

# HW 1: Preliminaries/Linear Regression/Classification

## 2.1. Data Manipulation

```
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([[ -0.9624, -1.0942, -0.6078, -1.8756],
          [ 0.6229, -0.1928, -0.7398, -0.0637],
          [ 1.4662, -0.4817,  0.2207,  0.1356]])
```

```
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): 135969404470432
   id(Z): 135969404470432
```

```
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)

## ✓ 2.1. Discussions

### takeaway messages

reshape() 함수를 사용할 때 원하는 축의 값만 지정하고 나머지는 -1로 설정하면 자동으로 크기를 맞추어준다. reshape()는 원본 텐서가 contiguous한 경우에만 메모리를 공유하고, 그렇지 않은 경우에는 새로운 메모리 공간을 할당한다.

$X = \text{torch.randn}(2, 3)$   $Z = X.t().\text{reshape}(6)$  #  $X.t()$ 는 전치(transpose)로 인해 non-contiguous한 텐서를 만듦  $\text{print}(X.\text{data\_ptr}() == Z.\text{data\_ptr}()) ==>$  False: 메모리 공유 안 함

PyTorch, MXNet, JAX, TensorFlow 등의 프레임워크에서 텐서를 NumPy 배열로 변환하거나 그 반대로 변환하는 것은 같은 메모리를 공유함.

## ✓ 2.2. Data Preprocessing

```
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\n'
            'NA, NA, 127500\n'
            '2, NA, 106000\n'
            '4, Slate, 178100\n'
            '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)
```

↔

	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

```
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))
```

## 2.2. Discussions

### takeaway messages

조건으로 행, 열을 분리하려면 loc() 사용

**inputs = pd.get\_dummies(inputs, dummy\_na=True)** pd.get\_dummies(): 범주형 데이터를 숫자로 변환하는 함수. 여기서는 RoofType과 같은 범주형 데이터를 원 핫 인코딩으로 변환. dummy\_na=True는 결측값(NaN)을 하나의 범주로 처리하도록 지정.

**inputs = inputs.fillna(inputs.mean())** fillna(): 결측값(NaN)을 특정 값으로 대체하는 함수 inputs.mean(): 각 열의 평균값을 계산

마지막의 y가 1차원 텐서가 된 이유는 많은 딥러닝 모델이 예측하는 값이 1차원이기 때문. .view(-1,1) 또는 .unsqueeze() 메서드를 통해 (4,1)형태로 변환 가능

## 2.3. Linear Algebra

```
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
```

```
↔ 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 , 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.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.)
```

## ✓ 2.3. discussions

### takeaway messages

`torch.mv()`은 행렬과 벡터간의 곱셈 / `torch.mm()`은 행렬과 행렬간의 곱셈 / `@`은 차원 상관없이 텐서곱을 수행할 수 있는 연산자

## ✓ 2.5. Automatic Differentiation

```
import torch
```

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

```
↔ tensor([0., 1., 2., 3.])
```

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

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

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

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

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

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

↔ tensor([True, True, True, True])

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

↔ tensor([1., 1., 1., 1.])

```
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)

## ✓ 2.5. Discussions

### takeaway messages

`x = torch.arange(4.0, requires_grad=True)`처럼 한번에 생성 가능

`y.backward(gradient=torch.ones(len(y)))` 은 `y.sum().backward()`로 더 빠르게 실행할 수 있음

## 3.1. Linear Regression

```
!pip install d2l
```

숨겨진 출력 표시

```
%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]
f'{time.time() - t:.5f} sec'
```

'0.12816 sec'

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

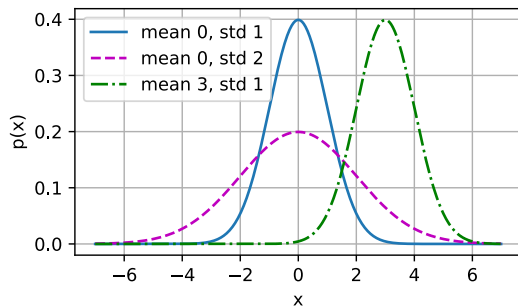
'0.00133 sec'

```
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, sigma in params])
```

0.4  
0.3  
0.2  
0.1  
0.0



## 3.1. Discussion

### takeaway messages

%matplotlib inline은 Jupyter Notebook이나 Google Colab 같은 인터랙티브 환경에서 자주 사용되는 매직 명령어

이 명령어의 의미는 그래프를 인라인(inline)으로, 즉 노트북의 출력 셀 안에 직접 표시하라는 것

벡터화 연산이 반복문보다 빠른 이유는 병렬 처리와 메모리 접근 효율성 때문

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

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

```
a = A()
```

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

a.do()
```

→ Class attribute "b" is 1

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

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

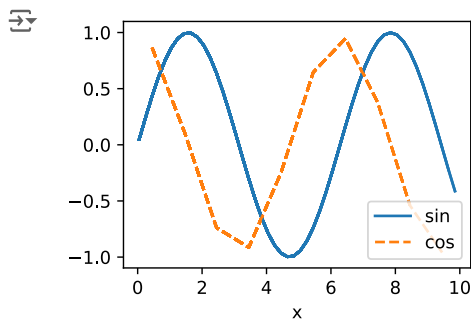
```
b = B(a=1, b=2, c=3)
```

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

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

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

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



```
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 initied'
    self.board.xlabel = 'epoch'
    if train:
        x = self.trainer.train_batch_idx / W
        self.trainer.num_train_batches
        n = self.trainer.num_train_batches / W
        self.plot_train_per_epoch
    else:
        x = self.trainer.epoch + 1
        n = self.trainer.num_val_batches / W
        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

```

## ✓ 3.2 Discussions

### takeaway messages

@add\_to\_class(A)은 파이썬에서 클래스에 속성을 추가하는 방법 중 하나. 일반적으로 이 문법은 클래스 A에 새로운 속성이나 메서드를 동적으로 추가할 때 사용.

## ✓ 3.4. Linear Regression Implementation from Scratch

```
%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_()
```

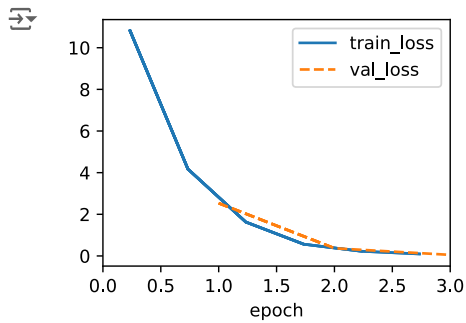
```
@d2l.add_to_class(LinearRegressionScratch)
def configure_optimizers(self):
    return SGD([self.w, self.b], self.lr)
```

```
@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:
                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}')

```

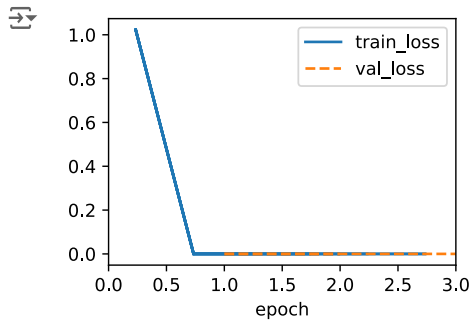
```
error in estimating w: tensor([ 0.1095, -0.1703])
error in estimating b: tensor([0.2419])
```

## 3.4. Discussions

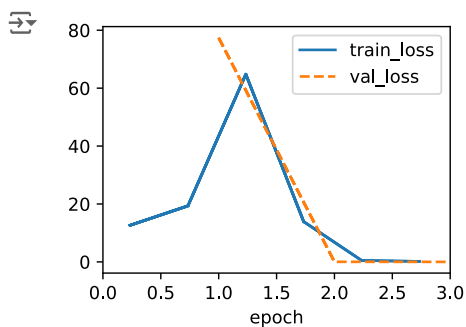
### self exercise

학습률이 회귀 모델 적합에 어떻게 영향을 주는지 알아보기 위해 학습률을 변경하여 학습시켜 보았다.

```
model3 = LinearRegressionScratch(2, lr=1)
data = d2l.SyntheticRegressionData(w=torch.tensor([2, -3.4]), b=4.2)
trainer = d2l.Trainer(max_epochs=3)
trainer.fit(model3, data)
```



```
model2 = LinearRegressionScratch(2, lr=2)
data = d2l.SyntheticRegressionData(w=torch.tensor([2, -3.4]), b=4.2)
trainer = d2l.Trainer(max_epochs=3)
trainer.fit(model2, data)
```



2이하의 학습률로 시행했을 때는 빠른 속도로 최적값에 수렴하는 것을 보이다가 2를 기준으로 손실값이 요동치는 경향을 보였고 그 이상의 학습률을 적용했을 시 손실값이 발산하는 경향을 보였다.

그런데 MLP 모델이나 분류 모델은 최적의 학습률의 범위가 작은 반면 회귀 모델의 경우 30배 이상의 차이가 나는 학습률을 적용시켜도 학습이 잘 되는 것으로 보아 학습률에 큰 민감성을 가지지는 않는 것으로 보인다. 이는 외 회귀 모델이 매우 단순한 구조를 가지고 있기 때문이라고 생각된다.

## 4.1. Softmax Regression

### ✓ 4.1. Discussions

memo

Key in its design was that we took a probabilistic approach, treating discrete categories as instances of draws from a probability distribution. As a side effect, we encountered the softmax, a convenient activation function that transforms outputs of an ordinary neural network layer into valid discrete probability distributions. We saw that the derivative of the cross-entropy loss when combined with softmax behaves very similarly to the derivative of squared error; namely by taking the difference between the expected behavior and its prediction. And, while we were only able to scratch the very surface of it, we encountered exciting connections to statistical physics and information theory.

questions

softmax에서 지수함수를 사용하는 이유는 뭘까.

이는 단순 숫자 합산을 하게 될 경우 0으로 나누게 되거나 음수가 나오는 경우가 생기기 때문에 확률을 정확하게 나타내기 어렵기 때문이다.

### ✓ 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, 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-ub
100%|██████████| 26421880/26421880 [00:02<00:00, 10555412.98it/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-ub
100%|██████████| 29515/29515 [00:00<00:00, 181392.68it/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-ub
100%|██████████| 4422102/4422102 [00:01<00:00, 3545495.66it/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-ub
100%|██████████| 5148/5148 [00:00<00:00, 14510938.84it/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
```

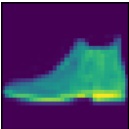
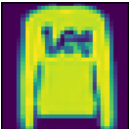
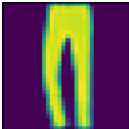
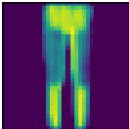
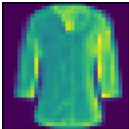
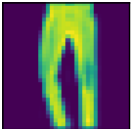
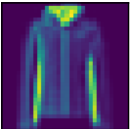
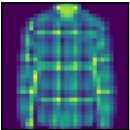
```
tic = time.time()
for X, y in data.train_dataloader():
    continue
f'{time.time() - tic:.2f} sec'
```

```
↗ '13.32 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)
```

```
↗
```

ankle boot	pullover	trouser	trouser	shirt	trouser	coat	shirt
							

## ✓ 4.2 Discussions

### takeaway messages

`iter()` 함수는 이터레이터를 반환하는 파이썬의 내장 함수

`next()` 함수는 이터레이터로부터 다음 값을 반환하는 함수 이 함수는 한 번 호출할 때마다 데이터로더로부터 하나의 배치를 불러온다. 데이터로더가 끝까지 다다르면 `StopIteration` 예외가 발생

## ✓ 4.3. The Base Classification Model

```
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
```

## ✓ 4.3 Discussions

### takeaway messages

회귀와 분류의 경우 그 구현 방식이 매우 비슷하며 이를 통해 분류가 **softmax regression**이라는 이름을 가지고 있는지 알 수 있었다.

## ✓ 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
```

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

```
↔ (tensor([[0.2424, 0.1682, 0.1118, 0.2359, 0.2417],
            [0.1830, 0.2366, 0.2428, 0.1583, 0.1794]]),
    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])
```

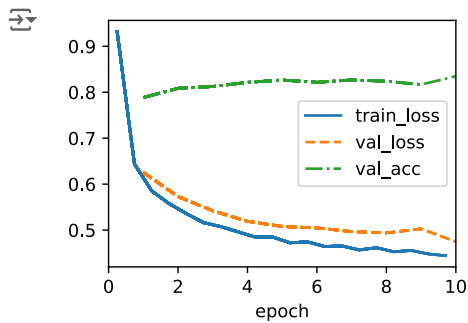
```
def cross_entropy(y_hat, y):
    return -torch.log(y_hat[list(range(len(y_hat))), y]).mean()
```

```
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)
```

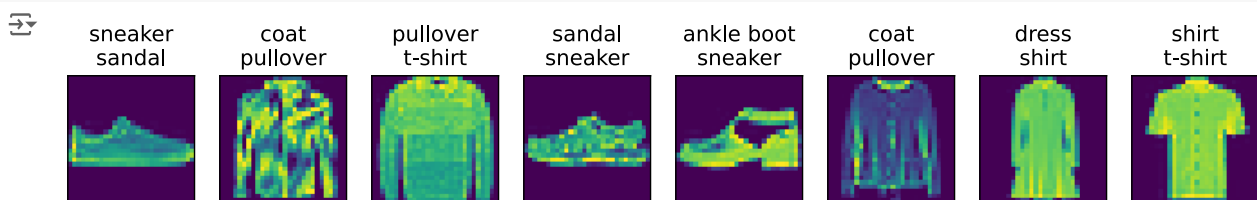
```
data = d2l.FashionMNIST(batch_size=256)
model = SoftmaxRegressionScratch(num_inputs=784, num_outputs=10, lr=0.1)
trainer = d2l.Trainer(max_epochs=10)
trainer.fit(model, data)
```



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



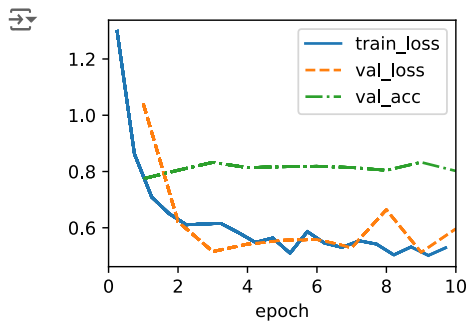
## 4.4. Discussions

### self exercise

학습률 파라미터가 모델 훈련에 얼마만큼의 영향을 미치는지 알아보기 위해 학습률을 0.3으로 조정하여 훈련시켜 보았다.

```
data = d2l.FashionMNIST(batch_size=256)
model2 = SoftmaxRegressionScratch(num_inputs=784, num_outputs=10, lr=0.3)
trainer = d2l.Trainer(max_epochs=10)
trainer.fit(model2, data)
```



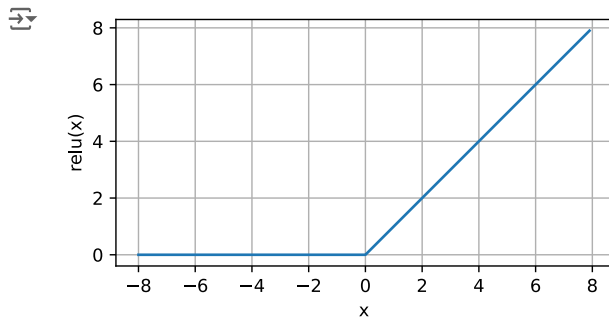


학습률이 0.1일때는 안정적으로 모델이 최적의 값에 수렴했다. 그러나 학습률이 0.3일때는 초기에 손실값이 빠르게 감소하다가 상당한 진동을 보인다. 이는 더 높은 학습률로 인해 모델이 최적의 해를 찾는 데 어려움을 겪고 있음을 나타내는 것으로 보인다.

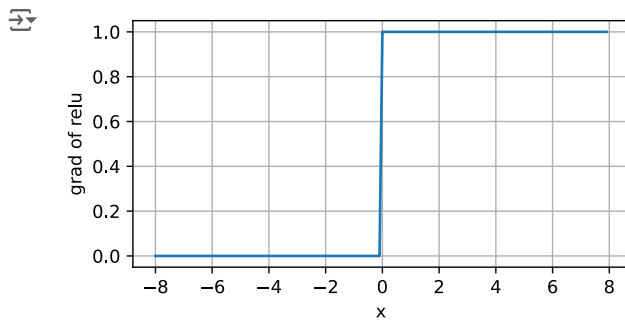
## 5.1. Multilayer Perceptrons

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

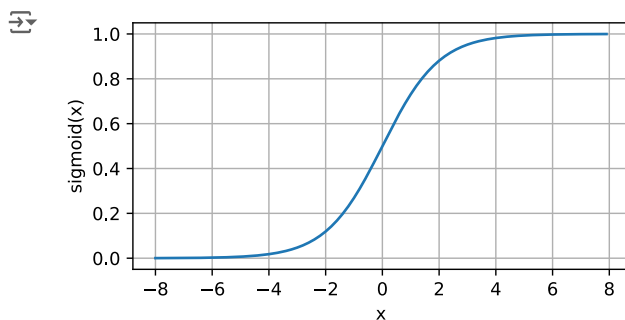
```
x = torch.arange(-8.0, 8.0, 0.1, requires_grad=True)
y = torch.relu(x)
d2l.plot(x.detach(), y.detach(), 'x', 'relu(x)', figsize=(5, 2.5))
```



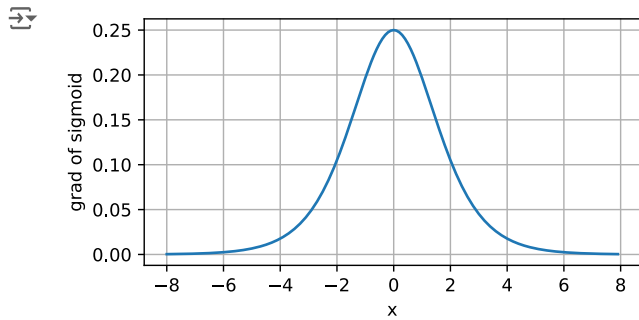
```
y.backward(torch.ones_like(x), retain_graph=True)
d2l.plot(x.detach(), x.grad, 'x', 'grad of relu', figsize=(5, 2.5))
```



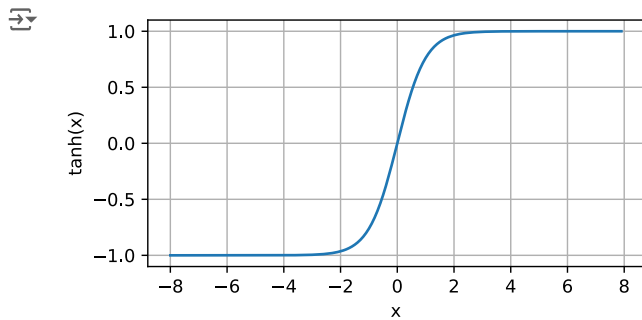
```
y = torch.sigmoid(x)
d2l.plot(x.detach(), y.detach(), 'x', 'sigmoid(x)', figsize=(5, 2.5))
```



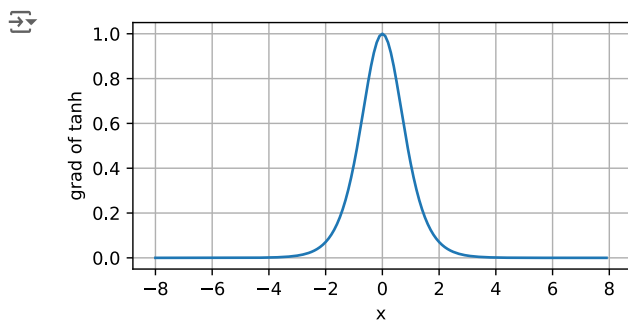
```
x.grad.data.zero_()
y.backward(torch.ones_like(x), retain_graph=True)
d2l.plot(x.detach(), x.grad, 'x', 'grad of sigmoid', figsize=(5, 2.5))
```



```
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))
```

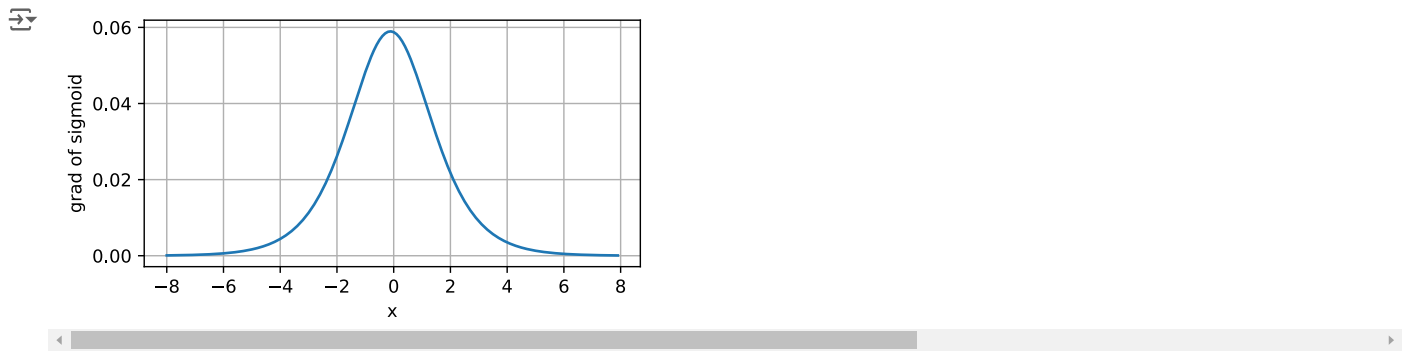


## 5.1 Discussions

### self exercise

시그모이드 활성화 함수를 2번 거칠 때 역전파의 기울기값이 어떻게 되는지 확인해보자

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



그래프를 보면 기울기 값이 엄청 감소했음을 알 수 있다. 이는 기울기 소실로 이어질 수 있음을 시각적으로 확인한 결과이다.

## question

시그모이드나  $\tanh$  함수가 Relu 함수에 비해 가지는 장점이 무엇이 있을까

=> 시그모이드 함수는 이진 분류 문제에서 확률로 해석할 수 있는 장점이 있음

=>  $\tanh$  함수는 데이터의 대칭성이나 음수 값을 처리하는데 적합함

=> RNN이나 GAN 같은 특수한 네트워크에서는 시그모이드나  $\tanh$  가 더 적합한 특성을 제공하는 경우가 있음

## ✓ 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
```

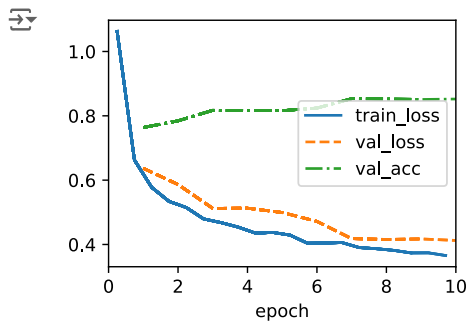
```
model = MLPScratch(num_inputs=784, num_outputs=10, num_hiddens=256, lr=0.1)
data = d2l.FashionMNIST(batch_size=256)
trainer = d2l.Trainer(max_epochs=10)
trainer.fit(model, data)
```



```
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)
```



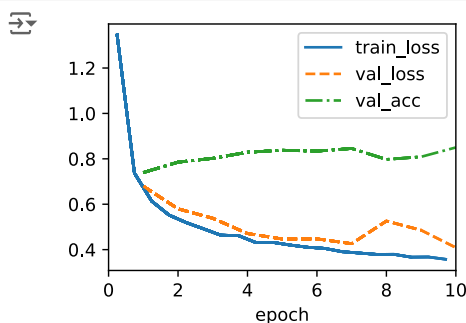
## 5.2 Discussions

### self exercise

MLP의 은닉층의 층수를 늘리고 뉴런의 개수는 동일하게 유지하였다.

```
class MLP2(d2l.Classifier):
    def __init__(self, num_outputs, num_hiddens1, num_hiddens2, lr):
        super().__init__()
        self.save_hyperparameters()
        self.net = nn.Sequential(
            nn.Flatten(),
            nn.LazyLinear(num_hiddens1),
            nn.ReLU(),
            nn.LazyLinear(num_hiddens2),
            nn.ReLU(),
            nn.LazyLinear(num_outputs)
        )
```

```
model = MLP2(10, 128, 128, 0.1)
trainer.fit(model, data)
```



train\_loss는 계속 감소하는데 val\_loss가 8epoch 부분에서 튀는걸 확인할 수 있다. 이는 과적합이 발생했음을 나타내고 이에 따라서 val\_acc도 그 이후로 잠깐 낮아졌음을 확인할 수 있다. 이는 기존 단층 히든 레이어를 사용한 결과와의 차이점이다.

## 5.3. Forward Propagation, Backward Propagation, and Computational Graphs

### 5.3 Discussions

#### Memo

Forward propagation sequentially calculates and stores intermediate variables within the computational graph defined by the neural network. It proceeds from the input to the output layer. Backpropagation sequentially calculates and stores the gradients of intermediate variables and

parameters within the neural network in the reversed order. When training deep learning models, forward propagation and backpropagation are interdependent, and training requires significantly more memory than prediction.

## ✓ question

역전파 알고리즘은 뇌의 뉴런이 작동하는 방식과는 다른 것 같다. 역전파는 오차를 계산해서 오차를 네트워크 전체로 역전파하지만, 뉴런에서는 그러한 과정이 있지는 않다. 이에 대한 개선 방식으로 로컬 학습 규칙 또는 Hebbian Learning같은 알고리즘이 있지만 여전히 역전파가 제일 효과적이다.

기울기 소실 문제처럼 반대로 기울기가 계속 커지는 문제도 발생할 수 있을 것 같다. 이는 기울기 폭발 문제라고 하는데 gradient clipping 기법 등을 사용해서 해결할 수 있다.