Neural Networks and Fuzzy Logic

BITS F312

Assignment 3

Name: Tejas Radhakrishnan

ID No.: 2017AAPS1736H

# 1. Convolutional Neural Network For ECG classification

Final validation accuracy : 90.2%

**Code**:

## Convolutional Neural Network classifier

from scipy.io import loadmat

import numpy as np

from keras.models import Sequential

from keras.layers import Dense, Activation, AveragePooling1D, Conv1D, Flatten

from keras import optimizers

from keras.utils import to\_categorical

from keras import regularizers

from sklearn.preprocessing import normalize

# Load data and shuffle

mat\_features = loadmat('data\_for\_cnn.mat')

mat\_labels = loadmat('y.mat')

features = mat\_features['ecg\_in\_window']

labels = mat\_labels['label']

data = np.zeros((1000, 1001))

data[:, :-1] = features

data[:, -1] = labels.flatten()

np.random.shuffle(data)

X = data[:, :-1]

y = data[:, -1]

X = X.reshape(X.shape[0], X.shape[1], 1)

y = to\_categorical(y, num\_classes = 2)

# Hyperparameters

reg = 0.9

epochs = 1000

learning\_rate = 1e-3

batch\_size = 256

holdout = 0.3

# Define model

model = Sequential()

model.add(Conv1D(50, 30, padding = 'same', input\_shape = (1000, 1), kernel\_regularizer=regularizers.l2(reg)))

model.add(AveragePooling1D())

model.add(Conv1D(50, 50, padding = 'same', kernel\_regularizer=regularizers.l2(reg)))

model.add(AveragePooling1D())

model.add(Flatten())

model.add(Dense(50, activation = 'relu', kernel\_regularizer=regularizers.l2(reg)))

model.add(Dense(30, activation = 'relu', kernel\_regularizer=regularizers.l2(reg)))

model.add(Dense(2, activation = 'softmax'))

# Train model and check validation accuracy

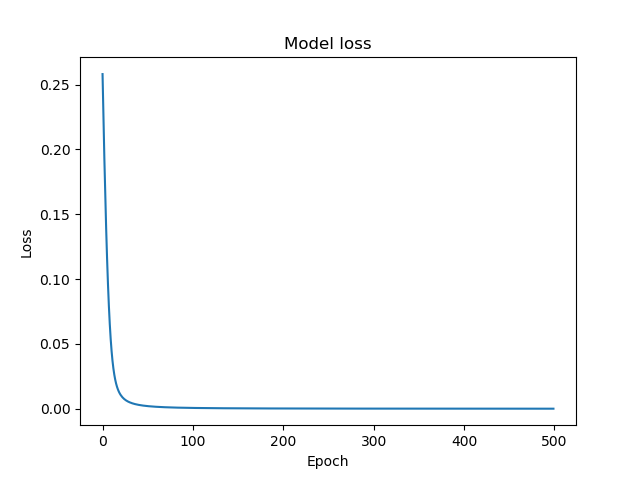
sgd = optimizers.SGD(lr = learning\_rate)

model.compile(loss = 'mean\_squared\_error', optimizer = sgd, metrics = ['accuracy'])

model.fit(X, y, validation\_split = holdout, epochs = epochs, batch\_size = batch\_size, use\_multiprocessing = True)

# 2. Convolutional Autoencoder

Loss graph:



**Code**:

## Convolutional Autoencoder

from scipy.io import loadmat

import numpy as np

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

# Load, shuffle and normalize data

mat\_features = loadmat('data\_for\_cnn.mat')

features = mat\_features['ecg\_in\_window']

data = np.zeros((1000, 1001))

np.random.shuffle(data)

X = data[:, :-1]

X = preprocessing.normalize(X, axis= 0)

X = X.reshape(X.shape[0], X.shape[1], 1)

# Input - Convolution layer - Max-Pooling layer - FC - Upsampling layer - Transpose convolution layer

model = Sequential()

model.add(Conv1D(32, 5, activation= 'relu' ,input\_shape = (1000, 1), padding= 'same'))

model.add(MaxPooling1D(4, padding= 'same'))

model.add(Flatten())

model.add(Dense(500, activation = 'sigmoid'))

model.add(UpSampling1D(2))

model.add(Reshape((1000, 1)))

model.add(Conv1D(1, 5, activation='sigmoid', padding='same'))

autoencoder = model

autoencoder.summary()

opt = optimizers.Adam(lr=0.01)

autoencoder.compile(optimizer= opt, loss='mse')

# Train model

history = autoencoder.fit(X, X, epochs=500, batch\_size=256, 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')

# 3. Neuro-fuzzy Inference System

**Accuracy** : 100%

Output Confusion Matrix:

|  |  |  |  |
| --- | --- | --- | --- |
| Actual output vs predicted output | 1 | 2 | 3 |
| 1 | 18 | 0 | 0 |
| 2 | 0 | 16 | 0 |
| 3 | 0 | 0 | 11 |

**Code (MATLAB):**

|  |
| --- |
| **%% Neuro Fuzzy Inference System clc; close all; clear all;  % Load, shuffle and normalize data [data4,a,b] = xlsread('data4.xlsx'); data4(:, end) = round(data4(:, end)); data4(:, 1:end-1) = normalize(data4(:, 1:end-1), 2); data = data4(randperm(size(data4, 1)), :);  % Holdout cross-validation X = data(1:105, :); X\_val = data(106:end, :);   % Train a system and predict on the validation set fis = anfis(X); anfisOutput = evalfis(fis, X\_val(:, 1:end-1));  % Generate confusion matrix out = round(anfisOutput); confusion\_matrix = zeros(3, 3);  targets = X\_val(:, end);  for i = 1:length(anfisOutput)  confusion\_matrix(targets(i), out(i)) = confusion\_matrix(targets(i), out(i)) + 1; end  accuracy = trace(confusion\_matrix)/45;** |