Deep Learning Assignment1

0.1.Installation

```
import torch
```

```
! pip install d2l
```

```
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/pytho
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3
Requirement already satisfied: ipython>=5.0.0 in /usr/local/lib/python3.1
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.1
Requirement already satisfied: tornado>=4.2 in /usr/local/lib/python3.10/
Requirement already satisfied: widgetsnbextension~=3.6.0 in /usr/local/li
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /usr/local/li
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.1
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/di
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/pytho
Requirement already satisfied: jinja2>=3.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: jupyter-core>=4.7 in /usr/local/lib/python
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/pyth
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.
Requirement already satisfied: nbformat>=5.1 in /usr/local/lib/python3.10
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/pyt
Requirement already satisfied: tinycss2 in /usr/local/lib/python3.10/dist
Requirement already satisfied: pyzmq<25,>=17 in /usr/local/lib/python3.10
Requirement already satisfied: argon2-cffi in /usr/local/lib/python3.10/d
Requirement already satisfied: nest-asyncio>=1.5 in /usr/local/lib/python
Requirement already satisfied: Send2Trash>=1.8.0 in /usr/local/lib/python
Requirement already satisfied: terminado>=0.8.3 in /usr/local/lib/python3
Requirement already satisfied: prometheus-client in /usr/local/lib/python
Requirement already satisfied: nbclassic>=0.4.7 in /usr/local/lib/python3
Requirement already satisfied: qtpy>=2.4.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3
Requirement already satisfied: jedi>=0.16 in /usr/local/lib/python3.10/di
Requirement already satisfied: decorator in /usr/local/lib/python3.10/dis
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/d
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/d
```

```
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python
Requirement already satisfied: notebook-shim>=0.2.3 in /usr/local/lib/pyt
Requirement already satisfied: fastjsonschema>=2.15 in /usr/local/lib/pyt
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.
Requirement already satisfied: wcwidth in /usr/local/lib/python3.10/dist-
Requirement already satisfied: ptyprocess in /usr/local/lib/python3.10/di
Requirement already satisfied: argon2-cffi-bindings in /usr/local/lib/pyt
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/
Requirement already satisfied: parso<0.9.0,>=0.8.3 in /usr/local/lib/pyth
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /u
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/pyth
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.1
Requirement already satisfied: jupyter-server<3,>=1.8 in /usr/local/lib/p
Requirement already satisfied: cffi>=1.0.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dis
Requirement already satisfied: anyio<4,>=3.1.0 in /usr/local/lib/python3.
Requirement already satisfied: websocket-client in /usr/local/lib/python3
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.10/
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.1
```

2.1.Data Manipulation

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

```
tensor([[ 0.,  1.,  2.,  3.],
        [ 4.,  5.,  6.,  7.],
        [ 8.,  9., 10., 11.]])
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([[-2.2815, -0.0781, -0.6076, -1.5795],
        [ 2.4762, -0.2083,  1.1259,  0.1583],
        [ 1.9715,  0.0754, -1.7220,  0.0183]])
```

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

```
tensor(1)
```

- torch를 사용하여 python에서 deeplearning 관련 프로젝트를 진행할 때 matrix를 관리할 수 있음.

```
print(X)
print(X[-1])
print(X[1:3])
```

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

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

```
tensor([1.0000e+00, 2.7183e+00, 7.3891e+00, 2.0086e+01, 5.4598e+01,
1.4841e+02,
        2.4155e+07, 1.0966e+03, 2.9810e+03, 8.1031e+03, 2.2026e+04,
5.9874e+04])
```

```
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]])

print(X)
print(Y)
print(torch.cat((X, Y), dim=0))
print(torch.cat((X, Y), dim=1))
```

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

```
print(X == Y)
print(X.sum())
```

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

브로드캐스팅이 일어날 수 있는 조건은 다음과 같다.차원의 크기가 1일때 가능하다 두 배열 간의 연산에서 최소한 하나의 배열의 차원이 1이라면(0번 축이든 1번 축이든; 1행이든 1열이든) 가능하다.차원의 짝이 맞을 때 가능하다 차원에 대해 축의 길이가 동일하면 브로드캐스팅이 가능하다. 출처: https://sacko.tistory.com/16 [데이터 분석하는 문과생, 싸코:티스토리]

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

```
False
```

```
Z = 1 //torch.zeros_like(Y)
print(id(Y))
print('id(Z):', id(Z))
Z[:] = X + Y
print('id(Z):', id(Z))
Z += Y
print(id(Z))
```

```
139141620424096
id(Z): 139141620319152
id(Z): 139141620319152
139141620319152
```

```
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)
```

- python에서 딥러닝 학습을 위해 tensor를 사용한다.
- tensor를 통해 matrix를 연산을 수행, 관리할 수 있다.
-

2.2.Data Preprocessing using Panda

```
# 2.2 Data Preprocessing using panda

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
```

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

```
   NumRooms  RoofType_Slate  RoofType_nan
0       NaN           False          True
1       2.0           False          True
2       4.0            True         False
3       NaN           False          True
0    127500
1    106000
2    178100
3    140000
Name: Price, dtype: int64
```

```
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
```

```
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

```
# 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, x[2], len(x), x.shape
```

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

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

```
(tensor([[0, 1],
         [2, 3],
         [4, 5]]),
 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()  # Assign a copy of A to B by allocating new memory
A, A + B, A * B
```

```
(tensor([[0., 1., 2.],
         [3., 4., 5.]]),
 tensor([[ 0.,  2.,  4.],
         [ 6.,  8., 10.]]),
 tensor([[ 0.,  1.,  4.],
         [ 9., 16., 25.]]))
```

```
a = 2
X = torch.arange(24).reshape(2, 3, 4)
a + X, 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]]]),
 tensor([[[ 0,  2,  4,  6],
          [ 8, 10, 12, 14],
          [16, 18, 20, 22]],

         [[24, 26, 28, 30],
          [32, 34, 36, 38],
          [40, 42, 44, 46]]]),
 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.))
```

```
print(A,
A.shape,
A.sum(axis=0),
A.sum(axis=0).shape,
A.sum(axis=1))
```

```
tensor([[0., 1., 2.],
        [3., 4., 5.]]) torch.Size([2, 3]) tensor([3., 5., 7.]) torch.Size([
```

```
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_B = A.sum(axis =1)
sum_A, sum_A.shape, sum_B, sum_B.shape
```

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

```
A / sum_A #keepdims를 했기 때문에 나누기 연산이 가능해짐.
```

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

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

```
(tensor([[0., 1., 2.],
         [3., 4., 5.]]),
 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, x, A.shape, x.shape, torch.mv(A, x), A@x
# torch.mv 는 matrix vector
```

```
(tensor([[0., 1., 2.],
         [3., 4., 5.]]),
 tensor([0., 1., 2.]),
 torch.Size([2, 3]),
 torch.Size([3]),
 tensor([ 5., 14.]),
 tensor([ 5., 14.]))
```

```
B = torch.ones(3, 4)
A, B, torch.mm(A, B), A@B
# torch.mm 은 matrix multiplication
```

```
(tensor([[0., 1., 2.],
         [3., 4., 5.]]),
 tensor([[1., 1., 1., 1.],
         [1., 1., 1., 1.],
         [1., 1., 1., 1.]]),
 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() # 절대값 하고 다 더함
```

```
tensor(7.)
```

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

```
tensor(6.)
```

2.5.Automatic Differentiation

```
# 2.5 Automatic Differentation

import torch

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

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

```
# Can also create x = torch.arange(4.0, requires_grad=True)
x.requires_grad_(True)
x.grad  # The gradient is None by default
```

```
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_()  # Reset the gradient
y = x.sum()
print(y)
y.backward()
x.grad
```

```
tensor(6., grad_fn=<SumBackward0>)
tensor([1., 1., 1., 1.])
```

```
x.grad.zero_()
y = x*x
y.backward(gradient = torch.ones(len(y))) # faster : y.sum().backward()
x.grad
```

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

이 경우에는 y가 vector값이 되어 수행됨.

```
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)
print(a)
d = f(a)
d.backward()
```

```
tensor(-1.5992, requires_grad=True)
```

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

```
tensor(False)
```

3.1.Linear Regression

```
# 3.1 Linear Regression
```

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

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

```
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'
```

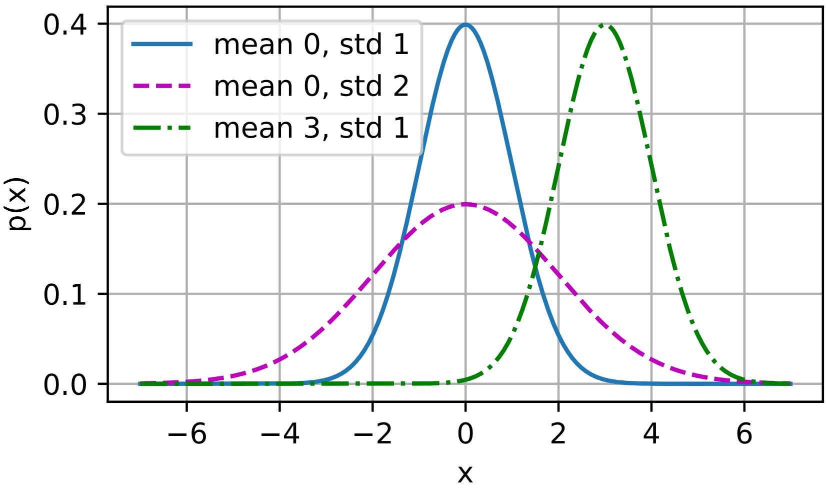
```
'0.26414 sec'
```

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

'0.00028 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)
```

```
# Use NumPy again for visualization
x = np.arange(-7, 7, 0.01)

# Mean and standard deviation pairs
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])
```



3.2.Object-Oriented Design for Implementation

```
# 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
```

```
def add_to_class(Class):  #@save
    """Register functions as methods ir
    def wrapper(obj):
        setattr(Class, obj.__name__, ot
    return wrapper
```

"@save" 주석은 허용되지 않습니다. 허용되는 값은 다음과 같습니다. [@param, @title, @markdown]

```
class HyperParameters:  #@save
    """The base class of hyperparameter
    def save_hyperparameters(self, ignc
        raise NotImplemented
```

"@save" 주석은 허용되지 않습니다. 허용되는 값은 다음과 같습니다. [@param, @title, @markdown]

```
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()
```

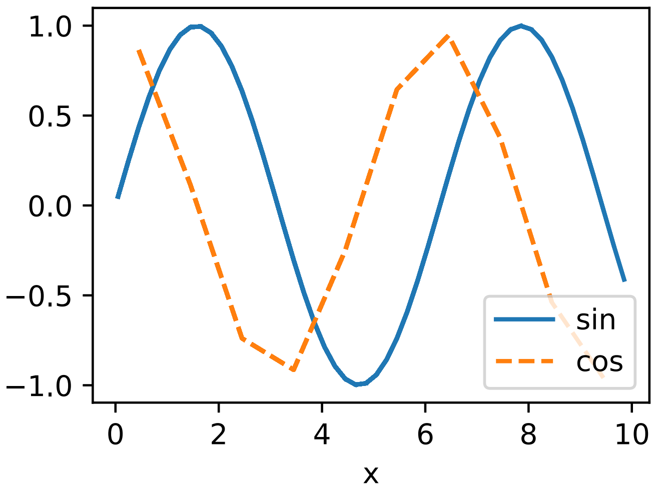
```
Class attribute "b" is 1
```

```
# Call the fully implemented HyperParameters class saved in d2l
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
    def __init__(self, xlabel=None, yla
                 ylim=None, xscale='lin
                 ls=['-', '--', '-.', '
                 fig=None, axes=None, f
        self.save_hyperparameters()

    def draw(self, x, y, label, every_n
        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)
```

"@save" 주석은 허용되지 않습니다. 허용되는 값은 다음과 같습니다. [@param, @title, @markdown]



```
class Module(nn.Module, d2l.HyperParame
    """The base class of models."""
    def __init__(self, plot_train_per_e
        super().__init__()
        self.save_hyperparameters()
        self.board = ProgressBoard()

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

    def forward(self, X):
        assert hasattr(self, 'net'), 'N
        return self.net(X)

    def plot(self, key, value, train):
        """Plot a point in animation."""
        assert hasattr(self, 'trainer')
        self.board.xlabel = 'epoch'
        if train:
            x = self.trainer.train_batc
                self.trainer.num_train_
            n = self.trainer.num_train_
                self.plot_train_per_epo
        else:
            x = self.trainer.epoch + 1
            n = self.trainer.num_val_ba
                self.plot_valid_per_epo
        self.board.draw(x, value.to(d2l
                        ('train_' if tr
                        every_n=int(n))

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

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

    def configure_optimizers(self):
        raise NotImplementedError
```

"@save" 주석은 허용되지 않습니다. 허용되는 값은 다음과 같습니다. [@param, @title, @markdown]

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

    def get_dataloader(self, train):
        raise NotImplementedError

    def train_dataloader(self):
        return self.get_dataloader(tra

    def val_dataloader(self):
        return self.get_dataloader(tra
```

"@save" 주석은 허용되지 않습니다. 허용되는 값은 다음과 같습니다. [@param, @title, @markdown]

```
class Trainer(d2l.HyperParameters):  #@
    """The base class for training mode
    def __init__(self, max_epochs, num_
        self.save_hyperparameters()
        assert num_gpus == 0, 'No GPU s

    def prepare_data(self, data):
        self.train_dataloader = data.tr
        self.val_dataloader = data.val_
        self.num_train_batches = len(se
        self.num_val_batches = (len(sel
                                if self

    def prepare_model(self, model):
        model.trainer = self
        model.board.xlim = [0, self.max
        self.model = model

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

    def fit_epoch(self):
        raise NotImplementedError
```

"@save" 주석은 허용되지 않습니다. 허용되는 값은 다음과 같습니다. [@param, @title, @markdown]

3.4.Linear Regression Implementation from Scratch

```
%matplotlib inline
import torch
from d2l import torch as d2l
```

```
class LinearRegressionScratch(d2l.Modul
    def __init__(self, num_inputs, lr,
        super().__init__()
        self.save_hyperparameters()
        self.w = torch.normal(0, sigma,
        self.b = torch.zeros(1, require
```

"@save" 주석은 허용되지 않습니다. 허용되는 값은 다음과 같습니다. [@param, @title, @markdown]

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

"@save" 주석은 허용되지 않습니다. 허용되는 값은 다음과 같습니다. [@param, @title, @markdown]

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

"@save" 주석은 허용되지 않습니다. 허용되는 값은 다음과 같습니다. [@param, @title, @markdown]

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

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

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

"@save" 주석은 허용되지 않습니다. 허용되는 값은 다음과 같습니다. [@param, @title, @markdown]

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

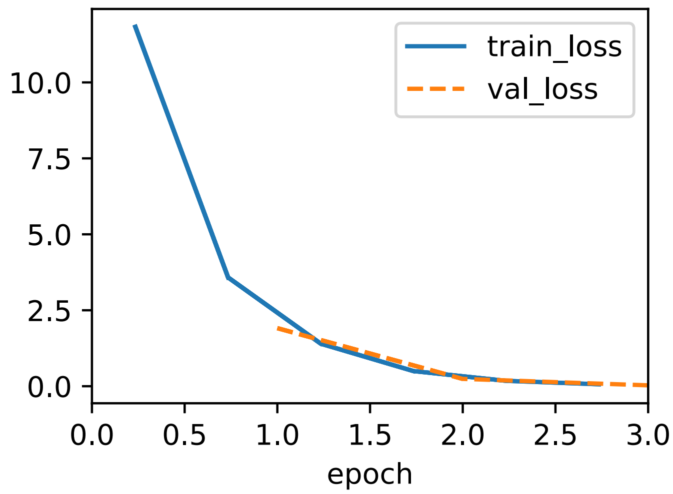
"@save" 주석은 허용되지 않습니다. 허용되는 값은 다음과 같습니다. [@param, @title, @markdown]

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

@d2l.add_to_class(d2l.Trainer)  #@save
def fit_epoch(self):
    self.model.train()
    for batch in self.train_dataloader:
        loss = self.model.training_step
        self.optim.zero_grad()
        with torch.no_grad():
            loss.backward()
            if self.gradient_clip_val >
                self.clip_gradients(sel
            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.val_batch_idx += 1
```

"@save" 주석은 허용되지 않습니다. 허용되는 값은 다음과 같습니다. [@param, @title, @markdown]

"@save" 주석은 허용되지 않습니다. 허용되는 값은 다음과 같습니다. [@param, @title, @markdown]

```
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)
```



4.1.Softmax Regression

코딩을 시작하거나 AI로 코드를 생성하세요.

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()
```

```
class FashionMNIST(d2l.DataModule):  #@
    """The Fashion-MNIST dataset."""
    def __init__(self, batch_size=64, r
        super().__init__()
        self.save_hyperparameters()
        trans = transforms.Compose([tra
                                    tra
        self.train = torchvision.datase
            root=self.root, train=True,
        self.val = torchvision.datasets
            root=self.root, train=False
```

"@save" 주석은 허용되지 않습니다. 허용되는 값은 다음과 같습니다. [@param, @title, @markdown]

```
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)  #@save
def text_labels(self, indices):
    """Return text labels."""
    labels = ['t-shirt', 'trouser', 'pu
              'sandal', 'shirt', 'sneak
    return [labels[int(i)] for i in ind
```

"@save" 주석은 허용되지 않습니다. 허용되는 값은 다음과 같습니다. [@param, @title, @markdown]

```
@d2l.add_to_class(FashionMNIST)  #@save
def get_dataloader(self, train):
    data = self.train if train else self.va
    return torch.utils.data.DataLoader(data
                                       num_
```

"@save" 주석은 허용되지 않습니다. 허용되는 값은 다음과 같습니다. [@param, @title, @markdown]

```
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:
  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'
```

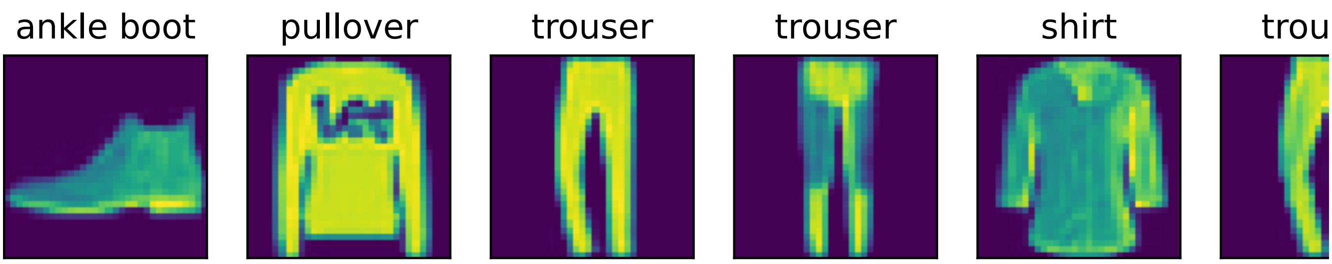
```
'11.83 sec'
```

```
def show_images(imgs, num_rows, num_cols, t
    """Plot a list of images."""
    raise NotImplementedError
```

"@save" 주석은 허용되지 않습니다. 허용되는 값은 다음과 같습니다. [@param, @title, @markdown]

```
@d2l.add_to_class(FashionMNIST)  #@save
def visualize(self, batch, nrows=1, ncols=8
    X, y = batch
    if not labels:
        labels = self.text_labels(y)
    d2l.show_images(X.squeeze(1), nrows, nc
batch = next(iter(data.val_dataloader()))
data.visualize(batch)
```

"@save" 주석은 허용되지 않습니다. 허용되는 값은 다음과 같습니다. [@param, @title, @markdown]



4.3.The Base Classification Model

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

```
class Classifier(d2l.Module):  #@save
    """The base class of classificatio
    def validation_step(self, batch):
        Y_hat = self(*batch[:-1])
        self.plot('loss', self.loss(Y_h
        self.plot('acc', self.accuracy(
```

"@save" 주석은 허용되지 않습니다. 허용되는 값은 다음과 같습니다. [@param, @title, @markdown]

```
@d2l.add_to_class(d2l.Module)  #@save
def configure_optimizers(self):
    return torch.optim.SGD(self.paramet
```

"@save" 주석은 허용되지 않습니다. 허용되는 값은 다음과 같습니다. [@param, @title, @markdown]

```
@d2l.add_to_class(Classifier)  #@save
def accuracy(self, Y_hat, Y, averaged=T
    """Compute the number of correct pr
    Y_hat = Y_hat.reshape((-1, Y_hat.sh
    preds = Y_hat.argmax(axis=1).type(Y
    compare = (preds == Y.reshape(-1)).
    return compare.mean() if averaged e
```

"@save" 주석은 허용되지 않습니다. 허용되는 값은 다음과 같습니다. [@param, @title, @markdown]

4.4.Softmax Regression Implementation from Scratch

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

```
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  # The broadcasting mechanism is applied here
```

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

```
(tensor([[0.2307, 0.2722, 0.2010, 0.1581, 0.1379],
         [0.1445, 0.2203, 0.1927, 0.2963, 0.1462]]),
 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]
```

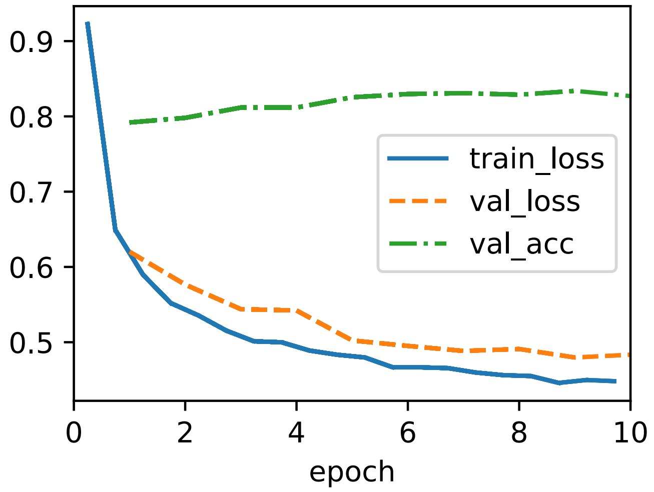
```
tensor([0.1000, 0.5000])
```

```
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)
```

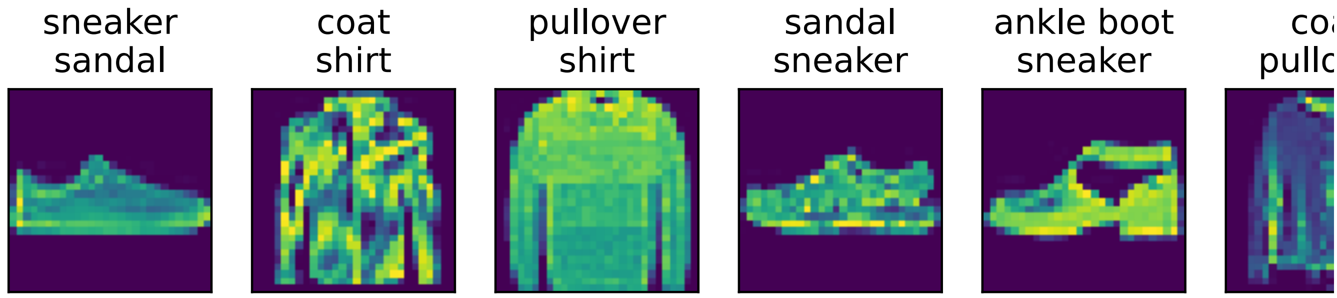
```
data = d2l.FashionMNIST(batch_size=256)
model = SoftmaxRegressionScratch(num_inputs=784, num_outputs=10, lr=0.1)
trainer = d2l.Trainer(max_epochs=10)
trainer.fit(model, data)
```




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

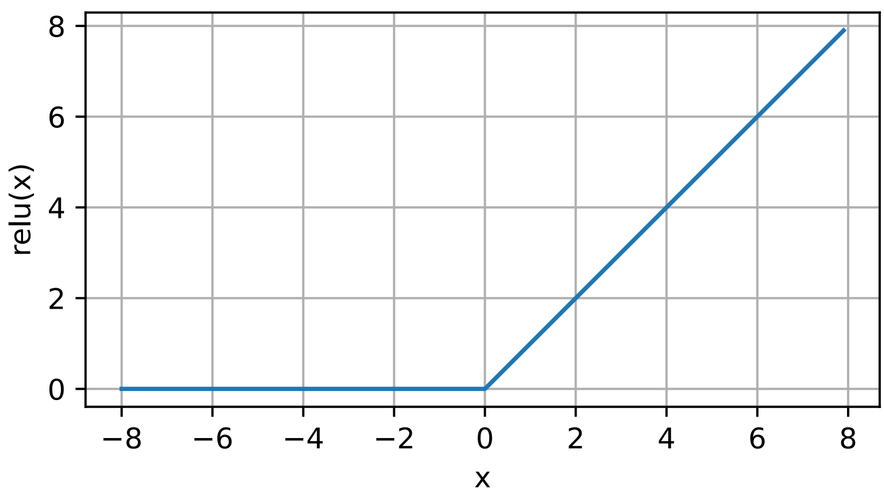


5.1.Multilayer Perceptrons

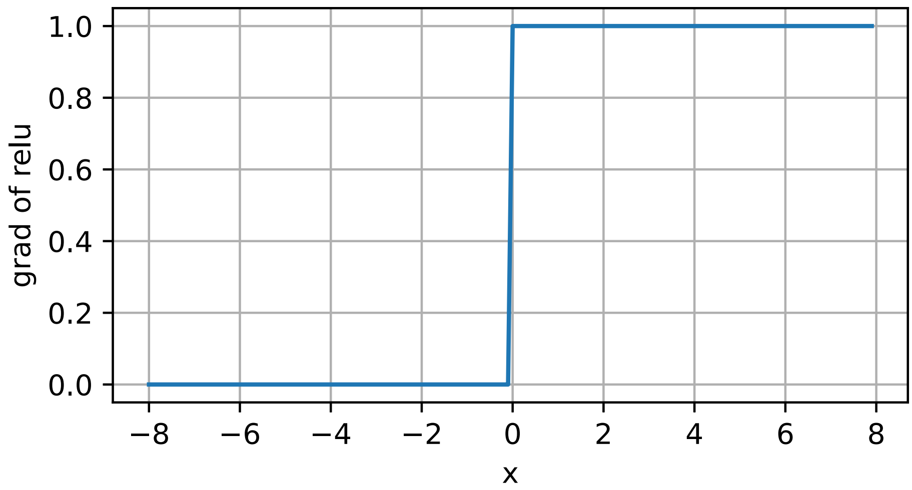
```
%matplotlib inline
import torch
from d2l import torch as d2l
```

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

```
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))
```

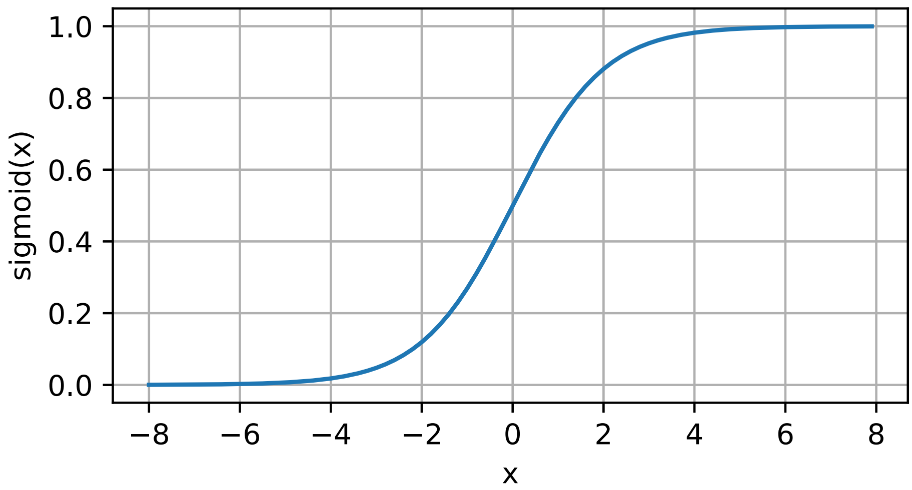


```
y.backward(torch.ones_like(x), retain_graph=True)
d2l.plot(x.detach(), x.grad, 'x', 'grad of relu', figsize=(5, 2.5))
```



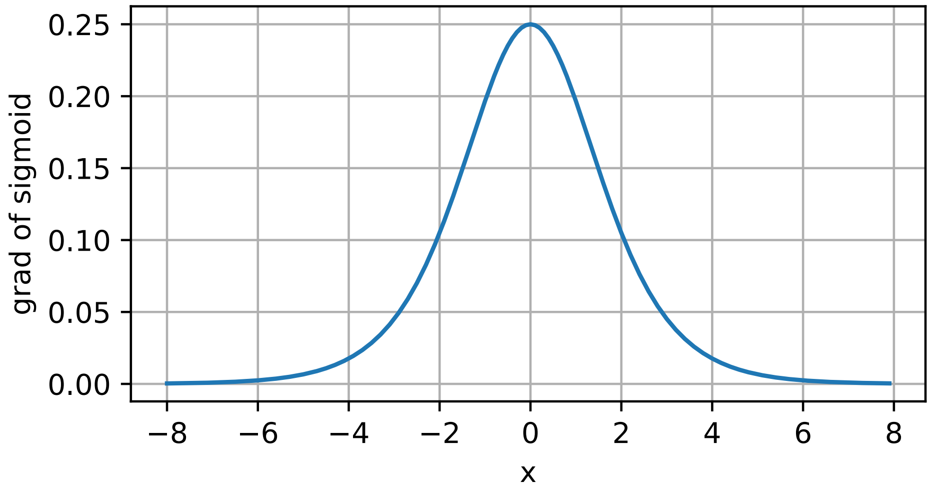
$$sigmoid(x) = \frac{1}{1 + exp(-x)}$$

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



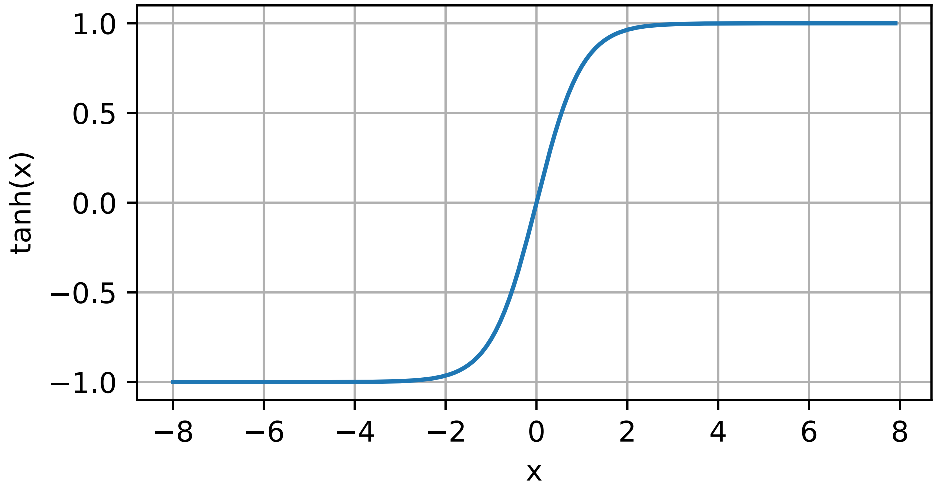
$$\frac{d}{dx}\text{sigmoid}(x) = \frac{\exp(-x)}{(1 + \exp(-x))^2} = \text{sigmoid}(x)(1 - \text{sigmoid}(x))$$

```
# Clear out previous gradients
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))
```



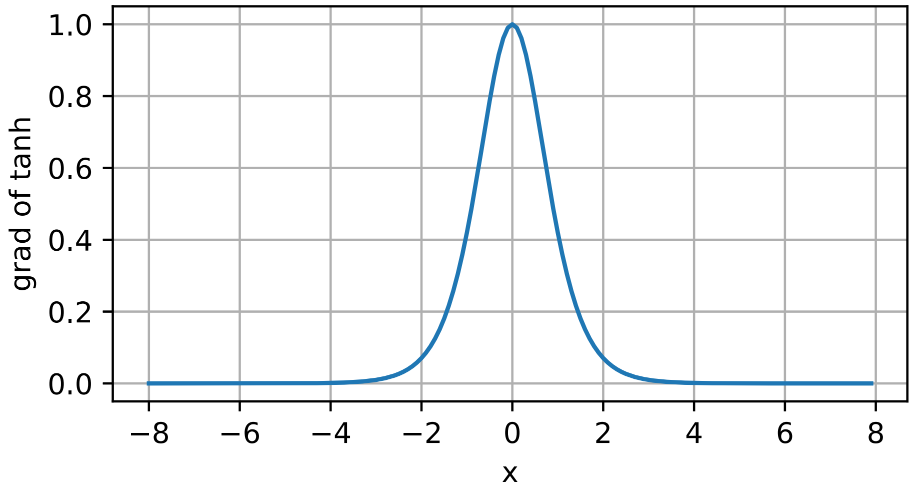
$$tanh(x) = \frac{1 - exp(-2x)}{1 + exp(-2x)}$$

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



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

```
# Clear out previous gradients
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

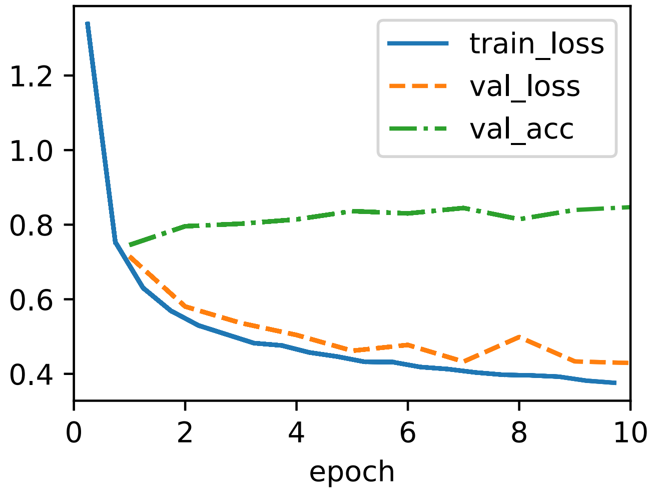
```
import torch
from torch import nn
from d2l import torch as d2l
```

```
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)
```

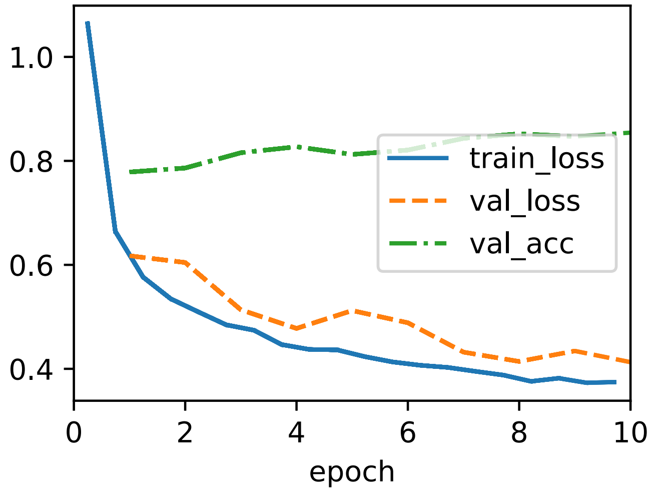
```
@d2l.add_to_class(MLPScratch)
def forward(self, X):
    X = X.reshape((-1, self.num_inputs))
    H = relu(torch.matmul(X, self.W1) + self.b1)
    return torch.matmul(H, self.W2) + self.b2
```

```
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

Discussions & Exercises

2.1.3.Discussions

tensor를 concat 할 수 있는데, dim에 따라서 행으로 옆에 붙일지 열로 밑으로 붙일지를 정할 수 있다. dim = 0 을 하면, 열의 개수가 유지되며 밑으로 붙이고 dim = 1 을 하면, 행의 개수가 유지되어 옆으로 붙인다.

2.2.2.Discussions

iloc[a:b, c:d] <- a<=x<b index에 대하여 c<=y<=d label 출력

2.3.6.Discussions

sum을 수행할 때, axis = 0 or 1 에 따라서 각각 행, 열에 대한 합을 도출한다. axis = 0,1로 하면 행, 열에 대한 합을 모두 수행하기 때문에 전체 합이 도출된다.

2.3.7.Discussions

keepdims가 의미하는 것은? -> 브로드캐스팅을 위해 차원을 유지하는 것 질문? 왜 keepdims를 제거했을 때 shape에서 torch.Size([1,2])가 되지 않을까?

2.5.1.1.Discussions

requires_grad = True 를 통해서 x의 gradient를 구할 준비를 하고, y를 backward()한 뒤, x.grad를 하면 gradient 값을 구할 수 있다.

- $2\mathcal{X}$^2를 미분하면 $4\mathcal{X}$

2.5.1.2.Discussions

x0, x1, x2, x3에 대해 미분한 값임.

- y = x0 + x1 + x2 + x3 인데 각각 xn에 대하여 미분하면 y=1이므로 [1,1,1,1]이 들어가게 됨.

2.5.Discussions

딥러닝에서는 학습에 있어서 파라미터들의 전파 역전파에 대한 정보를 얻기 위해 gradient값을 아는 것이 중요한데, 이를 grad를 통해서 쉽게 관리 하고 구할 수 있다. 이때, y가 갖는 값이 scalar 인지 vector인지 구분하는 것 또한 중요함.

3.1.Discussions

normal distribution을 통해서 linear regression을 연관지을 수 있다. (x - sigma)^2에 (y-wtx -b)^2을 넣어 loss에 대한 normal distribution을 만들고 이를 통해 maximum likelihood를 구할 수 있다.

3.2.Discussions

Model Definition (Module) Moudle 클래스의 서브클래스에서 모델 구조를 정의하고, forward propagation 및 최적화 설정 과 같은 메서드를 사용. 이를 통해 모델 구성 요소의 정의와 재사용이 간편해 짐.

Data Handling (DataModule) DataModule 클래스는 데이터 다운로드, 전처리 및 학습과 검증을 위이한 데이터 로더를 제공하는 기본 클래스. 이를 통해 일관되고 재사용 가능한 데이터 처리가 가능.

Training(Trainier) Trainer 클래스는 데이터 배치 반복, gradient 계산, 모델 파라미터 업데이트 등의 전체 훈련 루프를 담당. 이를 통해 epoch같은 훈련 세부 사항을 관리할 수 있음.

3.4.2.Discussions

Loss Function

$$L = \frac{(\hat{y} - y)^2}{2}$$

3.4.4.Discussions

Training

1. **Initialize parameters**

$$(\mathbf{w}, b)$$

2. **Repeat until done**
    - Compute gradient

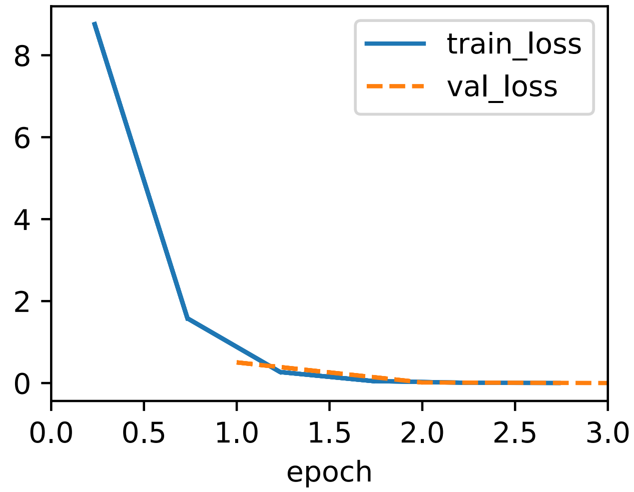
$$\mathbf{g} \leftarrow \partial_{(\mathbf{w},b)} \frac{1}{|B|} \sum_{i \in B} l(\mathbf{x}^{(i)}, y^{(i)}, \mathbf{w}, b)$$

    - Update parameters

$$(\mathbf{w}, b) \leftarrow (\mathbf{w}, b) - \eta \mathbf{g}$$

3.4.4.Exercises_Changing LearningRate

```
model = LinearRegressionScratch(2, lr=0.05)
data = d2l.SyntheticRegressionData(w=torch.tensor([2, -3.4]), b=4.2)
trainer = d2l.Trainer(max_epochs=3)
trainer.fit(model, data)
```



4.1.Discussions

Softmax?

classfication과 같은 것에서 각 x값들의 합을 1로 하고 범위가 0~1이 되도록 수치를 조종

$$O = XW + b$$

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

큰 값이 큰 값으로 유지됨.

Log-Likelihood?

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

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

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

흔히 cross-entropy loss라고 불림.

Entropy?

$$H[P] = \sum_j -P(j) \log P(j).$$

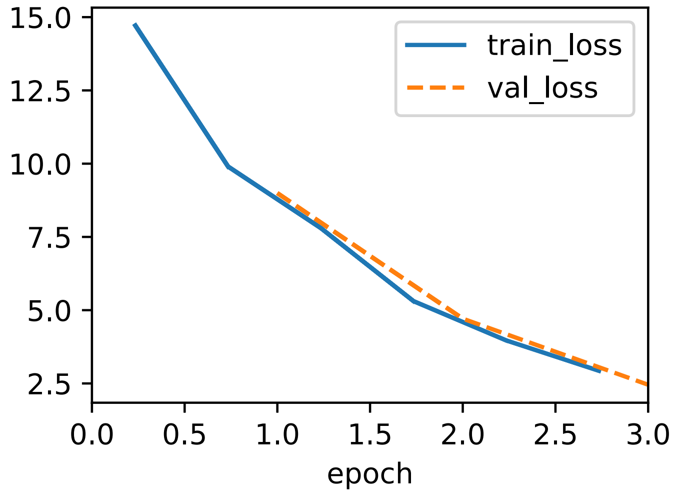
4.2.Discussions

FashoinMNIST 이미지 데이터셋을 통해 분류하는 classfier를 만들어 보았고 몇개의 예시가 분류되는 것을 보았음.

4.2.1. Memo

Fashion-MNIST consists of images from 10 categories, each represented by 6000 images in the training and by 1000 in the test dataeset. -> 60000, 10000.

```
model = LinearRegressionScratch(2, lr=0.01)
data = d2l.SyntheticRegressionData(w=torch.tensor([2, -3.4]), b=4.2)
trainer = d2l.Trainer(max_epochs=3)
trainer.fit(model, data)
```



4.3.Discussions

정확도는 예측된 값인 $\hat(y)$ 과 label 값인 y를 비교하여 계산된다. 이때, y의 '==' 비교는 data type에 민감하므로 data type을 맞추어준 뒤 0, 1로 비교하여 나타낸다.

4.4.MEMO

- 28 x 28 pixel image를 flatten 하여 784길이의 벡터값으로 변환한다.
- 결과가 class의 개수인 10개로 나와야 하기 때문에 weight vector가 784 x 10 에 bias 1 x 10 으로 하였음.
- W 를 Gaussian noise로 정함.
- bias는 0으로 초기화.

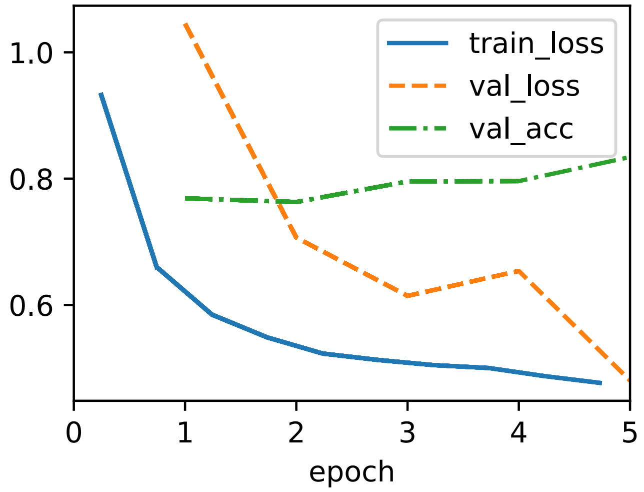
4.4.2.Discussions

reshape의 첫번째 인자 -1은 나머지를 통해 적절한 차원의 크기로 자동으로 저장해 주는 기능임.
self.W.shape[0]인 W의 첫번째 차원의 크기가 784 이므로 28 x 28 이 1 x 784가 됨.

4.4.5.Exercise

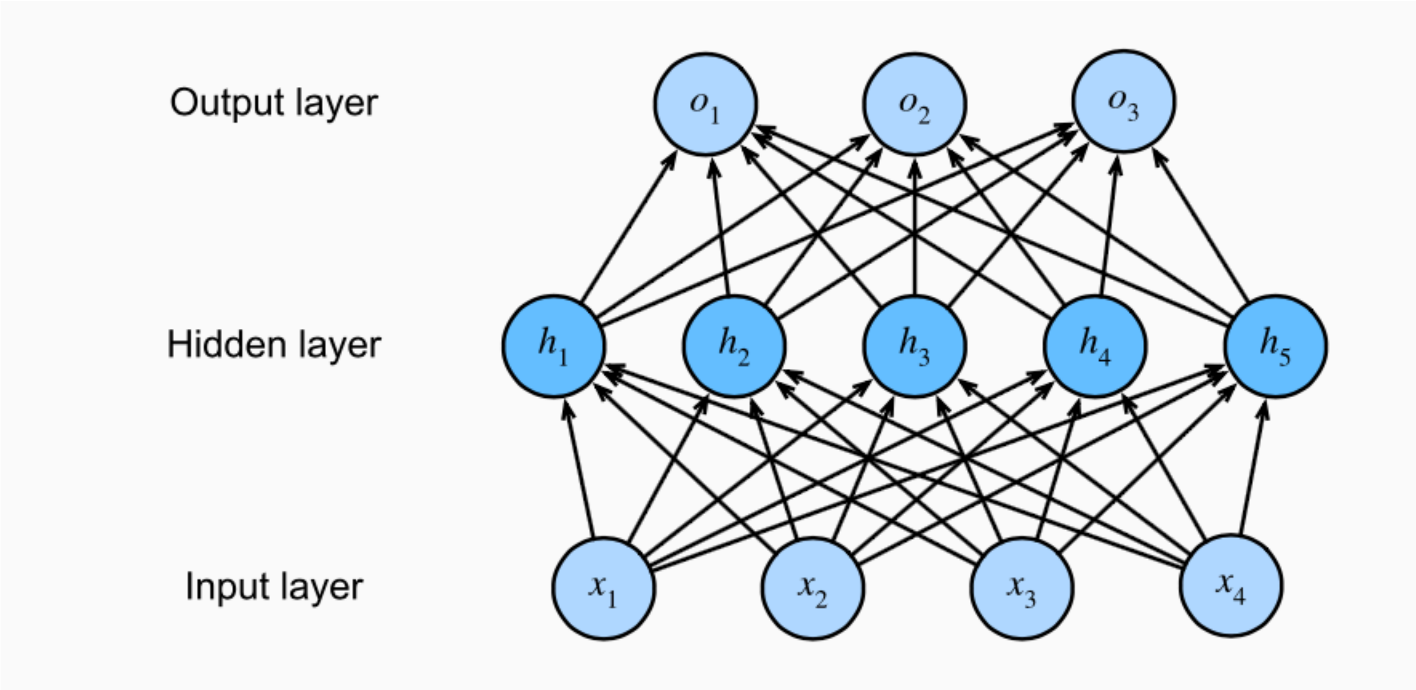
기존 에제에서 10 epoch 가량에 train_loss가 줄어들지만 val_loss가 늘어나는 overfitting이 생겼음. 또한 val_lossdml 기울기가 매우 완만하여 epoch를 5로 줄이고 lr을 0.2f로 늘려서 진행하여봄.

```
data = d2l.FashionMNIST(batch_size=256)
model = SoftmaxRegressionScratch(num_inputs=784, num_outputs=10, lr=0.2)
trainer = d2l.Trainer(max_epochs=5)
trainer.fit(model, data)
```



5.1.1.Discussions

선형성으로 모든 문제를 해결할 수 없음. 예를 들어 고양이와 개의 이미지를 분류할 때, 단순 한 픽셀의 밝기에 대하여 선형 모델을 사용하는 것은 실패할 가능성이 크다. 이를 신경망을 사용하여 hidden layer을 통해 학습할 수 있다.

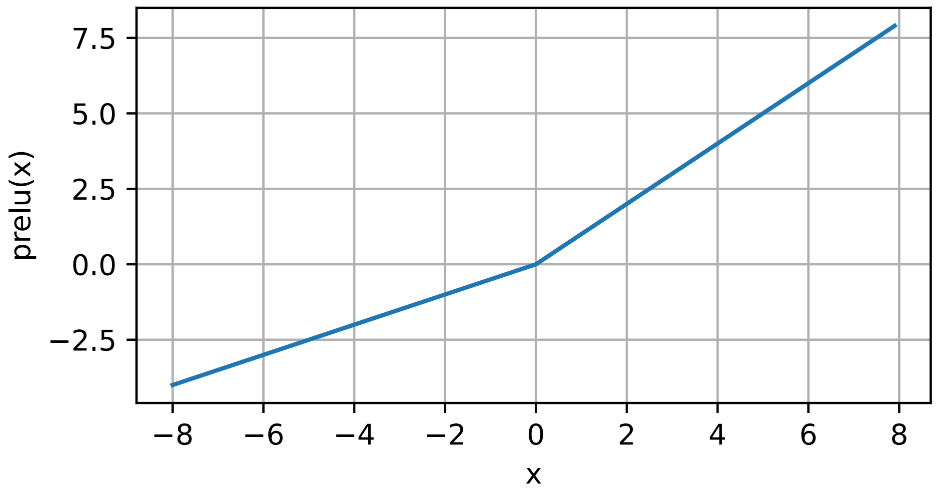


$$H = XW + b$$
$$O = HW + b$$

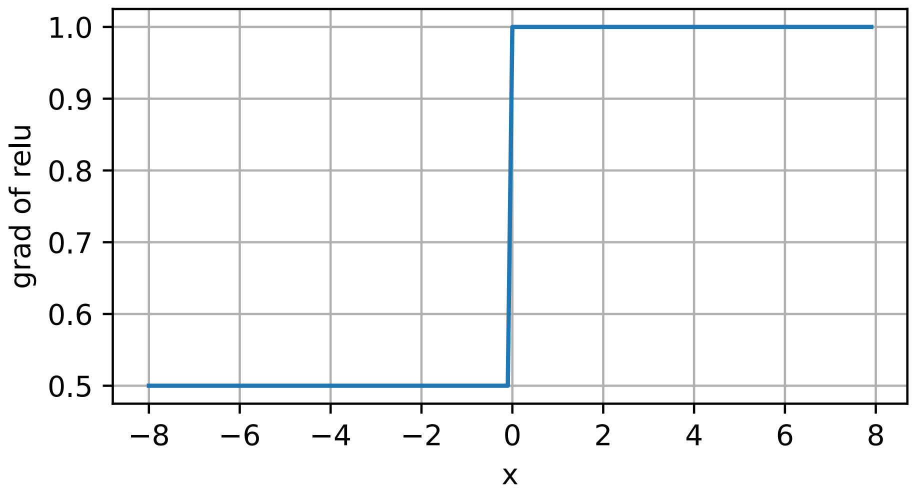
5.1.2.1. Exercises

$$pReLU(x) = max(0, x) + \alpha min(0, x)$$

```
alpha = 0.5
x = torch.arange(-8.0, 8.0, 0.1, requires_grad=True)
y = torch.where(x > 0, x, alpha * x)  # torch.where을 사용해 요소별로 조건 적용

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 relu', figsize=(5, 2.5))
```

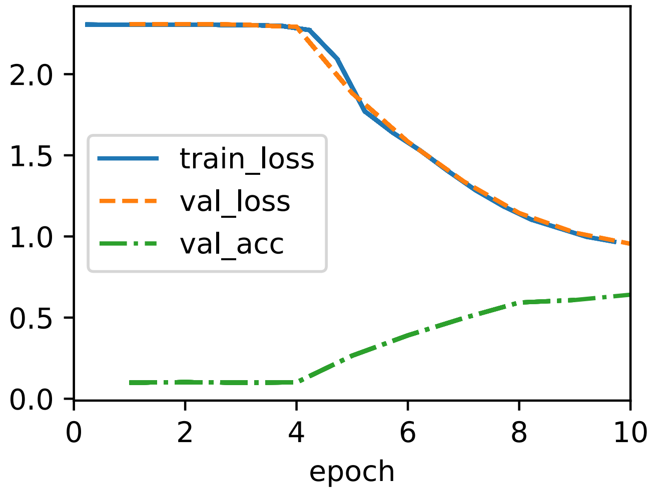


5.2.Exercises

Activate Function, LearningRate 혹은 hiddenlayer의 개수를 바꾸어서 실행해보았음.

```
class MLPScratchEx(d2l.Classifier):
    def __init__(self, num_inputs, num_outputs, num_hiddens1, num_hiddens2, lr,
        super().__init__()
        self.save_hyperparameters()
        self.W1 = nn.Parameter(torch.randn(num_inputs, num_hiddens1) * sigma)
        self.b1 = nn.Parameter(torch.zeros(num_hiddens1))
        self.W2 = nn.Parameter(torch.randn(num_hiddens1, num_hiddens2) * sigma)
        self.b2 = nn.Parameter(torch.zeros(num_hiddens2))
        self.W3 = nn.Parameter(torch.randn(num_hiddens2, num_outputs) * sigma)
        self.b3 = nn.Parameter(torch.zeros(num_outputs))


def relu(X):
    a = torch.zeros_like(X)
    return torch.max(X, a)
def sigmoid(X):
    return 1 / (1 + torch.exp(-X))



@d2l.add_to_class(MLPScratchEx)
def forward(self, X):
    X = X.reshape((-1, self.num_inputs))

    # 첫 번째 hidden layer: W1, b1
    H1 = sigmoid(torch.matmul(X, self.W1) + self.b1)

    # 두 번째 hidden layer 추가: W2, b2
    H2 = sigmoid(torch.matmul(H1, self.W2) + self.b2)

    # 출력 layer: W3, b3 (기존 W2, b2에서 W3, b3로 변경)
    return torch.matmul(H2, self.W3) + self.b3
```

```
model = MLPScratchEx(num_inputs=784, num_outputs=10, num_hiddens1=256, num_hidc
data = d2l.FashionMNIST(batch_size=256)
trainer = d2l.Trainer(max_epochs=10)
trainer.fit(model, data)
```



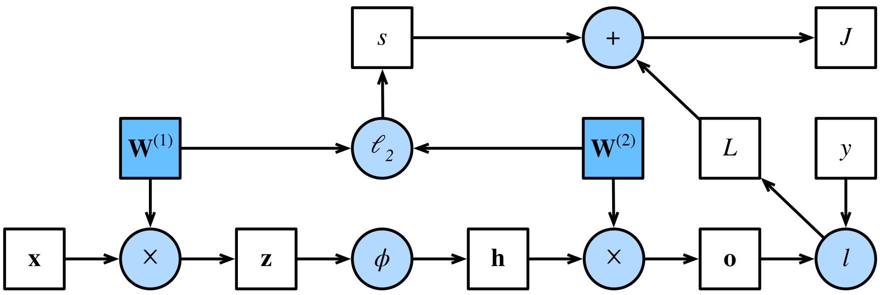
5.3.1.Discussions

Forward propagation

$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x}$$
$$\mathbf{h} = \phi(\mathbf{z})$$
$$\mathbf{o} = \mathbf{W}^{(2)}\mathbf{h}$$
$$L = l(\mathbf{o}, \mathbf{y})$$
$$s = \frac{\lambda}{2}\left(\|\mathbf{W}^{(1)}\|_F^2 + \|\mathbf{W}^{(2)}\|_F^2\right)$$
$$J = L + s$$

input 데이터에 대하여 가중치를 곱한 뒤 activation 함수 넣음. 그 결과에 대한 가중치를 또 곱하여 output 만들어냄. output과 실제 label간의 Loss를 구한다. reqularization을 통해 구한 값과 Loss값을 구하여 모델의 regularized loss를 구한다.

Computational Graph of Forward Propagation



5.3.2.Discussions

Back propagation

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

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

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

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

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

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

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

코딩을 시작하거나 AI로 코드를 생성하세요.