Table Of Contents:

今日工作

由于今天有考试和实验，学习内容有限。

1、阅读《计算机视觉 模型、学习和推理》第8章**回归模型**第一节**线性回归**的部分内容。

2、学习PyTorch反向传播。

回归模型

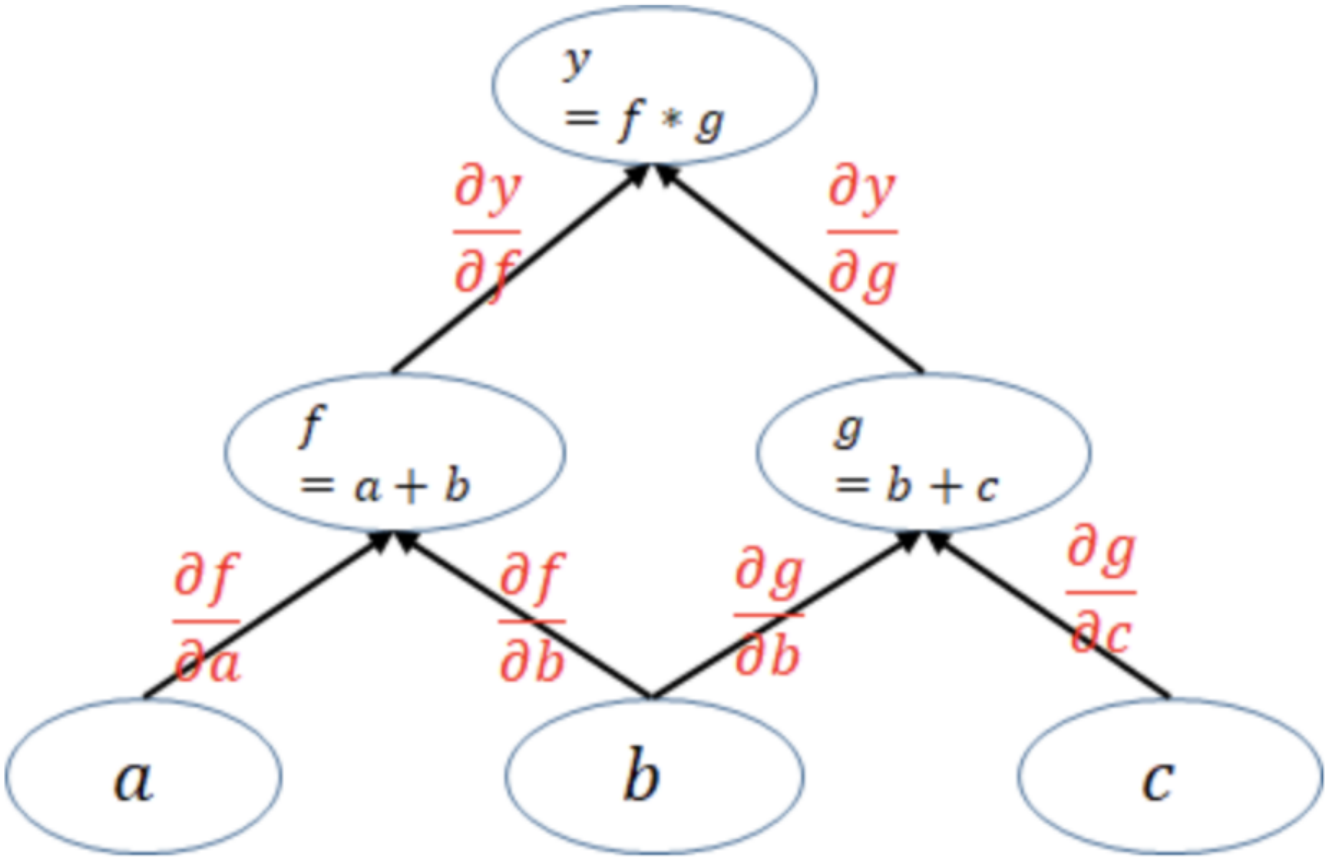
回归问题：根据观测值*x* 来估计**一元** 全局状态*ω* 。

本章讨论的是后验*Pr(ω|x)* 。与第七章讨论的似然*Pr(x|ω)* 形成对比。

线性回归

含义：全局状态*ω* 与观测数据*x* 的关系是线性的，并且对*ω* 的预测是不确定的，不确定性是一个具有常数协方差的正态分布。

PyTorch的反向传播



pytorch是动态图机制，所以在训练模型的时候，每迭代一次都会构建一个新的计算图。而计算图代表着程序中变量之间的关系。具体而言，每次向模型输入一个batch的训练数据，就会构建一个计算图。

当调用*.backward()* 方法时，默认情况下只会求位于叶子节点并且*requires_grad* 属性为*True* 的向量梯度。

```
def fun(x: torch.Tensor):
    return 1.3 * x + 4

a = torch.tensor(1, requires_grad=True, dtype=float)
b = fun(a)
# 不是叶子节点的向量要使用retain_grad()设置属性
# b.retain_grad()
y = fun(b)y.backward()
print(y)
print(a.grad)
print(b.grad)
```

输出结果为：

> *tensor(10.8900, dtype=torch.float64, grad_fn=<AddBackward0>) tensor(1.6900, dtype=torch.float64) None*

参数*b* 并没有梯度。因为其计算在图中的位置不是叶子节点，并且*requires_grad* 属性默认为*False* 。

因此需要手动设置他的*requires_grad* 属性，即代码中注释的部分。

在创建PyTorch的模型时，PyTorch框架会根据实际情况自动设置每层参数的属性，不需要我们关注*requires_grad* 属性。只需要关注网络结构。