Structural Machine Learning Models and Their Applications HW.1

Chen, Yi Chieh 8108050001

MISSION
1.Change MLP to regressor,prepare housing dataset
2 Prepare housing dataset
3 Implement the 3 -layer MLP
4 Implement the Xavier initialization
5.Implement the Dropout
6. (Bouns) Implement the n-layer
For MISSION-2 Use data.isnull().sum() find na,and use MinMaxScaler to X. Split data to 0.8 training 0.2 testing

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
data = pd.read_csv('E:/NCHU_PHD/10902sml/housing.data', header=None, sep='\s+')

X=data.iloc[:,:-1]
X[X.columns] = scaler.fit_transform(X[X.columns])
Y=data.iloc[:,-1]

data.isnull().sum()
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state=5)
X_train=X_train.reset_index(drop = True)
Y_train=Y_train.reset_index(drop=True)
```

For MISSION-1,3,6 the code is

```
class DeepLearning:

    def __init__(self, layer_of_units,epochs):
        self.random = np.random.RandomState(2)

        self._n_layers = len(layer_of_units)
        self.epochs=epochs
        parameters = {}
        for i in range(self._n_layers - 1):
            parameters['W{}'.format(i + 1)] = self.random.rand(\
layer_of_units[i + 1], layer_of_units[i])
            parameters['B{}'.format(i + 1)] = self.random.rand(\
layer_of_units[i + 1], 1)
        self._parameters = parameters

    def single_layer_forward_propagation(self, A_previous, W_current, B_current):

        Z_current = np.dot(W_current, A_previous) + B_current
        A_current = Z_current
        return A_current, Z_current
    def forward_propagation(self):

        self._m = self._X_train.shape[0]
        X_train_T = self._X_train.copy().T
        cache = {}
        A_current = X_train_T
        for i in range(self._n_layers - 1):
            A_previous = A_current
            W_current = self._parameters["W{}".format(i + 1)]
            B_current = self._parameters["B{}".format(i + 1)]
            A_current, Z_current = self.single_layer_forward_propagation\
(A_previous, W_current, B_current)
            cache["A{}".format(i)] = A_previous
            cache["Z{}".format(i + 1)] = Z_current
        self._cache = cache
        self._A_current = A_current

    def single_layer_backward_propagation(self, dA_current, W_current,\
 B_current, Z_current, A_previous):

        dZ_current = dA_current
        dW_current = np.dot(dZ_current, A_previous.T) / self._m
        dB_current = np.sum(dZ_current, axis=1, keepdims=True) / self._m
        dA_previous = np.dot(W_current.T, dZ_current)
        return dA_previous, dW_current, dB_current
    def backward_propagation(self):

        gradients = {}
        self.forward_propagation()
        Y_hat = self._A_current.copy()
        Y_train = self._y_train.copy().reshape(1, self._m)
        dA_previous = - 2*(Y_train-Y_hat)
        for i in reversed(range(dl._n_layers - 1)):
            dA_current = dA_previous
            A_previous = self._cache["A{}".format(i)]
            Z_current = self._cache["Z{}".format(i+1)]
            W_current = self._parameters["W{}".format(i+1)]
            B_current = self._parameters["B{}".format(i+1)]
            dA_previous, dW_current, dB_current = \
self.single_layer_backward_propagation\
(dA_current, W_current, B_current, Z_current, A_previous)
            gradients["dW{}".format(i + 1)] = dW_current
            gradients["dB{}".format(i + 1)] = dB_current
        self._gradients = gradients
    def cost_function(self):

        Y_hat = self._A_current.copy()
        self._Y_hat = Y_hat
        Y_train = self._y_train.copy().reshape(1, self._m)
        ce = (Y_train-Y_hat)**2

        return np.sum(ce)

    def gradient_descent(self):
        for i in range(self._n_layers - 1):
            self._parameters["W{}".format(i + 1)] -= self._learning_rate *\
 self._gradients["dW{}".format(i + 1)]
            self._parameters["B{}".format(i + 1)] -= self._learning_rate * \
self._gradients["dB{}".format(i + 1)]
    def fit(self, X_train, y_train, learning_rate=0.0001):
        self._X_train = X_train.copy()
        self._y_train = y_train.copy()
        self._learning_rate = learning_rate
        loss_history = []
        #accuracy_history = []
        n_prints = 10
        print_iter = self.epochs // n_prints
        for i in range(self.epochs):
            self.forward_propagation()
            ce = self.cost_function()
            #accuracy = self.accuracy_score()
            loss_history.append(ce)
            #accuracy_history.append(accuracy)
            self.backward_propagation()
            self.gradient_descent()
            if i % print_iter == 0:
                print("Iteration: {:6} - cost: {:.6f} ".format(i, ce))
        self._loss_history = loss_history
    def predict_proba(self, X_test):

        X_test_T = X_test.copy().T
        A_current = X_test_T
        for i in range(self._n_layers - 1):
            A_previous = A_current
            W_current = self._parameters["W{}".format(i + 1)]
            B_current = self._parameters["B{}".format(i + 1)]
            A_current, Z_current = self.single_layer_forward_propagation\
(A_previous, W_current, B_current)
            self._cache["A{}".format(i)] = A_previous
            self._cache["Z{}".format(i + 1)] = Z_current
        Y_hat_1 = A_current.copy().ravel()
        return Y_hat_1

dl = DeepLearning([13,13,13,1],epochs=10000)
dl.fit(X_train.to_numpy(), Y_train.to_numpy())
```

Input of DeepLearning [13,13,13,1] is that the node of hidden layer,this example is 3-layer.
If want to have anthoer n-layer just key in number of node,and number of keyin is layer count,like [13,13,13,4,1] is 4 layers. The Mission 4 , change code of the init the parameter weight to random uniform distribution.

```
class DeepLearning:

    def __init__(self, layer_of_units,epochs):
        self._n_layers = len(layer_of_units)
        self.epochs=epochs
        parameters = {}
        for i in range(self._n_layers - 1):
            limit = np.sqrt(6/(layer_of_units[i] + layer_of_units[i+1]))
            parameters['W{}'.format(i + 1)] = \
np.random.uniform(-limit,limit,size=(layer_of_units[i + 1],layer_of_units[i]))
            parameters['B{}'.format(i + 1)] = \
np.random.rand(layer_of_units[i + 1], 1)
        self._parameters = parameters
```

The Mission 5 , change code of forward propagation, use dropout in forward part. And dropout function is use numpy binomial distribution when n=1.

```
import numpy as np
def dropout(x, level):

    retain_prob = 1. - level
    random_tensor = np.random.binomial(n=1, p=retain_prob, size=x.shape)
    x *= random_tensor

    return x
```

```
def single_layer_forward_propagation(self, A_previous, W_current, B_current):
        Z_current = np.dot(W_current, A_previous) + B_current
        A_current = Z_current
        A_current = dropout(Z_current,self.level_)
        return A_current, Z_current
    def forward_propagation(self):
        self._m = self._X_train.shape[0]
        X_train_T = self._X_train.copy().T
        cache = {}
        A_current = X_train_T
        for i in range(self._n_layers - 1):
            A_previous = A_current
            W_current = self._parameters["W{}".format(i + 1)]
            B_current = self._parameters["B{}".format(i + 1)]
            A_current, Z_current = self.single_layer_forward_propagation\
(A_previous, W_current, B_current)
            cache["A{}".format(i)] = A_previous
            cache["Z{}".format(i + 1)] = Z_current
        self._cache = cache
        self._A_current = A_current
```

Output of some model
1.epochs = 10000
2. learning rate=0.0001

1.Model of 3-layer

```
dl = DeepLearning([13,13,13,1],0.1,10000)
dl.fit(X_train.to_numpy(), Y_train.to_numpy())
```

Iteration: 0 - cost: 3426147.491789
Iteration: 1000 - cost: 23000.677560
Iteration: 2000 - cost: 16421.698053
Iteration: 3000 - cost: 12065.490690
Iteration: 4000 - cost: 10254.320395
Iteration: 5000 - cost: 9768.831147
Iteration: 6000 - cost: 9567.730410
Iteration: 7000 - cost: 9436.585587
Iteration: 8000 - cost: 9340.518272
Iteration: 9000 - cost: 9269.009068
RMSE of test:4.527

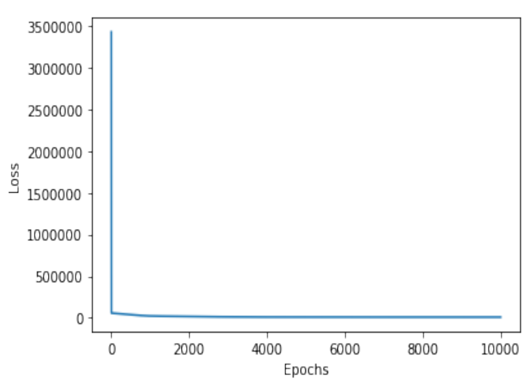

Fig. 1. Model of 3-layer

2.Model of 3-layer Xavier initialization

```
dl = DeepLearning([13,13,13,1],0.1,10000)
dl.fit(X_train.to_numpy(), Y_train.to_numpy())
```

Iteration: 0 - cost: 262876.252177
Iteration: 1000 - cost: 21409.026279
Iteration: 2000 - cost: 15370.741476
Iteration: 3000 - cost: 11994.653002
Iteration: 4000 - cost: 10576.787717
Iteration: 5000 - cost: 9997.744224
Iteration: 6000 - cost: 9706.158751
Iteration: 7000 - cost: 9526.194783
Iteration: 8000 - cost: 9403.117980
Iteration: 9000 - cost: 9315.264556
RMSE of test:4.560

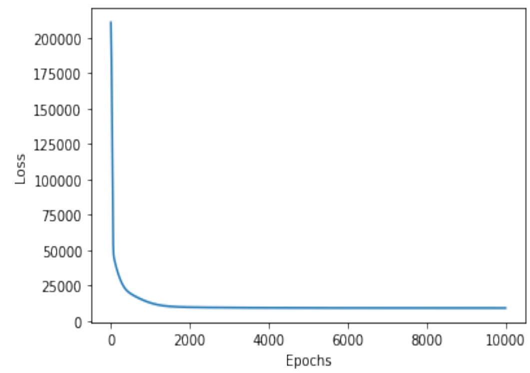

Fig. 2. Model of Xavier

3.Model of 3-layer dropout

```
dl = DeepLearning([13,13,13,1],0.1,10000)
dl.fit(X_train.to_numpy(), Y_train.to_numpy())
```

Iteration: 0 - cost: 1774857.617981
Iteration: 1000 - cost: 65645.537340
Iteration: 2000 - cost: 49379.204275
Iteration: 3000 - cost: 52185.937839
Iteration: 4000 - cost: 58104.616193
Iteration: 5000 - cost: 46140.681442
Iteration: 6000 - cost: 47287.619379
Iteration: 7000 - cost: 44761.640501
Iteration: 8000 - cost: 46760.891343
Iteration: 9000 - cost: 44222.422055
RMSE of test:8.952

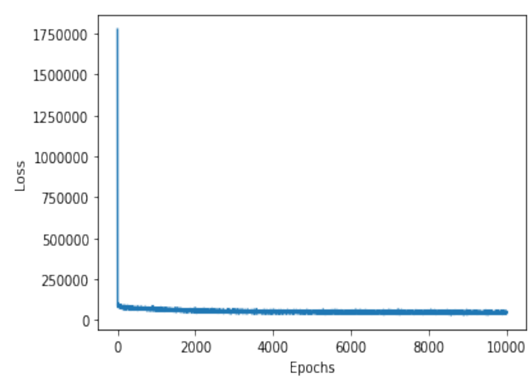

Fig. 3. Model of 3-layer dropout

4.Model of n-layer(n=12)

```
dl = DeepLearning([13,6,4,4,4,4,2,2,2,2,1,1,1],epochs=10000)
dl.fit(X_train.to_numpy(), Y_train.to_numpy())
```

Iteration: 0 - cost: 216963.470798
Iteration: 1000 - cost: 30987.216951
Iteration: 2000 - cost: 20024.868231
Iteration: 3000 - cost: 16734.625865
Iteration: 4000 - cost: 13621.155553
Iteration: 5000 - cost: 11263.205182
Iteration: 6000 - cost: 10313.903858
Iteration: 7000 - cost: 9974.288935
Iteration: 8000 - cost: 9755.395522
Iteration: 9000 - cost: 9582.578959

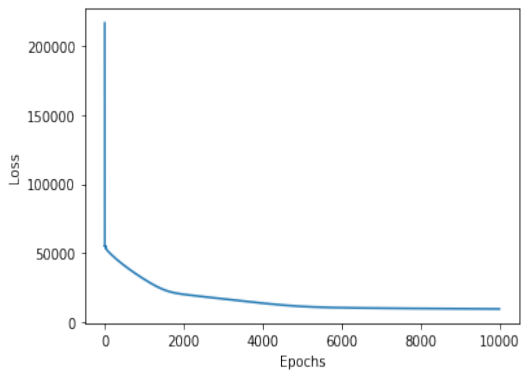

Fig. 4. Model of n-layer(n=12)

RMSE of test:4.55

Finally,see the loss of epochs plot.we can find the 3-layers model convergence fater than other model. The drop out model doesn't have good of convergence and RMSE of testing data,that I think model is not very deep and node numbers of a layer is small. The 3-layers model is better than 12-layers model,that show the deeper model is not necessarily better than ordinary model.

The code of HW1 is in 10902sml _Hw1.ipynb that have more detail.

Reference:https://yaojenkuo.io/ml-newbies/08-deep-learning.html