机器学习作业

shwj

2023 年 2 月 20 日

1 人工神经网络

1.1 试叙线性神经函数 $f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}$ 作神经元激活函数的缺陷

MLP 的隐藏层相当于对输入的线性变换，而激活函数为非线性变换。若把激活函数同样换成线性变换，设输入为 $\boldsymbol{x}$，令 $a^0 = \boldsymbol{x}$，第 $l-1$ 层到第 $l-1$ 层的连接权重为 W^l，第 l 层的激活函数为 $f_l = w^T$。则第 l 层的输出

$$\boldsymbol{a^l} = f_l(W^l a^{l-1}) = w^T W^l a^{l-1} = w^T W^l w^T W^{l-1} \cdots w^T W^1 \boldsymbol{x}$$

神经网络退化为了全连接网络，只能对目标作线性拟合，失去了表达能力。

1.2 对于图 5.7 中的 v_{ih} 试推导出 BP 算法的更新公式 $\Delta v_{ih} = \eta e_h x_i$

根据梯度下降策略的定义 $\Delta v_{ih} = -\eta \frac{\partial E_k}{\partial v_{ih}}$，而根据链式求导法则我们有

$$\frac{\partial E_k}{\partial v_{ih}} = \frac{\partial E_k}{\partial b_h} \cdot \frac{\partial b_h}{\partial \alpha_h} \cdot \frac{\partial \alpha_h}{\partial v_{ih}}$$

而 α 是第 h 个神经元的输入 $\alpha_h = \sum_{i=1}^{d} v_{ih} x_i$ (这里 $x_1, x_2 \cdots x_d$ 为前层神经元的的输入后经过激活函数的输出) 那么显然有 $\frac{\partial \alpha_h}{\partial v_{ih}} = x_i$
根据定义 $e_h = -\frac{\partial E_k}{\partial b_h} \cdot \frac{\partial b_h}{\partial \alpha_h}$ 故

$$\frac{\partial E_k}{\partial v_{ih}} = \frac{\partial E_k}{\partial b_h} \cdot \frac{\partial b_h}{\partial \alpha_h} \cdot \frac{\partial \alpha_h}{\partial v_{ih}} = -\eta \cdot -e_h \cdot x_i = \eta e_h x_i$$

推导《西瓜书》的时候我曾疑惑为什么要费劲地用和式定义一个 e_h，但是手推一遍反向传播算法后就能明白这么干的原因式 GD 算法也是一个递推的过程，对于靠近输入层的神经元的梯度的计算需要综合前面所有层神经元的信息。而链式求导法则却能保证结果是 $\Delta v_{ih} = \eta e_h x_i$ 的简洁表示，使得 GD 算法的复杂度并不是很高。

另外观察梯度更新的表达式，可以理解“梯度沿着神经元传播”。所以 MLP 选择用 DAG(有向无环图) 的结构不光是模仿人脑神经元，也是为了保证图的拓扑结构使得梯度可以回传。pytorch 的计算图也是 DAG 结构。

MLP 的梯度表达式是多个偏导数相乘，所以当偏导数累计后很容易出梯度爆炸或梯度消失。修改激活函数是一种做法，这样在 GD 时能控制梯度的大小 (如 $tanh$ 相对于 $sign$,$LeakyReLU$ 相对于 $ReLU$)。而 LSTM 或是 RestNet 这类网络就是靠结构上的改变使 GD 的表达式增加了加和项解决了梯度的问题。

1.3 试叙述 $\Delta w_{hj} = -\eta \frac{\partial E_k}{\partial w_{hj}}$ 中学习率的取值对神经网络训练的影响

一些理解是：学习率 η 过大时，神经网络每一步“更新的”步长太大，网络不容易收敛到极小值点，表现在极值点左右“横跳”；而当 η 太小时，一方面达到收敛的所需更新步数较大，另一方面是会因更新的步长太小，陷入局部最小值。

神经网络相当于以输入的高维数据为自变量的函数，设其损失函数为 $f(x)$，因为网络较为复杂，故难以写出显示表达式，自然无法用求导的方法求得最小值。但我们可以利用迭代的方法，使其每次迭代时函数值减小。具体地，对 f 在 x 处展开就有

$$f(\boldsymbol{x} + d\boldsymbol{x}) \approx f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^T \cdot d\boldsymbol{x}$$

为此取 $d\boldsymbol{x} = -\eta \nabla f(\boldsymbol{x})^T$ 则有

$$\nabla f(\boldsymbol{x}) \cdot d\boldsymbol{x} = -\eta \|\nabla f(\boldsymbol{x})\|^2 \leq 0$$

当 η 较小时 $\boldsymbol{x} + d\boldsymbol{x} \approx \boldsymbol{x}$，可以认为 $f(\boldsymbol{x} + d\boldsymbol{x}) = f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^T \cdot d\boldsymbol{x}$ 成立，

所以 $f(\boldsymbol{x}+d\boldsymbol{x}) < f(\boldsymbol{x})$ 成立，即函数确实做到了损失值的减小。但是模型很容易达到局部最小值 $\nabla f(\boldsymbol{x})^T = 0$ 处，此时梯度有 $d\boldsymbol{x} = 0$ 梯度无法更新。

对于 η 较大的情况，考虑 f 的二阶展开

$$f(\boldsymbol{x}+d\boldsymbol{x}) = f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^T \cdot d\boldsymbol{x} + \frac{1}{2} d\boldsymbol{x}^T \cdot H_{\boldsymbol{x_0}} \cdot d\boldsymbol{x}$$

其中 $H_{\boldsymbol{x_0}}$ 是 f 的 $Hessian\,Matrix$ 在 $\boldsymbol{x_0}$ 的值，$\boldsymbol{x_0} \in (\boldsymbol{x}, \boldsymbol{x}+d\boldsymbol{x})$ 当 $\eta = 1$ 时

$$f(\boldsymbol{x}+d\boldsymbol{x}) - f(\boldsymbol{x}) = \nabla f(\boldsymbol{x}) \cdot [\frac{1}{2} H_{\boldsymbol{x_0}} - 1] \cdot \nabla f(\boldsymbol{x})^T$$

无法保证 $f(\boldsymbol{x}+d\boldsymbol{x}) > f(\boldsymbol{x})$ 成立。

1.4 计算 Lenet 的参数量

卷积层的输出尺寸为

$$output_{size} = [\frac{input_{size} - kenrel_{size} + 2 * padding - 1}{stride}] + 1$$

1. 第一个卷积参数为 $5 * 5 * (6 * 1 + 1) = 156$
2. 第二个卷积层参数为 $5 * 5 * (16 * 6 + 1) = 2416$
3. 第一个全连接层参数为 $120 * (400 + 1) = 48120$
4. 第二个全连接层参数为 $84 * (120 + 1) = 10164$
5. 第三个全连接层参数为 $10 * (84 + 1) = 850$

故总参数量为 $156 + 2416 + 48120 + 10164 + 850 = 61706$，下代码可以验证：

```
import torch
import torch.nn as nn
LeNet_5 = nn.Sequential(
            nn.Conv2d(1, 6, 5), #  [6, 28, 28]
            nn.Sigmoid(),
            nn.MaxPool2d(2, 2), # [6, 14, 14]
            nn.Conv2d(6, 16, 5),#[16, 10, 10]
            nn.Sigmoid(),
            nn.MaxPool2d(2, 2),#([16, 5, 5])
```

```
            nn.Flatten(),
            nn.Linear(16*5*5, 120),
            nn.Sigmoid(),
            nn.Linear(120, 84),
            nn.Sigmoid(),
            nn.Linear(84, 10)
        )
print(LeNet_5)
input = torch.rand([32,1,32,32])
output = LeNet_5(input)
print(output.shape)

total = sum([param.nelement() for
                param in LeNet_5.parameters()])
print(total)
for param in LeNet_5.parameters():
    print(param.nelement())
```