Homework 4 - Berkeley STAT 157

**Name: Mandi Zhao; SID 3032880866; teammates: Christine Giang, Salim Damerdji, Jobe Cowen

In [203]:
```
%matplotlib inline
import d2l
from mxnet import autograd, gluon, init, nd
from mxnet.gluon import data as gdata, loss as gloss, nn, utils
import numpy as np
import pandas as pd
```

loading data

In [204]:
```
utils.download('https://github.com/d2l-ai/d2l-en/raw/master/data/kaggle_
house_pred_train.csv')
utils.download('https://github.com/d2l-ai/d2l-en/raw/master/data/kaggle_
house_pred_test.csv')
train_data = pd.read_csv('kaggle_house_pred_train.csv')
test_data = pd.read_csv('kaggle_house_pred_test.csv')
```

In [205]:
```
all_features = pd.concat((train_data.iloc[:, 1:-1], test_data.iloc[:, 1
:]))
```

In [206]:
```
numeric_features = all_features.dtypes[all_features.dtypes != 'object'].
index
all_features[numeric_features] = all_features[numeric_features].apply(
    lambda x: (x - x.mean()) / (x.std()))
# after standardizing the data all means vanish, hence we can set missin
g values to 0
all_features = all_features.fillna(0)
```

one - hit

In [207]:
```
# Dummy_na=True refers to a missing value being a legal eigenvalue, and
 creates an indicative feature for it.
all_features = pd.get_dummies(all_features, dummy_na=True, sparse = True
)
all_features.shape
```

Out[207]: (2919, 354)

In [208]:
```
n_train = train_data.shape[0]
train_features = nd.array(all_features[:n_train].values)
test_features = nd.array(all_features[n_train:].values)
train_labels = nd.array(train_data.SalePrice.values).reshape((-1, 1))
```

In [209]:
```
loss = gloss.HuberLoss()
#loss = gloss.L2Loss()
def get_net():
    net = nn.Sequential()
    net.add(nn.Dense(1))
    net.initialize()
    return net
```

In [210]:
```
def log_rmse(net, features, labels):
    # To further stabilize the value when the logarithm is taken, set th
e value less than 1 as 1.
    clipped_preds = nd.clip(net(features), 1, float('inf'))
    rmse = nd.sqrt(2 * loss(clipped_preds.log(), labels.log()).mean())
    return rmse.asscalar()
```

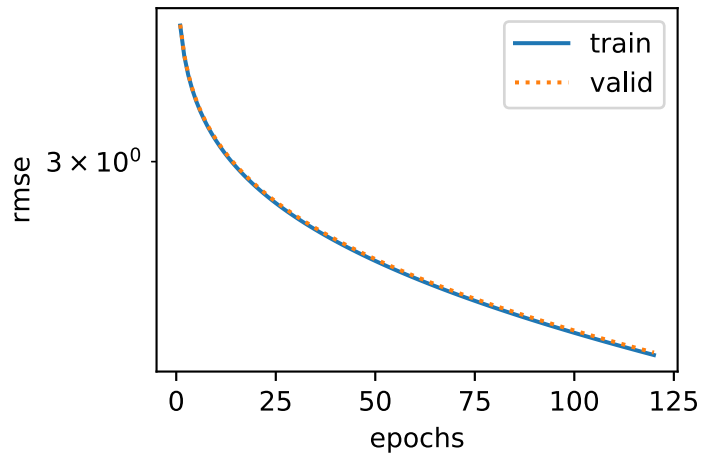
In [211]:
```
def train(net, train_features, train_labels, test_features, test_labels,
          num_epochs, learning_rate, weight_decay, batch_size):
    train_ls, test_ls = [], []
    train_iter = gdata.DataLoader(gdata.ArrayDataset(
        train_features, train_labels), batch_size, shuffle=True)
    # The Adam optimization algorithm is used here.
    trainer = gluon.Trainer(net.collect_params(), 'adam', {
        'learning_rate': learning_rate, 'wd': weight_decay})
    for epoch in range(num_epochs):
        for X, y in train_iter:
            with autograd.record():
                l = loss(net(X), y)
            l.backward()
            trainer.step(batch_size)
        train_ls.append(log_rmse(net, train_features, train_labels))
        if test_labels is not None:
            test_ls.append(log_rmse(net, test_features, test_labels))
    return train_ls, test_ls
```

k-fold

In [212]:
```
def get_k_fold_data(k, i, X, y):
    assert k > 1
    fold_size = X.shape[0] // k
    X_train, y_train = None, None
    for j in range(k):
        idx = slice(j * fold_size, (j + 1) * fold_size)
        X_part, y_part = X[idx, :], y[idx]
        if j == i:
            X_valid, y_valid = X_part, y_part
        elif X_train is None:
            X_train, y_train = X_part, y_part
        else:
            X_train = nd.concat(X_train, X_part, dim=0)
            y_train = nd.concat(y_train, y_part, dim=0)
    return X_train, y_train, X_valid, y_valid
```

In [213]:
```
def k_fold(k, X_train, y_train, num_epochs,
           learning_rate, weight_decay, batch_size):
    train_l_sum, valid_l_sum = 0, 0
    for i in range(k):
        data = get_k_fold_data(k, i, X_train, y_train)
        net = get_net()
        train_ls, valid_ls = train(net, *data, num_epochs, learning_rate
,
                                   weight_decay, batch_size)
        train_l_sum += train_ls[-1]
        valid_l_sum += valid_ls[-1]
        if i == 0:
            d2l.semilogy(range(1, num_epochs + 1), train_ls, 'epochs',
'rmse',
                         range(1, num_epochs + 1), valid_ls,
                         ['train', 'valid'])
        print('fold %d, train rmse: %f, valid rmse: %f' % (
            i, train_ls[-1], valid_ls[-1]))
    return train_l_sum / k, valid_l_sum / k
```

In [234]:
```
k, num_epochs, lr, weight_decay, batch_size = 5, 120, 0.1, 0, 64
train_l, valid_l = k_fold(k, train_features, train_labels, num_epochs, l
r,
                          weight_decay, batch_size)
print('%d-fold validation: avg train rmse: %f, avg valid rmse: %f'
      % (k, train_l, valid_l))
```




```
fold 0, train rmse: 2.164711, valid rmse: 2.173725
fold 1, train rmse: 2.165454, valid rmse: 2.170654
fold 2, train rmse: 2.164118, valid rmse: 2.174638
fold 3, train rmse: 2.169757, valid rmse: 2.153431
fold 4, train rmse: 2.165042, valid rmse: 2.172216
5-fold validation: avg train rmse: 2.165816, avg valid rmse: 2.168933
```

In [215]:
```
def train_and_pred(train_features, test_feature, train_labels, test_data
,
                   num_epochs, lr, weight_decay, batch_size):
    train_ls, _ = train(net, train_features, train_labels, None, None,
                        num_epochs, lr, weight_decay, batch_size)
    d2l.semilogy(range(1, num_epochs + 1), train_ls, 'epochs', 'rmse')
    print('train rmse %f' % train_ls[-1])
    # apply the network to the test set
    preds = net(test_features).asnumpy()
    # reformat it for export to Kaggle
    test_data['SalePrice'] = pd.Series(preds.reshape(1, -1)[0])
    submission = pd.concat([test_data['Id'], test_data['SalePrice']], ax
is=1)
    submission.to_csv('submission.csv', index=False)
    return submission
```

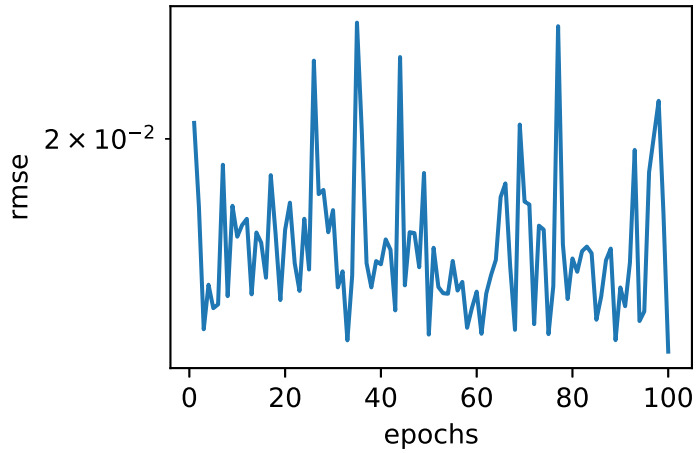
In [216]:
```
net = nn.Sequential()
net.add(nn.Dense(354, activation = "relu"))
net.add(nn.Dense(708, activation = "relu"))
net.add(nn.Dense(708, activation = "relu"))
net.add(nn.Dense(1))
net.initialize(init.Normal())
net
```

Out[216]:
```
Sequential(
  (0): Dense(None -> 354, Activation(relu))
  (1): Dense(None -> 708, Activation(relu))
  (2): Dense(None -> 708, Activation(relu))
  (3): Dense(None -> 1, linear)
)
```

In [217]:
```
test_pred = nd.array(subm.iloc[:,1]).asnumpy()
```

In [236]:
```
# test
batch_size = 64
num_epochs = 100
subm = train_and_pred(train_features, test_features, train_labels, test_
data,
                num_epochs, 0.1, weight_decay, batch_size)

subm
```




```
train rmse 0.012747
```

Out[236]:

	Id	SalePrice
0	1461	122721.242188
1	1462	166077.390625
2	1463	219726.687500
3	1464	201855.062500
4	1465	190429.390625
5	1466	181431.859375
6	1467	187720.359375
7	1468	168002.546875
8	1469	182461.046875
9	1470	149264.281250
10	1471	200611.468750
11	1472	117022.859375
12	1473	116876.632812
13	1474	150152.500000
14	1475	99173.359375
15	1476	405347.687500
16	1477	267398.500000
17	1478	274207.437500
18	1479	293760.656250
19	1480	516097.656250
20	1481	322091.312500
21	1482	209921.140625
22	1483	175623.093750
23	1484	167556.593750
24	1485	202856.515625
25	1486	200060.093750
26	1487	373066.875000
27	1488	162581.046875
28	1489	245785.375000
29	1490	265283.375000
...	...	...
1429	2890	91050.625000

	Id	SalePrice
1430	2891	132587.078125
1431	2892	23718.611328
1432	2893	60210.058594
1433	2894	46456.480469
1434	2895	366831.343750
1435	2896	298202.125000
1436	2897	204337.390625
1437	2898	143569.109375
1438	2899	203370.593750
1439	2900	163586.562500
1440	2901	205963.234375
1441	2902	192685.046875
1442	2903	335999.843750
1443	2904	335003.656250
1444	2905	69098.234375
1445	2906	208835.156250
1446	2907	107975.648438
1447	2908	144215.750000
1448	2909	144209.203125
1449	2910	71099.609375
1450	2911	69270.171875
1451	2912	163125.484375
1452	2913	87846.164062
1453	2914	66372.421875
1454	2915	80480.515625
1455	2916	69699.484375
1456	2917	180510.625000
1457	2918	115695.945312
1458	2919	223100.609375

1459 rows × 2 columns

In [228]:
```
test_pred[0:5]
```

Out[228]:
```
array([156647.33, 156647.33, 156647.33, 156647.33, 156647.33],
      dtype=float32)
```

In [189]:
```
#subm.iloc[:,1]
```

In [33]:
```
nd.array([3, 4]).log()
```

Out[33]:
```
[1.0986123 1.3862944]
<NDArray 2 @cpu(0)>
```