100502205 資工三 B 楊翔雲

程式使用說明

程式問題

「放射狀基底函數網路 RBF」，基本上，其網路架構如圖 1 所示，為兩層的網路；假設輸入維度是 p ，以及隱藏層類神經元的數目是 J ，那麼網路的輸入可以表示成：

$$F(\underline{x}) = \sum_{j=1}^{J} w_j \varphi_j(\underline{x}) + \theta$$

$$= \sum_{j=0}^{J} w_j \varphi_j(\underline{x}) \qquad (3.24)$$

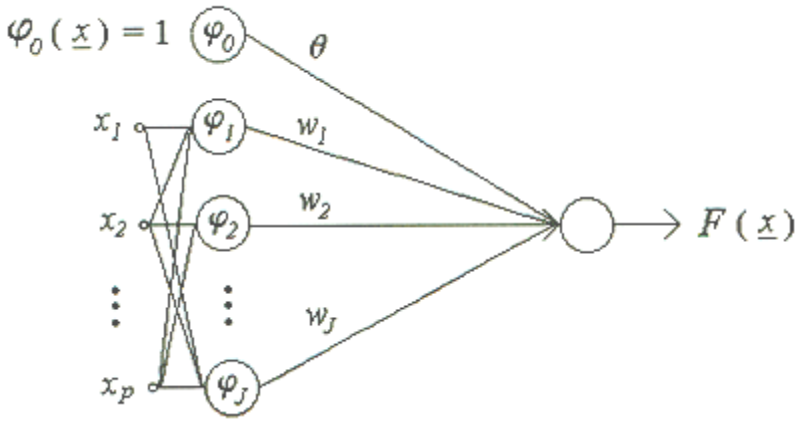


圖 1：放射性基底函數網路的架構。

其中選用高斯型基底函數：

$$\varphi_j(\underline{x}) = \exp\left(-\frac{\left\|\underline{x} - \underline{m}_j\right\|^2}{2\sigma_j^2}\right)$$

其中 $\underline{x} = (x1,x2,\ldots,xp)$ 、$\underline{m}_j = (m_{j1},m_{j2},\ldots,m_{jp})$ 、‖ ‖ 代表向量絕對值

適應函數為：

$$E(n) = \frac{1}{2}\sum_{1}^{N}(y_n - F(\underline{x}_n))^2 \qquad (1)$$

請用實數型基因演算法，找出 w_j , m_j , σ_j，在不同的數字 J 下，最好的基因向量 (例如 J 為 9、輸入 x 為 3 維向量，則表示基因向量是 1+9+3x9+9=46 維度的向量，請注意

這裡不是指族群數；又例如 J 為 7、輸入 x 為 3 維向量，則表示基因向量是 1+7+3x7+7=36 維度的向量)下，評估函數 E(式 1)為越小越好。其中基因向量維度公式為 1+J + p*J+J = (p+2)*J+1 維向量(θ,w_1 , w_2 ,.., w_J , m_{11}, m_{12},.., m_{1p}, m_{21}, m_{22},.., m_{2p},.., m_{J1}, m_{J2},…, m_{Jp}, σ_1, σ_2, ..., σ_J)。

參數說明：

N：作業 1 產生的 N 筆成功到達目的訓練資料(換不同起始點)

y_n：表示訓練資料的方向盤期望輸出值

<P.s.如果配合 wj 值的範圍為 0~1 之間，在此則必須把 yn 由-40~+40 度正規化到 0~1 之間；如果不想正規化　就必須把 wj 的值範圍調整到 -40~40 之間>

θ範圍為 0~1 之間;

Wj 範圍為 0~1 (或是-40~40)之間

<P.s.此需配合訓練集的 yn 跟 F(n)值範圍，所以皆需正規化到 0~1 之間；若不正規化，wj 的值範圍為 -40~40 之間>

mj 範圍跟 X 範圍一樣，如以提供的範例檔則為 0~30；

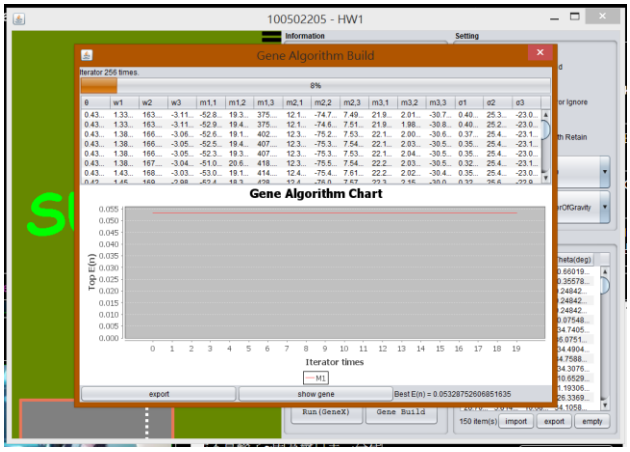
σ j 範圍為 0~10 之間；也可以設定更大的範圍做探討。

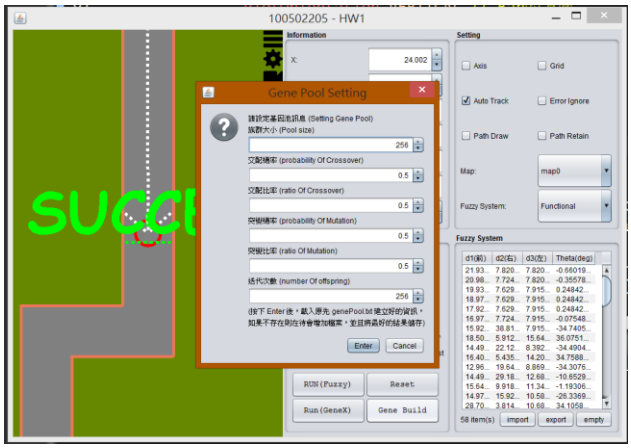
由於 $\varphi_j(\underline{x})$的值介於 0~1 之間，故 需把 輸出的 F(x) 由 -40~+40 度 正規化到 0~1 之間，再去計算，而期望輸出值 yn 也必須調整到 0~1 之間來做訓練。如沒有正規化 0~1，則把 Wj 範圍：-40~40 來解決(而不是 0~1 之間)。

需要有複製，輪盤式選擇，競爭式選擇，交配，突變。
須設定疊代次數，族群大小，突變機率，交配機率。

最後訓練完成的 F(x)當作規則 請跑出車子軌跡。

程式開啟時的畫面





執行環境

- JRE 1.7
- 如有發生短暫延遲問題，這將會導致模擬數據誤差，請將 Submit 面板中的 Stop – Normal – Fast 滾動條調至 Normal 數值以下。
- 測試於 Win 8, Mac OS X 執行正常。
- 若有其他問題請 e-mail 聯絡 morris821028@gmail.com

如何運行

1. 根據前一次作業的說明，可以即時導入測試資料，並且著手使用 gene algorithm(GA) 去完成 RBF 的訓練。
2. 可以從本地端將已知的結果，將其匯入訓練。
3. 對於每次運行最好的結果，將會存入 "genePool.txt" 中。如果想要本次運行的所有迭代最佳結果，請在表單中選擇 "export" 進行匯出 .txt。

更多支持

1. 多台車輛的設計，目前由於多執行緒的處理會動的障礙物上存有疑慮。
2. 可以開啟源代碼進行匯入地圖操作。
   地圖撰寫格式：
   基底多邊形的點個數 n，接下來會有 n 個浮點數座標值，接著會有一個整數(0/1)表示是否有終點線，若有終點線，接下來會有兩個浮點數座標值。
   接著將會有數個簡單多邊形的障礙物輸入。

實驗方法

1. 核心代碼，對於每次迭代，著手適應、選擇、交配、突變。

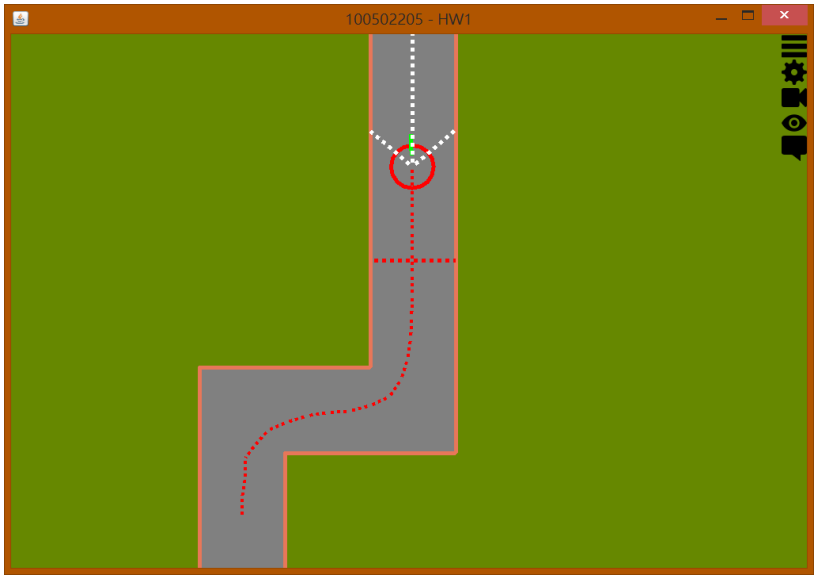
```
GenePair[] A = new GenePair[gene.length];
for (int i = 0; i < gene.length; i++) {
  double f = gene[i].calcuateFitness(dataOutput, dataInput);
  // small better than large. 計算適應程度。
  A[i] = new GenePair(f, gene[i]);
}
Arrays.sort(A); // 對適應程度由好排到壞。
double bestF = A[0].f;
Random intRand = new Random();
double[] copyP = new double[A.length];
double sumF = 0;

for (int i = 0; i < A.length; i++) {
  sumF += 1.0 / A[i].f;
}
// 計算複製的機率
for (int i = 0; i < A.length; i++) {
  copyP[i] = (1.0 / A[i].f) / sumF;
```

```
  }
  // 特別將最好的基因多複製幾次。
  int bestClone = poolSize / 10;

  for (int i = 0; i < bestClone; i++) {
    newGene[i] = A[0].gene.clone();
  }
  // 運行輪盤式選擇
  for (int i = 0, j = bestClone; j < A.length; i = (i + 1) % A.length) {
    double p = Math.random() * 10;
    if (copyP[i] > p) {
          newGene[j++] = A[i].gene.clone();
    }
  }
  for (int i = 0; i < A.length; i++) {
    gene[i] = newGene[i];
   }
  int reserve = 0; // 進行交配
  for (int i = 0; i < A.length; i++) {
    if (Math.random() < this.probabilityOfCrossover) { // 交配機率
          int x = Math.abs(intRand.nextInt()) % (A.length - reserve)
                + reserve; // 隨機與壞的基因做交配
          geneCrossover(i, x, gene[i], gene[x]);
    }
  }

  for (int i = 0; i < A.length; i++) {
    gene[i] = newGene[i];
  }

  for (int i = reserve; i < A.length; i++) {
    double p = Math.random();
    if (p < this.probabilityOfMutation) { // 突變機率
          geneMutation(gene[i]); // 運行突變
    }
  }

  for (int i = 0; i < A.length; i++) {
```

- gene[i].on(); // 啟動基因。
- }
- return bestF;

2. 細部"交配"代碼，採用實數型的分離和聚集兩種情況。
   - double ratio = (Math.random() - 0.5) * 2 * this.ratioOfCrossover;
   - Gene nx = new Gene(), ny = new Gene();
   - for (int i = 0; i < xg.getDNA().length; i++) {
     - nx.getDNA()[i] = xg.getDNA()[i] + ratio * (xg.getDNA()[i] - yg.getDNA()[i]);
     - ny.getDNA()[i] = yg.getDNA()[i] - ratio * (xg.getDNA()[i] - yg.getDNA()[i]);
   - }
   - newGene[x] = nx;
   - newGene[y] = ny;
3. 細部"突變"代碼，採用微量雜訊，唯一不同的地方在於，這個微量雜訊會根據當前數值的比例有關，為了避免大數字對於微量雜訊的干擾不夠強烈。
   - double ratio = (Math.random() - 0.5) * 2 * this.ratioOfMutation;
   - for (int i = 0; i < g.getDNA().length; i++) {
   - if (Math.random() < ratio)
   - g.getDNA()[i] = g.getDNA()[i] + ratio * Math.random() * g.getDNA()[i];

實驗結果



1. 運行結果 J = 3, p = 3

   (θ, w_1 , w_2 ,.., w_J , m_{11}, m_{12},.., m_{1p}, m_{21}, m_{22},.., m_{2p},.., m_{J1}, m_{J2},..., m_{Jp}, σ_1, σ_2, ..., σ_J) = (0.163568 0.851471 1.594151 1.379580 13.092824 0.000505 12.611789 30.000000 0.000368 18.715240 0.000850 2.708967 30.000000 6.307677 7.584529 10.000000)

2. 訓練數據為 "map2" 走一圈的訓練資訊，約為 500 筆。因為 "map0" (即本次作業給的地圖) 的複雜度不夠以至於無法充分表達原本設計的模糊系統。也就是根據當初模糊系統設計，收集的數據的多樣性和連續性不足。

實驗分析

根據 RBF 的神經元，這裡可以越少越好，在程式中，神經元個數(J) 設置 3 個。運行時採用輪盤式選擇，讓適應能力好的，具有較高的機會繁殖，採用一個隨機變數去挑選。而在基因方面使用實數型基因的形式。

1. 運行時，突變機率和突變比例相當重要，由於相同適應能力好的物種量很多，只保留其中一部分即可，因此可以將突變機率 0.5 左右，太低則會造成進化速度太慢，太高則容易失去原本適應好的物種，導致整體適應度的震盪。
2. 另外設置突變的比例，也就是該段實數值上下調整的比例。

```
g.getDNA()[i] = g.getDNA()[i] + ratio * Math.random() * g.getDNA()[i];
```

藉由上述的式子，將 ratio 設置成一個調變比例，來製造爆炸效應的規模。而在運行時，突變效果還能接受。

3. 原本運行時，只將 DNA 片段的值任意放置，並且約束在不會超出函數的定義域，在收斂速度上有明顯加速。但為了符合作業需求，將每個參數約束在指定範圍內，在收斂速度慢了一截，在預期結果並沒有很好。
4. 在不同地圖收集的預期資訊，會針對不同車子感測器的角度有所差異，因此不能拿不同型的感測數據，訓練出來的 RBF 不能給另外一台車子來使用，除非使用的感測器角度相當接近。
5. 對於死路的轉彎，在神經元個數(J) 等於 2 的時候，運行結果較為不佳，但是在本次作業中，並不會有這種數據的出現，也不用考慮這種情況。但是當神經元個數少時，GA 算法的運行速度是相當快的。