Neural Network Theory and Applications Homework Assignment 1

林煜

2017 年 3 月 12 日

1. **Proof.** 这里，传输函数使用硬极限传输函数。类别t及输出a$\subseteq \{0,1\}$,对于此问题，仅需证明一个神经元收敛性，对多个神经元同样适用。
   令$\vec{x} = \begin{bmatrix} W_i^T \\ b_i \end{bmatrix}$,输入$\vec{z} = \begin{bmatrix} \vec{p} \\ 1 \end{bmatrix}$.
   则学习规则可改写为
   $$x^{new} = x^{old} + e\vec{z}$$
   误差e可以取1，-1，0。假设解存在(即问题线性可分), 令解为$\vec{x}_*$, 在所有数据点中，存在$\delta > 0$.使得
   当$t_q = 1$,那么$x_*^T z_q > \delta > 0$
   当$t_q = 0$,那么$x_*^T z_q < -\delta < 0$
   现只考虑权值向量发生改变的那些迭代，令$\vec{x}_k$ 为迭代k次后的权值向量。则
   $$\vec{x}_*^T \vec{x}_k = \vec{x}_*^T(\vec{x}_{k-1} + \alpha e_{k-1}\vec{z}_{k-1})$$
   由于$\vec{x}_*^T \vec{z}_{k-1}$与$e_{k-1}$ 同号，这里e_{k-1}仅为1或-1，所以$e_{k-1}\vec{x}_*^T \vec{z}_{k-1} > \delta > 0$，于是
   $$\vec{x}_*^T \vec{x}_k = \vec{x}_*^T(\vec{x}_{k-1} + \alpha e_{k-1}\vec{z}_{k-1}) > \vec{x}_*^T \vec{x}_{k-1} + \alpha\delta > \vec{x}_*^T \vec{x}_{k-2} + 2\alpha\delta > ... > k\alpha\delta$$
   由柯西不等式得
   $$(\vec{x}_*^T \vec{x}_k)^2 \leq \| \vec{x}_* \|^2 \| \vec{x}_k \|^2$$
   故
   $$\| \vec{x}_k \|^2 \geq \frac{(\vec{x}_*^T \vec{x}_k)^2}{\| \vec{x}_* \|^2} > \frac{(k\alpha\delta)^2}{\| \vec{x}_* \|^2}$$
   另外
   $$\begin{aligned} \| \vec{x}_k \|^2 &= \vec{x}_k^T \vec{x}_k \\ &= [\vec{x}_{k-1} + \alpha e_{k-1}\vec{z}_{k-1}]^T [\vec{x}_{k-1} + \alpha e_{k-1}\vec{z}_{k-1}] \\ &= \vec{x}_{k-1}^T \vec{x}_{k-1} + 2\alpha e_{k-1}\vec{x}_{k-1}^T \vec{z}_{k-1} + \alpha^2 (e_{k-1}\vec{z}_{k-1})^T (e_{k-1}\vec{z}_{k-1}) \\ &= \| \vec{x}_{k-1} \|^2 + 2\alpha e_{k-1}\vec{x}_{k-1}^T \vec{z}_{k-1} + \alpha^2 \| \vec{z}_{k-1} \| \end{aligned}$$
   因为只有在前一输入被错误分类时权值向量才会更新，所以$e_{k-1}\vec{x}_{k-1}^T \vec{z}_{k-1} < 0$,由于我们要求的是上界，所以$\alpha > 0$, 则
   $$\| \vec{x}_k \|^2 \leq \| \vec{x}_{k-1} \|^2 + \alpha^2 \| \vec{z}_{k-1} \|^2 \leq ... \leq \alpha^2 (\| \vec{z}_0 \|^2 + ... + \| \vec{z}_{k-1} \|^2)$$
   令$\prod = max\{\| \vec{z}_i \|^2\}$,则上界可简化成
   $$\| \vec{x}_k \|^2 \leq k\alpha^2 \prod$$

所以

$$k\alpha^2 \prod \geq \| \vec{x}_k \|^2 > \frac{(k\alpha\delta)^2}{\| \vec{x}_* \|^2}$$

得$k < \frac{\|\vec{x}_*\|^2\Pi}{\delta^2}$，k有上界意味这改变次数是有限的，所以该学习规则将在有限次迭代后收敛。

对于学习速率α，求上界过程中需要满足$\alpha > 0$,否则将不存在上界。

2. 网络设计：
    (a) 采用单层神经网络，神经元数量为2
    (b) 权值函数W为2×2矩阵，偏置b为二维向量
    (c) 传输函数采用硬极限函数
    (d) 输出$\vec{a}$为二维向量$\subset \{0,1\}^2$，其中$a_i = hardlim(W_i p + b_i)$($W_i$对应W第i行). 对于类别，输出00代表第一类，01 代表第二类，10、11 代表第三类
    (e) 采用第一题的学习规则

程序设计：采用python3结合numpy库实现，下面是伪代码

Algorithm 1: 三分类感知机

input ：训练数据集data，格式:（数据item，所属类别target）
output: 权值矩阵W与偏置向量b

```
end ← False;
iteration ← 0;
while end=False and iteration < 50 do
    end ← True;
    i ← 0;
    while i < len(data) do
        p ← data[i].item;
        target ← data[i].target;
        res ← f(Wp + b)(f采用硬极限函数);
        res，00对应1类，01对应2类，10和11对应3类;
        if res对应类别与target不相等 then
            e = target对应向量−res;
            W = W + δep^T;
            b = b + δp;
            end ← False;
        i = i + 1;
    iteration = iteration + 1;
if end = False then
    print(没有收敛);
print(iteration);
return W and b;
```

运行结果：针对不同的学习速率（1，0.8，0.5，0.2），相同的数据输入顺序，程序均收敛。结果如下表

学习速率 \ 运行结果	W	b	迭代次数
1	$\begin{bmatrix} -4.0 & -1.0 \\ 1 & -5.0 \end{bmatrix}$	$\begin{bmatrix} -1.0 \\ 1 \end{bmatrix}$	5
0.8	$\begin{bmatrix} -2.4 & -0.8 \\ 0.8 & -4.0 \end{bmatrix}$	$\begin{bmatrix} -0.8 \\ 0.8 \end{bmatrix}$	4
0.5	$\begin{bmatrix} -2.0 & -0.5 \\ 0.5 & -2.5 \end{bmatrix}$	$\begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix}$	5
0.2	$\begin{bmatrix} -0.6 & -0.2 \\ 0.2 & -1.0 \end{bmatrix}$	$\begin{bmatrix} -0.2 \\ 0.2 \end{bmatrix}$	4

分析实验结果：实验结果显示针对不同的学习速率，程序均收敛，且很凑巧的是收敛到的分界面是一样的。而且迭代次数没有发生明显变化。可能是由于训练数据比较特殊，且我每次迭代时都是顺序输入训练数据，导致学习速率没有对其产生多大影响。后面我尝试改变数据输入顺序，虽然分界面改变了，但是四种情况也是迭代次数差不多，且各自分界面相同。所以，应该就是这个例子太小了，条件变化对结果没什么影响。

3. (a) i. on-line learning

考虑到下标的简洁性，这里将层下标省去，并对可能造成疑惑的地方说明。对于BP算法，其算法目标是均方误差最小化。令$\vec{z}$为输入向量，t 为期望输出，x 为实际输出向量(x 也可泛指任意一个神经元的输出)，设定目标函数为

$$F(z) = \frac{1}{2}\sum_{k}^{N_{out}} e_k^2 \tag{1}$$

其中k枚举输出层神经元

$$e_k = t_k - x_k \tag{2}$$

为第k个输出层神经元的输出误差。这里对于单个神经元j推导其权值更新公式。由于目标是均方误差最小化，采用最速下降法，则

$$u_{ji} = u_{ji} - \eta\frac{\partial F(z)}{\partial u_{ji}} \tag{3}$$

$$v_{ji} = v_{ji} - \eta\frac{\partial F(z)}{\partial v_{ji}} \tag{4}$$

u_{ji}, v_{ji}分别代表神经元j权值向量u_j, v_j的第i个分量。所以问题转变成求全局误差对每个神经元权值向量的梯度。利用求导的链式法则，

$$\frac{\partial F(z)}{\partial u_{ji}} = \frac{\partial F(z)}{\partial e_j}\frac{\partial e_j}{\partial x_j}\frac{\partial x_j}{\partial n_j}\frac{\partial n_j}{\partial u_{ji}} \tag{5}$$

$$\frac{\partial F(z)}{\partial v_{ji}} = \frac{\partial F(z)}{\partial e_j}\frac{\partial e_j}{\partial x_j}\frac{\partial x_j}{\partial n_j}\frac{\partial n_j}{\partial v_{ji}} \tag{6}$$

其中，n_j代表神经元j的净输出，x_j代表神经元j的实际输出，即

$$n_j = \sum_{i=1}^{N}(u_{ji}x_i^2 + v_{ji}x_i) + b_j \tag{7}$$

$$x_j = f(n_j) \tag{8}$$

函数$f(.)$表示sigmoid传输函数，b_j为神经元j的偏置，x_i是连接神经元j的神经元（j神经元的前一层），N 为前一层的神经元数量。由于

$$\frac{\partial F(z)}{\partial e_j} = e_j \tag{9}$$
$$\frac{\partial e_j}{\partial x_j} = -1 \tag{10}$$
$$\frac{\partial x_j}{\partial n_j} = f^{'}(n_j) \tag{11}$$
$$\frac{\partial n_j}{\partial u_{ji}} = x_i^2 \tag{12}$$
$$\frac{\partial n_j}{\partial v_{ji}} = x_i \tag{13}$$

结合$(5-6), (9-13)$得

$$\frac{\partial F(z)}{\partial u_{ji}} = -e_j f_j^{'}(n_j) x_i^2 \tag{14}$$
$$\frac{\partial F(z)}{\partial v_{ji}} = -e_j f_j^{'}(n_j) x_i \tag{15}$$

其中，$f_j^{'}(n_j)$和x_i, x_i^2均能直接计算，而对于输出层神经元，$e_j = t_j - x_j$也可直接计算。但是对于隐藏层神经元，没有标准输出参考，所以对于隐藏层神经元j，同样根据求导的链式法则，

$$\frac{\partial F(z)}{\partial u_{ji}} = \frac{\partial F(z)}{\partial x_j}\frac{\partial x_j}{\partial u_{ji}} \tag{16}$$
$$= \frac{\partial F(z)}{\partial x_j} f^{'}(n_j) x_i^2 \tag{17}$$
$$= \sum_k (e_k \frac{\partial e_k}{\partial x_j}) f^{'}(n_j) x_i^2 \tag{18}$$
$$= f^{'}(n_j) x_i^2 \sum_k^N (e_k \frac{\partial e_k}{\partial x_k}\frac{\partial x_k}{\partial n_k}\frac{\partial n_k}{\partial x_j}) \tag{19}$$
$$= f^{'}(n_j) x_i^2 \sum_k^N (-e_k f^{'}(n_k) \frac{\partial n_k}{\partial x_j}) \tag{20}$$

同理

$$\frac{\partial F(z)}{\partial v_{ji}} = \frac{\partial F(z)}{\partial x_j}\frac{\partial x_j}{\partial v_{ji}} \tag{21}$$
$$= \frac{\partial F(z)}{\partial x_j} f^{'}(n_j) x_i \tag{22}$$
$$= \sum_k^N (e_k \frac{\partial e_k}{\partial x_j}) f^{'}(n_j) x_i \tag{23}$$
$$= f^{'}(n_j) x_i \sum_k^N (-e_k f_k^{'}(n_k) \frac{\partial n_k}{\partial x_j}) \tag{24}$$

这里神经元k枚举了神经元j连接的下一层神经元（因为是全连接），注意到隐藏层神经元与输出层神经元求解式子中均存在$-e_k f_k'(n_k)$部分，故定义

$$\delta_j = -e_j f^{'}(n_j) \tag{25}$$

为局域梯度。则对于输出层神经元，式(14)、(15)可写成

$$\frac{\partial F(z)}{\partial u_{ji}} = \delta_j x_i^2 \tag{26}$$

$$\frac{\partial F(z)}{\partial v_{ji}} = \delta_j x_i \tag{27}$$

对于隐藏层神经元，式(20)、(24)可写成

$$\frac{\partial F(z)}{\partial u_{ji}} = f^{'}(n_j) x_i^2 \sum_k^N (\delta_k \frac{\partial n_k}{\partial x_j}) \tag{28}$$

$$\frac{\partial F(x)}{\partial v_{ji}} = f^{'}(n_j) x_i \sum_k^N (\delta_k \frac{\partial n_k}{\partial x_j}) \tag{29}$$

$$\tag{30}$$

因为

$$n_k = \Big(\sum_{i=1}^N (u_{ki} x_i^2 + v_{ki} x_i) + b_k) \Big) \tag{31}$$

所以

$$\frac{\partial n_k}{\partial x_j} = 2u_{kj} x_j + v_{kj} \tag{32}$$

所以隐藏神经元的局部梯度为

$$\delta_j = f^{'}(n_j) \sum_k^N (\delta_k (2u_{kj} x_j + v_{kj})) \tag{33}$$

所以得隐藏神经元全局误差对权值向量的梯度为

$$\frac{\partial F(z)}{\partial u_{ji}} = \delta_j x_i^2 = f^{'}(n_j) x_i^2 \sum_k^N (\delta_k (2u_{kj} x_j + v_{kj})) \tag{34}$$

$$\frac{\partial F(z)}{\partial v_{ji}} = \delta_j x_i = f^{'}(n_j) x_i \sum_k^N (\delta_k (2u_{kj} x_j + v_{kj})) \tag{35}$$

k枚举的是神经元j的下一层，下一层神经元数量为N。
对于偏置b_j，

$$\frac{\partial F(z)}{\partial b_j} = \frac{\partial F(z)}{\partial e_j} \frac{\partial e_j}{\partial x_j} \frac{\partial x_j}{\partial n_j} \frac{\partial n_j}{\partial b_j} \tag{36}$$

$$= -e_j f^{'}(n_j) \tag{37}$$

$$= \delta_j \tag{38}$$

同理，需要区分输出层和隐藏层，更新公式即为

$$b_j = b_j - \eta \frac{\partial F(z)}{\partial b_j} \tag{39}$$

综上，BP算法先前向传播输入，得到输出，再根据式(34)反向传播局域梯度，最后再利用每一个神经元的局域梯度根据式(26-27)(34-35) 求得全局误差对单个神经元的权值梯度，利用式(3-4)更新对应的权值。完成一个数据输入的权值更新过程。

对于on-line过程，用上述方法和公式对于每一个训练数据依次执行BP算法即可。

ii. batch learning:

对于batch learning过程，同样输出数据，前向传播数据，反向求梯度，但是不马上更新权值。而是保存下来。则对于某个神经元，经过N个数据训练后，将得到2N个梯度$\{\nabla_{1i}, \nabla_{2i}\}_{i=1}^{N}$，之后做一次更新操作，令

$$u_{ji} = u_{ji} + \eta \frac{1}{N} \sum_{i=1}^{N} \nabla_{1i} \tag{40}$$

$$v_{ji} = v_{ji} + \eta \frac{1}{N} \sum_{i=1}^{N} \nabla_{2i} \tag{41}$$

(b) i. 采用三层网络，输入层-隐藏层-输出层

ii. 传输函数采用logistic函数

程序采用python3结合numpy库实现，下面是伪代码

Algorithm 2: 反向传播BP网络

input : 训练数据集data和测试数据集，格式:（数据item，所属类别target）

output: 网络各个神经元权重及对测试数据集的测试结果

$rate \leftarrow 1.0$;
$iteration \leftarrow 0$;
读取文件到data;
while $rate > 0.0001$ ***and*** $iteration < 150$ **do**
  $i \leftarrow 0$;
  $cost \leftarrow 0$;
  while $i < len(data)$ **do**
    前向传播，计算输出res;
    计算error;
    $cost \leftarrow cost + error^2$;
    反向计算局域梯度和梯度;
    更新各个神经元权值和偏置;
    $i = i + 1$;
  比较前后cost，计算rate;
  $iteration = iteration + 1$;
print(iteration);
将测试数据输入，统计正确率。return W and b;

(c) 程序中，设置收敛条件为每一回合的均方误差变化的绝对速率小于0.01%时或者迭代次数大于300次时收敛。通过设置不同的学习速率，运行结果如下表：

运行结果 学习速率	训练时间(s)	迭代次数	测试正确率
1.0	2.33	50	65.63%(63/96)
0.8	6.62	139	78.13%(75/96)
0.5	9.97	215	93.75%(90/96)
0.2	14.04	300	76.04%(73/96)

可以看出随着学习速率的降低，训练时间加长，迭代次数增加。测试的正确率先稳步上升，最后在学习速率为0.2的时候下降。