COES474_HW_1_2019170361_윤동현

[9/24] HW 1: Preliminaries/Linear Regression/Classification

```
!pip install d2l==1.0.3
```

```
Requirement already satisfied: d2l==1.0.3 in /usr/local/lib/python3.10/dist-packages (1.0.3)
Requirement already satisfied: jupyter==1.0.0 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: numpy==1.23.5 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: matplotlib==3.7.2 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: matplotlib-inline==0.1.6 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: requests==2.31.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: pandas==2.0.3 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: scipy==1.10.1 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: notebook in /usr/local/lib/python3.10/dist-packages (from jup
Requirement already satisfied: qtconsole in /usr/local/lib/python3.10/dist-packages (from ju
Requirement already satisfied: jupyter-console in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: nbconvert in /usr/local/lib/python3.10/dist-packages (from ju
Requirement already satisfied: ipykernel in /usr/local/lib/python3.10/dist-packages (from ju
Requirement already satisfied: ipywidgets in /usr/local/lib/python3.10/dist-packages (from j
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: traitlets in /usr/local/lib/python3.10/dist-packages (from ma
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from pyt
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: ipython>=5.0.0 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: tornado>=4.2 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: widgetsnbextension~=3.6.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/local/li
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (from jup
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-packages (from nbconve
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from nbcon
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/dist-packages (from n
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: jupyter-core>=4.7 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: nbformat>=5.1 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: tinycss2 in /usr/local/lib/python3.10/dist-packages (from nbc
Requirement already satisfied: pyzmq<25,>=17 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: argon2-cffi in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: nest-asyncio>=1.5 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: Send2Trash>=1.8.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: terminado>=0.8.3 in /usr/local/lib/python3.10/dist-packages (
```

2.1 Data Manipulation

2.1.1 Getting Started

더블클릭 또는 Enter 키를 눌러 수정

```
import torch
```

```
x = torch.arange(12, dtype=torch.float32)
x
```

```
tensor([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11.])
```

```
x.numel()
```

```
12
```

```
x.shape
```

```
torch.Size([12])
```

```
X = x.reshape(3, 4)
X
```

```
tensor([[ 0.,  1.,  2.,  3.],
        [ 4.,  5.,  6.,  7.],
        [ 8.,  9., 10., 11.]])
```

```
torch.zeros((2, 3, 4))
```

```
tensor([[[0., 0., 0., 0.],
         [0., 0., 0., 0.],
         [0., 0., 0., 0.]],

        [[0., 0., 0., 0.],
         [0., 0., 0., 0.],
         [0., 0., 0., 0.]]])
```

```
torch.ones((2, 3, 4))
```

```
tensor([[[1., 1., 1., 1.],
         [1., 1., 1., 1.],
         [1., 1., 1., 1.]],

        [[1., 1., 1., 1.],
         [1., 1., 1., 1.],
         [1., 1., 1., 1.]]])
```

```
torch.randn(3, 4)
```

```
tensor([[ 0.5748,  0.4082, -1.1605, -1.5290],
        [ 0.1549,  2.2430,  0.1938,  0.6684],
        [ 0.5154, -0.0658,  1.2040, -0.9475]])
```

```
torch.tensor([[2, 1, 4, 3], [1, 2, 3, 4], [4, 3, 2, 1]])
```

```
tensor([[2, 1, 4, 3],
        [1, 2, 3, 4],
        [4, 3, 2, 1]])
```

2.1.2 Indexing and Slicing

```
X[-1], X[1:3]
```

```
(tensor([ 8.,  9., 10., 11.]),
 tensor([[ 4.,  5.,  6.,  7.],
         [ 8.,  9., 10., 11.]]))
```

```
X[1, 2] = 17
X
```

```
tensor([[ 0.,  1.,  2.,  3.],
        [ 4.,  5., 17.,  7.],
        [ 8.,  9., 10., 11.]])
```

```
X[:2, :] = 12
X
```

```
tensor([[12., 12., 12., 12.],
        [12., 12., 12., 12.],
        [ 8.,  9., 10., 11.]])
```

2.1.3. Operations

```
torch.exp(x)
```

```
tensor([162754.7969, 162754.7969, 162754.7969, 162754.7969, 162754.7969,
        162754.7969, 162754.7969, 162754.7969,   2980.9580,   8103.0840,
         22026.4648,  59874.1406])
```

```
x = torch.tensor([1.0, 2, 4, 8])
y = torch.tensor([2, 2, 2, 2])
x + y, x - y, x * y, x / y, x ** y
```

```
(tensor([ 3.,  4.,  6., 10.]),
 tensor([-1.,  0.,  2.,  6.]),
 tensor([ 2.,  4.,  8., 16.]),
 tensor([0.5000, 1.0000, 2.0000, 4.0000]),
 tensor([ 1.,  4., 16., 64.]))
```

```
X = torch.arange(12, dtype=torch.float32).reshape((3,4))
Y = torch.tensor([[2.0, 1, 4, 3], [1, 2, 3, 4], [4, 3, 2, 1]])
torch.cat((X, Y), dim=0), torch.cat((X, Y), dim=1)
```

```
(tensor([[ 0.,  1.,  2.,  3.],
         [ 4.,  5.,  6.,  7.],
         [ 8.,  9., 10., 11.],
         [ 2.,  1.,  4.,  3.],
         [ 1.,  2.,  3.,  4.],
         [ 4.,  3.,  2.,  1.]]),
 tensor([[ 0.,  1.,  2.,  3.,  2.,  1.,  4.,  3.],
         [ 4.,  5.,  6.,  7.,  1.,  2.,  3.,  4.],
         [ 8.,  9., 10., 11.,  4.,  3.,  2.,  1.]]))
```

```
X == Y
```

```
tensor([[False,  True, False,  True],
        [False, False, False, False],
        [False, False, False, False]])
```

```
X.sum()
```

```
tensor(66.)
```

2.1.4 Broadcasting

```
a = torch.arange(3).reshape((3, 1))
b = torch.arange(2).reshape((1, 2))
a, b
```

```
(tensor([[0],
         [1],
         [2]]),
 tensor([[0, 1]]))
```

```
a + b
```

```
tensor([[0, 1],
        [1, 2],
        [2, 3]])
```

2.1.5 Saving Memory

```
before = id(Y)
Y = Y + X
id(Y) == before
```

```
False
```

```
Z = torch.zeros_like(Y)
print('id(Z):', id(Z))
Z[:] = X + Y
print('id(Z):', id(Z))
```

```
id(Z): 133724702829088
id(Z): 133724702829088
```

2.1.6 Conversion to Other Python Objects

```
A = X.numpy()
B = torch.from_numpy(A)
type(A), type(B)
```

```
(numpy.ndarray, torch.Tensor)
```

```
a = torch.tensor([3.5])
a, a.item(), float(a), int(a)
```

```
(tensor([3.5000]), 3.5, 3.5, 3)
```

2.2 Data Preprocessing

2.2.1 Reading the Dataset

```
import os

os.makedirs(os.path.join('..', 'data'), exist_ok=True)
data_file = os.path.join('..', 'data', 'house_tiny.csv')
with open(data_file, 'w') as f:
    f.write('''NumRooms,RoofType,Price
NA,NA,127500
2,NA,106000
4,Slate,178100
NA,NA,140000''')
```

```
import pandas as pd

data = pd.read_csv(data_file)
print(data)
```

```
   NumRooms RoofType   Price
0       NaN      NaN  127500
1       2.0      NaN  106000
2       4.0    Slate  178100
3       NaN      NaN  140000
```

2.2.2 Data Preparation

```
inputs, targets = data.iloc[:, 0:2], data.iloc[:, 2]
inputs = pd.get_dummies(inputs, dummy_na=True)
print(inputs)
```

```
   NumRooms  RoofType_Slate  RoofType_nan
0       NaN           False          True
1       2.0           False          True
2       4.0            True         False
3       NaN           False          True
```

```
inputs = inputs.fillna(inputs.mean())
print(inputs)
```

```
   NumRooms  RoofType_Slate  RoofType_nan
0       3.0           False          True
1       2.0           False          True
2       4.0            True         False
3       3.0           False          True
```

2.2.3 Conversion to ther Tensor Format

```
import torch

X = torch.tensor(inputs.to_numpy(dtype=float))
y = torch.tensor(targets.to_numpy(dtype=float))
X, y
```

```
(tensor([[3., 0., 1.],
         [2., 0., 1.],
         [4., 1., 0.],
         [3., 0., 1.]], dtype=torch.float64),
 tensor([127500., 106000., 178100., 140000.], dtype=torch.float64))
```

2.3 Linear Algebra

```
import torch
```

2.3.1 Scalars

```
x = torch.tensor(3.0)
y = torch.tensor(2.0)

x + y, x * y, x / y, x**y
```

```
(tensor(5.), tensor(6.), tensor(1.5000), tensor(9.))
```

2.3.2 Vectors

```
x = torch.arange(3)
x
```

```
tensor([0, 1, 2])
```

```
x[2]
```

```
tensor(2)
```

```
len(x)
```

```
3
```

```
x.shape
```

```
torch.Size([3])
```

2.3.3 Matrices

```
A = torch.arange(6).reshape(3, 2)
A
```

```
tensor([[0, 1],
        [2, 3],
        [4, 5]])
```

```
A.T
```

```
tensor([[0, 2, 4],
        [1, 3, 5]])
```

```
A = torch.tensor([[1, 2, 3], [2, 0, 4], [3, 4, 5]])
A == A.T
```

```
tensor([[True, True, True],
        [True, True, True],
        [True, True, True]])
```

2.3.4 Tensors

```
torch.arange(24).reshape(2, 3, 4)
```

```
tensor([[[ 0,  1,  2,  3],
         [ 4,  5,  6,  7],
         [ 8,  9, 10, 11]],

        [[12, 13, 14, 15],
         [16, 17, 18, 19],
         [20, 21, 22, 23]]])
```

2.3.5 Basic Properties of Tensor Arithmetic

```
A = torch.arange(6, dtype=torch.float32).reshape(2, 3)
B = A.clone()
A, A + B
```

```
(tensor([[0., 1., 2.],
         [3., 4., 5.]]),
 tensor([[ 0.,  2.,  4.],
         [ 6.,  8., 10.]]))
```

```
A * B
```

```
tensor([[ 0.,  1.,  4.],
        [ 9., 16., 25.]])
```

```
a = 2
X = torch.arange(24).reshape(2, 3, 4)
a + X, (a * X).shape
```

```
(tensor([[[ 2,  3,  4,  5],
          [ 6,  7,  8,  9],
          [10, 11, 12, 13]],

         [[14, 15, 16, 17],
          [18, 19, 20, 21],
          [22, 23, 24, 25]]]),
 torch.Size([2, 3, 4]))
```

2.3.6 Reduction

```
x = torch.arange(3, dtype=torch.float32)
x, x.sum()
```

```
(tensor([0., 1., 2.]), tensor(3.))
```

```
A.shape, A.sum()
```

```
(torch.Size([2, 3]), tensor(15.))
```

```
A.shape, A.sum(axis=0).shape
```

```
(torch.Size([2, 3]), torch.Size([3]))
```

```
A.shape, A.sum(axis=1).shape
```

```
(torch.Size([2, 3]), torch.Size([2]))
```

```
A.sum(axis=[0, 1]) == A.sum()
```

```
tensor(True)
```

```
A.mean(), A.sum() / A.numel()
```

```
(tensor(2.5000), tensor(2.5000))
```

```
A.mean(axis=0), A.sum(axis=0) / A.shape[0]
```

```
(tensor([1.5000, 2.5000, 3.5000]), tensor([1.5000, 2.5000, 3.5000]))
```

2.3.7 Non-Reduction Sum

```
sum_A = A.sum(axis=1, keepdims=True)
sum_A, sum_A.shape
```

```
(tensor([[ 3.],
         [12.]]),
 torch.Size([2, 1]))
```

```
A / sum_A
```

```
tensor([[0.0000, 0.3333, 0.6667],
        [0.2500, 0.3333, 0.4167]])
```

```
A.cumsum(axis=0)
```

```
tensor([[0., 1., 2.],
        [3., 5., 7.]])
```

2.3.8 Dot Products

```
y = torch.ones(3, dtype = torch.float32)
x, y, torch.dot(x, y)
```

```
(tensor([0., 1., 2.]), tensor([1., 1., 1.]), tensor(3.))
```

```
torch.sum(x * y)
```

```
tensor(3.)
```

2.3.9 Matrix-Vector Products

```
A.shape, x.shape, torch.mv(A, x), A@x
```

```
(torch.Size([2, 3]), torch.Size([3]), tensor([ 5., 14.]), tensor([ 5., 14.]))
```

2.3.10. Matrix–Matrix Multiplication

```
B = torch.ones(3, 4)
torch.mm(A, B), A@B
```

```
(tensor([[ 3.,  3.,  3.,  3.],
         [12., 12., 12., 12.]]),
 tensor([[ 3.,  3.,  3.,  3.],
         [12., 12., 12., 12.]]))
```

2.3.11 Norms

```
u = torch.tensor([3.0, -4.0])
torch.norm(u)
```

```
tensor(5.)
```

```
torch.abs(u).sum()
```

```
tensor(7.)
```

```
torch.norm(torch.ones((4, 9)))
```

```
tensor(6.)
```

2.3.12 Discussion

2.5 Automatic Differentitation

2.5.1 A Simple Function

```
x = torch.arange(4.0)
x
```

```
tensor([0., 1., 2., 3.])
```

```
x.requires_grad_(True)
x.grad
```

```
y = 2 * torch.dot(x, x)
y
```

```
tensor(28., grad_fn=<MulBackward0>)
```

```
y.backward()
x.grad
```

```
tensor([ 0.,  4.,  8., 12.])
```

```
x.grad == 4 * x
```

```
tensor([True, True, True, True])
```

```
x.grad.zero_()
y = x.sum()
y.backward()
x.grad
```

```
tensor([1., 1., 1., 1.])
```

2.5.2 Backward for Non-Sclar Variables

```
x.grad.zero_()
y = x * x
y.backward(gradient=torch.ones(len(y)))
x.grad
```

```
tensor([0., 2., 4., 6.])
```

2.5.3 Detaching Computation

```
x.grad.zero_()
y = x * x
u = y.detach()
z = u * x

z.sum().backward()
x.grad == u
```

```
tensor([True, True, True, True])
```

```
x.grad.zero_()
y.sum().backward()
x.grad == 2 * x
```

```
tensor([True, True, True, True])
```

2.5.4 Gradients and Python Control Flow

```
def f(a):
    b = a * 2
    while b.norm() < 1000:
        b = b * 2
    if b.sum() > 0:
        c = b
    else:
        c = 100 * b
    return c
```

```
a = torch.randn(size=(), requires_grad=True)
d = f(a)
d.backward()
```

```
a.grad == d / a
```

```
tensor(True)
```

2.5.5 Discusion

코딩을 시작하거나 AI로 코드를 생성하세요.

3.1 Linear Regression

```
%matplotlib inline
import math
import time
import numpy as np
import torch
from d2l import torch as d2l
```

3.1.2 Vectorization for Speed

```
n = 10000
a = torch.ones(n)
b = torch.ones(n)
```

```
c = torch.zeros(n)
t = time.time()
for i in range(n):
    c[i] = a[i] + b[i]
f'{time.time() - t:.5f} sec'
```

```
'0.65555 sec'
```

```
t = time.time()
d = a + b
f'{time.time() - t:.5f} sec'
```

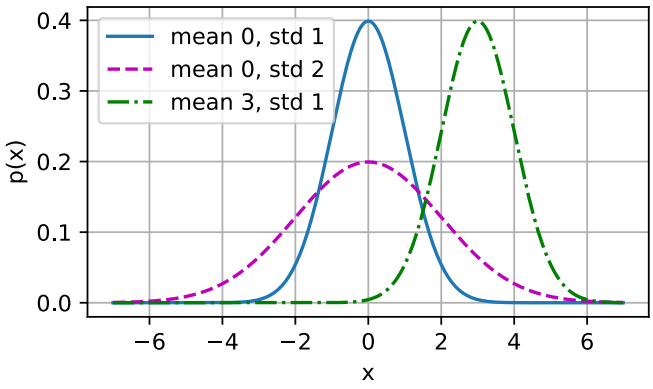
```
'0.00029 sec'
```

3.1.3 The Normal Distribution and Squared Loss

```
def normal(x, mu, sigma):
    p = 1 / math.sqrt(2 * math.pi * sigma**2)
    return p * np.exp(-0.5 * (x - mu)**2 / sigma**2)
```

```
x = np.arange(-7, 7, 0.01)

params = [(0, 1), (0, 2), (3, 1)]
d2l.plot(x, [normal(x, mu, sigma) for mu, sigma in params], xlabel='x',
```

```
        ylabel='p(x)', figsize=(4.5, 2.5),
        legend=[f'mean {mu}, std {sigma}' for mu, sigma in params])
```



3.1.4 Summary

3.2 Object-Oriented Design for Implementation

```
import time
import numpy as np
import torch
from torch import nn
from d2l import torch as d2l
```

3.2.1 Utilities

```
def add_to_class(Class):
    """Register functions as methods in created class."""
    def wrapper(obj):
        setattr(Class, obj.__name__, obj)
    return wrapper


class A:
    def __init__(self):
        self.b = 1

a = A()


@add_to_class(A)
def do(self):
    print('Class attribute "b" is', self.b)

a.do()
```

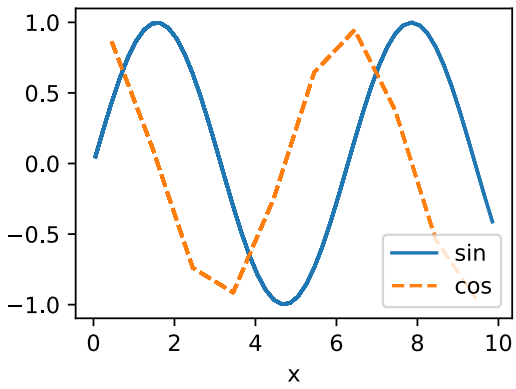
```
Class attribute "b" is 1
```

```
class HyperParameters:
    """The base class of hyperparameters."""
    def save_hyperparameters(self, ignore=[]):
        raise NotImplemented
```

```
class B(d2l.HyperParameters):
    def __init__(self, a, b, c):
        self.save_hyperparameters(ignore=['c'])
        print('self.a =', self.a, 'self.b =', self.b)
        print('There is no self.c =', not hasattr(self, 'c'))

b = B(a=1, b=2, c=3)
```

```
self.a = 1 self.b = 2
There is no self.c = True
```

```
class ProgressBoard(d2l.HyperParameters):
    """The board that plots data points in animation."""
    def __init__(self, xlabel=None, ylabel=None, xlim=None,
                 ylim=None, xscale='linear', yscale='linear',
                 ls=['-', '--', '-.', ':'], colors=['C0', 'C1', 'C2', 'C3'],
                 fig=None, axes=None, figsize=(3.5, 2.5), display=True):
        self.save_hyperparameters()

    def draw(self, x, y, label, every_n=1):
        raise NotImplemented


board = d2l.ProgressBoard('x')
for x in np.arange(0, 10, 0.1):
    board.draw(x, np.sin(x), 'sin', every_n=2)
    board.draw(x, np.cos(x), 'cos', every_n=10)
```



3.2.2 Models

```
class Module(nn.Module, d2l.HyperParameters):
    """The base class of models."""
    def __init__(self, plot_train_per_epoch=2, plot_valid_per_epoch=1):
        super().__init__()
        self.save_hyperparameters()
        self.board = ProgressBoard()

    def loss(self, y_hat, y):
        raise NotImplementedError

    def forward(self, X):
        assert hasattr(self, 'net'), 'Neural network is defined'
        return self.net(X)

    def plot(self, key, value, train):
        """Plot a point in animation."""
        assert hasattr(self, 'trainer'), 'Trainer is not inited'
        self.board.xlabel = 'epoch'
```

```
            if train:
                x = self.trainer.train_batch_idx / \
                    self.trainer.num_train_batches
                n = self.trainer.num_train_batches / \
                    self.plot_train_per_epoch
            else:
                x = self.trainer.epoch + 1
                n = self.trainer.num_val_batches / \
                    self.plot_valid_per_epoch
            self.board.draw(x, value.to(d2l.cpu()).detach().numpy(),
                            ('train_' if train else 'val_') + key,
                            every_n=int(n))

        def training_step(self, batch):
            l = self.loss(self(*batch[:-1]), batch[-1])
            self.plot('loss', l, train=True)
            return l

        def validation_step(self, batch):
            l = self.loss(self(*batch[:-1]), batch[-1])
            self.plot('loss', l, train=False)

        def configure_optimizers(self):
            raise NotImplementedError
```

3.2.3 Data

```
class DataModule(d2l.HyperParameters):
    """The base class of data."""
    def __init__(self, root='../data', num_workers=4):
        self.save_hyperparameters()

    def get_dataloader(self, train):
        raise NotImplementedError

    def train_dataloader(self):
        return self.get_dataloader(train=True)

    def val_dataloader(self):
        return self.get_dataloader(train=False)
```

3.2.4 Training

```
class Trainer(d2l.HyperParameters):
    """The base class for training models with data."""
    def __init__(self, max_epochs, num_gpus=0, gradient_clip_val=0):
        self.save_hyperparameters()
        assert num_gpus == 0, 'No GPU support yet'

    def prepare_data(self, data):
        self.train_dataloader = data.train_dataloader()
        self.val_dataloader = data.val_dataloader()
        self.num_train_batches = len(self.train_dataloader)
        self.num_val_batches = (len(self.val_dataloader)
                                if self.val_dataloader is not None else 0)

    def prepare_model(self, model):
        model.trainer = self
        model.board.xlim = [0, self.max_epochs]
        self.model = model
```

```
    def fit(self, model, data):
        self.prepare_data(data)
        self.prepare_model(model)
        self.optim = model.configure_optimizers()
        self.epoch = 0
        self.train_batch_idx = 0
        self.val_batch_idx = 0
        for self.epoch in range(self.max_epochs):
            self.fit_epoch()

    def fit_epoch(self):
        raise NotImplementedError
```

3.4 Linear Regressing Implementation from Scratch

3.4.1 Defining the Model

```
class LinearRegressionScratch(d2l.Module):
    """The linear regression model implemented from scratch."""
    def __init__(self, num_inputs, lr, sigma=0.01):
        super().__init__()
        self.save_hyperparameters()
        self.w = torch.normal(0, sigma, (num_inputs, 1), requires_grad=True)
        self.b = torch.zeros(1, requires_grad=True)


@d2l.add_to_class(LinearRegressionScratch)
def forward(self, X):
    return torch.matmul(X, self.w) + self.b
```

3.4.2 Defining the Loss Function

```
@d2l.add_to_class(LinearRegressionScratch)
def loss(self, y_hat, y):
    l = (y_hat - y) ** 2 / 2
    return l.mean()
```

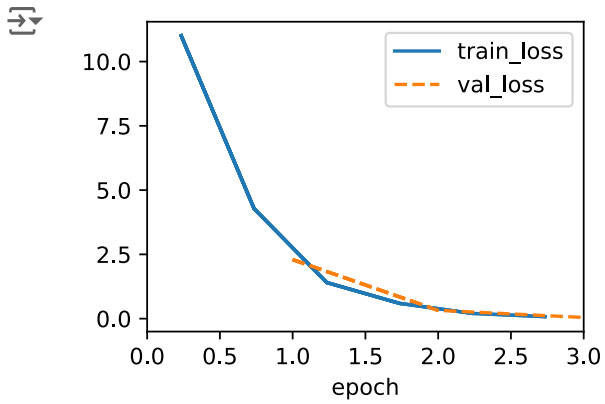
3.4.3 Defining the Optimization Algorithm

```
class SGD(d2l.HyperParameters):
    """Minibatch stochastic gradient descent."""
    def __init__(self, params, lr):
        self.save_hyperparameters()

    def step(self):
        for param in self.params:
            param -= self.lr * param.grad

    def zero_grad(self):
        for param in self.params:
            if param.grad is not None:
                param.grad.zero_()
```

```
@d2l.add_to_class(LinearRegressionScratch)
def configure_optimizers(self):
    return SGD([self.w, self.b], self.lr)
```

3.4.4 Training

```
@d2l.add_to_class(d2l.Trainer)
def prepare_batch(self, batch):
    return batch

@d2l.add_to_class(d2l.Trainer)
def fit_epoch(self):
    self.model.train()
    for batch in self.train_dataloader:
        loss = self.model.training_step(self.prepare_batch(batch))
        self.optim.zero_grad()
        with torch.no_grad():
            loss.backward()
            if self.gradient_clip_val > 0:
                self.clip_gradients(self.gradient_clip_val, self.model)
            self.optim.step()
        self.train_batch_idx += 1
    if self.val_dataloader is None:
        return
    self.model.eval()
    for batch in self.val_dataloader:
        with torch.no_grad():
            self.model.validation_step(self.prepare_batch(batch))
        self.val_batch_idx += 1


model = LinearRegressionScratch(2, lr=0.03)
data = d2l.SyntheticRegressionData(w=torch.tensor([2, -3.4]), b=4.2)
trainer = d2l.Trainer(max_epochs=3)
trainer.fit(model, data)
```




```
with torch.no_grad():
    print(f'error in estimating w: {data.w - model.w.reshape(data.w.shape)}')
    print(f'error in estimating b: {data.b - model.b}')
```

```
error in estimating w: tensor([ 0.0746, -0.1961])
error in estimating b: tensor([0.2051])
```

4.1 Softmax Regression

4.2 The Image Classification Dataset

```
%matplotlib inline
import time
import torch
import torchvision
from torchvision import transforms
from d2l import torch as d2l

d2l.use_svg_display()
```

4.2.1 Loading The Dataset

```
class FashionMNIST(d2l.DataModule):
    """The Fashion-MNIST dataset."""
    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))
len(data.train), len(data.val)
```

```
(60000, 10000)
```

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

```
torch.Size([1, 32, 32])
```

```
@d2l.add_to_class(FashionMNIST)
def text_labels(self, indices):
    """Return text labels."""
    labels = ['t-shirt', 'trouser', 'pullover', 'dress', 'coat',
              'sandal', 'shirt', 'sneaker', 'bag', 'ankle boot']
    return [labels[int(i)] for i in indices]
```

4.2.2 Reading a Minibatch

```
@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)
```

```
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This
  warnings.warn(_create_warning_msg(
torch.Size([64, 1, 32, 32]) torch.float32 torch.Size([64]) torch.int64
```

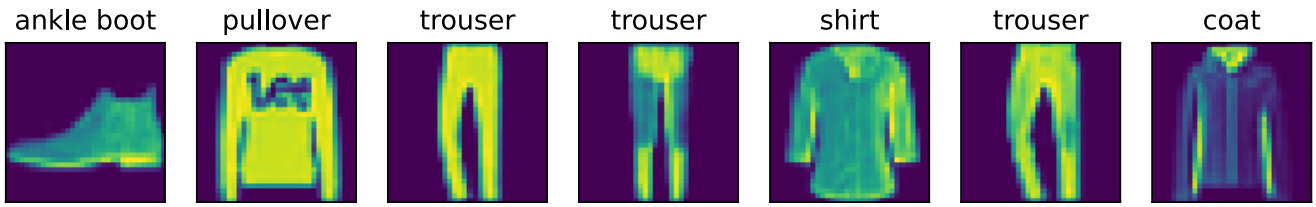
```
tic = time.time()
for X, y in data.train_dataloader():
    continue
f'{time.time() - tic:.2f} sec'
```

```
'13.31 sec'
```

4.2.3 Visualization

```
def show_images(imgs, num_rows, num_cols, titles=None, scale=1.5):
    """Plot a list of images."""
    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)
```



4.3 Th Base Classification Model

4.3.1. The Classifier Class

```
class Classifier(d2l.Module):
    """The base class of classification models."""
    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)
```

4.3.2 Accuracy

```
@d2l.add_to_class(Classifier)
def accuracy(self, Y_hat, Y, averaged=True):
    """Compute the number of correct predictions."""
```

```
    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
```

4.4. Softmax Regression Implementation from Scratch

4.4.1. The Softmax

```
X = torch.tensor([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]])
X.sum(0, keepdims=True), X.sum(1, keepdims=True)
```

```
(tensor([[5., 7., 9.]]),
 tensor([[ 6.],
         [15.]]))
```

```
def softmax(X):
    X_exp = torch.exp(X)
    partition = X_exp.sum(1, keepdims=True)
    return X_exp / partition
```

```
X = torch.rand((2, 5))
X_prob = softmax(X)
X_prob, X_prob.sum(1)
```

```
(tensor([[0.1587, 0.1788, 0.2771, 0.2316, 0.1538],
         [0.1539, 0.2123, 0.2334, 0.2209, 0.1796]]),
 tensor([1., 1.]))
```

4.4.2 The Model

```
class SoftmaxRegressionScratch(d2l.Classifier):
    def __init__(self, num_inputs, num_outputs, lr, sigma=0.01):
        super().__init__()
        self.save_hyperparameters()
        self.W = torch.normal(0, sigma, size=(num_inputs, num_outputs),
                              requires_grad=True)
        self.b = torch.zeros(num_outputs, requires_grad=True)

    def parameters(self):
        return [self.W, self.b]
```

```
@d2l.add_to_class(SoftmaxRegressionScratch)
def forward(self, X):
    X = X.reshape((-1, self.W.shape[0]))
    return softmax(torch.matmul(X, self.W) + self.b)
```

4.4.3. The Cross-Entropy Loss

```
y = torch.tensor([0, 2])
y_hat = torch.tensor([[0.1, 0.3, 0.6], [0.3, 0.2, 0.5]])
y_hat[[0, 1], y]
```

```
tensor([0.1000, 0.5000])
```

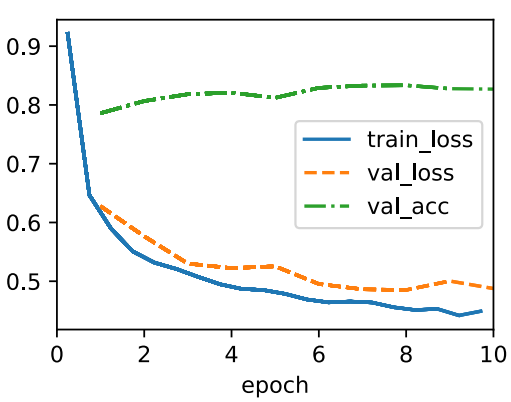
```
def cross_entropy(y_hat, y):
    return -torch.log(y_hat[list(range(len(y_hat))), y]).mean()

cross_entropy(y_hat, y)
```

```
tensor(1.4979)
```

```
@d2l.add_to_class(SoftmaxRegressionScratch)
def loss(self, y_hat, y):
    return cross_entropy(y_hat, y)
```

4.4.4 Training

```
data = d2l.FashionMNIST(batch_size=256)
model = SoftmaxRegressionScratch(num_inputs=784, num_outputs=10, lr=0.1)
trainer = d2l.Trainer(max_epochs=10)
trainer.fit(model, data)
```

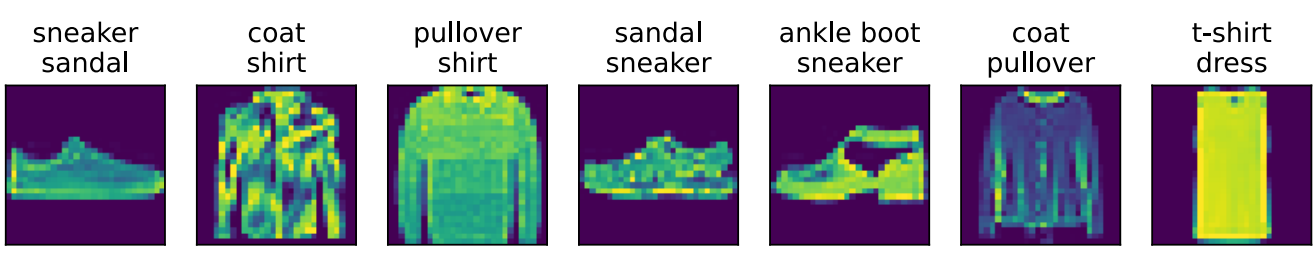


4.4.5 Prediction

```
X, y = next(iter(data.val_dataloader()))
preds = model(X).argmax(axis=1)
preds.shape
```

```
torch.Size([256])
```

```
wrong = preds.type(y.dtype) != y
X, y, preds = X[wrong], y[wrong], preds[wrong]
labels = [a+'\n'+b for a, b in zip(
    data.text_labels(y), data.text_labels(preds))]
data.visualize([X, y], labels=labels)
```



5.1 Multilayer Perceptrons

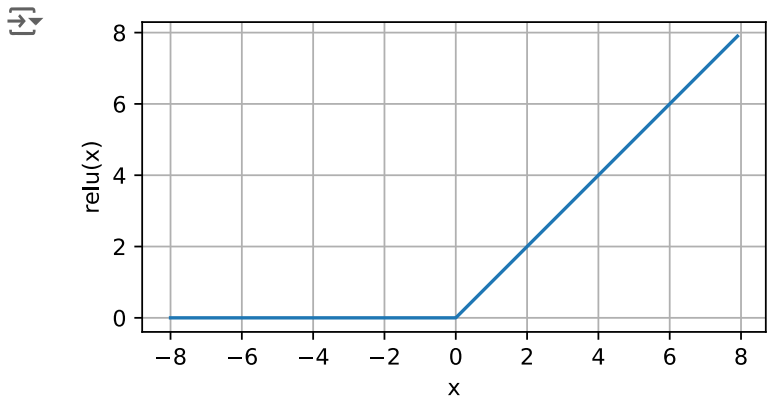
5.1.1 Hidden Layers

5.1.2 Activation Functions

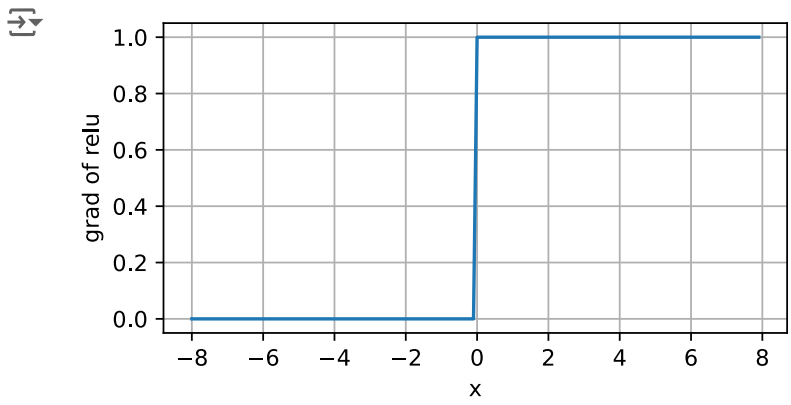
5.1.2.1 ReLU Function

$$ReLU(x) = max(x, 0).$$

```
x = torch.arange(-8.0, 8.0, 0.1, requires_grad=True)
y = torch.relu(x)
d2l.plot(x.detach(), y.detach(), 'x', 'relu(x)', figsize=(5, 2.5))
```



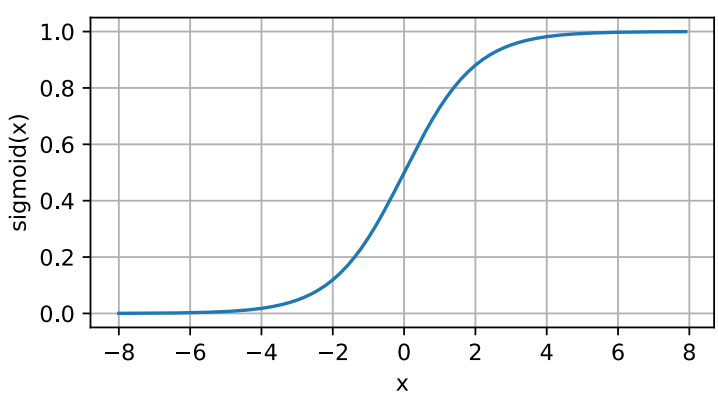

```
y.backward(torch.ones_like(x), retain_graph=True)
d2l.plot(x.detach(), x.grad, 'x', 'grad of relu', figsize=(5, 2.5))
```



5.1.2.2 Sigmoid Function

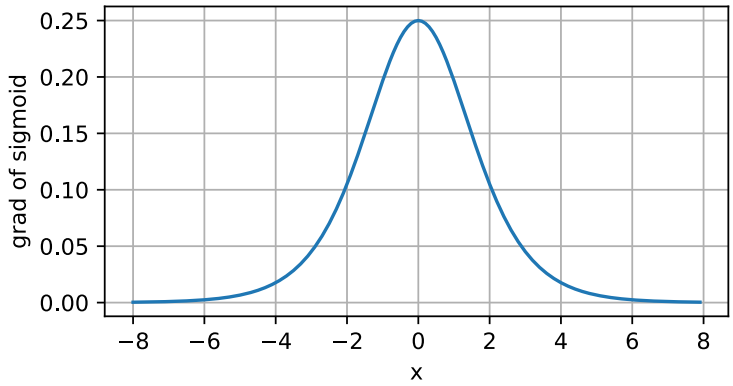
$$sigmoid(x) = \frac{1}{1+exp(-)}$$

```
y = torch.sigmoid(x)
d2l.plot(x.detach(), y.detach(), 'x', 'sigmoid(x)', figsize=(5, 2.5))
```



$\frac{d}{dx}\, sigmoid(x) = \frac{exp(-x)}{(1+exp(-x))^2} = sigmoid(x)(1 - sigmoid(x))$

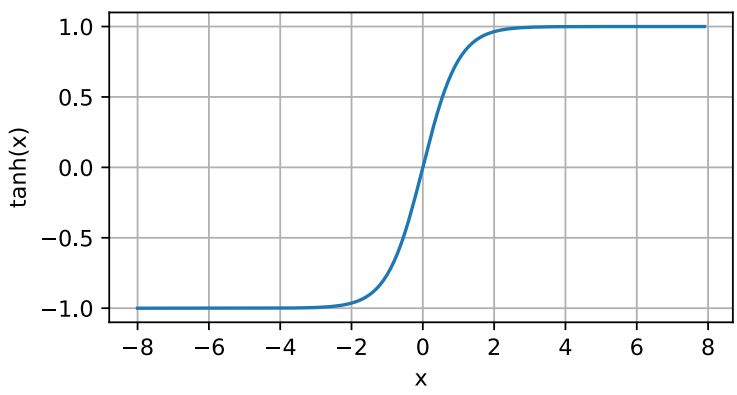
```
x.grad.data.zero_()
y.backward(torch.ones_like(x),retain_graph=True)
d2l.plot(x.detach(), x.grad, 'x', 'grad of sigmoid', figsize=(5, 2.5))
```



5.1.2.3 Tanh Function

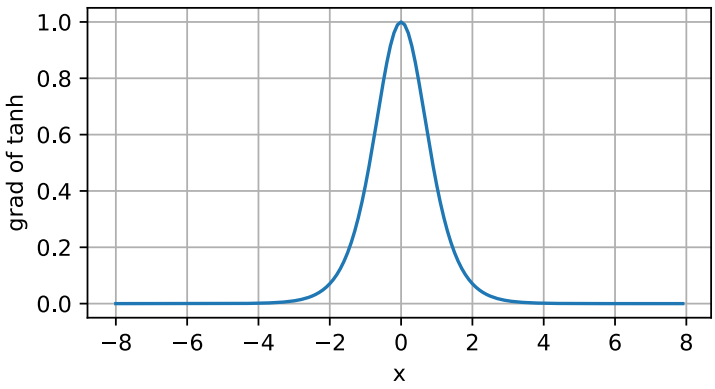
$tanh(x) = \frac{1-exp(-2x)}{1+exp(-2x)}$

```
y = torch.tanh(x)
d2l.plot(x.detach(), y.detach(), 'x', 'tanh(x)', figsize=(5, 2.5))
```



$\frac{d}{dx}\, tanh(x) = 1 - tanh^2(x)$

```
x.grad.data.zero_()
y.backward(torch.ones_like(x),retain_graph=True)
d2l.plot(x.detach(), x.grad, 'x', 'grad of tanh', figsize=(5, 2.5))
```



5.2 Implementation of Multilayer Pereceptrons

```
import torch
from torch import nn
from d2l import torch as d2l
```

5.2.1 Implementation from Scratch

5.2.1.1 Initializing Model Parameters

```
class MLPScratch(d2l.Classifier):
    def __init__(self, num_inputs, num_outputs, num_hiddens, lr, sigma=0.01):
        super().__init__()
        self.save_hyperparameters()
        self.W1 = nn.Parameter(torch.randn(num_inputs, num_hiddens) * sigma)
        self.b1 = nn.Parameter(torch.zeros(num_hiddens))
        self.W2 = nn.Parameter(torch.randn(num_hiddens, num_outputs) * sigma)
        self.b2 = nn.Parameter(torch.zeros(num_outputs))
```

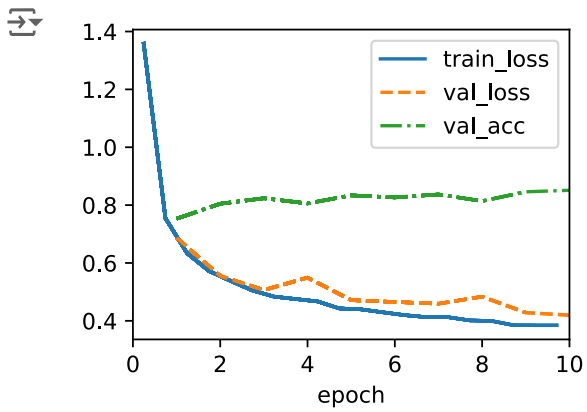
5.2.1.2 Model

```
def relu(X):
    a = torch.zeros_like(X)
    return torch.max(X, a)


@d2l.add_to_class(MLPScratch)
def forward(self, X):
    X = X.reshape((-1, self.num_inputs))
    H = relu(torch.matmul(X, self.W1) + self.b1)
    return torch.matmul(H, self.W2) + self.b2
```

5.2.1.3 Training

```
model = MLPScratch(num_inputs=784, num_outputs=10, num_hiddens=256, lr=0.1)
data = d2l.FashionMNIST(batch_size=256)
```

```
trainer = d2l.Trainer(max_epochs=10)
trainer.fit(model, data)
```



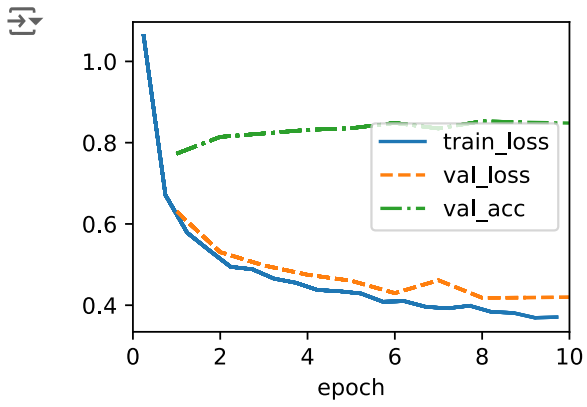
5.2.2 Concise Implementation

5.2.2.1 Model

```
class MLP(d2l.Classifier):
    def __init__(self, num_outputs, num_hiddens, lr):
        super().__init__()
        self.save_hyperparameters()
        self.net = nn.Sequential(nn.Flatten(), nn.LazyLinear(num_hiddens),
                                 nn.ReLU(), nn.LazyLinear(num_outputs))
```

5.2.2.2 Training

```
model = MLP(num_outputs=10, num_hiddens=256, lr=0.1)
trainer.fit(model, data)
```



5.3. Forward Propagation, Backward Propagation, and Computational Graphs

5.3.1 Forward Propagation

5.3.2 Computational Graph of Forward Propagation

5.3.3 BackPropagation

5.3.4 Training Neural Networks

Own Discussion 2019170361 윤동현

3.1.1.2 Loss function

Learned

1. Calculate difference between real value and calculated value
2. Training Goal is minimize Loss function value

Additional

1. Does Loss function represents model's accuracy?
2. Other Loss function
    - MSE(Mean Squared Error)
    - RMSE(Root Mean Squared Error)
    - Binary Crossentropy
    - Categorical Crossentropy
    - Sparse Categorical Crossentropy
    - Focal Loss
    - https://pytorch.org/docs/stable/nn#loss-functions

2.5.1 A Simple Function

y.backward() x.grad

y.backward()를 통해서 y의 기울기를 계산하여 x.grad에 저장한다. 이는 loss function을 최소화 하기 위해 loss function 의 결과값을 backward를 통해 기울기를 계산하여 loss function을 최소화 시킬 것이다.

5.3.3 Backpropagation

모델의 loss function을 최소화 하기 위한 과정 학습 단계에서 Front -> Back -> Front -> Back을 반복하면서 기울기를 최적화 한다.

5.1.2 Activation Function

Learned

1. 비선형성 도입
2. 활성화 여부 결정

비선형이 필요한 이유는 선형문제의 경우 단순 연산으로 복잡성을 추가하기 어렵기에 많은 연산을 하기 위해서는 비선형이 필요하다.

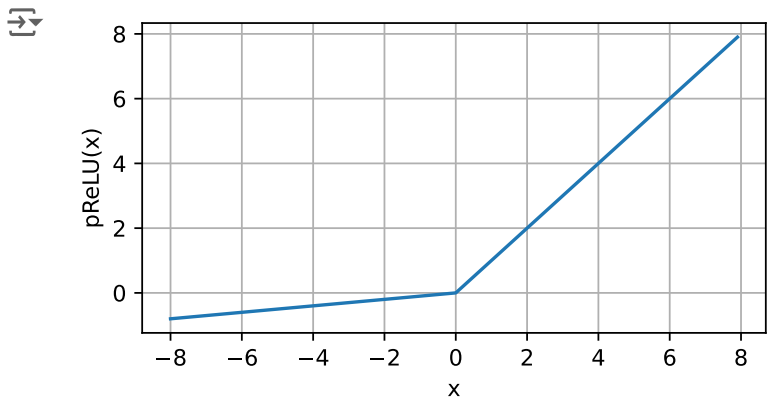
활성함수들 특징

1. ReLU: 0또는 양수 값으로 빠른 계산
2. Sigmoid: 0또는 1로 이진 분류 문제 적합

3. Tanh: -1~1 사이의 출력

5.1.2.4 pReLU Function

```
def pReLU(x, alpha):
  return torch.max(torch.tensor(0), x) + torch.min(torch.tensor(0), x) * alpha

x = torch.arange(-8.0, 8.0, 0.1, requires_grad=True)
y = pReLU(x, 0.1)
d2l.plot(x.detach(), y.detach(), 'x', 'pReLU(x)', figsize=(5, 2.5))
```




```
y.backward(torch.ones_like(x),retain_graph=True)
d2l.plot(x.detach(), x.grad, 'x', 'grad of sigmoid', figsize=(5, 2.5))
```