سوال ۳-۱)

```python
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score


def softmax(z):
    # Calculate exponent term first
    exp_scores = np.exp(z)
    return exp_scores / np.sum(exp_scores, axis=1, keepdims=True)


def softmax_loss(y, y_hat):
    # Clipping calue
    minval = 0.000000000001
    # Number of samples
    m = y.shape[0]
    # Loss formula, note that np.sum sums up the entire matrix and therefore does the job of two sums from the 
formula
    loss = -1 / m * np.sum(y * np.log(y_hat.clip(min=minval)))
    return loss


# Log loss derivative, equal to softmax loss derivative
def loss_derivative(y, y_hat):
    return (y_hat - y)


def tanh_derivative(x):
    return (1 - np.power(x, 2))

def initialize_parameters(nn_input_dim, nn_hdim1, nn_hdim2, nn_output_dim):
    # First layer weights
    W1 = 2 * np.random.randn(nn_input_dim, nn_hdim1) - 1

    # First layer bias
    b1 = np.zeros((1, nn_hdim1))

    # Second layer weights
    W2 = 2 * np.random.randn(nn_hdim1, nn_hdim2) - 1

    # Second layer bias
    b2 = np.zeros((1, nn_hdim2))

    # Third layer weights
    W3 = 2 * np.random.randn(nn_hdim2, nn_output_dim) - 1

    # Second layer bias
    b3 = np.zeros((1, nn_output_dim))

    # Package and return model
    model = {'W1': W1, 'b1': b1, 'W2': W2, 'b2': b2, 'W3': W3, 'b3': b3}
    return model


def forward_prop(model, a0, output):
    # Load parameters from model
    W1, b1, W2, b2, W3, b3 = model['W1'], model['b1'], model['W2'], model['b2'], model['W3'], model['b3']

    # Linear step
    z1 = a0.dot(W1) + b1

    # First activation function
    a1 = np.tanh(z1)

    # Second linear step
    z2 = a1.dot(W2) + b2

    # Second activation function
    a2 = np.tanh(z2)

    z3 = a2.dot(W3) + b3

    a3 = softmax(z3)

    cache = {'a0': a0, 'z1': z1, 'a1': a1, 'z2': z2, 'a2': a2, 'z3': z3, 'a3': a3}
    return a3 if output else cache


def backward_prop(model, cache, y):
    # Load parameters from model
    W2, W3 = model['W2'], model['W3']
    # Load forward propagation results
    a0, a1, a2, a3 = cache['a0'], cache['a1'], cache['a2'], cache['a3']

    # Get number of samples
    m = y.shape[0]

    # Backpropagation
    # Calculate loss derivative with respect to output
    dz3 = loss_derivative(y=y, y_hat=a3)

    # Calculate loss derivative with respect to second layer weights
    dW3 = 1 / m * (a2.T).dot(dz3)

    # Calculate loss derivative with respect to second layer bias
    db3 = 1 / m * np.sum(dz3, axis=0)

    # Calculate loss derivative with respect to third layer
    dz2 = np.multiply(dz3.dot(W3.T), tanh_derivative(a2))

    # Calculate loss derivative with respect to second layer weights
    dW2 = 1 / m * (a1.T).dot(dz2)

    # Calculate loss derivative with respect to second layer bias
    db2 = 1 / m * np.sum(dz2, axis=0)

    # Calculate loss derivative with respect to first layer
    dz1 = np.multiply(dz2.dot(W2.T), tanh_derivative(a1))

    # Calculate loss derivative with respect to first layer weights
    dW1 = 1 / m * np.dot(a0.T, dz1)

    # Calculate loss derivative with respect to first layer bias
    db1 = 1 / m * np.sum(dz1, axis=0)

    # Store gradients
    grads = {'dW3': dW3, 'db3': db3, 'dW2': dW2, 'db2': db2, 'dW1': dW1, 'db1': db1}
    return grads


def update_parameters(model, grads, learning_rate):
    # Load parameters
    W1, b1, W2, b2, W3, b3 = model['W1'], model['b1'], model['W2'], model['b2'], model['W3'], model['b3']

    # Update parameters
    W1 -= learning_rate * grads['dW1']
    b1 -= learning_rate * grads['db1']
    W2 -= learning_rate * grads['dW2']
    b2 -= learning_rate * grads['db2']
    W3 -= learning_rate * grads['dW3']
    b3 -= learning_rate * grads['db3']

    # Store and return parameters
    model = {'W1': W1, 'b1': b1, 'W2': W2, 'b2': b2, 'W3': W3, 'b3': b3}
    return model


def predict(model, x):
    # Do forward pass
    c = forward_prop(model, x, False)
    # get y_hat
    y_hat = np.argmax(c['a3'], axis=1)
    return y_hat
```

سوال ۳-۲)

```python
import keras
from keras import backend as K
from keras.models import Sequential
from keras.layers import Activation
from keras.layers.core import Dense
from keras.optimizers import Adam
from keras.metrics import categorical_crossentropy
import pandas as pd

store_data = pd.read_csv('Drinks.csv', sep=',')
# print(store_data)
scaled_train_samples = []
train_labels = []
train_labels.append([str(store_data.values[0,j]) for j in range(1, 13)])

for i in range(0, 177):
    scaled_train_samples.append(str(store_data.values[i,j]) for j in range(1, 13))
    train_labels.append(str(store_data.values[i, j]) for j in range(13, 13))
layers = [
    Dense(8, input_shape=(13,), activation='tanh'),
    Dense(5, activation='tanh'),
    Dense(3, activation='softmax')
]

model = Sequential(layers)

a = model.compile(
    Adam(lr=.7),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

model.fit(
    scaled_train_samples,
    train_labels,
    batch_size=10,
    epochs=20,
    shuffle=True,
    verbose=3
)
```

سوال ۴)

```
from __future__ import print_function
import numpy as np
np.random.seed(1337)  # for reproducibility

from keras.datasets import mnist
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.optimizers import SGD, Adam, RMSprop
from keras.utils import np_utils


batch_size = 128
nr_classes = 10
nr_iterations = 20

(X_train, y_train), (X_test, y_test) = mnist.load_data()

X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255

Y_train = np_utils.to_categorical(y_train, nr_classes)
Y_test = np_utils.to_categorical(y_test, nr_classes)

model = Sequential()
model.add(Dense(196, input_shape=(784,)))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(10))
model.add(Activation('softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer=Adam(),
              metrics=['accuracy'])

history = model.fit(X_train, Y_train,
                    batch_size = batch_size, nb_epoch = nr_iterations,
                    verbose = 1, validation_data = (X_test, Y_test))
score = model.evaluate(X_test, Y_test, verbose = 0)
```