IT 402

Assignment 3 - Multi Layer Perceptron

Name: Niraj Nandish

Roll no.: 191IT234

```
import math
import numpy as np
import pandas as pd
import seaborn as sn
from csv import reader
from random import seed
from random import randrange
import matplotlib.pyplot as plt
from sklearn import preprocessing, datasets
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import KFold
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
import warnings

warnings.filterwarnings("ignore")
np.random.seed(0)

```

```
!rm -rf SPECT.train
!wget https://archive.ics.uci.edu/ml/machine-learning-databases/spect/SPECT.train
```

```
--2022-09-10 18:56:32--  https://archive.ics.uci.edu/ml/machine-learning-databases/spect/SPECT.train
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.252
```

```
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.252|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3758 (3.7K) [application/x-httpd-php]
Saving to: ‘SPECT.train’

SPECT.train         100%[===================>]   3.67K  --.-KB/s    in 0s

2022-09-10 18:56:34 (33.2 MB/s) - ‘SPECT.train’ saved [3758/3758]
```

```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))


def loss(y, y_hat):
    minval = 0.000000000001
    m = y.shape[0]
    loss = -1 / m * np.sum(y * np.log(y_hat.clip(min=minval)))
    return loss


def sigmoid_derivative(z):
    return z * (1 - z)


def loss_derivative(y, y_hat):
    return y_hat - y


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


def calc_accuracy(model, x, y):
    m = y.shape[0]
    pred = predict(model, x)
    pred = pred.reshape(y.shape)
    error = np.sum(np.abs(pred - y))
```

```python
    return (m - error) / m * 100

```

```python
def forward_prop(model, a0):
    W1, b1, W2, b2, W3, b3, W4, b4 = (
        model["W1"],
        model["b1"],
        model["W2"],
        model["b2"],
        model["W3"],
        model["b3"],
        model["W4"],
        model["b4"],
    )
    z1 = a0.dot(W1) + b1
    a1 = np.tanh(z1)
    z2 = a1.dot(W2) + b2
    a2 = np.tanh(z2)
    z3 = a2.dot(W3) + b3
    a3 = np.tanh(z3)
    z4 = a3.dot(W4) + b4
    a4 = sigmoid(z4)
    cache = {
        "a0": a0,
        "z1": z1,
        "a1": a1,
        "z2": z2,
        "a2": a2,
        "a3": a3,
        "z3": z3,
        "a4": a4,
        "z4": z4,
    }
    return cache

```

```python

def backward_prop(model, cache, y):
    W1, b1, W2, b2, W3, b3, W4, b4 = (
        model["W1"],
        model["b1"],
        model["W2"],
        model["b2"],
        model["W3"],
        model["b3"],
        model["W4"],
        model["b4"],
    )
    a0, a1, a2, a3, a4 = cache["a0"], cache["a1"], cache["a2"], cache["a3"], cache["a4"]
    m = y.shape[0]
    dz4 = np.multiply(loss_derivative(y=y, y_hat=a4), sigmoid_derivative(a4))
    dW4 = 1 / m * (a3.T).dot(dz4)
    db4 = 1 / m * np.sum(dz4, axis=0)
    dz3 = np.multiply(dz4.dot(W4.T), tanh_derivative(a3))
    dW3 = 1 / m * np.dot(a2.T, dz3)
    db3 = 1 / m * np.sum(dz3, axis=0)
    dz2 = np.multiply(dz3.dot(W3.T), tanh_derivative(a2))
    dW2 = 1 / m * np.dot(a1.T, dz2)
    db2 = 1 / m * np.sum(dz2, axis=0)
    dz1 = np.multiply(dz2.dot(W2.T), tanh_derivative(a1))
    dW1 = 1 / m * np.dot(a0.T, dz1)
    db1 = 1 / m * np.sum(dz1, axis=0)
    grads = {
        "dW4": dW4,
        "db4": db4,
        "dW3": dW3,
        "db3": db3,
        "dW2": dW2,
        "db2": db2,
        "dW1": dW1,
        "db1": db1,
    }
    return grads
```

```python


def update_parameters(model, grads, learning_rate):
    W1, b1, W2, b2, W3, b3, W4, b4 = (
        model["W1"],
        model["b1"],
        model["W2"],
        model["b2"],
        model["W3"],
        model["b3"],
        model["W4"],
        model["b4"],
    )
    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"]
    W4 -= learning_rate * grads["dW4"]
    b4 -= learning_rate * grads["db4"]
    model = {
        "W1": W1,
        "b1": b1,
        "W2": W2,
        "b2": b2,
        "W3": W3,
        "b3": b3,
        "W4": W4,
        "b4": b4,
    }
    return model

```

```python
def initialize_parameters(nn_input_dim, nn_hdim, nn_output_dim):
    W1 = np.random.randn(nn_input_dim, nn_hdim[0])
    b1 = np.zeros((1, nn_hdim[0]))
```

```
    W2 = np.random.randn(nn_hdim[0], nn_hdim[1])
    b2 = np.zeros((1, nn_hdim[1]))
    W3 = np.random.randn(nn_hdim[1], nn_hdim[2])
    b3 = np.zeros((1, nn_hdim[2]))
    W4 = np.random.rand(nn_hdim[2], nn_output_dim)
    b4 = np.zeros((1, nn_output_dim))
    model = {
        "W1": W1,
        "b1": b1,
        "W2": W2,
        "b2": b2,
        "W3": W3,
        "b3": b3,
        "W4": W4,
        "b4": b4,
    }
    return model

```

```
def predict(model, x):
    c = forward_prop(model, x)
    y_hat = np.argmax(c["a4"], axis=1)
    return y_hat


def train(model, X_, y_, learning_rate, epochs=1000, print_loss=False):
    for i in range(0, epochs):
        cache = forward_prop(model, X_)
        grads = backward_prop(model, cache, y_)
        model = update_parameters(model=model, grads=grads, learning_rate=learning_rate)
    return model


def evaluate_algorithm(dataset, n_folds, l_rate, in_dim, hid_dim, out_dim):
    kf = KFold(n_splits=n_folds, random_state=None, shuffle=True)
    scores = list()
    f = 1
```

```python
    for train_index, test_index in kf.split(dataset):
        print("\nFold {}".format(f))
        f += 1
        df_train, df_test = dataset[train_index], dataset[test_index]
        df_train = pd.DataFrame(df_train)
        df_test = pd.DataFrame(df_test)
        train_Y = pd.get_dummies(df_train.iloc[:, -1]).values
        train_X = df_train.iloc[:, :-1].values
        test_Y = pd.get_dummies(df_test.iloc[:, -1]).values
        test_X = df_test.iloc[:, :-1].values
        model = initialize_parameters(
            nn_input_dim=in_dim, nn_hdim=hid_dim, nn_output_dim=out_dim
        )
        model = train(
            model, train_X, train_Y, learning_rate=l_rate, epochs=4500, print_loss=True
        )
        y_hat = predict(model, test_X)
        y_true = test_Y.argmax(axis=1)
        accuracy = accuracy_score(y_pred=y_hat, y_true=y_true) * 100
        print("Accuracy = {}".format(accuracy))
        scores.append(accuracy)
    return scores

```

SPECTF Dataset

```python
data = pd.read_csv("SPECT.train", header=None)
data.head()

```

	0	1	2	3	4	5	6	7	8	9	...	13	14	15	16	17	18	19	20	21	22
0	1	0	0	0	1	0	0	0	1	1	...	1	1	0	0	0	0	0	0	0	0
1	1	0	0	1	1	0	0	0	1	1	...	1	1	0	0	0	0	0	0	0	1
2	1	1	0	1	0	1	0	0	1	0	...	1	0	0	0	0	0	0	0	0	0

```
print("Learning Rate = {}".format(0.01))
scores = evaluate_algorithm(
    data.values, n_folds=5, l_rate=0.01, in_dim=22, hid_dim=[15, 10, 5], out_dim=2
)

```

```
Learning Rate = 0.01

Fold 1
Accuracy = 68.75

Fold 2
Accuracy = 43.75

Fold 3
Accuracy = 62.5

Fold 4
Accuracy = 50.0

Fold 5
Accuracy = 50.0
```

```
print("Learning Rate = {}".format(0.001))
scores = evaluate_algorithm(
    data.values, n_folds=5, l_rate=0.001, in_dim=22, hid_dim=[15, 10, 5], out_dim=2
)

```

```
Learning Rate = 0.001
```

```
Fold 1
Accuracy = 56.25

Fold 2
Accuracy = 62.5

Fold 3
Accuracy = 75.0

Fold 4
Accuracy = 62.5

Fold 5
Accuracy = 68.75
```

```
print("Learning Rate = {}".format(0.0001))
scores = evaluate_algorithm(
    data.values, n_folds=5, l_rate=0.0001, in_dim=22, hid_dim=[15, 10, 5], out_dim=2
)

```

```
Learning Rate = 0.0001

Fold 1
Accuracy = 62.5

Fold 2
Accuracy = 43.75

Fold 3
Accuracy = 50.0

Fold 4
Accuracy = 37.5

Fold 5
Accuracy = 50.0
```

Iris Dataset

```
iris_data = datasets.load_iris()
df = pd.DataFrame(data=iris_data.data, columns=iris_data.feature_names)
df["class"] = iris_data.target
df.head()

```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	class
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
print("Learning Rate = {}".format(0.01))
scores = evaluate_algorithm(
    df.values, n_folds=5, l_rate=0.01, in_dim=4, hid_dim=[4, 4, 3], out_dim=3
)

```

```
Learning Rate = 0.01

Fold 1
Accuracy = 90.0

Fold 2
Accuracy = 30.0

Fold 3
Accuracy = 20.0
```

```
Fold 4
Accuracy = 23.333333333333332

Fold 5
Accuracy = 26.666666666666668
```

```python
print("Learning Rate = {}".format(0.001))
scores = evaluate_algorithm(
    df.values, n_folds=5, l_rate=0.001, in_dim=4, hid_dim=[4, 4, 3], out_dim=3
)

```

```
Learning Rate = 0.001

Fold 1
Accuracy = 63.33333333333333

Fold 2
Accuracy = 33.33333333333333

Fold 3
Accuracy = 33.33333333333333

Fold 4
Accuracy = 33.33333333333333

Fold 5
Accuracy = 40.0
```

```python
print("Learning Rate = {}".format(0.0001))
scores = evaluate_algorithm(
    df.values, n_folds=5, l_rate=0.0001, in_dim=4, hid_dim=[4, 4, 3], out_dim=3
)

```

```
Learning Rate = 0.0001
```

```
Fold 1
Accuracy = 43.333333333333336

Fold 2
Accuracy = 30.0

Fold 3
Accuracy = 43.333333333333336

Fold 4
Accuracy = 43.333333333333336

Fold 5
Accuracy = 33.33333333333333
```