7/10
Great work in general!
Try more feature engineering, more hyperparameter tuning, a shallower neural network (if train set is small), and ensemble models!

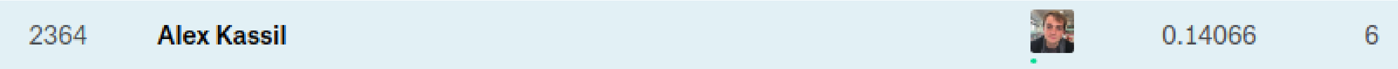
Homework 4 - Berkeley STAT 157

Your name: Alex Kassil 3032646190, Derek Topper 26861675, Catherine Cang 3032723839, Inna Chernomorets 3033363907

Handout 2/12/2019, due 2/19/2019 by 4pm in Git by committing to your repository.

In this homework, we will build a model based real house sale data from a Kaggle competition (https://www.kaggle.com/c/house-prices-advanced-regression-techniques). This notebook contains codes to download the dataset, build and train a baseline model, and save the results in the submission format. Your jobs are

1. Developing a better model to reduce the prediction error. You can find some hints on the last section.
2. Submitting your results into Kaggle and take a sceenshot of your score. Then replace the following image URL with your screenshot.



2364 **Alex Kassil** 0.14066 6

We have two suggestions for this homework:

1. Start as earlier as possible. Though we will cover this notebook on Thursday's lecture, tuning hyper-parameters takes time, and Kaggle limits #submissions per day.
2. Work with your project teammates. It's a good opportunity to get familiar with each other.

Your scores will depend your positions on Kaggle's Leaderboard. We will award the top-3 teams/individuals 500 AWS credits.

Accessing and Reading Data Sets

The competition data is separated into training and test sets. Each record includes the property values of the house and attributes such as street type, year of construction, roof type, basement condition. The data includes multiple datatypes, including integers (year of construction), discrete labels (roof type), floating point numbers, etc.; Some data is missing and is thus labeled 'na'. The price of each house, namely the label, is only included in the training data set (it's a competition after all). The 'Data' tab on the competition tab has links to download the data.

We will read and process the data using `pandas`, an efficient data analysis toolkit (http://pandas.pydata.org/pandas-docs/stable/). Make sure you have `pandas` installed for the experiments in this section.

In [1]:

```
# If pandas is not installed, please uncomment the following line:
!pip install d2l


%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

utils.download('https://github.com/d2l-ai/d2l-en/raw/master/data/kagg
le_house_pred_train.csv')
utils.download('https://github.com/d2l-ai/d2l-en/raw/master/data/kagg
le_house_pred_test.csv')
train_data = pd.read_csv('kaggle_house_pred_train.csv')
test_data = pd.read_csv('kaggle_house_pred_test.csv')
```

```
Requirement already satisfied: d2l in /home/alex/anaconda3/lib/python
3.5/site-packages (0.8.7)
Requirement already satisfied: numpy in /home/alex/anaconda3/lib/pyth
on3.5/site-packages (from d2l) (1.14.5)
Requirement already satisfied: matplotlib in /home/alex/anaconda3/li
b/python3.5/site-packages (from d2l) (3.0.0)
Requirement already satisfied: jupyter in /home/alex/anaconda3/lib/py
thon3.5/site-packages (from d2l) (1.0.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /home/alex/anacon
da3/lib/python3.5/site-packages (from matplotlib->d2l) (1.0.1)
Requirement already satisfied: python-dateutil>=2.1 in /home/alex/ana
conda3/lib/python3.5/site-packages (from matplotlib->d2l) (2.5.3)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.
0.1 in /home/alex/anaconda3/lib/python3.5/site-packages (from matplot
lib->d2l) (2.1.4)
Requirement already satisfied: cycler>=0.10 in /home/alex/anaconda3/l
ib/python3.5/site-packages (from matplotlib->d2l) (0.10.0)
Requirement already satisfied: setuptools in /home/alex/anaconda3/li
b/python3.5/site-packages/setuptools-23.0.0-py3.5.egg (from kiwisolve
r>=1.0.1->matplotlib->d2l) (23.0.0)
Requirement already satisfied: six>=1.5 in /home/alex/anaconda3/lib/p
ython3.5/site-packages (from python-dateutil>=2.1->matplotlib->d2l)
(1.10.0)
You are using pip version 19.0.1, however version 19.0.2 is availabl
e.
You should consider upgrading via the 'pip install --upgrade pip' com
mand.
```

We downloaded the data into the current directory. To load the two CSV (Comma Separated Values) files containing training and test data respectively we use Pandas.

In [2]:

```
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')
```

The training data set includes 1,460 examples, 80 features, and 1 label., the test data contains 1,459 examples and 80 features.

In [3]:

```
print(train_data.shape)
print(test_data.shape)
```

```
(1460, 81)
(1459, 80)
```

Let’s take a look at the first 4 and last 2 features as well as the label (SalePrice) from the first 4 examples:

In [4]:

```
train_data.iloc[0:4, [0, 1, 2, 3, -3, -2, -1]]
```

Out[4]:

	Id	MSSubClass	MSZoning	LotFrontage	SaleType	SaleCondition	SalePrice
0	1	60	RL	65.0	WD	Normal	208500
1	2	20	RL	80.0	WD	Normal	181500
2	3	60	RL	68.0	WD	Normal	223500
3	4	70	RL	60.0	WD	Abnorml	140000

We can see that in each example, the first feature is the ID. This helps the model identify each training example. While this is convenient, it doesn't carry any information for prediction purposes. Hence we remove it from the dataset before feeding the data into the network.

In [5]:

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

Data Preprocessing

As stated above, we have a wide variety of datatypes. Before we feed it into a deep network we need to perform some amount of processing. Let's start with the numerical features. We begin by replacing missing values with the mean. This is a reasonable strategy if features are missing at random. To adjust them to a common scale we rescale them to zero mean and unit variance. This is accomplished as follows:

$$x \leftarrow \frac{x - \mu}{\sigma}$$

To check that this transforms x to data with zero mean and unit variance simply calculate $\mathbf{E}[(x - \mu)/\sigma] = (\mu - \mu)/\sigma = 0$. To check the variance we use $\mathbf{E}[(x - \mu)^2] = \sigma^2$ and thus the transformed variable has unit variance. The reason for 'normalizing' the data is that it brings all features to the same order of magnitude. After all, we do not know *a priori* which features are likely to be relevant. Hence it makes sense to treat them equally.

```
In [6]: 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 missing values to 0
        all_features = all_features.fillna(0)
```

Next we deal with discrete values. This includes variables such as 'MSZoning'. We replace them by a one-hot encoding in the same manner as how we transformed multiclass classification data into a vector of 0 and 1. For instance, 'MSZoning' assumes the values 'RL' and 'RM'. They map into vectors $(1, 0)$ and $(0, 1)$ respectively. Pandas does this automatically for us.

```
In [7]: # 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)
        all_features.shape

Out[7]: (2919, 354)
```

You can see that this conversion increases the number of features from 79 to 331. Finally, via the `values` attribute we can extract the NumPy format from the Pandas dataframe and convert it into MXNet's native representation - NDArray for training.

```
In [8]: 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))
```

Training

To get started we train a linear model with squared loss. This will obviously not lead to a competition winning submission but it provides a sanity check to see whether there's meaningful information in the data. It also amounts to a minimum baseline of how well we should expect any 'fancy' model to work.

```
In [9]: loss = gloss.L2Loss()

        def get_net():
            net = nn.Sequential()

            net.add(nn.Dense(1024, activation='relu'))
            net.add(nn.BatchNorm())
            net.add(nn.Dropout(.5))

            net.add(nn.Dense(1))
            net.add(nn.BatchNorm())
            net.add(nn.Dropout(.5))

            net.add(nn.Dense(128, activation='relu'))
            net.add(nn.BatchNorm())
            net.add(nn.Dropout(.5))
            net.add(nn.Dense(64, activation='relu'))
            net.add(nn.BatchNorm())
            net.add(nn.Dropout(.5))
            net.add(nn.Dense(1))
            net.initialize()
            return net
```

too many layers may get overfitted

House prices, like shares, are relative. That is, we probably care more about the relative error $\frac{y-\hat{y}}{y}$ than about the absolute error. For instance, getting a house price wrong by USD 100,000 is terrible in Rural Ohio, where the value of the house is USD 125,000. On the other hand, if we err by this amount in Los Altos Hills, California, we can be proud of the accuracy of our model (the median house price there exceeds 4 million).

One way to address this problem is to measure the discrepancy in the logarithm of the price estimates. In fact, this is also the error that is being used to measure the quality in this competition. After all, a small value δ of $\log y - \log \hat{y}$ translates into $e^{-\delta} \leq \frac{\hat{y}}{y} \leq e^{\delta}$. This leads to the following loss function:

$$L = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \left(\log y_i - \log \hat{y}_i\right)^2}$$

In [10]:

```
def log_rmse(net, features, labels):
    # To further stabilize the value when the logarithm is taken, set
the 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()
```

Unlike in the previous sections, the following training functions use the Adam optimization algorithm. Compared to the previously used mini-batch stochastic gradient descent, the Adam optimization algorithm is relatively less sensitive to learning rates. This will be covered in further detail later on when we discuss the details on Optimization Algorithms (../chapter_optimization/index.md) in a separate chapter.

In [11]:

```
def train(net, train_features, train_labels, test_features, test_labe
ls,
           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 Cross-Validation

The k-fold cross-validation was introduced in the section where we discussed how to deal with “Model Selection, Underfitting and Overfitting" (underfit-overfit.md). We will put this to good use to select the model design and to adjust the hyperparameters. We first need a function that returns the i-th fold of the data in a k-fold cros-validation procedure. It proceeds by slicing out the i-th segment as validation data and returning the rest as training data. Note - this is not the most efficient way of handling data and we would use something much smarter if the amount of data was considerably larger. But this would obscure the function of the code considerably and we thus omit it.

In [12]:

```
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
```

The training and verification error averages are returned when we train k times in the k-fold cross-validation.

In [18]:

```
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_r
ate,
                                   weight_decay, batch_size)
        train_l_sum += train_ls[-1]
        valid_l_sum += valid_ls[-1]
        if True:
            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 [19]:

```
from sklearn.cross_decomposition import CCA
import matplotlib.pyplot as plt
from sklearn.preprocessing import normalize
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

!pip install xgboost
import xgboost

def check(X_train, y_train):
    xgb = xgboost.XGBRegressor(n_estimators=100, learning_rate=0.08,
gamma=0, subsample=0.75,
                               colsample_bytree=1, max_depth=7)

    X_train, X_test, y_train, y_test = train_test_split(X_train, y_tr
ain)

    xgb.fit(X_train, y_train)
    pred = xgb.predict(X_test)
    print('xgb:', r2_score(y_test, pred))

    return xgb

def preprocess(dat, test=False, obj_features=None, num_features=None
):

    data = dat.copy()

    if not test:
        data = data[data['SalePrice'] < data['SalePrice'].quantile(.9
5)]
        data = data[data['1stFlrSF'] < data['1stFlrSF'].quantile(.95
)]
        data = data[data['LotArea'] < data['LotArea'].quantile(.95)]

    data = data[data.dtypes[data.dtypes != 'object'].index]


    if obj_features is not None:
        if not test:
            data = pd.concat([data, train_data.iloc[data.index][obj_f
eatures]], axis=1)
        else:
            data = pd.concat([data, test_data.iloc[data.index][obj_fe
atures]], axis=1)

    data['Bath'] = data['FullBath'] + (0.5 * data['HalfBath'])
    data['BsmtBath'] = data['BsmtFullBath'] + (0.5 * data['BsmtHalfBa
th'])

    data['TotalFlrSF'] = data['1stFlrSF'] + (0.75 * data['2ndFlrSF'])
    data = data.drop(['1stFlrSF', '2ndFlrSF'], axis=1)

    if num_features is not None:
        data = data.drop(num_features, axis=1)
    # id for test data
    ids = None

    if test:
        ids = data['Id']

    data = data.drop(['FullBath', 'HalfBath', 'BsmtFullBath', 'BsmtHalfBath', 'Id', 'MiscVal'], axis=1)

    return data, ids

def get_net():
    net = nn.Sequential()
#     net.add(nn.Dense(80, activation='relu'))
#     net.add(nn.Dropout(.5))
#     net.add(nn.BatchNorm())
    net.add(nn.Dense(40, activation='relu'))
    net.add(nn.Dropout(.5))
    net.add(nn.BatchNorm())
    net.add(nn.Dense(10, activation='relu'))
    net.add(nn.Dropout(.5))
    net.add(nn.BatchNorm())
    net.add(nn.Dense(1))
    net.initialize()
    return net

train_data = pd.read_csv('kaggle_house_pred_train.csv')
test_data = pd.read_csv('kaggle_house_pred_test.csv')

obj_features = ['SaleCondition', 'MSZoning', 'BldgType', 'Neighborhood', 'LotConfig']
num_features = ['PoolArea']

data, _ = preprocess(train_data, obj_features=obj_features, num_features=num_features)
test, ids = preprocess(test_data, test=True, obj_features=obj_features, num_features=num_features)

X_train, y_train = data.drop('SalePrice', axis=1), data['SalePrice']

# Hot-encode
X_train = pd.get_dummies(X_train, dummy_na=True)
X_test = pd.get_dummies(test, dummy_na=True)


# Normalize
X_train = X_train.apply(lambda x: (x - x.mean()) / x.std()).fillna(0)
X_test = X_test.apply(lambda x: (x - x.mean()) / x.std()).fillna(0)

# check xgb accuracy for testing improvement of preprocessing
xgb = check(X_train, y_train)
```

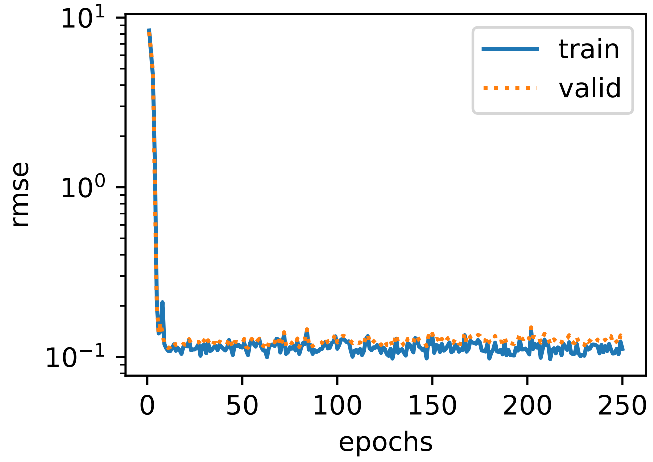
```
Requirement already satisfied: xgboost in /home/alex/anaconda3/lib/py
thon3.5/site-packages (0.81)
Requirement already satisfied: numpy in /home/alex/anaconda3/lib/pyth
on3.5/site-packages (from xgboost) (1.14.5)
Requirement already satisfied: scipy in /home/alex/anaconda3/lib/pyth
on3.5/site-packages (from xgboost) (0.19.1)
You are using pip version 19.0.1, however version 19.0.2 is availabl
e.
You should consider upgrading via the 'pip install --upgrade pip' com
mand.
xgb: 0.8829017872537941
```

Model Selection

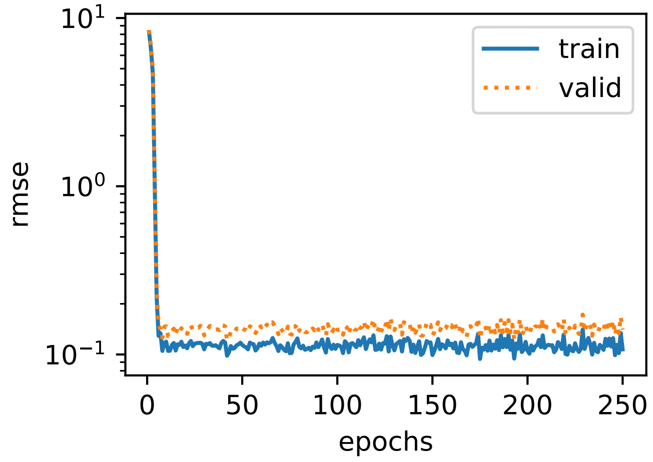
We pick a rather un-tuned set of hyperparameters and leave it up to the reader to improve the model considerably. Finding a good choice can take quite some time, depending on how many things one wants to optimize over. Within reason the k-fold cross-validation approach is resilient against multiple testing. However, if we were to try out an unreasonably large number of options it might fail since we might just get lucky on the validation split with a particular set of hyperparameters.

In [29]:

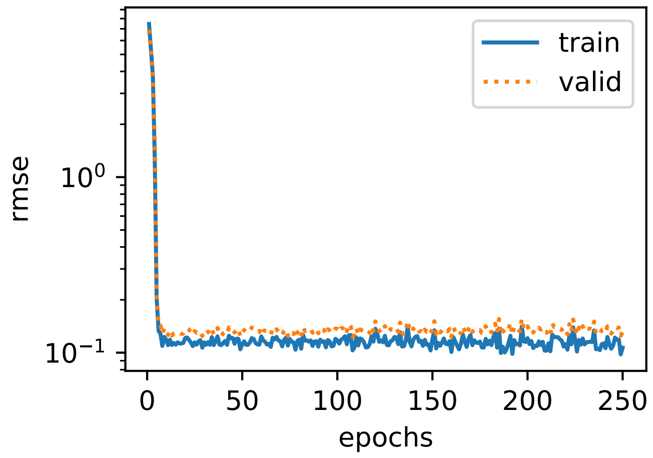
```
k, num_epochs, lr, weight_decay, batch_size = 5, 250, 5, 0, 128
train_l, valid_l = k_fold(k, nd.array(X_train), nd.array(y_train), num_epochs, lr,
                          weight_decay, batch_size)
print('%d-fold validation: avg train rmse: %f, avg valid rmse: %f'
      % (k, train_l, valid_l))



#k, num_epochs, lr, weight_decay, batch_size = 5, 350, 5, 0.99, 128
#params = {'k': k, 'num_epochs':num_epochs, 'lr':lr, 'weight_decay':weight_decay, 'batch_size':batch_size}
#net, train_l, valid_l = k_fold(k, nd.array(X_train), nd.array(y_train), num_epochs, lr,
#                          weight_decay, batch_size)

#print('%d-fold validation: avg train rmse: %f, avg valid rmse: %f'
#      % (k, train_l, valid_l))



#tbl = pd.DataFrame(data={'score':xgb.feature_importances_, 'features':list(X_train)})

# feature importance
#tbl.sort_values('score', ascending=False)
```



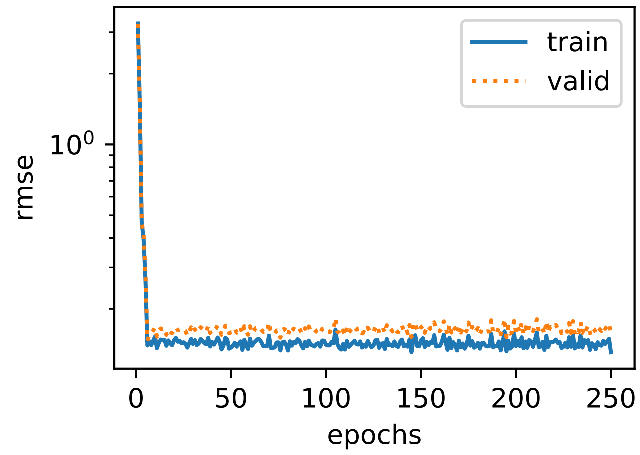
```
fold 0, train rmse: 0.111552, valid rmse: 0.123549
```



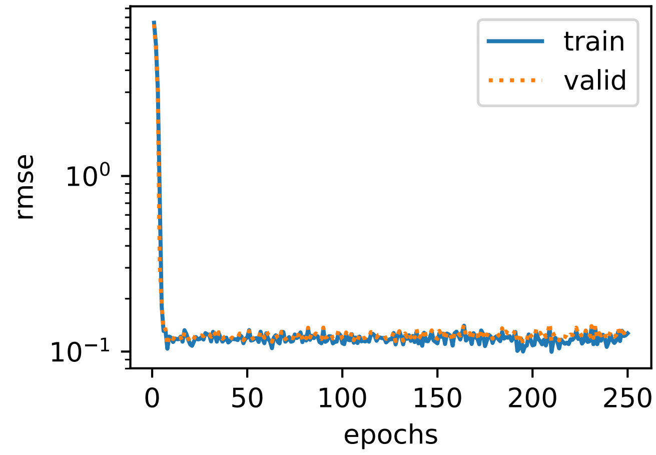
```
fold 1, train rmse: 0.107200, valid rmse: 0.140475
```



```
fold 2, train rmse: 0.106652, valid rmse: 0.131182
```



```
fold 3, train rmse: 0.130862, valid rmse: 0.155574
```



```
fold 4, train rmse: 0.127038, valid rmse: 0.131551
5-fold validation: avg train rmse: 0.116661, avg valid rmse: 0.136466
```

You will notice that sometimes the number of training errors for a set of hyper-parameters can be very low, while the number of errors for the K-fold cross validation may be higher. This is most likely a consequence of overfitting. Therefore, when we reduce the amount of training errors, we need to check whether the amount of errors in the k-fold cross-validation have also been reduced accordingly.

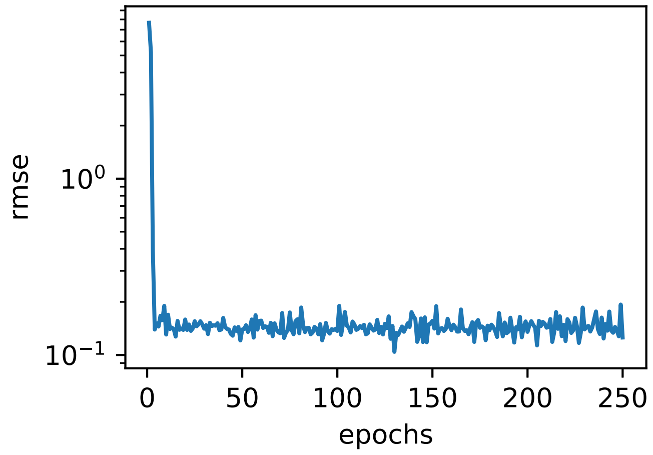
Predict and Submit

Now that we know what a good choice of hyperparameters should be, we might as well use all the data to train on it (rather than just $1 - 1/k$ of the data that is used in the crossvalidation slices). The model that we obtain in this way can then be applied to the test set. Saving the estimates in a CSV file will simplify uploading the results to Kaggle.

```
In [30]: def train_and_pred(train_features, test_feature, train_labels, test_data,
                        num_epochs, lr, weight_decay, batch_size):
             net = get_net()
             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']], axis=1)
             submission.to_csv('submission_alex.csv', index=False)
```

Let's invoke the model. A good sanity check is to see whether the predictions on the test set resemble those of the k-fold crossvalication process. If they do, it's time to upload them to Kaggle.

```
In [31]: train_and_pred(train_features, test_features, train_labels, test_data,
                        num_epochs, lr, weight_decay, batch_size)
```



```
train rmse 0.125626
```

```
In [ ]:
```

A file, `submission.csv` will be generated by the code above (CSV is one of the file formats accepted by Kaggle). Next, we can submit our predictions on Kaggle and compare them to the actual house price (label) on the testing data set, checking for errors. The steps are quite simple:

- Log in to the Kaggle website and visit the House Price Prediction Competition page.
- Click the “Submit Predictions” or “Late Submission” button on the right.
- Click the “Upload Submission File” button in the dashed box at the bottom of the page and select the prediction file you wish to upload.
- Click the “Make Submission” button at the bottom of the page to view your results.

Hints

1. Can you improve your model by minimizing the log-price directly? What happens if you try to predict the log price rather than the price?
2. Is it always a good idea to replace missing values by their mean? Hint - can you construct a situation where the values are not missing at random?
3. Find a better representation to deal with missing values. Hint - What happens if you add an indicator variable?
4. Improve the score on Kaggle by tuning the hyperparameters through k-fold crossvalidation.
5. Improve the score by improving the model (layers, regularization, dropout).
6. What happens if we do not standardize the continuous numerical features like we have done in this section?

Note for converting this notebook into PDF. If you use 'File -> Download as -> PDF', you may get the error that svg cannot converted because inkscape is not installed and cannot find PNG images. The easiest way is printing this notebook as a PDF in your browser. Or, you can install inkscape to convert SVG (On macOS, you may `brew cask install xquartz inkscape`, on Ubuntu, you may `sudo apt-get install inkscape`) and change the image URL to local filenames.