In [2]:
```
%matplotlib inline
import pandas as pd
import torch
from torch import nn
from d2l import torch as d2l
import warnings
warnings.filterwarnings('ignore')
```

Homework 1 Question 2

In [3]:
```
class KaggleHouse(d2l.DataModule):
    def __init__(self, batch_size, train=None, val=None):
        super().__init__()
        self.save_hyperparameters()
        if self.train is None:
            self.raw_train = pd.read_csv(d2l.download(
                d2l.DATA_URL + 'kaggle_house_pred_train.csv', self.root,
                sha1_hash='585e9cc93e70b39160e7921475f9bcd7d31219ce'))
            self.raw_val = pd.read_csv(d2l.download(
                d2l.DATA_URL + 'kaggle_house_pred_test.csv', self.root,
                sha1_hash='fa19780a7b011d9b009e8bff8e99922a8ee2eb90'))
```

In [4]:
```
data = KaggleHouse(batch_size=64)
print(data.raw_train.shape)
print(data.raw_val.shape)
```

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

In [5]:
```
print(data.raw_train.iloc[:4, [0, 1, 2, 3, -3, -2, -1]])
```

```
   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
```

In [6]:
```
@d2l.add_to_class(KaggleHouse)
def preprocess(self):
    # Remove the ID and label columns
    label = 'SalePrice'
    features = pd.concat(
        (self.raw_train.drop(columns=['Id', label]),
         self.raw_val.drop(columns=['Id'])))
    # Standardize numerical columns
    numeric_features = features.dtypes[features.dtypes != 'object'].index
    features[numeric_features] = features[numeric_features].apply(
        lambda x: (x - x.mean()) / (x.std()))
    # Replace NAN numerical features by 0
    features[numeric_features] = features[numeric_features].fillna(0)
    # Replace discrete features by one-hot encoding.
    features = pd.get_dummies(features, dummy_na=True)
    # Save preprocessed features
    self.train = features[:self.raw_train.shape[0]].copy()
    self.train[label] = self.raw_train[label]
    self.val = features[self.raw_train.shape[0]:].copy()
```

In [7]:
```
data.preprocess()
data.train.shape
```

Out[7]:
```
(1460, 332)
```

In [8]:
```
@d2l.add_to_class(KaggleHouse)
def get_dataloader(self, train):
    label = 'SalePrice'
    data = self.train if train else self.val
    if label not in data: return
    get_tensor = lambda x: torch.tensor(x.values, dtype=torch.float32)
    # Logarithm of prices
    tensors = (get_tensor(data.drop(columns=[label])),  # X
               torch.log(get_tensor(data[label])).reshape((-1, 1)))  # Y
    return self.get_tensorloader(tensors, train)
```

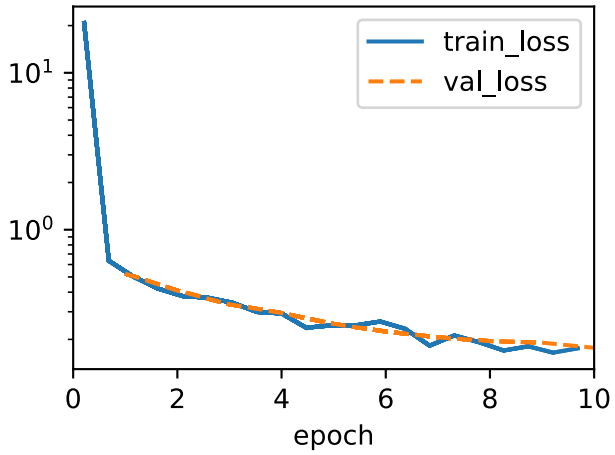
In [ ]:
```

```

In [9]:
```
def k_fold_data(data, k):
    rets = []
    fold_size = data.train.shape[0] // k
    for j in range(k):
        idx = range(j * fold_size, (j+1) * fold_size)
        rets.append(KaggleHouse(data.batch_size, data.train.drop(index=idx),
                                data.train.loc[idx]))
    return rets
```

In [10]:
```
def k_fold(trainer, data, k, lr):
    val_loss, models = [], []
    for i, data_fold in enumerate(k_fold_data(data, k)):
        model = d2l.LinearRegression(lr)
        model.board.yscale='log'
        if i != 0: model.board.display = False
        trainer.fit(model, data_fold)
        val_loss.append(float(model.board.data['val_loss'][-1].y))
        models.append(model)
    print(f'average validation log mse = {sum(val_loss)/len(val_loss)}')
    return models
```

In [11]:
```
trainer = d2l.Trainer(max_epochs=10)
models = k_fold(trainer, data, k=5, lr=0.01)
```

```
average validation log mse = 0.18146310120821
```



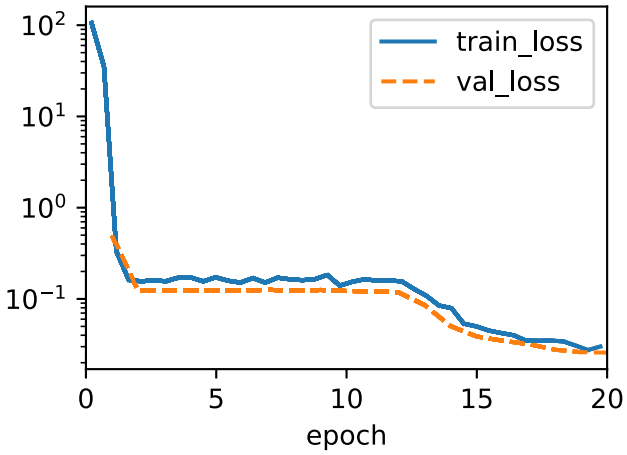
Homework 1 Question 2a

In [12]:
```
class WeightDecayLinearReg(d2l.LinearRegression):
    def __init__(self, num_outputs, num_hiddens_1, num_hiddens_2,
                 wd, lr):
        super().__init__(lr)
        self.save_hyperparameters()
        self.wd = wd
        self.net = nn.Sequential(
            nn.Flatten(), nn.LazyLinear(num_hiddens_1), nn.ReLU(),
            nn.LazyLinear(num_hiddens_1), nn.ReLU(),
            nn.LazyLinear(num_outputs))

@d2l.add_to_class(WeightDecayLinearReg)
def configure_optimizers(self):
    return torch.optim.SGD(self.parameters(), lr=self.lr,
                           weight_decay=self.wd)
```

In [13]:
```
def k_fold(trainer, data, k, lr, second_model=False):
    val_loss, models = [], []
    for i, data_fold in enumerate(k_fold_data(data, k)):
        hparams = {'num_outputs':1, 'num_hiddens_1':8, 'num_hiddens_2':4,
           'wd':0.01, 'lr':lr}
        model = d2l.LinearRegression(lr) if second_model==False else \
        WeightDecayLinearReg(**hparams)
        model.board.yscale='log'
        if i != 0: model.board.display = False
        trainer.fit(model, data_fold)
        val_loss.append(float(model.board.data['val_loss'][-1].y))
        models.append(model)
    print(f'average validation log mse = {sum(val_loss)/len(val_loss)}')
    return models
```

In [131…
```
trainer = d2l.Trainer(max_epochs=20)
models = k_fold(trainer, data, k=10, lr=0.01, second_model=True)
```

```
average validation log mse = 0.045676283352077005
```



In [15]:
```
preds = [model(torch.tensor(data.val.values, dtype=torch.float32))
         for model in models]

ensemble_preds = torch.exp(torch.cat(preds, 1)).mean(1)
submission = pd.DataFrame({'Id':data.raw_val.Id,
                           'SalePrice':ensemble_preds.detach().numpy()})
submission.to_csv('submission.csv', index=False)
```

In [ ]:
```
```