2.1

```
import torch
```

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

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

```
x.numel()
```

```
12
```

```
x.shape
```

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

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

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

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

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

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

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

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

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

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

```
tensor([[ 0.5988, -0.4802, -0.3269,  0.2743],
        [-0.3527, -0.2378,  0.1255,  0.0285],
        [ 0.9228,  0.9514, -0.1098,  1.2014]])
```

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

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

```
X2 = torch.arange(16).reshape(4, 4)
X2
```

```
tensor([[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11],
        [12, 13, 14, 15]])
```

```
X2[1:3, 1:3] = 0
X2
```

```
tensor([[ 0,  1,  2,  3],
        [ 4,  0,  0,  7],
```

```
             [ 8,  0,  0, 11],
             [12, 13, 14, 15]])
```

```
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)                #dim 이해
```

```
(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(3).reshape((1, 3))
a, b
```

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

```
a + b
```

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

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

```
False
```

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

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

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

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

```
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]])
X < Y, X > Y
```

```
(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]]))
```

```
a1 = torch.arange(6).reshape((3, 2, 1))
b1 = torch.arange(4).reshape((1, 2, 2))

a2 = torch.arange(12).reshape((2, 3, 2))
b2 = torch.arange(3).reshape((3, 1))

a1, b1, a2, b2
```

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

         [[2],
          [3]],

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

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

```
a1 + b1, a2 + b2
```

```
(tensor([[[0, 1],
          [3, 4]],

         [[2, 3],
          [5, 6]],

         [[4, 5],
          [7, 8]]]),
 tensor([[[ 0,  1],
          [ 3,  4],
          [ 6,  7]],

         [[ 6,  7],
          [ 9, 10],
          [12, 13]]]))
```

2.2

```
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,Tile,
4,Slate,178100
NA,NA,140000''')
```

```
import pandas as pd

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

```
   NumRooms RoofType     Price
0       NaN      NaN  127500.0
1       2.0     Tile       NaN
2       4.0    Slate  178100.0
3       NaN      NaN  140000.0
```

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

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

```
print(pd.get_dummies(data, dummy_na=True))
```

```
   NumRooms     Price  RoofType_Slate  RoofType_Tile  RoofType_nan
0       NaN  127500.0           False          False          True
1       2.0       NaN           False           True         False
2       4.0  178100.0            True          False         False
3       NaN  140000.0           False          False          True
```

function은 NumRooms과 Price가 아닌 RoofType만 Categorical input field 라고 판단했다. 구별의 기준이 뭘까? int와 float은 Categorical input field가 아니라고 판단하는가?

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

```
   NumRooms  RoofType_Slate  RoofType_Tile  RoofType_nan
0       3.0           False          False          True
1       2.0           False           True         False
2       4.0            True          False         False
3       3.0           False          False          True
0    127500.000000
1    148533.333333
2    178100.000000
3    140000.000000
Name: Price, dtype: float64
```

```
import torch

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

```
(tensor([[3., 0., 0., 1.],
         [2., 0., 1., 0.],
         [4., 1., 0., 0.],
         [3., 0., 0., 1.]], dtype=torch.float64),
 tensor([127500.0000, 148533.3333, 178100.0000, 140000.0000],
        dtype=torch.float64))
```

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

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

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

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

2.3

```
import torch

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]
```

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

```
x2 = torch.arange(24).reshape((4, 6))
len(x2), x2.shape
```

```
(4, torch.Size([4, 6]))
```

```
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.]]))
```

```
id(A) == id(B)
```

```
False
```

다른 위치에 저장됨

```
A * B     # A와 B는 같은 형태
```

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

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

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

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

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

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

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

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

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

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

```
tensor(True)
```

여러 axis 지정하기

```
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]))
```

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

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

기존 axis 1 자리를 크기 1개의 축으로 남겨둠 => broadcast 가능

```
A / sum_A
```

```
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.]]))
```

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

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

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

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

m*n Matrix와 n Vector의 Products: n Vector을 m Vector로 변환하는 행위.

m*n Matrix와 n*K Matrix의 Products: n Vector을 m Vector로 변환하는 행위.

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

```
tensor(7.)
```

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

```
tensor(6.)
```

2.5

```
import torch
```

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

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

```
x.requires_grad_(True)
#x = torch.arange(4.0, requires_grad=True)
x.grad #None
```

동일한 미분이 반복되므로 memory의 allocate이 반복됨을 피함

```
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_() #수동으로 reset해야함
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)))  #y.sum().backward()
x.grad
```

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

scalar로 합산하는 방법을 알려주기 위해, y길이와 같은 vector을 제공해야함

```
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
```

```
tensor(0.9940, requires_grad=True)
```

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

```
tensor(True)
```

즉, 미분할 변수에 미리 requires_grad를 달아두고, 미리 예측할 수 없는 function을 작동해도 backpropagation function을 통해 gradient를 알 수 있다.

3.1

```
pip install d2l
```

```
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter==1.0.0->d2l) (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) (7.34.0)
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter==1.0.0->d2l) (6.1.12)
Requirement already satisfied: tornado>=4.2 in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter==1.0.0->d2l) (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) (3.
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from ipywidgets->jupyter==1.0.0->d2l) (3.
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) (2.18.0)
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l) (4.9.4)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l) (4.12.3)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l) (6.1.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l) (0.7.1)
```

```
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) (0.5.1)
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->notebo
```

```
%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)
```

```
c = torch.zeros(n)
t = time.time()
for i in range(n):
  c[i] = a[i] + b[i]
time.time() - t
```

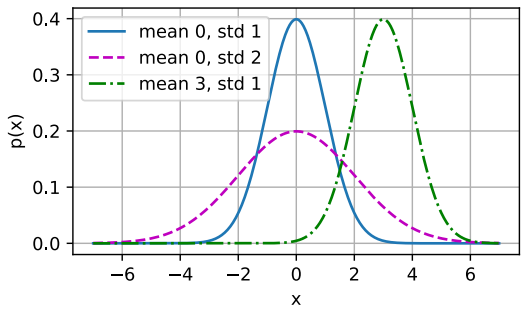
```
0.2072601318359375
```

```
t = time.time()
d = a + b
time.time() - t
```

```
0.00020694732666015625
```

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

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

params = [(0, 1), (0, 2), (3, 1)]
d2l.plot(x, [normal(x, mu, sigma) for mu, sigma in params], xlabel='x',
  ylabel='p(x)', figsize=(4.5, 2.5),
  legend=[f'mean {mu}, std {sigma}' for mu, sigma in params])
```



3.2

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

```
def add_to_class(Class):
  """
    Register functions as methods in created class.
```

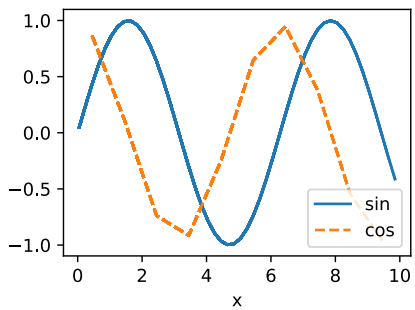
```
    """
    def wrapper(obj):
        setattr(Class, obj.__name__, obj)
    return wrapper


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

a = A()


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

a.do()
```

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

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


# 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 in animation.
    """
    def __init__(self, xlabel=None, ylabel=None, xlim=None,
                 ylim=None, xscale='linear', yscale='linear',
                 ls=['-', '--', '-.', ':'], colors=['C0', 'C1', 'C2', 'C3'],
                 fig=None, axes=None, figsize=(3.5, 2.5), display=True):
        self.save_hyperparameters()

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


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



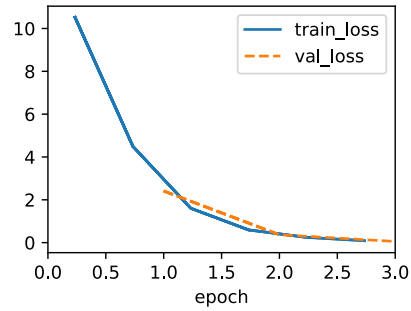

```
class Module(nn.Module, d2l.HyperParameters):
    """
    The base class of models.
    """
    def __init__(self, plot_train_per_epoch=2, plot_valid_per_epoch=1):
        super().__init__()
        self.save_hyperparameters()
        self.board = ProgressBoard()
```

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

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

    def plot(self, key, value, train):
        """Plot a point in animation."""
        assert hasattr(self, 'trainer'), 'Trainer is not inited'
        self.board.xlabel = 'epoch'
        if train:
            x = self.trainer.train_batch_idx / ₩
                self.trainer.num_train_batches
            n = self.trainer.num_train_batches / ₩
                self.plot_train_per_epoch
        else:
            x = self.trainer.epoch + 1
            n = self.trainer.num_val_batches / ₩
                self.plot_valid_per_epoch
        self.board.draw(x, value.to(d2l.cpu()).detach().numpy(),
                        ('train_' if train else 'val_') + key,
                        every_n=int(n))

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

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

    def configure_optimizers(self):
        raise NotImplementedError


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

    def get_dataloader(self, train):
        raise NotImplementedError

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

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


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

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

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

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

```
            self.fit_epoch()

    def fit_epoch(self):
        raise NotImplementedError
```

3.4

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


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


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


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


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

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

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


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


@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([ 0.1102, -0.2094])
error in estimating b: tensor([0.2361])
```

4.2

```
%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, 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)
```

```
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz to ../data/FashionMNIST/raw/train-images-idx3-u
100%|██████████| 26421880/26421880 [00:07<00:00, 3575454.23it/s]
Extracting ../data/FashionMNIST/raw/train-images-idx3-ubyte.gz to ../data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz to ../data/FashionMNIST/raw/train-labels-idx1-u
100%|██████████| 29515/29515 [00:00<00:00, 265929.89it/s]
Extracting ../data/FashionMNIST/raw/train-labels-idx1-ubyte.gz to ../data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz to ../data/FashionMNIST/raw/t10k-images-idx3-uby
100%|██████████| 4422102/4422102 [00:00<00:00, 5026009.89it/s]
Extracting ../data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz to ../data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz to ../data/FashionMNIST/raw/t10k-labels-idx1-uby
100%|██████████| 5148/5148 [00:00<00:00, 3356486.40it/s]
Extracting ../data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to ../data/FashionMNIST/raw

(60000, 10000)
```

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

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

```
@d2l.add_to_class(FashionMNIST)
def text_labels(self, indices):
    """Return text labels."""
    labels = ['t-shirt', 'trouser', 'pullover', 'dress', 'coat',
```

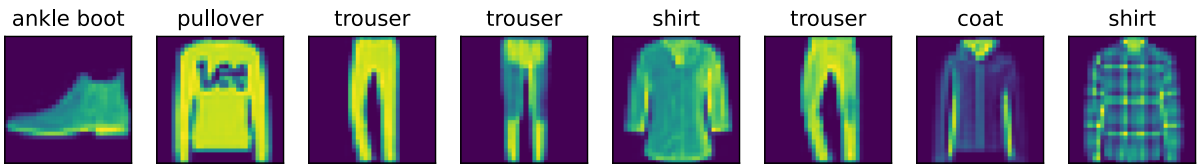
```
                'sandal', 'shirt', 'sneaker', 'bag', 'ankle boot']
    return [labels[int(i)] for i in indices]


@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'
```

```
'14.23 sec'
```

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


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



4.3

```
import torch
from d2l import torch as d2l


class Classifier(d2l.Module):
    """The base class of classification models."""
    def validation_step(self, batch):
        Y_hat = self(*batch[:-1])
        self.plot('loss', self.loss(Y_hat, batch[-1]), train=False)
        self.plot('acc', self.accuracy(Y_hat, batch[-1]), train=False)


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


@d2l.add_to_class(Classifier)
def accuracy(self, Y_hat, Y, averaged=True):
    """Compute the number of correct predictions."""
    Y_hat = Y_hat.reshape((-1, Y_hat.shape[-1]))
    preds = Y_hat.argmax(axis=1).type(Y.dtype)
    compare = (preds == Y.reshape(-1)).type(torch.float32)
    return compare.mean() if averaged else compare
```

4.4

```
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([[0.1539, 0.2338, 0.2184, 0.2076, 0.1862],
         [0.1335, 0.2618, 0.2843, 0.1353, 0.1851]]),
 tensor([1., 1.]))
```

```
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]
```

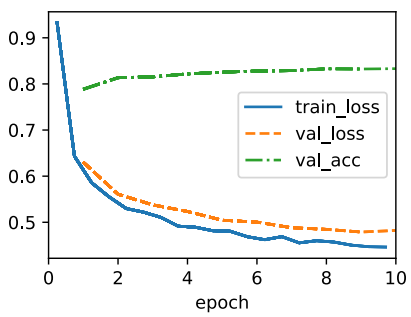
```
tensor([0.1000, 0.5000])
```

```
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)
```

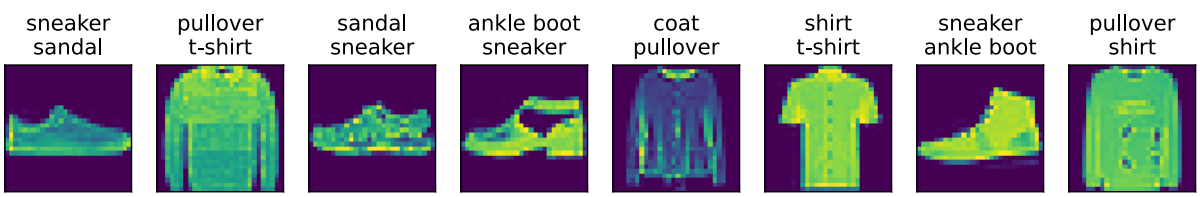
```
data = d2l.FashionMNIST(batch_size=256)
model = SoftmaxRegressionScratch(num_inputs=784, num_outputs=10, lr=0.1)
trainer = d2l.Trainer(max_epochs=10)
trainer.fit(model, data)
```




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

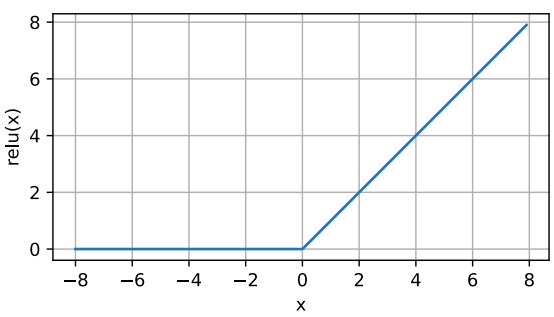
```
torch.Size([256])
```

```
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)
```

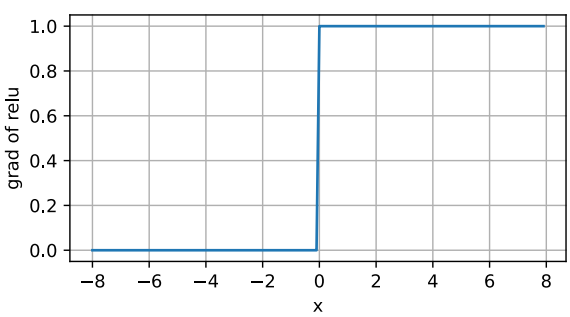



```
%matplotlib inline
import torch
from d2l import torch as d2l
```

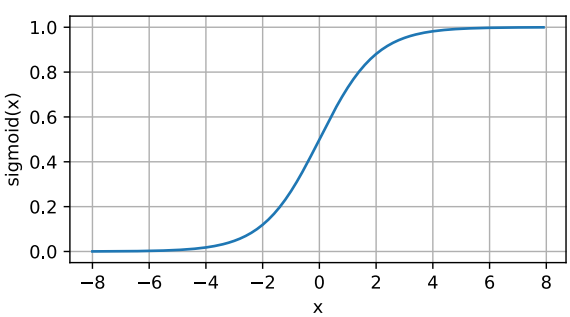
```
x = torch.arange(-8.0, 8.0, 0.1, requires_grad=True)
y = torch.relu(x)
d2l.plot(x.detach(), y.detach(), 'x', 'relu(x)', figsize=(5, 2.5))
```



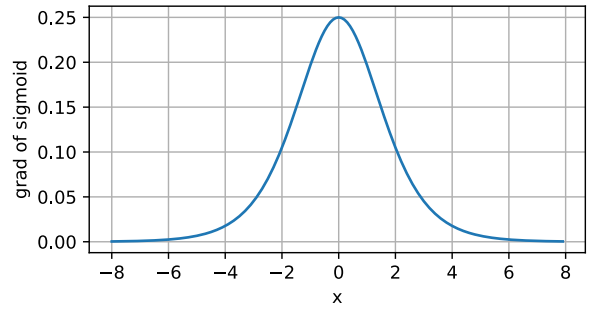

```
y.backward(torch.ones_like(x), retain_graph=True)
d2l.plot(x.detach(), x.grad, 'x', 'grad of relu', figsize=(5, 2.5))
```



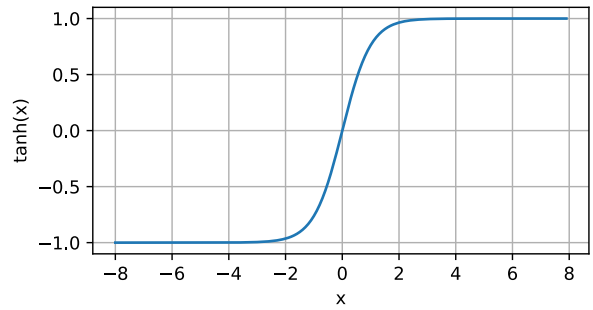

```
y = torch.sigmoid(x)
d2l.plot(x.detach(), y.detach(), 'x', 'sigmoid(x)', figsize=(5, 2.5))
```



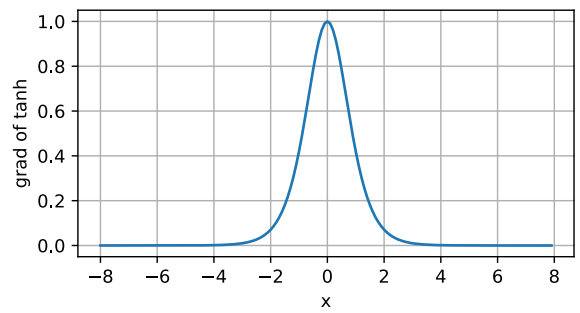

```
# Clear out previous gradients
x.grad.data.zero_()
y.backward(torch.ones_like(x),retain_graph=True)
d2l.plot(x.detach(), x.grad, 'x', 'grad of sigmoid', figsize=(5, 2.5))
```




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

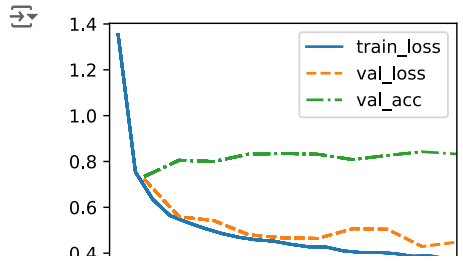



```
# Clear out previous gradients
x.grad.data.zero_()
y.backward(torch.ones_like(x),retain_graph=True)
d2l.plot(x.detach(), x.grad, 'x', 'grad of tanh', figsize=(5, 2.5))
```



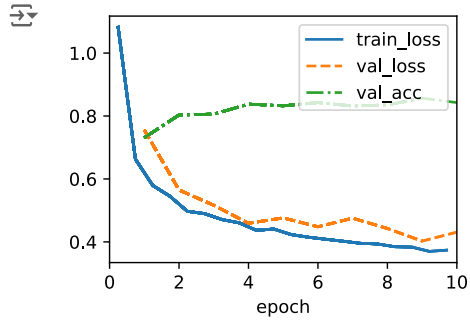

```
import torch
from torch import nn
from d2l import torch as d2l


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)
```



코딩을 시작하거나 AI로 코드를 생성하세요.

1.

broadcasting에서, 크기가 1인 axis가 있으면 그것을 확장하여, 한 tensor가 다른 tensor을 포함할 수 있는 형태로 만드는 것 같다. 예를 들어, 각 axis의 크기가 [3, 2, 1]인 a1의 axis-2의 element를 복사해 크기를 1->2 하여 3, 2, 2로 만들어서, [1, 2, 2]의 b1 (이 때 크기가 1인 axis를 무시하고 2, 2로 취급) 를 포함하도록 한 뒤, [2, 2]의 tensor에 element-wise 연산을 수행한다. (a1의 axis 1, 2와 b1의 axis 1, 2)

그런데 이런 작동과정이 비직관적으로 보인다. [3,1]의 tensor와 [1,3]의 tensor에 대한 broadcasting은 [3]의 vector에 대한 element-wise 연산이 아닌, [3,3]으로 확장한 matrix에 대한 element-wise 연산을 수행한다. broadcasting의 정확한 작동 원리는 무엇일까? 그리고 이러한 비직관적인 작동이 실제 활용에서 예기치못한 error을 일으키는 원인이 되지 않을까?

2.

우리는 optimization을 위해 Gradient Descent을 사용하며, 이 때 loss function을 미분한다. 결국 loss function의 형태가 critical point에 도달하는 결과에는 영향을 주지 않겠으나, 도달하는 과정에 영향을 줄 수 있다. 우리는 squared error를 사용했다.

또한 이러한 loss function의 형태 뿐 아니라, loss function의 input으로 들어갈 변수가 계산된 과정 또한 critical point에 도달하는 과정에 영향을 줄 수 있다. 우리는 Softmax Regression에서, model의 output에 softmax를 계산하여, squared error을 계산하는 데에 사용했다. 즉, o = Wx + b의 값에 softmax와 loss function의 두번의 가공을 거쳐, W와 b를 수정한 것이다.

이 때 사용한 softmax와 squared error는 충분히 다른 function으로 대체될 수 있다. 특정한 function이 softmax와 squared error를 대체했을 때, 어떤 방향으로 영향을 줄 수 있을까? 그리고 그러한 영향을 function의 형태로부터 예측할 수 있을까? 그리고 실제 현장에서, 그러한 차이가 유의미하게 고려될 정도로 성능에 변화가 있을까?

코딩을 시작하거나 AI로 코드를 생성하세요.