In [ ]:
```
import torch
```

In [ ]:
```
x = torch.arange(12, dtype=torch.float32)
x
```

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

In [ ]:
```
x.numel()
```

Out[ ]:
```
12
```

In [ ]:
```
x.shape
```

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

In [ ]:
```
X = x.reshape(3, 4)
X
```

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

In [ ]:
```
torch.zeros((2, 3, 4))
```

Out[ ]:
```
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.]]])
```

In [ ]:
```
torch.ones((2, 3, 4))
```

Out[ ]:
```
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.]]])
```

In [ ]:
```
torch.randn(3, 4)
```

Out[ ]:
```
tensor([[ 0.0606,  1.0689, -0.8443, -0.5900],
        [ 2.1162,  0.6721, -0.4968, -0.5011],
        [ 1.1143, -1.6892,  0.1058, -0.7911]])
```

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

Out[ ]:
```
tensor([[2, 1, 4, 3],
        [1, 2, 3, 4],
        [4, 3, 2, 1]])
```

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

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

In [ ]:
```
X[1, 2] = 17
X
```

Out[ ]:
```
tensor([[ 0.,  1.,  2.,  3.],
        [ 4.,  5., 17.,  7.],
        [ 8.,  9., 10., 11.]])
```

In [ ]:
```
X[:2, :] = 12
X
```

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

In [ ]:
```
torch.exp(x)
```

Out[ ]:
```
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])
```

In [ ]:
```
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
```

Out[ ]:
```
(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.]))
```

In [ ]:
```
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)
```

Out[ ]:
```
(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.]]))
```

In [ ]:
```
X == Y
```

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

In [ ]:
```
X.sum()
```

Out[ ]:
```
tensor(66.)
```

In [ ]:
```
a = torch.arange(3).reshape((3, 1))
b = torch.arange(2).reshape((1, 2))
a, b
```

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

In [ ]:
```
a + b
```

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

In [ ]:
```
before = id(Y)
Y = Y + X
id(Y) == before
```

Out[ ]:
```
False
```

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

```
id(Z): 134274196315232
id(Z): 134274196315232
```

In [ ]:
```
before = id(X)
X += Y
id(X) == before
```

Out[ ]:
```
True
```

In [ ]:
```
A = X.numpy()
B = torch.from_numpy(A)
type(A), type(B)
```

Out[ ]:
```
(numpy.ndarray, torch.Tensor)
```

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

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

2.1 discussion Broadcasting이라는 개념을 도입하여 효율적인 다차원 연산을 가능하게 한 것이 흥미로웠다. 이 방식은 배열의 크기를 자동으로 맞춰주기 때문에, 다른 크기의 배열 간에도 연산이 가능하다는 점에서 유용하다. 하지만 사실 배열의 연산을 위해 억지로 크기를 늘려주는 것이기 때문에, 의도한 결과와 다르게 나올 가능성이 있지 않을까 하는 의심이 들었다. 특히, 배열의 크기나 차원이 명확하지 않다면

예상치 못한 방식으로 결과가 나올 수 있기 때문에 조심해야 할 것 같다.

이를 방지하려면 엄밀한 계산을 위해서는 배열의 크기와 차원을 명확히 정의하고 사용하는 것이 중요할 것 같다. Broadcasting이 자동으로 배열을 늘려주기 때문에 편리한 점도 있지만, 이런 자동화된 과정이 항상 올바른 결과를 보장하는 것은 아니라고 생각된다. 배열의 구조를 확실히 파악하고, 필요하다면 명시적으로 배열을 조정해주는 것이 더 안전한 방법일 것 같다.

In [ ]:
```
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''')
```

In [ ]:
```
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
```

In [ ]:
```
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
```

In [ ]:
```
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
```

In [ ]:
```
import torch

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

Out[ ]:
```
(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 discussion 이 부분에서는 숫자상의 missing values를 발견했을 때 그 결측값을 포함한 열의 평균값으로 대체하는 방법이 있다는 사실이 흥미로웠다. 이 방법은 데이터를 완벽하게 복구하는 것은 아니지만, 실용적인 해결책으로서 널리 사용된다는 점에서 의미가 있다. 결측값을 단순히 제거하기보다는 평균값으로 대체하는 방식은 데이터 손실을 최소화하면서 분석을 계속할 수 있게 해준다. 물론 이 방법은 앞선 방법과 마찬가지로 엄밀하지 않다. 결측값이 중요한 패턴을 포함하고 있을 경우, 단순한 평균값으로 대체하는 것이 데이터의 의미를 왜곡할 가능성도 있을 것 같다.

하지만 이런 점 때문에 오히려 더 흥미로운 분야라고 생각한다. 결측값을 처리하는 방법에는 다양한 접근이 존재하며, 앞으로 더 정교하고 상황에 맞는 방법들이 개발될 가능성이 크기 때문이다. 현재의 방식은 간단하면서도 실용적이지만, 머신러닝이나 인공지능을 활용한 더 복잡한 결측값 처리 방식들이 등장한다면 데이터 전처리의 정확도와 효율성 모두에서 큰 발전을 이룰 수 있을 것이다.

In [ ]:
```
import torch
```

In [ ]:
```
x = torch.tensor(3.0)
y = torch.tensor(2.0)

x + y, x * y, x / y, x**y
```

Out[ ]:
```
(tensor(5.), tensor(6.), tensor(1.5000), tensor(9.))
```

In [ ]:
```
x = torch.arange(3)
x
```

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

In [ ]:
```
x[2]
```

Out[ ]:
```
tensor(2)
```

In [ ]:
```
len(x)
```

Out[ ]:
```
3
```

In [ ]:
```
x.shape
```

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

In [ ]:
```
A = torch.arange(6).reshape(3, 2)
A
```

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

In [ ]:
```
A.T
```

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

In [ ]:
```
A = torch.tensor([[1, 2, 3], [2, 0, 4], [3, 4, 5]])
A == A.T
```

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

In [ ]:
```
torch.arange(24).reshape(2, 3, 4)
```

Out[ ]:
```
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]]])
```

In [ ]:
```
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
```

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

In [ ]:
```
A * B
```

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

In [ ]:
```
a = 2
X = torch.arange(24).reshape(2, 3, 4)
a + X, (a * X).shape
```

Out[ ]:
```
(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]))
```

In [ ]:
```
x = torch.arange(3, dtype=torch.float32)
x, x.sum()
```

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

In [ ]:
```
A.shape, A.sum()
```

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

In [ ]:
```
A.shape, A.sum(axis=0).shape
```

Out[ ]: (torch.Size([2, 3]), torch.Size([3]))

In [ ]:
```
A.shape, A.sum(axis=1).shape
```

Out[ ]: (torch.Size([2, 3]), torch.Size([2]))

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

Out[ ]: tensor(True)

In [ ]:
```
A.mean(), A.sum() / A.numel()
```

Out[ ]: (tensor(2.5000), tensor(2.5000))

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

Out[ ]: (tensor([1.5000, 2.5000, 3.5000]), tensor([1.5000, 2.5000, 3.5000]))

In [ ]:
```
sum_A = A.sum(axis=1, keepdims=True)
sum_A, sum_A.shape
```

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

In [ ]:
```
A / sum_A
```

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

In [ ]:
```
A.cumsum(axis=0)
```

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

In [ ]:
```
y = torch.ones(3, dtype = torch.float32)
x, y, torch.dot(x, y)
```

Out[ ]: (tensor([0., 1., 2.]), tensor([1., 1., 1.]), tensor(3.))

In [ ]:
```
torch.sum(x * y)
```

Out[ ]: tensor(3.)

In [ ]:
```
A.shape, x.shape, torch.mv(A, x), A@x
```

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

In [ ]:
```
B = torch.ones(3, 4)
torch.mm(A, B), A@B
```

Out[ ]:
```
(tensor([[ 3.,  3.,  3.,  3.],
         [12., 12., 12., 12.]]),
 tensor([[ 3.,  3.,  3.,  3.],
         [12., 12., 12., 12.]]))
```

In [ ]:
```
u = torch.tensor([3.0, -4.0])
torch.norm(u)
```

Out[ ]: tensor(5.)

In [ ]:
```
torch.abs(u).sum()
```

Out[ ]: tensor(7.)

In [ ]:
```
torch.norm(torch.ones((4, 9)))
```

Out[ ]: tensor(6.)

2.3 discussion 새로운 내용은 아니었지만 선형대수학 부분이 딥러닝에 매우 중요하다는 걸 알기에 쭉 읽어보면서 복습하기 좋았다. 특히, 딥러닝에서 선형대수학이 모델의 수학적 기반을 제공한다는 점을 다시 한 번 느꼈다. 벡터, 행렬 등의 기본 개념들이 신경망의 구조와 학습 과정에 어떻게 적용되는지 다시 되새길 수 있어서 유익했다. 그리고 특히 tensor에 대해서 더 알아보는 것이 좋을 것 같다. Tensor는 벡터나 배열과 비슷하지만, 더 높은 차원을 다룰 수 있는 일반화된 형태라는 점에서 딥러닝에서 중요한 역할을 한다. 기본적으로 다차원 데이터를 효율적으로 표현할 수 있고, 다양한 연산을 병렬적으로 처리할 수 있다는 것이 큰 이점인 것 같다. 이를 통해 딥러닝에서 복잡한 연산을 처리하는 데 있어 보다 유연하고 강력한 도구가 된다는 사실을 다시 한 번 상기하게 되었다. 앞으로 tensor가 벡터나 배열에 비해 딥러닝에서 어떤 구체적인 이점을 제공하는지 더 깊이 찾아보면, 모델 설계나 최적화 과정에서도 더욱 효율적으로 접근할 수 있을 것 같다.

In [ ]:
```
import torch
```

In [ ]:
```
x = torch.arange(4.0)
x
```

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

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

In [ ]:
```
y = 2 * torch.dot(x, x)
y
```

Out[ ]:
```
tensor(28., grad_fn=<MulBackward0>)
```

In [ ]:
```
y.backward()
x.grad
```

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

In [ ]:
```
x.grad == 4 * x
```

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

In [ ]:
```
x.grad.zero_()  # Reset the gradient
y = x.sum()
y.backward()
x.grad
```

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

In [ ]:
```
x.grad.zero_()
y = x * x
y.backward(gradient=torch.ones(len(y)))  # Faster: y.sum().backward()
x.grad
```

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

In [ ]:
```
x.grad.zero_()
y = x * x
u = y.detach()
z = u * x

z.sum().backward()
x.grad == u
```

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

In [ ]:
```
x.grad.zero_()
y.sum().backward()
x.grad == 2 * x
```

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

In [ ]:
```
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
```

In [ ]:
```
a = torch.randn(size=(), requires_grad=True)
d = f(a)
d.backward()
```

In [ ]:
```
a.grad == d / a
```

Out[ ]:
```
tensor(True)
```

2.5 discussion Automatic Differentiation의 방법이 딥러닝에서 기울기를 확실히 효율적으로 계산하게 해준 것 같다. 특히, 딥러닝에서 기울기 계산은 손실 함수에 따라 모델의 파라미터를 업데이트하는 데 핵심적인 역할을 하므로, 이 부분이 얼마나 중요한지 다시금 느꼈다. 기울기를 정확하게 계산할 수 있어야 역전파(backpropagation)를 통해 모델을 학습시키고 최적의 파라미터를 찾아갈 수 있기 때문에, Automatic Differentiation은 딥러닝의 근간을 이루는 기술이라고 할 수 있다.무엇보다도, 수작업으로는 복잡한 네트워크에서 기울기를 계산하는 것이 거의 불가능에 가깝기 때문에, 이러한 자동 미분 기법 덕분에 복잡한 모델에서도 효율적이고 정확한 계산이 가능해졌다. 이 과정이 자동화되어 있기 때문에 연구자나 개발자는 기울기 계산의 세부적인 수학적 과정에 얽매이지 않고, 더 복잡한 구조나 알고리즘을 실

험할 수 있는 여유를 가질 수 있다는 점도 큰 장점이라고 생각된다.

In [ ]:
```
%matplotlib inline
import math
import time
import numpy as np
import torch
!pip install d2l==1.0.3
from d2l import torch as d2l
```

```
Collecting d2l==1.0.3
  Downloading d2l-1.0.3-py3-none-any.whl.metadata (556 bytes)
Collecting jupyter==1.0.0 (from d2l==1.0.3)
  Downloading jupyter-1.0.0-py2.py3-none-any.whl.metadata (995 bytes)
Collecting numpy==1.23.5 (from d2l==1.0.3)
  Downloading numpy-1.23.5-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (2.3 kB)
Collecting matplotlib==3.7.2 (from d2l==1.0.3)
  Downloading matplotlib-3.7.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (5.6 kB)
Collecting matplotlib-inline==0.1.6 (from d2l==1.0.3)
  Downloading matplotlib_inline-0.1.6-py3-none-any.whl.metadata (2.8 kB)
Collecting requests==2.31.0 (from d2l==1.0.3)
  Downloading requests-2.31.0-py3-none-any.whl.metadata (4.6 kB)
Collecting pandas==2.0.3 (from d2l==1.0.3)
  Downloading pandas-2.0.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (18 kB)
Collecting scipy==1.10.1 (from d2l==1.0.3)
  Downloading scipy-1.10.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (58 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.9/58.9 kB 4.8 MB/s eta 0:00:00
Requirement already satisfied: notebook in /usr/local/lib/python3.10/dist-packages (from jupyter==1.0.0->d2l==1.
0.3) (6.5.5)
Collecting qtconsole (from jupyter==1.0.0->d2l==1.0.3)
  Downloading qtconsole-5.6.0-py3-none-any.whl.metadata (5.0 kB)
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) (9.4.0)
Collecting pyparsing<3.1,>=2.3.1 (from matplotlib==3.7.2->d2l==1.0.3)
  Downloading pyparsing-3.0.9-py3-none-any.whl.metadata (4.2 kB)
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->d2
l==1.0.3) (2024.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from request
s==2.31.0->d2l==1.0.3) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests==2.31.0->d
2l==1.0.3) (3.8)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests==2.3
1.0->d2l==1.0.3) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests==2.3
1.0->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->m
atplotlib==3.7.2->d2l==1.0.3) (1.16.0)
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupy
ter==1.0.0->d2l==1.0.3) (0.2.0)
Requirement already satisfied: ipython>=5.0.0 in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyte
r==1.0.0->d2l==1.0.3) (7.34.0)
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyte
r==1.0.0->d2l==1.0.3) (6.1.12)
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 ipywid
gets->jupyter==1.0.0->d2l==1.0.3) (3.6.9)
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from ipywid
gets->jupyter==1.0.0->d2l==1.0.3) (3.0.13)
```

```
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.10/dist-pa
ckages (from jupyter-console->jupyter==1.0.0->d2l==1.0.3) (3.0.47)
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (from jupyter-console->jupyte
r==1.0.0->d2l==1.0.3) (2.16.1)
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->jupyte
r==1.0.0->d2l==1.0.3) (4.12.3)
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->ju
pyter==1.0.0->d2l==1.0.3) (0.4)
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->jup
yter==1.0.0->d2l==1.0.3) (5.7.2)
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.10/dist-packages (from nbconvert->j
upyter==1.0.0->d2l==1.0.3) (0.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyt
er==1.0.0->d2l==1.0.3) (2.1.5)
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jup
yter==1.0.0->d2l==1.0.3) (0.8.4)
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyt
er==1.0.0->d2l==1.0.3) (0.10.0)
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) (1.5.1)
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->jupy
ter==1.0.0->d2l==1.0.3) (1.6.0)
Requirement already satisfied: Send2Trash>=1.8.0 in /usr/local/lib/python3.10/dist-packages (from notebook->jupy
ter==1.0.0->d2l==1.0.3) (1.8.3)
Requirement already satisfied: terminado>=0.8.3 in /usr/local/lib/python3.10/dist-packages (from notebook->jupyt
er==1.0.0->d2l==1.0.3) (0.18.1)
Requirement already satisfied: prometheus-client in /usr/local/lib/python3.10/dist-packages (from notebook->jupy
ter==1.0.0->d2l==1.0.3) (0.20.0)
Requirement already satisfied: nbclassic>=0.4.7 in /usr/local/lib/python3.10/dist-packages (from notebook->jupyt
er==1.0.0->d2l==1.0.3) (1.1.0)
Collecting qtpy>=2.4.0 (from qtconsole->jupyter==1.0.0->d2l==1.0.3)
  Downloading QtPy-2.4.1-py3-none-any.whl.metadata (12 kB)
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0-
>ipykernel->jupyter==1.0.0->d2l==1.0.3) (71.0.4)
Collecting jedi>=0.16 (from ipython>=5.0.0->ipykernel->jupyter==1.0.0->d2l==1.0.3)
  Using cached jedi-0.19.1-py2.py3-none-any.whl.metadata (22 kB)
Requirement already satisfied: decorator in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipyker
nel->jupyter==1.0.0->d2l==1.0.3) (4.4.2)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipyk
ernel->jupyter==1.0.0->d2l==1.0.3) (0.7.5)
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipykern
el->jupyter==1.0.0->d2l==1.0.3) (0.2.0)
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipyk
ernel->jupyter==1.0.0->d2l==1.0.3) (4.9.0)
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.10/dist-packages (from jupyter-core>=
4.7->nbconvert->jupyter==1.0.0->d2l==1.0.3) (4.3.2)
Requirement already satisfied: notebook-shim>=0.2.3 in /usr/local/lib/python3.10/dist-packages (from nbclassic>=
0.4.7->notebook->jupyter==1.0.0->d2l==1.0.3) (0.2.4)
Requirement already satisfied: fastjsonschema>=2.15 in /usr/local/lib/python3.10/dist-packages (from nbformat>=5
.1->nbconvert->jupyter==1.0.0->d2l==1.0.3) (2.20.0)
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.10/dist-packages (from nbformat>=5.1->n
bconvert->jupyter==1.0.0->d2l==1.0.3) (4.23.0)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.10/dist-packages (from prompt-toolkit!=3.0.0,!=
3.0.1,<3.1.0,>=2.0.0->jupyter-console->jupyter==1.0.0->d2l==1.0.3) (0.2.13)
Requirement already satisfied: ptyprocess in /usr/local/lib/python3.10/dist-packages (from terminado>=0.8.3->not
ebook->jupyter==1.0.0->d2l==1.0.3) (0.7.0)
Requirement already satisfied: argon2-cffi-bindings in /usr/local/lib/python3.10/dist-packages (from argon2-cffi
->notebook->jupyter==1.0.0->d2l==1.0.3) (21.2.0)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->nb
convert->jupyter==1.0.0->d2l==1.0.3) (2.6)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->nbconvert->
jupyter==1.0.0->d2l==1.0.3) (0.5.1)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in /usr/local/lib/python3.10/dist-packages (from jedi>=0.16->
ipython>=5.0.0->ipykernel->jupyter==1.0.0->d2l==1.0.3) (0.8.4)
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->n
bformat>=5.1->nbconvert->jupyter==1.0.0->d2l==1.0.3) (24.2.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.10/dist-packages (
```

```
from jsonschema>=2.6->nbformat>=5.1->nbconvert->jupyter==1.0.0->d2l==1.0.3) (2023.12.1)
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=
2.6->nbformat>=5.1->nbconvert->jupyter==1.0.0->d2l==1.0.3) (0.35.1)
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->
nbformat>=5.1->nbconvert->jupyter==1.0.0->d2l==1.0.3) (0.20.0)
Requirement already satisfied: jupyter-server<3,>=1.8 in /usr/local/lib/python3.10/dist-packages (from notebook-
shim>=0.2.3->nbclassic>=0.4.7->notebook->jupyter==1.0.0->d2l==1.0.3) (1.24.0)
Requirement already satisfied: cffi>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from argon2-cffi-bindings
->argon2-cffi->notebook->jupyter==1.0.0->d2l==1.0.3) (1.17.1)
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-packages (from cffi>=1.0.1->argon2-cf
fi-bindings->argon2-cffi->notebook->jupyter==1.0.0->d2l==1.0.3) (2.22)
Requirement already satisfied: anyio<4,>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from jupyter-server<3
,>=1.8->notebook-shim>=0.2.3->nbclassic>=0.4.7->notebook->jupyter==1.0.0->d2l==1.0.3) (3.7.1)
Requirement already satisfied: websocket-client in /usr/local/lib/python3.10/dist-packages (from jupyter-server<
3,>=1.8->notebook-shim>=0.2.3->nbclassic>=0.4.7->notebook->jupyter==1.0.0->d2l==1.0.3) (1.8.0)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.10/dist-packages (from anyio<4,>=3.1.0->ju
pyter-server<3,>=1.8->notebook-shim>=0.2.3->nbclassic>=0.4.7->notebook->jupyter==1.0.0->d2l==1.0.3) (1.3.1)
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages (from anyio<4,>=3.1.0->
jupyter-server<3,>=1.8->notebook-shim>=0.2.3->nbclassic>=0.4.7->notebook->jupyter==1.0.0->d2l==1.0.3) (1.2.2)
Downloading d2l-1.0.3-py3-none-any.whl (111 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 111.7/111.7 kB 9.7 MB/s eta 0:00:00
Downloading jupyter-1.0.0-py2.py3-none-any.whl (2.7 kB)
Downloading matplotlib-3.7.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (11.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.6/11.6 MB 101.1 MB/s eta 0:00:00
Downloading matplotlib_inline-0.1.6-py3-none-any.whl (9.4 kB)
Downloading numpy-1.23.5-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (17.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 17.1/17.1 MB 80.9 MB/s eta 0:00:00
Downloading pandas-2.0.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (12.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.3/12.3 MB 103.8 MB/s eta 0:00:00
Downloading requests-2.31.0-py3-none-any.whl (62 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.6/62.6 kB 5.6 MB/s eta 0:00:00
Downloading scipy-1.10.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (34.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 34.4/34.4 MB 15.1 MB/s eta 0:00:00
Downloading pyparsing-3.0.9-py3-none-any.whl (98 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.3/98.3 kB 8.7 MB/s eta 0:00:00
Downloading qtconsole-5.6.0-py3-none-any.whl (124 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 124.7/124.7 kB 11.4 MB/s eta 0:00:00
Downloading QtPy-2.4.1-py3-none-any.whl (93 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 93.5/93.5 kB 8.3 MB/s eta 0:00:00
Using cached jedi-0.19.1-py2.py3-none-any.whl (1.6 MB)
Installing collected packages: requests, qtpy, pyparsing, numpy, matplotlib-inline, jedi, scipy, pandas, matplot
lib, qtconsole, jupyter, d2l
  Attempting uninstall: requests
    Found existing installation: requests 2.32.3
    Uninstalling requests-2.32.3:
      Successfully uninstalled requests-2.32.3
  Attempting uninstall: pyparsing
    Found existing installation: pyparsing 3.1.4
    Uninstalling pyparsing-3.1.4:
      Successfully uninstalled pyparsing-3.1.4
  Attempting uninstall: numpy
    Found existing installation: numpy 1.26.4
    Uninstalling numpy-1.26.4:
      Successfully uninstalled numpy-1.26.4
  Attempting uninstall: matplotlib-inline
    Found existing installation: matplotlib-inline 0.1.7
    Uninstalling matplotlib-inline-0.1.7:
      Successfully uninstalled matplotlib-inline-0.1.7
  Attempting uninstall: scipy
    Found existing installation: scipy 1.13.1
    Uninstalling scipy-1.13.1:
      Successfully uninstalled scipy-1.13.1
  Attempting uninstall: pandas
    Found existing installation: pandas 2.1.4
    Uninstalling pandas-2.1.4:
      Successfully uninstalled pandas-2.1.4
  Attempting uninstall: matplotlib
    Found existing installation: matplotlib 3.7.1
    Uninstalling matplotlib-3.7.1:
      Successfully uninstalled matplotlib-3.7.1
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This
behaviour is the source of the following dependency conflicts.
albucore 0.0.14 requires numpy>=1.24, but you have numpy 1.23.5 which is incompatible.
albumentations 1.4.14 requires numpy>=1.24.4, but you have numpy 1.23.5 which is incompatible.
bigframes 1.17.0 requires numpy>=1.24.0, but you have numpy 1.23.5 which is incompatible.
chex 0.1.86 requires numpy>=1.24.1, but you have numpy 1.23.5 which is incompatible.
google-colab 1.0.0 requires pandas==2.1.4, but you have pandas 2.0.3 which is incompatible.
google-colab 1.0.0 requires requests==2.32.3, but you have requests 2.31.0 which is incompatible.
pandas-stubs 2.1.4.231227 requires numpy>=1.26.0; python_version < "3.13", but you have numpy 1.23.5 which is in
compatible.
Successfully installed d2l-1.0.3 jedi-0.19.1 jupyter-1.0.0 matplotlib-3.7.2 matplotlib-inline-0.1.6 numpy-1.23.5
pandas-2.0.3 pyparsing-3.0.9 qtconsole-5.6.0 qtpy-2.4.1 requests-2.31.0 scipy-1.10.1
```

In [ ]:
```
n = 10000
a = torch.ones(n)
b = torch.ones(n)
```

In [ ]:
```
c = torch.zeros(n)
t = time.time()
for i in range(n):
    c[i] = a[i] + b[i]
f'{time.time() - t:.5f} sec'
```

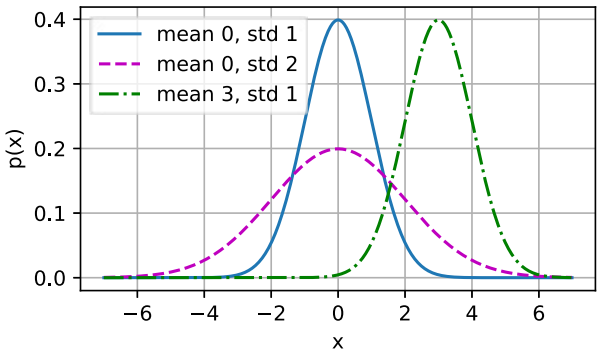
Out[ ]:
```
'0.11973 sec'
```

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

Out[ ]:
```
'0.00057 sec'
```

In [ ]:
```
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)
```

In [ ]:
```
# 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 discussion Linear Regression에서 batch 개념이 나름 흥미로웠다. 전체 데이터셋을 사용해 연산하는 것보다 훨씬 효율적이면서도, 단일 샘플을 기반으로 학습하는 확률적 경사 하강법(SGD)보다 안정적이라는 사실이 꽤나 놀라웠다. 이 방식은 적절한 크기의 데이터를 한번에 학습하여 연산 효율을 높이는 동시에, SGD처럼 학습이 불안정해지는 현상을 완화하는 데 도움을 주는 것 같다. 특히, mini-batch 방식을 사용하면 대용량 데이터에서 학습 속도를 크게 향상시킬 수 있다는 점도 주목할 만한 것 같다. 전체 데이터를 사용하면 시간이 오래 걸리고, 한 번에 하나의 샘플을 사용하는 경우는 기울기의 변동폭이 커서 학습이 느리게 이루어질 수 있다. 반면, batch를 활용하면 일정한 데이터 양으로 학습할 수 있어 속도와 안정성 사이의 균형을 맞출 수 있다는 점에서 매우 유용한 개념이라는 생각이 들었다.

In [ ]:
```
import time
import numpy as np
import torch
from torch import nn
from d2l import torch as d2l
```

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

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

a = A()
```

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

a.do()
```

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

In [ ]:
```
class HyperParameters:
```

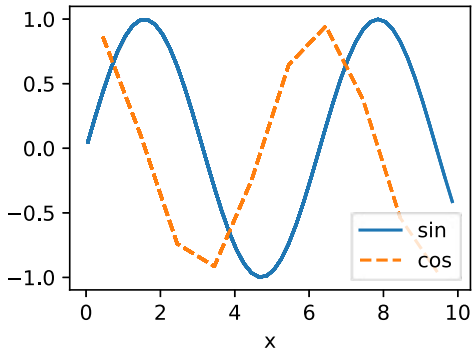
```
    """The base class of hyperparameters."""
    def save_hyperparameters(self, ignore=[]):
        raise NotImplemented
```

In [ ]:
```
# 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
```

In [ ]:
```
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
```

In [ ]:
```
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)
```



In [ ]:
```
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
```

In [ ]:
```
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)
```

In [ ]:
```
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 discussion 평소 사용하던 메서드들이 어떤 class에서 어떻게 정의되어 있었는지 구체적으로 알 수 있었다. 기존에는 주어진 라이브러리나 API의 메서드를 단순히 호출해 사용했지만, 이번 기회를 통해 각 메서드가 어떻게 class 내에서 구조화되고, 어떤 속성(attribute)들과 메서드(method)를 통해 상호작용하는지 직접 확인할 수 있었다.

In [ ]:
```
%matplotlib inline
import torch
from d2l import torch as d2l
```

In [ ]:
```
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)
```

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

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

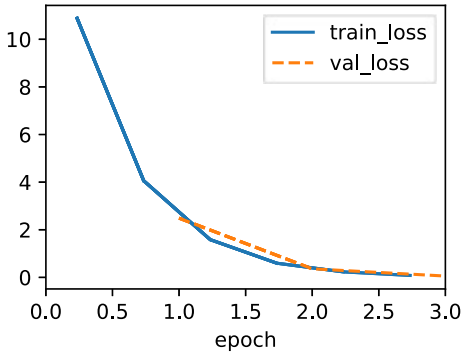
In [ ]:
```
class SGD(d2l.HyperParameters):
    """Minibatch stochastic gradient descent."""
    def __init__(self, params, lr):
```

```
        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_()
```

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

In [ ]:
```
@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
```

In [ ]:
```
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)
```



In [ ]:
```
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([ 0.0649, -0.2214])
error in estimating b: tensor([0.2359])
```

3.4 discussion Linear Regression을 기초부터 직접 구현하는 방법을 알 수 있었다. 특히, 경사하강법을 사용해 파라미터를 업데이트하는 과정을 직접 구현해 보면서 그 메커니즘을 더욱 확실하게 이해할 수 있었다. 이론적으로만 알던 경사하강법의 원리가, 실제로 코드로 구현해 보면서 어떻게 데이터에 맞춰 파라미터를 조정하고 손실 함수를 최소화하는지 구체적으로 확인할 수 있어서 유익했다.파라미터를 반복적으로 업데이트하면서, 점차적으로 모델이 데이터에 더 잘 맞아가는 과정을 볼 수 있었고, 이로 인해 경사하강법이 얼마나 강력한 최적화 도구인지 다시 한번 깨닫게 되었다. 또한, 학습률과 같은 하이퍼파라미터가 결과에 어떤 영향을 미치는지 실험을 통해 체감할 수 있었고, 잘못된 학습률이 모델의 성능에 미칠 수 있는 부정적인 영향도 직접 경험할 수 있었다.

In [ ]:
```
%matplotlib inline
import time
import torch
import torchvision
from torchvision import transforms
from d2l import torch as d2l

d2l.use_svg_display()
```

In [ ]:
```
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)
```

In [ ]:
```
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/Fas
hionMNIST/raw/train-images-idx3-ubyte.gz
100%|██████████| 26421880/26421880 [00:01<00:00, 20055122.29it/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/Fas
hionMNIST/raw/train-labels-idx1-ubyte.gz
100%|██████████| 29515/29515 [00:00<00:00, 354074.98it/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/Fash
ionMNIST/raw/t10k-images-idx3-ubyte.gz
100%|██████████| 4422102/4422102 [00:00<00:00, 6208174.04it/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/Fash
ionMNIST/raw/t10k-labels-idx1-ubyte.gz
100%|██████████| 5148/5148 [00:00<00:00, 21168899.01it/s]
Extracting ../data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to ../data/FashionMNIST/raw
```

Out[ ]:
```
(60000, 10000)
```

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

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

In [ ]:
```
@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]
```

In [ ]:
```
@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)
```

In [ ]:
```
X, y = next(iter(data.train_dataloader()))
print(X.shape, X.dtype, y.shape, y.dtype)
```

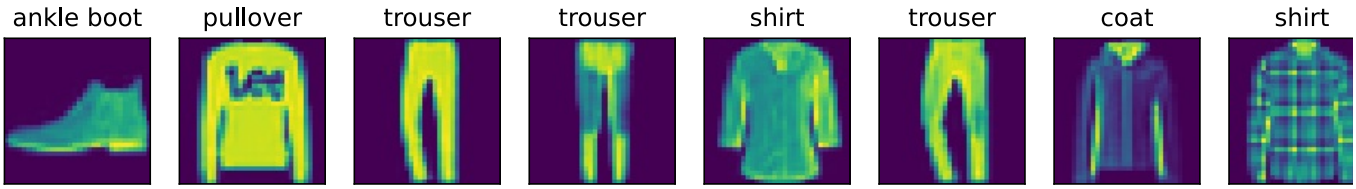
```
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will cr
eate 4 worker processes in total. Our suggested max number of worker in current system is 2, which is smaller th
an what this DataLoader is going to create. Please be aware that excessive worker creation might get DataLoader
running slow or even freeze, lower the worker number to avoid potential slowness/freeze if necessary.
  warnings.warn(_create_warning_msg(
torch.Size([64, 1, 32, 32]) torch.float32 torch.Size([64]) torch.int64
```

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

Out[ ]:
```
'17.86 sec'
```

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

In [ ]:
```
@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)
```



4.1,4.2 discussion 마지막 부분에 나온 compression 부분이 흥미로웠다. 데이터를 미니배치처럼 나누어 학습하는 방법은 알고 있었지만, 계산 효율성을 높이기 위해 압축까지 진행한다는 사실은 신기했다. 단순히 데이터를 나누는 것만으로는 한계가 있을 수 있는데, 데이터 자체를 압축해 더 적은 자원으로도 충분한 학습 성능을 얻을 수 있다는 점이 매우 인상 깊었다. 특히, 패션 이미지 데이터 셋(Fashion-MNIST) 같은 경우에도 성능 최적화와 효율성 극대화를 목표로 한 다양한 기법들이 적용된다는 점을 보며, 결국 딥러닝에서 가장 중요한 포커스는 최대한 적은 계산 자원으로 효율적인 학습을 목표로 한다는 것을 다시 한 번 느꼈다. 이러한 압축 기법을 통해 학습 시간도 줄이고, 연산 자원도 아낄 수 있다는 점에서 앞으로 더욱 주목받는 기술이 될 것 같다. 결론적으로, Softmax Regression에서의 압축과 같은 방법론들은 단순히 성능을 높이는 것뿐만 아니라, 실제 구현 시에 자원의 효율적 사용이라는 측면에서도 중요하게 작용한다는 것을 알 수 있었다.

In [ ]:
```
import torch
from d2l import torch as d2l
```

In [ ]:
```
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)
```

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

In [ ]:
```
@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 discussion 결국 분류에서의 핵심은 정확도이고, 이를 평가할 또 다른 함수가 필요하다면, 이 함수도 손실함수와 마찬가지로 파라미터 조정에 기여하는 것인가? 확실하진 않지만, 아마 그럴 것 같다. 보통 분류 문제에서 정확도는 모델 성능을 평가하는 주요 지표로 사용되지만, 그 자체가 직접적으로 파라미터를 업데이트하는 데 사용되지는 않는다. 대신, 정확도를 높이기 위한 손실함수가 존재하는데, 대표적으로 분류 문제에서는 교차 엔트로피 손실 함수(Cross-Entropy Loss)를 사용한다. 손실함수는 모델의 예측이 실제 레이블과 얼마나 다른지를 측정하고, 이 값을 최소화하는 방향으로 파라미터를 조정하는 역할을 한다. 즉, 손실함수가 파라미터 업데이트에 직접적인 영향을 미치는 것이다. 정확도는 평가 지표로, 모델이 얼마나 잘 예측하고 있는지를 파악하는 데 쓰이지만, 파라미터를 조정하는 목적은 손실함수를 최소화하는 것이다. 따라서, 정확도를 개선하려면 손실함수를 통해 간접적으로 모델을 최적화해야 한다. 만약 새로운 평가 함수가 도입된다면, 그 평가 함수도 손실함수와 연결되어 있을 가능성이 크며, 궁극적으로 파라미터 조정에 기여할 것이라고 생각된다.

In [ ]:
```
import torch
from d2l import torch as d2l
```

In [ ]:
```
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)
```

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

In [ ]:
```
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
```

In [ ]:
```
X = torch.rand((2, 5))
X_prob = softmax(X)
X_prob, X_prob.sum(1)
```

Out[ ]:
```
(tensor([[0.1207, 0.1808, 0.3096, 0.2003, 0.1886],
         [0.2554, 0.2165, 0.2148, 0.2134, 0.1000]]),
 tensor([1., 1.]))
```

In [ ]:
```
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]
```

In [ ]:
```
@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)
```

In [ ]:
```
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]
```

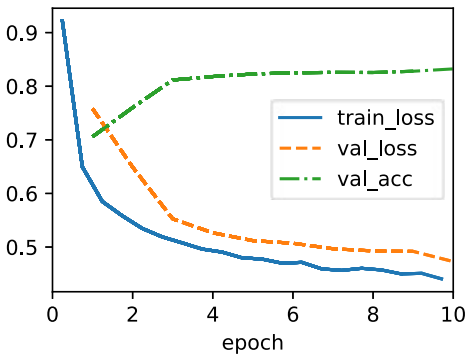
Out[ ]:
```
tensor([0.1000, 0.5000])
```

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

cross_entropy(y_hat, y)
```

Out[ ]:
```
tensor(1.4979)
```

In [ ]:
```
@d2l.add_to_class(SoftmaxRegressionScratch)
def loss(self, y_hat, y):
    return cross_entropy(y_hat, y)
```

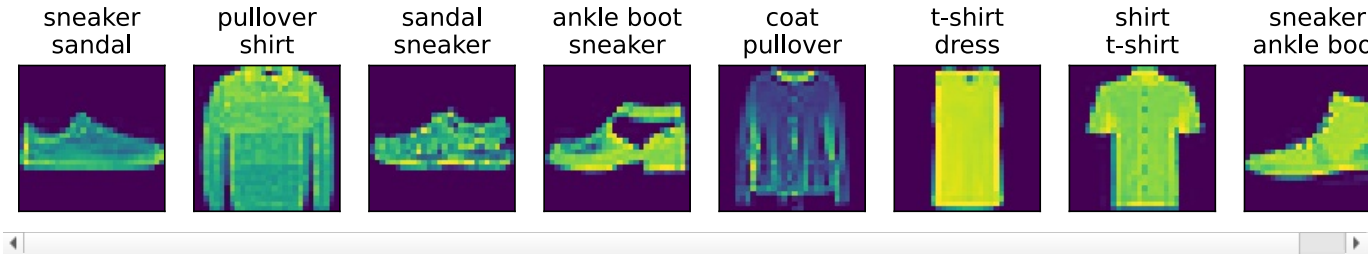
In [ ]:
```
data = d2l.FashionMNIST(batch_size=256)
model = SoftmaxRegressionScratch(num_inputs=784, num_outputs=10, lr=0.1)
trainer = d2l.Trainer(max_epochs=10)
trainer.fit(model, data)
```



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

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

In [ ]:
```
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)
```

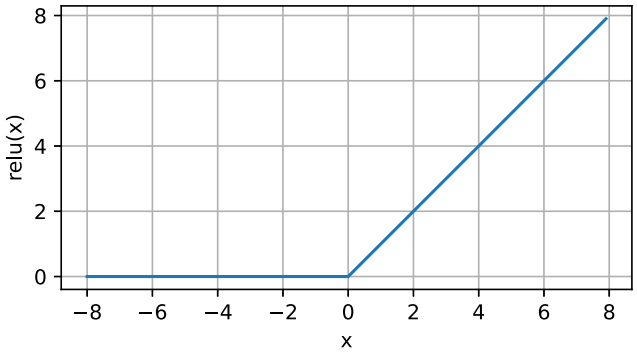


4.4 discussion 출력 값을 확률로 변환하여 모든 클래스의 합이 1이 되도록 보장하는 방법으로서 소프트맥스 함수를 사용하였다. 이 함수는

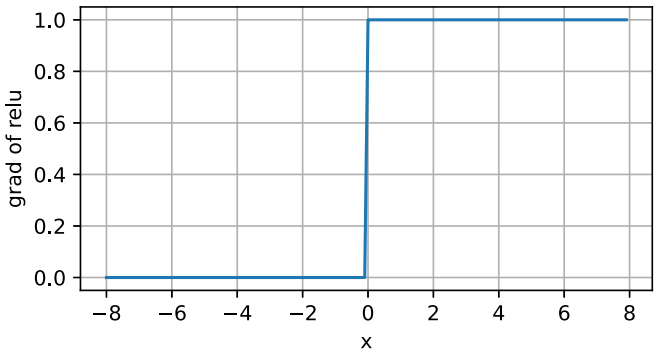
각 클래스에 대한 예측값을 확률 분포로 변환해주기 때문에, 분류 문제에서 확률적 방법을 적용하는 데 매우 중요한 역할을 한다고 생각한다.소프트맥스 함수의 가장 큰 장점은 각 클래스의 예측값을 비교적 직관적인 확률 값으로 바꿀 수 있다는 점이다. 이를 통해 모델이 특정 입력에 대해 어느 정도의 확신을 가지고 있는지 알 수 있고, 예측 결과에 대한 해석이 가능해진다. 또한, 모든 클래스의 확률 합이 1이 되도록 보장되므로, 확률 기반의 의사결정을 필요로 하는 분류 문제에서 매우 효과적인 도구인 것 같다.

In [ ]:
```
%matplotlib inline
import torch
from d2l import torch as d2l
```

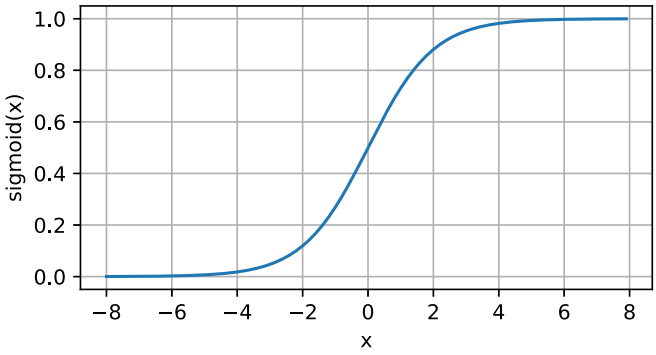
In [ ]:
```
x = torch.arange(-8.0, 8.0, 0.1, requires_grad=True)
y = torch.relu(x)
d2l.plot(x.detach(), y.detach(), 'x', 'relu(x)', figsize=(5, 2.5))
```



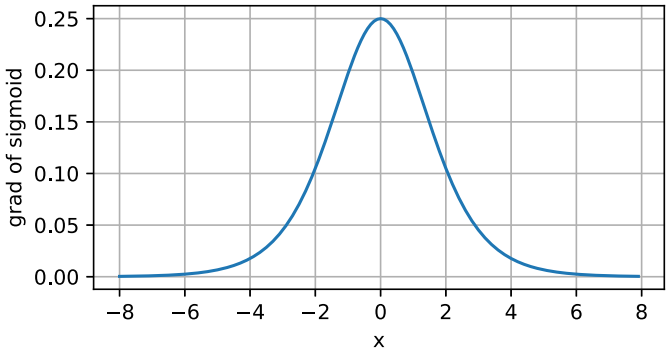
In [ ]:
```
y.backward(torch.ones_like(x), retain_graph=True)
d2l.plot(x.detach(), x.grad, 'x', 'grad of relu', figsize=(5, 2.5))
```



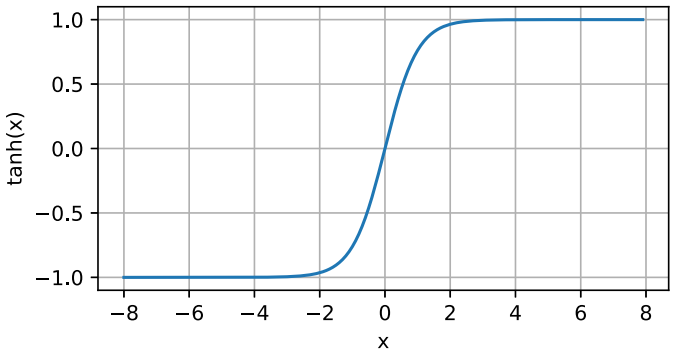
In [ ]:
```
y = torch.sigmoid(x)
d2l.plot(x.detach(), y.detach(), 'x', 'sigmoid(x)', figsize=(5, 2.5))
```



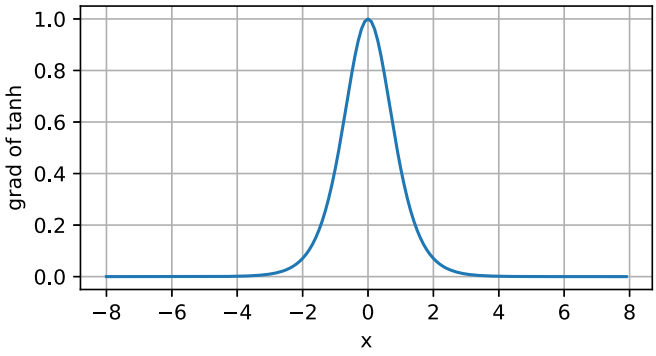
In [ ]:
```
# Clear out previous gradients
x.grad.data.zero_()
y.backward(torch.ones_like(x),retain_graph=True)
d2l.plot(x.detach(), x.grad, 'x', 'grad of sigmoid', figsize=(5, 2.5))
```



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



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



5.1 discussion Multilayer Perceptrons에서 선형성이 적합하지 않기 때문에 비선형성을 도입하여 사용하고 있다고 하였는데, 사실 실생활에서 관계들의 종류가 매우 다양하기 때문에 단순히 "선형이 아니다"라는 이유로 모두를 비선형이라는 하나의 카테고리로 묶는 것이 과연 맞는 방식인지 의문이 들었다. 현실 세계의 문제들은 매우 복잡하고 다양한 패턴과 관계가 존재하기 때문에, 비선형적 관계도 더 세분화되어야 하지 않을까 하는 생각이 든다. 실제로 비선형성은 단순히 선형이 아닌 모든 관계를 포함하는 매우 넓은 개념이다. 다양한 종류의 비선형적 관계, 예를 들어 다항식, 지수 함수, 로그 함수, 트리거 함수 등 여러 형태가 존재하며, 이들은 모두 서로 다른 패턴과 특징을 가지고 있다. 따라서, 비선형성이라는 큰 틀 안에서 이러한 다양한 관계들을 더 구체적으로 나누고 이해하는 것이 더 정확한 분석과 모델링에 기여할 수 있을 것 같다. 결론적으로, 비선형성을 도입하는 것은 확실히 선형적 한계를 극복하는 데 도움이 되지만, 단순히 선형이 아닌 모든 관계를 한데 묶는 것은 현실 세계의 복잡성을 반영하기에 충분하지 않을 수 있다. 비선형적 관계도 상황에 따라 세분화된 모델링이 필요하다고 생각된다.

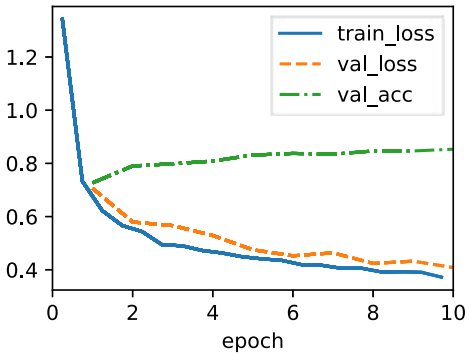
In [ ]:
```
import torch
from torch import nn
from d2l import torch as d2l
```

In [ ]:
```
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))
```

In [ ]:
```
def relu(X):
    a = torch.zeros_like(X)
    return torch.max(X, a)
```

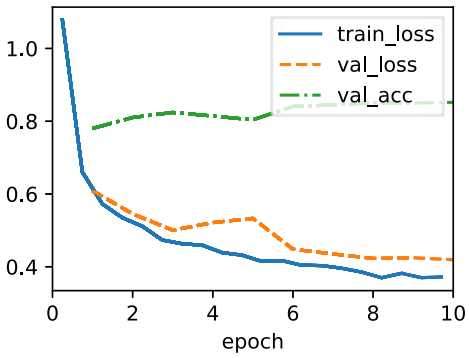
In [ ]:
```
@d2l.add_to_class(MLPScratch)
def forward(self, X):
    X = X.reshape((-1, self.num_inputs))
    H = relu(torch.matmul(X, self.W1) + self.b1)
    return torch.matmul(H, self.W2) + self.b2
```

In [ ]:
```
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)
```



In [ ]:
```
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))
```

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



5.2,5.3 discussion Multilayer Perceptron의 구현(implementation)과 Forward Propagation, Backward Propagation에 대해서 배웠다. 특히 딥러닝 모델 훈련 과정에서 중간 변수들과 기울기를 저장해야 하기 때문에 단순한 머신러닝 모델에 비해 훨씬 더 많은 메모리가 필요하다는 사실을 알게 되었다. 이 과정에서, 모델이 학습을 진행할 때 단순히 입력과 출력만 다루는 것이 아니라, 각 층에서 발생하는 중간 결과와 그에 따른 기울기까지 저장하고 활용해야 하기 때문에 메모리 효율성이 중요한 요소가 된다는 점이 흥미로웠다. 또한, forward propagation을 통해 각 층의 중간 값을 계산하고, backward propagation을 통해 이 값들을 활용하여 기울기를 계산한 후 파라미터를 업데이트하는 과정이 꽤나 복잡하다고 느꼈다. 단순한 개념으로 설명할 수 있지만, 실제로 이를 구현하고 효율적으로 관리하기 위해서는 계산 그래프를 통해 각 연산 단계를 명확히 정의하고 이를 따라 계산을 진행해야 한다는 점에서 매우 체계적인 접근이 필요하다는 생각이 들었다.딥러닝 모델의 학습 과정은 단순히 입력 데이터를 처리하는 것 이상으로, 중간 계산 과정의 모든 값을 기억하고 이를 기반으로 정확한 기울기를 구해야 하는 복잡한 연산의 연속이기 때문에, 메모리 관리와 효율적인 계산이 모델 성능에 큰 영향을 미친다는 것을 다시 한 번 깨달았다.