反向传播

笔记本:	pytorch		
创建时间:	2023/3/24 12:22	**更新时间:**	2023/3/24 21:59
作者:	22qdnlbn		
URL:	about:blank		

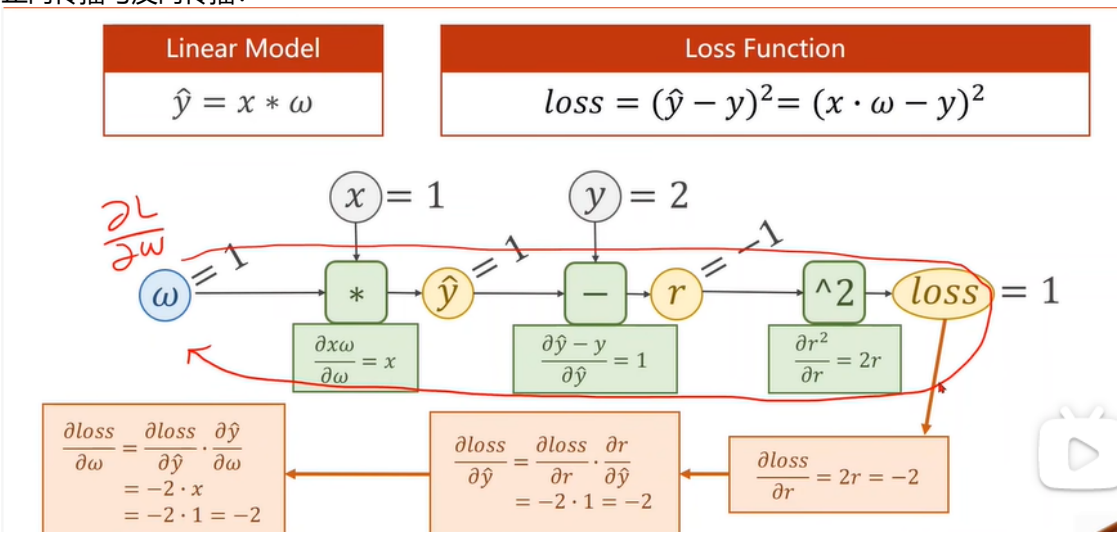
1、
神经网络学习包括正向传播和反向传播过程。

2、
对于一个计算节点，只要有其计算公式，我们就可以求出其对应的求导公式。

3、反向传播我们使用链式求导法则即可。

4、线性堆叠上线性仍然是线性函数，所以我们就需要一个激活函数，来解决非线性问题。

4.举一个简单的例子：
正向传播与反向传播：



如图，
只要我们有了计算公式，我们就可以对其公式中的参数来进行求导计算。
首先是正向传播计算出损失函数，之后通过反向传播来计算我们想要的导数。之后来更新参数。

5、
在pytorch中，Tensor在构建动态的图表时是一个重要的组成部分。

6、代码

```
# 时间: 2023/3/24 21:06
# cky
import torch


x_data=[1.0,2.0,3.0]
y_data=[2.0,4.0,6.0]
w=torch.Tensor([1.0])
w.requires_grad=True
def forword(x):
```

```
    return x*w
def loss(x,y):
    y_hat=forword(x)
    return (y_hat-y)**2
for epoch in range(101):
    for x,y in zip(x_data,y_data):
        l=loss(x,y)
        l.backward()
        w.data=w.data-0.05*w.grad.data
        w.grad.data.zero_()
    print("process:",epoch,l.item())
```

代码解析：

①
w=torch.Tensor([1.0])
w.requires_grad=True

将w定义为tensor类型，因为我们需要计算梯度，所以就为True。并且是将梯度信息存在w这个tensor张量中。

②
def forword(x):
  return x*w
def loss(x,y):
  y_hat=forword(x)
  return (y_hat-y)**2

每次调用一次loss，其实都在内存中有一张计算图。
w是tensor类型，所以x*w时，x也自动转换为了tensor类型。

③
for epoch in range(101):
  for x,y in zip(x_data,y_data):
    l=loss(x,y)
    l.backward()
    w.data=w.data-0.05*w.grad.data
    w.grad.data.zero_()
  print("pricess:",epoch,l.item())

l=loss（x,y）其实就是在内存中有了一个计算图，tensor类型。
backward（）即反向传播，将w这个计算出来的梯度保存在tensor w中，之后内存中的计算图便没有了
自动清空计算图。
w.data=w.data-0.05*w.grad.data 这个是更新权重w，我们不能直接使用w.grad来更新，因为这也是一个tensor类型，这样的话便会在内存中有一个计算图，但是由于我们没有backward的操作，所以这个计算图就会一直存在内存中。

我们使用.data的操作（这是一个属性），就会只对其中的数值来进行改变，而不会生成计算图。
.item()这个方法，是为了取得tensor中的值，将其变为一个python标量，item()只能取一个值。

易错点：
l=loss(x,y) 只是计算了一个样本的损失值，如果我们想要得到所有样本的损失值的话
应该
l_sum=0
l_sum+=l.item()
而不是 l_sum+=l
因为l是一个tensor类型，这样就会在内存中产生一个计算图，又由于没有backword操作，该计算图就会一直积累，如果数据过大， 就会出现不可预估的错误。

④
w.grad.data.zero_()
这是将我们求得的梯度进行清零操作。
我们求得的梯度已经用来计算梯度更新了，已经没有用了。

但是如果我们没有及逆行清零操作，本次求得的梯度就会一直保存在w中，每当求得一次梯度之后，梯度就会进行累加，但其实我们想要的梯度只是本次计算出来的梯度，所以要进行清零操作。

7、课后作业

y^=w1*x*x+w2*x+b
loss=(y^-y)的平方

```
# 时间：2023/3/24 21:44
# cky
import torch


x_data=[1,2,3]
y_data=[2,4,6]
w1=torch.Tensor([1.0])
w1.requires_grad=True
w2=torch.Tensor([1.0])
w2.requires_grad=True
b=torch.Tensor([1.0])
b.requires_grad=True


def forward(x):
    return w1*(x**2)+w2*x+b
def loss(x,y):
    y_hat=forward(x)
    return (y_hat-y)**2
print('Predict (befortraining)',4,forward(4).item())
for epoch in range(100):
    for x,y in zip(x_data,y_data):
        l=loss(x,y)
        l.backward()
        w1.data-=0.02*w1.grad.data
        w2.data -= 0.02 * w2.grad.data
        b.data -= 0.01 * b.grad.data
        w1.grad.zero_()
        w2.grad.zero_()
        b.grad.zero_()
    print('progress:',epoch,l.item())


print('Predict (aftertraining)',4,forward(4).item())
```