丢弃法

例如一个个单隐藏层的多层感知机中，隐藏层的计算表达式为：

$$h_i = \phi\left(x_1 w_{1i} + x_2 w_{2i} + x_3 w_{3i} + x_4 w_{4i} + b_i\right),$$

这里 ϕ 是激活函数， $x_1, \ldots, x_4$ 是输入，隐藏单元 i 的权重参数为 $w_{1i}, \ldots, w_{4i}$ ，偏差参数为 b_i 。

当对该隐藏层使用丢弃法时，该层的隐藏单元将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。

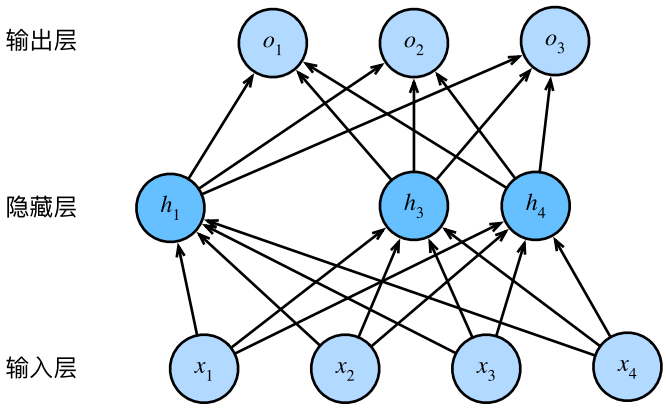
*丢弃概率是丢弃法的超参数

设随机变量ξ_i为0和1的概率分别为p和1−p。使用丢弃法时我们计算新的隐藏单元 h'_i

$$h'_i = \frac{\xi_i}{1-p} h_i.$$

由于 E(ξi)=1−p ， 因此

$$E(h'_i) = \frac{E(\xi_i)}{1-p} h_i = h_i.$$



从零开始实现

导入必要的包

```
import d2lzh as d2l
from mxnet import nd,autograd,init,gluon
from mxnet.gluon import data as gdata,loss as gloss,nn
```

定义丢弃函数

```
def dropout(X,drop_prob):
    assert 0 <= drop_prob <= 1
    keep_prob = 1 - drop_prob
    if keep_prob == 0:#这种情况全部丢弃
        return X.zeros_like()
    mask = nd.random.uniform(0,1,X.shape) < keep_prob
    return mask * X / keep_prob
```

测试dropout函数

```
X = nd.arange(16).reshape((2,8))
X
```

```
[[ 0.  1.  2.  3.  4.  5.  6.  7.]
 [ 8.  9. 10. 11. 12. 13. 14. 15.]]
<NDArray 2x8 @cpu(0)>
```

```
dropout(X,0),dropout(X,1)
```

```
(
 [[ 0.  1.  2.  3.  4.  5.  6.  7.]
  [ 8.  9. 10. 11. 12. 13. 14. 15.]]
 <NDArray 2x8 @cpu(0)>,
 [[0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 0.]]
 <NDArray 2x8 @cpu(0)>)
```

```
dropout(X,0.5)
```

```
[[ 0.  2.  4.  6.  0. 10.  0. 14.]
 [ 0. 18.  0.  0.  0. 26. 28.  0.]]
<NDArray 2x8 @cpu(0)>
```

定义模型参数

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

W1 = nd.random.normal(scale=0.01,shape=(num_inputs,num_hiddens1))
b1 = nd.zeros(shape=num_hiddens1)
W2 = nd.random.normal(scale=0.01,shape=(num_hiddens1,num_hiddens2))
b2 = nd.zeros(shape=num_hiddens2)
W3 = nd.random.normal(scale=0.01,shape=(num_hiddens2,num_outputs))
b3 = nd.zeros(shape=num_outputs)

params = [W1,b1,W2,b2,W3,b3]
for param in params:
    param.attach_grad()
```

定义模型

```
drop_prob1,drop_prob2 = 0.2,0.5

def net(X):
    X = X.reshape((-1,num_inputs))
    H1 = (nd.dot(X,W1) + b1).relu()
    if autograd.is_training():
        H1 = dropout(H1,drop_prob1)
    H2 = (nd.dot(H1,W2) + b2).relu()
    if autograd.is_training():
        H2 = dropout(H2,drop_prob2)
    return nd.dot(H2,W3) + b3
```

训练和测试模型

```
num_epochs,lr,batch_size = 5,0.5,256

loss = gloss.SoftmaxCrossEntropyLoss()

train_iter,test_iter = d2l.load_data_fashion_mnist(batch_size)

d2l.train_ch3(net, train_iter, test_iter, loss, num_epochs, batch_size,
              params, lr)
```

```
epoch 1, loss 1.1829, train acc 0.544, test acc 0.752
epoch 2, loss 0.6024, train acc 0.775, test acc 0.836
epoch 3, loss 0.5026, train acc 0.816, test acc 0.843
epoch 4, loss 0.4528, train acc 0.835, test acc 0.860
epoch 5, loss 0.4230, train acc 0.845, test acc 0.868
```

简洁实现

```
net = nn.Sequential()
net.add(nn.Dense(256,activation='relu'),nn.Dropout(drop_prob1),
       nn.Dense(256,activation='relu'),nn.Dropout(drop_prob2),
       nn.Dense(10))
net.initialize(init.Normal(sigma=0.01))
```

测试模型

```
trainer = gluon.Trainer(net.collect_params(),'sgd',{'learning_rate':lr})
d2l.train_ch3(net, train_iter, test_iter, loss, num_epochs, batch_size,
              None, None,trainer)
```

```
epoch 1, loss 1.1567, train acc 0.551, test acc 0.739
epoch 2, loss 0.5895, train acc 0.782, test acc 0.829
epoch 3, loss 0.4952, train acc 0.817, test acc 0.827
epoch 4, loss 0.4511, train acc 0.836, test acc 0.853
epoch 5, loss 0.4238, train acc 0.845, test acc 0.864
```