Tensor(x): Num_Val들의 ls/ x.numel= elem#

차원(=Axis+1)/ 현재: 1Axis= 2차원

In [4]:
```
import torch
```

In [5]:
```
x= torch.arange(12, dtype=torch.float32)
x
```

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

In [6]:
```
x.numel()
```

Out[6]:
```
12
```

Shape: 12: 1차원/ 3,4 2차원

.reshape(col,row)

In [7]:
```
x.shape
```

Out[7]:
```
torch.Size([12])
```

In [8]:
```
X= x.reshape(3,4)
X
```

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

Elem 설정: zeros, ones, randn

In [9]:
```
torch.zeros((2,3,4))
```

Out[9]:
```
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 [10]:
```
torch.ones(2,3,4)
```

Out[10]:
```
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 [11]:
```
torch.randn(3,4)
```

Out[11]:
```
tensor([[ 2.3700,  1.0899, -0.3480,  0.0726],
        [ 0.0463,  0.6103,  2.2053, -1.3175],
        [ 0.0591,  0.8719, -0.8769,  0.8759]])
```

Tensor shape을 직접_val로 설정가능

In [12]:
```
torch.tensor([[2,1,4,3],[1,2,3,4],[4,3,2,1]])
```

Out[12]:
```
tensor([[2, 1, 4, 3],
        [1, 2, 3, 4],
        [4, 3, 2, 1]])
```

X[:2, :]: Access 1st,2nd row

In [13]:
```
X[:2, :] = 12
X
```

Out[13]:
```
tensor([[12., 12., 12., 12.],
        [12., 12., 12., 12.],
        [ 8.,  9., 10., 11.]])
```

In [14]:
```
x
```

Out[14]:
```
tensor([12., 12., 12., 12., 12., 12., 12., 12.,  8.,  9., 10., 11.])
```

.exp: map(R_num->R_num)

In [15]:
```
torch.exp(x)
```

Out[15]:
```
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])
```

두 tensor간 관계

In [16]:
```
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[16]:
```
(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.]))
```

.cat((X,Y)): tensor 합치기/ dim=0: col로 더하기, 1: row로 더하기

X==Y: 같은지, .sum()=값

In [17]:
```
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[17]:
```
(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 [18]:
```
X == Y
```

Out[18]:
```
tensor([[False,  True, False,  True],
        [False, False, False, False],
        [False, False, False, False]])
```

In [19]:
```
X.sum()
```

Out[19]:
```
tensor(66.)
```

In [20]:
```
a= torch.arange(3).reshape((3,1))
b = torch.arange(2).reshape((1, 2))
a, b
```

Out[20]:
```
(tensor([[0],
         [1],
         [2]]),
 tensor([[0, 1]]))
```

In [21]:
```
a + b
```

Out[21]:
```
tensor([[0, 1],
        [1, 2],
        [2, 3]])
```

값 비교

In [22]:
```
before=id(Y)
Y=Y+X
id(Y)==before
```

Out[22]:
```
False
```

In [23]:
```
before = id(X)
X += Y
id(X) == before
```

Out[23]:
```
True
```

In [24]:
```
Z = torch.zeros_like(Y)
print('id(Z):', id(Z))
Z[:] = X + Y
print('id(Z):', id(Z))
```

```
id(Z): 1936900511440
id(Z): 1936900511440
```

Convert to other py_obj

In [25]:
```
A = X.numpy()
B = torch.from_numpy(A)
type(A), type(B)
```

Out[25]: (numpy.ndarray, torch.Tensor)

In [26]:
```
a = torch.tensor([3.5])
a, a.item(), float(a), int(a)
```

Out[26]: (tensor([3.5000]), 3.5, 3.5, 3)

2.2. Data Preprocessing

In [27]:
```
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 [28]:
```
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 [29]:
```
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
```

NaN-> mean_Val로 바꾸기

In [30]:
```
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
```

Tensor로 바꾸기: 값.to_numpy(dtype=float)

In [31]:
```
import torch

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

Out[31]:
```
(tensor([[3., 0., 1.],
         [2., 0., 1.],
         [4., 1., 0.],
         [3., 0., 1.]], dtype=torch.float64),
 tensor([127500., 106000., 178100., 140000.], dtype=torch.float64))
```

선형대수학(Linear Algebra):

scala: tensor(elem 1개)

In [32]:
```
import torch
```

In [33]:
```
x = torch.tensor(3.0)
y = torch.tensor(2.0)

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

Out[33]:
```
(tensor(5.), tensor(6.), tensor(1.5000), tensor(9.))
```

**vector**(1차원 배열): .arange(#) 접근: x[idx]

In [34]:
```
x = torch.arange(3)
x
```

Out[34]:
```
tensor([0, 1, 2])
```

In [35]:
```
x[2]
```

Out[35]:
```
tensor(2)
```

In [36]:
```
len(x)
```

Out[36]:
```
3
```

In [37]:
```
x.shape
```

Out[37]:
```
torch.Size([3])
```

Matrix: 2차원 배열

In [38]:
```
A = torch.arange(6).reshape(3, 2)
A
```

Out[38]:
```
tensor([[0, 1],
        [2, 3],
        [4, 5]])
```

A.T: transpose(전치행렬): a11, a22 기준 뒤집기

같으면: symmetric(대칭)

In [39]:
```
A.T
```

Out[39]:
```
tensor([[0, 2, 4],
        [1, 3, 5]])
```

In [40]:
```
A = torch.tensor([[1, 2, 3], [2, 0, 4], [3, 4, 5]])
A == A.T
```

Out[40]:
```
tensor([[True, True, True],
        [True, True, True],
        [True, True, True]])
```

3차원: 2,3,4중 2(앞 값)을 제일 큰 단위부터 고려

In [41]:
```
torch.arange(24).reshape(2, 3, 4)
```

Out[41]:
```
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.clone(): 복사값

A+B, A*B: 같은 mat값끼리 계산

In [42]:
```
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[42]:
```
(tensor([[0., 1., 2.],
         [3., 4., 5.]]),
 tensor([[ 0.,  2.,  4.],
         [ 6.,  8., 10.]]))
```

In [43]:
```
A * B
```

Out[43]:
```
tensor([[ 0.,  1.,  4.],
        [ 9., 16., 25.]])
```

In [44]:
```
a = 2
X = torch.arange(24).reshape(2, 3, 4)
```

```
a + X, (a * X).shape
```

Out[44]:
```
(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 [45]:
```
x = torch.arange(3, dtype=torch.float32)
x, x.sum()
```

Out[45]:
```
(tensor([0., 1., 2.]), tensor(3.))
```

In [46]:
```
A.shape, A.sum()
```

Out[46]:
```
(torch.Size([2, 3]), tensor(15.))
```

axis=#: #번째 r

In [47]:
```
A.shape, A.sum(axis=0).shape
```

Out[47]:
```
(torch.Size([2, 3]), torch.Size([3]))
```

In [48]:
```
A.shape, A.sum(axis=1).shape
```

Out[48]:
```
(torch.Size([2, 3]), torch.Size([2]))
```

In [49]:
```
A.sum(axis=[0, 1]) == A.sum()  # Same as A.sum()
```

Out[49]:
```
tensor(True)
```

In [50]:
```
A.mean(), A.sum() / A.numel()
```

Out[50]:
```
(tensor(2.5000), tensor(2.5000))
```

In [51]:
```
A.mean(axis=0), A.sum(axis=0) / A.shape[0]
```

Out[51]:
```
(tensor([1.5000, 2.5000, 3.5000]), tensor([1.5000, 2.5000, 3.5000]))
```

In [52]:
```
sum_A = A.sum(axis=1, keepdims=True)
sum_A, sum_A.shape
```

Out[52]:
```
(tensor([[ 3.],
         [12.]]),
 torch.Size([2, 1]))
```

In [53]:
```
A / sum_A
```

Out[53]:
```
tensor([[0.0000, 0.3333, 0.6667],
        [0.2500, 0.3333, 0.4167]])
```

In [54]:
```
A.cumsum(axis=0)
```

Out[54]:
```
tensor([[0., 1., 2.],
        [3., 5., 7.]])
```

In [55]:
```
y = torch.ones(3, dtype = torch.float32)
x, y, torch.dot(x, y)
```

Out[55]:
```
(tensor([0., 1., 2.]), tensor([1., 1., 1.]), tensor(3.))
```

In [56]:
```
torch.sum(x * y)
```

Out[56]:
```
tensor(3.)
```

In [57]:
```
A.shape, x.shape, torch.mv(A, x), A@x
```

Out[57]:
```
(torch.Size([2, 3]), torch.Size([3]), tensor([ 5., 14.]), tensor([ 5., 14.]))
```

In [58]:
```
B = torch.ones(3, 4)
torch.mm(A, B), A@B
```

Out[58]:
```
(tensor([[ 3.,  3.,  3.,  3.],
         [12., 12., 12., 12.]]),
 tensor([[ 3.,  3.,  3.,  3.],
         [12., 12., 12., 12.]]))
```

.norm: 피타고라스(총 거리값)

In [59]:
```
u = torch.tensor([3.0, -4.0])
torch.norm(u)
```

Out[59]:
```
tensor(5.)
```

In [60]:
```
torch.abs(u).sum()
```

Out[60]:
```
tensor(7.)
```

In [61]:
```
torch.norm(torch.ones((4, 9)))
```

Out[61]:
```
tensor(6.)
```

Diff

In [62]:
```
import torch
```

In [63]:
```
x = torch.arange(4.0)
x
```

Out[63]:
```
tensor([0., 1., 2., 3.])
```

In [64]:
```
# Can also create x = torch.arange(4.0, requires_grad=True)
x.requires_grad_(True)
x.grad  # The gradient is None by default
```

In [65]:
```
y = 2 * torch.dot(x, x)
y
```

Out[65]:
```
tensor(28., grad_fn=<MulBackward0>)
```

In [66]:
```
y.backward()
x.grad
```

Out[66]:
```
tensor([ 0.,  4.,  8., 12.])
```

In [67]:
```
x.grad == 4 * x
```

Out[67]:
```
tensor([True, True, True, True])
```

In [68]:
```
x.grad.zero_()  # Reset the gradient
y = x.sum()
y.backward()
x.grad
```

Out[68]:
```
tensor([1., 1., 1., 1.])
```

In [69]:
```
x.grad.zero_()
y = x * x
y.backward(gradient=torch.ones(len(y)))  # Faster: y.sum().backward()
x.grad
```

Out[69]:
```
tensor([0., 2., 4., 6.])
```

In [70]:
```
x.grad.zero_()
y = x * x
u = y.detach()
z = u * x

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

Out[70]:
```
tensor([True, True, True, True])
```

In [71]:
```
x.grad.zero_()
y.sum().backward()
x.grad == 2 * x
```

Out[71]:
```
tensor([True, True, True, True])
```

In [72]:
```
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 [73]:
```
a = torch.randn(size=(), requires_grad=True)
d = f(a)
d.backward()
```

In [74]:
```
a.grad == d / a
```

Out[74]:
```
tensor(False)
```

Linear Regression: W1x1+W2x2+b

In [75]:
```
# !pip install d2l==1.0.3
```

In [76]:
```
%matplotlib inline
import math
import time
import numpy as np
import torch
from d2l import torch as d2l
```

a,b: vec(all 1)

In [77]:
```
n = 10000
a = torch.ones(n)
b = torch.ones(n)
```

Vectorization for speed

In [78]:
```
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[78]:
```
'0.12574 sec'
```

"2nd" is faster: **Vectorizing** is Fast

In [79]:
```
t = time.time()
d = a + b
f'{time.time() - t:.5f} sec'
```

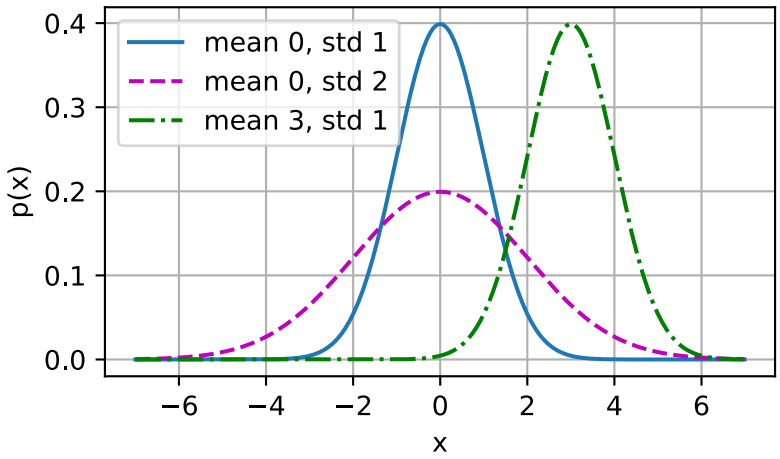
Out[79]:
```
'0.00100 sec'
```

In [80]:
```
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 [81]:
```
# 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])
```




In [83]:
```
import time
import numpy as np
import torch
from torch import nn
from d2l import torch as d2l
```

In [84]:
```
def add_to_class(Class):  #@save
    """Register functions as methods in created class."""
    def wrapper(obj):
        setattr(Class, obj.__name__, obj)
    return wrapper
```

In [85]:
```
class A:
    def __init__(self):
        self.b = 1

a = A()
```

In [86]:
```
@add_to_class(A)
def do(self):
    print('Class attribute "b" is', self.b)

a.do()
```

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

In [87]:
```
class HyperParameters:  #@save
    """The base class of hyperparameters."""
    def save_hyperparameters(self, ignore=[]):
        raise NotImplemented
```

In [88]:
```
# 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'))
```

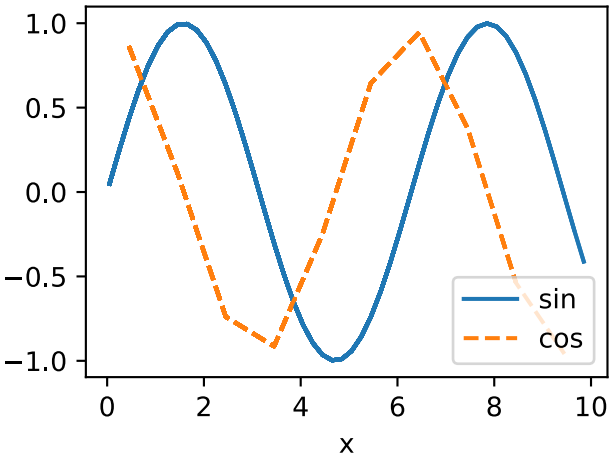
```
b = B(a=1, b=2, c=3)
```

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

In [89]:
```
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', '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 [90]:
```
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 [91]:
```
class Module(nn.Module, d2l.HyperParameters):  #@save
    """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 [92]:
```
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)
```

In [93]:
```
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
```

In [95]:
```
%matplotlib inline
import torch
from d2l import torch as d2l
```

In [96]:
```
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=Tru
        self.b = torch.zeros(1, requires_grad=True)
```

In [97]:
```
@d2l.add_to_class(LinearRegressionScratch)  #@save
def forward(self, X):
    return torch.matmul(X, self.w) + self.b
```

In [98]:
```
@d2l.add_to_class(LinearRegressionScratch)  #@save
def loss(self, y_hat, y):
    l = (y_hat - y) ** 2 / 2
    return l.mean()
```

In [99]:
```
class SGD(d2l.HyperParameters):  #@save
    """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_()
```

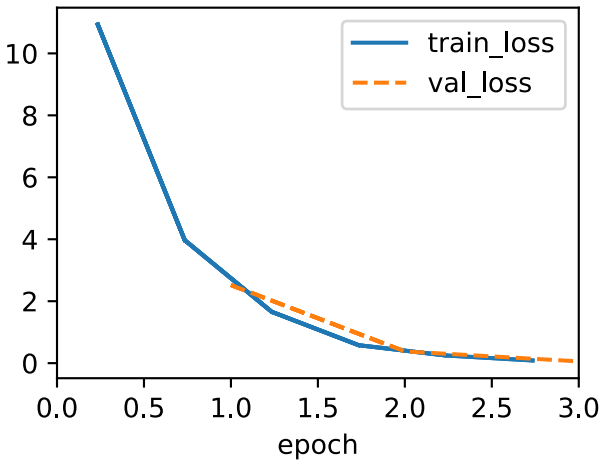
In [100…
```
@d2l.add_to_class(LinearRegressionScratch)  #@save
def configure_optimizers(self):
    return SGD([self.w, self.b], self.lr)
```

In [101…
```
@d2l.add_to_class(d2l.Trainer)  #@save
def prepare_batch(self, batch):
```

```
    return batch
```

In [102…
```
@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
```

In [103…
```
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 [104…
```
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([ 0.1175, -0.2042])
error in estimating b: tensor([0.2403])
```

In [106…
```
%matplotlib inline
import time
import torch
import torchvision
from torchvision import transforms
from d2l import torch as d2l
```

```
d2l.use_svg_display()
```

# Softmax Class: 1Layer

Loss_Func

**Img Classif**: Fashion img를 Classif하기

In [107…
```
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=True)
        self.val = torchvision.datasets.FashionMNIST(
            root=self.root, train=False, transform=trans, download=True)
```

In [108…
```
data = FashionMNIST(resize=(32, 32))
len(data.train), len(data.val)
```

```
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz to ../data\FashionMNIST\raw\train-images-idx3-ubyte.gz
100.0%
Extracting ../data\FashionMNIST\raw\train-images-idx3-ubyte.gz to ../data\FashionMNIST\raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz to ../data\FashionMNIST\raw\train-labels-idx1-ubyte.gz
100.0%
Extracting ../data\FashionMNIST\raw\train-labels-idx1-ubyte.gz to ../data\FashionMNIST\raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz to ../data\FashionMNIST\raw\t10k-images-idx3-ubyte.gz
100.0%
```

```
Extracting ../data\FashionMNIST\raw\t10k-images-idx3-ubyte.gz to ../data\Fa
shionMNIST\raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k
-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k
-labels-idx1-ubyte.gz to ../data\FashionMNIST\raw\t10k-labels-idx1-ubyte.gz
100.0%
Extracting ../data\FashionMNIST\raw\t10k-labels-idx1-ubyte.gz to ../data\Fa
shionMNIST\raw
```

Out[108… (60000, 10000)

32*32 pixel

In [109…
```
data.train[0][0].shape
```

Out[109… torch.Size([1, 32, 32])

In [110…
```
@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]
```

In [111…
```
@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, self.batch_size, shuffle=trai
                                       num_workers=self.num_workers)
```

In [112…
```
X, y = next(iter(data.train_dataloader()))
print(X.shape, X.dtype, y.shape, y.dtype)
```

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

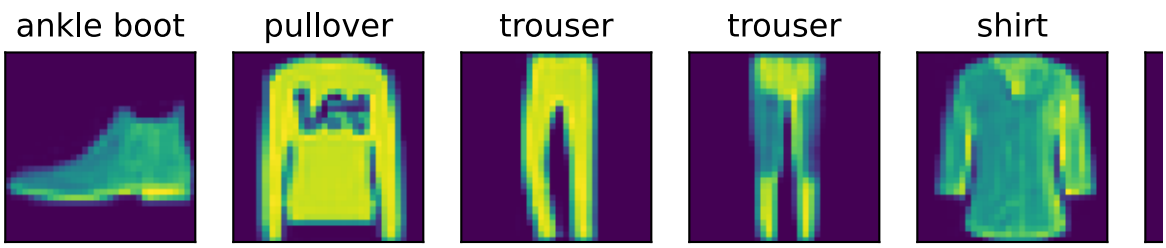
In [113…
```
tic = time.time()
for X, y in data.train_dataloader():
    continue
f'{time.time() - tic:.2f} sec'
```

Out[113… '9.16 sec'

In [114…
```
def show_images(imgs, num_rows, num_cols, titles=None, scale=1.5):  #@save
    """Plot a list of images."""
    raise NotImplementedError
```

In [115…
```
@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)
```




In [116…
```
# !pip install d2l==1.0.3s
```

Base Classif Model

In [117…
```
import torch
from d2l import torch as d2l
```

In [118…
```
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)
```

In [119…
```
@d2l.add_to_class(d2l.Module)  #@save
def configure_optimizers(self):
    return torch.optim.SGD(self.parameters(), lr=self.lr)
```

In [120…
```
@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
```

In [122…
```
import torch
from d2l import torch as d2l
```

Dim 0: Sum w/ **row** Dim 1: Sum w/ **col**

In [123…
```
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[123…
```
(tensor([[5., 7., 9.]]),
 tensor([[ 6.],
         [15.]]))
```

In [124…
```
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 [125…
```
X = torch.rand((2, 5))
X_prob = softmax(X)
X_prob, X_prob.sum(1)
```

Out[125…
```
(tensor([[0.1293, 0.2529, 0.2729, 0.1535, 0.1914],
         [0.3094, 0.1895, 0.2066, 0.1500, 0.1445]]),
 tensor([1., 1.]))
```

In [126…
```
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 [127…
```
@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 [128…
```
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[128…
```
tensor([0.1000, 0.5000])
```

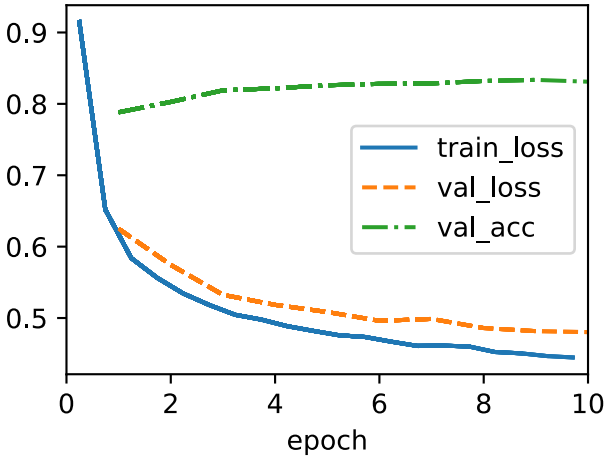
In [129…
```
def cross_entropy(y_hat, y):
    return -torch.log(y_hat[list(range(len(y_hat))), y]).mean()

cross_entropy(y_hat, y)
```

Out[129…
```
tensor(1.4979)
```

In [130…
```
@d2l.add_to_class(SoftmaxRegressionScratch)
def loss(self, y_hat, y):
    return cross_entropy(y_hat, y)
```

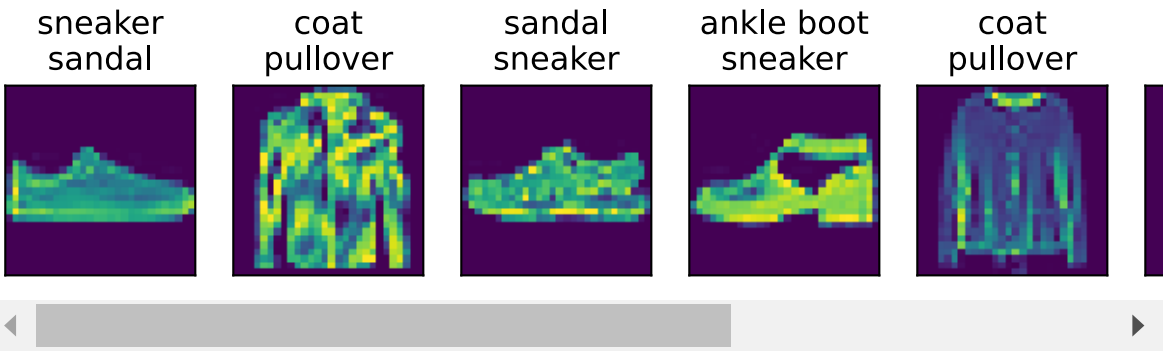
In [131…
```
data = d2l.FashionMNIST(batch_size=256)
model = SoftmaxRegressionScratch(num_inputs=784, num_outputs=10, lr=0.1)
trainer = d2l.Trainer(max_epochs=10)
trainer.fit(model, data)
```




In [132…
```
X, y = next(iter(data.val_dataloader()))
preds = model(X).argmax(axis=1)
preds.shape
```

Out[132… torch.Size([256])

In [133…
```
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)
```




In [134…
```
# !pip install d2l==1.0.3
```

# MLP: NonLinear Model

Solve: Integrate **Hidden_Layer**

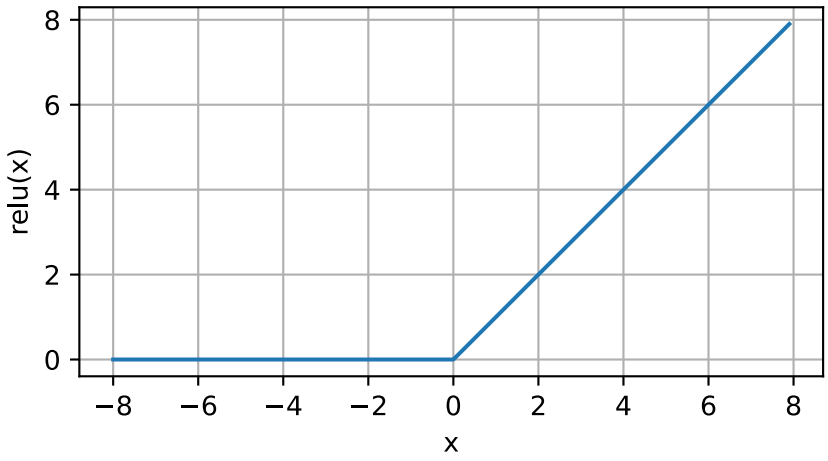
In [135…
```
%matplotlib inline
import torch
from d2l import torch as d2l
```

## Activ Func

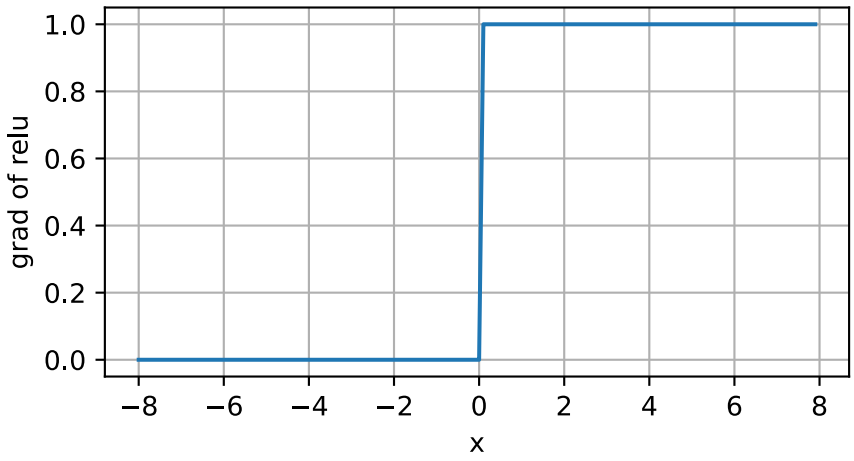
1. ReLU: Maintain Plus_Val

- Diff_Val: P=1, N=0

In [136…
```
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))
```



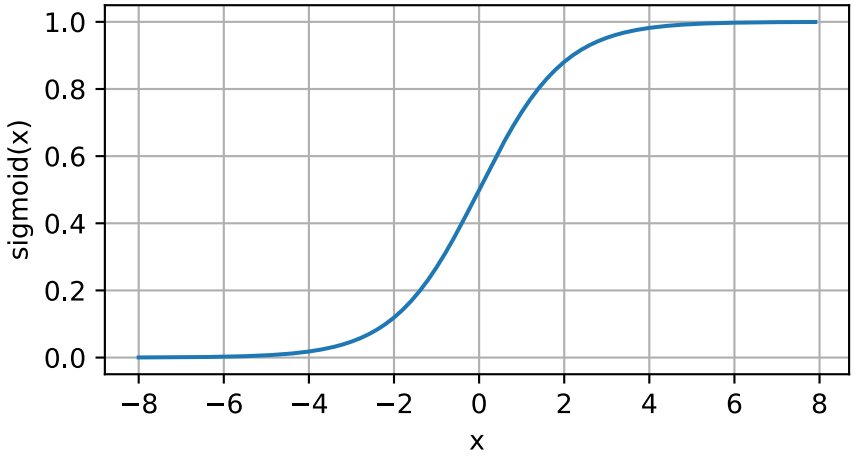

In [137…
```
y.backward(torch.ones_like(x), retain_graph=True)
d2l.plot(x.detach(), x.grad, 'x', 'grad of relu', figsize=(5, 2.5))
```




2. Sigmoid:

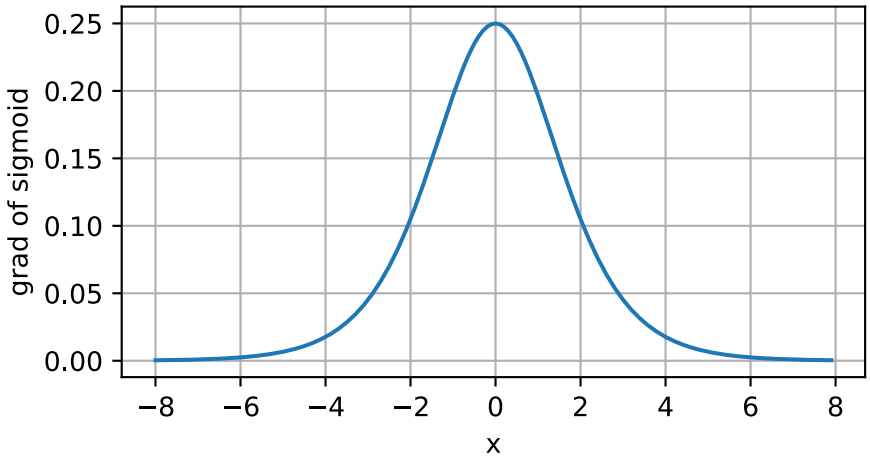
- Diff_Val: 0:0.25, 벗어나며: 0

In [138…
```
y = torch.sigmoid(x)
d2l.plot(x.detach(), y.detach(), 'x', 'sigmoid(x)', figsize=(5, 2.5))
```

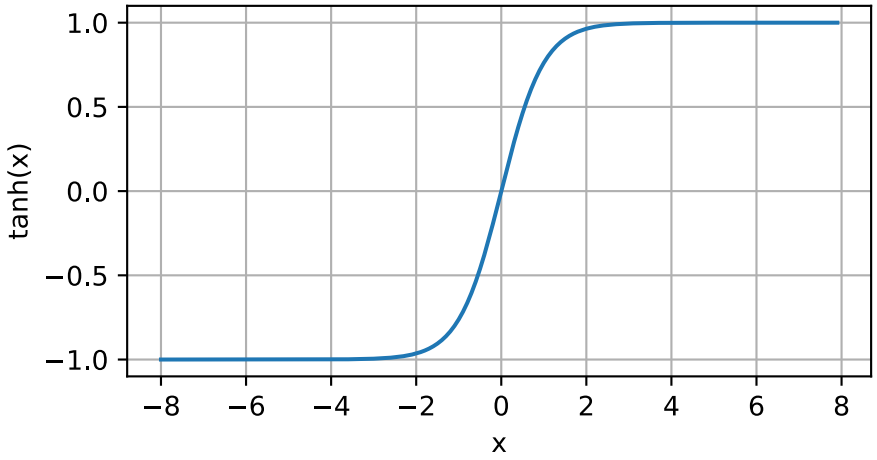



3. tan

In [139…
```
# 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))
```

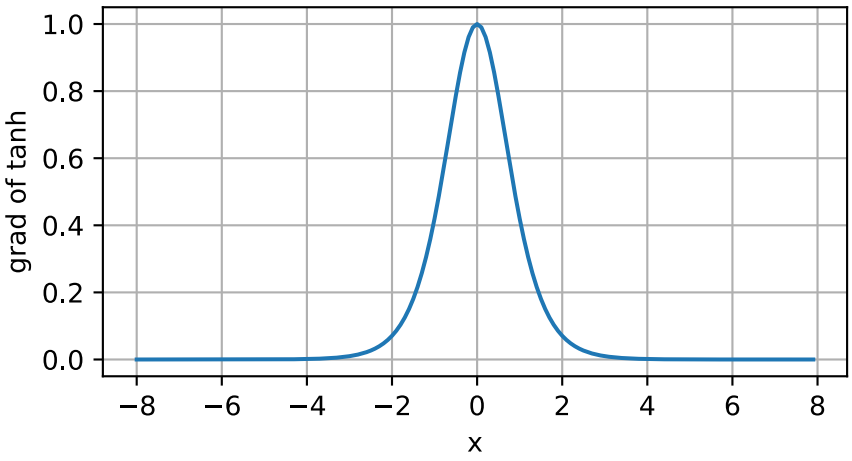



In [140…
```
y = torch.tanh(x)
d2l.plot(x.detach(), y.detach(), 'x', 'tanh(x)', figsize=(5, 2.5))
```




In [141…
```
# 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))
```




In [142…
```
# !pip install d2l==1.0.3
```

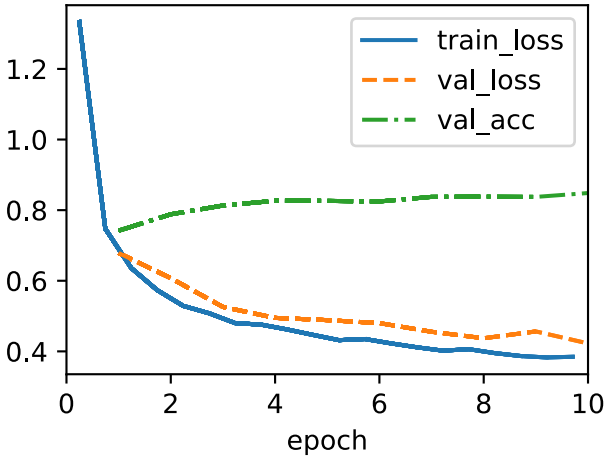
In [143…
```
import torch
from torch import nn
from d2l import torch as d2l
```

In [144…
```
class MLPScratch(d2l.Classifier):
    def __init__(self, num_inputs, num_outputs, num_hiddens, lr, sigma=0.0
        super().__init__()
        self.save_hyperparameters()
        self.W1 = nn.Parameter(torch.randn(num_inputs, num_hiddens) * sigm
        self.b1 = nn.Parameter(torch.zeros(num_hiddens))
        self.W2 = nn.Parameter(torch.randn(num_hiddens, num_outputs) * sig
        self.b2 = nn.Parameter(torch.zeros(num_outputs))
```

In [145…
```
def relu(X):
    a = torch.zeros_like(X)
    return torch.max(X, a)
```

In [146…
```
@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 [147…
```
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 [148…
```
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 [149…
```
model = MLP(num_outputs=10, num_hiddens=256, lr=0.1)
trainer.fit(model, data)
```