COSE474-2024F: Deep Learning HW1

0.1 Installation

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

```
Requirement already satisfied: d2l==1.0.3 in /usr/local/lib/python3.10/dist-packages (1.0.3)
Requirement already satisfied: jupyter==1.0.0 in /usr/local/lib/python3.10/dist-packages (from d2l==1.0.3) (1.0.0)
Requirement already satisfied: numpy==1.23.5 in /usr/local/lib/python3.10/dist-packages (from d2l==1.0.3) (1.23.5)
Requirement already satisfied: matplotlib==3.7.2 in /usr/local/lib/python3.10/dist-packages (from d2l==1.0.3) (3.7.2)
Requirement already satisfied: matplotlib-inline==0.1.6 in /usr/local/lib/python3.10/dist-packages (from d2l==1.0.3) (0.1.6)
Requirement already satisfied: requests==2.31.0 in /usr/local/lib/python3.10/dist-packages (from d2l==1.0.3) (2.31.0)
Requirement already satisfied: pandas==2.0.3 in /usr/local/lib/python3.10/dist-packages (from d2l==1.0.3) (2.0.3)
Requirement already satisfied: scipy==1.10.1 in /usr/local/lib/python3.10/dist-packages (from d2l==1.0.3) (1.10.1)
Requirement already satisfied: notebook in /usr/local/lib/python3.10/dist-packages (from jupyter==1.0.0->d2l==1.0.3) (6.5.5)
Requirement already satisfied: qtconsole in /usr/local/lib/python3.10/dist-packages (from jupyter==1.0.0->d2l==1.0.3) (5.6.0)
Requirement already satisfied: jupyter-console in /usr/local/lib/python3.10/dist-packages (from jupyter==1.0.0->d2l==1.0.3) (6.1.0)
Requirement already satisfied: nbconvert in /usr/local/lib/python3.10/dist-packages (from jupyter==1.0.0->d2l==1.0.3) (6.5.4)
Requirement already satisfied: ipykernel in /usr/local/lib/python3.10/dist-packages (from jupyter==1.0.0->d2l==1.0.3) (5.5.6)
Requirement already satisfied: ipywidgets in /usr/local/lib/python3.10/dist-packages (from jupyter==1.0.0->d2l==1.0.3) (7.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.2->d2l==1.0.3) (1.3.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.2->d2l==1.0.3) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.2->d2l==1.0.3) (4.53.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.2->d2l==1.0.3) (1.4.7)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.2->d2l==1.0.3) (24.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.2->d2l==1.0.3) (10.4.0)
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.2->d2l==1.0.3) (3.0.9
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib==3.7.2->d2l==1.0.3) (2.8.2)
Requirement already satisfied: traitlets in /usr/local/lib/python3.10/dist-packages (from matplotlib-inline==0.1.6->d2l==1.0.3) (5.7.1)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas==2.0.3->d2l==1.0.3) (2024.2)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas==2.0.3->d2l==1.0.3) (2024.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests==2.31.0->d2l==1.0.3) (3.3
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests==2.31.0->d2l==1.0.3) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests==2.31.0->d2l==1.0.3) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests==2.31.0->d2l==1.0.3) (2024.8.30
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib==3.7.2->d2l==1.0.
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter==1.0.0->d2l==1.0.3) (0.
Requirement already satisfied: ipython>=5.0.0 in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter==1.0.0->d2l==1.0.3) (7.34
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter==1.0.0->d2l==1.0.3) (6.1.
Requirement already satisfied: tornado>=4.2 in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter==1.0.0->d2l==1.0.3) (6.3.3)
Requirement already satisfied: widgetsnbextension~=3.6.0 in /usr/local/lib/python3.10/dist-packages (from ipywidgets->jupyter==1.0.0->d2l==
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from ipywidgets->jupyter==1.0.0->d2l==
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from jupyter-console
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (from jupyter-console->jupyter==1.0.0->d2l==1.0.3) (2.18
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (4.9.4)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (4.12
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (6.1.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (0.7.1)
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (
Requirement already satisfied: jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (3.1.4)
Requirement already satisfied: jupyter-core>=4.7 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (5.
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (2.1
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (0.
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (0.10
Requirement already satisfied: nbformat>=5.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (5.10.4
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3)
Requirement already satisfied: tinycss2 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3) (1.3.0)
Requirement already satisfied: pyzmq<25,>=17 in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter==1.0.0->d2l==1.0.3) (24.0.1
Requirement already satisfied: argon2-cffi in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter==1.0.0->d2l==1.0.3) (23.1.0)
Requirement already satisfied: nest-asyncio>=1.5 in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter==1.0.0->d2l==1.0.3) (1.6
Requirement already satisfied: Send2Trash>=1.8.0 in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter==1.0.0->d2l==1.0.3) (1.8
```

2.1 Data Manipulation

2.1.1. Getting Started

```
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.]])
```

```
X.shape
```

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

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

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

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

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

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

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

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

```
tensor([[-0.1964,  0.7157,  1.0787,  0.2879],
        [ 0.1396, -1.1011, -1.1256, -0.9597],
        [-0.4434, -0.4670,  0.0306,  1.0650]])
```

```
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]])
```

2.1.2. Indexing and Slicing

```
X[-1], X[1:3] #-1 is the last row
```

```
(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 # :는 해당 row의 모든 원소를 의미
X
```

```
tensor([[12., 12., 12., 12.],
        [12., 12., 12., 12.],
        [ 8.,  9., 10., 11.]])
```

더블클릭 또는 Enter 키를 눌러 수정

2.1.3. Operations

```
torch.exp(x) # e^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
```

```
tensor([12., 12., 12., 12., 12., 12., 12., 12.,  8.,  9., 10., 11.])
```

```
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)) # X initilize
X
```

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

```
Y = torch.tensor([[2.0, 1, 4, 3], [1, 2, 3, 4], [4, 3, 2, 1]])
Y
```

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

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

```
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.]])
```

```
torch.cat((X,Y), dim=1) # dim =0 : axis0이 row, dim=1: axis1이 column인듯
```

```
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.)
```

2.1.4. BroadCasting

```
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 # broadcasting이 형식 안맞는거 맞게 해줌. 3*2를 알아서 만듬. 기존 원소 복사해서
```

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

2.1.5. Saving Memory

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

```
False
```

```
before = id(Y)
Y+=X  #save memory by not allocating new memory for Y+X
id(Y) == before
```

```
True
```

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

```
id(Z): 136281620011648
id(Z): 136281620011648
```

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

```
True
```

2.1.6. Conversion to Other Python Objects

```
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)
```

2.1.6. my own exercise

1. what is item() function?
2. what happens if tensor is not a single value?

1. item() is used to convert the tensor value into native Python type like int.
2. it converts into single value scalar that it makes error when the tensor is composed of more than a value

```
b = torch.tensor([3.5,3.6])
#b.item() # error
```

```
type(a.item())
```

```
float
```

2.2 Data Processing

2.2.1 Reading the Dataset

```
import os

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

```
import pandas as pd

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

```
   NumRooms RoofType   Price
0       NaN      NaN  127500
1       2.0      NaN  106000
2       4.0    Slate  178100
3       NaN      NaN  140000
```

2.2.2 Data Preparation

```
inputs, targets = data.iloc[:, 0:2], data.iloc[:, 2]
inputs = pd.get_dummies(inputs, dummy_na=True) # make NAN cateogory column
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())
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

2.2.3 Conversion to the Tensor Format

```
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))
```

2.2.4 Discussion

I agree that there are myriads of data types and data frames that it is hard to combine every related info into a single file format. Thus, it is essential to know how to convert data types from one to the other.

2.3 Linear Algebra

```
import torch
```

2.3.1 Scalars

```
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.))
```

2.3.2 Vectors

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

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

```
x[2]
```

```
tensor(2)
```

```
len(x)
```

```
3
```

```
x.shape
```

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

2.3.3 Matrices

```
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]])
```

2.3.4 Tensors

```
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]]])
```

2.3.5. Basic Properties of Tensor Arithmetic

```
A = torch.arange(6, dtype=torch.float32).reshape(2, 3)
B = A.clone()  # Assign a copy of A to B by allocating new memory
A, A + B
```

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

```
A * B # 원소끼리의 단순곱
```

```
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]))
```

2.3.6 Reduction

```
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, A, A.sum(axis=0) # asix = 0 means performing down the rows
```

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

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

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

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

```
tensor(True)
```

```
A.mean(), A.sum() / A.numel(), A.numel() # A.numel() 개수
```

```
(tensor(2.5000), tensor(2.5000), 6)
```

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

```
(tensor([1.5000, 2.5000, 3.5000]), tensor([1.5000, 2.5000, 3.5000]), 3)
```

2.3.7 NonReduction Sum

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

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

```
A / sum_A
```

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

```
A.cumsum(axis=0) # does not reduce
```

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

2.3.8 Dot Products

```
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) # same for only dot product of vectors
```

```
tensor(3.)
```

2.3.9 Matrix Vector Products

```
A.shape, x.shape, torch.mv(A, x), A@x # mv : matrix vector multiplication, @ : matrix-vector and matrix-matrix
```

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

2.3.10 Matrix-Matrix Multiplication

```
B = torch.ones(3, 4)
A, torch.mm(A, B), A@B
```

```
(tensor([[0., 1., 2.],
         [3., 4., 5.]]),
 tensor([[ 3.,  3.,  3.,  3.],
         [12., 12., 12., 12.]]),
 tensor([[ 3.,  3.,  3.,  3.],
         [12., 12., 12., 12.]]))
```

2.3.11 Norms

```
u = torch.tensor([3.0, -4.0])
torch.norm(u) # unclidean length of the vector
```

```
tensor(5.)
```

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

```
tensor(7.)
```

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

```
tensor(6.)
```

2.3.12 Exercise

Qs. Prove that the transpose of the transpose of a matrix is the matrix itself:

$$\left(A^T\right)^T = A$$

```
ATrans = A.T
ADoubleTrans = ATrans.T
A, ATrans, ADoubleTrans, A == ADoubleTrans
```

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

2.5 Automatic Differentiation

```
import torch
```

2.5.1. A simple Function

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

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

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

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

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

```
y.backward() # y 함수 x에 대해 미분.
x.grad
```

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

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

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

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

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

2.5.2 Backward for Nonscalar Variables

```
x.grad.zero_()
y = x * x
y.backward(gradient=torch.ones(len(y)))  # Faster: y.sum().backward()
x.grad # shows the result of gradient of y respect to x
```

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

2.5.3 Detaching Computation

```
x.grad.zero_()
y = x * x
u = y.detach()
z = u * x

z.sum().backward() #backward는 : 미분은 scalar 값에 대해서만 구할 수 있다.
x.grad == u
```

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

```
x.grad.zero_()
y.sum().backward()
x.grad == 2 * x
```

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

2.5.4 Gradients and Python Control Flow

```
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 # a.grad는 diffrentiation of d with respect to a
```

```
tensor(True)
```

2.5.5 Exercise

Qs. Why is the second derivative much more expensive to compute than the first derivative?

It's because you have to find the derivative of the derivative. For the second derivative, you have to calculate the derivative twice as first derivative is always needed to calculate the second derivative. And also, there should be memory allocation for the result of first derivative. In conclusion, memory and caculating expenses are both higher.

3.1 Linear Regression

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

3.1.1. Basics

Linear: Weighted Sum of the features

Loss: 1/2(squared error) : 1/2makes it convenient for derivative

Loss 최소로 하는 W :

$$\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{y}$$

Gradient Descent: weight_new = weight_old -gradient(weight_old)

3.1.2. Vectorization for Speed

```
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'
```

```
'0.23011 sec'
```

```
t = time.time()
d = a + b # 병렬 처리 능력을 활용한 벡터 연산
f'{time.time() - t:.5f} sec'
```

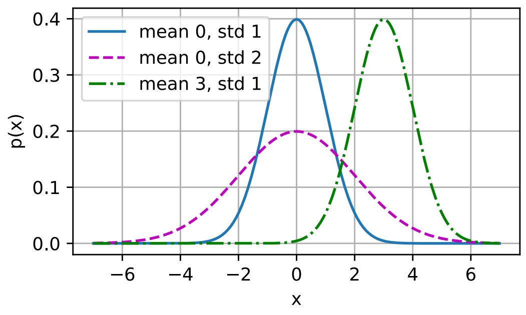
```
'0.00073 sec'
```

3.1.3. The Normal Distribution and Squared Loss

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

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

# Mean and standard deviation pairs
```

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



3.1.4. Linear Regression as a Neural Network

Linear Regression: 단순, 다중 선형 회귀 (변수 개수에 따라서)

- 차수가 1이다
- w1x + w2x ... +.... = y

3.1.5. Exercise

Qs. Can we find the optimal solution for b with using MAE not MSE?

I think the answer depends on the dataset. It sometimes give better solution but sometimes not. However, MSE is more preferred as we can differentiate it and it is more sensitive to outliers.

Qs by myself. Relation between Linear Regression and Normal Distribution

선형 회귀 모델을 구할 때, 잔차가 정규 분포 모델을 따른다고 가정하고 구한다. 그 이유는, residual이 정규 분포를 따를 때 MSE를 통해 구한 모델은 편향적이지 않으며 분산이 가장 작다. 이러한 특징은 선형 회귀 모델이 신뢰가능한 지표가 된다.

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
```

3.2.1. Utilities

```
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()
```

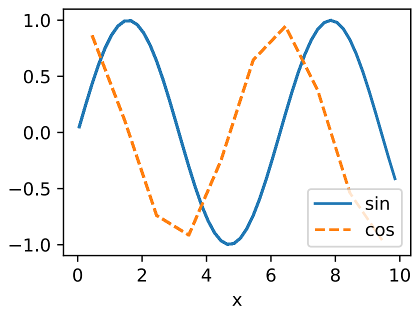
```
@add_to_class(A)
def do(self):
    print('Class attribute "b" is', self.b)

a.do()
```

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

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


# 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)
```



3.2.2 Models

```
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
```

3.2.3 Data

```
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)
```

3.2.4 Training

```
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

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

3.4.1. Defining the Model

```
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) # random tensor of randomly distributed value over normal distribution
        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
```

3.4.2. Defining the Loss Function

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

3.4.3. Defining the Optimization Algorithm

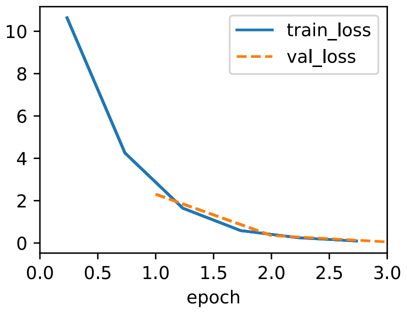
```
class SGD(d2l.HyperParameters):
    """Minibatch stochastic gradient descent."""
    def __init__(self, params, lr):
        self.save_hyperparameters()

    def step(self): # learning rate: lr, -lr*Gradient 는 weight를 optimizing하는 gradient descent rule
        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)
```

3.4.4. Training

```
@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([ 0.1455, -0.2105])
error in estimating b: tensor([0.2276])
```

3.4.5 Exercise

Qs. What would happen if we were to initialize the weights to zero. Would the algorithm still work? What if we initialized the parameters with variance 1000 rather than 0.01?

Weight가 0로 시작되면 모든 neuron의 값들이 똑같은 값을 가지고, back propagation 단계에서 똑같은 gradient를 가져 gradient descent 알고리즘이 제대로 작동하지 않을 것이다.

weight이 매우 큰 값으로 초기에 설정이 된다면, back propagation단계에서 gradient가 너무 큰 값으로 나와 가중치가 급격하게 변하여 최적화 과정이 실패할 수 있을 것이다.

4.1 Softmax Regression

4.1.1. Classification

Classification is done through **one-hot encoding**

ex: {(1,0,0), (0,1,0), (0,0,1)}

Classification 이 3개라고 가정, feature이 4개라고 가정했을 때,

- classification 개수 만큼의 affine functions가 필요하다.
- 각 function마다 4개의 feature (weight) 와 1개의 bias가 필요하다.

Softmax 는 입력받은 값을 0~1사이로 변환하여 총 확률 합이 1이 되도록 출력한다.

4.1.2. Loss Function

Corss Entrophy Loss: softmax로 표현된 값과 실제 값과의 차이를 계산하여, 예측 확률이 정답에 가까울수록 작은 loss를 주고 멀수록 큰 loss를 발생시켜 loss를 최소화하는 방향으로 모델을 학습시키는 것 같다..

4.1.3. Information Theory Basics

Entropy

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

- quantify amount of information contained in data.
- One of the fundamental theorems of information theory states that in order to encode data drawn randomly from the distribution P , we need at least H[P] “nats” to encode it (Shannon, 1948).

4.1.4 Discussion & Exercise

Why is one-hot encoding used for classification?

{1,2,3} 과 같은 natural order를 가진 값을 사용하게 되면 ML 과정에서 class 사이의 상관관계, 예를 들어 대소 등을 유추하게 되고 이는 잘못된 과정을 야기할 수 있다.

Qs. Assume that we have three classes which occur with equal probability, i.e., the probability vector is (1/3, 1/3, 1/3)?

- What is the problem if we try to design a binary code for it?
- Can you design a better code? Hint: what happens if we try to encode two independent observations? What if we encode 'n' observations jointly?

제 생각엔 binary code로 design하게 될 경우 00,01,10,11로 4가지 category가 나오는데 주어진 classification은 3개로 분류되기 때문에 적합하지 않은 것 같습니다. 2개를 합쳐 (2/3, 1/3)으로 나타내면 0,1로 표현하는 것도 괜찮은 방법 같습니다..

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()
```

4.2.1. Loading the Dataset

```
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 [00:01<00:00, 16798033.40it/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, 271092.98it/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, 5029324.31it/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, 15423054.99it/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]
```

4.2.2. Reading a Minibatch

```
@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)
```

```
torch.Size([64, 1, 32, 32]) torch.float32 torch.Size([64]) torch.int64
```

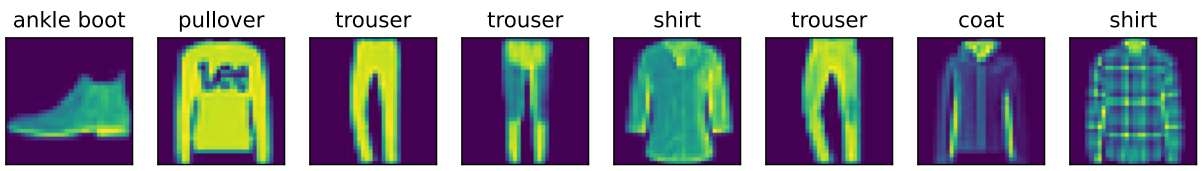
```
tic = time.time()
for X, y in data.train_dataloader():
    continue
f'{time.time() - tic:.2f} sec'
```

```
'13.61 sec'
```

4.2.3. Visualization

```
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)
```



4.2.4 Exercise

Qs. Does reducing the batch_size (for instance, to 1) affect the reading performance?

```
@d2l.add_to_class(FashionMNIST)
def get_dataloader(self, train):
    data = self.train if train else self.val
    return torch.utils.data.DataLoader(data, 1, shuffle=train,
                                       num_workers=self.num_workers)
```

배치 사이즈가 클수록 메모리에 한번에 로드하기 때문에 배치 사이즈가 1이면 그만큼 읽어오는 작업이 많아져 읽기 성능이 떨어집니다. 그래서 reading performance가 현저히 떨어집니다. 실제로 7초가 걸리던 것이 142초가 걸렸습니다.

4.3. The Base Classification Model

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

4.3.1. The Classifier Class

```
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)
```

4.3.2. Accuracy

```
@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

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

4.4.1. The Softmax

```
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.]]))
```

$$\mathrm{softmax}(X)_{ij} = \frac{\exp(X_{ij})}{\sum_k \exp(X_{ik})}$$

- e^x/ 총합 of e^x

```
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) # axis=1 column끼리의 합
```

```
(tensor([[0.1853, 0.3537, 0.1402, 0.1649, 0.1559],
         [0.1316, 0.2631, 0.2092, 0.2412, 0.1549]]),
 tensor([1.0000, 1.0000]))
```

4.4.2. The Model

Softmax Regressioin Model: 입력 데이터에 대해 선형 변환 후, softmax를 통해 0~1사이의 값으로 변환한다.

```
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)
```

4.4.3. The Cross-Entropy Loss

```
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])
```

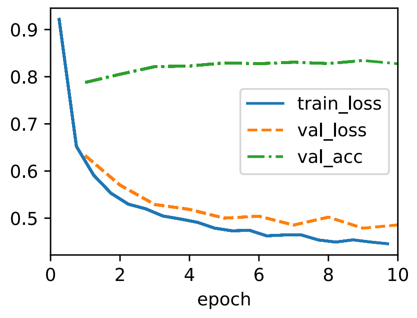
```
def cross_entropy(y_hat, y):
    return -torch.log(y_hat[list(range(len(y_hat))), y]).mean()

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)
```

4.4.4. Training

```
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)
```

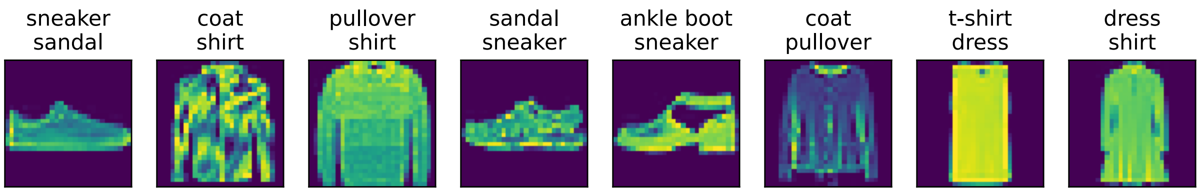


4.4.5. Prediction

```
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)
```



4.4.6 Exercise

Qs. In this section, we directly implemented the softmax function based on the mathematical definition of the softmax operation. As discussed in Section 4.1 this can cause numerical instabilities.

Test whether softmax still works correctly if an input has a value of 100 .

Test whether softmax still works correctly if the largest of all inputs is smaller than -100 ?

Implement a fix by looking at the value relative to the largest entry in the argument.

```
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.tensor([[100, 200, 50], [30,20,1]], dtype=torch.float32)
X_prob = softmax(X)
X_prob, X_prob.sum(1) # axis=1 column끼리의 합
```

```
(tensor([[       nan,        nan, 0.0000e+00],
         [9.9995e-01, 4.5398e-05, 2.5436e-13]]),
 tensor([nan, 1.]))
```

It does not work when an input has a value of 100. The value gets too big for e^x.

```
X = torch.tensor([[-101, -104, -110], [-200,-300,-400]], dtype=torch.float32)
X_prob = softmax(X)
X_prob, X_prob.sum(1) # axis=1 column끼리의 합
```

```
(tensor([[1., 0., 0.],
         [nan, nan, nan]]),
 tensor([1., nan]))
```

It does not work when all input values are smaller than -100.

The value e^x gets close to 0.

```
def softmax(X):
    X_shifted = X - X.max(dim=1, keepdim=True)[0]
    X_exp = torch.exp(X_shifted)
    partition = X_exp.sum(dim=1, keepdim=True)
    print(X_shifted)
    return X_exp / partition
```

```
X = torch.tensor([[100, 200, 50], [30,20,1]], dtype=torch.float32)
X_prob = softmax(X)
X_prob, X_prob.sum(1) # axis=1 column끼리의 합
```

```
tensor([[-100.,    0., -150.],
        [   0.,  -10.,  -29.]])
(tensor([[3.7835e-44, 1.0000e+00, 0.0000e+00],
         [9.9995e-01, 4.5398e-05, 2.5436e-13]]),
 tensor([1., 1.]))
```

```
X = torch.tensor([[-101, -104, -110], [-200,-300,-400]], dtype=torch.float32)
X_prob = softmax(X)
X_prob, X_prob.sum(1) # axis=1 column끼리의 합
```

```
tensor([[   0.,   -3.,   -9.],
        [   0., -100., -200.]])
(tensor([[9.5246e-01, 4.7420e-02, 1.1754e-04],
         [1.0000e+00, 3.7835e-44, 0.0000e+00]]),
 tensor([1.0000, 1.0000]))
```

각 row마다 최댓값을 찾아서 모든 input value에서 빼주는 과정을 통해 새로운 data set을 만들어 사용한다.

그 이유는 가장 큰 값을 빼줌으로서 값이 너무 크거나 작지 않게 해주는 동시에 값의 상대성을 해치지 않을 수 있다.

5.1 Multilayer Perceptrons

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

5.1.1 Hidden Layers

Linear Model의 한계가 있기 때문에 더 잘 학습시키기 위하여 더 complex하게 학습시키기 위해 hidden layers를 도입함.

Activation function을 사용해 Hidden Layers 층들이 linear 하지 않고 nonlinear하게 되도록 함.

- linear model은 y= ax+b의 꼴로 1차 함수로 이루어짐.
- a: 선형 변환 matrix, x: input vector, b: 평행 이동을 나타내는 vector
- activation function이 없으면 hidden을 쌓아도 결국 linear이 됨.

5.1.2 Activation Functions

Decides whether neuron should be activated or not.

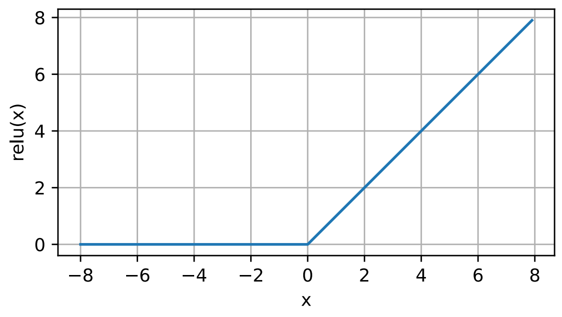
5.1.2.1 ReLU Fuction

1. Universial Approximators

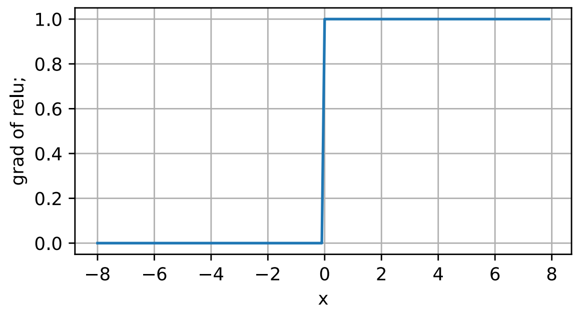
- A single-hidden-layer netwrok, given enough nodes (possibly absurdly many), and the right set of weights, we can model any function.
- Kernel methods are way more effective, since they are capable of solving the problem exactly even in **infinite dimensional spalces** (Kimeldorf and Wahba), "Universial Approxiamtion Theorem"
- We can approximate many functions more **compactly** by using deeper(rather than wider) networks.(Simonyan and Zisserman, 2014).

$$\mathrm{ReLU}(x) = \max(x, 0)$$

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



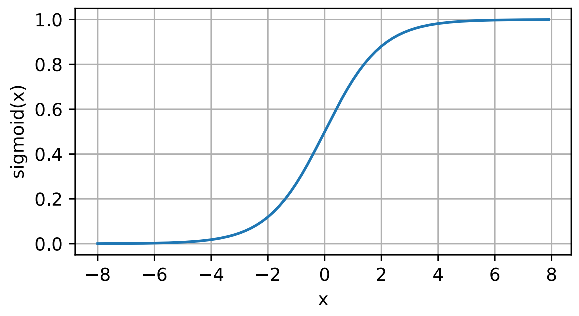
```
y.backward(torch.ones_like(x), retain_graph=True)
d2l.plot(x.detach(), x.grad, 'x', 'grad of relu;', figsize=(5,2.5))
```



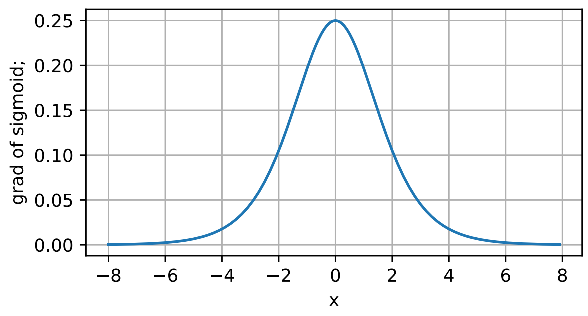
5.1.2.2 Sigmoid Function

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

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



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

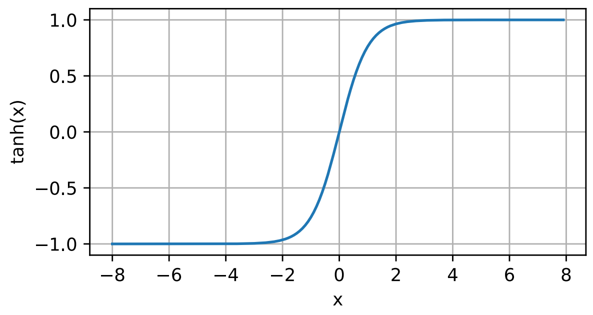


- We prefer ReLU because the biggest gradient value of sigmoide is 0.25 that when it is multiplied several times it becomes almost 0.
- Gradient Vanishing Problem ~ Backpropagation / Activate functions

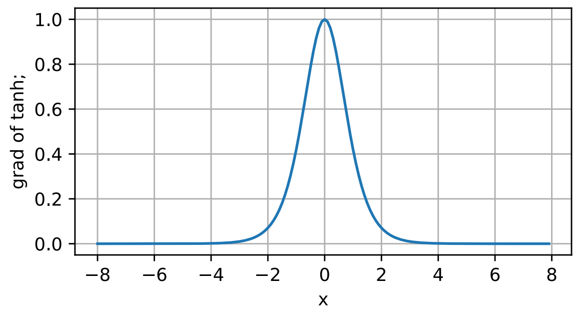
5.1.2.3. Tanh Function (Discussion)

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

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



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



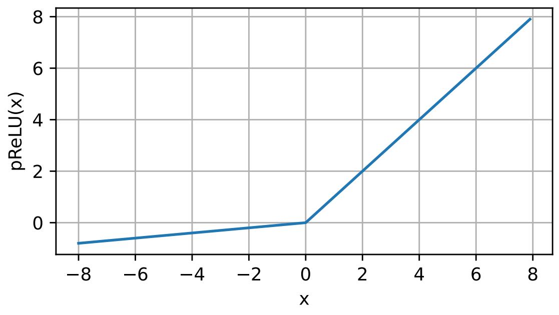
5.1.3 (Exercise) My own exercise/experiment

$$\mathrm{pRelu}(x) = \max(0, x) + \alpha \min(0, x)$$

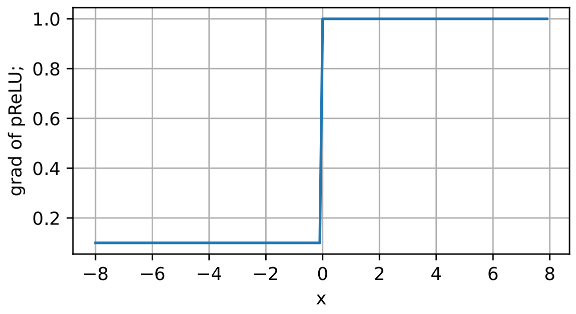
```
pReLU = lambda x, a: torch.max(torch.tensor(0), x) + a * torch.min(torch.tensor(0), x)
```

```
y= pReLU(x=x, a= 0.1)
```

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



```
x.grad.data.zero_() #x에 누적된 미분 초기화
y.backward(torch.ones_like(x), retain_graph=True)
d2l.plot(x.detach(), x.grad, 'x', 'grad of pReLU;', figsize=(5,2.5))
```



5.2. Implementation of Multilayer Perceptrons

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

5.2.1. Implementation from Scratch

5.2.1.1. Initializing Model Parameters

```
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))
```

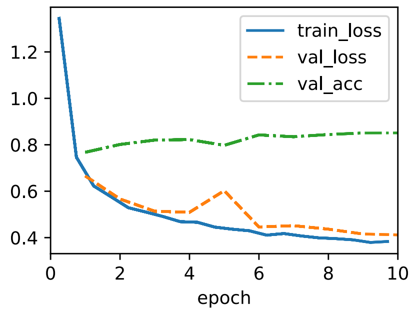
5.2.1.2. Model

```
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
```

5.2.1.3. Training

```
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)
```



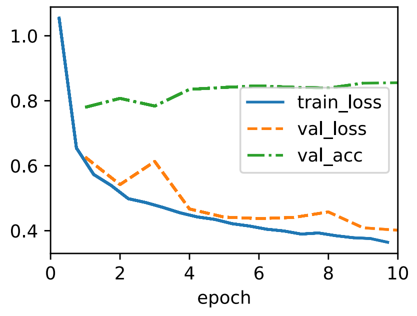
5.2.2. Concise Implementation

5.2.2.1. Model

```
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))
```

5.2.2.2. Training

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



5.3. Forward Propagation, Backward Propagation, and Computational Graphs

5.3.1. Forward Propagation

- Forward propagation (or forward pass) refers to the calculation and storage of intermediate variables (including outputs) for a neural network in order from the input layer to the output layer.

5.3.2. Computational Graph of Forward Propagation

5.3.3. Backpropagation

- Backpropagation refers to the method of calculating the gradient of neural network parameters.
- In short, the method traverses the network in reverse order, from the output to the input layer, according to the chain rule from calculus.
- 여기서 구한 gradient가 descent gradient algorithm에 쓰인다.

5.3.4. Training Neural Networks

- traning을 위해서 forward와 backward 가 모두 쓰임
- forward를 먼저 하고, 구한 값들을 사용해 backward를 할 수 있음.
- Therefore when training neural networks, once model parameters are initialized, we alternate forward propagation with backpropagation, updating model parameters using gradients given by backpropagation.

5.3.5. Exercise

Qs. Assume that the inputs X to some scalar function f are n x m matrices. What is the dimensionality of the gradient of f with respect to X ?

처음에는 Dimension과 degree의 개념이 헷갈려서 n x (m-1)인줄 알았으나, 미분을 통해 degree 차수는 줄어들지만 dimension은 줄어들지 않는다는 것을 깨달았다. f의 gradient는 X가 입력일 때, X의 각 원소에 대한 편미분 값으로, X의 gradient의 dimension은 똑같이 n x m으로 유지된다.

Discussions & Exercises

2.1.6. my own exercise

1. what is item() function?
2. what happens if tensor is not a single value?

1. item() is used to convert the tensor value into native Python type like int.
2. it converts into single value scalar that it makes error when the tensor is composed of more than a value

```
b = torch.tensor([3.5,3.6])
#b.item() # error
```

```
type(a.item())
```

```
float
```

2.2.4 Discussion

I agree that there are myriads of data types and data frames that it is hard to combine every related info into a single file format. Thus, it is essential to know how to convert data types from one to the other.

2.3.12 Exercise

Qs. Prove that the transpose of the transpose of a matrix is the matrix itself:

$$\left(A^T\right)^T = A$$

```
ATrans = A.T
ADoubleTrans = ATrans.T
A, ATrans, ADoubleTrans, A == ADoubleTrans
```

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

2.5.5 Exercise

Qs. Why is the second derivative much more expensive to compute than the first derivative?

It's because you have to find the derivative of the derivative. For the second derivative, you have to calculate the derivative twice as first derivative is always needed to calculate the second derivative. And also, there should be memory allocation for the result of first derivative. In conclusion, memory and caculating expenses are both higher.

3.1.5. Exercise

Qs. Can we find the optimal solution for b with using MAE not MSE?

I think the answer depends on the dataset. It sometimes give better solution but sometimes not. However, MSE is more preferred as we can differentiate it and it is more sensitive to outliers.

Qs by myself. Relation between Linear Regression and Normal Distribution

선형 회귀 모델을 구할 때, 잔차가 정규 분포 모델을 따른다고 가정하고 구한다. 그 이유는, residual이 정규 분포를 따를 때 MSE를 통해 구한 모델은 편향적이지 않으며 분산이 가장 작다. 이러한 특징은 선형 회귀 모델이 신뢰가능한 지표가 된다.

3.4.5 Exercise

Qs. What would happen if we were to initialize the weights to zero. Would the algorithm still work? What if we initialized the parameters with variance 1000 rather than 0.01?

Weight가 0로 시작되면 모든 neuron의 값들이 똑같은 값을 가지고, back propagation 단계에서 똑같은 gradient를 가져 gradient descent 알고리즘이 제대로 작동하지 않을 것이다.

weight이 매우 큰 값으로 초기에 설정이 된다면, back propagation단계에서 gradient가 너무 큰 값으로 나와 가중치가 급격하게 변하여 최적화 과정이 실패할 수 있을 것이다.

4.1.4 Discussion & Exercise

Why is one-hot encoding used for classification?

{1,2,3} 과 같은 natural order를 가진 값을 사용하게 되면 ML 과정에서 class 사이의 상관관계, 예를 들어 대소 등을 유추하게 되고 이는 잘못된 과정을 야기할 수 있다.

Qs. Assume that we have three classes which occur with equal probability, i.e., the probability vector is (1/3, 1/3, 1/3)?

- What is the problem if we try to design a binary code for it?
- Can you design a better code? Hint: what happens if we try to encode two independent observations? What if we encode 'n' observations jointly?

제 생각엔 binary code로 design하게 될 경우 00,01,10,11로 4가지 category가 나오는데 주어진 classification은 3개로 분류되기 때문에 적합하지 않은 것 같습니다. 2개를 합쳐 (2/3, 1/3)으로 나타내면 0,1로 표현하는 것도 괜찮은 방법 같습니다..

4.2.4 Exercise

Qs. Does reducing the batch_size (for instance, to 1) affect the reading performance?

```
@d2l.add_to_class(FashionMNIST)
def get_dataloader(self, train):
    data = self.train if train else self.val
    return torch.utils.data.DataLoader(data, 1, shuffle=train,
                                       num_workers=self.num_workers)
```

배치 사이즈가 클수록 메모리에 한번에 로드하기 때문에 배치 사이즈가 1이면 그만큼 읽어오는 작업이 많아져 읽기 성능이 떨어집니다. 그래서 reading performance가 현저히 떨어집니다. 실제로 7초가 걸리던 것이 142초가 걸렸습니다.

4.4.6 Exercise

Qs. In this section, we directly implemented the softmax function based on the mathematical definition of the softmax operation. As discussed in Section 4.1 this can cause numerical instabilities.

Test whether softmax still works correctly if an input has a value of 100 .

Test whether softmax still works correctly if the largest of all inputs is smaller than -100 ?

Implement a fix by looking at the value relative to the largest entry in the argument.

```
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.tensor([[100, 200, 50], [30,20,1]], dtype=torch.float32)
X_prob = softmax(X)
X_prob, X_prob.sum(1) # axis=1 column끼리의 합
```

```
(tensor([[       nan,        nan, 0.0000e+00],
         [9.9995e-01, 4.5398e-05, 2.5436e-13]]),
 tensor([nan, 1.]))
```

It does not work when an input has a value of 100. The value gets too big for e^x.

```
X = torch.tensor([[-101, -104, -110], [-200,-300,-400]], dtype=torch.float32)
X_prob = softmax(X)
X_prob, X_prob.sum(1) # axis=1 column끼리의 합
```

```
(tensor([[1., 0., 0.],
         [nan, nan, nan]]),
 tensor([1., nan]))
```

It does not work when all input values are smaller than -100.

The value e^x gets close to 0.

```
def softmax(X):
    X_shifted = X - X.max(dim=1, keepdim=True)[0]
    X_exp = torch.exp(X_shifted)
    partition = X_exp.sum(dim=1, keepdim=True)
    print(X_shifted)
    return X_exp / partition
```

```
X = torch.tensor([[100, 200, 50], [30,20,1]], dtype=torch.float32)
X_prob = softmax(X)
X_prob, X_prob.sum(1) # axis=1 column끼리의 합
```

```
tensor([[-100.,    0., -150.],
        [   0.,  -10.,  -29.]])
(tensor([[3.7835e-44, 1.0000e+00, 0.0000e+00],
         [9.9995e-01, 4.5398e-05, 2.5436e-13]]),
 tensor([1., 1.]))
```

```
X = torch.tensor([[-101, -104, -110], [-200,-300,-400]], dtype=torch.float32)
X_prob = softmax(X)
X_prob, X_prob.sum(1) # axis=1 column끼리의 합
```

```
tensor([[   0.,   -3.,   -9.],
        [   0., -100., -200.]])
(tensor([[9.5246e-01, 4.7420e-02, 1.1754e-04],
         [1.0000e+00, 3.7835e-44, 0.0000e+00]]),
 tensor([1.0000, 1.0000]))
```

각 row마다 최댓값을 찾아서 모든 input value에서 빼주는 과정을 통해 새로운 data set을 만들어 사용한다.

그 이유는 가장 큰 값을 빼줌으로서 값이 너무 크거나 작지 않게 해주는 동시에 값의 상대성을 해치지 않을 수 있다.

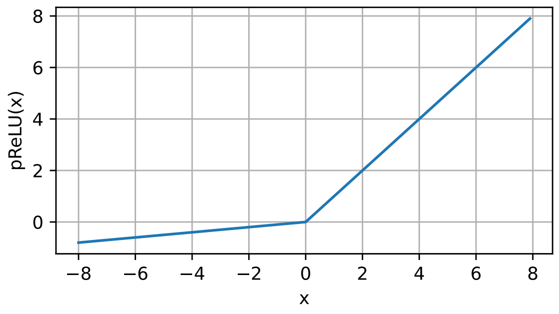
5.1.3 (Exercise) My own exercise/experiment

$$\mathrm{pRelu}(x) = \max(0, x) + \alpha \min(0, x)$$

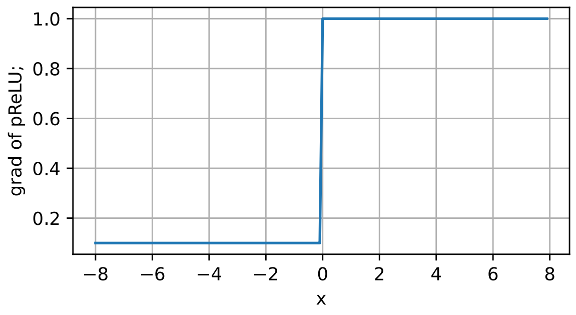
```
pReLU = lambda x, a: torch.max(torch.tensor(0), x) + a * torch.min(torch.tensor(0), x)
```

```
y= pReLU(x=x, a= 0.1)
```

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



```
x.grad.data.zero_() #x에 누적된 미분 초기화
y.backward(torch.ones_like(x), retain_graph=True)
d2l.plot(x.detach(), x.grad, 'x', 'grad of pReLU;', figsize=(5,2.5))
```



5.3.5. Exercise

Qs. Assume that the inputs X to some scalar function f are n x m matrices. What is the dimensionality of the gradient of f with respect to X ?

처음에는 Dimension과 degree의 개념이 헷갈려서 n x (m-1)인줄 알았으나, 미분을 통해 degree 차수는 줄어들지만 dimension은 줄어들지 않는다는 것을 깨달았다. f의 gradient는 X가 입력일 때, X의 각 원소에 대한 편미분 값으로, X의 gradient의 dimension은 똑같이 n x m으로 유지된다.