目录

第一章 算法描述

第一节 BP 神经网络

训练 BP 神经网络的伪代码如下所示：

```
1：  bias_sum ← 0
2：  repeat
3：      for p in testset
4：          bias ← Bp(p)
5：          bias_sum += bias
6：      update_matrix ← bias_sum
7：      end for
8：  until meet termination conditions
```

将测试集合的每一次测试作为 BP 算法的输入进行一次 BP 操作，得到对权重矩阵的改变值存入 bias，直到所有的测试集都被输入一次，得到权重改变之和，这时候再对权重矩阵进行更新操作。重复迭代上述过程，迭代次数可自行修改。

实现 BP 操作的伪代码如下：

```
1：  initial_ANN
2：  output ← predict(case)
3：  error ← f(expect, output)
4：  for all hidden nodes and output node
5：      for all edges connect the node
6：          ∇w_ij = (−1) * η * ∂E/∂I * ∂I/∂w_ij
7：          Bias[i][j]=∇w_ij
8：      end for
9：  end for
```

首先初始化一个神经网络，权重矩阵随机生成，对于输入值进行一次前馈操作(此操作比较简单不再展示伪代码)，前馈操作依据公式 $y = f(\sum_{i=1}^{n} w_i x_i)$. 每个节点的 output 通过上述公式得到，其中的w_i为该结点输入边的权重，x_i为与和该节点相连的上层节点的输出，其中的f函数我们采用 sigmoid 函数。通过对一层结点的输出值进行计算最终得到最后输出节点的输出值。进行一次前馈操作后，我们可以将正确结果与神经网络输出值进行比较得到误差值。然后通过该误差值，一层一层循序渐进地算出所有边权的改变值。具体算法如下：

对于输出节点，与输出结点相连接的边的改变值计算公式为:$\nabla w_{ij} = -\eta * \delta_j * \frac{\partial I}{\partial w_{ij}}$其中 $\delta_j = \frac{\partial E}{\partial O} * \frac{\partial O}{\partial I}$. 隐藏节点$\delta_m$值的计算可以通过已算出的节点的δ值得出,具体公式为：$\delta_m = \sum_{i=1}^{n} \delta_i * w_{mi} * f'(O_m)$， δ_i为所有与该节点连接的上层节点的δ值，w_{mi}为所有与该节点连接的上层节点之间的权重值，O_m为该节点的输出值。根据上述公式依次算出所有边权的改变值。将这些改变值存入一个矩阵中以便后续对矩阵进行更新操作。

尝试了 Adam 算法，发现效果还不如传统 BP 算法，仍然跳不出局部最优，准确率最高只能达到 30/32.

第二节 遗传算法

采用遗传算法来优化神经网络，首先来介绍一下遗传算法。遗传算法的伪代码如下图所示：

```
1:  P←initial_population(P, sizepop)
2:  repeat
3:      Newpop ← ∅
4:      Mutate_pop ← ∅
5:      Sum_pop ← ∅
6:      Best ←selectmax(P, f)
7:      repeat for sizepop times
8:          Mutate_pop ← Mutate_pop ∪ {mutate(Best)}
9:      end
10:     Sum_pop=Mutate_pop + P
11:     Newpop ← selectmax_5(Sum_pop)
12:   until f(V) = max_fitness for some V contained in P
13:   return V
```

第 1 行伪代码中的 initial_population 用来随机初始化种群；在第 3 到第 5 行伪代码中，*Newpop* 用来存放新产生的种群，Mutate_sizepop 用来存放变异产生的几个个体，Sum_sizepop 是变异产生个体和原种群个体合并后的集合；第 6 行伪代码是通过适应性函数来选择最优个体；第 7 到 9 行伪码是对最优个体进行变异操作，变异 sizepop 次产生的变异个体存放在 Mutate_sizepop 中。接下来将 Mutate_sizepop 与原种群合并产生 *Sum*_sizepop，并且从这几个个体中通过比较适应性函数值选择 sizepop 个值最大的个体组成一个新的种群。重复上述操作直到最优个体达到适应性函数最大值或者迭代次数超过规定次数。最后 return 最优个体。

下面对遗传算法里的几个重要概念进行说明：

个体：将 BP 神经网络里的初始权重矩阵作为个体，BP 神经网络已经在上一章节进行了介绍。

变异操作：变异操作是通过适应性函数选择最优个体后，对最优的这个权重矩阵，矩阵中的每个权重加上一个随机扰动，得到变异后的个体。

适应性函数：每个个体即 BP 神经网络的初始化权重矩阵，适应性函数值就是通过此初始化矩阵训练出来的 BP 神经网络通过的测试用例的个数。

第二章 实验结果分析

将参数设置为 bp 算法中的学习率设置为 0.3，bp 算法迭代 50 次。遗传算法中的种群个数 sizepop 设置为 5，种群进化次数设置为 5000。

进行 10 次实验结果如下：

序号	1	2	3	4	5	6	7	8	9	10	平均
成功个数	30	32	30	30	30	32	31	30	30	30	30

表 1

实验得到的最好的神经网络示意图及权重矩阵如下：

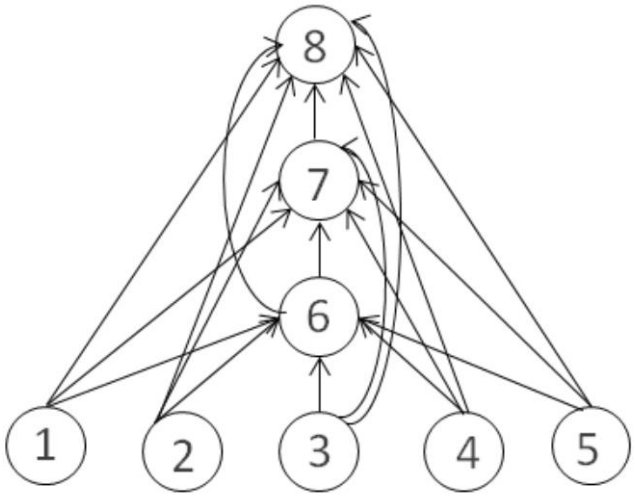

Figure 1

	T	1	2	3	4	5	6	7
6	-1.6	-8.2	-14.1	7.5	-12.9	7.8	0	2.7
7	9.8	6.6	6.3	-7.2	6.0	-7.1	15.3	0
8	-2.5	-6.2	-6.7	6.1	-7.5	6.7	-13.7	14.1

Table 1

第三章 程序不足和改进

首先分析一下程序的正确率，造成达不到百分百正确的原因是：BP 算法总是找到局部最小解，达不到全局最优；可以将 BP 算法进行改进，将学习率不断改变，采用 MBP 算法；在程序中使用爬山算法，扩大范围找到最优解；在遗传算法中的变异操作中，可以考虑采用适应性函数值越大，被选中的概率越大的方法，在本算法中直接选中了适应性函数值最大的个体。

其次分析下程序运行时间过长的问题：权重矩阵存于矩阵中，在运用 BP 算法进行反馈时，由于对矩阵值进行运算，使用了大量的 for 循环消耗了大量时间；代码不够精炼，有很多冗余操作；初始化新的变量过多，可以将代码进行优化。