# Linear regression

线性回归可以近似理解为收集一些数据点来决定一些参数值（如权重和偏差），这些被称为训练数据，通常越多越好。

线性回归可以理解为单层神经网络

# Optimization methods

## Gradient descent

梯度下降最简单的用法是计算损失函数（数据集中所有样本的损失均值） 关于模型参数的导数（在这里也可以称为梯度）。 但实际中的执行可能会非常慢：因为在每一次更新参数之前，我们必须遍历整个数据集。

# Traditional linear regression

根据带有噪声的线性模型构造一个人造数据集。

我们使用线性模型参数

$$\mathbf{w} = [2, -3.4]^\top$$

和噪声项$\epsilon$生成数据集及其标签：

$$\mathbf{y} = \mathbf{X}\mathbf{w} + b + \epsilon$$

```python
'''
Traditional linear regression
idea:
First, set the ture weight and bais, and generated 1000 data prints.
Next,assuming we don't know these true values,
we train the model to infer the weights and bias that are closest to the true values.
'''
def date_generator(w,b,num_exaples):
    X = torch.normal(0,1,(num_exaples,len(w)))
    print('X:',X)
    y = torch.matmul(X,w) + b
    y += torch.normal(0,0.01,y.shape)
    return X, y.reshape((-1,1))

true_w = torch.tensor([2,-3.4])
true_b = 4.2
features, labels = date_generator(true_w, true_b, 1000)
"""
 使用真实的权重 true_w 和偏置 true_b 生成 1000 个样本的数据
 synthetic_data 函数返回特征矩阵 features 和标签向量 labels
 features 的形状是 (1000, len(true_w))，其中每一行是一个样本的特征
 labels 的形状是 (1000, 1)，每一行是对应样本的标签
"""
print('features:',features[0],'\nlabel:',labels[0])

d2l.set_figsize()
d2l.plt.scatter(features[:,1].detach().numpy(),labels.detach().numpy(),1)
d2l.plt.show()
```

```python
def data_iter(batch_size,features,labels):
    num_examples = len(features)
    indices = list(range(num_examples))
    random.shuffle(indices)
    for i in range(0, num_examples, batch_size):
        batch_indices = torch.tensor(indices[i:min(i+batch_size,num_examples)])
        # 当i+batch_size超出时，取num_examples
        yield features[batch_indices], labels[batch_indices]
        # 获得随即顺序的特征，及对应的标签

batch_size = 10

for X,y in data_iter(batch_size, features, labels):
    print(X, '\n', y) # 取一个批次后，就break跳出了
    break

# 定义初始化模型参数
train_w = torch.normal(0,0.01,size=(2,1),requires_grad=True)
train_b = torch.zeros(1, requires_grad=True)

# 定义模型
def linreg(X,w,b):
    return torch.matmul(X,w)+b

def squared_loss(y_hat,y):
    return (y_hat - y.reshape(y_hat.shape))**2/2 # 将y统一成与y_hat一样同尺寸

def sgd(params,lr,batch_size):
    with torch.no_grad(): # 不要产生梯度计算，减少内存消耗
        for param in params:
            param -= lr * param.grad / batch_size
            # 每个参数进行更新，损失函数没有求均值，所以这里除以 batch_size 求了均值。
            # 由于乘法的线性关系，这里除以放在loss的除以是等价的。
            param.grad.zero_()

lr = 0.03
num_epochs = 3
net = linreg
loss = squared_loss

for epoch in range(num_epochs):
    for X,y in data_iter(batch_size,features,labels):
        l = loss(net(X, train_w, train_b), y) # X和y的小批量损失
        # 因为l是形状是(batch_size,1)，而不是一个标量。l中所有元素被加到一起
        # 并以此计算关于[w,b]的梯度
        l.sum().backward()
        sgd([train_w, train_b], lr, batch_size) # 使用参数的梯度更新参数
    with torch.no_grad():
        train_l = loss(net(features, train_w, train_b), labels)
        print(f'epoch{epoch+1},loss{float(train_l.mean()):f}')

    # 比较真实参数和通过训练学到的参数来评估训练的成功程度

print(train_w)
print(train_b)
print(f'w的估计误差：{true_w - train_w.reshape(true_w.shape)}')
print(f'b的估计误差：{true_b - train_b}')
```

## Pytorch linear regression

```
true_w = torch.tensor([2,-3.4])
true_b = 4.2
features, labels = d2l.synthetic_data(true_w,true_b,1000)
# Generate synthetic dataset by library
def data_generator_pytorch(w,b,num_exaples):
    X = torch.normal(0,1,(num_exaples,len(w)))
    y = torch.matmul(X,w) + b
    y += torch.normal(0,0.01,y.shape)
    return X, y.reshape((-1,1))

true_w = torch.tensor([2,-3.4])
true_b = 4.2
features, labels = data_generator_pytorch(true_w, true_b, 1000)

def load_array(data_arrays, batch_size, is_train=True):
    dataset = data.TensorDataset(*data_arrays)
    return data.DataLoader(dataset, batch_size, shuffle=is_train)

net = nn.Sequential(nn.Linear(2,1))

net[0].weight.data.normal_(0,0.01)
# Replace the data values in the weight variable with values
net[0].bias.data.fill_(0)

loss = nn.MSELoss()
trainer = torch.optim.SGD(net.parameters(), lr=0.03) # SGD instance

num_epochs = 5
for epoch in range(num_epochs):
    for X, y in load_array((features, labels), batch_size=10):
        l = loss(net(X), y)
        trainer.zero_grad()
        l.backward()
        trainer.step()
    l = loss(net(features), labels)
    print(f'epoch {epoch+1}, loss {float(l.mean()):f}')
```