

✓ COSE474-2024F: Deep Learning HW1

✓ 0.1 Installation

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

```
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib==3.7.2->
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter==1.0.0->d2l==1.0.3)
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)
Requirement already satisfied: jupyter-client in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter==1.0.0->d2l==1.0.3)
Requirement already satisfied: tornado>=4.2 in /usr/local/lib/python3.10/dist-packages (from ipykernel->jupyter==1.0.0->d2l==1.0.3)
Requirement already satisfied: widgetsnbextension~=3.6.0 in /usr/local/lib/python3.10/dist-packages (from ipywidgets->jupyter==1.0.0->d2l==1.0.3)
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from ipywidgets->jupyter==1.0.0->d2l==1.0.3)
Requirement already satisfied: prompt-toolkit!=3.0.0,!<3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from jupyter-console->jupyter==1.0.0->d2l==1.0.3)
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (from jupyter-console->jupyter==1.0.0->d2l==1.0.3)
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) (4.12.3)
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->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)
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)
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)
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)
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)
Requirement already satisfied: nbformat>=5.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter==1.0.0->d2l==1.0.3)
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) (1.2.1)
Requirement already satisfied: pyzmq<25,>=17 in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter==1.0.0->d2l==1.0.3) (25.1.2)
Requirement already satisfied: argon2-cffi in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter==1.0.0->d2l==1.0.3) (25.1.2)
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)
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)
Requirement already satisfied: terminado>=0.8.3 in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter==1.0.0->d2l==1.0.3)
Requirement already satisfied: prometheus-client in /usr/local/lib/python3.10/dist-packages (from notebook->jupyter==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->jupyter==1.0.0->d2l==1.0.3)
Requirement already satisfied: qtpy>=2.4.0 in /usr/local/lib/python3.10/dist-packages (from qtconsole->jupyter==1.0.0->d2l==1.0.3)
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)
Requirement already satisfied: jedi>=0.16 in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipykernel->jupyter==1.0.0->d2l==1.0.3)
Requirement already satisfied: decorator in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipykernel->jupyter==1.0.0->d2l==1.0.3)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipykernel->jupyter==1.0.0->d2l==1.0.3)
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipykernel->jupyter==1.0.0->d2l==1.0.3)
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-packages (from ipython>=5.0.0->ipykernel->jupyter==1.0.0->d2l==1.0.3)
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)
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)
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)
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.10/dist-packages (from nbformat>=5.1->nbconvert->jupyter==1.0.0->d2l==1.0.3)
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)
Requirement already satisfied: ptyprocess in /usr/local/lib/python3.10/dist-packages (from terminado>=0.8.3->notebook->jupyter==1.0.0->d2l==1.0.3)
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)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->nbconvert->jupyter==1.0.0->d2l==1.0.3)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->nbconvert->jupyter==1.0.0->d2l==1.0.3)
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)
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat>=5.1->nbconvert->jupyter==1.0.0->d2l==1.0.3)
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)
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)
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)
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)
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)
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-packages (from cffi>=1.0.1->argon2-cffi-bindings->argon2-cffi->notebook->jupyter==1.0.0->d2l==1.0.3)
Requirement already satisfied: anyio<4,>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from jupyter-server<3,>=1.8->notebook->jupyter==1.0.0->d2l==1.0.3)
Requirement already satisfied: websocket-client in /usr/local/lib/python3.10/dist-packages (from jupyter-server<3,>=1.8->notebook->jupyter==1.0.0->d2l==1.0.3)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.10/dist-packages (from anyio<4,>=3.1.0->jupyter-server<3,>=1.8->notebook->jupyter==1.0.0->d2l==1.0.3)
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages (from anyio<4,>=3.1.0->jupyter-server<3,>=1.8->notebook->jupyter==1.0.0->d2l==1.0.3)
```

✓ 2.1 Data Manipulation

```
import torch
```

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

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

```
x.numel()
```

```
12
```

```
x.shape
```

```
↔ torch.Size([12])
```

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

```
X
```

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

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

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

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

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

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

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

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

```
↔ tensor([[ -0.8988,  1.3145, -1.2134,  1.8122],
          [ 0.6653,  0.9110, -0.5269, -1.3125],
          [ 0.0824, -1.2536,  0.3277,  1.1098]])
```

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

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

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

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

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

```
X
```

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

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

```
X
```

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

```
torch.exp(x)
```

```
↔ tensor([162754.7969, 162754.7969, 162754.7969, 162754.7969, 162754.7969,
          162754.7969, 162754.7969, 162754.7969, 2980.9580,  8103.0840,
          22026.4648,  59874.1406])
```

```
x = torch.tensor([1.0, 2, 4, 8])
```

```
y = torch.tensor([2, 2, 2, 2])
```

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

```
↔ (tensor([ 3.,  4.,  6., 10.]),
   tensor([-1.,  0.,  2.,  6.]),
   tensor([ 2.,  4.,  8., 16.]),
   tensor([0.5000, 1.0000, 2.0000, 4.0000]),
   tensor([ 1.,  4., 16., 64.]])
```

```
X = torch.arange(12, dtype=torch.float32).reshape((3,4))
Y = torch.tensor([[2.0, 1, 4, 3], [1, 2, 3, 4], [4, 3, 2, 1]])
torch.cat((X, Y), dim=0), torch.cat((X, Y), dim=1)
```

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

```
X == Y
```

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

```
X.sum()
```

```
↔ tensor(66.)
```

```
a = torch.arange(3).reshape((3, 1))
b = torch.arange(2).reshape((1, 2))
a, b
```

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

```
a + b
```

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

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

```
↔ False
```

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

```
↔ id(Z): 134959597609632
   id(Z): 134959597609632
```

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

```
↔ True
```

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

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

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

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

✓ 2.2 Data Preprocessing

```
import os
```

```
os.makedirs(os.path.join('.', 'data'), exist_ok=True)
```

```
data_file = os.path.join('.', 'data', 'house_tiny.csv')
with open(data_file, 'w') as f:
    f.write('''NumRooms,RoofType,Price
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

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

```
↕
```

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

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

```
↕
```

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

```
import torch
```

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

```
↕ (tensor([[3., 0., 1.],
          [2., 0., 1.],
          [4., 1., 0.],
          [3., 0., 1.]], dtype=torch.float64),
  tensor([127500., 106000., 178100., 140000.], dtype=torch.float64))
```

2.3 Linear Algebra

```
import torch
```

```
x = torch.tensor(3.0)
y = torch.tensor(2.0)
```

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

```
↕ (tensor(5.), tensor(6.), tensor(1.5000), tensor(9.))
```

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

```
↕ tensor([0, 1, 2])
```

```
x[2]
```

```
↕ tensor(2)
```

```
len(x)
```

```
↕ 3
```

```
x.shape
```

```
↔ torch.Size([3])
```

```
A = torch.arange(6).reshape(3, 2)
A
```

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

```
A.T
```

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

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

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

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

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

         [[12, 13, 14, 15],
           [16, 17, 18, 19],
           [20, 21, 22, 23]]])
```

```
A = torch.arange(6, dtype=torch.float32).reshape(2, 3)
B = A.clone()
A, A + B
```

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

```
A * B
```

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

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

```
↔ (tensor([[[ 2,  3,  4,  5],
           [ 6,  7,  8,  9],
           [10, 11, 12, 13]],

         [[14, 15, 16, 17],
           [18, 19, 20, 21],
           [22, 23, 24, 25]]]),
   torch.Size([2, 3, 4]))
```

```
x = torch.arange(3, dtype=torch.float32)
x, x.sum()
```

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

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

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

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

```
↔ (torch.Size([2, 3]), torch.Size([3]))
```


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

```
↔ (torch.Size([2, 3]), torch.Size([2]))
```


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

 `tensor(True)`


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

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


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

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


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

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


```
A / sum_A
```

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


```
A.cumsum(axis=0)
```

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


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

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


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

 `tensor(3.)`


```
A.shape, x.shape, torch.mv(A, x), A@x
```

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


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

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


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

 `tensor(5.)`

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

 `tensor(7.)`


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

 `tensor(6.)`

✓ 2.5 Automatic Differentiation

```
import torch
```

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

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

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

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

↔ tensor(28., grad_fn=<MulBackward0>)

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

↔ tensor([0., 4., 8., 12.])

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

↔ tensor([True, True, True, True])

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

↔ tensor([1., 1., 1., 1.])

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

↔ tensor([0., 2., 4., 6.])

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

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

↔ tensor([True, True, True, True])

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

↔ tensor([True, True, True, True])

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

```
a = torch.randn(size=(), requires_grad=True)
d = f(a)
d.backward()
```

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

↔ tensor(True)

✓ 3.1 Linear Regression

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

```
n = 10000
a = torch.ones(n)
```

```
b = torch.ones(n)
```

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

→ '0.40247 sec'

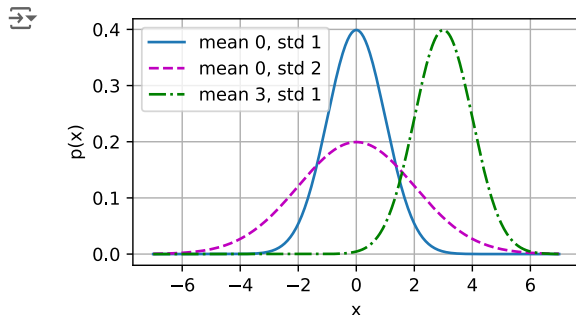
```
t = time.time()
d = a + b
f'{time.time() - t:.5f} sec'
```

→ '0.00026 sec'

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

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

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



✓ 3.2 Object-Oriented Design for Implementation

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

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

"@save" is not an allowed annotation - allowed values include [@param, @title, @markdown].

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

"@save" is not an allowed annotation - allowed values include [@param, @title, @markdown].


```
# 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'))

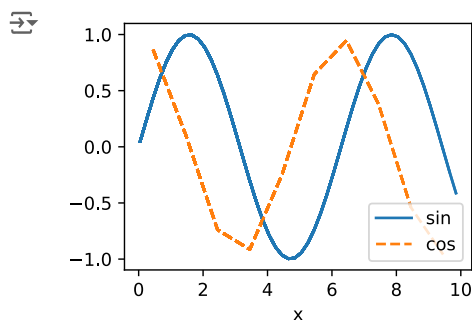
b = B(a=1, b=2, c=3)
```

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

```
class ProgressBoard(d2l.HyperParameters): #@save
    """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'],
                 fig=None, axes=None, figsize=(3.5, 2.5), display=True):
        self.save_hyperparameters()

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

```
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)
```



"@save" is not an allowed annotation - allowed values include [@param, @title, @markdown].

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

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

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

    def plot(self, key, value, train):
        """Plot a point in animation."""
        assert hasattr(self, 'trainer'), 'Trainer is not initied'
        self.board.xlabel = 'epoch'
        if train:
            x = self.trainer.train_batch_idx / \
                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])
```

"@save" is not an allowed annotation - allowed values include [@param, @title, @markdown].

```
self.plot('loss', 1, train=False)
```

```
def configure_optimizers(self):  
    raise NotImplementedError
```

```
class DataModule(d2l.HyperParameters): #@save  
    """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)
```

"@save" is not an allowed annotation - allowed values include [[@param](#), [@title](#), [@markdown](#)].

```
class Trainer(d2l.HyperParameters): #@save  
    """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
```

"@save" is not an allowed annotation - allowed values include [[@param](#), [@title](#), [@markdown](#)].

3.4 Linear Regression Implementation from Scratch

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

```
class LinearRegressionScratch(d2l.Module): #@save  
    """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)
```

"@save" is not an allowed annotation - allowed values include [[@param](#), [@title](#), [@markdown](#)].

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

"@save" is not an allowed annotation - allowed values include [[@param](#), [@title](#), [@markdown](#)].

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

"@save" is not an allowed annotation - allowed values include [[@param](#), [@title](#), [@markdown](#)].

```
class SGD(d2l.HyperParameters): #@save  
    """Minibatch stochastic gradient descent."""
```

"@save" is not an allowed annotation - allowed values include [[@param](#), [@title](#), [@markdown](#)].

```
def __init__(self, params, lr):
    self.save_hyperparameters()

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

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

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

"@save" is not an allowed annotation - allowed values include [@param, @title, @markdown].

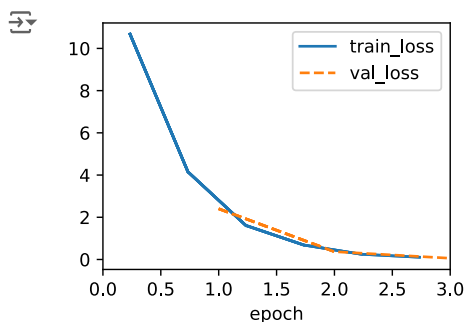
```
@d2l.add_to_class(d2l.Trainer) #@save
def prepare_batch(self, batch):
    return batch

@d2l.add_to_class(d2l.Trainer) #@save
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
```

"@save" is not an allowed annotation - allowed values include [@param, @title, @markdown].

"@save" is not an allowed annotation - allowed values include [@param, @title, @markdown].

```
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([ 0.1249, -0.2251])
error in estimating b: tensor([0.2425])
```

✓ 4.1 Softmax Regression

Instead of answering *how much?*, focus on *which category?* questions instead.

Two types of Classification Problems

- Hard assignments of categories (classes)
- Soft assignments or probability assessments (multi-label)

On-hot Encoding Example

y would be a three-dimensional vector, with $(1, 0, 0)$ corresponding to "cat", $(0, 1, 0)$ to "chicken", and $(0, 0, 1)$ to "dog":

$$y \in \{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}.$$

Softmax

To ensure nonnegativity in the outputs, use an exponential function

$$\hat{y} = \text{softmax}(\mathbf{o}) \quad \text{where} \quad \hat{y}_i = \frac{\exp(o_i)}{\sum_j \exp(o_j)}.$$

Loss function (Cross-entropy Loss)

$$l(y, \hat{y}) = - \sum_{j=1}^q y_j \log \hat{y}_j.$$

Understanding Softmax and Cross-Entropy

$$\begin{aligned} l(y, \hat{y}) &= - \sum_{j=1}^q y_j \log \frac{\exp(o_j)}{\sum_{k=1}^q \exp(o_k)} \\ &= \sum_{j=1}^q y_j \log \sum_{k=1}^q \exp(o_k) - \sum_{j=1}^q y_j o_j \\ &= \log \sum_{k=1}^q \exp(o_k) - \sum_{j=1}^q y_j o_j. \\ \partial_{o_j} l(y, \hat{y}) &= \frac{\exp(o_j)}{\sum_{k=1}^q \exp(o_k)} - y_j = \text{softmax}(\mathbf{o})_j - y_j. \end{aligned}$$

The derivative of cross-entropy loss is the difference between the softmax of our prediction and the actual ground truth, same as in linear regression.

✓ 4.2 The Image Classification Dataset


```
%matplotlib inline
import time
import torch
import torchvision
from torchvision import transforms
from d2l import torch as d2l

d2l.use_svg_display()
```


```
class FashionMNIST(d2l.DataModule): #@save
    """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=T
        )
        self.val = torchvision.datasets.FashionMNIST(
            root=self.root, train=False, transform=trans, download=
```

"@save" is not an allowed annotation - allowed values include [@param, @title, @markdown].

```
data = FashionMNIST(resize=(32, 32))
len(data.train), len(data.val)
```

 (60000, 10000)

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

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

```
@d2l.add_to_class(FashionMNIST) #@save
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]
```

"@save" is not an allowed annotation - allowed values include [@param, @title, @markdown].

```
@d2l.add_to_class(FashionMNIST) #@save
def get_dataloader(self, train):
    data = self.train if train else self.val
```

"@save" is not an allowed annotation - allowed values include [@param, @title, @markdown].

```
→ /usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes with 16 workers in total. The original dataloader has 16 workers. Overriding the number of workers found in the original dataloader. To avoid additional overhead you may want to explicitly set `num_workers` in the `DataLoader` that you wish to create.
  warnings.warn(_create_warning_msg(
torch.Size([64, 1, 32, 32]) torch.float32 torch.Size([64]) torch.int64
```

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

→ '12.78 sec'

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

"@save" is not an allowed annotation - allowed values include [@param, @title, @markdown].

```
@d2l.add_to_class(FashionMNIST) #@save
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)
```

"@save" is not an allowed annotation - allowed values include [@param, @title, @markdown].

ankle boot pullover trouser trouser shirt trouser coat shirt

4.3. The Base Classification Model

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

```
class Classifier(d2l.Module): #@save
    """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)
```

"@save" is not an allowed annotation - allowed values include [@param, @title, @markdown].

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

"@save" is not an allowed annotation - allowed values include [@param, @title, @markdown].

```
@d2l.add_to_class(Classifier) #@save
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
```

"@save" is not an allowed annotation - allowed values include [@param, @title, @markdown].

- 4.4. Softmax Regression Implementation from Scratch

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

```
X = torch.tensor([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]])
X.sum(0, keepdims=True), X.sum(1, keepdims=True)
```

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

```
def softmax(X):
    X_exp = torch.exp(X)
    partition = X_exp.sum(1, keepdims=True)
    return X_exp / partition # The broadcasting mechanism is applied here
```

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

```
(tensor([[0.1626, 0.2146, 0.1726, 0.1735, 0.2768],
        [0.2600, 0.1646, 0.1612, 0.2458, 0.1684]]),
 tensor([1., 1.]))
```

```
class SoftmaxRegressionScratch(d2l.Classifier):
    def __init__(self, num_inputs, num_outputs, lr, sigma=0.01):
        super().__init__()
        self.save_hyperparameters()
        self.W = torch.normal(0, sigma, size=(num_inputs, num_outputs),
                                   requires_grad=True)
        self.b = torch.zeros(num_outputs, requires_grad=True)

    def parameters(self):
        return [self.W, self.b]
```

```
@d2l.add_to_class(SoftmaxRegressionScratch)
def forward(self, X):
    X = X.reshape((-1, self.W.shape[0]))
    return softmax(torch.matmul(X, self.W) + self.b)
```

```
y = torch.tensor([0, 2])
y_hat = torch.tensor([[0.1, 0.3, 0.6], [0.3, 0.2, 0.5]])
y_hat[[0, 1], y]
```

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

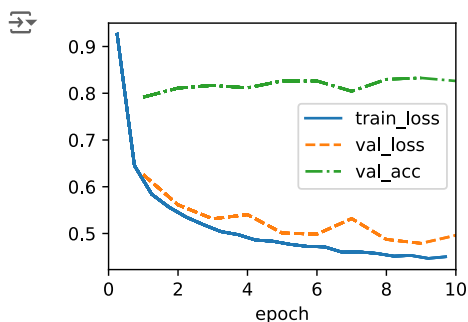
```
def cross_entropy(y_hat, y):
    return -torch.log(y_hat[list(range(len(y_hat))), y]).mean()
```

```
cross_entropy(y_hat, y)
```

```
tensor(1.4979)
```

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

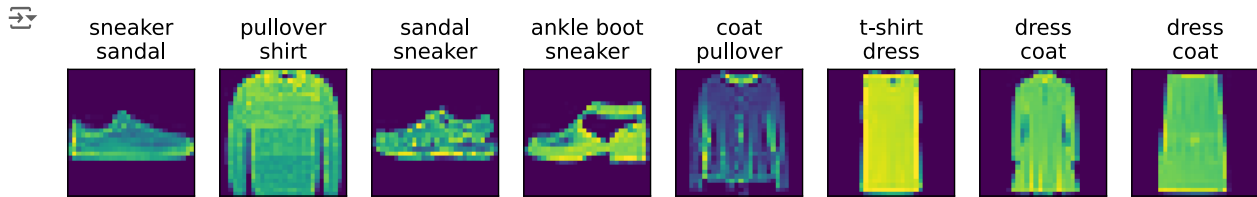
```
data = d2l.FashionMNIST(batch_size=256)
model = SoftmaxRegressionScratch(num_inputs=784, num_outputs=10, lr=0.1)
trainer = d2l.Trainer(max_epochs=10)
trainer.fit(model, data)
```



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

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

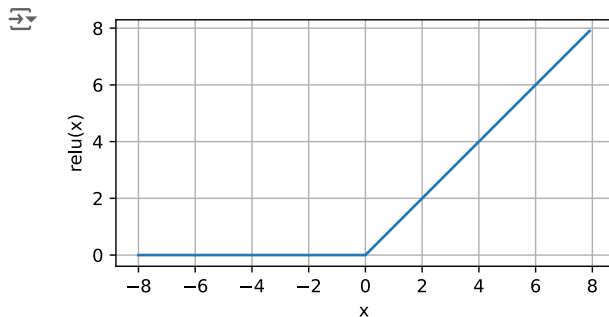
```
wrong = preds.type(y.dtype) != y
X, y, preds = X[wrong], y[wrong], preds[wrong]
labels = [a+'\n'+b for a, b in zip(
    data.text_labels(y), data.text_labels(preds))]
data.visualize([X, y], labels=labels)
```



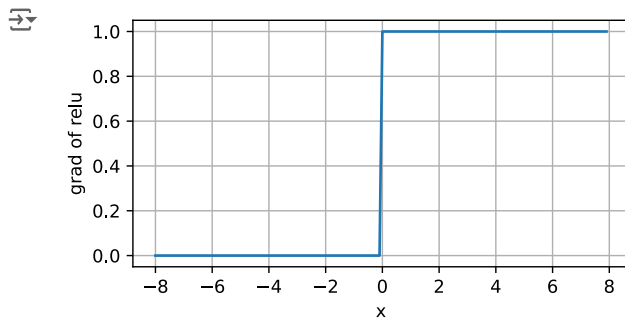
5.1 Multilayer Perceptrons

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

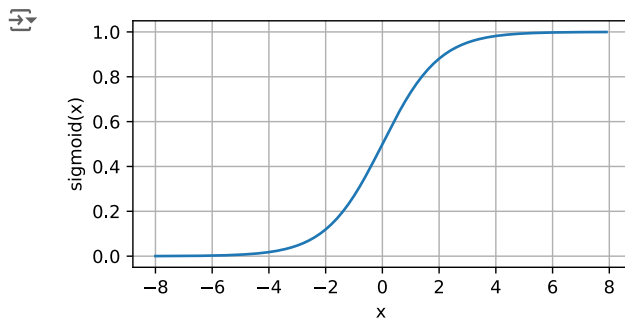
```
x = torch.arange(-8.0, 8.0, 0.1, requires_grad=True)
y = torch.relu(x)
d2l.plot(x.detach(), y.detach(), 'x', 'relu(x)', figsize=(5, 2.5))
```



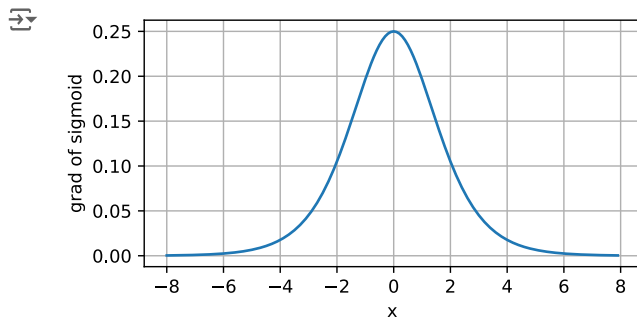
```
y.backward(torch.ones_like(x), retain_graph=True)
d2l.plot(x.detach(), x.grad, 'x', 'grad of relu', figsize=(5, 2.5))
```



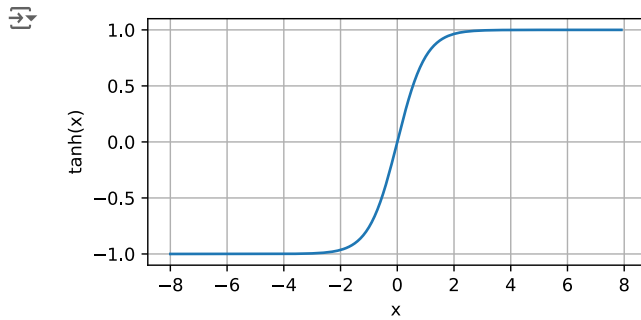
```
y = torch.sigmoid(x)
d2l.plot(x.detach(), y.detach(), 'x', 'sigmoid(x)', figsize=(5, 2.5))
```



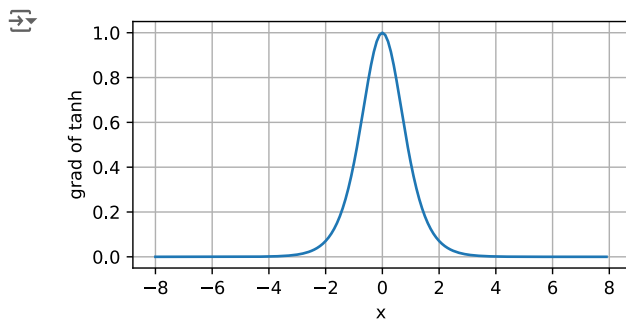
```
x.grad.data.zero_()
y.backward(torch.ones_like(x), retain_graph=True)
d2l.plot(x.detach(), x.grad, 'x', 'grad of sigmoid', figsize=(5, 2.5))
```



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



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



✓ 5.2 Implementation of Multilayer Perceptrons

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

```
class MLPScratch(d2l.Classifier):
    def __init__(self, num_inputs, num_outputs, num_hiddens, lr, sigma=0.01):
        super().__init__()
        self.save_hyperparameters()
        self.W1 = nn.Parameter(torch.randn(num_inputs, num_hiddens) * sigma)
        self.b1 = nn.Parameter(torch.zeros(num_hiddens))
        self.W2 = nn.Parameter(torch.randn(num_hiddens, num_outputs) * sigma)
        self.b2 = nn.Parameter(torch.zeros(num_outputs))
```

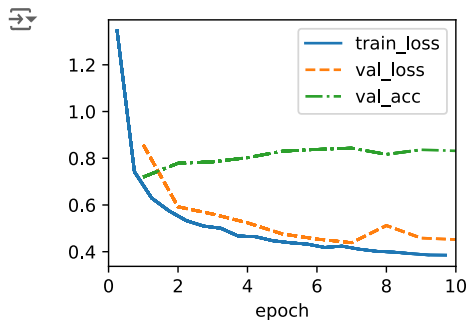
```
def relu(X):
    a = torch.zeros_like(X)
    return torch.max(X, a)
```

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

```
model = MLPScratch(num_inputs=784, num_outputs=10, num_hiddens=256, lr=0.1)
```

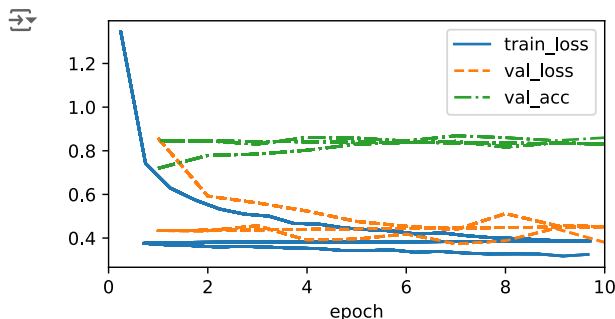


```
data = d2l.FashionMNIST(batch_size=256)
trainer = d2l.Trainer(max_epochs=10)
trainer.fit(model, data)
```



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

```
trainer.fit(model, data)
```



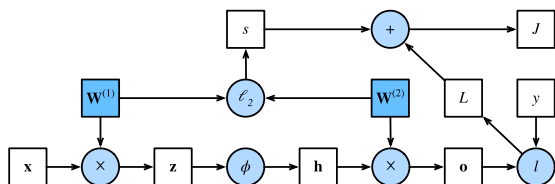
✓ 5.3 Forward Propagation, Backward Propagation, and Computational Graphs

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. We now work step-by-step through the mechanics of a neural network with one hidden layer. This may seem tedious but in the eternal words of funk virtuoso James Brown, you must "pay the cost to be the boss".

Computational Graph of Forward Propagation

Plotting computational graphs helps us visualize the dependencies of operators and variables within the calculation.



Backpropagation

Assume that we have functions $Y = f(X)$ and $Z = g(Y)$, in which the input and the output X, Y, Z are tensors of arbitrary shapes. By using the chain rule, we can compute the derivative of Z with respect to X via

$$\frac{\partial Z}{\partial X} = \text{prod} \left(\frac{\partial Z}{\partial Y}, \frac{\partial Y}{\partial X} \right).$$

Training Neural Networks

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

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

✓ Notes & Exercises

✓ 2.1 Notes

Very similar to Numpy array manipulation operations.

Broadcasting works according to the following two-step procedure: (i) expand one or both arrays by copying elements along axes with length 1 so that after this transformation, the two tensors have the same shape; (ii) perform an elementwise operation on the resulting arrays.

In machine learning, we often have hundreds of megabytes of parameters and update all of them multiple times per second. Whenever possible, we want to perform these updates in place.

Memory-efficient Assignment $\rightarrow X[:] = X + Y$ or $X += Y$

✓ 2.2 Notes

For imputating categorical data with NaN values, make a new column for the NaN values with boolean values instead. Numerical data with NaN values are usually imputed with the mean values.

✓ 2.3 Notes

- Scalars, vectors, matrices, and tensors are the basic mathematical objects used in linear algebra and have zero, one, two, and an arbitrary number of axes, respectively.
- Tensors can be sliced or reduced along specified axes via indexing, or operations such as `sum` and `mean`, respectively.
- Elementwise products are called Hadamard products. By contrast, dot products, matrix–vector products, and matrix–matrix products are not elementwise operations and in general return objects having shapes that are different from the the operands.
- Compared to Hadamard products, matrix–matrix products take considerably longer to compute (cubic rather than quadratic time).
- Norms capture various notions of the magnitude of a vector (or matrix), and are commonly applied to the difference of two vectors to measure their distance apart.
- Common vector norms include the ℓ_1 and ℓ_2 norms, and common matrix norms include the *spectral* and *Frobenius* norms.

✓ 2.5 Notes

1. Attach gradients to those variables with respect to which we desire derivatives.
2. Record the computation of the target value.
3. Execute the backpropagation function.
4. Access the resulting gradient.

✓ 3.1 Notes

Analytic Solution (Restrictive)

Solving for \mathbf{w} provides us with the optimal solution for the optimization problem. Note that this solution

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

will only be unique when the matrix $\mathbf{X}^\top \mathbf{X}$ is invertible, i.e., when the columns of the design matrix are linearly independent.

Minibatch Stochastic Gradient Descent

1. Initialize the values of the model parameters, typically at random.
2. Iteratively sample random minibatches from the data, updating the parameters in the direction of the negative gradient.

We can express the update as follows:

$$(\mathbf{w}, b) \leftarrow (\mathbf{w}, b) - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}_t} \partial_{(\mathbf{w}, b)} l^{(i)}(\mathbf{w}, b).$$

3.2 Exercises

```
# Call the fully implemented HyperParameters class saved in d2l
class B(d2l.HyperParameters):
    def __init__(self, a, b, c):
        #self.save_hyperparameters(ignore=['c']) <--- removed
        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)
```

```
-----
AttributeError                                Traceback (most recent call last)
<ipython-input-91-0c2234f973a4> in <cell line: 8>()
      6         print('There is no self.c =', not hasattr(self, 'c'))
      7
----> 8 b = B(a=1, b=2, c=3)

<ipython-input-91-0c2234f973a4> in __init__(self, a, b, c)
      3     def __init__(self, a, b, c):
      4         #self.save_hyperparameters(ignore=['c']) <--- removed
----> 5         print('self.a =', self.a, 'self.b =', self.b)
      6         print('There is no self.c =', not hasattr(self, 'c'))
      7

AttributeError: 'B' object has no attribute 'a'
```

When removing the `save_hyperparameters`. The attributes of the object was not associated with the object itself, therefore invoking `self` calls results in an error.

3.4 Notes

- **Model definition:** Defining the linear relationship between input features and output.
- **Loss function:** Measuring the difference between predicted and actual values (e.g. Mean Squared Error).
- **Optimization algorithm:** Updating model parameters to minimize the loss function (e.g. Stochastic Gradient Descent).
- **Training loop:** Iteratively processing data, computing loss, and updating parameters.

4.2 Exercises

```
@d2l.add_to_class(FashionMNIST) #@save
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)
```

"@save" is not an allowed annotation - allowed values include [`@param`, `@title`, `@markdown`].

```
X, y = next(iter(data.train_dataloader()))
print(X.shape, X.dtype, y.shape, y.dtype)
```

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

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

```
'124.14 sec'
```

The time it takes to read is much longer since it needs to process each sample individually which costs a lot of overhead for small amounts of data.

4.3 Notes

Although accuracy is not differentiable, it is usually the most important metric to quantify how good a classifier model is.

4.4 Exercises

Inputting 100 into the softmax function.

```
test = torch.tensor([[100.0, 2.0, 3.0], [4.0, 5.0, 6.0]])
result = softmax(test)
result, result.sum(1)
```

```
(tensor([[ nan, 0.0000, 0.0000],
          [0.0900, 0.2447, 0.6652]]),
 tensor([nan, 1.]))
```

Returned NaN values, due to exponentiation of large number causing integer overflow.

```
test = torch.tensor([[-100.0, 2.0, 3.0], [4.0, 5.0, 6.0]])
result = softmax(test)
result, result.sum(1)
```

```
(tensor([[1.4013e-45, 2.6894e-01, 7.3106e-01],
          [9.0031e-02, 2.4473e-01, 6.6524e-01]]),
 tensor([1.0000, 1.0000]))
```

Works fine since exponentiation negative numbers lead to small numbers

```
def softmax(X):
    X_exp = torch.exp(X - X.max(axis=1, keepdims=True).values)
    # Shifting all the values of the same row to be closer to the negatives
    partition = X_exp.sum(1, keepdims=True)
    return X_exp / partition
```

Preventing the softmax function from exponentiating large numbers

```
test = torch.tensor([[100.0, 2.0, 3.0], [4.0, 5.0, 6.0]])
result = softmax(test)
result, result.sum(1)
```

```
(tensor([[1.0000e+00, 2.7465e-43, 7.4689e-43],
          [9.0031e-02, 2.4473e-01, 6.6524e-01]]),
 tensor([1., 1.]))
```

Works fine as intended.

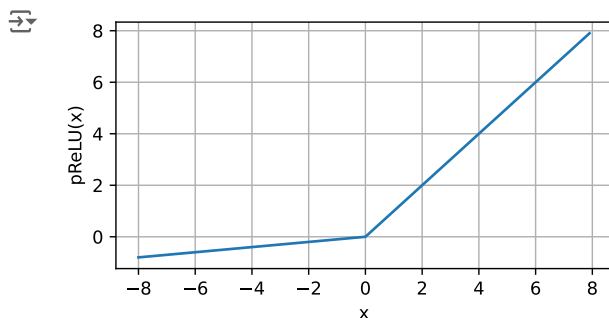
5.1 Exercises

pReLU

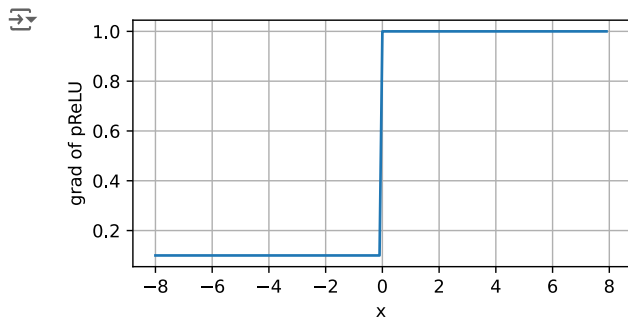
$$\text{pReLU}(x) = \max(0, x) + \alpha \min(0, x).$$

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

```
y = pReLU(x, 0.1)
d2l.plot(x.detach(), y.detach(), 'x', 'pReLU(x)', figsize=(5, 2.5))
```



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

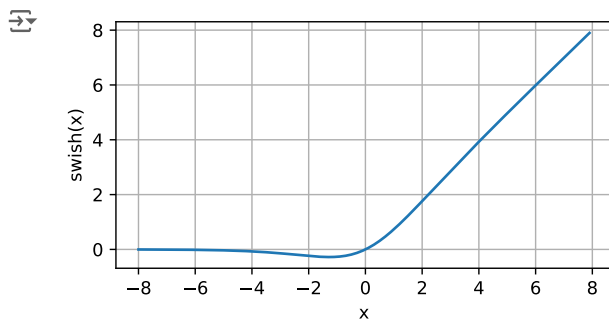


Swish

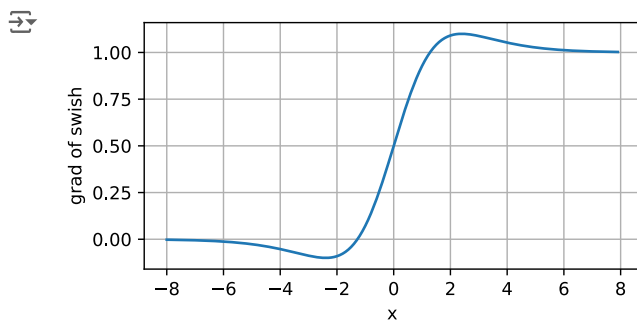
$$\text{swish}(x) = x \text{sigmoid}(\beta x) = \frac{x}{1 + e^{-\beta x}}$$

```
swish = lambda x, b: x / (1 + torch.exp(-b * x))
```

```
y = swish(x, 1)
d2l.plot(x.detach(), y.detach(), 'x', 'swish(x)', figsize=(5, 2.5))
```

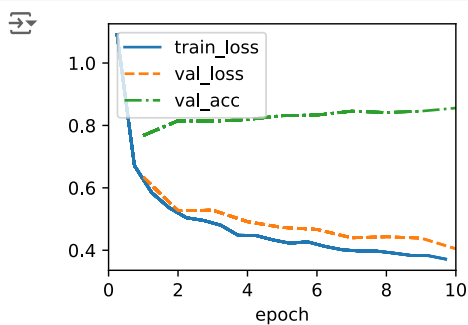


```
x.grad.data.zero_()
y.backward(torch.ones_like(x), retain_graph=True)
d2l.plot(x.detach(), x.grad, 'x', 'grad of swish', figsize=(5, 2.5))
```



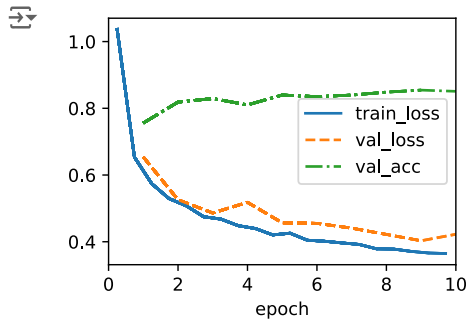
5.2 Exercises

```
model = MLP(num_outputs=10, num_hiddens=128, lr=0.1)
trainer.fit(model, data)
```



Lower number of hidden nodes leads to slower convergence to a low training loss but suprisingly high validation accuracy.

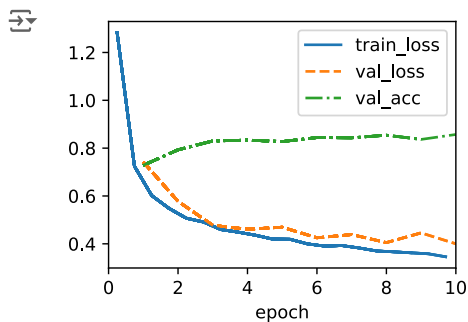
```
model = MLP(num_outputs=10, num_hiddens=512, lr=0.1)
trainer.fit(model, data)
```



Higher number of hidden nodes seems to lead to lower training loss but the validation accuracy does not see much improvement.

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

```
model = MLP(num_outputs=10, num_hiddens1=256, num_hiddens2=256, lr=0.1)
trainer.fit(model, data)
```



The result of adding another hidden layer leads to a bit higher validation accuracy but started from a higher training loss.

✓ 5.3 Exercises

- Assume that the inputs \mathbf{X} to some scalar function f are $n \times m$ matrices. What is the dimensionality of the gradient of f with respect to \mathbf{X} ?

The derivative is calculated by finding how much each values of \mathbf{X} affects the resulting f , which means it is the partial derivative of all inputs of \mathbf{X} with f . Since the number of partial derivatives is the same as the number of inputs, the resulting derivative also has the same shape or dimension of \mathbf{X} which is an $n \times m$ matrix.