```
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn import model_selection
from sklearn import preprocessing


class NeuralNet:
    def __init__(self, train, w12=0, w23=0, w34=0, trsf1=False, trsf2=False, trsf3=False):
        h1 = 10
        h2 = 5
        h3 = 5
        np.random.seed(1)
        train_dataset = train
        nrows, ncols = train_dataset.shape
        self.X = train_dataset[:, :ncols-1]
        self.y = train_dataset[:, ncols-1:ncols]
        #
        # Find number of input and output layers from the dataset
        #
        input_layer_size = len(self.X[0])
        if not isinstance(self.y, np.ndarray):
            output_layer_size = 1
        else:
            output_layer_size = len(np.unique(self.y))
        output_layer_size = 1
        # print(self.y)
        # print(output_layer_size)

        # assign random weights to matrices in network
        # number of weights connecting layers = (no. of nodes in previous layer) x (no. of no
        self.w01 = 2 * np.random.random((input_layer_size, h1)) - 1
        self.X01 = self.X
        self.w12 = 2 * np.random.random((h1, h2)) - 1
        self.X12 = np.zeros((len(self.X), h1))
        self.delta12 = np.zeros((nrows, h1))
        self.w23 = 2 * np.random.random((h2, h3)) - 1
        self.X23 = np.zeros((len(self.X), h2))
        self.delta23 = np.zeros((nrows, h2))
        self.w34 = 2 * np.random.random((h3, output_layer_size)) - 1
        self.X34 = np.zeros((len(self.X), h3))
        self.delta34 = np.zeros((nrows, h3))
        self.deltaOut = np.zeros((nrows, output_layer_size))
        if trsf1 != 0:
            self.w12 = w12
        if trsf2 != 0:
            self.w23 = w23
        if trsf3 != 0:
            self.w34 = w34
        # print(self.w01.shape)
```

```
        # print(self.X01.shape)
        # print(self.w12.shape)
        # print(self.X12.shape)
        # print(self.w23.shape)
        # print(self.X23.shape)
        # print(self.w34.shape)
        # print(self.X34.shape)
        # print(self.delta12.shape)
        # print(self.delta23.shape)
        # print(self.delta34.shape)
        # print(self.deltaOut.shape)


    def __activation(self, x, activation="sigmoid"):
        if activation == "sigmoid":
            x = np.clip(x, -500, 500)
            return 1 / (1 + np.exp(-x))
        if activation == "tanh":
            return (np.exp(x) - np.exp(-x)) / (np.exp(x) + np.exp(-x))
            # return 2*self.__sigmoid(2*x) - 1
        if activation == "ReLu":
            return np.maximum(x, 0)

    def __activation_derivative(self, x, activation="sigmoid"):
        if activation == "sigmoid":
            return x * (1 - x)
        if activation == "tanh":
            return 1 - np.power(x, 2)
        if activation == "ReLu":
            return 1 * (x > 0)


    def train(self, max_iterations = 1000, learning_rate = 0.05, activation="sigmoid"):
        self.activation = activation
        for iteration in range(max_iterations):
            out = self.forward_pass(activation)
            # print(out)
            error = 0.5 * np.power((out - self.y), 2)
            self.backward_pass(out, activation)
            update_layer3 = learning_rate * self.X34.T.dot(self.deltaOut)
            update_layer2 = learning_rate * self.X23.T.dot(self.delta34)
            update_layer1 = learning_rate * self.X12.T.dot(self.delta23)
            update_input = learning_rate * self.X01.T.dot(self.delta12)
            self.w34 += update_layer3
            self.w23 += update_layer2
            self.w12 += update_layer1
            self.w01 += update_input

        # print("After " + str(max_iterations) + " iterations, the total error is " + str(np.
        # print("The final weight vectors are (starting from input to output layers)")
        # print(self.w01)
        # print(self.w12)
```

```
        # print(self.w23)
        # print(self.w34)

    def forward_pass(self, activation="sigmoid"):
        # pass our inputs through our neural network
        in1 = np.dot(self.X01, self.w01)
        self.X12 = self.__activation(in1, activation)
        in2 = np.dot(self.X12, self.w12)
        self.X23 = self.__activation(in2, activation)
        in3 = np.dot(self.X23, self.w23)
        self.X34 = self.__activation(in3, activation)
        in4 = np.dot(self.X34, self.w34)
        out = self.__activation(in4, "sigmoid")
        return out

    def backward_pass(self, out, activation="sigmoid"):
        # pass our inputs through our neural network
        self.deltaOut = (self.y - out) * (self.__activation_derivative(out, "sigmoid"))
        self.delta34 = (self.deltaOut.dot(self.w34.T)) * (self.__activation_derivative(self.X
        self.delta23 = (self.delta34.dot(self.w23.T)) * (self.__activation_derivative(self.X2
        self.delta12 = (self.delta23.dot(self.w12.T)) * (self.__activation_derivative(self.X1

    def predict(self, test):
        # raw_input = pd.read_csv(test)
        test_dataset = test
        nrows, ncols = test_dataset.shape
        self.X = test_dataset[:, :ncols-1]
        self.y = test_dataset[:, ncols-1:ncols]
        self.X01 = self.X
        out = self.forward_pass(self.activation)
        error = 0.5 * np.power((out - self.y), 2)
        rows, cols = out.shape
        # print(error)
        # print(self.activation)
        # print(out)
        for row in range(rows):
            for col in range(cols):
                if out[row][col] >= 0.5:
                    out[row][col] = 1.
                else:
                    out[row][col] = 0.
        # print(out)
        # print(self.y)
        right = int(np.sum(out==self.y))
        total = len(self.y)
        print("right classification: %f" % right)
        print("total data: %f" % total)
        return right / total
```

```
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/hepatitis/hepatit
names = ['Class','Age','SEX','STEROID', 'ANTIVIRALS', 'FATIGUE', 'Deg-MALAISE', 'ANOREXIA', '
'SPLEEN PALPABLE', 'SPIDERS', 'ASCITES', 'VARICES', 'BILIRUBIN', 'ALK PHOSPHATE', 'SGOT', 'AL
# print(df)

# fill ? to most frequent value
del df['LIVER BIG']
del df['LIVER FIRM']
del df['ALK PHOSPHATE']
del df['ALBUMIN']
del df['PROTIME']
# print(df)
imp = SimpleImputer(missing_values='?', strategy='most_frequent')
df = imp.fit_transform(df)
df = df.astype(np.float)
min_max_scaler = preprocessing.MinMaxScaler()
df = min_max_scaler.fit_transform(df)
# print(df)

trainDF, testDF = model_selection.train_test_split(df, test_size=0.2)
neural_network = NeuralNet(trainDF)

neural_network.train(max_iterations=100000, learning_rate=0.001, activation="sigmoid")
# neural_network.train(max_iterations=10000, learning_rate=0.2, activation="tanh")
# neural_network.train(max_iterations=50000, learning_rate=0.1, activation="ReLu")

acc = neural_network.predict(testDF)
print("Accuracy: %f" % acc)
```

```
right classification: 20.000000
total data: 31.000000
Accuracy: 0.645161
```

```
from keras.models import Sequential
from keras.layers import Flatten, Dense, Activation
import pandas as pd
import numpy as np
import sklearn as sk
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from keras.callbacks import EarlyStopping, ModelCheckpoint

import warnings
warnings.filterwarnings("ignore")
X = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer/brea

X = X.replace("?", np.nan)
datacopy = X.dropna()
datacopy['Class'] = X.Class.map({'no-recurrence-events':0, 'recurrence-events':1})
datacopy['Age'] = X.Age.map({'10-19':1, '20-29':2,'30-39':3, '40-49':4,'50-59':5, '60-69':6,'
datacopy['Menopause'] = X.Menopause.map({'lt40':1, 'ge40':2, 'premeno':3})
datacopy['TumorSize'] = X.TumorSize.map({'0-4':1, '5-9':2,'10-14':3, '15-19':4,'20-24':5, '25
```

```
datacopy['InvNodes'] = X.InvNodes.map({'0-2':1, '3-5':2,'6-8':3, '9-11':4,'12-14':5, '15-17':
datacopy['NodeCaps'] = X.NodeCaps.map({'no':0, 'yes':1})
datacopy['Breast'] = X.Breast.map({'left':1, 'right':2})
datacopy['BreastQuadrant'] = X.BreastQuadrant.map({'left_up':1, 'left_low':2, 'right_up':3, '
datacopy['Irradiated'] = X.Irradiated.map({'no':0, 'yes':1})
x = datacopy.values
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
datacopy = pd.DataFrame(x_scaled)
train, test = train_test_split(datacopy, test_size=0.20, random_state=42)

ncols = len(train.columns)
nrows = len(train.index)
ncolstest = len(test.columns)
nrowstest = len(test.index)
xtrain = train.iloc[:, 0:(ncols -1)].values.reshape(nrows, ncols-1)
ytrain = train.iloc[:, (ncols-1)].values.reshape(nrows, 1)
xtest = test.iloc[:, 0:(ncolstest -1)].values.reshape(nrowstest, ncolstest-1)
ytest = test.iloc[:, (ncolstest-1)].values.reshape(nrowstest, 1)
xtrain = xtrain.reshape(221,9,1)
print(xtrain.shape)
print(ytrain.shape)


model = Sequential()
model.add(Flatten())
model.add(Dense(10, input_shape=(221,9,), activation='sigmoid'))
model.add(Dense(5, activation='sigmoid'))
model.add(Dense(5, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['mse', 'accuracy'])
earlystop = [EarlyStopping(monitor='val_loss', patience=2)]
model.fit(xtrain, ytrain, epochs=1000, verbose=1, validation_split = 0.20, callbacks = earlys
```

```
(221, 9, 1)
(221, 1)
Train on 176 samples, validate on 45 samples
Epoch 1/1000
176/176 [==============================] - 0s 1ms/step - loss: 0.5392 - mean_squared_err
Epoch 2/1000
176/176 [==============================] - 0s 163us/step - loss: 0.5382 - mean_squared_e
Epoch 3/1000
176/176 [==============================] - 0s 147us/step - loss: 0.5376 - mean_squared_e
Epoch 4/1000
176/176 [==============================] - 0s 179us/step - loss: 0.5373 - mean_squared_e
Epoch 5/1000
176/176 [==============================] - 0s 157us/step - loss: 0.5373 - mean_squared_e
Epoch 6/1000
176/176 [==============================] - 0s 185us/step - loss: 0.5366 - mean_squared_e
Epoch 7/1000
176/176 [==============================] - 0s 175us/step - loss: 0.5367 - mean_squared_e
Epoch 8/1000
176/176 [==============================] - 0s 137us/step - loss: 0.5364 - mean_squared_e
Epoch 9/1000
176/176 [==============================] - 0s 150us/step - loss: 0.5363 - mean_squared_e
Epoch 10/1000
176/176 [==============================] - 0s 137us/step - loss: 0.5362 - mean_squared_e
Epoch 11/1000
176/176 [==============================] - 0s 147us/step - loss: 0.5358 - mean_squared_e
Epoch 12/1000
176/176 [==============================] - 0s 161us/step - loss: 0.5360 - mean_squared_e
Epoch 13/1000
176/176 [==============================] - 0s 164us/step - loss: 0.5359 - mean_squared_e
Epoch 14/1000
176/176 [==============================] - 0s 159us/step - loss: 0.5357 - mean_squared_e
Epoch 15/1000
176/176 [==============================] - 0s 190us/step - loss: 0.5357 - mean_squared_e
<keras.callbacks.History at 0x7f57bca84cc0>
```

```
for layer in model.layers:
    weights = layer.get_weights()
    print(len(weights))
    for i in range(len(weights)):
      print(weights[i].shape)
```

```
w12 = model.layers[2].get_weights()[0]
w23 = model.layers[3].get_weights()[0]
w34 = model.layers[4].get_weights()[0]
print(w12)
print(w23)
print(w34)
```

```
[[-0.4411568  -0.2563787  -0.15654486  0.32602698  0.3450132 ]
 [ 0.53321457  0.20898926  0.42458516 -0.27172434 -0.3139483 ]
 [ 0.09064068 -0.4219764  -0.00227138 -0.16774897 -0.18097131]
 [ 0.40318683 -0.08572397  0.44866502  0.10444221  0.58725965]
 [ 0.25082833 -0.32222587  0.36827046  0.25045776  0.330408  ]
 [ 0.36708882  0.35438073  0.35897493  0.08795936  0.25760293]
 [-0.02878102 -0.14062709 -0.4504123  -0.602353    0.27582568]
 [-0.5969011  -0.00346398 -0.5146781  -0.05060013 -0.18235168]
 [ 0.07163066  0.14692973 -0.5016739   0.28238168  0.23550548]
 [ 0.12030546  0.57316875 -0.20257989  0.3613475  -0.18705091]]
[[-0.449419    0.6755632  -0.03689308 -0.05011914 -0.40595615]
 [-0.02633006  0.3525872  -0.13608697  0.75131136 -0.23000073]
 [-0.28917846  0.01169886  0.37944195 -0.071438   -0.24511667]
 [ 0.5935708   0.6429499   0.6306464   0.6380768  -0.3104679 ]
 [-0.510252   -0.43931666  0.46053568  0.30114168  0.51932776]]
[[ 0.82083035]
 [-0.82470363]
 [-0.83127767]
 [-0.19471633]
 [-0.9502757 ]]
```

```
neural_network = NeuralNet(trainDF, w12=w12, trsf1=True)
neural_network.train(max_iterations=100000, learning_rate=0.001, activation="sigmoid")
acc = neural_network.predict(testDF)
print("Accuracy: %f" % acc)
neural_network = NeuralNet(trainDF, w23=w23, trsf2=True)
neural_network.train(max_iterations=100000, learning_rate=0.001, activation="sigmoid")
acc = neural_network.predict(testDF)
print("Accuracy: %f" % acc)
neural_network = NeuralNet(trainDF, w34=w34, trsf3=True)
neural_network.train(max_iterations=100000, learning_rate=0.001, activation="sigmoid")
acc = neural_network.predict(testDF)
print("Accuracy: %f" % acc)
neural_network = NeuralNet(trainDF, w12, w23, w34, trsf1=True, trsf2=True, trsf3=True)
neural_network.train(max_iterations=100000, learning_rate=0.001, activation="sigmoid")
acc = neural_network.predict(testDF)
print("Accuracy: %f" % acc)
```

```
right classification: 21.000000
total data: 31.000000
Accuracy: 0.677419
right classification: 23.000000
total data: 31.000000
Accuracy: 0.741935
right classification: 22.000000
total data: 31.000000
Accuracy: 0.709677
right classification: 23.000000
total data: 31.000000
Accuracy: 0.741935
```