Homework #4 Answers

Chongye Feng

In [1]:
```
# importing libraries
import numba
import numpy as np
import pandas as pd
from pylab import *
from mpl_toolkits.mplot3d import axes3d
from scipy.optimize import minimize

# setting the random seed
np.random.seed(0)
```

In [2]:
```
def show_correlation(xs,ys):
    plt.figure()
    plt.scatter(xs,ys,s=0.5)
    r = [np.min([np.min(xs),np.min(ys)]),np.max([np.max(xs),np.max(ys)])]
    plt.plot(r,r,'r')
    plt.xlabel("Predictions")
    plt.ylabel("Ground truth")
    corr=np.corrcoef([xs,ys])[1,0]
    print("Correlation coefficient:",corr)
```

In [3]:
```
import time

def timeit(func):
    def wrapper(*args, **kwargs):
        start_time = time.time()
        result = func(*args, **kwargs)
        end_time = time.time()
        elapsed_time = end_time - start_time
        print("Elapsed time: {:.6f} seconds".format(elapsed_time))
        return result
    return wrapper
```

Q1

In [4]:
```
admissions = pd.read_csv('Admission_Predict_Ver1.1.csv')
```

In [5]:
```
admissions.shape
```
Out[5]:
```
(500, 9)
```

In [6]:
```
admissions.head(3)
```
Out[6]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72

a:

In [7]:
```
chance_max = admissions["Chance of Admit "].max()
chance_max
```
Out[7]:
```
0.97
```

In [8]:
```
chance_min = admissions["Chance of Admit "].min()
chance_min
```
Out[8]:
```
0.34
```

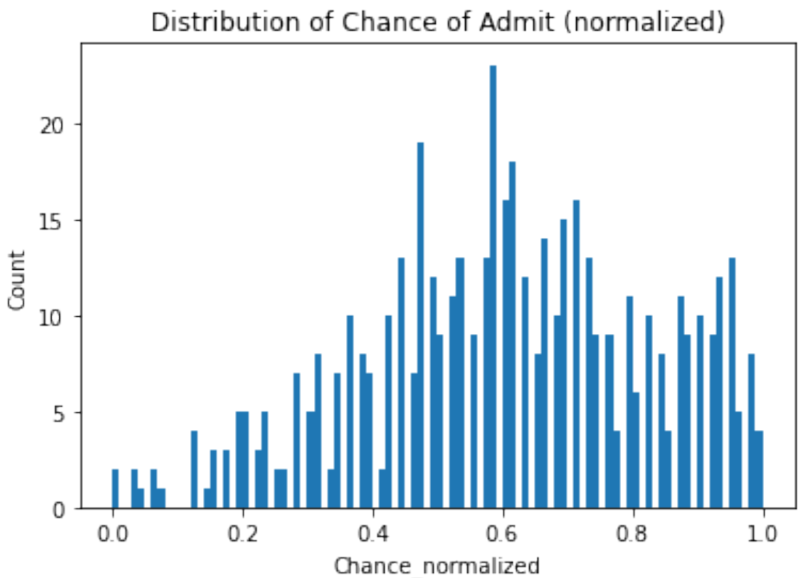
In [9]:
```
# Using the max and min to normalize the 'Chance of Admit '
admissions['Chance'] = (admissions["Chance of Admit "] - chance_min) / (chan
```

In [10]:
```
admissions.head(3)
```
Out[10]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit	Chance
0	1	337	118	4	4.5	4.5	9.65	1	0.92	0.920635
1	2	324	107	4	4.0	4.5	8.87	1	0.76	0.666667
2	3	316	104	3	3.0	3.5	8.00	1	0.72	0.603175

Column 'Chance' was created as the nomalized chance of admit.

In [11]:
```
plt.hist(x=admissions['Chance'], bins=100)
plt.xlabel('Chance_normalized')
plt.ylabel('Count')
plt.title('Distribution of Chance of Admit (normalized)')
plt.show()
```



In the same way, each parameters could be normalized.

In [12]:
```
admissions.columns
```

Out[12]:
```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit ', 'Chance'],
      dtype='object')
```

In [13]:
```
admissions['GRE Score'] = (admissions["GRE Score"] - admissions["GRE Score"]
admissions['TOEFL Score'] = (admissions["TOEFL Score"] - admissions["TOEFL S
admissions['University Rating'] = (admissions["University Rating"] - admissi
admissions['SOP'] = (admissions["SOP"] - admissions["SOP"].min()) / (admissi
admissions['LOR '] = (admissions["LOR "] - admissions["LOR "].min()) / (admi
admissions['CGPA'] = (admissions["CGPA"] - admissions["CGPA"].min()) / (admi
admissions['Research'] = (admissions["Research"] - admissions["Research"].mi
```

In [14]:
```
admissions.head(3)
```

Out[14]:

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit	Chance
0	1	0.94	0.928571	0.75	0.875	0.875	0.913462	1.0	0.92	0.920635
1	2	0.68	0.535714	0.75	0.750	0.875	0.663462	1.0	0.76	0.666667
2	3	0.52	0.428571	0.50	0.500	0.625	0.384615	1.0	0.72	0.603175

b:

In [15]:
```
import random

class simple_perceptron():
    def __init__(self,input_dim,output_dim,learning_rate=0.01,activation=lam

        self.input_dim=input_dim
        self.output_dim=output_dim
        self.activation=activation
        self.activation_grad=activation_grad
        self.lr=learning_rate
        ### initialize parameters ###
        # Weight and bias are between 0 and 0.05
        self.weights= np.random.rand(output_dim,input_dim) / 20
        self.biases= np.random.rand(1,output_dim) / 20

    def predict(self,X):
        if len(X.shape)==1:
            X=X.reshape((-1,1))
        dim=X.shape[1]
        # Check that the dimension of accepted input data is the same as exp
        if not dim==self.input_dim:
            raise Exception("Expected input size %d, accepted %d!"%(self.inp
        ### Calculate logit and activation ###
        self.z = X @ self.weights.T + self.biases           #shape(X.shape[
        self.a = self.activation(self.z)                                  #sha
        return self.a

    def fit(self,X,y):
        # Transform the single-sample data into 2-dimensional, for the conve
        if len(X.shape)==1:
            X=X.reshape((-1,1))
        if len(y.shape)==1:
            y=y.reshape((-1,1))
        self.predict(X)
        errors=(self.a-y)*self.activation_grad(self.z)
        weights_grad=errors.T.dot(X)
        bias_grad=np.sum(errors,axis=0)
        ### Update weights and biases from the gradient ###
        self.weights -= self.lr * weights_grad
        self.biases -= self.lr * bias_grad


    def train_on_epoch(self,X,y,batch_size=32):
        # Every time select batch_size samples from the training set, until
        order=list(range(X.shape[0]))
        random.shuffle(order)
        n=0
        while n<math.ceil(len(order)/batch_size)-1: # Parts that can fill on
            self.fit(X[order[n*batch_size:(n+1)*batch_size]],y[order[n*batch
```

```
            n+=1
        # Parts that cannot fill one batch
        self.fit(X[order[n*batch_size:]],y[order[n*batch_size:]])

    def evaluate(self,X,y):
         # Transform the single-sample data into 2-dimensional
        if len(X.shape)==1:
            X=X.reshape((1,-1))
        if len(y.shape)==1:
            y=y.reshape((1,-1))
        ### means square error ###
        return np.mean(np.square(self.predict(X) - y))

    def get_weights(self):
        return (self.weights,self.biases)

    def set_weights(self,weights):
        self.weights=weights[0]
        self.biases=weights[1]
```

C:

In [16]:
```
from sklearn.model_selection import train_test_split,KFold


def Kfold(k,Xs,ys,epochs,learning_rate=0.0001,draw_curve=True):
    # The total number of examples for training the network
    total_num=len(Xs)

    # Built in K-fold function in Sci-Kit Learn
    kf=KFold(n_splits=k,shuffle=True)
    # record error for each model
    train_error_all=[]
    test_error_all=[]

    for train_selector,test_selector in kf.split(range(total_num)):
        ### Decide training examples and testing examples for this fold ###
        train_Xs = Xs[train_selector]
        test_Xs = Xs[test_selector]
        train_ys = ys[train_selector]
        test_ys = ys[test_selector]


        val_array=[]
        # Split training examples further into training and validation
        train_in,val_in,train_real,val_real=train_test_split(train_Xs,train_

        ### Establish the model for simple perceptron here ###
        model=simple_perceptron(Xs.shape[1], ys.shape[1], learning_rate)

        # Save the lowest weights, so that we can recover the best model
        weights = model.get_weights()
```

```
        lowest_val_err = np.inf
        for _ in range(epochs):
            # Train model on a number of epochs, and test performance in the
            model.train_on_epoch(train_in,train_real)
            val_err = model.evaluate(val_in,val_real)
            val_array.append(val_err)
            if val_err < lowest_val_err:
                lowest_val_err = val_err
                weights = model.get_weights()

        # The final number of epochs is when the minimum error in validation
        final_epochs= epochs + 1
        print("Number of epochs with lowest validation:",final_epochs)
        # Recover the model weight
        model.set_weights(weights)

        # Report result for this fold
        train_error=model.evaluate(train_Xs, train_ys)
        train_error_all.append(train_error)
        test_error=model.evaluate(test_Xs, test_ys)
        test_error_all.append(test_error)
        print("Train error:",train_error)
        print("Test error:",test_error)

        if draw_curve:
            plt.figure()
            plt.plot(np.arange(len(val_array))+1,val_array,label='Validation
            plt.xlabel('Epochs')
            plt.ylabel('Loss')
            plt.legend()

    print("Final results:")
    print("Training error:%f+-%f"%(np.average(train_error_all),np.std(train_
    print("Testing error:%f+-%f"%(np.average(test_error_all),np.std(test_err

    # return the last model
    return model
```

In [17]:
```
admissions.columns
```

Out[17]:
```
Index(['Serial No.', 'GRE Score', 'TOEFL Score', 'University Rating', 'SOP',
       'LOR ', 'CGPA', 'Research', 'Chance of Admit ', 'Chance'],
      dtype='object')
```

In [18]:
```
X = admissions.loc[:,['GRE Score', 'TOEFL Score', 'University Rating', 'SOP'
X.shape
```

Out[18]:
```
(500, 7)
```

In [19]:
```
X_no_GRE = admissions.loc[:,['TOEFL Score', 'University Rating', 'SOP', 'LOR
X_no_GRE.shape
```

Out[19]:
```
(500, 6)
```

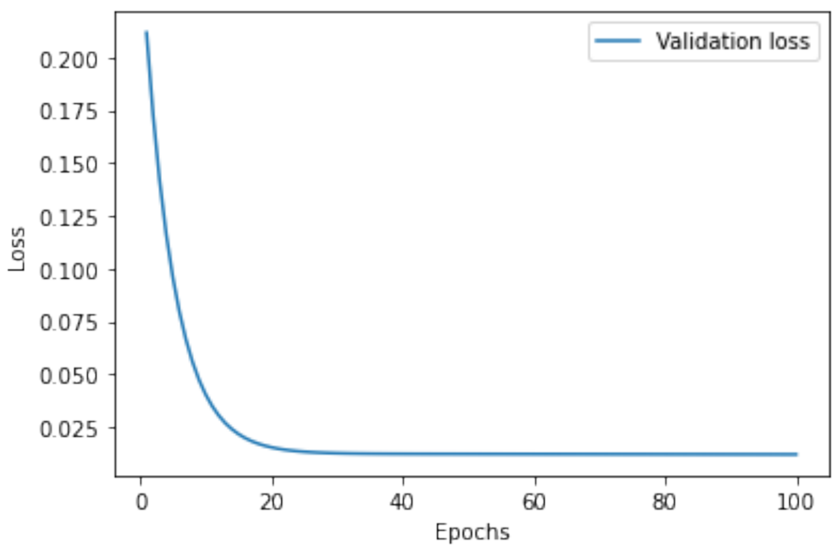
In [20]:
```
y = admissions[['Chance']].to_numpy()
y.shape
```

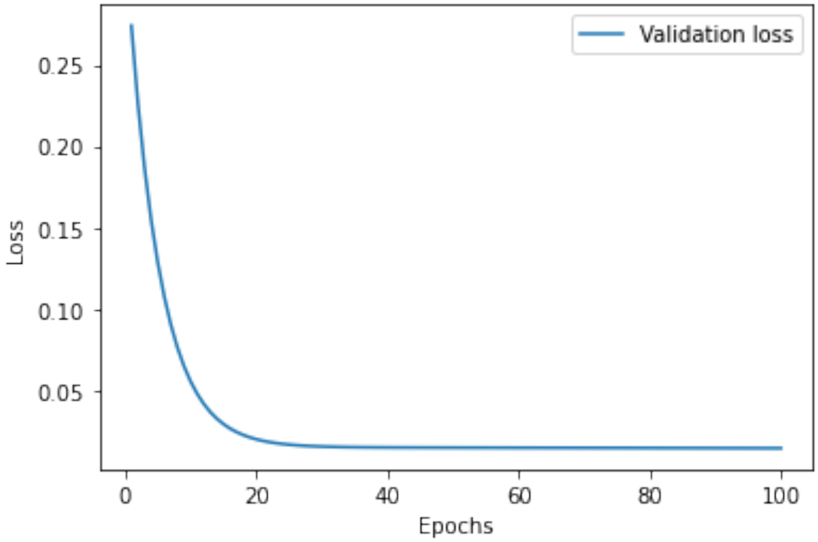
Out[20]:
```
(500, 1)
```

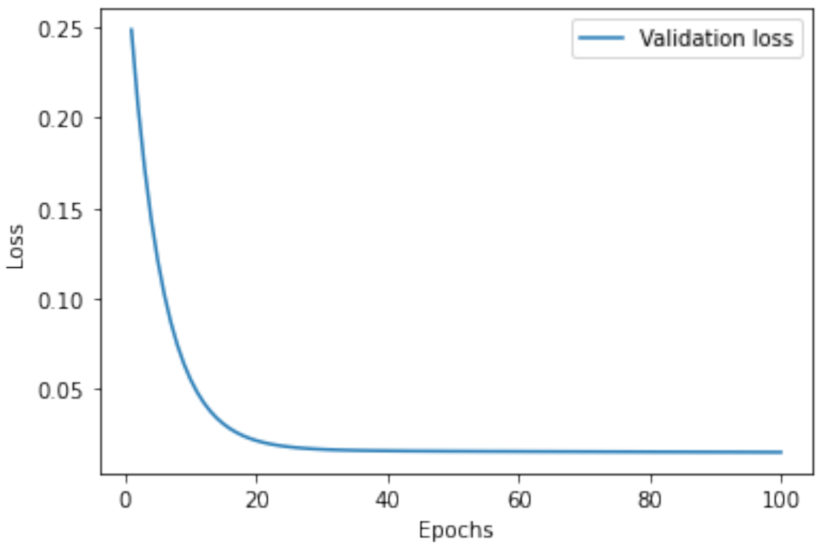
In [21]:
```
Kfold(k=5, Xs=X, ys=y, epochs=100, learning_rate=0.0001, draw_curve=True)
```

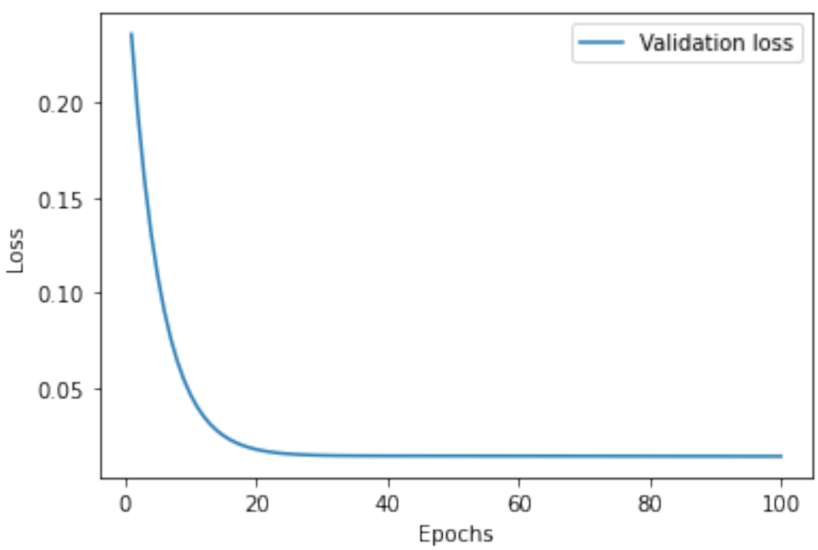
```
Number of epochs with lowest validation: 101
Train error: 0.011964759612224487
Test error: 0.012290975549225364
Number of epochs with lowest validation: 101
Train error: 0.012729968049578218
Test error: 0.012748203360374329
Number of epochs with lowest validation: 101
Train error: 0.013359712570870819
Test error: 0.012041614906070641
Number of epochs with lowest validation: 101
Train error: 0.012694540968836286
Test error: 0.012592473576736842
Number of epochs with lowest validation: 101
Train error: 0.012888242565825792
Test error: 0.015045621051065 93
Final results:
Training error:0.012727+-0.000449
Testing error:0.012944+-0.001079
```

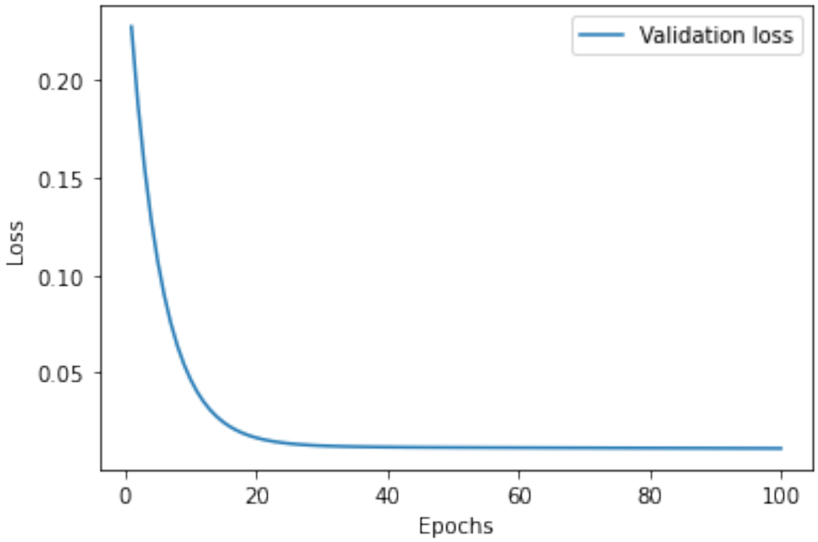
Out[21]:
```
<__main__.simple_perceptron at 0x7fe008969fd0>
```




Validation loss
0.25
0.20
0.15
0.10
0.05
Loss
0
20
40
60
80
100
Epochs


Validation loss
0.25
0.20
0.15
0.10
0.05
Loss
0
20
40
60
80
100
Epochs

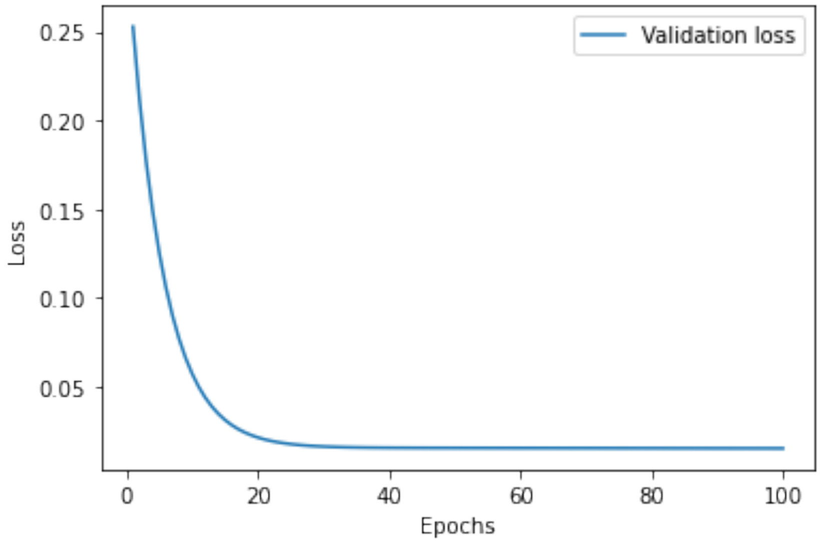

Validation loss
0.20
0.15
0.10
0.05
Loss
0
20
40
60
80
100
Epochs

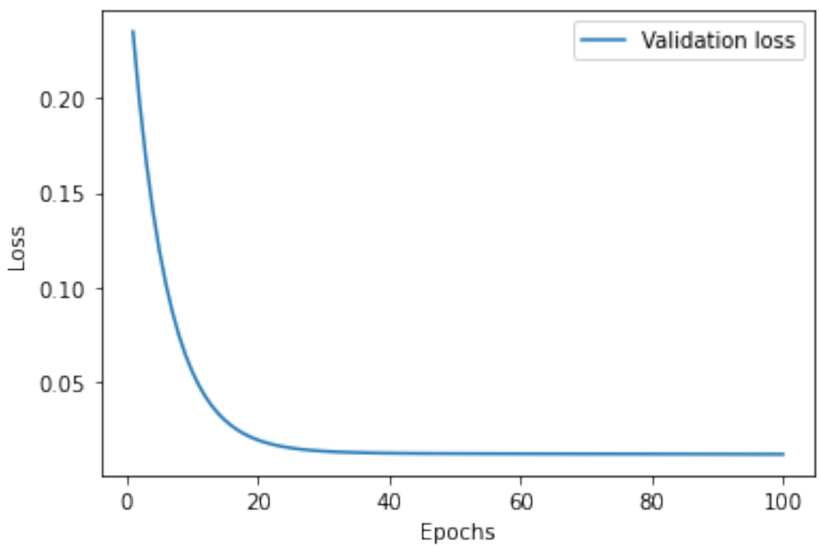


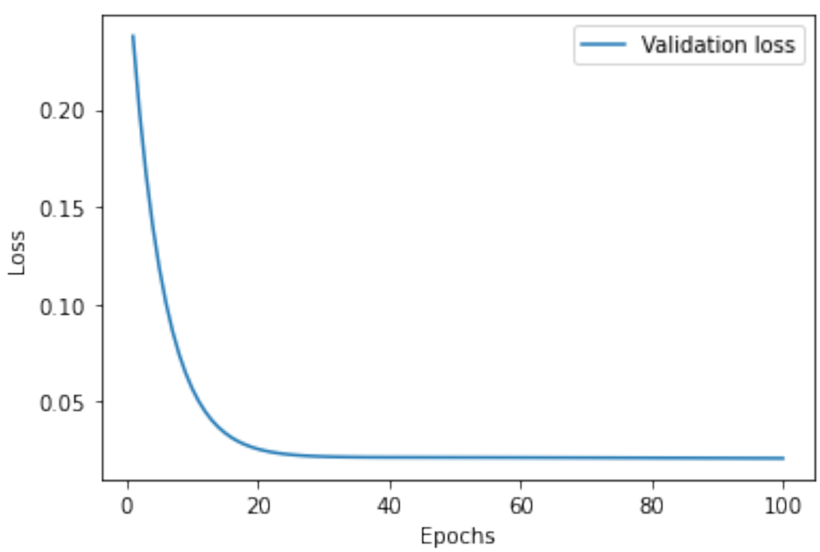
In [22]:
```
Kfold(k=5, Xs=X_no_GRE, ys=y, epochs=100, learning_rate=0.0001, draw_curve=T
```

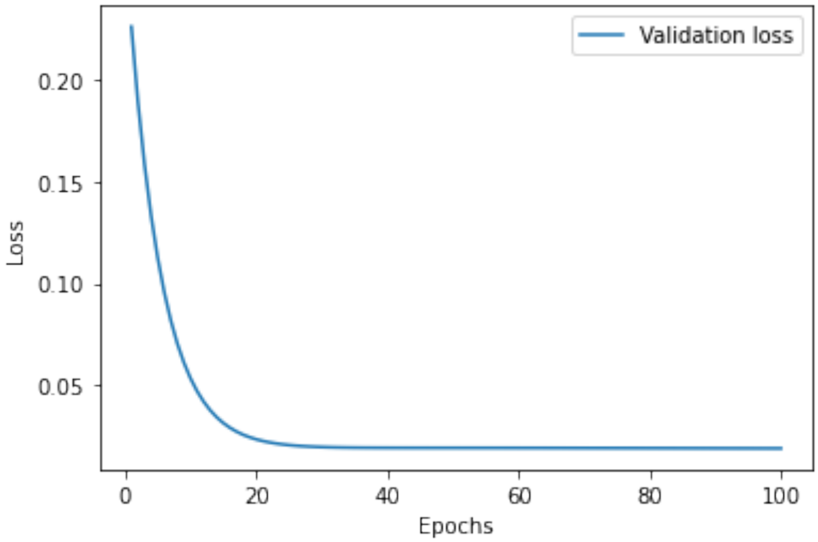
```
Number of epochs with lowest validation: 101
Train error: 0.01286871985367504
Test error: 0.01556288653660874
Number of epochs with lowest validation: 101
Train error: 0.014257661147348707
Test error: 0.014051898618966677
Number of epochs with lowest validation: 101
Train error: 0.014561702175504919
Test error: 0.011150138023280947
Number of epochs with lowest validation: 101
Train error: 0.014305633510406462
Test error: 0.01144074509548585
Number of epochs with lowest validation: 101
Train error: 0.01286280638754163
Test error: 0.01682012305716432
Final results:
Training error:0.013771+-0.000747
Testing error:0.013805+-0.002231
```

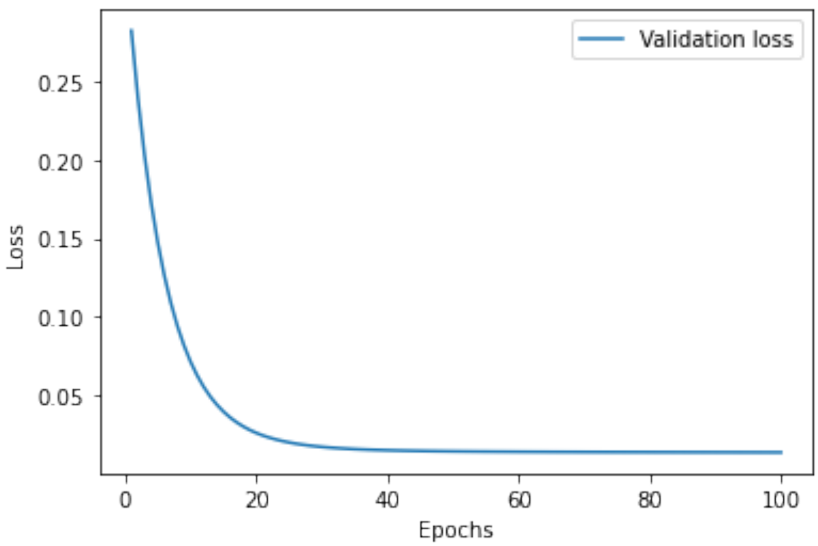
Out[22]:
```
<__main__.simple_perceptron at 0x7fe00884ee20>
```


Validation loss
0.25
0.20
0.15
0.10
0.05
Loss
0
20
40
60
80
100
Epochs


Validation loss
0.20
0.15
0.10
0.05
Loss
0
20
40
60
80
100
Epochs


Validation loss
0.20
0.15
0.10
0.05
Loss
0
20
40
60
80
100
Epochs





It shows that the 7 features are indicators of getting into graduate school. With all 7 features, the final test error would be 0.0126. With only 6 features (excluding the GRE score), the final test error would be 0.0139. Both errors are very small (around 1%), and it appears that the GRE score is not a very important feature.

Q2

In [23]:
```
titanic = pd.read_csv('titantic.csv')
```

In [24]:
```
titanic.head(3)
```

Out[24]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250

In [25]:
```
titanic.columns
```
Out[25]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

In [26]:
```
titanic.shape
```
Out[26]:
```
(891, 12)
```

In [27]:
```
titanic_filtered = titanic.dropna(subset=['Age', 'Embarked'])
titanic_filtered.shape
```
Out[27]:
```
(712, 12)
```

I think the column 'Name' and 'Ticket' could be dropped since they are some identificational information and hard to be in a factor of survival.

In [28]:
```
titanic_filtered = titanic_filtered.drop(['PassengerId', 'Name', 'Ticket', '
titanic_filtered.reindex()
```

Out[28]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22.0	1	0	7.2500	S
1	1	1	female	38.0	1	0	71.2833	C
2	1	3	female	26.0	0	0	7.9250	S
3	1	1	female	35.0	1	0	53.1000	S
4	0	3	male	35.0	0	0	8.0500	S
...	...	...	...	...	...	...	...	...
885	0	3	female	39.0	0	5	29.1250	Q
886	0	2	male	27.0	0	0	13.0000	S
887	1	1	female	19.0	0	0	30.0000	S
889	1	1	male	26.0	0	0	30.0000	C
890	0	3	male	32.0	0	0	7.7500	Q

712 rows × 8 columns

Normalizing the 'Age', 'sibsp', 'parch' and 'Fare' column

In [29]:

```
titanic_filtered['Age'] = (titanic_filtered["Age"] - titanic_filtered["Age"]
titanic_filtered['Fare'] = (titanic_filtered["Fare"] - titanic_filtered["Far
titanic_filtered['SibSp'] = (titanic_filtered["SibSp"] - titanic_filtered["S
titanic_filtered['Parch'] = (titanic_filtered["Parch"] - titanic_filtered["P

titanic_filtered.head(3)
```

Out[29]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	0.271174	0.2	0.0	0.014151	S
1	1	1	female	0.472229	0.2	0.0	0.139136	C
2	1	3	female	0.321438	0.0	0.0	0.015469	S

In [30]:

```
categorical_feats = titanic_filtered[['Pclass', 'Sex','Embarked']]
continuous_feats = titanic_filtered[['Age', 'Fare', 'SibSp', 'Parch']]
continuous_feats
```

Out[30]:

	Age	Fare	SibSp	Parch
0	0.271174	0.014151	0.2	0.000000
1	0.472229	0.139136	0.2	0.000000
2	0.321438	0.015469	0.0	0.000000
3	0.434531	0.103644	0.2	0.000000
4	0.434531	0.015713	0.0	0.000000
...	...	...	...	...
885	0.484795	0.056848	0.0	0.833333
886	0.334004	0.025374	0.0	0.000000
887	0.233476	0.058556	0.0	0.000000
889	0.321438	0.058556	0.0	0.000000
890	0.396833	0.015127	0.0	0.000000

712 rows × 4 columns

Encoding categorical features

In [31]:
```
from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder()
encoder.fit(categorical_feats)
input_cate_feats = encoder.transform(categorical_feats).toarray()
input_cate_feats.shape
```

Out[31]:
```
(712, 8)
```

['Pclass' : 3, 'Sex': 2,'Embarked': 3]

In [32]:
```
input_cate_feats
```

Out[32]:
```
array([[0., 0., 1., ..., 0., 0., 1.],
       [1., 0., 0., ..., 1., 0., 0.],
       [0., 0., 1., ..., 0., 0., 1.],
       ...,
       [1., 0., 0., ..., 0., 0., 1.],
       [1., 0., 0., ..., 1., 0., 0.],
       [0., 0., 1., ..., 0., 1., 0.]])
```

In [33]:
```
feats = np.hstack((continuous_feats, input_cate_feats))
```

In [34]:
```
feats.shape
```

Out[34]:
```
(712, 12)
```

In [35]:
```
output_a = titanic_filtered[['Survived']]

encoder = OneHotEncoder()
encoder.fit(output_a)
output = encoder.transform(output_a).toarray()
output.shape
```

Out[35]:
```
(712, 2)
```

b:

In [36]:
```
Kfold(k=5, Xs=feats, ys=output, epochs=100, learning_rate=0.0001, draw_curve
```

```
Number of epochs with lowest validation: 101
Train error: 0.15639389143800286
Test error: 0.13760341965470163
Number of epochs with lowest validation: 101
Train error: 0.15394013089211792
Test error: 0.1479434449584229
Number of epochs with lowest validation: 101
Train error: 0.1473159510389259
Test error: 0.17231430002039122
Number of epochs with lowest validation: 101
Train error: 0.1509599593567362
Test error: 0.1567362809040447
Number of epochs with lowest validation: 101
Train error: 0.15231544503656125
Test error: 0.15798592528227834
Final results:
Training error:0.152185+-0.003034
Testing error:0.154517+-0.011515
```

Out[36]:
```
<__main__.simple_perceptron at 0x7fdff8015c40>
```

In [37]:
```
feats_no_age = feats[:,1:]
Kfold(k=5, Xs=feats_no_age, ys=output, epochs=100, learning_rate=0.0001, dra
```

```
Number of epochs with lowest validation: 101
Train error: 0.15366265618206562
Test error: 0.15362643620714242
Number of epochs with lowest validation: 101
Train error: 0.16301370393532372
Test error: 0.12453380364309222
Number of epochs with lowest validation: 101
Train error: 0.15106189166563508
Test error: 0.16053961284334156
Number of epochs with lowest validation: 101
Train error: 0.15180567216638827
Test error: 0.16486675786862662
Number of epochs with lowest validation: 101
Train error: 0.14630482409124151
Test error: 0.177514516297542
Final results:
Training error:0.153170+-0.005488
Testing error:0.156216+-0.017650
```

Out[37]: `<__main__.simple_perceptron at 0x7fe039d9c4c0>`

In [38]:
```
feats_no_fare = feats[:,[0,2,3,4,5,6,7,8,9,10,11]]
Kfold(k=5, Xs=feats_no_age, ys=output, epochs=100, learning_rate=0.0001, dra
```

```
Number of epochs with lowest validation: 101
Train error: 0.1550357105664611
Test error: 0.1586620573531606
Number of epochs with lowest validation: 101
Train error: 0.15446448010738312
Test error: 0.1568271044329718
Number of epochs with lowest validation: 101
Train error: 0.1526357526468373
Test error: 0.1575384917662574
Number of epochs with lowest validation: 101
Train error: 0.14990743305584525
Test error: 0.16680730162907423
Number of epochs with lowest validation: 101
Train error: 0.1578633967278791
Test error: 0.14114884924322726
Final results:
Training error:0.153981+-0.002639
Testing error:0.156197+-0.008334
```

Out[38]: `<__main__.simple_perceptron at 0x7fe03c46e4f0>`

In [39]:
```
feats_no_sib = feats[:,[0,1,3,4,5,6,7,8,9,10,11]]
Kfold(k=5, Xs=feats_no_sib, ys=output, epochs=100, learning_rate=0.0001, dra
```

```
Number of epochs with lowest validation: 101
Train error: 0.15895540506191808
Test error: 0.13518258971292962
Number of epochs with lowest validation: 101
Train error: 0.14777392700970282
Test error: 0.17316093431504326
Number of epochs with lowest validation: 101
Train error: 0.15199009135656524
Test error: 0.15539990092590397
Number of epochs with lowest validation: 101
Train error: 0.15515104061939583
Test error: 0.14956824425230936
Number of epochs with lowest validation: 101
Train error: 0.15131195679650883
Test error: 0.16512091526176512
Final results:
Training error:0.153036+-0.003775
Testing error:0.155687+-0.013056
```

Out[39]:
```
<__main__.simple_perceptron at 0x7fe008863a90>
```

In [40]:
```python
feats_no_par = feats[:,[0,1,2,4,5,6,7,8,9,10,11]]
Kfold(k=5, Xs=feats_no_par, ys=output, epochs=100, learning_rate=0.0001, dra
```

```
Number of epochs with lowest validation: 101
Train error: 0.1574600671900209
Test error: 0.14149171970843058
Number of epochs with lowest validation: 101
Train error: 0.14934124584811695
Test error: 0.16695012714300903
Number of epochs with lowest validation: 101
Train error: 0.15684613804751807
Test error: 0.13817514878691142
Number of epochs with lowest validation: 101
Train error: 0.14349310180604033
Test error: 0.1850583971383505
Number of epochs with lowest validation: 101
Train error: 0.15754874171594482
Test error: 0.14664555683629704
Final results:
Training error:0.152938+-0.005641
Testing error:0.155664+-0.017773
```

Out[40]:
```
<__main__.simple_perceptron at 0x7fe03c529fd0>
```

In [41]:
```python
feats_no_pclass = feats[:,[0,1,2,3,7,8,9,10,11]]
Kfold(k=5, Xs=feats_no_pclass, ys=output, epochs=100, learning_rate=0.0001,
```

```
Number of epochs with lowest validation: 101
Train error: 0.17282546967731433
Test error: 0.15856500673708143
Number of epochs with lowest validation: 101
Train error: 0.1690179943419059
Test error: 0.17392103519516233
Number of epochs with lowest validation: 101
Train error: 0.15710100861279264
Test error: 0.209203866796951
Number of epochs with lowest validation: 101
Train error: 0.16897753434323834
Test error: 0.16865214183412106
Number of epochs with lowest validation: 101
Train error: 0.17573093955418911
Test error: 0.14585019048955097
Final results:
Training error:0.168731+-0.006343
Testing error:0.171238+-0.021262
```

Out[41]:
```
<__main__.simple_perceptron at 0x7fe03c551970>
```

In [42]:
```
feats_no_sex = feats[:,[0,1,2,3,4,5,6,9,10,11]]
Kfold(k=5, Xs=feats_no_sex, ys=output, epochs=100, learning_rate=0.0001, dra
```

```
Number of epochs with lowest validation: 101
Train error: 0.20516010441762433
Test error: 0.2121284801761996
Number of epochs with lowest validation: 101
Train error: 0.20126305393905491
Test error: 0.23774453307050472
Number of epochs with lowest validation: 101
Train error: 0.20967053225376725
Test error: 0.19705903964764004
Number of epochs with lowest validation: 101
Train error: 0.21018915006045955
Test error: 0.196179495459286
Number of epochs with lowest validation: 101
Train error: 0.2072075690440787
Test error: 0.20629829390470647
Final results:
Training error:0.206698+-0.003262
Testing error:0.209882+-0.015142
```

Out[42]:
```
<__main__.simple_perceptron at 0x7fe03ccdca30>
```

In [43]:
```
feats_no_embark = feats[:,[0,1,2,3,4,5,6,7,8]]
Kfold(k=5, Xs=feats_no_embark, ys=output, epochs=100, learning_rate=0.0001,
```

```
Number of epochs with lowest validation: 101
Train error: 0.15459599735556387
Test error: 0.15502141125794014
Number of epochs with lowest validation: 101
Train error: 0.1460565444514795
Test error: 0.1854207903247474
Number of epochs with lowest validation: 101
Train error: 0.1514817982095674
Test error: 0.16379409762982913
Number of epochs with lowest validation: 101
Train error: 0.15483960227748667
Test error: 0.14994257748267245
Number of epochs with lowest validation: 101
Train error: 0.16203529514701245
Test error: 0.12107377074145843
Final results:
Training error:0.153802+-0.005194
Testing error:0.155051+-0.020877
```

Out[43]:

```
<__main__.simple_perceptron at 0x7fe03c524190>
```

It has been shown that features such as 'PClass' and 'Sex' are two important factors that can increase the chances of survival. The inclusion of 'PClass' increases the final testing error from 0.155 to 0.169, while 'Sex' increases it to 0.210. 'PClass' suggests that passengers with higher class might be closer to the deck and have a higher chance of getting onto the lifeboats. 'Sex' might also play a role, as people may be more likely to leave their chance of survival to women, or conversely, men may be stronger and have an easier time surviving.

Q3

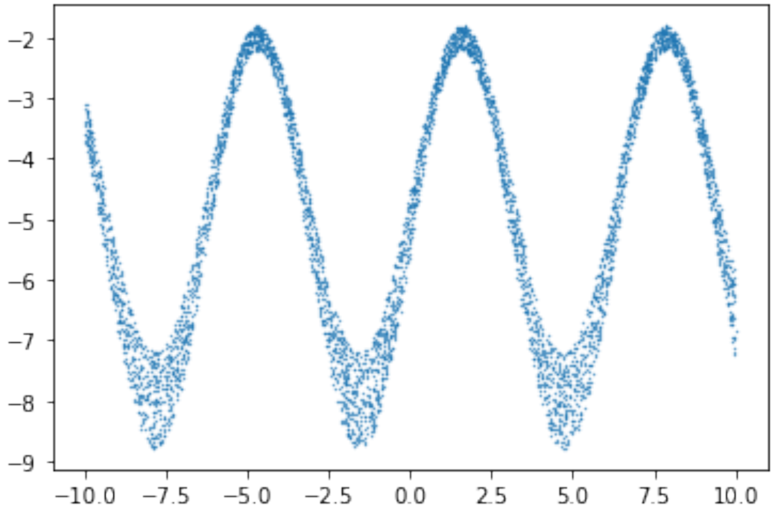
In [44]:

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def generate_X(number):
    xs=(np.random.random(number)*2-1)*10
    return xs

def generate_data(number,stochascity=0.05):
    xs=generate_X(number)
    fs=3*np.sin(xs)-5
    stochastic_ratio=(np.random.random(number)*2-1)*stochascity+1
    return xs,fs*stochastic_ratio
```

In [45]:

```
x,y=generate_data(5000,0.1)
plt.scatter(x,y,s=0.1)
```

Out[45]: <matplotlib.collections.PathCollection at 0x7fe03c4a51f0>



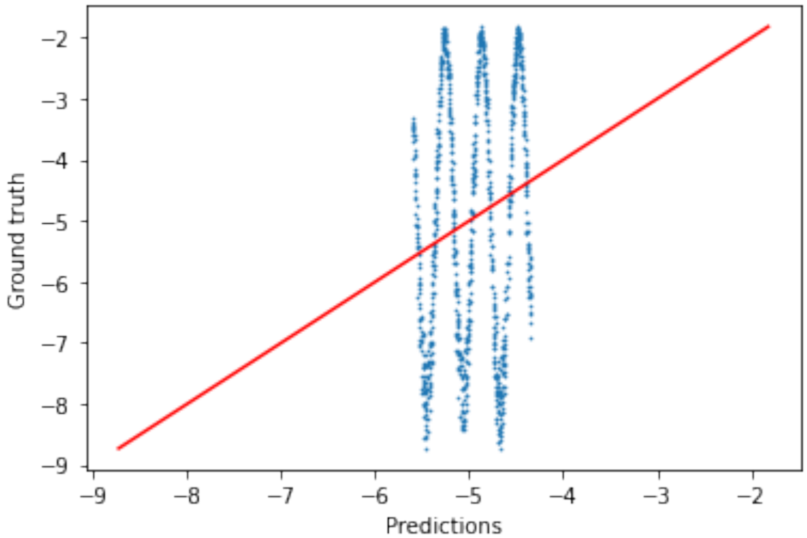
In [46]:
```
x = x.reshape(-1,1)
y = y.reshape(-1,1)
```

In [47]:
```
model = Kfold(k=5, Xs=x, ys=y, epochs=100, learning_rate=0.0001, draw_curve=
```

```
Number of epochs with lowest validation: 101
Train error: 4.265830540993415
Test error: 4.184408996898839
Number of epochs with lowest validation: 101
Train error: 4.250041181179471
Test error: 4.273170793956684
Number of epochs with lowest validation: 101
Train error: 4.248489488541902
Test error: 4.224598800435798
Number of epochs with lowest validation: 101
Train error: 4.2673885934845615
Test error: 4.148415040409795
Number of epochs with lowest validation: 101
Train error: 4.194691543668467
Test error: 4.438961570627624
Final results:
Training error:4.245288+-0.026470
Testing error:4.253911+-0.101424
```

In [48]:
```
x_test, y_test = generate_data(1000,0.1)
predict = model.predict(x_test).flatten()
ground = y_test.flatten()
show_correlation(predict, ground)
```

```
Correlation coefficient: 0.18481776973829822
```



The model does not agree with the test data. The Correlation coefficient is pretty bad, around 0.

b:

In [49]:
```
from sklearn.neural_network import MLPRegressor

@timeit
def KFold_NN(k,Xs,ys,hidden_layers,epochs=1000,lr=0.001):
    # The total number of examples for training the network
    total_num=len(Xs)

    # Built in K-fold function in Sci-Kit Learn
    kf=KFold(n_splits=k,shuffle=True)
    train_error_all=[]
    test_error_all=[]
    for train_selector,test_selector in kf.split(range(total_num)):
        # Decide training examples and testing examples for this fold
        train_Xs = Xs[train_selector]
        test_Xs = Xs[test_selector]
        train_ys = ys[train_selector].reshape(-1) #reshape to get rid of the
        test_ys = ys[test_selector].reshape(-1)

        # Establish the model here
        model = MLPRegressor(max_iter=epochs, activation='tanh', early_stopp
                             validation_fraction=0.25, learning_rate='consta
                             hidden_layer_sizes=hidden_layers).fit(train_Xs,

        ### Report result for this fold ##
        train_error= np.mean(np.square(model.predict(train_Xs) - train_ys))
        train_error_all.append(train_error)
        test_error = np.mean(np.square(model.predict(test_Xs) - test_ys))
        test_error_all.append(test_error)
        print("Train error:",train_error)
        print("Test error:",test_error)

    print("Final results:")
    print("Training error:%f+-%f"%(np.average(train_error_all),np.std(train_
    print("Testing error:%f+-%f"%(np.average(test_error_all),np.std(test_err

    # return the last model
    return model
```

In [50]:
```
Xs, ys = generate_data(5000,0.1)
Xs = Xs.reshape(-1,1)
ys = ys.reshape(-1,1)

model = KFold_NN(5, Xs, ys, 8)
```

```
Train error: 4.060684328216927
Test error: 4.104209063047867
```

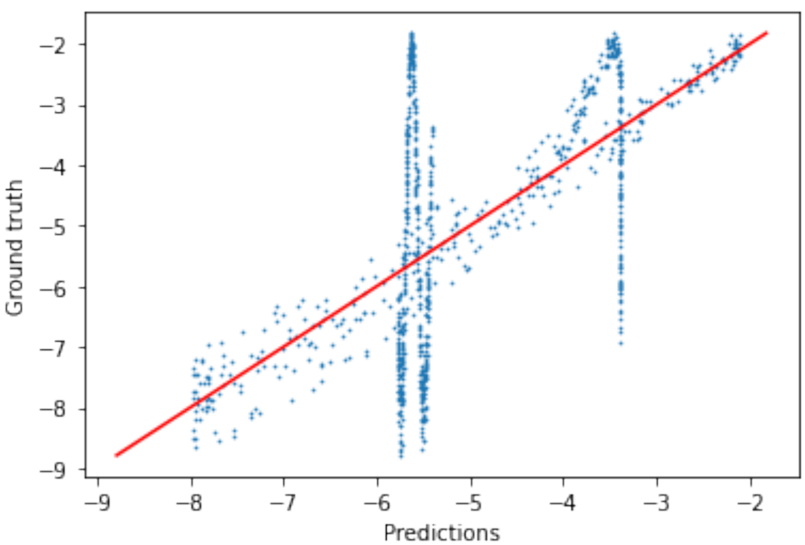
```
/Users/chongyefeng/opt/anaconda3/envs/msse-python/lib/python3.9/site-package
s/sklearn/neural_network/_multilayer_perceptron.py:684: ConvergenceWarning:
Stochastic Optimizer: Maximum iterations (1000) reached and the optimization
hasn't converged yet.
  warnings.warn(
```

```
Train error: 0.5773190186251262
Test error: 0.5903504415889401
/Users/chongyefeng/opt/anaconda3/envs/msse-python/lib/python3.9/site-package
s/sklearn/neural_network/_multilayer_perceptron.py:684: ConvergenceWarning:
Stochastic Optimizer: Maximum iterations (1000) reached and the optimization
hasn't converged yet.
  warnings.warn(
Train error: 1.3162858780687738
Test error: 1.3215828621247991
Train error: 4.210142009468109
Test error: 4.234891323991862
Train error: 2.3616652130881284
Test error: 2.3049011144371137
Final results:
Training error:2.505219+-1.447564
Testing error:2.511187+-1.459877
Elapsed time: 7.297183 seconds
```

In [51]:

```
x_test, y_test = generate_data(1000,0.1)

x_test = x_test.reshape(-1,1)
y_test = y_test.reshape(-1,1)

predict = model.predict(x_test).flatten()
ground = y_test.flatten()
show_correlation(predict, ground)
```

```
Correlation coefficient: 0.6486201728551388
```



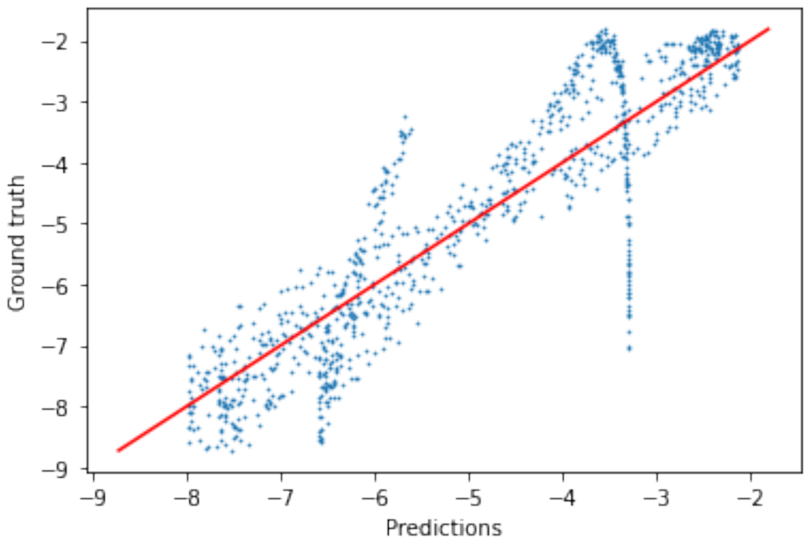
Yes, the ANN with 8 hidden layers performs a better prediction of the sin() function than the one-layer simple perceptron. The correlation coefficient increased from 0.19 to 0.66.

C:

In [52]:
```
Xs, ys = generate_data(5000,0.1)
Xs = Xs.reshape(-1,1)
ys = ys.reshape(-1,1)

model = KFold_NN(5, Xs, ys, 16)

x_test, y_test = generate_data(1000,0.1)

x_test = x_test.reshape(-1,1)
y_test = y_test.reshape(-1,1)

predict = model.predict(x_test).flatten()
ground = y_test.flatten()
show_correlation(predict, ground)
```

```
/Users/chongyefeng/opt/anaconda3/envs/msse-python/lib/python3.9/site-package
s/sklearn/neural_network/_multilayer_perceptron.py:684: ConvergenceWarning:
Stochastic Optimizer: Maximum iterations (1000) reached and the optimization
hasn't converged yet.
  warnings.warn(
Train error: 0.7750601071178586
Test error: 0.6728888371766616
/Users/chongyefeng/opt/anaconda3/envs/msse-python/lib/python3.9/site-package
s/sklearn/neural_network/_multilayer_perceptron.py:684: ConvergenceWarning:
Stochastic Optimizer: Maximum iterations (1000) reached and the optimization
hasn't converged yet.
  warnings.warn(
Train error: 0.6700583120266426
Test error: 0.7177745700691864
/Users/chongyefeng/opt/anaconda3/envs/msse-python/lib/python3.9/site-package
s/sklearn/neural_network/_multilayer_perceptron.py:684: ConvergenceWarning:
Stochastic Optimizer: Maximum iterations (1000) reached and the optimization
hasn't converged yet.
  warnings.warn(
Train error: 0.5178638460607986
Test error: 0.5528629274035308
/Users/chongyefeng/opt/anaconda3/envs/msse-python/lib/python3.9/site-package
s/sklearn/neural_network/_multilayer_perceptron.py:684: ConvergenceWarning:
Stochastic Optimizer: Maximum iterations (1000) reached and the optimization
hasn't converged yet.
  warnings.warn(
Train error: 0.5187916311810994
Test error: 0.48343128828774384
Train error: 0.7824181173048907
Test error: 0.8401253385729671
Final results:
Training error:0.652838+-0.116801
Testing error:0.653417+-0.125210
Elapsed time: 13.839354 seconds
Correlation coefficient: 0.9067403160744317
```

```
/Users/chongyefeng/opt/anaconda3/envs/msse-python/lib/python3.9/site-package
s/sklearn/neural_network/_multilayer_perceptron.py:684: ConvergenceWarning:
Stochastic Optimizer: Maximum iterations (1000) reached and the optimization
hasn't converged yet.
  warnings.warn(
```



Doubling the number of hidden layers from 8 to 16 increases the correlation coefficient, which becomes closer to 1 at 0.9. Including more hidden layers in the ANN improves its performance. However, it should be noted that the additional hidden layers come with a significant increase in computation time. Therefore, it is not advisable to add too many hidden layers to the ANN.