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dense(80, activation='relu'))
model.add(Dense(1,activation='linear'))

model.compile(optimizer='adam',loss='mean_squared_error',metrics=['accuracy'])
model.summary()
model.fit(x_tr,y_tr, epochs=epochs)

# testing the model
yp = model.predict_classes(x_ts)

y_temp3 = []
for i in range(len(yp)):
   y_temp3.append(yp[i][0])
y_temp3 = np.asarray(y_temp3)
yp = y_temp3

y_actual = pd.Series(y_ts, name='Actual')
y_pred = pd.Series(yp, name='Predicted')
confmat = pd.crosstab(y_actual, y_pred)

print(confmat)
confmat = np.asarray(confmat)
tp = confmat[1][1]
tn = confmat[0][0]
fp = confmat[0][1]
fn = confmat[1][0]
Acc = float(tp+tn)/float(tp+tn+fp+fn)
SE = float(tp)/float(tp+fn)
SP = float(tn)/float(tn+fp)
print('Accuracy : ' + str(Acc))
print('sensitivity : ' + str(SE))
print('specificity : ' + str(SP))
```

Q1 Results :

```
Predicted     0     1
Actual
0           145    14
1            12   129
Accuracy : 0.9133333333333333
sensitivity : 0.9148936170212766
specificity : 0.9119496855345912
```

Question 2

Implement the 1D convolutional autoencoder for the dataset given in data_for_cnn.mat file. The architecture is given as

(input-convolution layer-pooling layer-FC-upsampling layer-transpose convolution layer)

(For MATLAB, you can use the inbuilt functions from deep learning toolbox) (You can use Python with Keras, and tensorflow at the backend.)

Solution :

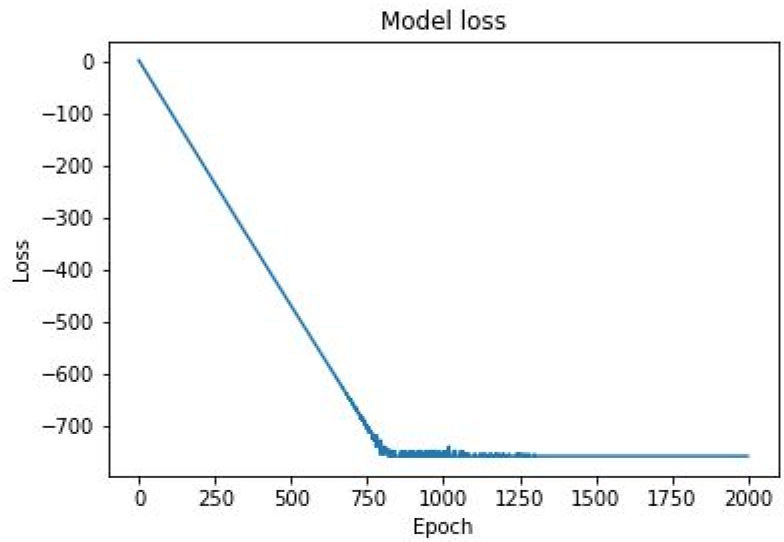
Code :

```
'''
*** 1D Convolutional Autoencoder  ***

Author :
Pranath Reddy
2016B5A30572H
'''

from mat4py import loadmat
import numpy as np
import pandas as pd
from keras import Sequential
from keras import optimizers
from keras.models import Model
from keras.layers import Input,Dense,Conv1D,MaxPooling1D,UpSampling1D,Reshape,Flatten
```

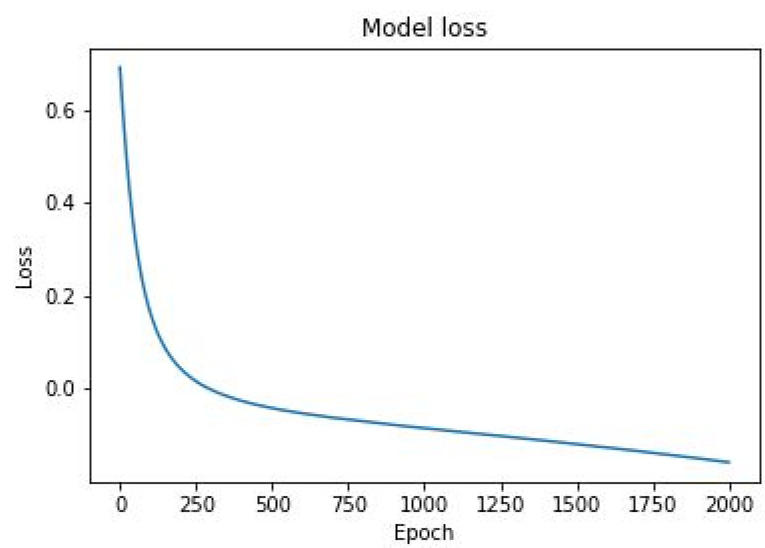
```
import matplotlib.pyplot as plt
from sklearn import preprocessing

x = loadmat('data_for_cnn.mat')
x = pd.DataFrame(x)
x = np.asarray(x)

x_temp = []
for i in range(len(x)):
   x_temp.append(x[i][0])
x_temp = np.asarray(x_temp)
x = x_temp
x = preprocessing.normalize(x)
x = x.reshape(x.shape[0],x.shape[1],1)

#input-convolution layer-pooling layer-FC-upsampling layer-transpose convolution layer
input = Input(shape=(1000,1))
encoder = Conv1D(32, 5, activation= 'relu' , padding= 'same')(input)
encoder = MaxPooling1D(4, padding= 'same')(encoder)
encoder = Flatten()(encoder)
encoded = Dense(500, activation='softmax')(encoder)
decoder = UpSampling1D(2)(encoded)
decoder = Reshape((1000,1))(decoder)
decoded = Conv1D(1, 5, activation='sigmoid', padding='same')(decoder)
autoencoder = Model(input, decoded)
autoencoder.summary()

opt = optimizers.Adam(lr=0.01)
autoencoder.compile(optimizer= opt, loss='mse')
#autoencoder.compile(optimizer= opt, loss='binary_crossentropy')
history = autoencoder.fit(x, x, epochs=2000, batch_size=512, shuffle=True)

# Plot training loss
plt.plot(history.history['loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.show()
plt.savefig('loss.png')
```

Q2 Results :

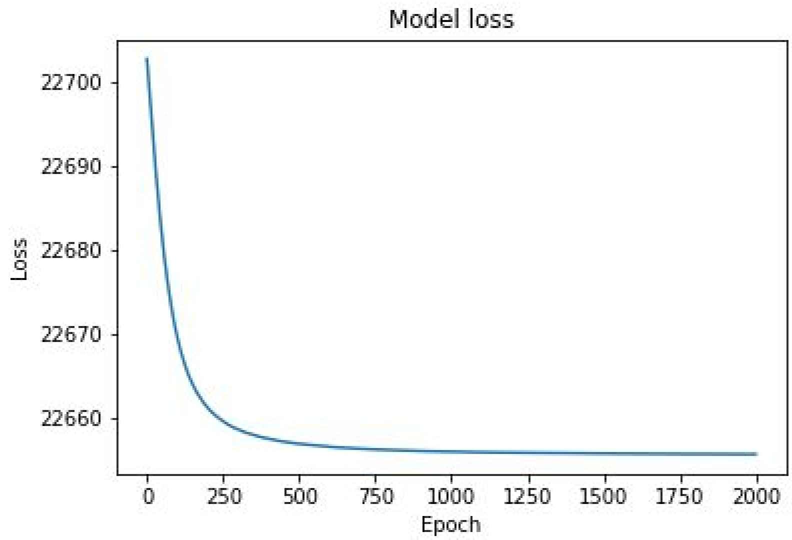
With Binary cross entropy, without normalizing data



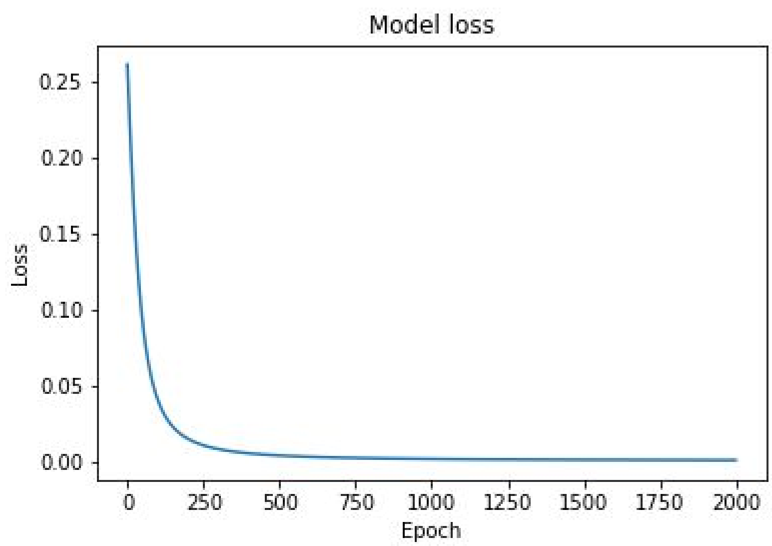
With Binary cross entropy and normalizing data



With Mean Squared Error, without normalizing data



With Mean Squared Error and normalizing data



Question 3

Implement the neuro-fuzzy inference system (NFIS) classifier for the classification task. You can use data4.xlsx file. The training and test instances can be selected using hold out cross- validation.

Solution :

Code :

```
'''
*** ANFIS ***
Multiclass Classification

Author :
Pranath Reddy
2016B5A30572H
'''

import keras
import tensorflow as tf
import numpy as np
import pandas as pd
import time
from sklearn.model_selection import train_test_split

# Function to set the class labels to predictions
def set(y):
   for i in range(len(y)):
       if(0.0<y[i]<=1.5):
           y[i] = 1.0
       if(1.5<y[i]<=2.5):
           y[i] = 2.0
       if(y[i]>=2.5):
           y[i] = 3.0
   return y

# Function to normalize the data
def norm(x):
```

```
    return (x - x.mean(axis=0))/x.std(axis=0)

# Adaptive neuro-fuzzy inference system implementation
class ANFIS:

    def __init__(self, n_inputs, n_rules, learning_rate=1e-2):
        self.n = n_inputs
        self.m = n_rules
        self.inputs = tf.placeholder(tf.float32, shape=(None, n_inputs))  # Input
        self.targets = tf.placeholder(tf.float32, shape=None)  # Desired output
        mu = tf.get_variable("mu", [n_rules * n_inputs],
                             initializer=tf.random_normal_initializer(0, 1))  # Means
of Gaussian MFS
        sigma = tf.get_variable("sigma", [n_rules * n_inputs],
                                initializer=tf.random_normal_initializer(0, 1))  #
Standard deviations of Gaussian MFS
        y = tf.get_variable("y", [1, n_rules],
initializer=tf.random_normal_initializer(0, 1))  # Sequent centers

        self.params = tf.trainable_variables()

        self.rul = tf.reduce_prod(
            tf.reshape(tf.exp(-0.5 * tf.square(tf.subtract(tf.tile(self.inputs, (1,
n_rules)), mu)) / tf.square(sigma)),
                       (-1, n_rules, n_inputs)), axis=2)  # Rule activations
        # Fuzzy base expansion function:
        num = tf.reduce_sum(tf.multiply(self.rul, y), axis=1)
        den = tf.clip_by_value(tf.reduce_sum(self.rul, axis=1), 1e-12, 1e12)
        self.out = tf.divide(num, den)

        self.loss = tf.losses.huber_loss(self.targets, self.out)  # Loss function
computation
        self.optimize =
tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(self.loss)  #
Optimization step
        self.init_variables = tf.global_variables_initializer()  # Variable initializer

    # Function to get predictions from test samples
    def infer(self, sess, x, targets=None):
```

```
        if targets is None:
            return sess.run(self.out, feed_dict={self.inputs: x})
        else:
            return sess.run([self.out, self.loss], feed_dict={self.inputs: x,
self.targets: targets})

    # Function to initiate and train the graph
    def train(self, sess, x, targets):
        yp, l, _ = sess.run([self.out, self.loss, self.optimize],
feed_dict={self.inputs: x, self.targets: targets})
        return l, yp

# Importing the data
data = pd.read_excel('data4.xlsx')
data = pd.DataFrame(data)
data = np.asarray(data)
y = data[:,-1]
x = data[:,:-1]
x = norm(x)

# Split train and test set
x_tr, x_ts, y_tr, y_ts = train_test_split(x, y, test_size=0.3)
m = x_tr.shape[0]
n = x_tr.shape[1]

# Hyperparameters
m = 16  # number of rules
alpha = 0.01  # learning rate
epochs= 2000
fis = ANFIS(n_inputs=7, n_rules=m, learning_rate=alpha)

# Initialize session to make computations on the Tensorflow graph
with tf.Session() as sess:
    # Initialize model parameters
    sess.run(fis.init_variables)
    trn_costs = []
    val_costs = []
    time_start = time.time()
    for epoch in range(epochs):
```

```
        # Train the model
        trn_loss, train_pred = fis.train(sess, x_tr, y_tr)
        # Evaluate on test set
        test_pred, val_loss = fis.infer(sess, x_ts, y_ts)
        # Print the training cost
        if epoch % 10 == 0:
            print("Train cost after epoch %i: %f" % (epoch, trn_loss))
        if epoch == epochs - 1:
            time_end = time.time()

yp = test_pred # Get the predictions
yp = set(yp)

# Confusion matrix and accuracy
y_actual = pd.Series(y_ts, name='Actual')
y_pred = pd.Series(yp, name='Predicted')
confmat = pd.crosstab(y_actual, y_pred)
print(confmat)

confmat = np.asarray(confmat)
Accuracy = float(confmat[0][0]+confmat[1][1]+confmat[2][2])/float(yp.shape[0])
print('Accuracy :' + ' ' + str(Accuracy))
```

Q3 Results :

```
Predicted   1.0   2.0   3.0
Actual
1.0          11     0     0
2.0           1    13     1
3.0           0     0    19
Accuracy : 0.9555555555555556
```