COSE474-2024F: Deep Learning HW1

> Excercises & Discussions은 챕터별 코드 실습 이후 한 번에 정리해두었습니다.

0.1 Installation

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

```
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib==3.7.2->d2l==1.0.3)
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter==1.0.0->d2l==1.0.3) (0.2.0
Requirement already satisfied: ipython>=5.0.0 in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter==1.0.0->d2l==1.0.3) (7.34.0)
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter==1.0.0->d2l==1.0.3) (6.1.12)
Requirement already satisfied: tornado>=4.2 in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter==1.0.0->d2l==1.0.3) (6.3.3)
Requirement already satisfied: widgetsnbextension~=3.6.0 in /usr/local/lib/python3.10/dist-packages (from ipywidgets->jupyter==1.0.0->d2l==1.0
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from ipywidgets->jupyter==1.0.0->d2l==1.0
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from jupyter-console->
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (from jupyter-console->jupyter==1.0.0->d2l==1.0.3) (2.18.0)
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (4.9.4)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (4.12.3)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (6.1.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (0.7.1)
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (0.4
Requirement already satisfied: jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (3.1.4)
Requirement already satisfied: jupyter-core>=4.7 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (5.7.
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (0.
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (2.1.5)
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (0.8.
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (0.10.0
Requirement already satisfied: nbformat>=5.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (5.10.4)
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (1
Requirement already satisfied: tinycss2 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (1.3.0)
Requirement already satisfied: pyzmq<25,>=17 in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter==1.0.0->d2l==1.0.3) (24.0.1)
Requirement already satisfied: argon2-cffi in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter==1.0.0->d2l==1.0.3) (23.1.0)
Requirement already satisfied: nest-asyncio>=1.5 in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter==1.0.0->d2l==1.0.3) (1.6.0
Requirement already satisfied: Send2Trash>=1.8.0 in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter==1.0.0->d2l==1.0.3) (1.8.3
Requirement already satisfied: terminado>=0.8.3 in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter==1.0.0->d2l==1.0.3) (0.18.1
Requirement already satisfied: prometheus-client in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter==1.0.0->d2l==1.0.3) (0.20.
Requirement already satisfied: nbclassic>=0.4.7 in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter==1.0.0->d2l==1.0.3) (1.1.0)
Requirement already satisfied: qtpy>=2.4.0 in /usr/local/lib/python3.10/dist-packages (from qtconsole->jupyter==1.0.0->d2l==1.0.3) (2.4.1)
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipykernel->jupyter==1.0.0->d2
Requirement already satisfied: jedi>=0.16 in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipykernel->jupyter==1.0.0->d2l==1.0
Requirement already satisfied: decorator in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipykernel->jupyter==1.0.0->d2l==1.0.
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipykernel->jupyter==1.0.0->d2l==1.
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipykernel->jupyter==1.0.0->d2l==1.0.3
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipykernel->jupyter==1.0.0->d2l==1.
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.10/dist-packages (from jupyter-core>=4.7->nbconvert->jupyter==1.0.0
Requirement already satisfied: notebook-shim>=0.2.3 in /usr/local/lib/python3.10/dist-packages (from nbclassic>=0.4.7->notebook->jupyter==1.0.
Requirement already satisfied: fastjsonschema>=2.15 in /usr/local/lib/python3.10/dist-packages (from nbformat>=5.1->nbconvert->jupyter==1.0.0-
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.10/dist-packages (from nbformat>=5.1->nbconvert->jupyter==1.0.0->d2l=
Requirement already satisfied: wcwidth in /usr/local/lib/python3.10/dist-packages (from prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->jupyter-
Requirement already satisfied: ptyprocess in /usr/local/lib/python3.10/dist-packages (from terminado>=0.8.3->notebook->jupyter==1.0.0->d2l==1.
Requirement already satisfied: argon2-cffi-bindings in /usr/local/lib/python3.10/dist-packages (from argon2-cffi->notebook->jupyter==1.0.0->d2
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->nbconvert->jupyter==1.0.0->d2l==
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->nbconvert->jupyter==1.0.0->d2l==1.0.3) (0
Requirement already satisfied: parso<0.9.0,>=0.8.3 in /usr/local/lib/python3.10/dist-packages (from jedi>=0.16->ipython>=5.0.0->ipykernel->jup
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat>=5.1->nbconvert->jupyt
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat>=5.1->nbconvert-
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat>=5.1->nbconvert->jupy
Requirement already satisfied: jupyter-server<3,>=1.8 in /usr/local/lib/python3.10/dist-packages (from notebook-shim>=0.2.3->nbclassic>=0.4.7-
Requirement already satisfied: cffi>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from argon2-cffi-bindings->argon2-cffi->notebook->jupyt
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-packages (from cffi>=1.0.1->argon2-cffi-bindings->argon2-cffi->note
Requirement already satisfied: anyio<4,>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from jupyter-server<3,>=1.8->notebook-shim>=0.2.3->
Requirement already satisfied: websocket-client in /usr/local/lib/python3.10/dist-packages (from jupyter-server<3,>=1.8->notebook-shim>=0.2.3-
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.10/dist-packages (from anyio<4,>=3.1.0->jupyter-server<3,>=1.8->notebook
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages (from anyio<4,>=3.1.0->jupyter-server<3,>=1.8->noteb
```

```
pip install ucimlrepo
```

```
Requirement already satisfied: ucimlrepo in /usr/local/lib/python3.10/dist-packages (0.0.7)
Requirement already satisfied: pandas>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from ucimlrepo) (2.0.3)
Requirement already satisfied: certifi>=2020.12.5 in /usr/local/lib/python3.10/dist-packages (from ucimlrepo) (2024.8.30)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.0->ucimlrepo) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.0->ucimlrepo) (2024.2)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.0->ucimlrepo) (2024.1)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.0->ucimlrepo) (1.23.5)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas>=1.0.0->ucimlrepo) (1.16
```

```
import torch
import torchvision
from torch import nn
from d2l import torch as d2l
from torchvision import transforms

%matplotlib inline
import matplotlib.pyplot as plt

import os
import math
import time
import numpy as np
import pandas as pd

from ucimlrepo import fetch_ucirepo

d2l.use_svg_display()
```

2.1 Data Manipulation

```
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.]])
```

```
X.shape
```

```
torch.Size([3, 4])
```

```
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([[ 0.6818,  0.8824, -0.6560, -1.0802],
        [-0.9979, -0.6502, -0.6389,  0.0887],
        [ 0.8019,  0.1271, -0.1325, -0.8667]])
```

```
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]])
```

```
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.]])
```

```
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.)
```

```
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]])
```

```
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): 132451210201024
id(Z): 132451210201024
```

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

```
True
```

```
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

```
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''')
```

```
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
```

```
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
```

```
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

```
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.))
```

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

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

```
x[2]
```

```
tensor(2)
```

```
len(x)
```

```
3
```

```
x.shape
```

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

```
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]])
```

```
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]]])
```

```
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]))
```

```
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]))
```

```
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.]])
```

```
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.)
```

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

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

```
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.]]))
```

```
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.)
```

2.5 Automatic Differentiation

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

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

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

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

```
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.])
```

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

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

```
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])
```

```
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)
```

3.1 Linear Regression

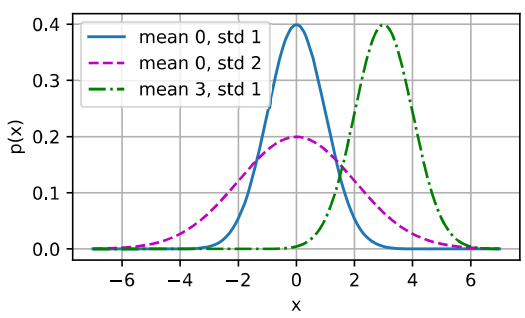
```
n = 10000
a = torch.ones(n)
b = torch.ones(n)


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

```
'0.26372 sec'
```

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

```
'0.00072 sec'
```

```
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, si
```



3.2 Object-Oriented Design for Implementation

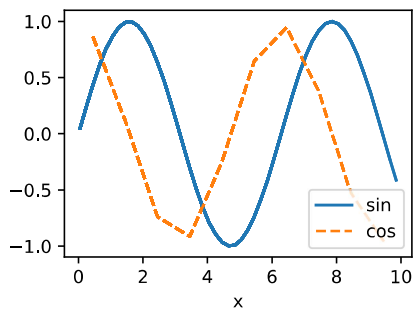
```
def add_to_class(Class):
  def wrapper(obj):
    setattr(Class, obj.__name__, obj)
  return wrapper


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:
  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):
  def __init__(self, xlabel=None, ylabel=None, xlim=None, ylim=None, xscale='linear', yscale='linear', ls=['-', '--', '-.', ':'], colors=['C0', 'C1',
    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)
```



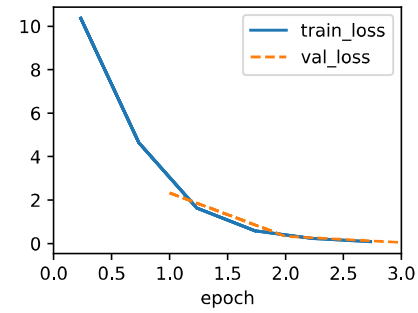

```
class Module(nn.Module, d2l.HyperParameters):
  def __init__(self, plot_train_per_epoch=2, plot_valid_per_epoch=1):
    super().__init__()
    self.save_hyperparameters()
    self.board = ProgressBoard()

  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):
    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
```

```
class DataModule(d2l.HyperParameters):
  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)


class Trainer(d2l.HyperParameters):
  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 Regression Implementation from Scratch

```
class LinearRegressionScratch(d2l.Module):
  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


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


class SGD(d2l.HyperParameters):
  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)
```

```
    return SGD([self.w, self.b], self.lr)

@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:  # To be discussed later
        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([ 0.1257, -0.1725])
error in estimating b: tensor([0.2345])
```

4.1 Softmax Regression

> 이 단원은 코드가 없다고 비워 두기 좀 그러니 수식을 latex로 옮겨 적도록 하겠습니다.

$$\hat{\mathbf{y}} = \mathrm{softmax}(\mathbf{o}) \quad \text{where} \quad \hat{y}_i = \frac{\exp(o_i)}{\sum_j \exp(o_j)}$$

$$\arg\max_j \hat{y}_j = \arg\max_j o_j.$$

$$\mathbf{O} = \mathbf{X}\mathbf{W} + \mathbf{b},$$
$$\hat{\mathbf{Y}} = \mathrm{softmax}(\mathbf{O}).$$

$$P(\mathbf{Y} \mid \mathbf{X}) = \prod_{i=1}^{n} P(y^{(i)} \mid \mathbf{x}^{(i)}).$$

$$-\log P(\mathbf{Y} \mid \mathbf{X}) = \sum_{i=1}^{n} -\log P(y^{(i)} \mid \mathbf{x}^{(i)}) = \sum_{i=1}^{n} l(y^{(i)}, \hat{y}^{(i)}),$$

$$-\log P(\mathbf{Y} \mid \mathbf{X}) = \sum_{i=1}^{n} -\log P(y^{(i)} \mid \mathbf{x}^{(i)}) = \sum_{i=1}^{n} l(y^{(i)}, \hat{y}^{(i)}),$$

$$l(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{j=1}^{q} y_j \log \hat{y}_j.$$

$$\begin{aligned} l(\mathbf{y}, \hat{\mathbf{y}}) &= -\sum_{j=1}^{q} y_j \log \frac{\exp(o_j)}{\sum_{k=1}^{q} \exp(o_k)} \\ &= \sum_{j=1}^{q} y_j \log \sum_{k=1}^{q} \exp(o_k) - \sum_{j=1}^{q} y_j o_j \\ &= \log \sum_{k=1}^{q} \exp(o_k) - \sum_{j=1}^{q} y_j o_j. \end{aligned}$$

$$\frac{\partial l(\mathbf{y}, \hat{\mathbf{y}})}{\partial o_j} = \frac{\exp(o_j)}{\sum_{k=1}^{q} \exp(o_k)} - y_j = \mathrm{softmax}(\mathbf{o})_j - y_j.$$

$$H[P] = \sum_{j} -P(j) \log P(j).$$

4.2 The Image Classification 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))
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):
  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)
```

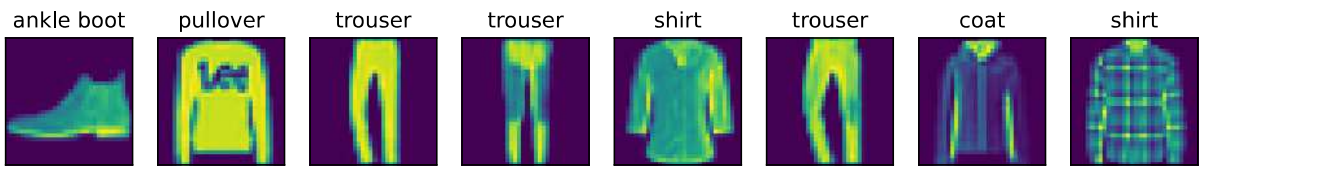
```
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes in total
  warnings.warn(_create_warning_msg(
torch.Size([64, 1, 32, 32]) torch.float32 torch.Size([64]) torch.int64
```

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

```
'14.36 sec'
```

```
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)
```



4.3 The Base Classification Model

```
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
```

4.4 Softmax Regression Implementation from Scratch

```
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([[0.1632, 0.1510, 0.1505, 0.3378, 0.1974],
         [0.2569, 0.2974, 0.1546, 0.1421, 0.1491]]),
 tensor([1.0000, 1.0000]))
```

```
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]
```

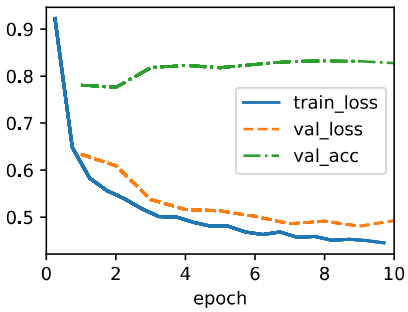
```
@d2l.add_to_class(SoftmaxRegressionScratch)
def forward(self, X):
  X = X.reshape((-1, self.W.shape[0]))
  return softmax(torch.matmul(X, self.W) + self.b)


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)
```

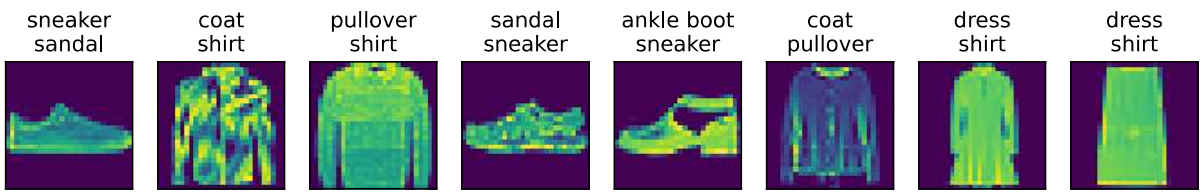
```
@d2l.add_to_class(SoftmaxRegressionScratch)
def loss(self, y_hat, y):
  return cross_entropy(y_hat, y)


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)
```




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

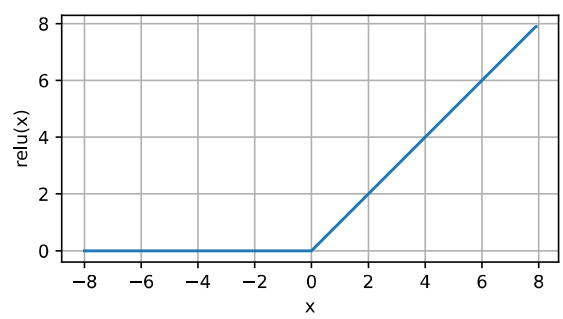
```
torch.Size([256])
```

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

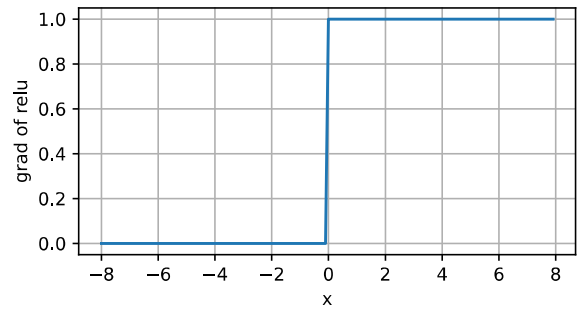


5.1 Multilayer Perceptrons

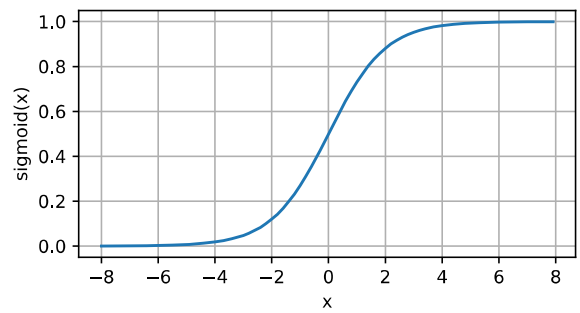
```
x = torch.arange(-8.0, 8.0, 0.1, requires_grad=True)
y = torch.relu(x)
d2l.plot(x.detach(), y.detach(), 'x', 'relu(x)', figsize=(5, 2.5))
```



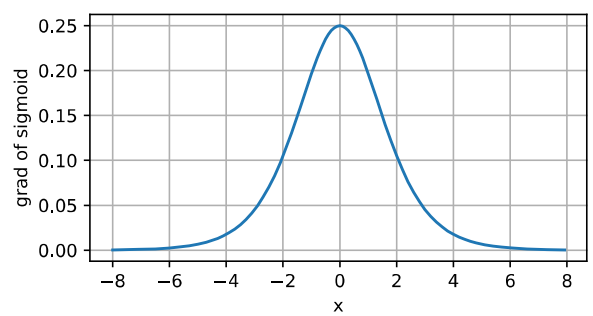

```
y.backward(torch.ones_like(x), retain_graph=True)
d2l.plot(x.detach(), x.grad, 'x', 'grad of relu', figsize=(5, 2.5))
```



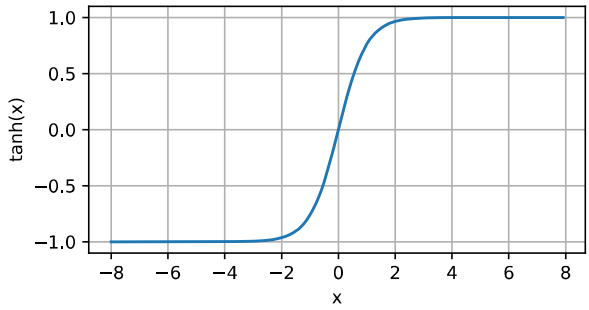

```
y = torch.sigmoid(x)
d2l.plot(x.detach(), y.detach(), 'x', 'sigmoid(x)', figsize=(5, 2.5))
```



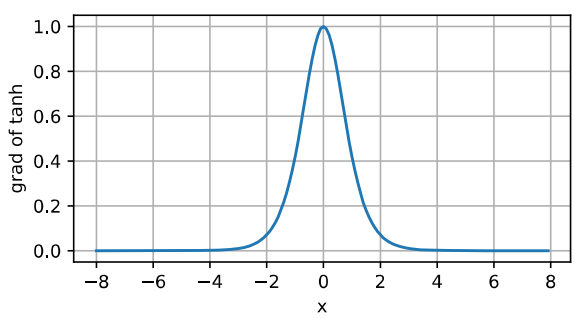

```
x.grad.data.zero_()
y.backward(torch.ones_like(x),retain_graph=True)
d2l.plot(x.detach(), x.grad, 'x', 'grad of sigmoid', figsize=(5, 2.5))
```




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

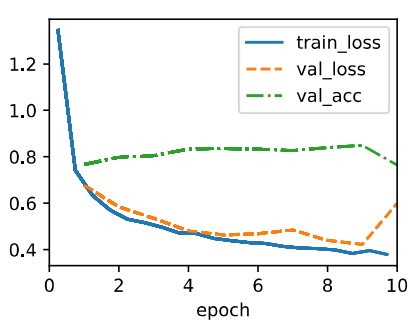



```
x.grad.data.zero_()
y.backward(torch.ones_like(x),retain_graph=True)
d2l.plot(x.detach(), x.grad, 'x', 'grad of tanh', figsize=(5, 2.5))
```



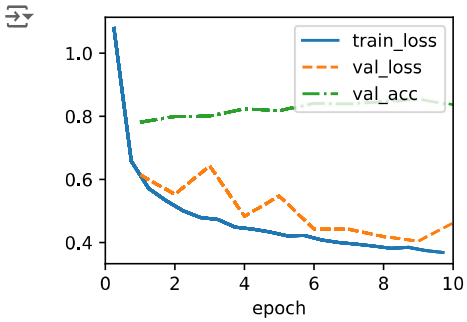
5.2 Implementation of Multilayer Perceptrons

```
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))


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


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)
```




```
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))

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



5.3 Forward Propagation, Backward Propagation, and Computational Graphs

> 이 단원도 코드가 없다고 비워 두기 좀 그러니 수식을 latex로 옮겨 적도록 하겠습니다.

$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x},$$

$$\mathbf{h} = \phi(\mathbf{z}).$$

$$\mathbf{o} = \mathbf{W}^{(2)}\mathbf{h}.$$

$$L = l(\mathbf{o}, \mathbf{y}).$$

$$s = \frac{\lambda}{2}\left(\left\|\mathbf{W}^{(1)}\right\|_F^2 + \left\|\mathbf{W}^{(2)}\right\|_F^2\right),$$

$$J = L + s.$$

$$\frac{\partial \mathbf{Z}}{\partial \mathbf{X}} = \text{prod}\left(\frac{\partial \mathbf{Z}}{\partial \mathbf{Y}}, \frac{\partial \mathbf{Y}}{\partial \mathbf{X}}\right).$$

$$\frac{\partial J}{\partial L} = 1 \quad \text{and} \quad \frac{\partial J}{\partial s} = 1.$$

$$\frac{\partial J}{\partial \mathbf{o}} = \text{prod}\left(\frac{\partial J}{\partial L}, \frac{\partial L}{\partial \mathbf{o}}\right) = \frac{\partial L}{\partial \mathbf{o}} \in \mathbb{R}^q.$$

$$\frac{\partial s}{\partial \mathbf{W}^{(1)}} = \lambda \mathbf{W}^{(1)} \quad \text{and} \quad \frac{\partial s}{\partial \mathbf{W}^{(2)}} = \lambda \mathbf{W}^{(2)}.$$

$$\frac{\partial J}{\partial \mathbf{W}^{(2)}} = \text{prod}\left(\frac{\partial J}{\partial \mathbf{o}}, \frac{\partial \mathbf{o}}{\partial \mathbf{W}^{(2)}}\right) + \text{prod}\left(\frac{\partial J}{\partial s}, \frac{\partial s}{\partial \mathbf{W}^{(2)}}\right) = \frac{\partial J}{\partial \mathbf{o}}\mathbf{h}^\top + \lambda \mathbf{W}^{(2)}.$$

$$\frac{\partial J}{\partial \mathbf{h}} = \text{prod}\left(\frac{\partial J}{\partial \mathbf{o}}, \frac{\partial \mathbf{o}}{\partial \mathbf{h}}\right) = \mathbf{W}^{(2)\top}\frac{\partial J}{\partial \mathbf{o}}.$$

$$\frac{\partial J}{\partial \mathbf{z}} = \text{prod}\left(\frac{\partial J}{\partial \mathbf{h}}, \frac{\partial \mathbf{h}}{\partial \mathbf{z}}\right) = \frac{\partial J}{\partial \mathbf{h}} \odot \phi'(\mathbf{z}).$$

$$\frac{\partial J}{\partial \mathbf{W}^{(1)}} = \text{prod}\left(\frac{\partial J}{\partial \mathbf{z}}, \frac{\partial \mathbf{z}}{\partial \mathbf{W}^{(1)}}\right) + \text{prod}\left(\frac{\partial J}{\partial s}, \frac{\partial s}{\partial \mathbf{W}^{(1)}}\right) = \frac{\partial J}{\partial \mathbf{z}}\mathbf{x}^\top + \lambda \mathbf{W}^{(1)}.$$

Excercises & Discussions

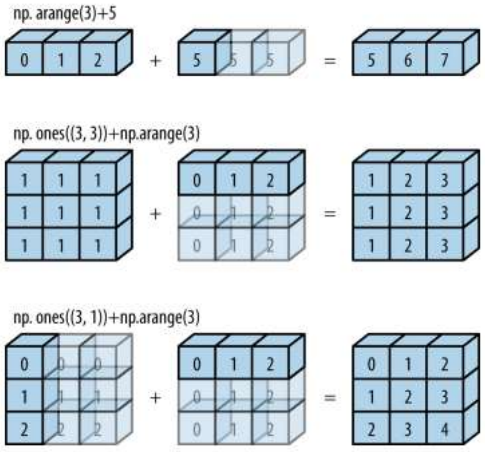
2.1

> Run the code in this section. Change the conditional statement X == Y to X < Y or X > Y, and then see what kind of tensor you can get.
> 합리적인 결과가 나왔다. '<'도 bool값을 반환한다는 점에서 '=='와 큰 차이가 없으니까.

```
X_2_1 = torch.arange(12, dtype=torch.float32).reshape((3,4))
Y_2_1 = torch.tensor([[2.0, 1, 4, 3], [1, 2, 3, 4], [4, 3, 2, 1]])
X_2_1 < Y_2_1, X_2_1 > Y_2_1
```

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

> Replace the two tensors that operate by element in the broadcasting mechanism with other shapes, e.g., 3-dimensional tensors. Is the result the same as expected?
> 처음에는 +의 조건이 잘 이해가 가지 않았으나 전공서의 그림을 보고 이해할 수 있었다.




```
a_2_1 = torch.arange(6).reshape(1, 2, 3)
b_2_1 = torch.arange(8).reshape(4, 2, 1)
a_2_1, b_2_1, a_2_1 + b_2_1
```

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

         [[2],
          [3]],

         [[4],
          [5]],

         [[6],
          [7]]]),
 tensor([[[ 0,  1,  2],
          [ 4,  5,  6]],

         [[ 2,  3,  4],
          [ 6,  7,  8]],

         [[ 4,  5,  6],
          [ 8,  9, 10]],

         [[ 6,  7,  8],
          [10, 11, 12]]]))
```

2.2

> Try loading datasets, e.g., Abalone from the UCI Machine Learning Repository and inspect their properties. What fraction of them has missing values? What fraction of the variables is numerical, categorical, or text?

```
abalone = fetch_ucirepo(id=1)
for i in abalone.data.keys():
  if type(abalone.data[i]) == pd.core.frame.DataFrame:
    size_2_2 = abalone.data[i].size
    isnull_2_2 = abalone.data[i].isnull().sum().sum()
    number_2_2 = abalone.data[i].select_dtypes(include=['number']).size
    category_2_2 = abalone.data[i].select_dtypes(include=['category']).size
    object_2_2 = abalone.data[i].select_dtypes(include=['object']).size
    print("데이터프레임 abalone.data['%s']에는 누락된 값 %d개, 숫자형 변수 %d개, 범주형 변수 %d개, 문자형 변수 %d개로 총 변수 %d개가 있습니다.₩n"%(i,
```

```
데이터프레임 abalone.data['features']에는 누락된 값 0개, 숫자형 변수 29239개, 범주형 변수 0개, 문자형 변수 4177개로 총 변수 33416개가 있습니다.

데이터프레임 abalone.data['targets']에는 누락된 값 0개, 숫자형 변수 4177개, 범주형 변수 0개, 문자형 변수 0개로 총 변수 4177개가 있습니다.

데이터프레임 abalone.data['original']에는 누락된 값 0개, 숫자형 변수 33416개, 범주형 변수 0개, 문자형 변수 4177개로 총 변수 37593개가 있습니다.
```

> How large a dataset do you think you could load this way? What might be the limitations?
> Hint: consider the time to read the data, representation, processing, and memory footprint. Try this out on your laptop. What happens if you try it out on a server?
>
> 이전에 프로젝트를 진행할 때 47304000 rows * 6 cols 데이터프레임에 과거 1000 기반 lstm을 진행하다 colab a100에서 감당하지 못하고 터졌던 경험이 있는 것으로 보아(노트북 기반 환경에선 당연히 실패)RAM의 크기에 따라 다르다. 물론 데이터의 type도 영향이 있을 것이다.

> How would you deal with data that has a very large number of categories?
> What if the category labels are all unique?
> Should you include the latter?
>
> 물론 데이터의 종류와 그걸 이용해 도출해 내고 싶은 목표와 방법론에 따라 달라질 수 있는 문제지만, 개인적으로 너무 많은 categories가 있을 경우 하나 하나 처리하는 건 비효율적이라고 생각한다.
> groupby()를 통해 유의미한 결과를 낼 수 없을 정도로 많은 가짓수가 존재한다면,(위에서 말한 것처럼 전부 unique한 값인 경우처럼) 오히려 배제하거나 적당한 규칙성을 찾아 적용하는 편이 맞다고 생각한다.
> (예시로, id값은 모두 unique하나 보통 데이터베이스에 저장된 순서대로 오름차순이므로 일정 구간별로 잘라 처리한다면 가입한 시간 대비 이용자들의 경향성 변화를 관측할 수 있을 지도 모른다.)

2.3

> Prove that the transpose of the transpose of a matrix is the matrix itself:
> $$(A^T)^T = A$$
>
> A를 m x n 행렬이라 할 때, A의 요소는 a_{ij}의 형식으로(a_{ij}는 i행 j열에 존재한다는 뜻)주어지며, 전치행렬 A^T는 열과 행이 바뀐 구조이기에 a_{ji}의 값들을 가진다. 그 전치행렬의 전치행렬은 다시 a_{ij}의 값을 가지기에, 모든 구성 요소의 크기와 위치가 동일하므로 전치행렬의 전치행렬은 그 스스로와 같다.

```
A_2_3 = torch.randn(3, 4)

A_2_3 == (A_2_3.T).T
```

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

> Given two matrices A and B, show that sum and transposition commute:
> $$A^T + B^T = (A + B)^T$$
>
> (A와 B가 모두 m x n 행렬이라는 가정 하에) $A + B = C$라 가정한다면, C의 요소 c_{ij}는 A와 B의 요소인 a_{ij}, b_{ij}의 합과 같으며, 그 전치행렬은 $c_{ji} = a_{ji} + b_{ji}$의 요소를 가진다.

> 또한, A의 전치행렬과 B의 전치행렬의 합은 $a_{ji} + b_{ji}$의 요소를 가지기에, 크기와 위치가 동일하므로 두 행렬의 전치행렬의 합은 그 두 행렬의 합의 전치행렬과 같다.

```
A_2_3 = torch.randn(3, 4)
B_2_3 = torch.randn(3, 4)

(A_2_3.T + B_2_3.T) == (A_2_3 + B_2_3).T
```

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

> Given any square matrix A, is $A + A^T$ always symmetric? Can you prove the result by using only the results of the previous two exercises?
>
> symmetric하기 위해선, 다음을 만족해야 한다:
>
> $$A + A^T = (A + A^T)^T$$
>
> 이 때, 두 행렬의 합의 전치행렬은 전치행렬의 합과 같으므로, 다음과 같으며:
>
> $$A + A^T = (A + A^T)^T = A^T + (A^T)^T$$
>
> 전치행렬의 전치행렬은 그 스스로와 같으므로 다음과 같다:
>
> $$A + A^T = (A + A^T)^T = A^T + (A^T)^T = A^T + A$$
>
> 따라서, 모든 조건 하에서 한 사각행렬과 그 역행렬의 합은 symmetric하다.

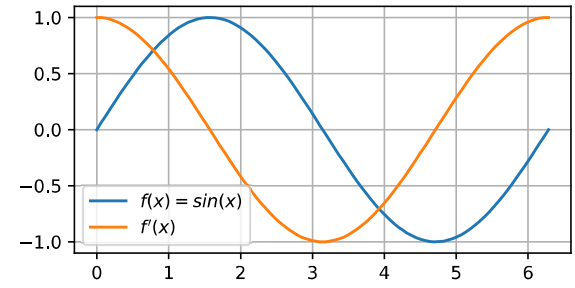
```
A_2_3 = torch.randn(3, 3)
(A_2_3 + A_2_3.T) == (A_2_3 + A_2_3.T).T
```

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

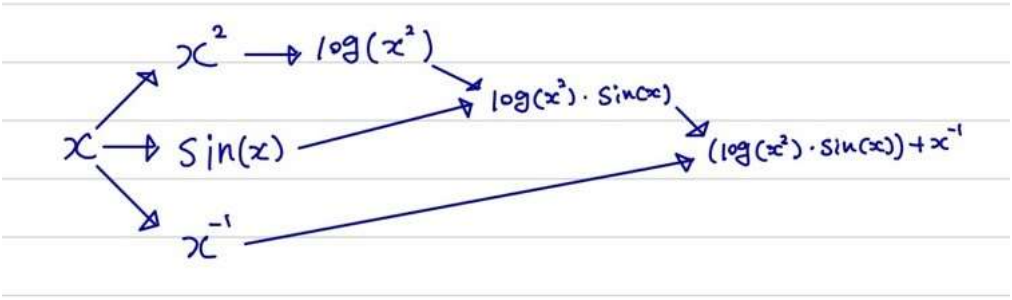
2.5

> Let $f(x) = sin(x)$. Plot the graph of f and of its derivative f'. Do not exploit the fact that $f'(x) = cos(x)$ but rather use automatic differentiation to get the result.
>
> $sin(x)$를 코드를 이용해 미분하고 시각화한 결과 $cos(x)$와 같은 값이 나옴을 확인할 수 있다.

```
def f(x):
  return torch.sin(x)

x_2_5 = torch.linspace(0, 2 * np.pi, 100, requires_grad=True)
y_2_5 = f(x_2_5)

y_2_5.backward(torch.ones_like(x_2_5))

dy_dx_np = x_2_5.grad.numpy()
x_np = x_2_5.detach().numpy()
y_np = y_2_5.detach().numpy()

plt.plot(x_np, y_np, label='$f(x) = sin(x)$')
plt.plot(x_np, dy_dx_np, label="$f'(x)$")
plt.legend()
plt.grid(True)
plt.show()
```



Let $f(x) = ((log(x^2)) * sin(x)) + x^{-1}$. Write out a dependency graph tracing results from x to $f(x)$



3.1

Assume that we have some data $x_1, \ldots, x_n \in R$, Our goal is to find a constant b such that $\sum_i (x_i - b)^2$ is minimized.

주어진 손실 함수 $L(b) = \sum_i (x_i - b)^2$는 b에 대해 미분할 시 $-2\sum_i (x_i - b)$의 값을 가지며, 이는 b가 x의 평균값을 가질 시 가장 작은 손실값을 가진다.

또, 손실 함수 $L(b)$가 $\sum_i |x_i - b|$로 바뀐다면, b에 대해 미분할 시 다음과 같은 손실값을 가진다.

$$\sum_i \left(\begin{cases} 1 & x_i < b \\ -1 & otherwise \end{cases} \right)$$

이 경우 b는 x의 중앙값에서 가장 작은 손실값을 가진다.

3.2

Remove the save_hyperparameters statement in the B class. Can you still print self.a and self.b? Optional: if you have dived into the full implementation of the HyperParameters class, can you explain why?

```
[10] class B(d2l.HyperParameters):
       def __init__(self, a, b, 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)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-10-c900fb1db10d> in <cell line: 6>()
      4     print('There is no self.c =', not hasattr(self, 'c'))
      5
----> 6 b = B(a=1, b=2, c=3)

<ipython-input-10-c900fb1db10d> in __init__(self, a, b, c)
      1 class B(d2l.HyperParameters):
      2   def __init__(self, a, b, c):
----> 3     print('self.a =', self.a, 'self.b =', self.b)
      4     print('There is no self.c =', not hasattr(self, 'c'))
      5

AttributeError: 'B' object has no attribute 'a'
```

> save_hyperparameters는 주로 클래스의 인스턴스 변수를 저장하는 역할을 한다. 즉, 전달받은 매개변수를 인스턴스의 속성으로 자동으로 저장하는 메서드로, 개발자는 일일이 self.a = a와 같이 초기화할 필요 없이, 한 번의 호출로 해당 인스턴스 변수들을 저장할 수 있기에, save_hyperparameters가 없는 한 self.a 값을 따로 지정해 주어야 하며, 그렇지 않을 경우 위와 같이 에러가 난다.

3.4

> Why is the reshape method needed in the loss function?
>
> reshape method는 loss function에서 $\hat{y}$와 y의 구조를 맞추기 위해 존재(한다고 알고 있습니다)하며, 만일 사용하지 않을 경우 손실함수값 연산이 불가능할 수 있다.

> Why do we need to reshuffle the dataset? Can you design a case where a maliciously constructed dataset would break the optimization algorithm otherwise?
>
> dataset을 reshuffling하는 건 모델의 학습 과정 중 편향되는 구간이 생겨 학습이 원활이 이루어지지 않는 상황을 방지하기 위해서이다. 예를 들어, 2개의 class(A와 B)가 있다고 할 때, class A, class A, class A, class A, class A, ..., class B, class B, class B, class B와 같은 전혀 shuffle되지 않은 데이터셋의 경우 순서대로 학습할 시 가중치 업데이트가 편향적으로 일어날 수 밖에 없을 것이다.

4.1

> Softmax gets its name from the following mapping:
>
> $$RealSoftMax(a,b) = log(exp(a) + exp(b)).$$

> Prove that $RealSoftMax(a,b) > max(a,b)$
>
> 만일 $a = b$라면, $RealSoftMax(a,b) = a + log(2) > a$이며,
> $a > b$라면, $RealSoftMax(a,b) = a + log(1 + exp(b-a)) > a + log(1) = a$이다.
> $b > a$일때도 위와 같다.

> How small can you make the difference between both functions?
>
> $min(|RealSoftMax(a,b) - max(a,b)|)$를 위해 $b = 0, a > b$라 둔다면,
> $RealSoftMax(a,b) - max(a,b) = log(1 + exp(-a))$가 되며, a가 양의 무한대로 발산할 때 이 값은 0에 수렴한다.

4.2

> Does reducing the batch_size (for instance, to 1) affect the reading performance?
> batch_size를 기본값(64)의 0.5배수/2배수로 적용하였을 때, 각각 기존 batch_size보다 느려지는/빨라지는 것을 볼 수 있었다.

```
data = FashionMNIST(resize=(32, 32), batch_size = 32)
X, y = next(iter(data.train_dataloader()))
tic = time.time()
for X, y in data.train_dataloader():
  continue
f'{time.time() - tic:.2f} sec'
```

```
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes in total
  warnings.warn(_create_warning_msg(
'18.35 sec'
```

```
#기본 수치
data = FashionMNIST(resize=(32, 32), batch_size = 64)
X, y = next(iter(data.train_dataloader()))
tic = time.time()
for X, y in data.train_dataloader():
  continue
f'{time.time() - tic:.2f} sec'
```

```
'14.79 sec'
```

```
data = FashionMNIST(resize=(32, 32), batch_size = 128)
X, y = next(iter(data.train_dataloader()))
tic = time.time()
for X, y in data.train_dataloader():
  continue
f'{time.time() - tic:.2f} sec'
```

```
'12.51 sec'
```

4.3

> Denote by L_v the validation loss, and let L_v^q be its quick and dirty estimate computed by the loss function averaging in this section. Lastly, denote by l_v^q the loss on the last minibatch. Express L_v in terms of L_v^q, l_v^q, and the sample and minibatch sizes.

$$L_\mathcal{V} = \frac{1}{N}\sum_{i=1}^{N} l_\mathcal{V}^i$$

$$L_\mathcal{V} = \frac{1}{M}\sum_{i=1}^{M} L_\mathcal{V}^q + \frac{1}{N} l_\mathcal{V}^b$$

> Show that the quick and dirty estimate L_v^q is unbiased. That is, show that $E[L_v] = E[L_v^q]$. Why would you still want to use L_v instead?
>
> L_v^q는 특정 샘플 집합의 검증 손실 추정치이다. 따라서 전체 검증 손실 추정치에 대해 unbiased하다고 볼 수 있을 것이다. 또한, 모든 특정 샘플 집합의 검증 손실 추정치의 평균값은 전체 샘플 집합의 검증 손실 추정치와 같다. 하지만 그럼에도 L_v를 사용하는 이유는 작은 오차조차 줄이기 위해서일 것이다.

4.4

> Is it always a good idea to return the most likely label? For example, would you do this for medical diagnosis? How would you try to address this?
>
> medical problem이라는 고전적 예제를 포함한 많은 real world problem에서, most likely라는 애매모호한 기준으로는 한참 부족할 때가 있다. 틀린 선택일 때의 리스크 대비 성공 확률이라는 문제와, 위양성 위음성 같은 class 불균형 문제까지 존재한다. 대표적인 해결 방법으론 threshold를 적용하는 방법이 존재한다.

> Assume that we want to use softmax regression to predict the next word based on some features. What are some problems that might arise from a large vocabulary?
>
> softmax로 자연어처리를 하는 건 약간의 problems이 있을 수 있다. 단어의 수많은 가짓수와 조사의 결합 등으로 생기는 수많은 class가 생기며, bert처럼 tokenizer를 적용하지 못하기에 시간이 오래 걸리고, 등장 빈도가 낮은 단어에 대해서는 제 효과를 내기 힘들 것이다.

5.1

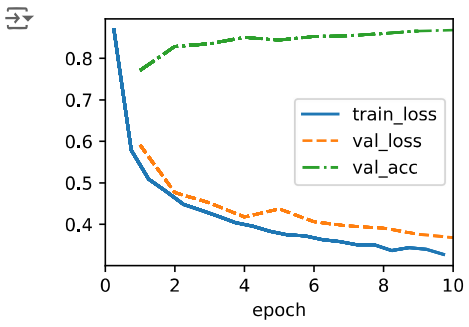
> show that $tanh(x) + 1 = 2 \times sigmoid(2x)$
>
> $$tanh(x) + 1 = \frac{e^x - e^{-x}}{e^x + e^{-x}} + 1 = \frac{2e^x}{e^x + e^{-x}}$$
>
> $$2 \times sigmoid(2x) = \frac{2}{1 + e^{-2x}} = \frac{2e^x}{e^x + e^{-x}}$$

> Prove that the function classes parametrized by both nonlinearities are identical. Hint: affine layers have bias terms, too.
>
> 위에서 본 것처럼, $sigmoid(x) = \frac{1}{2}(tanh(\frac{x}{2}) + 1)$이기에,
>
> $$f_{sig}(x) = sigmoid(Wx + b) = \frac{1}{2}(tanh(\frac{Wx + b}{2}) + 1)$$
> $$= \frac{1}{2}(tanh(\frac{W}{2}x + \frac{b}{2}) + 1)$$
>
> scaling 과 shifting의 상관관계를 가진다(다시 말해, 가중치와 bias의 차이만을 보인다).

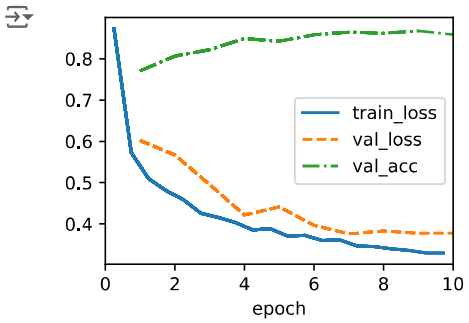
5.2

> Change the number of hidden units num_hiddens and plot how its number affects the accuracy of the model

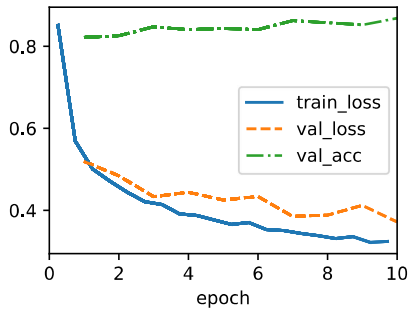
```
model = MLP(num_outputs=10, num_hiddens=128, lr=0.1)
trainer.fit(model, data)
```




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



```
model = MLP(num_outputs=10, num_hiddens=512, lr=0.1)
trainer.fit(model, data)
```



> Why is it a bad idea to insert a hidden layer with a single neuron? What could go wrong?
> 병목 현상이 일어날 것이다. 히든 레이어의 뉴런 갯수가 부족하면 주어진 데이터를 온전히 표현하기 힘들기에 전체적인 능력 저하가 일어나기에, 단 하나뿐인 뉴런을 가지는 레이어가 추가된다면 성능이 심각하게 감소할 것이다.

5.3

> Assume that the inputs X to some scalar function f are $n \times m$ matrices. What is the dimensionality of the gradient of f with respect to X?
>
> $n \times m$ matrices의 각 성분 x_{ij}에 대해 f의 편미분은 계산 가능하다. f는 스칼라 값을 출력하므로, 그 gradient는 행렬이고, X와 동일한 차원을 가진다.

Additional

> 왜 PyTorch는 float, int가 아니라 torch.float, torch.int를 사용하는 것일까?
>
> torch.float는 32비트 부동소수점 자료형으로 64비트 부동소수점이 기준인 float와는 달리 GPU와 같은 하드웨어에서 효율적으로 연