NEURAL NETWORK AND FUZZY LOGIC

ASSIGNMENT-3

PROBLEM 1

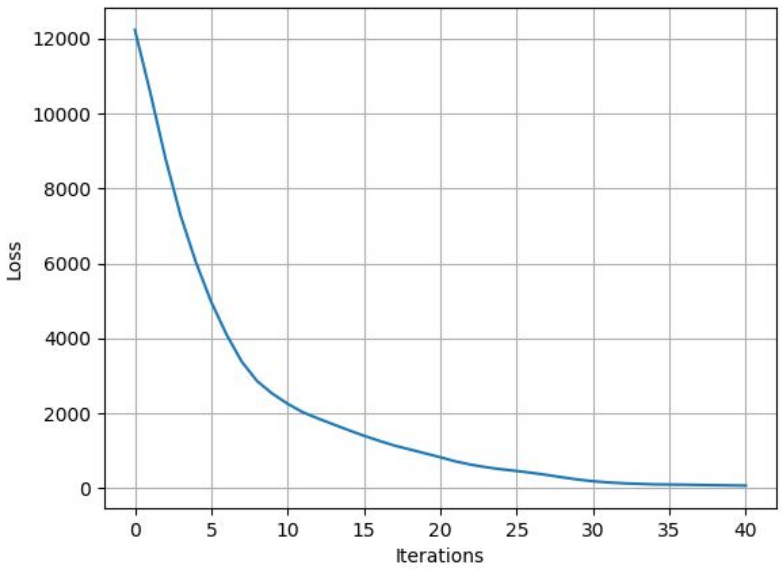
```python
import numpy as np
from scipy.io import loadmat
import tensorflow as tf
from matplotlib import pyplot as plt

m = n = 1000
def to_one_hot(Y):
    mm = Y.shape[0]
    Y_one_hot = np.zeros((mm, 2))
    for i in range(mm):
        Y_one_hot[i, Y[i]] = 1
    return Y_one_hot

def normalize_features(X_train, X_test):
    mu = np.mean(X_train, axis=0, keepdims=True)
    sigma = np.std(X_train, axis=0, keepdims=True)
    X_train = (X_train - mu) / sigma
    X_test = (X_test - mu) / sigma
    return X_train, X_test

def import_data():
    """Import data files and divide into training and test sets."""
    X = loadmat('data_for_cnn.mat')['ecg_in_window']
    Y = loadmat('class_label.mat')['label']
    # Randomly permute the data
    random_indices = np.random.permutation(m)
    X = X[random_indices]
    Y = Y[random_indices]
    train_set_size = 768
    test_set_size = m - train_set_size
    X_train, X_test = X[0:train_set_size], X[train_set_size:]
    Y_train, Y_test = Y[0:train_set_size], Y[train_set_size:]
    X_train, X_test = normalize_features(X_train, X_test)
    Y_train = to_one_hot(Y_train)
    Y_test = to_one_hot(Y_test)
    return X_train, X_test, Y_train, Y_test


```

```
def conv1d(x, W, b, s=1):
    """Conv1D wrapper, with bias and relu activation"""
    x = tf.nn.conv1d(x, W, stride=s, padding='SAME')
    x = tf.nn.bias_add(x, b)
    return tf.nn.relu(x)

def maxpool1d(x, k=2):
    """MaxPool1D wrapper"""
    return tf.layers.max_pooling1d(x, pool_size=k, strides=k, padding='SAME')

def conv_net(x, weights, biases, dropout):
    # Reshape to match ECG format [Width x Channel]
    # Tensor input become 4-D: [Batch Size, Width, Channel]
    x = tf.reshape(x, shape=[-1, m, 1])

    # Convolution Layer
    conv1 = conv1d(x, weights['wc1'], biases['bc1'])
    # Max Pooling (down-sampling)
    conv1 = maxpool1d(conv1, k=2)

    # Fully connected layers
    fc1 = tf.reshape(conv1, [-1, weights['wd1'].get_shape().as_list()[0]])
    fc1 = tf.add(tf.matmul(fc1, weights['wd1']), biases['bd1'])
    fc1 = tf.nn.relu(fc1)

    fc2 = tf.add(tf.matmul(fc1, weights['wd2']), biases['bd2'])
    fc2 = tf.nn.relu(fc2)
    fc2 = tf.nn.dropout(fc2, dropout)

    # Output, class prediction
    out = tf.add(tf.matmul(fc2, weights['out']), biases['out'])
    return out

if __name__ == '__main__':
    X_train, X_test, Y_train, Y_test = import_data()

    # Training parameters
```

```
    return out

if __name__ == '__main__':
    X_train, X_test, Y_train, Y_test = import_data()

    # Training parameters
    num_steps = 400
    learning_rate = 0.00001
    batch_size = 64
    display_step = 10

    # Network Parameters
    num_input = 1000
    num_classes = 2
    dropout = 0.70

    # tf Graph input
    X = tf.placeholder(tf.float32, [None, num_input])
    Y = tf.placeholder(tf.float32, [None, num_classes])
    keep_prob = tf.placeholder(tf.float32)


    weights = {
    'wc1': tf.Variable(tf.random_normal([ , 1, 64])),
    'wd1': tf.Variable(tf.random_normal([ 00*64, 1024])),
    'wd2': tf.Variable(tf.random_normal([1024, 20])),
    'out': tf.Variable(tf.random_normal([20, num_classes]))
    }

    biases = {
    'bc1': tf.Variable(tf.random_normal([64])),
    'bd1': tf.Variable(tf.random_normal([1024])),
    'bd2': tf.Variable(tf.random_normal([20])),
    'out': tf.Variable(tf.random_normal([num_classes]))
    }

    # Construct model
    logits = conv_net(X, weights, biases, keep_prob)
```

```
    prediction = tf.nn.softmax(logits)

    # Define loss and optimizer
    loss_op = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=Y))
    optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
    train_op = optimizer.minimize(loss_op)


    # Evaluate model
    correct_pred = tf.equal(tf.argmax(prediction, 1), tf.argmax(Y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

    # Initialize the variables (i.e. assign their default value)
    init = tf.global_variables_initializer()

    losses = []

    # Start training
    with tf.Session() as sess:

        sess.run(init)
        batch_start = 0
        for step in range(1, num_steps+1):
            batch_x, batch_y = X_train[batch_start : batch_start+batch_size], Y_train[batch_start : batch_start+batch_size]
            batch_start = (batch_start + batch_size) % batch_size
            sess.run(train_op, feed_dict={X: batch_x, Y: batch_y, keep_prob: 0.8})
            if step % display_step == 0 or step == 1:
                # Calculate batch loss and accuracy
                loss, acc = sess.run([loss_op, accuracy], feed_dict={X: batch_x, Y: batch_y, keep_prob: 1.0})
                print("Step " + str(step) + ", Minibatch Loss= " + \
                "{:.4f}".format(loss) + ", Training Accuracy= " + \
                "{:.3f}".format(acc))
                losses.append(loss)

    print("Optimization Finished!")

    print("Testing Accuracy:", \
    sess.run(accuracy, feed_dict={X: X_test, Y: Y_test, keep_prob: 1.0}))
```

Training Plot:



Test Evaluation:

Accuracy = 87.92%