由 Xnip 截图

权重衰减（Weight Decay）

1、权重衰减（Weight Decay）是一种常见的正则化方法，用于防止机器学习模型过度拟合训练数据，提高模型泛化性能。

权重衰减通常通过向损失函数中添加一个正则化项来实现。这个正则化项是模型 所有权重的平方和 乘以一个 正则化系数（lambda）。

$$L = loss(y, y\_pred) + lambda * sum(w^2)$$

其中，loss(y, y_pred)是模型的预测损失，sum(w^2)是所有权重的平方和，lambda是正则化系数。

通过添加这个正则化项，优化器在更新权重时不仅要最小化预测损失，还要让权重的平方和最小化。这使得模型在学习过程中更加趋向于选择较小的权重值，从而减少模型的过拟合风险。

2、回顾3.1节中的线性回归例子。 我们的损失由下式给出：

$$L(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}\left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)}\right)^2$$

在实践中，通常使用L2正则化:

$$L(\mathbf{w}, b) + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

(除以2--当我们取一个二次函数的导数时抵消)

3、权重衰减为我们提供了一种连续的机制来调整函数的复杂度。 较小的lambda值对应较少约束的权重w，而较大的lambda对w的约束更大。

是否对相应的偏置b^2进行惩罚在不同的实践中会有所不同， 在神经网络的不同层中也会有所不同。 通常，网络输出层的偏置项不会被正则化。

4、L1正则化和L2正则化

L2正则化性模型构成经典的岭回归（ridge regression）算法。使用L2范数的一个原因是它对权重向量的大分量施加了巨大的惩罚。 这使得我们的学习算法偏向于在大量特征上均匀分布权重的模型。在实践中，这可能使它们对单个变量中的观测误差更为稳定。

L1正则化线性回归是统计学中类似的基本模型， 通常被称为套索回归（lasso regression）。 L1惩罚会导致模型将权重集中在一小部分特征上， 而将其他权重清除为零。这称为特征选择（feature selection）

In [1]:
```
pip install d2l
```

```
Requirement already satisfied: requests==2.25.1 in /Users/palekiller/.local/lib/python3.9/site-packages (from d2
l) (2.25.1)
Requirement already satisfied: nbconvert in /Users/palekiller/.local/lib/python3.9/site-packages (from jupyter==
1.0.0->d2l) (7.2.10)
Requirement already satisfied: jupyter-console in /Users/palekiller/.local/lib/python3.9/site-packages (from jupy
ter==1.0.0->d2l) (6.6.2)
Requirement already satisfied: notebook in /Users/palekiller/opt/anaconda3/envs/DeepLearning/lib/python3.9/site-p
ackages (from jupyter==1.0.0->d2l) (6.5.2)
Requirement already satisfied: ipywidgets in /Users/palekiller/.local/lib/python3.9/site-packages (from jupyter==
1.0.0->d2l) (8.0.4)
Requirement already satisfied: qtconsole in /Users/palekiller/.local/lib/python3.9/site-packages (from jupyter==
1.0.0->d2l) (5.4.0)
Requirement already satisfied: ipykernel in /Users/palekiller/opt/anaconda3/envs/DeepLearning/lib/python3.9/site-
packages (from jupyter==1.0.0->d2l) (6.19.2)
Requirement already satisfied: fonttools>=4.22.0 in /Users/palekiller/.local/lib/python3.9/site-packages (from ma
tplotlib==3.5.1->d2l) (4.38.0)
Requirement already satisfied: packaging>=20.0 in /Users/palekiller/opt/anaconda3/envs/DeepLearning/lib/python3.9
/site-packages (from matplotlib==3.5.1->d2l) (22.0)
Requirement already satisfied: pillow>=6.2.0 in /Users/palekiller/opt/anaconda3/envs/DeepLearning/lib/python3.9/s
```

In [2]:
```
%matplotlib inline
import torch
from torch import nn
from d2l import torch as d2l
```

生成一些数据

In [3]:
```
n_train, n_test, num_inputs, batch_size = 20, 100, 200, 5
# 训练集大小 n_train，测试集大小 n_test，输入特征数量 num_inputs，批次大小 batch_size。

true_w, true_b = torch.ones((num_inputs, 1)) * 0.01, 0.05
# 真实权重向量 true_w，偏差 true_b，并将它们初始化为一个全为 0.01 和 0.05 的张量。这些值将用于生成合成数据集。

# synthetic_data()生成大小为 n_train 的训练集和大小为 n_test 的测试集。使用上面定义的真实权重向量 true_w 和偏差 true_b 生成服从正
# synthetic_data()返回两个张量，n*d 的特征矩阵features和一个 n*1 的标签向量labels
# load_array()将训练集和测试集的数据加载到数据迭代器中，以便于后续训练和测试模型。
# 将数据集按照批次大小 batch_size 划分为多个小批次，以便于使用随机梯度下降等优化算法进行训练和测试。
train_data = d2l.synthetic_data(true_w, true_b, n_train)
train_iter = d2l.load_array(train_data, batch_size)

test_data = d2l.synthetic_data(true_w, true_b, n_test)
test_iter = d2l.load_array(test_data, batch_size, is_train=False)
```

初始化模型参数

In [4]:
```
def init_params():

    w = torch.normal(0, 1, size=(num_inputs, 1), requires_grad=True)
    # w是神经网络的权重，使用从均值为0，标准差为1的正态分布中抽取的随机值进行初始化。是一个具有num_inputs行和一列的张量。

    b = torch.zeros(1, requires_grad=True)
    # b是神经网络的偏置，使用零进行初始化。是一个具有一行和一列的张量，

    # w和b都将requires_grad设置为True，这意味着它们将使用自动微分在训练过程中进行更新。

    return [w, b]
```

定义L2范数惩罚

L2正则化的惩罚项：

$$\frac{1}{2}\|\mathbf{w}\|^2$$

In [5]:
```
def l2_penalty(w):

    # 函数接收一个张量w作为输入，计算出w的平方和并除以2，作为L2正则化的惩罚项

    return torch.sum(w.pow(2)) / 2  # w.pow(2)：对w中的每个元素进行平方。
```

定义训练代码实现

In [6]:
```
def train(lambd):

    # 实现了一个带L2正则化的线性回归模型的训练，输入为正则化系数lambd

    w, b = init_params()  # 初始化模型参数，权重w和偏置b
    net, loss = lambda X: d2l.linreg(X, w, b), d2l.squared_loss  # 定义模型和损失函数
    # lambda X: d2l.linreg(X, w, b)：一个匿名函数，定义了一个线性回归模型。它接受一个输入参数X，并返回一个基于输入特征X、权重w和偏置
    # d2l.squared_loss：平方损失函数
    num_epochs, lr = 100, 0.003  # 定义超参数，num_epochs训练的总epoch数，lr学习率

    # 定义可视化动画
    animator = d2l.Animator(xlabel='epochs', ylabel='loss', yscale='log',
                            xlim=[5, num_epochs], legend=['train', 'test'])
    # yscale: y轴刻度的缩放方式为对数刻度，xlim: x轴的显示范围，legend: 图例标签

    # 开始训练
    for epoch in range(num_epochs):
        for X, y in train_iter:

            # 计算模型在当前batch上的损失，并增加L2正则化项 l2_penalty(w)，广播机制使l2_penalty(w)成为一个长度为batch_size的向量
            l = loss(net(X), y) + lambd * l2_penalty(w)
            # 对损失函数求梯度
            l.sum().backward()
            # 更新模型参数
            d2l.sgd([w, b], lr, batch_size)  # sgd()小批量随机梯度下降法（见补充）

        # 每隔一定epoch，可视化一次训练过程
        if (epoch + 1) % 5 == 0:
            # 绘制训练和测试损失的动画曲线
            animator.add(epoch + 1, (d2l.evaluate_loss(net, train_iter, loss),
                                     d2l.evaluate_loss(net, test_iter, loss)))
            # 当前迭代次数epoch+1，d2l.evaluate_loss()计算训练集和测试集上的损失值

    # 输出模型参数w的L2范数
    print('w的L2范数是：', torch.norm(w).item())
```

补充：小批量随机梯度下降法sgd

sgd每次迭代更新模型参数的过程如下：

1、随机选择一个小批量的训练样本。

2、根据当前模型参数，通过前向计算得到模型的预测值。

3、计算预测值与真实值之间的误差，并根据误差计算损失函数的梯度。

4、沿着梯度反方向更新模型参数

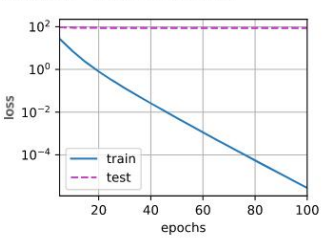
$$w = w - lr * w.grad/batch\_size$$

$$b = b - lr * b.grad/batch\_size$$

5、清除参数的梯度，以便下次迭代使用。

In [7]:
```
train(lambd=0)
```

```
w的L2范数是： 13.119410514831543
```

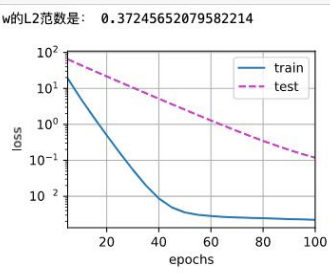


lambd = 0 （没有使用L2范数惩罚）

此时训练误差有了减少，但测试误差没有减少， 这意味着出现了严重的过拟合。

In [8]:
```
train(lambd=3)
```

```
w的L2范数是： 0.37245652079582214
```



lambd=3

此时训练误差增大，但测试误差减小。 没有过拟合现象，这正是我们期望从正则化中得到的效果。

lambd 取值

lambd是L2范数惩罚项的系数，它的取值会影响模型在训练集和测试集上的准确率和损失值。当lambda越大时，模型对参数的惩罚越强，模型会更加倾向于选择那些参数较小的模型。

当lambd取值较小时（如0），模型的训练准确率和测试准确率可能会较高，但可能会出现过拟合的情况：模型在训练集上的准确率很高，但在测试集上的表现较差。

当lambd取值较大时（如3），模型的训练准确率和测试准确率可能会较低，但模型的泛化能力会更好：模型能够更好地适应未见过的数据。

简洁实现

In [9]:
```
def train_concise(wd):

    # 一个更加简洁的实现带权重衰减（L2范数惩罚）的线性回归的函数，相比于之前的实现，它使用了PyTorch中的高级API

    net = nn.Sequential(nn.Linear(num_inputs, 1))  # 只有一个全连接层的神经网络net

    for param in net.parameters():
        param.data.normal_()  # 对神经网络的权重参数进行初始化，使其服从标准正态分布

    loss = nn.MSELoss(reduction='none')  # 使用均方误差MSE作为损失函数，并将reduction参数设置为none，以便后续计算L2范数惩罚项

    num_epochs, lr = 100, 0.003  # 定义超参数，num_epochs训练的总epoch数，lr学习率

    # 直接调用PyTorch中的优化器SGD，偏置参数 net[0].bias和 net[0].weight需要被优化器更新
    trainer = torch.optim.SGD([
        {"params":net[0].weight,'weight_decay': wd},
        {"params":net[0].bias}], lr=lr)  # 'weight_decay': wd ——权重没有衰减

    # 定义可视化动画
    animator = d2l.Animator(xlabel='epochs', ylabel='loss', yscale='log',
                            xlim=[5, num_epochs], legend=['train', 'test'])

    # 开始训练
    for epoch in range(num_epochs):
        for X, y in train_iter:

            trainer.zero_grad()  # 清除之前计算的梯度
            l = loss(net(X), y)  # 当前批次的损失函数
            l.mean().backward()  # 当前批次的平均损失函数，并进行反向传播计算梯度
            trainer.step()  # 根据计算的梯度更新参数

        # 每隔一定epoch，可视化一次训练过程
        if (epoch + 1) % 5 == 0:
            animator.add(epoch + 1,
                         (d2l.evaluate_loss(net, train_iter, loss),
                          d2l.evaluate_loss(net, test_iter, loss)))

    # 输出模型参数w的L2范数
    print('w的L2范数：', net[0].weight.norm().item())
```

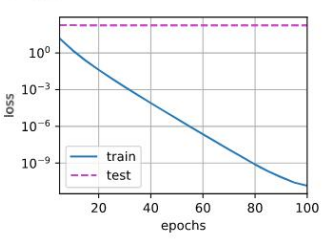
补充：权重衰减

L2正则化将模型的权重参数平方和添加到损失函数中（权重参数w）

可以应用权重衰减（weight decay）来对权重参数w进行正则化惩罚：即每次更新权重参数w时，将w减去一个衰减常数乘以它本身。这样做可以有效地防止模型过度拟合。

In [10]:
```
train_concise(0)
```

```
w的L2范数： 13.793651580810547
```



In [11]:
```
train_concise(3)
```

```
w的L2范数： 0.5285437107086182
```