1.a

```
!pip install d2l==1.0.0b0
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting d2l==1.0.0b0
  Downloading d2l-1.0.0b0-py3-none-any.whl (141 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 141.6/141.6 KB 5.5 MB/s eta 0:00:00
Requirement already satisfied: requests in /usr/local/lib/python3.8/dist-packages (from d2l==1.0.0b0) (2.25.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.8/dist-packages (from d2l==1.0.0b0) (1.3.5)
Collecting jupyter
  Downloading jupyter-1.0.0-py2.py3-none-any.whl (2.7 kB)
Requirement already satisfied: scipy in /usr/local/lib/python3.8/dist-packages (from d2l==1.0.0b0) (1.7.3)
Collecting gym==0.21.0
  Downloading gym-0.21.0.tar.gz (1.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.5/1.5 MB 43.0 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: matplotlib in /usr/local/lib/python3.8/dist-packages (from d2l==1.0.0b0) (3.2.2)
Collecting gpytorch
  Downloading gpytorch-1.9.1-py3-none-any.whl (250 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 250.9/250.9 KB 33.2 MB/s eta 0:00:00
Collecting matplotlib-inline
  Downloading matplotlib_inline-0.1.6-py3-none-any.whl (9.4 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (from d2l==1.0.0b0) (1.21.6)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.8/dist-packages (from gym==0.21.0->d2l==1.0.0b0) (2.2.1)
Collecting linear-operator>=0.2.0
  Downloading linear_operator-0.3.0-py3-none-any.whl (155 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 155.6/155.6 KB 23.4 MB/s eta 0:00:00
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.8/dist-packages (from gpytorch->d2l==1.0.0b0) (1.0.2)
Requirement already satisfied: nbconvert in /usr/local/lib/python3.8/dist-packages (from jupyter->d2l==1.0.0b0) (5.6.1)
Requirement already satisfied: ipykernel in /usr/local/lib/python3.8/dist-packages (from jupyter->d2l==1.0.0b0) (5.3.4)
Requirement already satisfied: ipywidgets in /usr/local/lib/python3.8/dist-packages (from jupyter->d2l==1.0.0b0) (7.7.1)
Requirement already satisfied: notebook in /usr/local/lib/python3.8/dist-packages (from jupyter->d2l==1.0.0b0) (5.7.16)
Requirement already satisfied: jupyter-console in /usr/local/lib/python3.8/dist-packages (from jupyter->d2l==1.0.0b0) (6.1.0)
Collecting qtconsole
  Downloading qtconsole-5.4.0-py3-none-any.whl (121 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 121.0/121.0 KB 17.7 MB/s eta 0:00:00
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.8/dist-packages (from matplotlib->d2l==1.0.0b0) (0.11.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib->
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib->d2l==1.0.0b0) (2.8.2
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib->d2l==1.0.0b0) (1.4.4)
Requirement already satisfied: traitlets in /usr/local/lib/python3.8/dist-packages (from matplotlib-inline->d2l==1.0.0b0) (5.7.1)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-packages (from pandas->d2l==1.0.0b0) (2022.7.1)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.8/dist-packages (from requests->d2l==1.0.0b0) (1.24.3
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/dist-packages (from requests->d2l==1.0.0b0) (2022.12.7
Requirement already satisfied: chardet<5,>=3.0.2 in /usr/local/lib/python3.8/dist-packages (from requests->d2l==1.0.0b0) (4.0.0)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-packages (from requests->d2l==1.0.0b0) (2.10)
Requirement already satisfied: torch>=1.11 in /usr/local/lib/python3.8/dist-packages (from linear-operator>=0.2.0->gpytorch->d2l==1.0
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-packages (from python-dateutil>=2.1->matplotlib->d2l==1.0.0
Requirement already satisfied: ipython>=5.0.0 in /usr/local/lib/python3.8/dist-packages (from ipykernel->jupyter->d2l==1.0.0b0) (7.9
Requirement already satisfied: tornado>=4.2 in /usr/local/lib/python3.8/dist-packages (from ipykernel->jupyter->d2l==1.0.0b0) (6.0.4
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.8/dist-packages (from ipykernel->jupyter->d2l==1.0.0b0) (6.1
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /usr/local/lib/python3.8/dist-packages (from ipywidgets->jupyter->d2l==1
Requirement already satisfied: widgetsnbextension~=3.6.0 in /usr/local/lib/python3.8/dist-packages (from ipywidgets->jupyter->d2l==1
Requirement already satisfied: ipython-genutils~=0.2.0 in /usr/local/lib/python3.8/dist-packages (from ipywidgets->jupyter->d2l==1.0
Requirement already satisfied: pygments in /usr/local/lib/python3.8/dist-packages (from jupyter-console->jupyter->d2l==1.0.0b0) (2.6
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.8/dist-packages (from jupyter-
Requirement already satisfied: testpath in /usr/local/lib/python3.8/dist-packages (from nbconvert->jupyter->d2l==1.0.0b0) (0.6.0)
Requirement already satisfied: bleach in /usr/local/lib/python3.8/dist-packages (from nbconvert->jupyter->d2l==1.0.0b0) (6.0.0)
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.8/dist-packages (from nbconvert->jupyter->d2l==1.0.0b0
Requirement already satisfied: jinja2>=2.4 in /usr/local/lib/python3.8/dist-packages (from nbconvert->jupyter->d2l==1.0.0b0) (2.11.3
```

```
# Importing all the Necessary Libraries

%matplotlib inline
import pandas as pd
import numpy as np

import torch.nn.functional as F
from torchvision import transforms
from d2l import torch as d2l
from torch import nn

import matplotlib.pyplot as plt
import time
import torchvision
```

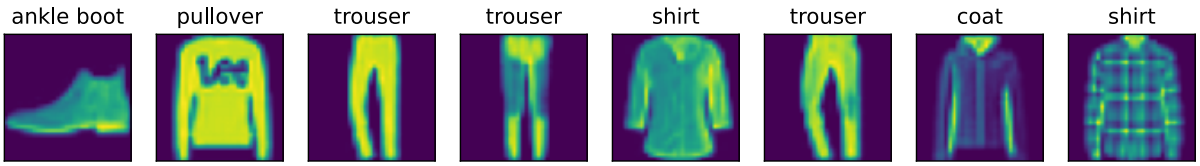
```
import torch
torch.__version__
torch.cuda.current_device()
torch.cuda.get_device_name(0)

d2l.use_svg_display()


import warnings
warnings.filterwarnings("ignore")


# Loading the FashionMNIST Dataset
class FashionMNIST(d2l.DataModule):
  def __init__(self, batch_size = 64, resize = (28, 28)):
    super().__init__()
    self.save_hyperparameters()
    trans = transforms.Compose([transforms.Resize(resize),
                                transforms.ToTensor()])
    self.train = torchvision.datasets.FashionMNIST(
        root = self.root, train = True, transform = trans, download = True)
    self.val = torchvision.datasets.FashionMNIST(
        root = self.root, train = False, transform = trans, download = True)



data = FashionMNIST(resize = (32, 32))
print("images trained   = ", len(data.train))
print("Images validated = ", len(data.val))

data.train[0][0].shape
```

```
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3
100%                                          26421880/26421880 [00:02<00:00, 19862841.00it/s]
Extracting ../data/FashionMNIST/raw/train-images-idx3-ubyte.gz to ../data/FashionMNIST/r

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1
100%                                          29515/29515 [00:00<00:00, 210591.74it/s]
Extracting ../data/FashionMNIST/raw/train-labels-idx1-ubyte.gz to ../data/FashionMNIST/r

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-
100%                                          4422102/4422102 [00:01<00:00, 6685056.30it/s]
Extracting ../data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz to ../data/FashionMNIST/ra

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-
100%                                          5148/5148 [00:00<00:00, 294932.14it/s]
Extracting ../data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to ../data/FashionMNIST/ra

Training Images   =  60000
Validation Images =  10000
torch.Size([1, 32, 32])
```

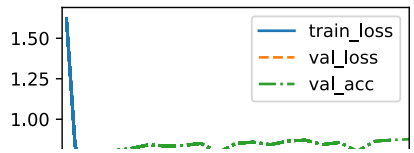
```
# Function that Converts between Numeric Labels and the Class Names
@d2l.add_to_class(FashionMNIST)
def text_labels(self, indices):
  labels = ['t-shirt', 'trouser', 'pullover', 'dress', 'coat',
            'sandal', 'shirt', 'sneaker', 'bag', 'ankle boot']
  return [labels[int(i)] for i in indices]


@d2l.add_to_class(FashionMNIST)
def get_dataloader(self, train):
  data = self.train if train else self.val
  return torch.utils.data.DataLoader(data, self.batch_size, shuffle = train,
                                     num_workers = self.num_workers)

X, Y = next(iter(data.train_dataloader()))
print(X.shape, X.dtype, Y.shape, Y.dtype)
```

```
torch.Size([64, 1, 32, 32]) torch.float32 torch.Size([64]) torch.int64
```

```
# Function to Display and Visualize the Images and their Corresponding Labels

def show_images(imgs, num_rows, num_cols, titles = None, scale = 1.5):
  raise NotImplementedError

@d2l.add_to_class(FashionMNIST)
def visualize(self, batch, nrows = 1, ncols = 8, labels = []):
  X, Y = batch
  if not labels:
    labels = self.text_labels(Y)
  d2l.show_images(X.squeeze(1), nrows, ncols, titles = labels)

batch = next(iter(data.val_dataloader()))
data.visualize(batch)
```




```
class Classifier(d2l.Module):
  def validation_step(self, batch):
    Y_hat = self(*batch[:-1])
    self.plot('Loss', self.loss(Y_hat, batch[-1]), train = False)
    self.plot('Acc', self.accuracy(Y_hat, batch[-1]), train = False)

@d2l.add_to_class(d2l.Module)
def configure_optimizers(self):
  return torch.optim.SGD(self.parameters(), lr = self.lr)

@d2l.add_to_class(Classifier)
def accuracy(self, Y_hat, Y, averaged = True):
  Y_hat = Y_hat.reshape((-1, Y_hat.shape[-1]))
  Preds = Y_hat.argmax(axis = 1).type(Y.dtype)
  compare = (Preds == Y.reshape(-1)).type(torch.float32)
  return compare.mean() if averaged else compare


class SoftmaxRegression(d2l.Classifier):
  def __init__(self, num_outputs, hidden_layer1, hidden_layer2, hidden_layer3, lr):
    super().__init__()
    self.save_hyperparameters()
    self.net = nn.Sequential(nn.Flatten(),nn.LazyLinear(hidden_layer1),nn.ReLU(),
                             nn.LazyLinear(hidden_layer2),nn.ReLU(),nn.LazyLinear(hidden_layer3),
                             nn.ReLU(),nn.LazyLinear(num_outputs))
  def forward(self, X):
    return self.net(X)

@d2l.add_to_class(Classifier)
def loss(self, Y_hat, Y, averaged = True):
  Y_hat = Y_hat.reshape((-1, Y_hat.shape[-1]))
  Y = Y.reshape((-1))
  return F.cross_entropy(
      Y_hat, Y, reduction = 'mean' if averaged else 'none')


# Plotting over 20 Epochs (Basic Required for HW)
data = d2l.FashionMNIST(batch_size = 256)
model = SoftmaxRegression(num_outputs = 10, hidden_layer1 = 512, hidden_layer2 = 256, hidden_layer3 = 256, lr = 0.1)
trainer = d2l.Trainer(max_epochs = 20)
trainer.fit(model, data)
```




```
# Plotting over 60 Epochs for Comparison
data = d2l.FashionMNIST(batch_size = 256)
model = SoftmaxRegression(num_outputs = 10, hidden_layer1 = 512, hidden_layer2 = 256, hidden_layer3 = 256, lr = 0.1)
trainer = d2l.Trainer(max_epochs = 60)
trainer.fit(model, data)
```

1.b.

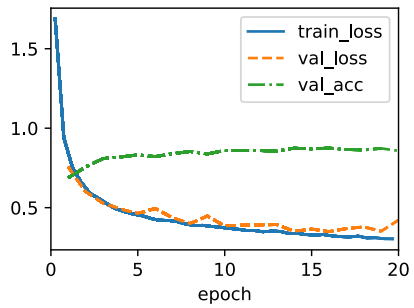
```
# Defining the the l2 Norm Penalty
def l2_penalty(w):
  return (w ** 2).sum() / 2


# Defining the Model with Weight Decay Property added
class WeightDecay(Classifier):
  def __init__(self, num_outputs, hidden_layer1, hidden_layer2, hidden_layer3, wd, lr, sigma = 0.01):
    super().__init__(lr)
    self.save_hyperparameters()
    self.wd = wd
    self.weights = torch.normal(0, sigma, (hidden_layer1, 1), requires_grad=True)
    self.net = nn.Sequential(nn.Flatten(),nn.LazyLinear(hidden_layer1),nn.ReLU(),
                             nn.LazyLinear(hidden_layer2),nn.ReLU(),nn.LazyLinear(hidden_layer3),
                             nn.ReLU(),nn.LazyLinear(num_outputs))
  def forward(self, X):
    return self.net(X)

  # Defining the Loss Function with Penalty Added
  def loss(self, Y_hat, Y, averaged = True):
    Y_hat = Y_hat.reshape((-1, Y_hat.shape[-1]))
    Y = Y.reshape((-1))
    return (super().loss(Y_hat, Y) +
            self.wd * l2_penalty(self.weights))

data = d2l.FashionMNIST(batch_size = 256)
trainer = d2l.Trainer(max_epochs = 20)

def train_scratch():
  t_0 = time.time()
  model.board.yscale = 'log'
  trainer.fit(model, data)
  t_1 = time.time()
  print("Total Training Time (From Saved Model): ", t_1 - t_0)
  print()
  print()


# Plotting the Results with Weight Decay Added
model = WeightDecay(num_outputs = 10, hidden_layer1 = 512, hidden_layer2 = 256, hidden_layer3 = 256, wd = 2, lr = 0.1)
train_scratch()

# Saving the Weight Decay Model Parameters for Future Use
torch.save(model.state_dict(), 'MLP_weightDecay.params')
```

```
     Total Training Time (From Scratch):  255.93493556976318
```

1.c.

```
class DropoutMLP(d2l.Classifier):
  def __init__(self, num_outputs, hidden_layer1, hidden_layer2, hidden_layer3, dropout_1, dropout_2, dropout_3, lr):
    super().__init__()
    self.save_hyperparameters()
    self.net = nn.Sequential(nn.Flatten(),
                             nn.LazyLinear(hidden_layer1),
                             nn.ReLU(),
                             nn.Dropout(dropout_1),
                             nn.LazyLinear(hidden_layer2),
                             nn.ReLU(),
                             nn.Dropout(dropout_2),
                             nn.LazyLinear(hidden_layer3),
                             nn.ReLU(),
                             nn.Dropout(dropout_3),
                             nn.LazyLinear(num_outputs))
  def forward(self, X):
    return self.net(X)

@d2l.add_to_class(Classifier)
def loss(self, Y_hat, Y, averaged = True):
  Y_hat = Y_hat.reshape((-1, Y_hat.shape[-1]))
  Y = Y.reshape((-1))
  return F.cross_entropy(
      Y_hat, Y, reduction = 'mean' if averaged else 'none')


# Plotting over 20 Epochs
t_0 = time.time()
data = d2l.FashionMNIST(batch_size = 256)
model = DropoutMLP(num_outputs = 10, hidden_layer1 = 512, hidden_layer2 = 256, hidden_layer3 = 256, dropout_1 = 0.3, dropout_2 = 0.3, dropout
trainer = d2l.Trainer(max_epochs = 20)
trainer.fit(model, data)
t_1 = time.time()
print("Total Training Time (From Scratch): ", t_1 - t_0)
print()
print()

# Saving the Weight Decay Model Parameters for Future Use
torch.save(model.state_dict(), 'MLP_Dropout.params')
```

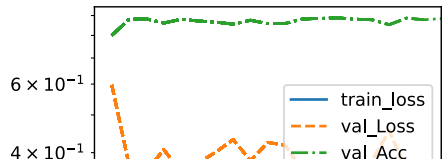
```
     Total Training Time (From Scratch):  262.92932295799255
```



1.d.

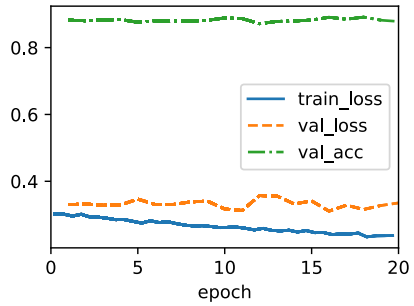
```
model = WeightDecay(num_outputs = 10, hidden_layer1 = 512, hidden_layer2 = 256, hidden_layer3 = 128, wd = 1, lr = 0.1)
model.load_state_dict(torch.load('MLP_weightDecay.params'))
train_scratch()
```

```
Total Training Time (From Saved Model):  255.7039134502411
```




```
# Plotting over 20 Epochs
t_0 = time.time()
model = DropoutMLP(num_outputs = 10, hidden_layer1 = 512, hidden_layer2 = 256, hidden_layer3 = 128, dropout_1 = 0.3, dropout_2 = 0.3, dropout
model.load_state_dict(torch.load('MLP_Dropout.params'))
#saved_MLP_WeightDecay.eval()
trainer = d2l.Trainer(max_epochs = 20)
trainer.fit(model, data)
t_1 = time.time()
print("Total Training Time (From Saved Model): ", t_1 - t_0)
print()
print()
```

```
Total Training Time (From Saved Model):  266.3024890422821
```



PROBLEM 2

2.a. When the dataset has different ranges or the characteristics are in different units, standardization is applied on continuous numerical features. In our example, standardization is employed to prevent the date from yielding inaccurate data.*italicized text*

2.b.

```
    # Downloading and Caching the Housing Dataset from Kaggle
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'))
```

```
# Loading the Dataset
data = KaggleHouse(batch_size = 64)
print(data.raw_train.shape)
print(data.raw_val.shape)
```

Downloading ../data/kaggle_house_pred_train.csv from http://d2l-data.s3-accelerate.amazonaws.com/kaggle_house_pred_train.csv...
Downloading ../data/kaggle_house_pred_test.csv from http://d2l-data.s3-accelerate.amazonaws.com/kaggle_house_pred_test.csv... (1460, 81)
(1459, 80)

```
@d2l.add_to_class(KaggleHouse)
def preprocess(self):
    # Removing the ID and label columns
    label = 'SalePrice'
    features = pd.concat(
        (self.raw_train.drop(columns=['Id', label]),
         self.raw_val.drop(columns=['Id'])))

    # Standardizing the numerical columns
    numeric_features = features.dtypes[features.dtypes != 'object'].index
    features[numeric_features] = features[numeric_features].apply(
        lambda x: (x - x.mean()) / (x.std()))

    # Replacing NAN numerical features by 0
    features[numeric_features] = features[numeric_features].fillna(0)

    # Replacing discrete features by one-hot encoding.
    features = pd.get_dummies(features, dummy_na=True)

    # Saving 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()



# Preprocessing the Data
data.preprocess()
print(data.train.shape)
print(data.val.shape)
```

```
(1460, 332)
(1459, 331)
```

```
# Creation of a Dataloader Function
@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)
```

2.c

```
# Defining the the l2 Norm Penalty for Weight Decay
def l2_penalty(w):
  return (w ** 2).sum() / 2

class LinearRegression(d2l.Module):
    def __init__(self, lr, input_size, hidden_sizes, dropout_prob, weight_decay):
        super().__init__()
        self.lr = lr
        self.weight_decay = weight_decay
        self.layers = []
        for i, hidden_size in enumerate(hidden_sizes):
            self.layers.append(nn.Linear(input_size, hidden_size))
            self.layers.append(nn.ReLU())
            self.layers.append(nn.Dropout(p=dropout_prob))
            input_size = hidden_size
        self.layers.append(nn.Linear(input_size, 1))
        self.net = nn.Sequential(*self.layers)
        self.reset_parameters()

    def reset_parameters(self):
        for layer in self.layers:
            if isinstance(layer, nn.Linear):
                nn.init.normal_(layer.weight, mean=0.5, std=0.5)
                nn.init.constant_(layer.bias, 0)
```

```
        def forward(self, X):
            return self.net(X)

        def loss(self, Y_hat, Y):
            Y_hat = Y_hat.view(-1)
            Y = Y.view(-1)
            mse_loss = nn.MSELoss()(Y_hat, Y)
            l2_loss = self.weight_decay * nn.functional.l1_loss(self.net.weight)
            return mse_loss + l2_loss

        def configure_optimizers(self):
            optimizer = optim.SGD(self.parameters(), lr=self.lr)
            return [optimizer]

    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


    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


    # Plotting the Complex model with both Dropout and Weight Decay
    trainer = d2l.Trainer(max_epochs=20)
    models = k_fold(trainer, data, k=12, lr=0.01)
```

average validation log mse = 0.11221548200895388

2.d.

```
    # Getting the .csv File ready for submission on Kaggle
    preds = [model(torch.tensor(data.val.values, dtype=torch.float32))
             for model in models]
    # Taking exponentiation of predictions in the logarithm scale
    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)
```

0s completed at 2:00 AM