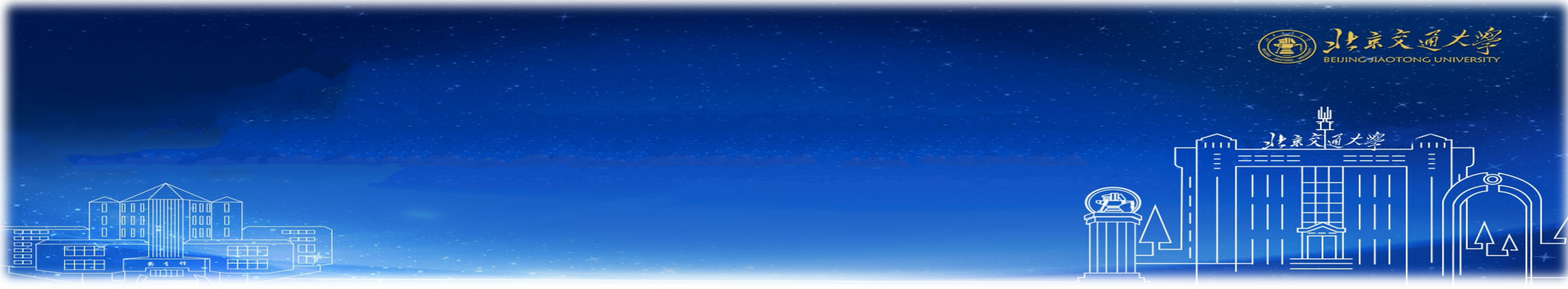
实验3 网络优化实验

北京交通大学 《深度学习》课程组



目录

1.1 前馈神经网络的复习

➢ 组成结构

- 输入神经元个数：$d = 4$
- 隐藏层神经元个数：$h = 5$
- 输出层神经元个数：$q = 3$
- 给的小批量样本 $X \in R^{n*d}$
- 隐藏层的输出为 $H \in R^{n*h}$
- 输出层的输出为 $O \in R^{n*q}$

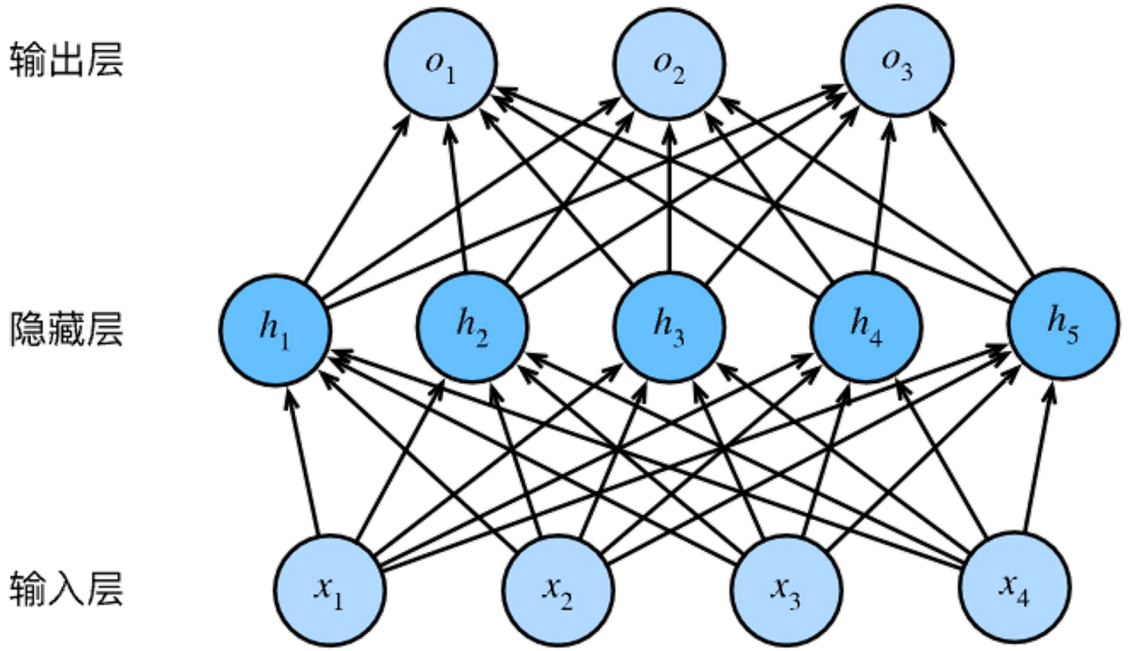
➢ 计算公式

$$H = XW_h^T + b_h$$

$$O = HW_O^T + b_O$$

$W_h \in R^{h*d}$, $b_h \in R^{1*h}$

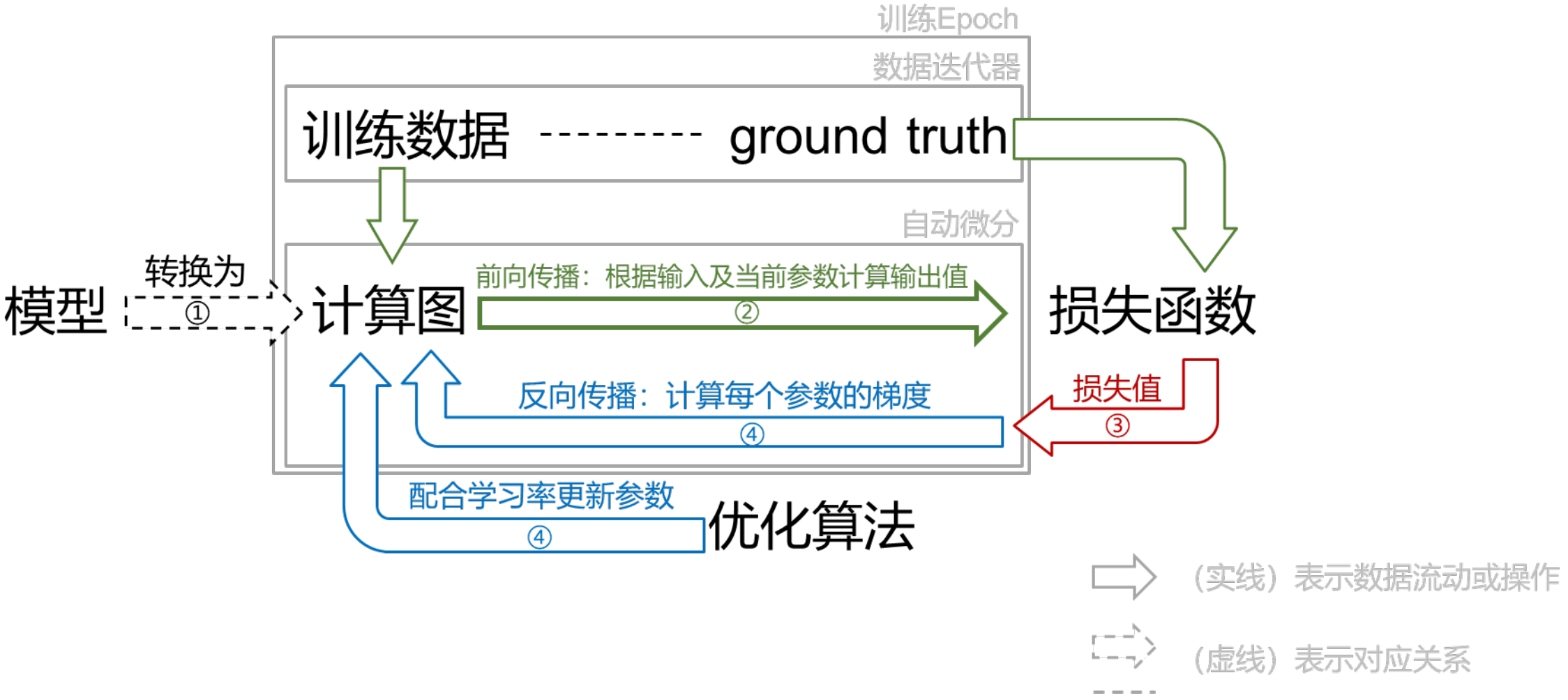
$W_o \in R^{q*h}$, $b_o \in R^{1*q}$



包含一个隐藏层的前馈神经网络

1.1 前馈神经网络的复习

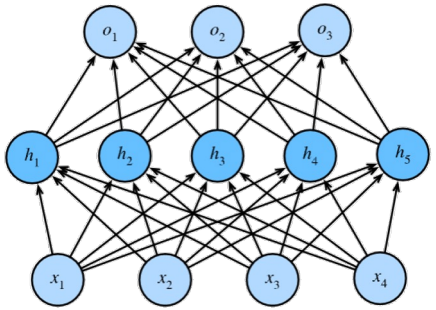
■ 前馈神经网络的参数学习过程



1.1 前馈神经网络的复习

■ 手动实现多分类任务

```python
# 2) 定义模型
class Net():
    def __init__(self):
        # 定义并初始化模型参数
        num_inputs, num_outputs, num_hiddens = 784, 10, 256
        W1 = torch.tensor(np.random.normal(0, 0.01, (num_hiddens, num_inputs)), dtype=torch.float)
        b1 = torch.zeros(num_hiddens, dtype=torch.float)
        W2 = torch.tensor(np.random.normal(0, 0.01, (num_outputs, num_hiddens)), dtype=torch.float)
        b2 = torch.zeros(num_outputs, dtype=torch.float)
        # 告知PyTorch框架，上述四个变量需求梯度
        self.params = [W1, b1, W2, b2]
        for param in self.params:
            param.requires_grad = True

        # 定义模型结构
        self.input_layer = lambda x: x.view(x.shape[0], -1)
        self.hidden_layer = lambda x: self.my_ReLU(torch.matmul(x, W1.t()) + b1)
        self.output_layer = lambda x: torch.matmul(x, W2.t()) + b2

    @staticmethod
    def my_ReLU(x):
        return torch.max(input=x, other=torch.tensor(0.0))

    def forward(self, x):
# 3) 定义模型前向传播过程
        flatten_input = self.input_layer(x)
        hidden_output = self.hidden_layer(flatten_input)
        final_output = self.output_layer(hidden_output)
        return final_output
```



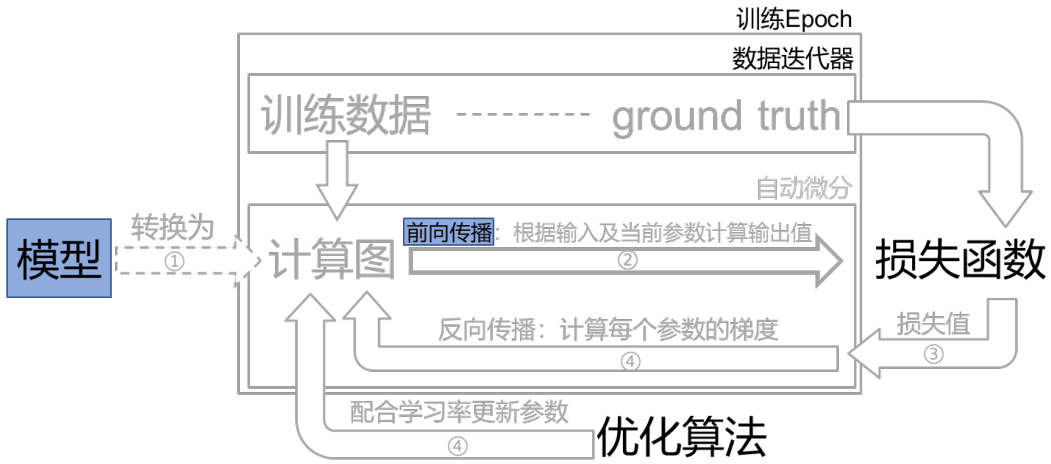
$Output\ Layer$: 10

$W_2 \in R^{10\times256}$

$Hidden\ Layer$: 256

$W_1 \in R^{256\times784}$

$Input\ Layer$: $28\times28\times1 = 784$



1.2 优化器的使用

■ 常用的优化器：SGD、AdaGrad、RMSProp等

➢ **PyTorch中实现的SGD**

```
optimizer = torch.optim.SGD(model.parameters(), lr)
```

➢ **手动实现SGD**

- **求得的梯度是一定要除batch_size的，下面二者的区别在于这个求平均的操作是交由优化函数还是损失函数处理**

```
# 带小批量的随机梯度下降
def sgd(params, lr, batch_size):
    for param in params:
        param.data -= lr * param.grad / batch_size

# 均方误差损失
def squared_loss(y_hat, y):
    return (y_hat - y.view(y_hat.size())) ** 2 / 2

# 训练函数
for epoch in range(num_epochs):
    for X, y in data_iter(batch_size, features, labels):
        l = loss(net(X, w, b), y).sum()
        # ...
```

实验一实现的，**带小批量**的随机梯度下降

```
# 随机梯度下降
def sgd(params, lr):
    for param in params:
        param.data -= lr * param.grad

# 均方误差损失
loss = torch.nn.CrossEntropyLoss()

# 训练函数
for epoch in range(num_epochs):
    for X, y in data_iter(batch_size, features, labels):
        l = loss(net(X), y)
        # ...
```

随机梯度下降

目录

1. 基本概念

- 前馈神经网络的复习
- 优化器的使用

2. 模型调优

- 交叉验证
- 过拟合&欠拟合
- 探究导致过拟合、欠拟合的因素
- 过拟合解决办法：正则化、dropout
- 不同的优化算法

3. 实验要求

- 数据集介绍
- 多分类任务数据集下载和读取
- 课程实验要求

2.1 交叉验证

■ K折交叉验证

- 将数据集划分为k个大小相似的互斥子集，每次用k-1个子集的并集作为训练集，余下的子集作为测试集，最终返回k个测试结果的均值，k 最常用的取值是10。

■ 手动实现K折交叉验证

➢ 创建数据集

```
# 导入模块
import numpy as np
import random
# 创建一个数据集
X = torch.rand(100, 32, 32)
Y = torch.rand(100, 1)
# random shuffle
index = [i for i in range(len(X))]
random.shuffle(index)
X = X[index]
Y = Y[index]
```

2.1 交叉验证

- **获取k折交叉验证某一折的训练集和验证集**

```
def get_kfold_data(k, i, X, y):

    # 返回第 i+1 折 (i = 0 -> k-1) 交叉验证时所需要的训练和验证数据，X_train为训练集，X_valid为验证集
    fold_size = X.shape[0] // k  # 每份的个数:数据总条数/折数（组数）

    val_start = i * fold_size
    if i != k - 1:
        val_end = (i + 1) * fold_size
        X_valid, y_valid = X[val_start:val_end], y[val_start:val_end]
        X_train = torch.cat((X[0:val_start], X[val_end:]), dim = 0)
        y_train = torch.cat((y[0:val_start], y[val_end:]), dim = 0)
    else:  # 若是最后一折交叉验证
        X_valid, y_valid = X[val_start:], y[val_start:]      # 若不能整除，将多的样本放在最后一折里
        X_train = X[0:val_start]
        y_train = y[0:val_start]

    return X_train, y_train, X_valid,y_valid
```

分为k份后每份的个数

2.1 交叉验证

- **依次对每一折数据进行训练和测试，并计算k折平均值**

```
def k_fold(k, X_train, y_train):

    train_loss_sum, valid_loss_sum = 0, 0
    train_acc_sum , valid_acc_sum = 0, 0

    for i in range(k):
        print('第', i + 1,'折验证结果')
        data = get_kfold_data(k, i, X_train, y_train)     # 获取k折交叉验证的训练和验证数据
        net = Net()  # 实例化模型（某已经定义好的模型）
        # 对每份数据进行训练
        train_loss, val_loss, train_acc, val_acc = train(net, *data)

        train_loss_sum += train_loss
        valid_loss_sum += val_loss
        train_acc_sum += train_acc
        valid_acc_sum += val_acc

    print('\n','最终k折交叉验证结果：')


    print('average train loss:{:.4f}, average train accuracy:{:.3f}%'.format(train_loss_sum/k, train_acc_sum/k))
    print('average valid loss:{:.4f}, average valid accuracy:{:.3f}%'.format(valid_loss_sum/k, valid_acc_sum/k))

    return
```

循环K次，取平均值

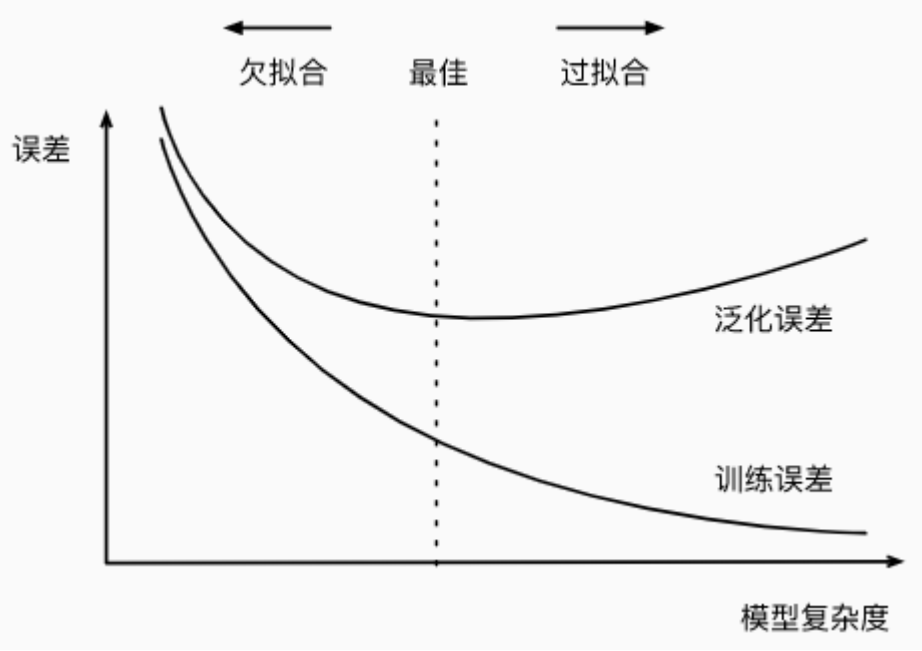
2.2 过拟合&欠拟合

过拟合

- 表现：模型在训练集上正确率很高，但是在测试集上正确率很低
- 造成原因：由于训练数据少、数据存在噪声以及模型能力过强等原因造成的过拟合
- 解决办法：优化目标加正则项；Dropout；早停机制

欠拟合

- 表现：模型在训练集和测试集上的正确率都很低
- 造成原因：由于模型能力不足造成的
- 解决办法：增加模型复杂度



2.3 多项式函数拟合实验探究影响欠拟合、过拟合的因素

■ 给定样本特征，使用如下的三阶多项式函数来生成样本的标签

$$y = 1.2x - 3.4x^2 + 5.6x^3 + 5 + \epsilon$$

➢ 设置噪声项ϵ服从均值为0、标准差为0.1的正态分布。训练数据集和测试数据集的样本数都设为100

```
n_train, n_test, true_w, true_b = 100, 100, [1.2, -3.4, 5.6], 5
features = torch.randn((n_train + n_test, 1))
poly_features = torch.cat((features, torch.pow(features, 2), torch.pow(features, 3)), 1)
labels = (true_w[0] * poly_features[:, 0] + true_w[1] * poly_features[:, 1]
          + true_w[2] * poly_features[:, 2] + true_b)
labels += torch.tensor(np.random.normal(0, 0.01, size=labels.size()), dtype=torch.float)
print(features[0], labels[0])
```

```
tensor([0.3509]) tensor(5.2411)
```

构造成$[x, x^2, x^3]$的形式

➢ 定义作图函数Draw_Loss_Curve

```
def Draw_Loss_Curve(x_vals, y_vals, x_label, y_label, x2_vals=None, y2_vals=None,
                    legend=None, figsize=(3.5, 2.5)):
    display.set_matplotlib_formats('svg')
    plt.rcParams['figure.figsize'] = figsize
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.semilogy(x_vals, y_vals)
    if x2_vals and y2_vals:
        plt.semilogy(x2_vals, y2_vals, linestyle=':')
        plt.legend(legend)
```

2.3 多项式函数拟合实验探究导致欠拟合、过拟合的因素

➢ 模型定义和训练函数定义

```
num_epochs, loss = 100, torch.nn.MSELoss()
def fit_and_plot(train_features, test_features, train_labels, test_labels):
    #参数形状由输入数据的形状决定，由此来控制模型不同函数对原函数的拟合
    net = torch.nn.Linear(train_features.shape[-1], 1)
    #数据划分
    batch_size = min(10, train_labels.shape[0])
    dataset = torch.utils.data.TensorDataset(train_features, train_labels)
    train_iter = torch.utils.data.DataLoader(dataset, batch_size, shuffle=True)
    #训练模型
    optimizer = torch.optim.SGD(net.parameters(), lr=0.01)
    train_ls, test_ls = [], []
    for _ in range(num_epochs):
        for X, y in train_iter:
            l = loss(net(X), y.view(-1, 1))
            optimizer.zero_grad()
            l.backward()
            optimizer.step()
        train_labels = train_labels.view(-1, 1)
        test_labels = test_labels.view(-1, 1)
        train_ls.append(loss(net(train_features), train_labels).item())
        test_ls.append(loss(net(test_features), test_labels).item())
    print('final epoch: train loss', train_ls[-1], 'test loss', test_ls[-1])
    Draw_Loss_Curve(range(1, num_epochs + 1), train_ls, 'epochs', 'loss',
             range(1, num_epochs + 1), test_ls, ['train', 'test'])
    print('weight:', net.weight.data,
          '\nbias:', net.bias.data)
```

由传的参数来控制构造不同的模型

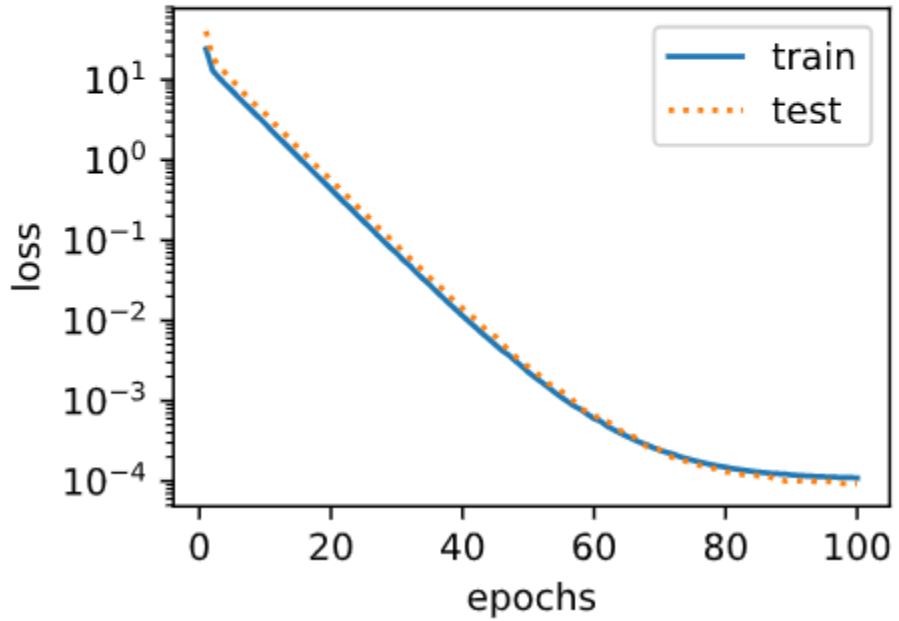
2.3 多项式函数拟合实验探究导致欠拟合、过拟合的因素

■ 使用三阶多项式函数拟合

➢ 使用与数据生成函数同阶的**三阶多项式函数**拟合，学习到的模型参数**接近真实值**

```
fit_and_plot(poly_features[:n_train, :], poly_features[n_train:, :],
             labels[:n_train], labels[n_train:])
```

```
final epoch: train loss 0.00010961104999296367 test loss 9.270678856410086e-05
weight: tensor([[ 1.2058, -3.4001,  5.5984]])
bias: tensor([4.9980])
```



真实：$y = 1.2x - 3.4x^2 + 5.6x^3 + 5 + \epsilon$

训练：$y = 1.206x - 3.4x^2 + 5.598x^3 + 4.998$

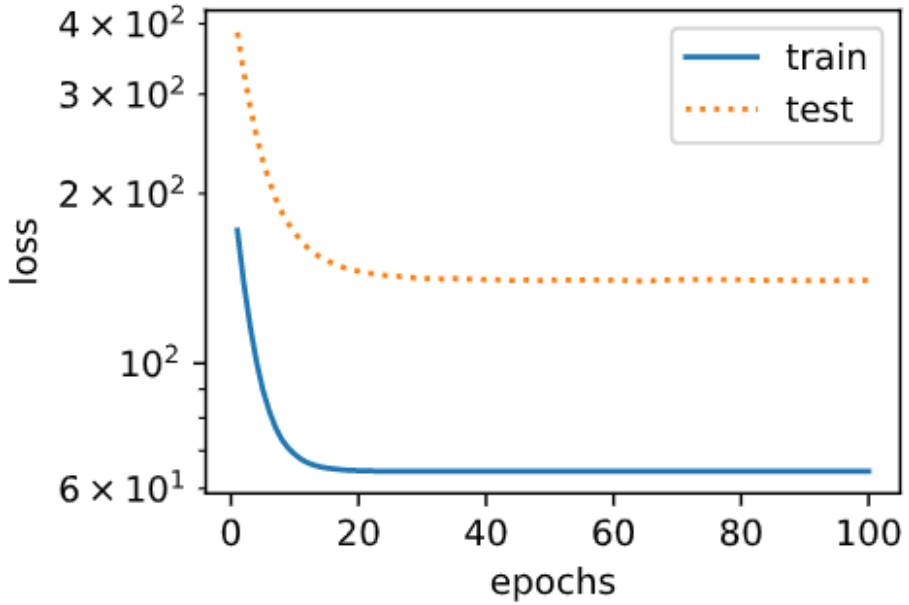
2.3 多项式函数拟合实验探究导致欠拟合、过拟合的因素

■ 使用线性函数拟合（欠拟合）

➢ 将模型复杂度降低：使用**线性函数**。训练集和测试集的loss在后期均很难下降，出现**欠拟合**

```
fit_and_plot(features[:n_train, :], features[n_train:, :], labels[:n_train],
             labels[n_train:])
```

```
final epoch: train loss 64.31674194335938 test loss 140.50250244140625
weight: tensor([[12.6037]])
bias: tensor([1.7815])
```



真实：$y = 1.2x - 3.4x^2 + 5.6x^3 + 5 + \epsilon$

训练：$y = 12.6x + 1.78$

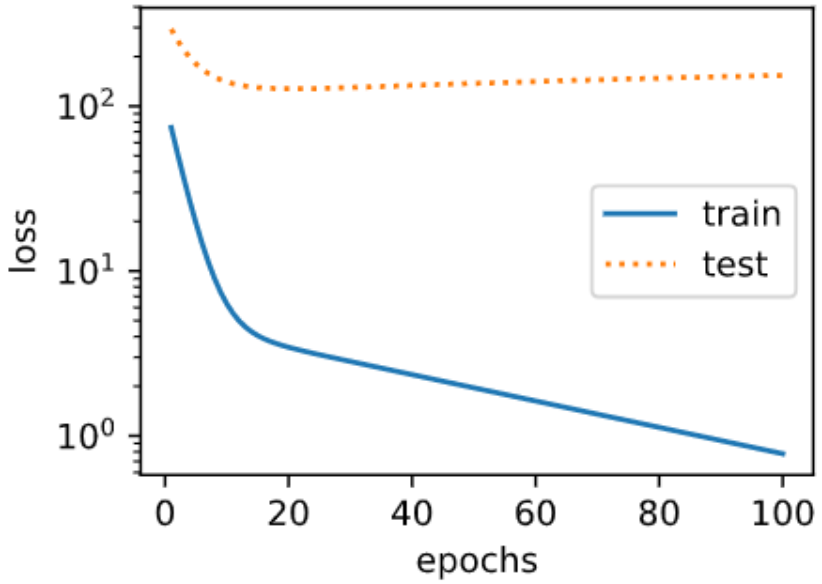
2.3 函数拟合实验探究导致欠拟合、过拟合的因素

■ 训练样本过少（过拟合）

➢ 只使用**两个样本**来训练模型，训练集loss持续下降，测试集loss上升，出现了**过拟合**

```
fit_and_plot(poly_features[0:2, :], poly_features[n_train:, :], labels[0:2],
             labels[n_train:])
```

```
final epoch: train loss 0.7790262699127197 test loss 153.86997985839844
weight: tensor([[2.0786, 1.6235, 2.4390]])
bias: tensor([2.9667])
```



真实：$y = 1.2x - 3.4x^2 + 5.6x^3 + 5 + \epsilon$

训练：$y = 2.08x + 1.62x^2 + 2.44x^3 + 2.97$

2.4 过拟合问题的常用方法——L_2范数正则化

- 在模型原损失函数基础上添加L_2范数惩罚项，通过惩罚绝对值较大的模型参数为需要学习的模型增加限制，来应对过拟合问题。带有L_2范数惩罚项的模型的新损失函数为:

$$\ell_0 + \frac{\lambda}{2}|\mathbf{w}|^2$$

其中w是参数向量，ℓ_o是模型原损失函数，n是样本个数，λ是超参数

■ 以高维线性回归为例来引入一个过拟合问题，并使用权重衰减来应对过拟合

设数据样本特征的维度为p，使用如下函数生成样本的标签

$$y = 0.05 + \sum_{i=1}^{p} 0.01x_i + \epsilon$$

其中噪声项ϵ服从均值为0、标准差为0.01的正态分布。设p=200，设置训练集样本数为20，测试集样本数为100来引入过拟合的情况。

2.4 过拟合问题的常用方法—— L_2 范数正则化

➢ 生成数据集

In [30]:
```
%matplotlib inline
import torch
import torch.nn as nn
from torch.utils import data
import numpy as np
import sys
sys.path.append("..")
from matplotlib import pyplot as plt
from IPython import display
```

In [16]:
```
n_train, n_test, num_inputs = 20, 100, 200
true_w, true_b = torch.ones(num_inputs, 1) * 0.01, 0.05
#生成数据集
features = torch.randn((n_train + n_test, num_inputs))
labels = torch.matmul(features, true_w) + true_b
labels += torch.tensor(np.random.normal(0, 0.01, size=labels.size()), dtype=torch.float)
train_features, test_features = features[:n_train, :], features[n_train:, :]
train_labels, test_labels = labels[:n_train], labels[n_train:]
print(train_features[0][:5]) #输出第一个样本特征向量的前五维的元素
print(train_labels[0])
```

$$y = 0.05 + \sum_{i=1}^{p} 0.01x_i + \epsilon$$

```
tensor([ 0.1348,  0.3261, -1.4309, -1.4814,  0.4257])
tensor([0.2408])
```

2.4 过拟合问题的常用方法——L_2范数正则化

■ 手动实现L_2范数正则化

➢ 定义随机初始化模型参数的函数

```
def init_params():
    w = torch.randn((num_inputs, 1), requires_grad=True)
    b = torch.zeros(1, requires_grad=True)
    return [w, b]
```

➢ 定义均方误差

```
def squared_loss(y_hat, y):
    # 返回的是向量,注意：pytorch里的MSELoss并没有除以 2
    return ((y_hat - y.view(y_hat.size())) ** 2) / 2
```

➢ 定义L_2范数惩罚项

```
def l2_penalty(w):
    return (w**2).sum() / 2
```

➢ 定义随机梯度下降函数

```
def SGD(params, lr):
    for param in params:
        # 注意这里参数赋值用的是param.data
        param.data -= lr * param.grad
```

➢ 定义模型

```
def linear(X, w, b):
    return torch.mm(X, w) + b
```

2.4 过拟合问题的常用方法—— L_2 范数正则化

➢ 定义训练函数

```
batch_size, num_epochs, lr = 1, 100, 0.003
net, loss = linear, squared_loss
#划分数据集
dataset = torch.utils.data.TensorDataset(train_features, tr
train_iter = torch.utils.data.DataLoader(dataset, batch_siz
#训练模型
def fit_and_plot(lambd):
    w, b = init_params()
    train_ls, test_ls = [], []
    for _ in range(num_epochs):
        for X, y in train_iter:
            # 添加了L2范数惩罚项
            l = loss(net(X, w, b), y) + lambd * l2_penalty
            l = l.sum()

            if w.grad is not None:
                w.grad.data.zero_()
                b.grad.data.zero_()
            l.backward()
            SGD([w, b], lr)
        train_ls.append(loss(net(train_features, w, b), train_labels).mean().item())
        test_ls.append(loss(net(test_features, w, b), test_labels).mean().item())
    Draw_Loss_Curve(range(1, num_epochs + 1), train_ls, 'epochs', 'loss',
                    range(1, num_epochs + 1), test_ls, ['train', 'test'])
    print('L2 norm of w:', w.norm().item())
```

指数衰减平均

➢ 定义均方误差

```
def squared_loss(y_hat, y):
    # 返回的是向量,注意：pytorch里的MSELoss并没有除以 2
    return ((y_hat - y.view(y_hat.size())) ** 2) / 2
```

➢ 定义随机梯度下降函数

```
def SGD(params, lr):
    for param in params:
        # 注意这里参数赋值用的是param.data
        param.data -= lr * param.grad
```

2.4 过拟合问题的常用方法—— L_2 范数正则化

➢ 定义训练函数

```
batch_size, num_epochs, lr = 1, 100, 0.003
net, loss = linear, squared_loss
#划分数据集
dataset = torch.utils.data.TensorDataset(train_features, train_labels)
train_iter = torch.utils.data.DataLoader(dataset, batch_size, shuffle=True)
#训练模型
def fit_and_plot(lambd):
    w, b = init_params()
    train_ls, test_ls = [], []
    for _ in range(num_epochs):
        for X, y in train_iter:
            # 添加了L2范数惩罚项
            l = loss(net(X, w, b), y) + lambd * l2_penalty(w)
            l = l.sum()

            if w.grad is not None:
                w.grad.data.zero_()
                b.grad.data.zero_()
            l.backward()
            SGD([w, b], lr)
        train_ls.append(loss(net(train_features, w, b), train_labels).mean().item())
        test_ls.append(loss(net(test_features, w, b), test_labels).mean().item())
    Draw_Loss_Curve(range(1, num_epochs + 1), train_ls, 'epochs', 'loss',
                    range(1, num_epochs + 1), test_ls, ['train', 'test'])
    print('L2 norm of w:', w.norm().item())
```

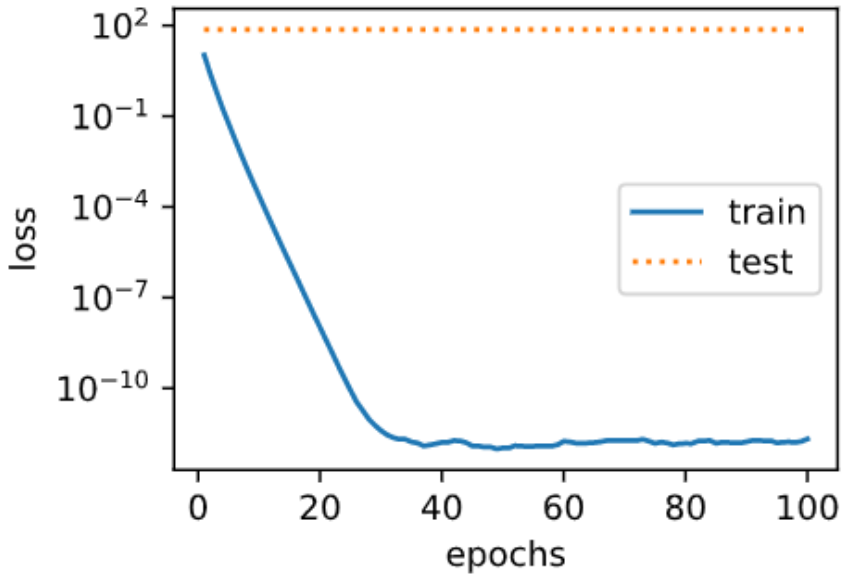
添加惩罚项，用lambd控制惩罚权重

2.4 过拟合问题的常用方法——L_2范数正则化

- $\lambda = 0$（即不使用L_2范数正则化）时的实验结果，出现了**过拟合**的现象。

```
fit_and_plot(lambd=0)
```

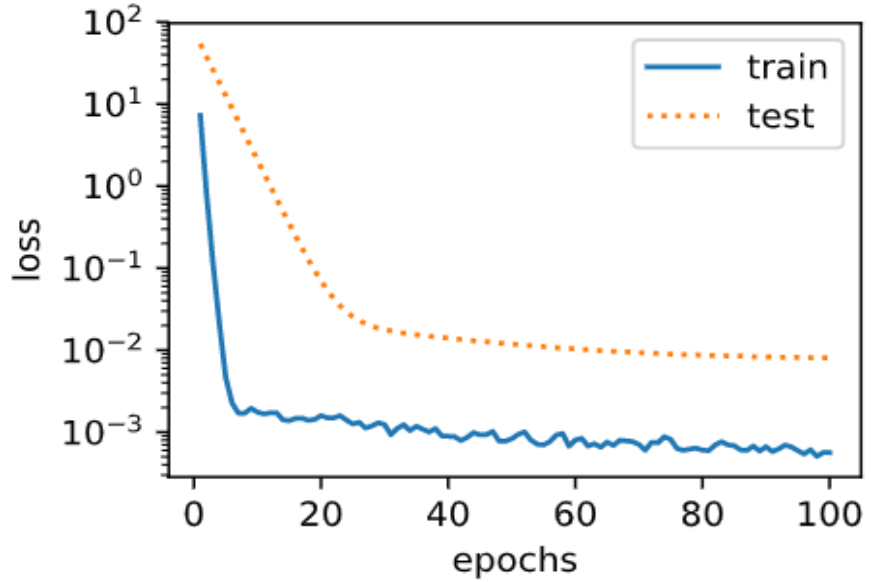
```
L2 norm of w: 12.559598922729492
```



- $\lambda = 3$（即使用L_2范数正则化）时的实验结果，一定程度地**缓解了过拟合**。同时可以看到参数L_2范数变小，**参数更接近0**。

```
fit_and_plot(lambd=3)
```

```
L2 norm of w: 0.04879558086395264
```



2.4 过拟合问题的常用方法—— L_2 范数正则化

■ 利用torch.optim的weight_decay参数实现L_2范数正则化

➢ 定义训练函数

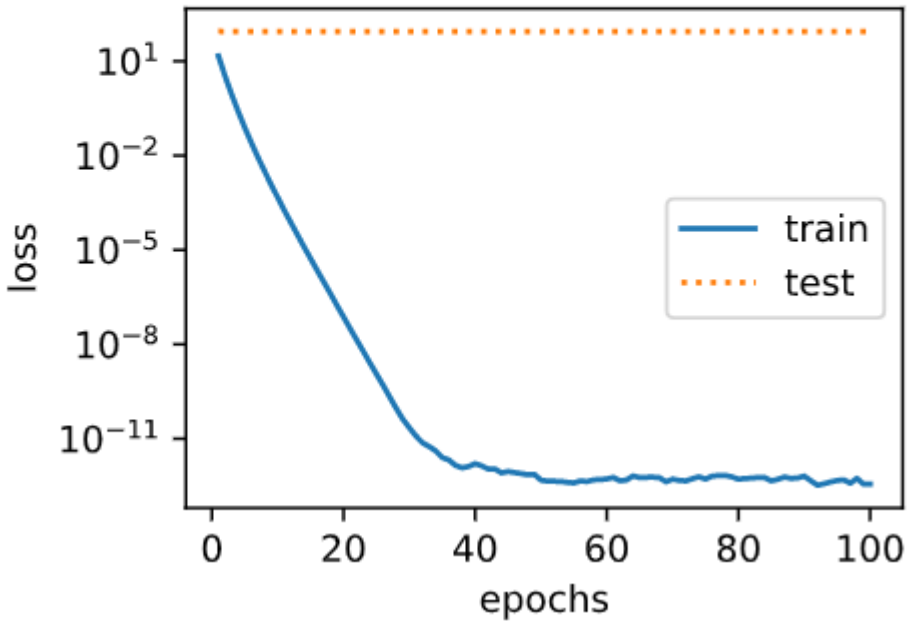
```
def fit_and_plot_pytorch(wd):
    # 对权重参数衰减。权重名称一般是以weight结尾
    net = nn.Linear(num_inputs, 1)
    nn.init.normal_(net.weight, mean=0, std=1)
    nn.init.normal_(net.bias, mean=0, std=1)
    # 使用weight_decay参数实现L2范数正则化
    optimizer_w = torch.optim.SGD(params=[net.weight], lr=lr, weight_decay=wd)
    optimizer_b = torch.optim.SGD(params=[net.bias], lr=lr)

    train_ls, test_ls = [], []
    for _ in range(num_epochs):
        for X, y in train_iter:
            l = loss(net(X), y).mean()
            optimizer_w.zero_grad()
            optimizer_b.zero_grad()

            l.backward()

            # 对两个optimizer实例分别调用step函数，从而分别更新权重和偏差
            optimizer_w.step()
            optimizer_b.step()
        train_ls.append(loss(net(train_features), train_labels).mean().item())
        test_ls.append(loss(net(test_features), test_labels).mean().item())
    Draw_Loss_Curve(range(1, num_epochs + 1), train_ls, 'epochs', 'loss',
                    range(1, num_epochs + 1), test_ls, ['train', 'test'])
    print('L2 norm of w:', net.weight.data.norm().item())
```

2.4 过拟合问题的常用方法——L_2范数正则化

- $\lambda = 0$（即不使用L_2范数正则化）时的实验结果，出现了过拟合的现象。

```
fit_and_plot_pytorch(0)
```

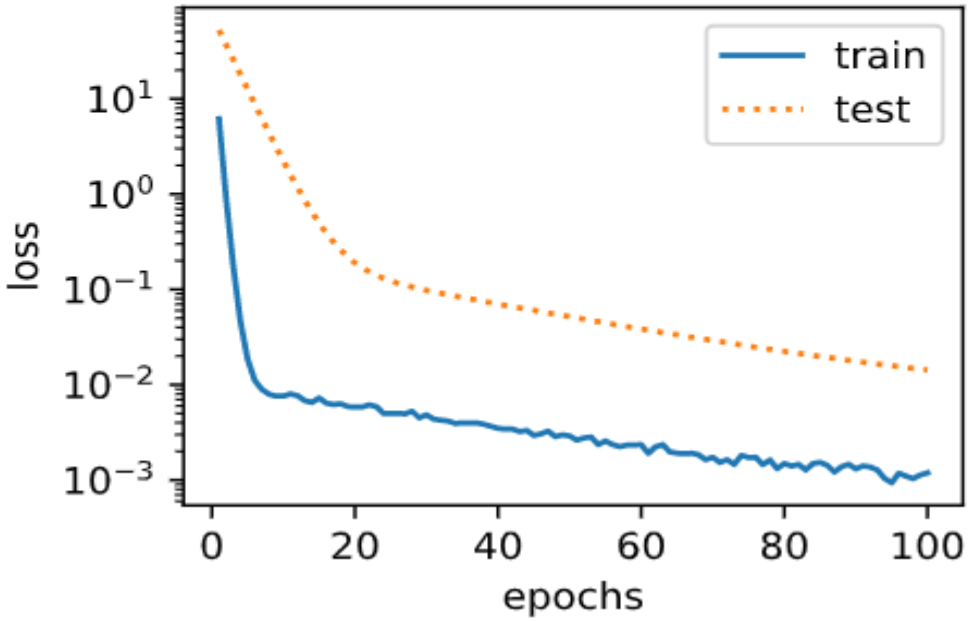
```
L2 norm of w: 13.343997955322266
```



- $\lambda = 3$（即使用L_2范数正则化）时的实验结果，一定程度的**缓解了过拟合**。同时可以看到参数L_2范数变小，**参数更接近0**。

```
fit_and_plot_pytorch(3)
```

```
L2 norm of w: 0.060846149921417236
```



2.5 应对过拟合问题的常用方法——Dropout

■ 手动实现dropout

以前馈神经网络为例，当使用dropout时，前馈神经网络隐藏层中的隐藏单元h_i有一定概率被丢弃掉。

- **设丢弃概率为p， 那么有p的概率h_i会被清零，有1−p的概率h_i会除以1−p做拉伸。由此定义进行dropout操作的函数**

```
def dropout(X, drop_prob):
    X = X.float()
    #检查丢弃概率是否在0到1之间
    assert 0 <= drop_prob <= 1
    keep_prob = 1 - drop_prob
    # 这种情况下把全部元素都丢弃
    if keep_prob == 0:
        return torch.zeros_like(X)
    #生成mask矩阵（向量）
    mask = (torch.rand(X.shape) < keep_prob).float()
    #按照mask进行对X进行变换
    return mask * X / keep_prob
```

- **初始化一个向量X，对X进行dropout，分别设置丢弃率为0、0.5、1，实验结果如下：**

```
X = torch.arange(10).view(2, 5)
print(dropout(X, 0),'\n')
print(dropout(X, 0.5),'\n')
print(dropout(X, 1))
```

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

tensor([[ 0.,  2.,  0.,  0.,  8.],
        [ 0., 12.,  0., 16., 18.]])

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

2.5 应对过拟合问题的常用方法——Dropout

- **定义模型参数（使用Fashion-MNIST数据集进行实验）**

```
num_inputs, num_outputs, num_hiddens1, num_hiddens2 = 784, 10, 256, 256

W1 = torch.tensor(np.random.normal(0, 0.01, size=(num_hiddens1, num_inputs)), dtype=torch.float, requires_grad=True)
b1 = torch.zeros(num_hiddens1, requires_grad=True)
W2 = torch.tensor(np.random.normal(0, 0.01, size=(num_hiddens2, num_hiddens1)), dtype=torch.float, requires_grad=True)
b2 = torch.zeros(num_hiddens2, requires_grad=True)
W3 = torch.tensor(np.random.normal(0, 0.01, size=(num_outputs, num_hiddens2)), dtype=torch.float, requires_grad=True)
b3 = torch.zeros(num_outputs, requires_grad=True)

params = [W1, b1, W2, b2, W3, b3]
```

- **定义使用dropout的网络模型，两个隐藏层的丢弃率分别为0.2和0.5**

```
drop_prob1, drop_prob2 = 0.2, 0.5

def net(X, is_training=True):
    X = X.view(-1, num_inputs)
    H1 = (torch.matmul(X, W1.t()) + b1).relu()
    if is_training:  # 如果是在训练则使用dropout
        H1 = dropout(H1, drop_prob1)  # 在第一层全连接后进行dropout
    H2 = (torch.matmul(H1, W2.t()) + b2).relu()
    if is_training:
        H2 = dropout(H2, drop_prob2)  # 在第二层全连接后进行dropout
    return torch.matmul(H2, W3.t()) + b3
```

2.5 应对过拟合问题的常用方法——Dropout

- **定义计算准确率的函数**

```
def evaluate_accuracy(data_iter, net):
    acc_sum, n = 0.0, 0
    for X, y in data_iter:
        acc_sum += (net(X, is_training=False).argmax(dim=1) == y).float().sum().item()
        n += y.shape[0]
    return acc_sum / n
```

测试时不使用dropout

- **训练模型结果**

```
num_epochs, lr, batch_size = 5, 0.1, 128
loss = torch.nn.CrossEntropyLoss()
train(net, train_iter, test_iter, loss, num_epochs, batch_size, params, lr, None)
```

```
epoch 1, loss 0.0105, train acc 0.492, test acc 0.689
epoch 2, loss 0.0052, train acc 0.759, test acc 0.802
epoch 3, loss 0.0042, train acc 0.810, test acc 0.827
epoch 4, loss 0.0037, train acc 0.831, test acc 0.825
epoch 5, loss 0.0034, train acc 0.844, test acc 0.849
```

2.5 应对过拟合问题的常用方法——Dropout

■ 利用torch.nn.Dropout层实现dropout

➢ 定义模型

```
class FlattenLayer(torch.nn.Module):
    def __init__(self):
        super(FlattenLayer, self).__init__()
    def forward(self, x):
        return x.view(x.shape[0], -1)
```

```
net_pytorch = nn.Sequential(
        FlattenLayer(),
        nn.Linear(num_inputs, num_hiddens1),
        nn.ReLU(),
        nn.Dropout(drop_prob1),
        nn.Linear(num_hiddens1, num_hiddens2),
        nn.ReLU(),
        nn.Dropout(drop_prob2),
        nn.Linear(num_hiddens2, 10)
        )

for param in net_pytorch.parameters():
    nn.init.normal_(param, mean=0, std=0.01)
```

2.5 应对过拟合问题的常用方法——Dropout

- **定义计算准确率的函数（eval()和train()来切换模型的状态）**

```
def evaluate_accuracy(data_iter, net):
    acc_sum, n = 0.0, 0
    for X, y in data_iter:
        if isinstance(net, torch.nn.Module):
            net.eval() # 评估模式, 不使用dropout
            acc_sum += (net(X).argmax(dim=1) == y).float().sum().item()
            net.train() # 改回训练模式
        n += y.shape[0]
    return acc_sum / n
```

先用eval()函数切换模式，再进行测试

- **实验结果**

```
optimizer = torch.optim.SGD(net_pytorch.parameters(), lr=0.1)
train(net_pytorch, train_iter, test_iter, loss, num_epochs, batch_size, None, None, optimizer)
```

```
epoch 1, loss 0.0104, train acc 0.490, test acc 0.746
epoch 2, loss 0.0051, train acc 0.764, test acc 0.801
epoch 3, loss 0.0042, train acc 0.809, test acc 0.825
epoch 4, loss 0.0037, train acc 0.830, test acc 0.840
epoch 5, loss 0.0034, train acc 0.843, test acc 0.837
```

2.6 不同的优化算法-RMSprop算法

➢ RMSprop算法

思想：使用梯度平方的“指数衰减移动平均”修正学习率
参数取值：γ可以设为0.9

$$s_t \leftarrow \gamma s_{t-1} + (1-\gamma) g_t \odot g_t$$

$$\theta_t \leftarrow \theta_{t-1} - \frac{lr}{\sqrt{s_t + \epsilon}} \odot g_t$$

➢ RMSprop算法的手动实现

初始化s_0

对每一个参数进行RMSprop法

```
def init_rmsprop_states(params):
    s_w1,s_b1,s_w2,s_b2 = torch.zeros(params[0].shape), \
                          torch.zeros(params[1].shape), \
                          torch.zeros(params[2].shape), \
                          torch.zeros(params[3].shape)
    return (s_w1, s_b1, s_w2, s_b2)

def rmsprop(params, states, lr,gamma):
    gamma, eps = gamma, 1e-6
    for p, s in zip(params, states):
        with torch.no_grad():
            s[:] = gamma * s + (1 - gamma) * torch.square(p.grad)
            p[:] -= lr * p.grad / torch.sqrt(s + eps)
        p.grad.data.zero_()
```

```
states = init_rmsprop_states(net.params)

for epoch in range(num_epochs):
    train_l_sum, train_acc_sum, n, c = 0.0, 0.0, 0, 0
    for X, y in train_iter:
        y_hat = net.forward(X)
        l = loss_func(y_hat, y).sum()
        l.backward()
        optimizer(net.params,states,lr,gamma)
```

➢ RMSprop算法的torch.nn实现

```
optimizer = torch.optim.RMSprop(net.parameters(), lr=lr,alpha=alpha)
```

➢ 提出RMSprop的原教案 https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

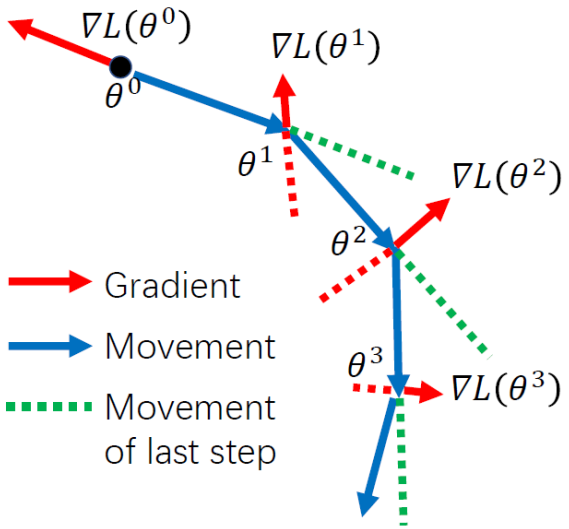
2.6 不同的优化算法-动量(momentum)法

momentum算法

思想：每次迭代，计算负梯度的“加权移动平均”作为参数的更新方向
参数取值：γ可以设为0.5

$$v_t \leftarrow \gamma v_{t-1} - lr g_t$$

$$\theta_t \leftarrow \theta_{t-1} + v_t$$



momentum算法的手动实现

初始化v_0

对每一个参数进行动量法

```
def init_momentum_states(params):
    v_w1,v_b1,v_w2,v_b2 = torch.zeros(params[0].shape), \
                          torch.zeros(params[1].shape), \
                          torch.zeros(params[2].shape), \
                          torch.zeros(params[3].shape)
    return (v_w1,v_b1,v_w2,v_b2)

def sgd_momentum(params, states,  lr, momentum):
    for p, v in zip(params, states):
        with torch.no_grad():
            v[:] = momentum * v - p.grad
            p[:] += lr * v
        p.grad.data.zero_()
```

```
states = init_rmsprop_states(net.params)

for epoch in range(num_epochs):
    train_l_sum, train_acc_sum, n, c = 0.0, 0.0, 0, 0
    for X, y in train_iter:
        y_hat = net.forward(X)
        l = loss_func(y_hat, y).sum()
        l.backward()
        optimizer(net.params,states,lr,momentum)
        for param in net.params:
            param.grad.data.zero_()

        train_l_sum += l.item()
        train_acc_sum += (y_hat.argmax(dim=1) == y).sum().item()
        n += y.shape[0]
        c += 1
```

momentum算法的torch.nn实现

```
optimizer = torch.optim.SGD(net.parameters(), lr=lr,momentum=momentum)
```

- 为什么动量法有效 https://distill.pub/2017/momentum/

2.6 不同的优化算法-Adam算法

➢ Adam算法

思想：同时使用“指数衰减移动平均”和“加权移动平均”
参数取值：β_1可以设为0.9，β_2设为0.999

$$v_t \leftarrow \beta_1 v_{t-1} + (1-\beta_1) g_t$$

$$s_t \leftarrow \beta_2 s_{t-1} + (1-\beta_2) g_t \odot g_t$$

$$\hat{v}_t \leftarrow \frac{v_t}{1-\beta_1^t} \qquad \hat{s}_t \leftarrow \frac{s_f}{1-\beta_2^t}$$

$$\theta_t \leftarrow \theta_{t-1} - \frac{\eta \hat{v}_t}{\sqrt{\hat{s}_t + \epsilon}}$$

初始化 s_0，v_0

对每一个参数进行Adam法

➢ Adam算法的手动实现

```
def init_adam_states(params):
    v_w1, v_b1,v_w2, v_b2 = torch.zeros(params[0].shape), \
                            torch.zeros(params[1].shape), \
                            torch.zeros(params[2].shape), \
                            torch.zeros(params[3].shape)
    s_w1, s_b1,s_w2, s_b2 = torch.zeros(params[0].shape), \
                            torch.zeros(params[1].shape), \
                            torch.zeros(params[2].shape), \
                            torch.zeros(params[3].shape)
    return ((v_w1, s_w1), (v_b1, s_b1),(v_w2, s_w2), (v_b2, s_b2))

def adam(params, states,lr,t):
    beta1, beta2, eps = 0.9, 0.999, 1e-6
    for p, (v, s) in zip(params, states):
        with torch.no_grad():
            [                                                    ]
            p.grad.data.zero_()
    t += 1
```

```
states = init_adam_states(net.params)

for epoch in range(num_epochs):
    train_l_sum, train_acc_sum, n, c = 0.0, 0.0, 0, 0
    for X, y in train_iter:
        y_hat = net.forward(X)
        l = loss_func(y_hat, y).sum()
        l.backward()
        optimizer(net.params,states,lr,t)
```

➢ Adam算法的torch.nn实现

```
optimizer = torch.optim.Adam(net.parameters(), lr=lr)
```

➢ Adam原论文 https://arxiv.org/abs/1412.6980

目录

1. 基本概念

- 前馈神经网络的复习
- 优化器的使用

2. 模型调优

- 交叉验证
- 过拟合&欠拟合
- 探究导致过拟合、欠拟合的因素
- 过拟合解决办法：正则化、dropout
- 不同的优化算法

3. 实验要求

- 数据集介绍
- 多分类任务数据集下载和读取
- 课程实验要求

3.1 数据集介绍——与实验二相同

■ 手动生成回归任务的数据集，要求：

- 生成单个数据集。
- 数据集的大小为10000且训练集大小为7000，测试集大小为3000。
- 数据集的样本特征维度p为500，且服从如下的高维线性函数： $y = 0.028 + \sum_{i=1}^{p} 0.0056 x_i + \epsilon$

■ 手动生成二分类任务的数据集，要求：

- 共生成两个数据集。
- 两个数据集的大小均为10000且训练集大小为7000，测试集大小为3000。
- 两个数据集的样本特征x的维度均为200，且分别服从均值互为相反数且方差相同的正态分布。
- 两个数据集的样本标签分别为0和1。

■ MNIST手写体数据集介绍：

- 该数据集包含60,000个用于训练的图像样本和10,000个用于测试的图像样本。
- 图像是固定大小(28x28像素)，其值为0到1。为每个图像都被平展并转换为784(28 * 28)个特征的一维numpy数组。

3.2 多分类任务数据集下载和读取

MNIST数据集下载和读取：

```
#下载MNIST手写数字数据集
train_dataset = torchvision.datasets.MNIST(root="./Datasets/MNIST", train=True, transform= transforms.ToTensor(),download=True)
test_dataset = torchvision.datasets.MNIST(root="./Datasets/MNIST", train=False, transform= transforms.ToTensor())
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Processing...
Done!
```

```
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=32, shuffle=False)
```

```
for X, y in train_loader:
    print(X.shape,y.shape)
    break
```

```
torch.Size([32, 1, 28, 28]) torch.Size([32])
```

3.3 课程实验要求

- **任务**

（1）在多分类任务实验中分别手动实现和用torch.nn实现dropout

- 探究不同丢弃率对实验结果的影响（可用loss曲线进行展示）

（2）在多分类任务实验中分别手动实现和用torch.nn实现L_2正则化

- 探究惩罚项的权重对实验结果的影响（可用loss曲线进行展示）

（3）在多分类任务实验中实现momentum、rmsprop、adam优化器

- 在手动实现多分类的任务中手动实现三种优化算法，并补全Adam中计算部分的内容
- 在torch.nn实现多分类的任务中使用torch.nn实现各种优化器，并对比其效果

（4）对多分类任务实验中实现早停机制，并在测试集上测试

- 选择上述实验中效果最好的组合，手动将训练数据划分为训练集和验证集，实现早停机制，并在测试集上进行测试。训练集：验证集=8：2，早停轮数为5.

- **截止时间**

- 2022年11月24日23:55前（即下一次实验课之前）

3.3 课程实验要求

提交作业要求

两种提交实验报告的方式（选择一种提交）：

- word版报告：
  - 根据实验报告中的相应提示内容，完成相应实验报告的部分，展示实验结果时需要图文并茂，并进行相应的分析。
  - 在word文件中需要粘贴关键代码，尽量将代码进行格式化，不要直接截图。
    - 代码格式化网站 http://www.planetb.ca/syntax-highlight-word 或 http://codeinword.com/
  - 同时要求提交实验的完整代码，不同的题目放在不同的.ipynb文件中，需要有相应的注释。
- jupyter版报告：
  - 根据实验报告中的相应提示内容，完成相应实验报告的部分，需要保留相应的代码运行结果、图等内容。
  - 在给定的jupyter模板中提供了编写代码的部分，代码直接编写在相应的块中即可，需要有相应的注释。
  - 实验代码全部写在jupyter中，无需另外提交其他.ipynb文件