Notes: Neural Networks

2020 年 10 月 18 日

目录

1 感知机 Perceptron

前提：数据是线性可分的！

点到平面的距离公式：$d = \frac{|Ax_0+By_0+Cz_0+D|}{\sqrt{A^2+B^2+C^2}}$

输入:m 个样本即 $\left(x_1^{(0)}, x_2^{(0)}, \ldots x_n^{(0)}, y_0\right), \left(x_1^{(1)}, x_2^{(1)}, \ldots x_n^{(1)}, y_1\right), \ldots \left(x_1^{(m)}, x_2^{(m)}, \ldots x_n^{(m)}, y_m\right)$, 标签 y 是二元类别。

目标：找到一个超平面 (hyperplane) 即 $\theta_0 + \theta_1 x_1 + \ldots + \theta_n x_n = 0$，让其中一种类别的样本都满足 $\theta_0 + \theta_1 x_1 + \ldots + \theta_n x_n > 0$；而另一种类别的样本都满足 $\theta_0 + \theta_1 x_1 + \ldots + \theta_n x_n < 0$，从而线性可分！**简化写法：**增加一个特征 $x_0 = 1$，所以有超平面 $\sum_{i=0}^{n} \theta_i x_i = 0$ **向量表示：**

$$\theta_{(n+1)\times 1} \cdot x_{1\times(n+1)} = 0$$

感知机模型定义:

$$y = \text{sign}(\theta \bullet x)$$

$$\text{sign}(x) = \begin{cases} -1 & x < 0 \\ 1 & x \geq 0 \end{cases}$$

Gram 矩阵 (可以看成没有减去均值的协方差矩阵), 感知机**原始形式以及对偶形式**

2 DNN 反向传播

2.1 要解决的问题

在监督学习中往往要利用一个 loss function 来度量当前模型从输入得到的结果与真实值的误差。通过最小化这个误差，把整个模型往最好的方向调整即得到最好的 weight 和 b; 而最小化的方法便有梯度下降，牛顿法，拟牛顿法等...

3 RNN BPTT(back propagation through time)

模型定义参考刘建平 RNN 推导

3.1 前向传播

1. 对于任意一个序列索引号 t：$h^{(t)} = \sigma\left(z^{(t)}\right) = \sigma\left(Ux^{(t)} + Wh^{(t-1)} + b\right)$，这里的 σ 是 tanh 函数
2. 序列索引号 t 时模型的输出 o^t 的表达式为：$o^{(t)} = Vh^{(t)} + c$，c b 一样，都是偏置向量
3. 最终在序列索引号 t 时预测输出 $\hat{y}^{(t)} = \sigma\left(o^{(t)}\right)$，这里的 σ 是 softmax 函数

3.2 反向传播

通过梯度下降法的一轮迭代，得到模型参数 U, W, V, b, c，**这些参数在序列的各个位置是共享的**，BP 时更新的是相同的参数！

由于在序列的每个位置都有损失函数 $L^{(t)} = -{y^{(t)}}^T log\hat{y}^{(t)}$，因此最终的损失 L 为：$L = \sum_{t=1}^{\tau} L^{(t)}$

下面的推导详见草稿纸笔记。。。

4 Convolution

Convolution (from en.wikipedia) 、卷积 (from zh.wikipedia)

卷积的连续、离散定义见上述两个链接。

卷积性质及其证明：

- 第 2 条微分性质是连续卷积的；
- 第 5 条是从信号处理的角度推导出离散的卷积形式；

在上述两个维基百科链接中，**卷积的微分性质同时适用于连续和离散的形式，只是在离散情况下微分算子 D 变成差分算子！**

卷积的含义及直观地理解

复旦邱锡鹏《神经网络与深度学习》(下文均称作 nndl) 第 5 章 5.1 详细介绍了数学上的一维、二维卷积定义以及互相关定义！(零基础入门深度学习

(4) - 卷积神经网络、卷积神经网络 (CNN) 刘建平 与上述书本介绍的定义是类似的，只是书上的卷积核下标从 1 开始，而这两个链接卷积核的下标从 0 开始！)

从信号叠加的角度来看比较容易理解，**一维卷积** $y_t = \sum_{k=1}^{K} w_k x_{t-k+1}$ 表示在时刻 t 收到的信号 y_t 为当前时刻产生的信息 $\{x_t\}$ 和以前时刻延迟信息 $\{x_{t-k+1}\}$ 的叠加。一般来说，每个时间点 t 都有完整时序 (数目) 变化 $[1, K]$ 的 w_k 被称作卷积核或滤波器 (在二维 (图像) 情况下同理)。

二维卷积：给定一个图像 $\boldsymbol{X} \in \mathbb{R}^{M \times N}$ 和一个滤波器 $\boldsymbol{W} \in \mathbb{R}^{U \times V}$ 一般 $U \ll M, V \ll N$，则 $y_{ij} = \sum_{u=1}^{U} \sum_{v=1}^{V} w_{uv} x_{i-u+1, j-v+1}$，这里假设输出 y_{ij} 的下标 (i, j) 从 (U, V) 开始 (Y 矩阵的第一个元素)，这样输出矩阵才不会出现下标为负的情况。可以看出在二维卷积的定义下，**卷积核是翻转了 180 度**在输入矩阵上滑动并做加权求和操作。参考零基础入门链接里的二维卷积定义的图示并将其写成表达式即可理解！(翻转指从两个维度 (从上到下、从左到右) 颠倒次序, 即旋转 180 度)

互相关：设定如上述二维卷积，但公式变为 $y_{ij} = \sum_{u=1}^{U} \sum_{v=1}^{V} w_{uv} x_{i+u-1, j+v-1}$ 和二维卷积公式对比可知，**互相关的卷积核不进行翻转**。

4.1 Back Propagation

书本 nndl 以及上述两个链接均采用链式法则 + 逐标量的推导方式来找到反向传播的矩阵表达式；但是直接用矩阵、向量的链式法则是推导不出来的 (因为是互相关操作而不是矩阵乘法)！

nndl 中 Sec 5.1.4.2 从互相关定义出发，逐分量给出了标量对进行了互相关操作后的矩阵的导数！这与上述两个链接里求反向传播参数中使用到的推导方法是一样的！

理论推导看上述两个链接 +nndl 第五章即可！

4.1.1 Naive 实现

具体细节说明见零基础入门深度学习 (4)-卷积神经网络；实现参考：零基础入门深度学习 (4)-卷积神经网络具体实现；numpy-ml 中也有 navie 的 backward() 实现。

在 Naive 实现反向传播时：

- 求 $\delta^{l-1} = \dfrac{\partial L}{\partial Z^{l-1}}$ 时，注意 δ^l 需要根据 stride 在对应位置补零从而还

原成 stride=1 时的大小 (即 δ^l 周围补一圈零)；第 l 层卷积核 W^l 需要翻转 180 度；然后 W^l 在还原后的 δ^l 上进行互相关操作再按照上述链接推导出的公式得出结果！

- 求 $\frac{\partial J}{\partial W^l}, \frac{\partial J}{\partial b}$ 时，不需要翻转！

4.1.2 向量化实现

如果在 CNN layer 的计算中，把前向传播变成矩阵乘法 (下文提到的向量化)，那么在做反向传播的时候，就不需要去关心误差项矩阵补零 (padding) 或者还原成步长 (stride) 为 1 时的矩阵了！直接利用类似全连接层的 forward()+backward() 最后通过逆向量化操作 (col2im) 完成计算！

向量化 im2col() 原理及基本实现参考:

- 最清晰的 im2col 过程及实现(不带 padding)
- 前向使用 im2col，反向时的卷积操作也是用了 im2col(不带 padding)，没有利用类似全连接层 BP 的操作 +col2im
- 前向 im2col+ 反向 col2im(没有 Batch 维度)(主要看**类似全连接层的算法**过程)

numpy-ml 的 Conv2D 层就是利用 im2col+col2im 等向量化手段实现了类似全连接层的操作，输入维度 $=(B,H,W,C)$。

《深度学习入门：基于 Python 的理论与实现》中第 7 章 7.4 实现卷积层和池化层 forward() 使用了 im2col()；实现 backward() 使用了 col2im()，但是它的 im2col() 和 col2im() 都是用两层 for 循环实现的。而且是基于 PyTorch 的输入格式 (B,C,H,W)。

(去掉 for 循环的 im2col() 实现见卷积算法另一种高效实现，as_strided 详解)

该书的实现流程如下：假设输入张量为 $X=(B,C,H,W)$，卷积核为 $W=(C_{out},C_{in},FH,FW)$，C_{out} 是卷积核数目，下文中 $C=C_{in}$。

forward():

- 输入张量 $X=(B,C,H,W) \overset{im2col()}{\Longrightarrow} X=(B*H_{out}*W_{out}, FH*FW*C_{in})$

- 卷积核 $W = (C_{out}, C_{in}, FH, FW) \overset{reshape}{\Longrightarrow} W = (C_{out}, C_{in} * FH * FW) \overset{transpose}{\Longrightarrow} W = (C_{in} * FH * FW, C_{out})$

- 输出矩阵 $Y = XW + \mathbf{1}\boldsymbol{b}^T$，这里的 $\mathbf{1} = (B * H_{out} * W_{out}, 1)$ 是全 1 列向量，$\boldsymbol{b} = (B * H_{out} * W_{out}, 1)$ 也是列向量，$\mathbf{1}\boldsymbol{b}^T$ 表明每个卷积核对应一个 b_i 元素！

- 所以将矩阵还原回张量 $Y = (B * H_{out} * W_{out}, C_{out}) \overset{reshape}{\Longrightarrow} Y = (B, H_{out}, W_{out}, C_{out}) \overset{transpose}{\Longrightarrow} Y = (B, C_{out}, H_{out}, W_{out})$

backward():

- 给定误差项张量 $dY = (B, C_{out}, H_{out}, W_{out})$，先转为二维矩阵形式以便于利用链式法则直接求出导数，$\overset{transpose}{\Longrightarrow} dY = (B, H_{out}, W_{out}, C_{out}) \overset{reshape}{\Longrightarrow} dY = (B * H_{out} * W_{out}, C_{out})$

- 根据标量对矩阵求导的链式法则 (见 matrix derivative.pdf)，则有 $dW = X^T dY, \quad (X = (B * H_{out} * W_{out}, FH * FW * C_{in}))$；再将 $dW = (FH * FW * C_{in}, C_{out}) \overset{transpose}{\Longrightarrow} dW = (C_{out}, C_{in} * H_{out} * W_{out}) \overset{reshape}{\Longrightarrow} dW = (C_{out}, C_{in}, H_{out}, W_{out})$

- $dX = dY \cdot W^T, (W = (C_{in} * FH * FW, C_{out})), \quad dX = (B * H_{out} * W_{out}, FH * FW * C_{in}) \overset{col2im()}{\Longrightarrow} dX = (B, C_{in}, H, W)$

- $d\boldsymbol{b} = dY^T \cdot \mathbf{1}, \quad d\boldsymbol{b} = (C_{out}, 1)$，相当于 np.sum(dY, axis=0)，有两种推导方法：
  1. 按照链式法则 $\boldsymbol{b} \to Y \to J$ 逐分量的推导，可以假设一个很小的例子开始；
  2. 按照微分法推导 $dJ = tr[(\frac{\partial J}{\partial Y})^T dY] = tr[((\frac{\partial J}{\partial Y})^T \cdot \mathbf{1})^T d\boldsymbol{b}]$

(np.reshape 是按照行的顺序叠加)

具体实现细节见上面给出对应书本源码的链接。需要注意的是，书上的 im2col() 和 col2im() 实现思路与“向量化 im2col() 原理及基本实现参考”给出的 3 个链接中的思路不相同，但除了书上的方法较完善 (带 pading) 以及两者针对输入张量的维度顺序不同之外，最终的功能是一样的！

5 Attention 机制

参考Attention Review，再看JayLou 娄杰的回答! 能比较系统地了解传统 attention(soft) 和 self-attention 的计算过程!

5.1 Q,K,V 三元组

soft-attention 中有两种模式：

1. Key=Value=X
2. Key!=Value

假设输入 $X_{m\times n}$ 是由样本行向量拼成的矩阵，$Q_{m\times n}, K_{m\times n}, V_{m\times n}$，且对应的 q_i, k_i, v_i 均是行向量！(写成列向量也行，后面的推导做转置即可)

则进行 attention 得到的特征变化结果：

- $att(\boldsymbol{q_i}, (K, V)) = \sum_{j=1}^{m} \alpha_j \boldsymbol{v}_j$
- $\alpha_j = sim(\boldsymbol{q_i}, \boldsymbol{k_j}) = softmax(\boldsymbol{q_i}, \boldsymbol{k_j}) = \frac{\boldsymbol{q_i k_j^T}}{\sum_{l=1}^{m} \boldsymbol{q_i k_l^T}}$(这里假设注意力分布采用内积形式)
- 所以 $att(\boldsymbol{q_i}, (K, V)) = \sum_{j=1}^{m} \frac{\boldsymbol{q_i k_j^T}}{\sum_{l=1}^{m} \boldsymbol{q_i k_l^T}} \boldsymbol{v}_j$

写成矩阵化形式：

- $att(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V})_{m\times n} = softmax(\boldsymbol{QK}^T, dim = 1)\boldsymbol{V}$，此处的 softmax 作用于矩阵的每一行!
- 与前面的求和式进行对比，$(\boldsymbol{QK}^T)_{m\times m}$ 的第 i 行经过 softmax 后中的第 j 个值对应于 q_i 与行向量 k_j 的相似度 α_j；
- 然后 $(\boldsymbol{QK}^T)_{m\times m}$ 的第 i 行 (1 x m) 乘上矩阵 $\boldsymbol{V}_{m\times n}$ 相当于前面求和式中最外层的求和表达式！可以用矩阵乘法—> 行 x 矩阵

Pytorch 实现 self-attention & transformer