https://d2l.ai/chapter_preliminaries/index.html

2.1. Data Manipulation
2.2. Data Preprocessing
2.3. Linear Algebra
2.5. Automatic Differentiation
3.1. Linear Regression
3.2. Object-Oriented Design for Implementation
3.4. Linear Regression Implementation from Scratch
4.1. Softmax Regression
4.2. The Image Classification Dataset
4.3. The Base Classification Model
4.4. Softmax Regression Implementation from Scratch
5.1. Multilayer Perceptrons
5.2. Implementation of Multilayer Perceptrons
5.3. Forward Propagation, Backward Propagation, and Computational Graphs

2.1 Data Manipulation

기본적인 텐서를 생성하고 조작하는 방법을 익힌다.

```
import torch
import numpy as np
```

```
x = torch.arange(16)
print(x)

x = torch.arange(16, dtype = torch.float32)
print(x)

# New tensor
```

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

```
print(x.numel())
print(x.shape)
# Number of elements / Tensor size
```

```
16
torch.Size([16])
```

```
X = x.reshape(4,4)
print(X)
# Reshape
```

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

```
print(torch.zeros((2,3))) # All zeros
print(torch.ones((2,3))) # All ones
print(torch.randn((2,3))) # Random numbers
print(torch.tensor([[1,2,3],[4,5,6]])) # Tensor construction
```

```
tensor([[0., 0., 0.],
        [0., 0., 0.]])
tensor([[1., 1., 1.],
        [1., 1., 1.]])
tensor([[-0.0256, -1.1732,  0.6747],
        [ 0.4313, -1.4670,  0.2227]])
tensor([[1, 2, 3],
        [4, 5, 6]])
```

```
print("-----Original-----")
print(X)

print("-----Indexing-----")
```

```
print(X[1][2])
print(X[-1])

print("-----Slicing-----")
print(X[1:3])
print(X[:2, 1:])
```

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

```
torch.exp(X)
# Exponential
```

```
tensor([1.0000e+00, 2.7183e+00, 7.3891e+00, 2.0086e+01, 5.4598e+01, 1.4841e+02,
        4.0343e+02, 1.0966e+03, 2.9810e+03, 8.1031e+03, 2.2026e+04, 5.9874e+04,
        1.6275e+05, 4.4241e+05, 1.2026e+06, 3.2690e+06])
```

```
x = torch.tensor([[1,2,3,4], [2,3,4,5]])
y = x+3
print(x)
print(y)
```

```
tensor([[1, 2, 3, 4],
        [2, 3, 4, 5]])
tensor([[4, 5, 6, 7],
        [5, 6, 7, 8]])
```

```
print(x+y)
print(x-y)
print(x*y)
print(x/y)
print(x**y)
# Operation
```

```
tensor([[ 5,  7,  9, 11],
        [ 7,  9, 11, 13]])
tensor([[-3, -3, -3, -3],
        [-3, -3, -3, -3]])
tensor([[ 4, 10, 18, 28],
        [10, 18, 28, 40]])
tensor([[0.2500, 0.4000, 0.5000, 0.5714],
        [0.4000, 0.5000, 0.5714, 0.6250]])
tensor([[     1,     32,    729,  16384],
        [    32,    729,  16384, 390625]])
```

```
print(torch.cat((x,y), dim = 0))
print(torch.cat((x,y), dim = 1))
# Concatenation
```

```
tensor([[1, 2, 3, 4],
        [2, 3, 4, 5],
        [4, 5, 6, 7],
        [5, 6, 7, 8]])
tensor([[1, 2, 3, 4, 4, 5, 6, 7],
        [2, 3, 4, 5, 5, 6, 7, 8]])
```

```
print(x==y)
print(x.sum())
```

```
tensor([[False, False, False, False],
        [False, False, False, False]])
tensor(24)
```

```
a = torch.arange(3).reshape((3, 1))
b = torch.arange(2).reshape((1, 2))
print(a+b)
# Dimension broadcasting
```

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

```
x = np.array([1,2,3,4])
y = torch.tensor([1,2,3,4])
print(type(x), type(y))

x = torch.from_numpy(x)
y = y.numpy()
print(type(x), type(y))

# Object conversion
```

```
<class 'numpy.ndarray'> <class 'torch.Tensor'>
<class 'torch.Tensor'> <class 'numpy.ndarray'>
```

2.2 Data Preprocessing

Pandas의 Dataframe을 다루고, 또 csv파일을 다루는 법을 익힌다.

의문점

pd.get_dummies(inputs, dummy_na=True)는 무엇인가?

인터넷 검색을 통해 해결. 예를 들어 'Color'라는 열에 'Red', 'Blue' 등 문자열 데이터가 있을 수 있다.
이를 컴퓨터가 처리할 수 있도록 Red 여부(0 or 1), Blue 여부(0 or 1)와 같이 더미 열을 만든다.
dummy_na=True는 NaN도 포함시킬 것이라는 의미이다.

```
import os
import pandas as pd
```

```
os.getcwd()
# Current directory
```

```
'/content'
```

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

```
data = pd.read_csv(data_file)
print(data)
# Pandas csv reading
```

```
   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] # Slicing
inputs = pd.get_dummies(inputs, dummy_na=True)
print(inputs)
# dummy_na = NaN도 분류할 것인가?
```

```
   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)
# Change NaN -> Mean value
```

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

```
X = torch.tensor(inputs.to_numpy(dtype=float))
y = torch.tensor(targets.to_numpy(dtype=float))
X, y
# Pandas Dataframe to torch.tensor
```

```
(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

Vector, Matrix 등을 다루는데 이들 역시 Tensor의 일종이므로
2.1에서의 Data Manipulation과 상당 부분 겹치는 내용이다.

```
x = torch.tensor([[1,2,3],[4,5,6]], dtype=float)
print(x.T)
# Matrix Transpose
```

```
tensor([[1., 4.],
        [2., 5.],
        [3., 6.]], dtype=torch.float64)
```

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

```
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]]])
```

```
print(x.sum(axis=0))
print(x.sum(axis=1))
# Summation - Reduction
```

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

```
print(x.sum(axis=0, keepdim=True))
print(x.sum(axis=1, keepdim=True))
# Summation - Non-Reduction
```

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

```
a = torch.arange(3, dtype=float);
b = torch.ones(3, dtype=float);
print(torch.dot(a,b))
# Dot product
```

```
tensor(3., dtype=torch.float64)
```

```
torch.mv(x, a)
# Matrix-Vector product
```

```
tensor([ 8., 17.], dtype=torch.float64)
```

```
y = torch.tensor([[1,0,-1],[0,0,1],[1,1,1]], dtype=float)
torch.mm(x, y)
# Matrix-Matrix product
```

```
tensor([[ 4.,  3.,  4.],
        [10.,  6.,  7.]], dtype=torch.float64)
```

```
print(torch.norm(a))
# L2 Norm

print(abs(y))
# Absolute value
```

```
tensor(2.2361, dtype=torch.float64)
tensor([[1., 0., 1.],
        [0., 0., 1.],
        [1., 1., 1.]], dtype=torch.float64)
```

2.5 Automatic Differentiation

딥러닝에 있어서 필수적인 Backpropagation을 위해서는 Gradient 미분 계산이 필요하다.
Torch는 이러한 계산을 자동적으로 수행할 수 있도록 해 준다.
변수 x에 requires_grad=True를 설정하여 학습하는 Parameter임을 명시하면
y.backward() 함수 호출 시 x.grad에 미분계수가 축적된다.
이는 x.grad.zero_()로 초기화 할 수 있다.

```
x = torch.arange(4.0, requires_grad=True)
y = torch.dot(x, x)
```

```
y.backward()
x.grad
# (x^2)' = 2x
```

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

```
x.grad.zero_() # Reset Gradient Buffer
y = x.sum() # Change y

y.backward()
x.grad
```

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

```
x.grad.zero_()
y = x * x
# y is vector

y.sum().backward()
x.grad
# Use sum for vector gradient
```

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

```
x.grad.zero_()
y = x * x
u = y.detach()
z = u * x
# Detach => u has no ancestor

z.sum().backward()
x.grad
# (ux)' = u
```

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

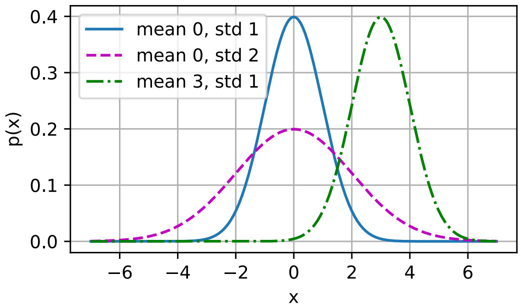
```
def f(a):
  b = a * 2
  while b.norm() < 1000:
        b = b * 2
  if b.sum() > 0:
        c = b
  else:
        c = 100 * b
  return c
# Function

a = torch.randn(size=(), requires_grad=True)
d = f(a)
d.backward()
# Function gradient

a.grad == d / a
# df/da
```

```
tensor(True)
```

3.1 Linear Regression

데이터에 가장 적합한 y=wx+b꼴의 w,b를 찾는다.
MSE Loss를 사용한다. (Gradient 계산이 쉬워지므로)

```
import math
from d2l import torch as d2l


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)
# Normal distribution


x = np.arange(-7, 7, 0.01)
params = [(0, 1), (0, 2), (3, 1)]
# mu, sigma

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

딥러닝 모델은 객체 지향(Object-Oriented)의 클래스로 정의된다.
그러나 클래스 코드는 길어지기 쉽다는 문제가 있다.
주피터 노트북, 구글 코랩 등은 코드를 짧고 간결하게 쓰는 것이 좋다.
그래서 클래스 선언 후 메소드를 이후 추가하는 등의 기능들을 만들 수 있다.

```
from torch import nn


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
# Class

a = A()
# Object


@add_to_class(A)
def do(self):
    print('Class attribute "b" is', self.b)
# Decorator(@) => Add new method

a.do()
```

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

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


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


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'
```

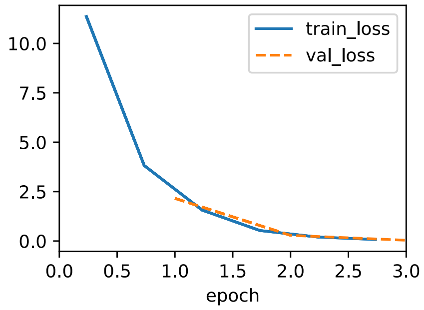
```
    def prepare_data(self, data):
        self.train_dataloader = data.train_dataloader()
        self.val_dataloader = data.val_dataloader()
        self.num_train_batches = len(self.train_dataloader)
        self.num_val_batches = (len(self.val_dataloader)
                                if self.val_dataloader 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 Linear Regression Implementation from Scratch

Stochastic Gradient Descent(SGD)를 실제로 구현해본다.

```
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

# -------------- Add methods --------------


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([ 0.1036, -0.1603])
error in estimating b: tensor([0.2145])
```

4.1 Softmax Regression

Softmax는 여러 변수들을, 확률처럼 합이 1이 되도록 만들어주는 함수이다.
딥러닝 이후 Output들은 임의의 실수 값이 될 수 있기에, Classification 등에 Softmax가 자주 쓰인다.

4.2 The Image Classification Dataset

MNIST란 사람이 쓴 숫자 이미지들을 모은 데이터셋으로, AI테스트용으로 많이 쓰여 유명하다.
여기서는 조금 더 난이도 높은 FashionMNIST라는 데이터셋을 알아본다.
셔츠, 바지, 신발 등 의류 이미지들의 데이터셋이고, 10개의 클래스로 분류를 해야 한다.

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


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()]) # Size를 조절
        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, 3711212.03it/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, 280002.90it/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, 4860642.99it/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, 12914041.26it/s]
Extracting ../data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to ../data/FashionMNIST/raw

(60000, 10000)
```

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

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

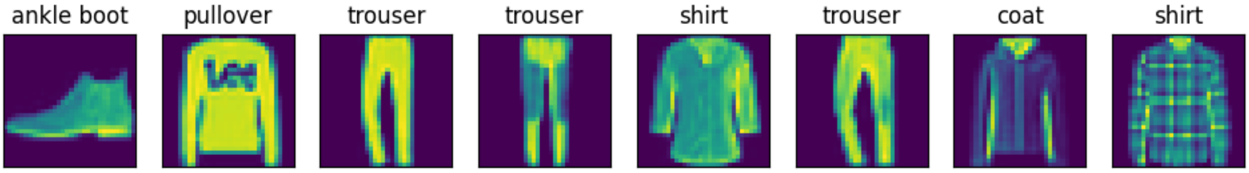
```python
@d2l.add_to_class(FashionMNIST)
def text_labels(self, indices):
    """Return text labels."""
    labels = ['t-shirt', 'trouser', 'pullover', 'dress', 'coat',
              'sandal', 'shirt', 'sneaker', 'bag', 'ankle boot']
    return [labels[int(i)] for i in indices]
```

```python
@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)
```

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

```python
@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)
```

```
/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(
```



4.3 The Base Classification Model

이제 실제로 FashionMNIST의 Classifier model을 작성해 본다.
Accuracy를 통해 성능을 보여주는 코드도 작성한다.
Accuracy란 여러 성능 지표 중 하나로, (맞은 개수)/(전체 개수)로 정의된다.
가장 단순한 형태의 성능 지표이며, 데이터 편향 등 문제가 없을 때 사용할 수 있다.

```python
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 Softmax Regression Implementation from Scratch

Softmax 및 CrossEntropy Loss의 코드를 작성한다.
Softmax는 변수들을 확률처럼, 0 이상이고 합계가 1이 되도록 변환해준다.
CrossEntropy는 딥러닝에서 자주 쓰이는 Loss로, 두 변수가 같을수록 엔트로피는 낮다.

```
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)

# Sum함수의 첫번째 인자는 어느 방향의 차원으로 더할지를 의미한다
```

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

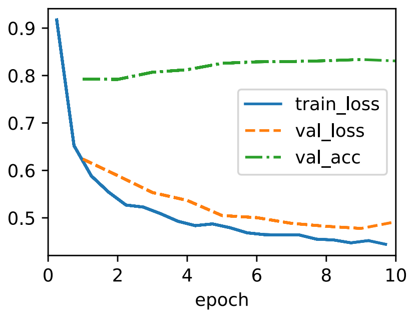
```
def softmax(X):
    X_exp = torch.exp(X)
    partition = X_exp.sum(1, keepdims=True)
    return X_exp / partition

# Softmax는 변수들을 확률의 형태로 변환해 준다.
```

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

# 확률처럼 전부 0 이상이고, 합계는 1인 것을 확인할 수 있다
```

```
(tensor([[0.2053, 0.2783, 0.1158, 0.1805, 0.2200],
         [0.1133, 0.1370, 0.2162, 0.2304, 0.3031]]),
 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]

# Classification에서, 최종 Output layer에 Softmax를 취하면,
# 각 Class일 확률을 나타내는 것으로 볼 수 있다.


@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)


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


@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)
```

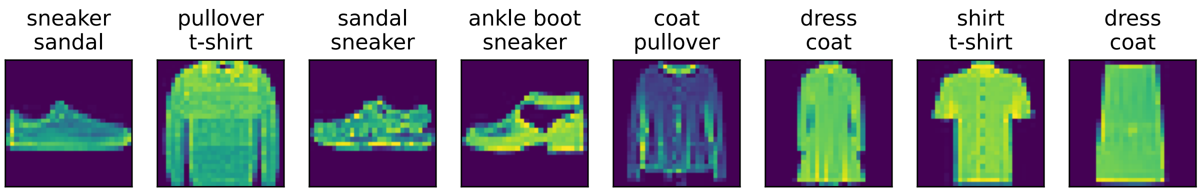



```
# ------------ 여기까지 Training ------------
```

```
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)
```



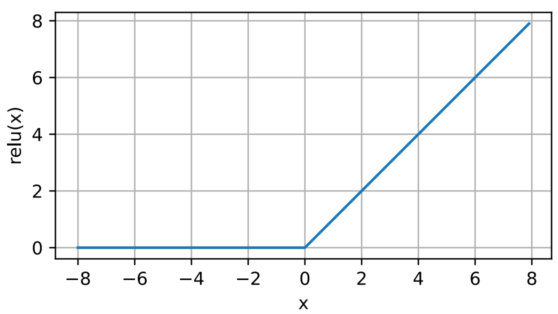

```
# ------------ 실제 Test ------------
```

5.1 Multilayer Perceptrons

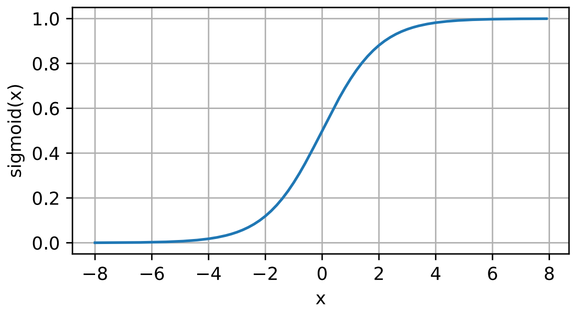
단순 Linear model은 단순한 만큼 단순한 문제만 해결할 수 있다.
Layer 수를 늘려 모델이 복잡해지면, 더 복잡한 문제도 해결할 수 있게 된다.
그러나 y=wx+b꼴의 Linear function만 있으면 아무리 여러 층이어도, 합성결과 여전히 Linear이다.
따라서 다양한 Nonlinear activation function들을 조합하는데, 이 함수들도 알아본다.

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

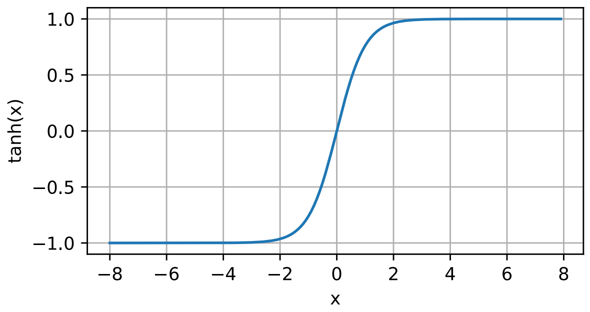
```
x = torch.arange(-8.0, 8.0, 0.1, requires_grad=True)
y = torch.relu(x)
d2l.plot(x.detach(), y.detach(), 'x', 'relu(x)', figsize=(5, 2.5))
```



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




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



5.2 Implementation of Multilayer Perceptrons

Torch의 nn(neural network)를 이용하여 실제로 구현을 해 본다.

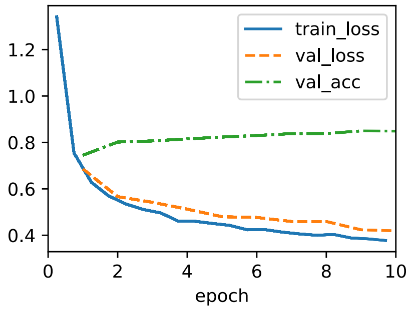
```
from torch import nn
```

```
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)
```

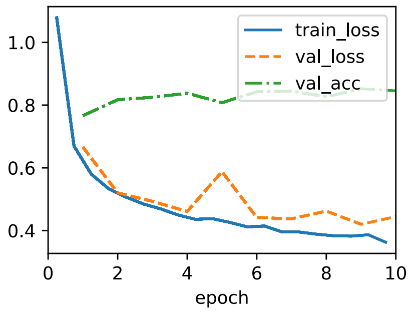
```
@d2l.add_to_class(MLPScratch)
def forward(self, X):
    X = X.reshape((-1, self.num_inputs))
    H = relu(torch.matmul(X, self.W1) + self.b1)
    return torch.matmul(H, self.W2) + self.b2
```

```
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

Forward pass는 딥러닝 모델이 계산을 하는 과정으로, 그래프 상에서 Input->Output 방향으로 이루어진다.
노드에 Input x가 들어오고, Weight를 곱하여 Activation(Wx+b)의 형태로 Output이 출력된다.
Backpropagation은 Output과 True value 사이 오차(Loss)를 통해 W를 업데이트한다.