南開大學
Nankai University

南　　开　　大　　学

网 络 空 间 安 全 学 院

机器学习实验报告

实验二：MLP_BP

2014074 费泽锟

年级：2020 级

指导教师：谢晋

2022 年 11 月 24 日

摘要

在本次实验之中，将以三层感知机模型为例，使用反向传播算法也就是 BP 算法，更新 MLP 各层参数和偏置项，通过链式法则从输出层开始计算，逐步进行各个权重和偏置项的偏导计算，以此计算求得梯度来更新 MLP 的权重和偏置项。

关键字：MLP; Back Propagation; Vectorization; Derivative; Softmax

目录

一、 实验内容

在本次实验之中，我们需要以三层感知机为例，使用反向传播算法更新 MLP 的权重和偏置项，将推导过程通过公式编辑器进行编辑，并将推导过程以报告的形式进行提交。我们先来看看具体的英文原版要求。

Define S_w and S_b as:

$$S_w = \sum_{c=1}^{C} \sum_{\boldsymbol{y}_i^M \in c} (\boldsymbol{y}_i^M - \boldsymbol{m}_c^M)(\boldsymbol{y}_i^M - \boldsymbol{m}_c^M)^T$$
$$S_b = \sum_{c=1}^{C} n_c (\boldsymbol{m}_c^M - \boldsymbol{m}^M)(\boldsymbol{m}_c^M - \boldsymbol{m}^M)^T \quad (1)$$

where $\boldsymbol{m}_c^M$ is the mean vector of $\boldsymbol{y}_i^M$ (the output of the ith sample from the cth class), $\boldsymbol{m}^M$ is the mean vector of the output $\boldsymbol{y}_i^M$ from all classes, n_c is the number of samples from the cth class. Define the discriminative regularization term $tr(S_w) - tr(S_b)$ and incorporate it into the objective function of the MLP:

$$E = \sum_i \sum_j \frac{1}{2}(\boldsymbol{y}_{i,j}^M - \boldsymbol{d}_{i,j})^2 + \frac{1}{2}\gamma(tr(S_w) - tr(S_b)). \quad (2)$$

where $\boldsymbol{y}_{i,j}^M$ is the jth element in the vector $\boldsymbol{y}_i^M$, $\boldsymbol{d}_{i,j}$ is the jth element in the label vector $\boldsymbol{d}_i$, tr denotes the trace of the matrix. Use the BP algorithm to update parameters $\boldsymbol{W}$ and $\boldsymbol{b}$ of the MLP.

图 1: 实验内容

我们来看看其中具体的各个函数定义。其中我们可以看到 Sw 和 Sb 的定义，这两个运算之中 Sw 和 Sb 经过计算后均为 C*C 的矩阵，我们可以紧接着在 Loss Function 之中看到使用了这两个矩阵，Loss Function 损失函数根据定义形式如下：

$$E = \sum_i \sum_j \frac{1}{2}\left(\boldsymbol{y}_{i,j}^M - \boldsymbol{d}_{i,j}\right)^2 + \frac{1}{2}\gamma\left(\mathrm{tr}\left(S_w\right) - \mathrm{tr}\left(S_b\right)\right).$$

对 Loss Function 进行分析，可以发现这两个矩阵出现了正则项部分，发现在正则项部分均对这两个矩阵进行了 trace 运算，也就是对角线各数值之和。

认真分析如何进行运算得到的这两个矩阵，可以发现对于 Sw 矩阵的对角线的数据，其实就是对 c 类样本中的每一个样本求取预测值与平均值之差的平方。而对于每一类样本，以及每一类样本之中的每一个样本都进行求和，进而求得了参数 w 对损失函数正则项之中的贡献。

也就是说通过每一类样本之间的差距求和来表征正则项之中的 w 的部分，通过每一类样本的平均值减去总体的平均值的矩阵运算来表征正则项部分之中的 b 偏置部分。

对于较为简单的三层感知机模型而言，可以得到对应的函数表达形式如下：

$$f(x) = G(b^{(2)} + W^{(2)}(s(\boldsymbol{b}^{(1)} + \boldsymbol{W}^{(1)}x)))$$

从这其中我们也能看出其实对于 W 参数部分的 Sw 只需要将每一类之中的样本减去平均值，即可消去相应 b 偏置的影响，而 Sb 的计算需要以整个该类的整体携带 b 偏置项的部分来进行计算。

二、 MLP BP 推导过程

我们先来看看具体的三层感知机模型的示意图：

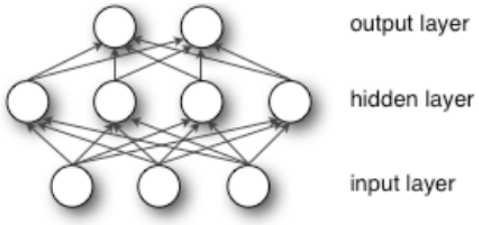

图 2: 三层感知机

在本次实验之中，我们的损失函数是：

$$E = \sum_i \sum_j \frac{1}{2} \left(\boldsymbol{y}_{i,j}^M - \boldsymbol{d}_{i,j} \right)^2 + \frac{1}{2} \gamma \left(\mathrm{tr} \left(S_w \right) - \mathrm{tr} \left(S_b \right) \right).$$

可以看到这里的损失函数并没有使用交叉熵损失函数，而是使用的是 MSE 损失函数（Mean Squared Error Function）来进行的计算。也就是说这里并没有进行最后一步的 softmax 函数将对应预测的概率映射到 [0,1] 的空间之内，所以我们在推导计算的过程之中，从均方误差损失函数开始推导即可。这样就可以得到三层感知机模型的函数运算流程如下：

$$\begin{aligned} \boldsymbol{z}^{(1)} &= \boldsymbol{W}^{(1)} x + \boldsymbol{b}^{(1)} \\ \boldsymbol{y}^{(1)} &= \sigma(\boldsymbol{z}^{(1)}) \\ \boldsymbol{z}^{(2)} &= \boldsymbol{W}^{(2)} y^{(1)} + \boldsymbol{b}^{(2)} \\ \boldsymbol{y}^{(2)} &= \boldsymbol{z}^{(2)} \end{aligned}$$

得到如上的 MLP 感知机模型之中的函数运行流程，因为接着没有使用 softmax 函数进行映射操作，所以我们接着使用 $\boldsymbol{y}^{(2)}$ 的值，也就是各个样本的预测值来进行给出的损失函数计算。再给出一遍损失函数的形式：

$$E = \sum_i \sum_j \frac{1}{2} \left(\boldsymbol{y}_{i,j}^M - \boldsymbol{d}_{i,j} \right)^2 + \frac{1}{2} \gamma \left(\mathrm{tr} \left(S_w \right) - \mathrm{tr} \left(S_b \right) \right).$$

从损失函数这一步骤进行逆向回推运算，需要注意的是这里的全连接层激活函数 σ 为 sigmoid 函数，由链式法则可得损失函数对第 i 层输出的梯度 $\frac{\partial E}{\partial y^{(i)}}$(称为激活值梯度)，可以由前一层迭代求出：

$$\frac{\partial E}{\partial \boldsymbol{y}^{(i)}} = \frac{\partial \boldsymbol{y}^{(i+1)}}{\partial \boldsymbol{y}^{(i)}} \frac{\partial E}{\partial \boldsymbol{y}^{(i+1)}}$$

如果该激活层刚好为输出层前的最后一层，则使用 Loss Function 求偏导求出：

$$\frac{\partial E}{\partial \boldsymbol{y}^{(i)}} = \boldsymbol{y}_i - \boldsymbol{d}_i + \frac{\partial Norm}{\partial \boldsymbol{y}^{(i)}}$$

接着就是在神经元节点处，通过链式法则，计算激活值梯度的前一步也就是状态值梯度：

$$\frac{\partial E}{\partial \boldsymbol{z}^{(i)}} = \frac{\partial \boldsymbol{y}^{(i)}}{\partial \boldsymbol{z}^{(i)}} \cdot \frac{\partial E}{\partial \boldsymbol{y}^{(i)}}$$

$$\frac{\partial E}{\partial W^{(i)}} = \frac{\partial \boldsymbol{z}^{(i)}}{\partial W^{(i)}} \cdot \boldsymbol{\delta}^{(i)}$$

$$\frac{\partial E}{\partial \boldsymbol{b}^{(i)}} = \frac{\partial \boldsymbol{z}^{(i)}}{\partial \boldsymbol{b}^{(i)}} \cdot \boldsymbol{\delta}^{(i)}$$

如果保留出从损失函数到目前状态梯度的求解过程即为上述所示，因为是三层感知机模型，如果是多层感知机模型全连接层超出三层，需要对多层的状态梯度进行递归求解。

计算出状态值梯度，一般记为 $\boldsymbol{\delta}^{(i)}$，接下来将用 $\boldsymbol{\delta}^{(i)}$ 来表示状态部分的梯度。

上述仅仅是 BP 算法的整体流程，接下来我们来进行包含具体细节的实操，我们先求输出层的参数梯度，首先根据上述对损失函数分析，我们对第一项 $\sum_i \sum_j \frac{1}{2}\left(\boldsymbol{y}_{i,j}^M - \boldsymbol{d}_{i,j}\right)^2$ 进行改写，然后对其求导。

我们可以将其改写成 $E = \frac{1}{2}\left(\boldsymbol{y}^* - \boldsymbol{y}^{(2)}\right)^T\left(\boldsymbol{y}^* - \boldsymbol{y}^{(2)}\right)$

其中，$\boldsymbol{y}^{(2)}$ 即为三层全连接神经网络的输出 $(y_1^M, y_2^M, ...y_n^M)$, $\boldsymbol{y}^*$ 表示其真实分布，也就是式中的 $\boldsymbol{d}_{i,j}$，样本的真实标签。

先将第一项进行求偏导，可得:

$$\frac{\partial E}{\partial \boldsymbol{y}^{(2)}} = \boldsymbol{y}^{(2)} - \boldsymbol{y}^*$$

这里直接用输出层 $\boldsymbol{y}^{(2)}$ 来表示了（因为是三层感知机）。

因为没有使用 softmax 进行归一化操作，所以 $\boldsymbol{y}^{(2)} = \boldsymbol{z}^{(2)}$，有:

$$\boldsymbol{\delta}^{(2)} = \boldsymbol{y}^{(2)} - \boldsymbol{y}^*$$

而对于损失函数之中的正则项部分的处理，则会有点麻烦因为涉及到了类内取平均和类间取平均的操作。

所以对于损失函数的正则项部分 $\frac{1}{2}\gamma\left(\mathrm{tr}\left(S_w\right) - \mathrm{tr}\left(S_b\right)\right)$，这里我们可以用另一种形式来表示该正则项:

$$\frac{1}{2}\gamma[\sum_{c=1}^{C}\sum_{\boldsymbol{y}_i^M \in c}\left(\boldsymbol{y}_i^M - \boldsymbol{m}_c^M\right)^T\left(\boldsymbol{y}_i^M - \boldsymbol{m}_c^M\right) - \sum_{c=1}^{C} n_c\left(\boldsymbol{m}_c^M - \boldsymbol{m}^M\right)^T\left(\boldsymbol{m}_c^M - \boldsymbol{m}^M\right)]$$

该项对 $\boldsymbol{y}^{(2)}$ 求偏导，因为是标量函数对类别的列向量求偏导，所以应当得到的偏导值为列向量表示，对应的不同类别的即将进行 BP 更新的梯度值:

$$\frac{\partial E}{\partial \boldsymbol{y}^{(2)}} = \gamma(\sum_{\boldsymbol{y}_i^M \in c}(1 - \frac{1}{n}) \cdot (y_i^M - m_c^M) - n_c(\frac{1}{n} - \frac{1}{N}) \cdot (m_c^M - m^M))$$

因此，损失函数对输出层求偏导（对于正则项部分不是很确定），也就是该层激活值梯度最终为:

$$\boldsymbol{\delta}^{(2)} = \sum_j (y_{i,j}^M - d_{i,j}^M) + \gamma(\sum_{\boldsymbol{y}_i^M \in c}(1 - \frac{1}{n}) \cdot (y_i^M - m_c^M) - n_c(\frac{1}{n} - \frac{1}{N}) \cdot (m_c^M - m^M)) \tag{1}$$

已经完成了从 Loss Function 进行参数更新的第一步逆推，接下来就是线性函数的梯度逆推，对于连接输出层的线性函数：$\boldsymbol{z}^{(2)} = W^{(2)}\boldsymbol{y}^{(1)} + \boldsymbol{b}^{(2)}$，对其求微分可得:

$$d\boldsymbol{z}^{(2)} = dW^{(2)}\boldsymbol{y}^{(1)} + W^{(2)}d\boldsymbol{y}^{(1)} + d\boldsymbol{b}^{(2)}$$

将上述表达式进行向量化（具体内容可见参考文献 1、2）

$$\mathrm{vec}\left(d\boldsymbol{z}^{(2)}\right) = \left(\boldsymbol{y}^{(1)^T} \otimes I^{(2)}\right)\mathrm{vec}\left(dW^{(2)}\right) + \left(1 \otimes W^{(2)}\right)\mathrm{vec}\left(d\boldsymbol{y}^{(1)}\right) + \mathrm{vec}\left(d\boldsymbol{b}^{(2)}\right)$$

使用上式，也就是经过向量化表达的微分表达式，对函数之中的参数 $\boldsymbol{W}^{(2)}$ 进行求导，可得结果如下:

$$\frac{\partial \boldsymbol{z}^{(2)}}{\partial W^{(2)}} = \boldsymbol{y}^{(1)} \otimes I^{(2)};$$

$$\frac{\partial \boldsymbol{z}^{(2)}}{\partial \boldsymbol{b}^{(2)}} = I^{(2)};$$

$$\frac{\partial \boldsymbol{z}^{(2)}}{\partial \boldsymbol{y}^{(1)}} = {W^{(2)}}^{T};$$

其中 $I^{(2)}$ 表示单位矩阵，其大小与全连接层输出维度大小一致。完成这一步后，会发现一个重要的问题就是上述三个表达式之中第一个表达式是暂时无法处理的表达式，这时就需要进行消掉 Kronecker 积的操作。

Kronecker 积：克罗内克积是两个任意大小的矩阵间的运算，克罗内克积是张量积的特殊形式。（这有点难理解，但我们可以根据运算规则进行消除计算）

对于 Kronecker 积而言，m 阶矩阵和 n 阶矩阵之间是无法直接求和的，所以，我们通过对单位阵的 Kronecker 积运算，同时把他们化为 m*n 阶方阵。这样，就可以对矩阵进行求和运算了。

于是对于表达式 $\frac{\partial E}{\partial W^{(2)}}$ 作进一步化简，消掉 Kronecker 积:

$$\begin{aligned}\frac{\partial E}{\partial W^{(2)}} &= \left(\boldsymbol{y}^{(1)} \otimes I^{(2)}\right)\left(1 \otimes \boldsymbol{\delta}^{(2)}\right) \\ &= \left(\boldsymbol{y}^{(1)}\right) \otimes \left(I^{(2)}\boldsymbol{\delta}^{(2)}\right) \\ &= \boldsymbol{y}^{(1)} \otimes \boldsymbol{\delta}^{(2)} \\ &= \mathrm{vec}\left(\boldsymbol{\delta}^{(2)}{\boldsymbol{y}^{(1)}}^{T}\right)\end{aligned}$$

对于隐藏层的参数梯度 $\frac{\partial E}{\partial W^{(1)}}$, $\frac{\partial E}{\partial b^{(1)}}$，假设我们使用的是 sigmoid 激活函数（也可以使用 tanh 函数），即 $\boldsymbol{y}^{(1)} = \sigma\left(\boldsymbol{z}^{(1)}\right)$，那我们可以轻松地知道 sigmoid 函数的求导后的表达式的模样。

$$f(z)' = f(z)(1 - f(z)$$

接着可以根据链式法则，将之前的运算结果代入 sigmoid 函数求导后的表达式之中继续进行逆推导:

$$\boldsymbol{\delta}^{(1)} = \mathrm{diag}\left(\sigma'\left(\boldsymbol{z}^{(1)}\right)\right){W^{(2)}}^{T}\boldsymbol{\delta}^{(2)} \tag{2}$$

当我们完成了唯一激活层函数的求梯度过程之后，就只剩下从输入层到全连接层的线性函数部分的推导了，这一部分的推导结果与从输出层到全连接层的线性函数推导部分相似，这里就不再赘述了。

于是，我们可以得到最终的参数与偏置项的梯度推导结果如下:

$$\frac{\partial E}{\partial W^{(2)}} = \mathrm{vec}\left(\boldsymbol{\delta}^{(2)}{\boldsymbol{y}^{(1)}}^{T}\right); \quad \frac{\partial E}{\partial \boldsymbol{b}^{(2)}} = \mathrm{vec}\left(\boldsymbol{\delta}^{(2)}\right) \tag{3}$$

$$\frac{\partial E}{\partial W^{(1)}} = \mathrm{vec}\left(\boldsymbol{\delta}^{(1)}\boldsymbol{X}\right); \quad \frac{\partial E}{\partial \boldsymbol{b}^{(1)}} = \mathrm{vec}\left(\boldsymbol{\delta}^{(1)}\right) \tag{4}$$

三、 实验总结

在本次实验之中，深入地了解了 MLP_BP 算法的数学原理，虽然对于 BP 算法而言如果只是计算某一个参数的梯度值那么可以直接针对该参数进行运算，对于 BP 算法的推导可以基于对每一个参数也就是 w(i,j) 进行推导（参考文献 3、4 对其进行了描述），也可以使用基于向量的

表达形式的推导（这里主要学习的是参考文献 1、2），通过学习和运算验证，完成了本次的推导过程。

通过对三层感知机模型进行了每层的分析与逆推过程，熟悉了神经网络训练过程之中的一些细节内容，也初步了解了向量化的运算流程（便于矩阵编程运算），为后续的搭建 LeNet5 深度学习模型做准备。

参考文献

https://zhuanlan.zhihu.com/p/431886591
https://zhuanlan.zhihu.com/p/141745257
https://blog.csdn.net/qq_24739717/article/details/98474606
https://blog.csdn.net/z_feng12489/article/details/89187037