

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DIRETORIA DE PESQUISA E PÓS GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

JÔNATAS TRABUCO BELOTTI

IMPLEMENTAÇÃO REDE NEURAL PERCEPTRON

RELATÓRIO

PONTA GROSSA
2017

JÔNATAS TRABUCO BELOTTI

IMPLEMENTAÇÃO REDE NEURAL PERCEPTRON

Relatório apresentado como requisito parcial à obtenção de nota na disciplina de Fundamentos de Redes Neurais Artificiais do Programa de Pós-Graduação em Engenharia Elétrica, da Universidade Tecnológica Federal do Paraná–Campus Ponta Grossa.

Professor: Prof. Dr. Sérgio Okida

PONTA GROSSA

2017

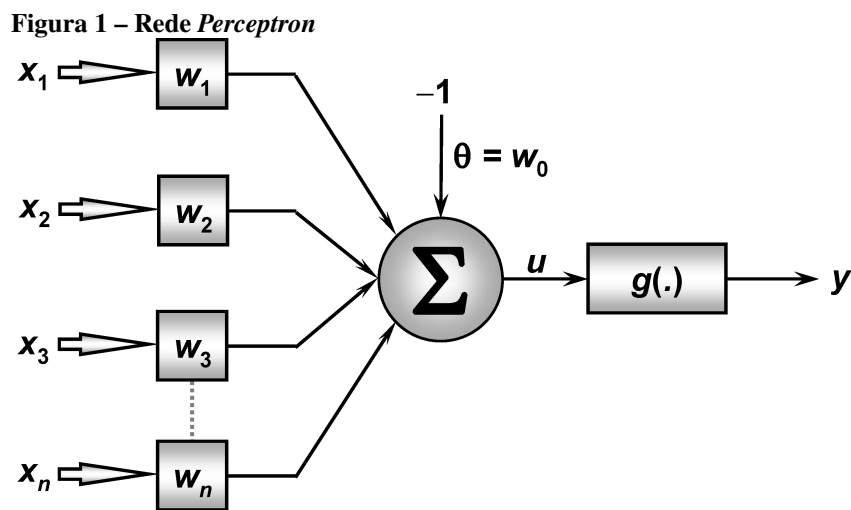
SUMÁRIO

1 INTRODUÇÃO	3
1.1 ESTUDO DE CASO	3
2 DESENVOLVIMENTO ESTUDO DE CASO	5
2.1 IMPLEMENTAÇÃO E TESTE	5
2.1.1 Inicialização dos pesos sinápticos	5
2.1.2 Ajuste dos pesos sinápticos	6
2.1.3 Função degrau bipolar	8
2.1.4 Generalização	9
2.2 TREINAMENTO	10
2.3 EXECUÇÃO	10
2.4 DISCUSSÕES	12
3 CONCLUSÃO	13
REFERÊNCIAS	14
APÊNDICE A - Implementação da classe <i>Perceptron</i>	15
ANEXO A - CONJUNTO DE TREINAMENTO	20

1 INTRODUÇÃO

A rede neural mais simples conhecida é a *Perceptron*, idealizada por Rosenblatt (1958), seu propósito era implementar um modelo computacional baseado no funcionamento da retina (da Silva; SPATTI; FLAUZINO, 2010).

O *Perceptron* é constituído por apenas 1 neurônio artificial que recebe todos os sinais de entrada e devolve uma única saída. A Imagem 1 ilustra a arquitetura de uma rede *Perceptron*.



Fonte: da Silva, Spatti e Flauzino (2010).

Na Figura 1 é possível ver os sinais de entrada $\{x_1, x_2, x_3, \dots, x_n\}$, os pesos sinápticos $\{w_0, w_1, w_2, w_3, \dots, w_n\}$ sendo que o peso sináptico w_0 é referente ao limiar de ativação (θ), o limiar de ativação denotado por θ , o combinador linear denotado por Σ , o potencial de ativação denotado por u , a função de ativação $g(\cdot)$ e por fim o sinal de saída da rede y .

Este relatório tem como objetivo descrever o desenvolvimento do projeto prático 3.6 do livro *Redes neurais artificiais para engenharia e ciências aplicadas* de da Silva, Spatti e Flauzino (2010), o projeto consiste na implementação, treinamento e teste de uma rede neural do tipo *Perceptron* para a classificação de amostras de óleo.

1.1 ESTUDO DE CASO

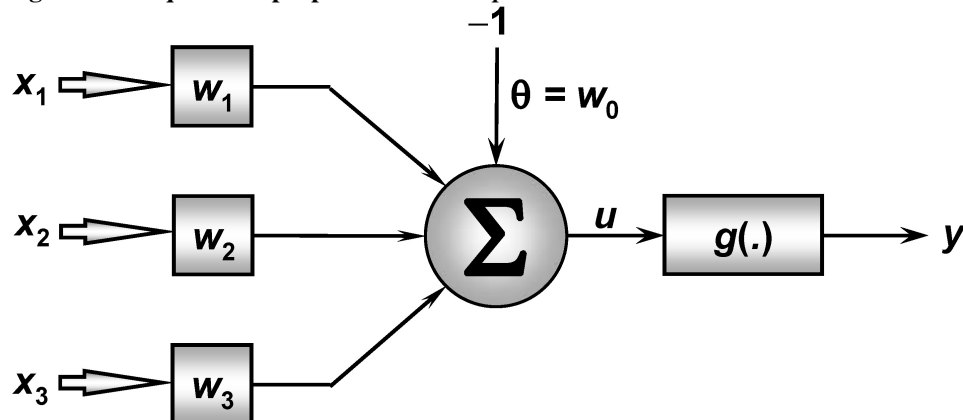
O projeto prático 3.6 do livro *Redes neurais artificiais para engenharia e ciências aplicadas* de da Silva, Spatti e Flauzino (2010), propões o desenvolvimento de uma rede neural *Perceptron* para a classificação de amostras de óleo. As amostras de óleo podem ser classificadas em duas classes de pureza $\{P_1$ e $P_2\}$, essa classificação se dá pela medição de três propriedades físico-químicas do óleo representadas por $\{x_1, x_2$ e $x_3\}$.

O livro trás o conjunto de treinamento da rede, aqui apresentado no Anexo A, e define

o valor -1 para óleo pertencente a classe P_1 e o valor 1 para óleo pertencente a classe P_2 .

Também é descrita a arquitetura que a rede deve ter, possuindo três sinais de entrada $\{x_1, x_2 \text{ e } x_3\}$ com seus respectivos pesos sinápticos $\{w_1, w_2 \text{ e } w_3\}$, sendo os sinais de entrada referentes as três propriedades físico-químicas do óleo. Além disso a rede deve possuir um limiar de ativação θ de valor -1 com um peso sináptico w_0 associado a ele, um combinador linear Σ , o potencial de ativação u , uma função de ativação $g(.)$ e por fim um sinal de saída y que pode assumir os valores de 1 ou -1 . A Figura 2 mostra detalhadamente a arquitetura que a rede deve ter.

Figura 2 – Arquitetura proposta rede *Perceptron*



Fonte: da Silva, Spatti e Flauzino (2010).

Para o treinamento da rede é especificado a utilização do algoritmo supervisionado de *Hebb* para a classificação de padrões com uma taxa de aprendizagem de $0,01$.

2 DESENVOLVIMENTO ESTUDO DE CASO

Esse capítulo apresenta o desenvolvimento do estudo de caso proposto na Seção 1.1, apresentando a implementação da rede na Seção 2.1, o treinamento na Seção 2.2, a execução da rede na Seção 2.3 e a discussão dos resultados obtidos na Seção 2.4.

2.1 IMPLEMENTAÇÃO E TESTE

A rede neural Perceptron foi implementada na linguagem Java, visando facilitar o acesso ao código fonte o mesmo, juntamente com um arquivo contendo o conjunto de dados de treinamento foram disponibilizados em um repositório do *GitHub* que pode ser acessado pelo link <<https://github.com/jonatastbelotti/RedePerceptron>>.

A rede neural foi implementada através da classe *Perceptron*, que é apresentada no Apêndice A. As Seções 2.1.1, 2.1.2, 2.1.3 e 2.1.4 trazem a explicação detalhada dos principais métodos da classe.

2.1.1 Inicialização dos pesos sinápticos

Inicialmente os pesos sinápticos da rede são números aleatórios gerados entre 0 e 1, no Código 1 nota-se, entre as linhas 14 e 17, que o construtor vazio da classe *Perceptron* inicia o valor dos pesos sinápticos através do método *random.nextDouble()*, método esse que gera números aleatórios entre 0 e 1.

Código 1 – Construtor da classe *Perceptron*

```
1 public class Perceptron {
2     private final double PASSO_APRENDIZAGEM = 0.01;

4     private double peso0;
5     private double peso1;
6     private double peso2;
7     private double peso3;
8     private double ultimaResposta;
9     private int numEpocasTreinamento;

10

11     public Perceptron() {
12         Random random = new Random();
```

```

14     this.peso0 = random.nextDouble();
        this.peso1 = random.nextDouble();
16     this.peso2 = random.nextDouble();
        this.peso3 = random.nextDouble();
18     this.ultimaResposta = 0.0;
        this.numEpocasTreinamento = 0;
20 }

```

2.1.2 Ajuste dos pesos sinápticos

O processo de treinamento é implementado na classe *Perceptron* pelo método *treinarRede*, que recebe como parâmetro o arquivo contendo os dados de treinamento e através do algoritmo supervisionado de Hebb atualiza os pesos sinápticos a cada iteração até que não haja mais erro de classificação. O Código 2 apresenta a implementação em Java do método *treinarRede*.

Analizando o código, podemos verificar que o laço que compreende as linhas 18 à 65 realiza várias épocas de treinamento enquanto houver erro de classificação. As linhas 22 a 40 são responsáveis por lerem cada linha do arquivo com os dados de treinamento e retirarem os respectivos sinais de entrada e a saída esperada. Na linha 43 é realizado o cálculo que gera o potencial de ativação. Nessa mesma linha o potencial de ativação já é inserido na função de ativação gerando assim a saída da rede.

Na linha 45 a saída da rede é comparada com a saída esperada para essa entrada, se os valores forem iguais nada é feito, entretanto se os valores forem diferentes, nas linhas 46 a 49 os pesos sinápticos são ajustados.

Código 2 – Método *treinarRede* classe *Perceptron*

```

1 public boolean treinarRede(ArquivoDadosTreinamento
    arquivoTreinamento) {
2     FileReader arq;
    BufferedReader lerArq;
4     String linha;
    String[] vetor;
6     int i;
    int acertou;
8     int maiorAcerto = 0;
    boolean erro = true;
10    double entrada1;
    double entrada2;
12    double entrada3;

```

```

double saidaEsperada;

14
this.numEpocasTreinamento = 0;

16
try {
18
    while (erro) {
        this.numEpocasTreinamento++;
20
        acertou = 0;
        erro = false;
22
        arq = new FileReader(arquivoTreinamento.getCaminhoCompleto
            ());
        lerArq = new BufferedReader(arq);

24

        linha = lerArq.readLine();
26
        if (linha.contains("x1")) {
            linha = lerArq.readLine();
28
        }

30
        while (linha != null) {
            vetor = linha.split("\\s+");
32
            i = 0;

34
            if (vetor[0].equals("")) {
                i = 1;
36
            }

38
            entrada1 = Double.parseDouble(vetor[i++].replace(",", "."));
            entrada2 = Double.parseDouble(vetor[i++].replace(",", "."));
40
            entrada3 = Double.parseDouble(vetor[i++].replace(",", "."));
            saidaEsperada = Double.parseDouble(vetor[i].replace(",",
                "."));

42

            this.ultimaResposta = funcaoDegrauBipolar((entrada1 *
                this.peso1) + (entrada2 * this.peso2) + (entrada3 *
                this.peso3) + (-1.0 * this.peso0));

44

            if (this.ultimaResposta != saidaEsperada) {
46
                peso0 = peso0 + (PASSO_APRENDIZAGEM * (saidaEsperada -
                    this.ultimaResposta) * -1.0);

```



```

        peso1 = peso1 + (PASSO_APRENDIZAGEM * (saidaEsperada -
            this.ultimaResposta) * entrada1);
48      peso2 = peso2 + (PASSO_APRENDIZAGEM * (saidaEsperada -
            this.ultimaResposta) * entrada2);
        peso3 = peso3 + (PASSO_APRENDIZAGEM * (saidaEsperada -
            this.ultimaResposta) * entrada3);
50      erro = true;
    } else {
52      acertou++;
    }

54

    linha = lerArq.readLine();
56  }

58  arq.close();

60  if (acertou > maiorAcerto) {
        maiorAcerto = acertou;
62  Comunicador.addLog("Epoca " + this.numEpocasTreinamento +
        " acertou " + acertou + " (" + peso0 + "; " + peso1 + "; " +
        peso2 + "; " + peso3 + ")");
    }

64

    }

66

    Comunicador.addLog("Fim do treinamento.");
68  } catch (FileNotFoundException ex) {
        return false;
70  } catch (IOException ex) {
        return false;
72  }

74  return true;
    }

```

2.1.3 Função degrau bipolar

Como foi definido na Seção 1.1, a saída da rede deve ser -1 para óleo pertencente a classe P_1 e 1 para óleo pertencente a classe P_2 , dessa forma a função de ativação utilizada foi a função degrau bipartido, que segue o funcionamento:

$$f(x) = \begin{cases} -1 & \text{se } x < 0, \\ 0 & \text{se } x = 0, \\ 1 & \text{em } x > 0 \end{cases}$$

Na implementação em Java da rede *Perceptron* a função de ativação do tipo degrau bipartido foi implementada pelo método `funcaoDegrauBipolar` que pode ser visto no Código 3.

Código 3 – Método `funcaoDegrauBipolar` da classe `Perceptron`

```

1 private double funcaoDegrauBipolar(double valor) {
2     if (valor == 0.0) {
3         return 0.0;
4     }

5
6     if (valor < 0.0) {
7         return -1.0;
8     }

9
10    if (valor > 0.0) {
11        return 1.0;
12    }

13
14    return 0.0;
15 }

```

Perceba que este método foi invocado na linha 43 do Código 2.

2.1.4 Generalização

A etapa de generalização da rede foi implementada através do método `classificar`, que é apresentado no Código 4. Nele verifica-se que a linha 6 é responsável por gerar o potencial de ativação e aplica-lo a função de ativação, gerando assim a saída da rede. Por fim nas linhas 8 a 13 a saída da rede é verificada para imprimir a classificação correspondente.

Código 4 – Método `classificar` da classe `Perceptron`

```

1 public String classificar(double valor1, double valor2, double
2     valor3) {
3     String resposta;

4     resposta = "";
5
6
7
8
9
10
11
12
13

```

```

6      this.ultimaResposta = funcaoDegrauBipolar((valor1 * this.peso1)
          + (valor2 * this.peso2) + (valor3 * this.peso3) + (-1.0 *
              this.peso0));

8      if (this.ultimaResposta == -1.0) {
          resposta = "Classe P1";
10     }
        if (this.ultimaResposta == 1.0) {
12         resposta = "Classe P2";
        }

14     return resposta;
16 }

```

2.2 TREINAMENTO

Com a finalidade de testar a rede neural desenvolvida a mesma foi treinada 5 vezes com o conjunto de treinamento contido no Anexo Anexo A. Em cada treinamento o número de épocas necessárias para treinar a rede foi anotado, assim como o vetor contendo os pesos sinápticos iniciais e finais. O resultado dos 5 treinamentos podem ser visto nos dados da Tabela 1.

Analisando a Tabela 1 é possível verificar que a média do número de épocas necessárias para treinar a rede é de aproximadamente 400 épocas.

2.3 EXECUÇÃO

Para cada um dos 5 treinamentos descritos na Seção 2.2 a rede *Perceptron* foi testada na classificação das amostras de óleo para que os resultados obtidos em cada treinamento pudessem ser comparados. Os resultados obtidos na classificação mediante cada treinamento pode ser visto na Tabela 2.

Note que para a mesma amostra de óleo todos os treinamentos apresentaram a mesma classificação, não havendo divergência entre os resultados.

Tabela 1 – Resultados do treinamento da rede

Treinamento	Vetor de pesos iniciais				Vetor de pesos finais				Nº épocas
	w_0	w_1	w_2	w_3	w_0	w_1	w_2	w_3	
T(1)	0.546383366	0.914173290	0.4531600950	0.831476594	−2.91361663	1.426099290	2.402110095	−0.678903405	351
T(2)	0.805904511	0.2463353299	0.56257503	0.2327497693	−3.014095488	1.463087329	2.443543031	−0.720236230	391
T(3)	0.896602871	0.808698103	0.4878806631	0.2303045852	−3.083397128	1.56789210	2.483794663	−0.735089414	428
T(4)	0.3882159565	0.626372812	0.1246781379	0.1962799425	−3.13178404	1.6000948	2.514918137	−0.745256057	421
T(5)	0.600485862	0.52229995	0.485077395	0.953812771	−3.05951413	1.55214595	2.468979395	−0.730609228	408

Fonte: Autoria própria.

Tabela 2 – Amostras de óleo para validar a rede *Perceptron*

Amostra	x_1	x_2	x_3	y T(1)	y T(2)	y T(3)	y T(4)	y T(5)
1	−0,3665	0,0620	5,9891	P_1	P_1	P_1	P_1	P_1
2	−0,7842	1,1267	5,5912	P_2	P_2	P_2	P_2	P_2
3	0,3012	0,5611	5,8234	P_2	P_2	P_2	P_2	P_2
4	0,7757	1,0648	8,0677	P_2	P_2	P_2	P_2	P_2
5	0,1570	0,8028	6,3040	P_2	P_2	P_2	P_2	P_2
6	−0,7014	1,0316	3,6005	P_2	P_2	P_2	P_2	P_2
7	0,3748	0,1536	6,1537	P_1	P_1	P_1	P_1	P_1
8	−0,6920	0,9004	4,4058	P_2	P_2	P_2	P_2	P_2
9	−1,3970	0,7141	4,9263	P_1	P_1	P_1	P_1	P_1
10	−1,8842	−0,2805	1,2548	P_1	P_1	P_1	P_1	P_1

Fonte: Autoria própria.

2.4 DISCUSSÕES

3 CONCLUSÃO

REFERÊNCIAS

da Silva, Ivan Nunes; SPATTI, Danilo Hernane; FLAUZINO, Rogério Andrade. **Redes Neurais Artificiais Para Engenharia e Cincias Aplicadas - Curso Pratico**. 1. ed. São Paulo: ARTLIBER, 2010. ISBN 978-85-88098-53-4.

ROSENBLATT, Frank. The perceptron: A probabilistic model for information storage and organization in the brain. **Psychological review**, American Psychological Association, v. 65, n. 6, p. 386, 1958.

APÊNDICE A - IMPLEMENTAÇÃO DA CLASSE *PERCEPTRON*

```

1  package Model;
2
3  import View.Comunicador;
4  import java.io.BufferedReader;
5  import java.io.FileNotFoundException;
6  import java.io.FileReader;
7  import java.io.IOException;
8  import java.util.Random;
9
10 /**
11  *
12  * @author Jonatas Trabuco Belotti [jonatas.t.belotti@hotmail.com]
13  */
14 public class Perceptron {
15
16     private final double PASSO_APRENDIZAGEM = 0.01;
17
18     private double peso0;
19     private double peso1;
20     private double peso2;
21     private double peso3;
22     private double ultimaResposta;
23     private int numEpocasTreinamento;
24
25     public Perceptron() {
26         Random random = new Random();
27
28         this.peso0 = random.nextDouble();
29         this.peso1 = random.nextDouble();
30         this.peso2 = random.nextDouble();
31         this.peso3 = random.nextDouble();
32         this.ultimaResposta = 0.0;
33         this.numEpocasTreinamento = 0;
34     }
35
36     public Perceptron(double peso0, double peso1,
37         double peso2, double peso3) {

```



```
38     this.peso0 = peso0;
    this.peso1 = peso1;
40     this.peso2 = peso2;
    this.peso3 = peso3;
42     this.ultimaResposta = 0.0;
    this.numEpocasTreinamento = 0;
44 }

46 public double getUltimaResposta() {
    return ultimaResposta;
48 }

50 public int getNumEpocasTreinamento() {
    return numEpocasTreinamento;
52 }

54 public double getPeso0() {
    return peso0;
56 }

58 public double getPeso1() {
    return peso1;
60 }

62 public double getPeso2() {
    return peso2;
64 }

66 public double getPeso3() {
    return peso3;
68 }

70 public boolean treinarRede(ArquivoDadosTreinamento
    arquivoTreinamento) {
72     FileReader arq;
    BufferedReader lerArq;
74     String linha;
    String[] vetor;
76     int i;
```

```

    int acertou;
78    int maiorAcerto = 0;
    boolean erro = true;
80    double entrada1;
    double entrada2;
82    double entrada3;
    double saidaEsperada;
84
    Comunicador.iniciarLog("Inicio_treinamento_(" + peso0
86        + ";\_ + peso1 + ";\_ + peso2 + ";\_ + peso3 + ")");
    this.numEpocasTreinamento = 0;
88
    try {
90        while (erro) {
            this.numEpocasTreinamento++;
92            acertou = 0;
            erro = false;
94            arq = new FileReader(arquivoTreinamento.getCaminhoCompleto());
            lerArq = new BufferedReader(arq);
96
            linha = lerArq.readLine();
98            if (linha.contains("x1")) {
                linha = lerArq.readLine();
100        }

102        while (linha != null) {
            vetor = linha.split("\\s+");
104            i = 0;

106            if (vetor[0].equals("")) {
                i = 1;
108            }

110            entrada1 = Double.parseDouble(vetor[i++].replace(",", "."));
            entrada2 = Double.parseDouble(vetor[i++].replace(",", "."));
112            entrada3 = Double.parseDouble(vetor[i++].replace(",", "."));
            saidaEsperada = Double.parseDouble(vetor[i].replace(",", "."));
114
            ultimaResposta = funcaoDegrauBipolar((entrada1 * this.peso1)

```

```

116             + (entrada2 * this.peso2) + (entrada3 * this.peso3)
               + (-1.0 * this.peso0));
118
119         if (this.ultimaResposta != saidaEsperada) {
120             peso0 = peso0 + (PASSO_APRENDIZAGEM * (saidaEsperada
               - this.ultimaResposta) * -1.0);
122             peso1 = peso1 + (PASSO_APRENDIZAGEM * (saidaEsperada
               - this.ultimaResposta) * entrada1);
124             peso2 = peso2 + (PASSO_APRENDIZAGEM * (saidaEsperada
               - this.ultimaResposta) * entrada2);
126             peso3 = peso3 + (PASSO_APRENDIZAGEM * (saidaEsperada
               - this.ultimaResposta) * entrada3);
128             erro = true;
129         } else {
130             acertou++;
131         }
132
133         linha = lerArq.readLine();
134     }
135
136     arq.close();
137
138     if (acertou > maiorAcerto) {
139         maiorAcerto = acertou;
140         Comunicador.addLog("Epoca_" + numEpocasTreinamento
               + "_acertou_" + acertou + "_( " + peso0 + ";" +
142             + peso1 + ";" + peso2 + ";" + peso3 + ")");
143     }
144
145 }
146
147 Comunicador.addLog("Fim_do_treinamento.");
148 } catch (FileNotFoundException ex) {
149     return false;
150 } catch (IOException ex) {
151     return false;
152 }
153
154 return true;

```

```
    }  
156  
    public String classificar(double valor1, double valor2,  
158    double valor3) {  
        String resposta;  
160  
        resposta = "Sem_classificacao";  
162  
        this.ultimaResposta = funcaoDegrauBipolar((valor1 * this.peso1)  
164            + (valor2 * this.peso2) + (valor3 * this.peso3)  
            + (-1.0 * this.peso0));  
166  
        if (this.ultimaResposta == -1.0) {  
168            resposta = "Classe_P1";  
        }  
170        if (this.ultimaResposta == 1.0) {  
            resposta = "Classe_P2";  
172        }  
  
174        return resposta;  
    }  
176  
    private double funcaoDegrauBipolar(double valor) {  
178        if (valor == 0.0) {  
            return 0.0;  
180        }  
  
182        if (valor < 0.0) {  
            return -1.0;  
184        }  
  
186        if (valor > 0.0) {  
            return 1.0;  
188        }  
  
190        return 0.0;  
    }  
192  
}
```

ANEXO A - CONJUNTO DE TREINAMENTO

x1	x2	x3	d
-0.6508	0.1097	4.0009	-1.0000
-1.4492	0.8896	4.4005	-1.0000
2.0850	0.6876	12.0710	-1.0000
0.2626	1.1476	7.7985	1.0000
0.6418	1.0234	7.0427	1.0000
0.2569	0.6730	8.3265	-1.0000
1.1155	0.6043	7.4446	1.0000
0.0914	0.3399	7.0677	-1.0000
0.0121	0.5256	4.6316	1.0000
-0.0429	0.4660	5.4323	1.0000
0.4340	0.6870	8.2287	-1.0000
0.2735	1.0287	7.1934	1.0000
0.4839	0.4851	7.4850	-1.0000
0.4089	-0.1267	5.5019	-1.0000
1.4391	0.1614	8.5843	-1.0000
-0.9115	-0.1973	2.1962	-1.0000
0.3654	1.0475	7.4858	1.0000
0.2144	0.7515	7.1699	1.0000
0.2013	1.0014	6.5489	1.0000
0.6483	0.2183	5.8991	1.0000
-0.1147	0.2242	7.2435	-1.0000
-0.7970	0.8795	3.8762	1.0000
-1.0625	0.6366	2.4707	1.0000
0.5307	0.1285	5.6883	1.0000
-1.2200	0.7777	1.7252	1.0000
0.3957	0.1076	5.6623	-1.0000
-0.1013	0.5989	7.1812	-1.0000
2.4482	0.9455	11.2095	1.0000
2.0149	0.6192	10.9263	-1.0000
0.2012	0.2611	5.4631	1.0000