```
from random import seed
from random import random
from math import exp
import pandas as pd
# import matplotlib.pyplot as plt
import numpy as np
from string import digits
import string



def data_prep():
  data = pd.read_csv("spam.csv")
  print("Initial Dataset")
  print(data.head())
  data["v1"] = data['v1'].map({"ham" : 0, "spam" : 1})
  class_label = data['v1']
  bag_of_words = set()
  clean_messages = []
  symbols = string.punctuation
  printable = string.ascii_letters + string.digits + string.punctuation + ' '
  remove_digits = str.maketrans('', '', digits)
  stop_words = ['her', 'during', 'among', 'thereafter', 'only', 'hers', 'in', 'none', 'wit
  for mail in data["v2"]:
    mail = mail.lower()
    mail = ''.join(c if c in printable else r''.format(ord(c)) for c in mail)
    for char in symbols:
      if(char == "'"):
        mail = mail.replace(char, "")
      else:
        mail = mail.replace(char, " ")

    mail = ''.join(c + " " if c not in stop_words else r'' for c in mail.split())
    mail = mail.replace("  ", " ")
    mail = mail.translate(remove_digits)
    clean_messages.append(mail)
    bag_of_words = bag_of_words.union(set(mail.split()))

  data["v2"] = clean_messages
  print("After Cleaning the data : ")
  print(data.head())

  feature_vectors = []
  bag_of_words = list(bag_of_words)

  for i  in range(len(data)):
    feature_vector = [0]*(len(bag_of_words) + 1)
    feature_vector[-1] = data['v1'][i]
    for word in data['v2'][i].split():
      feature_vector[bag_of_words.index(word)] += 1
    feature_vectors.append(feature_vector)
  # print("The final featurelist is : ")
  # print(feature_vectors)
  return feature_vectors
```

```
def initialize_network(n_inputs, n_hidden, n_outputs):
  network = list()
  hidden_layer = [{'weights':[random() for i in range(n_inputs + 1)]} for i in range(n_hic
  network.append(hidden_layer)
  output_layer = [{'weights':[random() for i in range(n_hidden + 1)]} for i in range(n_out
  network.append(output_layer)
  return network


def activate(weights, inputs):
  activation = weights[-1]
  for i in range(len(weights)-1):
    activation += weights[i] * inputs[i]
  return activation


def transfer(activation):
  return 1.0 / (1.0 + exp(-activation))


def forward_propagate(network, row):
  inputs = row
  for layer in network:
    new_inputs = []
    for neuron in layer:
      activation = activate(neuron['weights'], inputs)
      neuron['output'] = transfer(activation)
      new_inputs.append(neuron['output'])
    inputs = new_inputs
  return inputs


def transfer_derivative(output):
  return output * (1.0 - output)



def backward_propagate_error(network, expected):
  for i in reversed(range(len(network))):
    layer = network[i]
    errors = list()
    if i != len(network)-1:
      for j in range(len(layer)):
        error = 0.0
        for neuron in network[i + 1]:
          error += (neuron['weights'][j] * neuron['delta'])
        errors.append(error)
    else:
      for j in range(len(layer)):
        neuron = layer[j]
        errors.append(expected[j] - neuron['output'])
    for j in range(len(layer)):
      neuron = layer[j]
      neuron['delta'] = errors[j] * transfer_derivative(neuron['output'])
```

```
def update_weights(network, row, l_rate):
  for i in range(len(network)):
    inputs = row[:-1]
    if i != 0:
      inputs = [neuron['output'] for neuron in network[i - 1]]
    for neuron in network[i]:
      for j in range(len(inputs)):
        neuron['weights'][j] += l_rate * neuron['delta'] * inputs[j]
      neuron['weights'][-1] += l_rate * neuron['delta']


def train_network(network, train, l_rate, n_epoch, n_outputs):
  for epoch in range(n_epoch):
    sum_error = 0
    if(epoch > 50000):
      l_rate = 1/epoch * 100
    for row in train:
      outputs = forward_propagate(network, row)
      expected = [0 for i in range(n_outputs)]
      expected[int(row[-1])] = 1
      sum_error += sum([(expected[i]-outputs[i])**2 for i in range(len(expected))])
      backward_propagate_error(network, expected)
      update_weights(network, row, l_rate)
    # if(epoch % 100 == 0):
    print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))


def predict(network, row):
  outputs = forward_propagate(network, row)
  return outputs.index(max(outputs))


dataset = data_prep()
seed(1)
n_inputs = len(dataset[0]) - 1
n_outputs = len(set([row[-1] for row in dataset]))
network = initialize_network(n_inputs, 4, n_outputs)
print("The initial network is : ")
print(network)
train_network(network, dataset, 0.5, 100, n_outputs)
print("The trained nueral network is : ")
print(network)
```

```
Initial Dataset
     v1                                                  v2
0   ham  Go until jurong point, crazy.. Available only ...
1   ham                      Ok lar... Joking wif u oni...
2  spam  Free entry in 2 a wkly comp to win FA Cup fina...
3   ham  U dun say so early hor... U c already then say...
4   ham  Nah I don't think he goes to usf, he lives aro...
After Cleaning the data :
   v1                                                 v2
0   0  jurong point crazy available bugis n great wor...
1   0                           ok lar joking wif u oni
2   1  free entry  wkly comp win fa cup final tkts st...
3   0                                u dun early hor u c
4   0                     nah dont think goes usf lives
The initial network is :
[[{'weights': [0.13436424411240122, 0.8474337369372327, 0.763774618976614, 0.2550690
>epoch=0, lrate=0.500, error=1266.664
>epoch=1, lrate=0.500, error=454.102
>epoch=2, lrate=0.500, error=216.186
>epoch=3, lrate=0.500, error=150.878
>epoch=4, lrate=0.500, error=117.550
>epoch=5, lrate=0.500, error=105.443
>epoch=6, lrate=0.500, error=87.698
>epoch=7, lrate=0.500, error=79.919
>epoch=8, lrate=0.500, error=75.005
>epoch=9, lrate=0.500, error=72.211
>epoch=10, lrate=0.500, error=71.368
>epoch=11, lrate=0.500, error=69.479
>epoch=12, lrate=0.500, error=67.933
>epoch=13, lrate=0.500, error=67.753
>epoch=14, lrate=0.500, error=67.526
>epoch=15, lrate=0.500, error=67.047
>epoch=16, lrate=0.500, error=64.523
>epoch=17, lrate=0.500, error=60.351
>epoch=18, lrate=0.500, error=57.837
>epoch=19, lrate=0.500, error=54.655
>epoch=20, lrate=0.500, error=49.509
>epoch=21, lrate=0.500, error=47.957
>epoch=22, lrate=0.500, error=46.677
>epoch=23, lrate=0.500, error=46.331
>epoch=24, lrate=0.500, error=46.213
>epoch=25, lrate=0.500, error=46.121
>epoch=26, lrate=0.500, error=46.027
>epoch=27, lrate=0.500, error=45.893
>epoch=28, lrate=0.500, error=45.519
>epoch=29, lrate=0.500, error=43.782
>epoch=30, lrate=0.500, error=42.340
>epoch=31, lrate=0.500, error=40.481
>epoch=32, lrate=0.500, error=40.019
>epoch=33, lrate=0.500, error=40.363
>epoch=34, lrate=0.500, error=35.802
>epoch=35, lrate=0.500, error=32.354
>epoch=36, lrate=0.500, error=32.260
>epoch=37, lrate=0.500, error=32.226
>epoch=38, lrate=0.500, error=32.200
>epoch=39, lrate=0.500, error=32.178
>epoch=40, lrate=0.500, error=32.158
>epoch=41, lrate=0.500, error=32.141
>epoch=42, lrate=0.500, error=32.126
>epoch=43, lrate=0.500, error=32.111
>epoch=44, lrate=0.500, error=32.098
```

```
>epoch=45, lrate=0.500, error=32.086
>epoch=46, lrate=0.500, error=32.075
>epoch=47, lrate=0.500, error=32.065
>epoch=48, lrate=0.500, error=32.055
>epoch=49, lrate=0.500, error=32.046
>epoch=50, lrate=0.500, error=32.037
>epoch=51, lrate=0.500, error=32.028
>epoch=52, lrate=0.500, error=32.020
>epoch=53, lrate=0.500, error=32.012
>epoch=54, lrate=0.500, error=32.004
>epoch=55, lrate=0.500, error=31.996
>epoch=56, lrate=0.500, error=31.989
>epoch=57, lrate=0.500, error=31.981
>epoch=58, lrate=0.500, error=31.973
>epoch=59, lrate=0.500, error=31.965
>epoch=60, lrate=0.500, error=31.956
>epoch=61, lrate=0.500, error=31.947
>epoch=62, lrate=0.500, error=31.937
>epoch=63, lrate=0.500, error=31.926
>epoch=64, lrate=0.500, error=31.914
>epoch=65, lrate=0.500, error=31.898
>epoch=66, lrate=0.500, error=31.878
>epoch=67, lrate=0.500, error=31.849
>epoch=68, lrate=0.500, error=31.804
>epoch=69, lrate=0.500, error=31.712
>epoch=70, lrate=0.500, error=31.431
>epoch=71, lrate=0.500, error=30.083
>epoch=72, lrate=0.500, error=28.367
>epoch=73, lrate=0.500, error=27.893
>epoch=74, lrate=0.500, error=27.779
>epoch=75, lrate=0.500, error=27.663
>epoch=76, lrate=0.500, error=27.461
>epoch=77, lrate=0.500, error=26.809
>epoch=78, lrate=0.500, error=25.962
>epoch=79, lrate=0.500, error=25.316
>epoch=80, lrate=0.500, error=24.701
>epoch=81, lrate=0.500, error=24.092
>epoch=82, lrate=0.500, error=23.987
>epoch=83, lrate=0.500, error=23.836
>epoch=84, lrate=0.500, error=22.969
>epoch=85, lrate=0.500, error=22.139
>epoch=86, lrate=0.500, error=22.122
>epoch=87, lrate=0.500, error=22.110
>epoch=88, lrate=0.500, error=22.099
>epoch=89, lrate=0.500, error=22.090
>epoch=90, lrate=0.500, error=22.081
>epoch=91, lrate=0.500, error=22.073
```

```
#As there was a reconnection error after completion of the perivious cell, I had to copy p
network = [[{'weights': [0.13432625139028556, 0.847323888780226, 0.7640138287736761, 0.255
print(network)
```

```
[[{'weights': [0.13432625139028556, 0.847323888780226, 0.7640138287736761, 0.2550731
```

The trained nueral network is :

```
data_set = data_prep()
```

```
Initial Dataset
     v1                                                  v2
0   ham  Go until jurong point, crazy.. Available only ...
1   ham                      Ok lar... Joking wif u oni...
2  spam  Free entry in 2 a wkly comp to win FA Cup fina...
3   ham  U dun say so early hor... U c already then say...
4   ham  Nah I don't think he goes to usf, he lives aro...
After Cleaning the data :
   v1                                                  v2
0   0  jurong point crazy available bugis n great wor...
1   0                           ok lar joking wif u oni
2   1  free entry  wkly comp win fa cup final tkts st
```

```
acc = 0
for i in range(len(data_set)):
  if(predict(network, data_set[i]) == list(data_set[i])[-1]):
    acc += 1

print("accuracy is : ", acc/len(data_set))
```

```
accuracy is :  0.7986360373295046
```