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

JÔNATAS TRABUCO BELOTTI

IMPLEMENTAÇÃO REDE NEURAL ADALINE

RELATÓRIO

PONTA GROSSA
2017

JÔNATAS TRABUCO BELOTTI

IMPLEMENTAÇÃO REDE NEURAL ADALINE

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

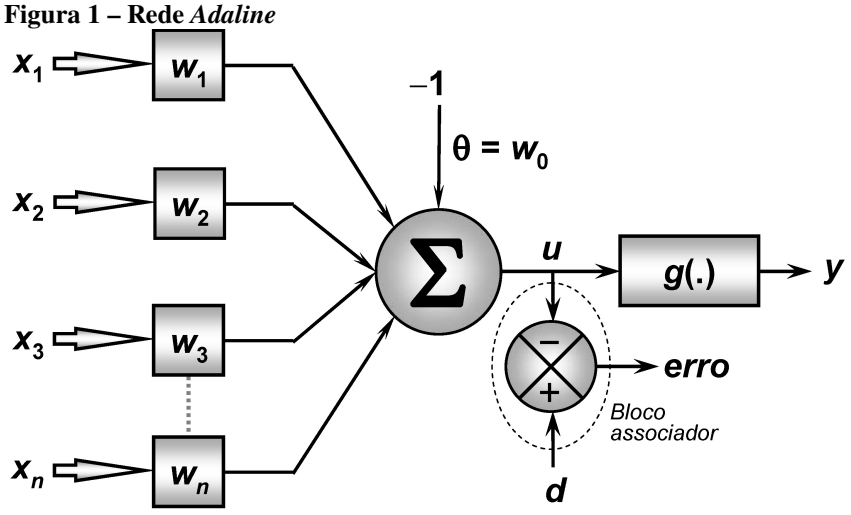
1 INTRODUÇÃO

Em 1960 Widrow e Hoff propuseram a rede neural *Adaline*, sendo amplamente utilizada em sistemas de chaveamento de circuitos telefônicos, foi a primeira rede neural a ser efetivamente utilizada pela industria (WIDROW; HOFF, 1960). Silva, Spatti e Flauzino (2010) relatam que apesar da rede *Adaline* possuir uma arquitetura simples, ela promoveu avanços muito importantes para a evolução das redes neurais, dentre eles:

- Desenvolvimento do algoritmo de aprendizado regra Delta;
- Aplicações em diversos problemas práticos envolvendo processamento de sinais analógicos;
- Primeiras aplicações industriais de redes neurais artificiais.

A arquitetura da rede *Adaline* é similar a da rede *Perceptron*, também sendo constituída por apenas uma camada neural com apenas um único neurônio, diferenciado-se com a presença de um bloco associador que tem como finalidade calcular o erro gerado pela rede para auxiliar o processo de treinamento (SILVA; SPATTI; FLAUZINO, 2010). A Figura 1 mostra a arquitetura da rede *Adaline*.

Figura 1 – Rede *Adaline*



Fonte: (SILVA; SPATTI; FLAUZINO, 2010).

Na Figura 1 é possível ver os sinais de entrada $\{x_1, x_2, x_3, \ldots, x_n\}$, os pesos sinápticos $\{w_0, w_1, w_2, w_3, \ldots, w_n\}$ 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 $\sum$, o potencial de ativação denotado por u, a função de ativação $g(.)$, o bloco associador que recebe o potencial de ativação u juntamente com a saída desejada para cada entrada d gerando assim o erro da rede, e por fim o sinal de saída da rede y.

Este relatório tem como objetivo descrever o desenvolvimento do projeto prático 4.6 do livro Redes neurais artificias para engenharia e ciências aplicadas de Silva, Spatti e Flauzino (2010), o projeto consiste na implementação, treinamento e teste de uma rede neural do tipo *Adaline* para o gerenciamento automático de duas válvulas.

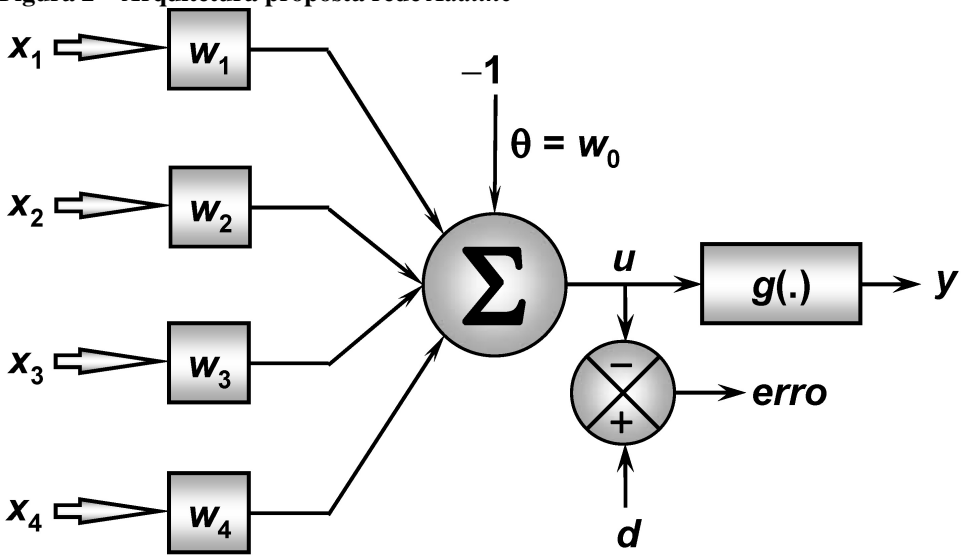
1.1 ESTUDO DE CASO

O projeto prático 4.6 do livro Redes neurais artificias para engenharia e ciências aplicadas de Silva, Spatti e Flauzino (2010), propõe o desenvolvimento de uma rede neural Adaline para o gerenciamento automático de duas válvulas. A partir da análise de 4 grandezas representadas por $\{x_1, x_2, x_3$ e $x4\}$ o Sistema comutador deve decidir se os dados vão para a válvula A ou B.

O livro trás o conjunto de treinamento da rede, aqui apresentado no Anexo A, e define o valor -1 para os sinais que devem ser enviados para a válvula A e o valor 1 para os dados que devem ser enviados para a válvula B.

Também é descrita a arquitetura que a rede deve ter, possuindo 4 sinais de entrada $\{x_1, x_2, x_3$ e $x4\}$ com seus respectivos pesos sinápticos $\{w_1, w_2, w_3$ e $w4\}$. 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 $\sum$, o potencial de ativação u, uma função de ativação $g(.)$, um bloco associador 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 *Adaline*



Fonte: (SILVA; SPATTI; FLAUZINO, 2010).

Para o treinamento da rede é especificado a utilização do algoritmo regra Delta.

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

A rede neural Adaline 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/RedeAdaline>.

A rede neural foi implementada através da classe *Adaline*, que é apresentada no Apêndice A. As Seções 2.1.1, 2.1.2, 2.1.3, 2.1.4 e 2.1.5 trazem a explicação detalhada dos principais métodos da classe.

2.1.1 Inicialização dos pesos sinápticos

Incialmente os pesos sinápticos da rede são números aleatórios gerados entre 0 e 1, no Código 1 nota-se, entre as linhas 18 e 22, que o construtor vazio da classe *Adaline* 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 Adaline

```
1  public class Adaline {
2
3    public static final int NUM_SINAIS_ENTRADA = 4;
4    private final double TAXA_APRENDIZAGEM = 0.0025;
5    private final double PRECISAO = 0.000001;
6
7    private double peso0;
8    private double peso1;
9    private double peso2;
10   private double peso3;
11   private double peso4;
12   private double ultimaResposta;
13   private int numEpocasTreinamento;
```

```
14
  public Adaline() {
16    Random random = new Random();

18    this.peso0 = random.nextDouble();
      this.peso1 = random.nextDouble();
20    this.peso2 = random.nextDouble();
      this.peso3 = random.nextDouble();
22    this.peso4 = random.nextDouble();
      this.ultimaResposta = 0.0;
24    this.numEpocasTreinamento = 0;
  }
```

2.1.2 Ajuste dos pesos sinápticos

O processo de treinamento é implementado na classe *Adaline* pelo método treinarRede, que recebe como parâmetro o arquivo contendo os dados de treinamento e através do algoritmo de aprendizado regra Delta atualiza os pesos sinápticos a cada iteração até que o erro gerado pela rede esteja dentro do limite estabelecido. O Código 2 apresenta a implementação em Java do método treinarRede.

Código 2 – Método treinarRede classe Adaline

```
1  public boolean treinarRede(ArquivoDadosTreinamento
     arquivoTreinamento) {
2      FileReader arq;
       BufferedReader lerArq;
4      String linha;
       String[] vetor;
6      int i;
       double entrada0;
8      double entrada1;
       double entrada2;
10     double entrada3;
       double entrada4;
12     double saidaEsperada;
       double erroAnterior;
14     double erroAtual;

16     this.numEpocasTreinamento = 0;
```

```
18      Comunicador.iniciarLog("Inicio treinamento");
        imprimirSituacao(arquivoTreinamento);
20      Comunicador.addLog("");
        Comunicador.addLog("" + this.numEpocasTreinamento + " " +
           Double.toString(getErro(arquivoTreinamento)).replace(".", ",
           "));
22
        try {
24        do {
            this.numEpocasTreinamento++;
26          erroAnterior = getErro(arquivoTreinamento);
            arq = new FileReader(arquivoTreinamento.getCaminhoCompleto
               ());
28          lerArq = new BufferedReader(arq);

30          linha = lerArq.readLine();
            if (linha.contains("x1")) {
32            linha = lerArq.readLine();
            }
34
            while (linha != null) {
36            vetor = linha.split("\\s+");
              i = 0;
38
              if (vetor[0].equals("")) {
40              i = 1;
              }
42
              entrada0 = -1.0;
44            entrada1 = Double.parseDouble(vetor[i++].replace(",", "."
                 ));
              entrada2 = Double.parseDouble(vetor[i++].replace(",", "."
                 ));
46            entrada3 = Double.parseDouble(vetor[i++].replace(",", "."
                 ));
              entrada4 = Double.parseDouble(vetor[i++].replace(",", "."
                 ));
48            saidaEsperada = Double.parseDouble(vetor[i].replace(",",
                 "."));

50            this.ultimaResposta = (entrada1 * this.peso1) + (entrada2
                  * this.peso2) + (entrada3 * this.peso3) + (entrada4 *
```

```
                this.peso4) + (entrada0 * this.peso0);

52              this.peso0 = peso0 + (TAXA_APRENDIZAGEM * (saidaEsperada
                    - this.ultimaResposta) * entrada0);
                this.peso1 = peso1 + (TAXA_APRENDIZAGEM * (saidaEsperada
                    - this.ultimaResposta) * entrada1);
54              this.peso2 = peso2 + (TAXA_APRENDIZAGEM * (saidaEsperada
                    - this.ultimaResposta) * entrada2);
                this.peso3 = peso3 + (TAXA_APRENDIZAGEM * (saidaEsperada
                    - this.ultimaResposta) * entrada3);
56              this.peso4 = peso4 + (TAXA_APRENDIZAGEM * (saidaEsperada
                    - this.ultimaResposta) * entrada4);

58              linha = lerArq.readLine();
            }
60
            arq.close();
62          erroAtual = getErro(arquivoTreinamento);
            Comunicador.addLog("" + this.numEpocasTreinamento + " " +
                Double.toString(erroAtual).replace(".", ","));
64      } while (Math.abs(erroAtual - erroAnterior) > PRECISAO);

66      Comunicador.addLog("\nFim do treinamento.");
        imprimirSituacao(arquivoTreinamento);
68    } catch (FileNotFoundException ex) {
        return false;
70    } catch (IOException ex) {
        return false;
72    }

74    return true;
}
```

Analisando o código, podemos verificar que o laço que compreende as linhas 26 à 74 realiza várias épocas de treinamento enquanto $\text{Eqm}_{\text{atual}} - \text{Eqm}_{\text{anterior}} > \text{PRECISAO}$, o valor da precisão foi definido em 10^{-6}. As linhas 44 a 48 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 50 é realizado o cálculo que gera o potencial de ativação para a respectiva entrada. Entre as linhas 52 e 56 os pesos sinápticos são ajustados levando em consideração a taxa de aprendizagem, definida em $0,0025$.

O cálculo do Erro quadrático médio é realizado nas linhas 26 e 62 através do método getErro, que tem sua implementação descrita na Seção 2.1.3.

2.1.3 Cálculo do erro quadrático

Em cada iteração do processo de treinamento o erro quadrático médio da rede *Adaline* é medido através da Equação 2.1.

$$\mathrm{E}_{\mathrm{qm}}(\mathbf{w}) = \frac{1}{p}\sum_{k=1}^{p}(d^{(k)} - u)^2 \tag{2.1}$$

Na implementação em Java, esse cálculo é realizado pelo método getErro, método este que é apresentado no Código 3.

Código 3 – Método getErro classe Adaline

```
1  private double getErro(ArquivoDadosTreinamento arquivoTreinamento)
       {
2    FileReader arq;
     BufferedReader lerArq;
4    String linha;
     String[] vetor;
6    int i;
     int numDadosTreinamento;
8    double entrada1;
     double entrada2;
10   double entrada3;
     double entrada4;
12   double saidaEsperada;
     double erro;
14
     try {
16     erro = 0D;
       numDadosTreinamento = 0;
18     arq = new FileReader(arquivoTreinamento.getCaminhoCompleto());
       lerArq = new BufferedReader(arq);
20
       linha = lerArq.readLine();
22     if (linha.contains("x1")) {
         linha = lerArq.readLine();
24     }

26     while (linha != null) {
         numDadosTreinamento++;
28       vetor = linha.split("\\s+");
         i = 0;
```

```
30
        if (vetor[0].equals("")) {
32        i = 1;
        }
34
        entrada1 = Double.parseDouble(vetor[i++].replace(",", "."));
36      entrada2 = Double.parseDouble(vetor[i++].replace(",", "."));
        entrada3 = Double.parseDouble(vetor[i++].replace(",", "."));
38      entrada4 = Double.parseDouble(vetor[i++].replace(",", "."));
        saidaEsperada = Double.parseDouble(vetor[i].replace(",", ".")
           );
40
        this.ultimaResposta = (entrada1 * this.peso1) + (entrada2 *
           this.peso2) + (entrada3 * this.peso3) + (entrada4 * this.
           peso4) + (-1.0 * this.peso0);
42      erro = erro + Math.pow((saidaEsperada - this.ultimaResposta),
            2.0);

44      linha = lerArq.readLine();
      }
46
      arq.close();
48  } catch(IOException | NumberFormatException e) {
      return 0D;
50  }

52  erro = erro / numDadosTreinamento;

54  return erro;
}
```

Perceba que o laço da linha 26 percorre todas as entradas do conjunto de treinamento. Para cada entrada é calculado o potencial de ativação da rede (linha 41) e o seu respectivo erro em comparação com a saída desejada (linha 42).

2.1.4 Função de ativação

Como foi definido na Seção 1.1, a saída da rede deve ser -1 para válvula A e 1 para válvula B, dessa forma a função de ativação utilizada foi a função degrau bipartido, que segue o seguinte funcionamento:

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

Entretanto, como pode ser visto em Silva, Spatti e Flauzino (2010), problemas que envolvem a classificação de padrões, a função degrau bipolar pode ser aproximada de acordo com a Equação 2.2.

$$g(u) = \begin{cases} -1 & \text{se} \quad x < 0, \\ 1 & \text{em} \quad x \geq 0 \end{cases} \tag{2.2}$$

Para a implementação em Java da rede Adaline, foi utilizada uma função de ativação do tipo degrau bipartido que segue a simplificação da Equação 2.2, sua implementação é apresentada no Código 4.

Código 4 – Método funcaoDegrauBipolar da classe Adaline

```
1  private double funcaoDegrauBipolar(double valor) {
2    if (valor < 0.0) {
       return -1.0;
4    }

6    if (valor >= 0.0) {
       return 1.0;
8    }

10   return 1.0;
   }
```

2.1.5 Generalização

A etapa de generalização da rede foi implementada através do método classificar, que é apresentado no Código 5. Nele verifica-se que a linha 6 é responsável por gerar o potencial de ativação e aplica-lo a função de ativação (descrita na Seção 2.1.4), 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 5 – Método classificar da classe Adaline

```
1  public String classificar(double valor1, double valor2, double
       valor3, double valor4) {
2    String resposta;
```

```
4   resposta = "Sem classificacao";

6   this.ultimaResposta = funcaoDegrauBipolar((valor1 * this.peso1) +
        (valor2 * this.peso2) + (valor3 * this.peso3) + (valor4 *
       this.peso4) + (-1.0 * this.peso0));

8   if (this.ultimaResposta == -1.0) {
      resposta = "Valvula A";
10  }
    if (this.ultimaResposta == 1.0) {
12    resposta = "Valvula B";
    }
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 A. Em cada treinamento os pesos sinápticos foram gerados de forma aleatória, para os 5 teinamentos propostos os pesos sinápticos iniciais são apresentados na Tabela 1.

Tabela 1 – Pesos sinápticos iniciais treinamentos rede *Adaline*

Treinamento	**Vetor de pesos iniciais**				
	w_0	w_1	w_2	w_3	w_4
1º(T1)	0, 12297317	0, 93595065	0, 94040583	0, 44155809	0, 097655164
2º(T2)	0, 23826105	0, 69843747	0, 107894106	0, 20899798	0, 410361382
3º(T3)	0, 68583734	0, 70955435	0, 81653310	0, 95195375	0, 132813647
4º(T4)	0, 40362737	0, 9700491	0, 483165134	0, 317028810	0, 075172864
5º(T5)	0, 88800531	0, 47934855	0, 67343039	0, 27201520	0, 182427612

Fonte: Autoria própria.

Em cada treinamento o número de épocas necessárias para treinar a rede foi anotado, assim como o vetor contendo os pesos sinápticos finais. Os resultados dos 5 treinamentos podem ser vistos nos dados da Tabela 2.

Nota-se pela Tabela 2 que a média de épocas necessárias para treinar a rede Adaline com a precisão desejada é de 891, sendo que o 3º treinamento foi o com maior número de épocas, 916, e o 1º treinamento foi o com o menor número de épocas, 860.

Ainda analisando os dados da Tabela 2, verifica-se que independentemente dos valores dos pesos iniciais apresentados na Tabela 1 os pesos finais obtidos pela rede *Adaline* ficaram

Tabela 2 – Pesos sinápticos finais treinamentos rede *Adaline*

T(n)	Vetor de pesos finais					Nº de épocas
	$\mathbf{w_0}$	$\mathbf{w_1}$	$\mathbf{w_2}$	$\mathbf{w_3}$	$\mathbf{w_4}$	
1º(T1)	$-1,8130802$	$1,3129178$	$1,642355$	$-0,42758170$	$-1,1778083$	860
2º(T2)	$-1,8130271$	$1,3128249$	$1,642223$	$-0,42771124$	$-1,1777279$	890
3º(T3)	$-1,81300$	$1,3129484$	$1,642370$	$-0,42744081$	$-1,1778118$	916
4º(T4)	$-1,8130491$	$1,3128822$	$1,6423015$	$-0,4276202$	$-1,177774$	884
5º(T5)	$-1,8130862$	$1,3128883$	$1,642320$	$-0,4276464$	$-1,1777883$	909

Fonte: Autoria própria.

próximos em todos os treinamentos. Tomando como exemplo o peso sináptico w_1, nos 5 treinamentos realizados o valor obtido para esse peso foi $1,3128249$ e o maior valor foi $1,3129484$, realizando a subtração nota-se que nos 5 treinamentos a variação no valor final do peso sináptico foi de $0,0001235$.

2.3 EXECUÇÃO

Para cada um dos 5 treinamentos descritos na Seção 2.2 a rede *Adaline* foi testada na classificação das amostras de sinais para que os resultados obtidos em cada treinamento pudessem ser comparados. Os resultados obtidos na classificação mediante cada treinamento podem ser vistos na Tabela 3.

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

Tabela 3 – Amostras de sinais para validar a rede *Adaline*

Amostra	x_1	x_2	x_3	x_4	y (T1)	y (T2)	y (T3)	y (T4)	y (T5)
1	0.9694	0.6909	0.4334	3.4965	-1	-1	-1	-1	-1
2	0.5427	1.3832	0.6390	4.0352	-1	-1	-1	-1	-1
3	0.6081	-0.9196	0.5925	0.1016	1	1	1	1	1
4	-0.1618	0.4694	0.2030	3.0117	-1	-1	-1	-1	-1
5	0.1870	-0.2578	0.6124	1.7749	-1	-1	-1	-1	-1
6	0.4891	-0.5276	0.4378	0.6439	1	1	1	1	1
7	0.3777	2.0149	0.7423	3.3932	1	1	1	1	1
8	1.1498	-0.4067	0.2469	1.5866	1	1	1	1	1
9	0.9325	1.0950	1.0359	3.3591	1	1	1	1	1
10	0.5060	1.3317	0.9222	3.7174	-1	-1	-1	-1	-1
11	0.0497	-2.0656	0.6124	-0.6585	-1	-1	-1	-1	-1
12	0.4004	3.5369	0.9766	5.3532	1	1	1	1	1
13	-0.1874	1.3343	0.5374	3.2189	-1	-1	-1	-1	-1
14	0.5060	1.3317	0.9222	3.7174	-1	-1	-1	-1	-1
15	1.6375	-0.7911	0.7537	0.5515	1	1	1	1	1

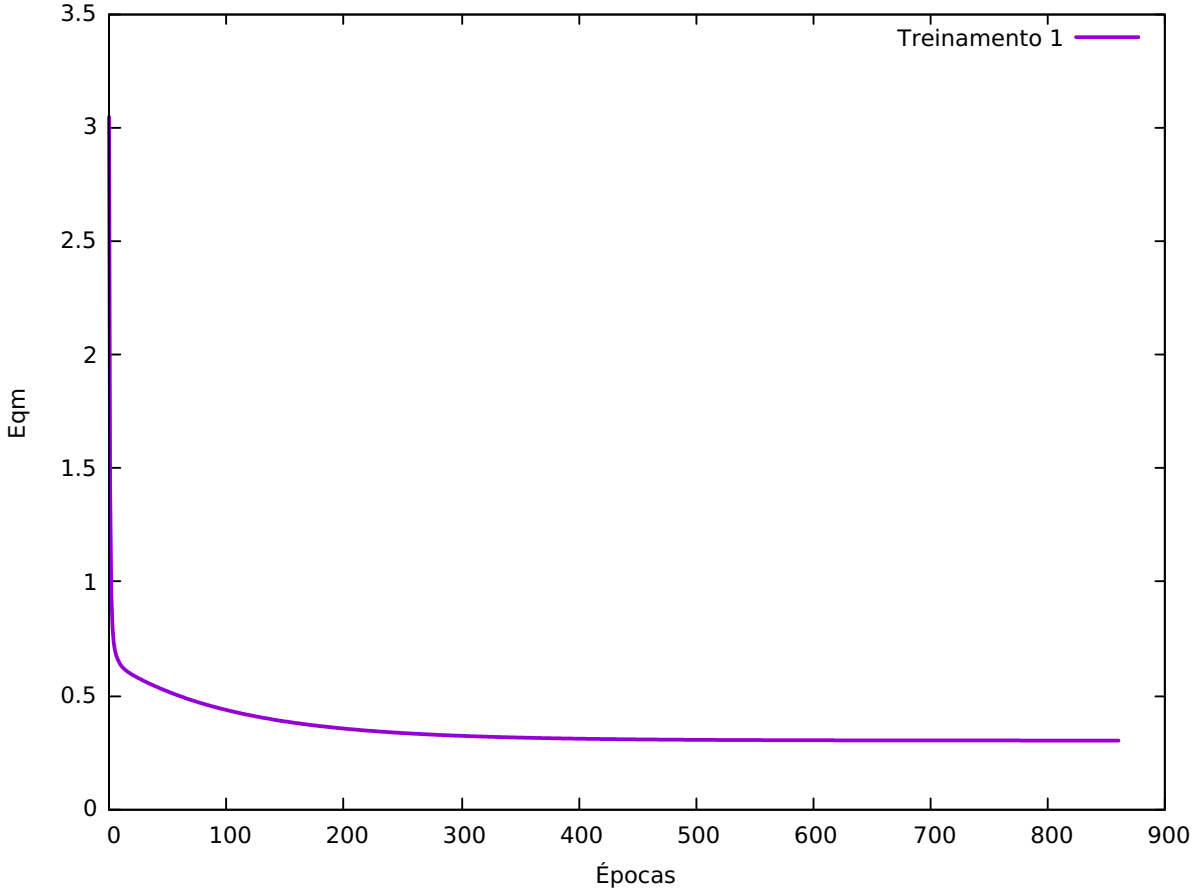
Fonte: Autoria própria.

2.4 DISCUSSÕES

Observando a Tabela 2 nota-se que o número de épocas necessárias para treinar a rede foi diferente em cada um dos 5 treinamentos realizados. Isso ocorre devido aos pesos sinápticos iniciais terem sido gerados aleatoriamente, de forma que quanto mais próximos os pesos estiverem dos valores ótimos mais rápido será o treinamento da rede.

Ainda em relação ao treinamento da rede *Adaline*, o ajuste dos pesos sinápticos realizado em cada iteração do treinamento da rede tem como finalidade a diminuição do erro quadrático médio apresentado pela rede. O gráfico da Figura 3 mostra qual foi o erro quadrático médio em cada época do primeiro treinamento realizado na rede *Adaline*.

Figura 3 – Gráfico E_{qm} treinamento 1



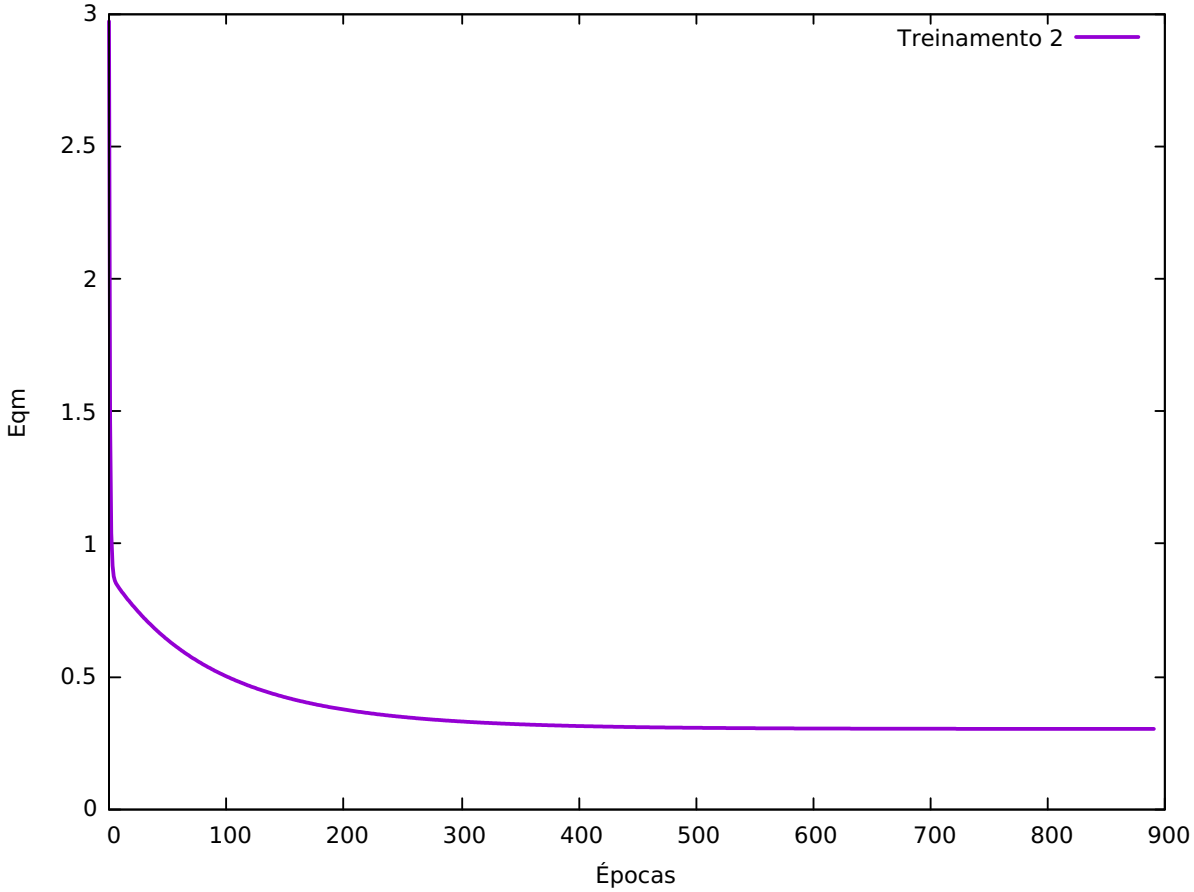
Fonte: Autoria própria.

Observando o gráfico da Figura 3 nota-se que até a época 200, o valor do erro quadrático médio, denotado por E_{qm}, vem sofrendo uma queda acentuada, e a partir da época 200 a queda no valor do erro quadrático médio é amena. A cada época de treinamento o valor que é retirado de erro é menor que o da época anterior, de tal forma que o treinamento da rede chega ao fim quando esse valor é menor que a precisão estabelecida.

O mesmo comportamento pode ser observado no erro quadrático médio da rede Adaline durante o 2º treinamento, apresentado no gráfico da Figura 4. Nele também é possível observar

uma área de queda acentuada no valor do erro quadrático médio até a época 200 e uma área onde essa queda é bem mais amena, de maneira quase estável, a partir da época 200 .

Figura 4 – Gráfico E_{qm} treinamento 2

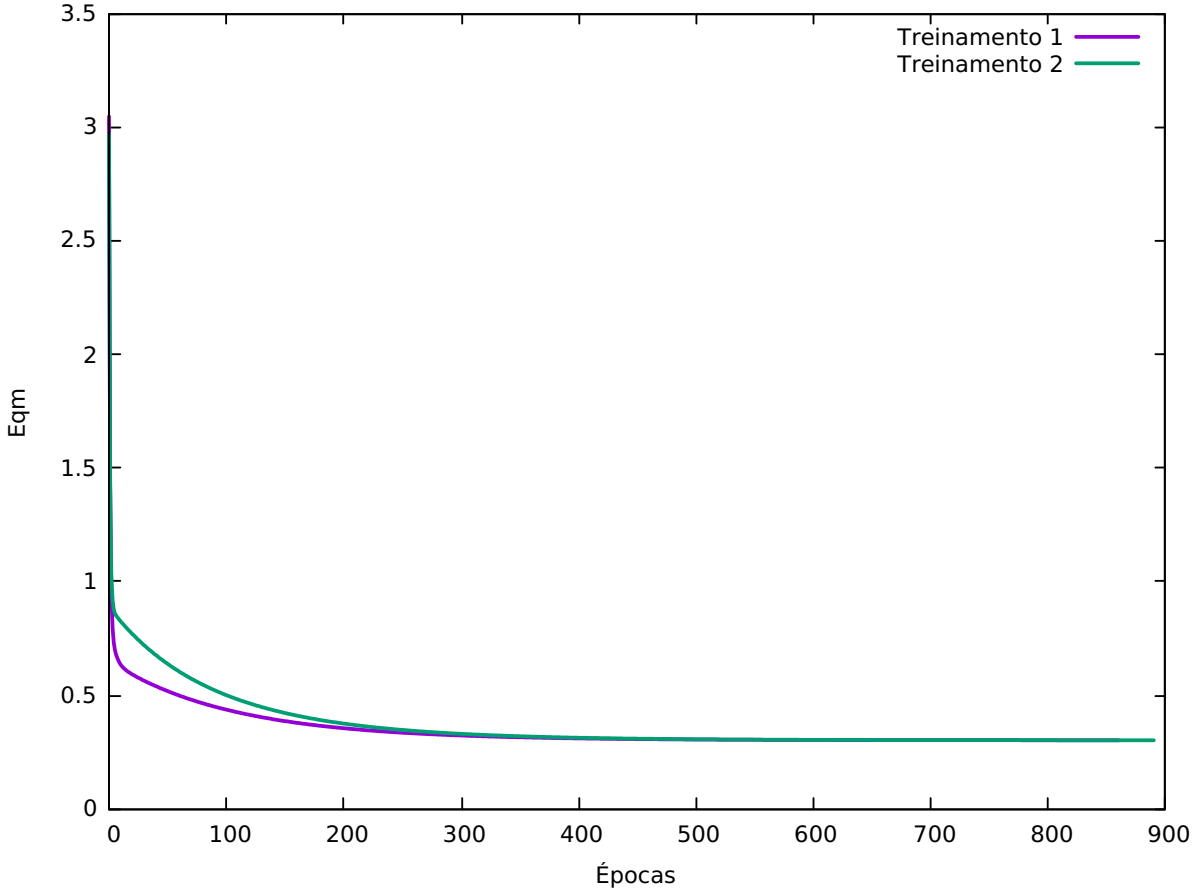


Fonte: Autoria própria.

Tal similaridade é ainda mais visível ao analisar o gráfico da Figura 5, que apresenta a curva do erro quadrático médio para os dois primeiros treinamentos realizados na rede *Adaline*. Perceba que as curvas praticamente se sobrepõe.

A rede *Adaline* desenvolvida nesse trabalho foi capaz de mapear o conjunto de dados utilizado no treinamento de forma que uma mesma entrada não teve duas classificações diferentes em todas as vezes que a rede foi testada, como a rede *Adaline* funciona corretamente apenas para classificações linearmente separáveis pode-se dizer que o conjunto de dados utilizado nesse trabalho é linearmente separável, visto que se isso não fosse verdade em pelo menos 1 teste uma mesma entrada apresentaria duas classificações diferentes.

Figura 5 – Gráfico comparação E_{qm} dos treinamentos



Fonte: Autoria própria.

3 CONCLUSÃO

A rede *Adaline*, apesar de ser um modelo simples de rede neural se mostrou capaz de resolver o problema da classificação de sinais. No que diz respeito aos treinamentos realizados a rede apresentou um número médio de 891 épocas para a realização do treinamento. A limitação da rede *Adaline* em classificar apenas dados linearmente separáveis não foi um problema para este trabalho, como o conjunto de dados tratado aqui é linearmente separável a rede realizou classificações corretas em todos os teste realizados.

REFERÊNCIAS

SILVA, Ivan Nunes da; 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.

WIDROW, Bernard; HOFF, Marcian E. **Adaptive switching circuits**. [S.l.], 1960. Disponível em: <http://www.dtic.mil/get-tr-doc/pdf?AD=AD0241531>. Acesso em: 09 set. 2017.

APÊNDICE A - IMPLEMENTAÇÃO DA CLASSE *ADALINE*

```
1  package Model;
2
   import View.Comunicador;
4  import java.io.BufferedReader;
   import java.io.FileNotFoundException;
6  import java.io.FileReader;
   import java.io.IOException;
8  import java.util.Random;

10 /**
    *
12  * @author Jonatas Trabuco Belotti [jonatas.t.belotti@hotmail.com]
    */
14 public class Adaline {

16   public static final int NUM_SINAIS_ENTRADA = 4;
     private final double TAXA_APRENDIZAGEM = 0.0025;
18   private final double PRECISAO = 0.000001;

20   private double peso0;
     private double peso1;
22   private double peso2;
     private double peso3;
24   private double peso4;
     private double ultimaResposta;
26   private int numEpocasTreinamento;

28   public Adaline() {
       Random random = new Random();
30
       this.peso0 = random.nextDouble();
32     this.peso1 = random.nextDouble();
       this.peso2 = random.nextDouble();
34     this.peso3 = random.nextDouble();
       this.peso4 = random.nextDouble();
36     this.ultimaResposta = 0.0;
       this.numEpocasTreinamento = 0;
```

```
38    }

40    public Adaline(double peso0, double peso1, double peso2,
        double peso3, double peso4) {
42      this.peso0 = peso0;
        this.peso1 = peso1;
44      this.peso2 = peso2;
        this.peso3 = peso3;
46      this.peso4 = peso4;
        this.ultimaResposta = 0.0;
48      this.numEpocasTreinamento = 0;
      }
50
      public double getUltimaResposta() {
52      return ultimaResposta;
      }
54
      public int getNumEpocasTreinamento() {
56      return numEpocasTreinamento;
      }
58
      public double getPeso0() {
60      return peso0;
      }
62
      public double getPeso1() {
64      return peso1;
      }
66
      public double getPeso2() {
68      return peso2;
      }
70
      public double getPeso3() {
72      return peso3;
      }
74
      public double getPeso4() {
76      return peso4;
```

```
    }
78
    public void setPeso4(double peso4) {
80    this.peso4 = peso4;
    }
82
    public boolean treinarRede(ArquivoDadosTreinamento
84    arquivoTreinamento) {
      FileReader arq;
86    BufferedReader lerArq;
      String linha;
88    String[] vetor;
      int i;
90    double entrada0;
      double entrada1;
92    double entrada2;
      double entrada3;
94    double entrada4;
      double saidaEsperada;
96    double erroAnterior;
      double erroAtual;
98
      this.numEpocasTreinamento = 0;
100
      Comunicador.iniciarLog("Inicio treinamento");
102   imprimirSituacao(arquivoTreinamento);
      Comunicador.addLog("");
104   Comunicador.addLog("" + this.numEpocasTreinamento +
        " " + Double.toString(getErro(arquivoTreinamento))
106     .replace(".", ","));

108   try {
        do {
110       this.numEpocasTreinamento++;
          erroAnterior = getErro(arquivoTreinamento);
112       arq = new FileReader(arquivoTreinamento
            .getCaminhoCompleto());
114       lerArq = new BufferedReader(arq);
```

```
116             linha = lerArq.readLine();
                if (linha.contains("x1")) {
118               linha = lerArq.readLine();
                }
120
                while (linha != null) {
122               vetor = linha.split("\\s+");
                  i = 0;
124
                  if (vetor[0].equals("")) {
126                 i = 1;
                  }
128
                  entrada0 = -1.0;
130               entrada1 = Double.parseDouble(vetor[i++]
                  .replace(",", "."));
132               entrada2 = Double.parseDouble(vetor[i++]
                  .replace(",", "."));
134               entrada3 = Double.parseDouble(vetor[i++]
                  .replace(",", "."));
136               entrada4 = Double.parseDouble(vetor[i++]
                  .replace(",", "."));
138               saidaEsperada = Double.parseDouble(vetor[i]
                  .replace(",", "."));
140
                  this.ultimaResposta = (entrada1 * this.peso1) +
142                 (entrada2 * this.peso2) + (entrada3 * this.peso3) +
                    (entrada4 * this.peso4) + (entrada0 * this.peso0);
144
                  this.peso0 = peso0 + (TAXA_APRENDIZAGEM * (saidaEsperada
146                 - this.ultimaResposta) * entrada0);
                  this.peso1 = peso1 + (TAXA_APRENDIZAGEM * (saidaEsperada
148                 - this.ultimaResposta) * entrada1);
                  this.peso2 = peso2 + (TAXA_APRENDIZAGEM * (saidaEsperada
150                 - this.ultimaResposta) * entrada2);
                  this.peso3 = peso3 + (TAXA_APRENDIZAGEM * (saidaEsperada
152                 - this.ultimaResposta) * entrada3);
                  this.peso4 = peso4 + (TAXA_APRENDIZAGEM * (saidaEsperada
154                 - this.ultimaResposta) * entrada4);
```

```
156             linha = lerArq.readLine();
                }
158
                arq.close();
160             erroAtual = getErro(arquivoTreinamento);
                Comunicador.addLog("" + this.numEpocasTreinamento + "␣" +
162               Double.toString(erroAtual).replace(".", ","));
            } while (Math.abs(erroAtual - erroAnterior) > PRECISAO);
164
            Comunicador.addLog("\nFim␣do␣treinamento.");
166         imprimirSituacao(arquivoTreinamento);
        } catch (FileNotFoundException ex) {
168         return false;
        } catch (IOException ex) {
170         return false;
        }
172
        return true;
174     }

176     public String classificar(double valor1, double valor2,
          double valor3, double valor4) {
178       String resposta;

180       resposta = "Sem␣classificacao";

182       this.ultimaResposta = funcaoDegrauBipolar((valor1 * this.peso1)
            + (valor2 * this.peso2) + (valor3 * this.peso3)
184         + (valor4 * this.peso4) + (-1.0 * this.peso0));

186       if (this.ultimaResposta == -1.0) {
            resposta = "Valvula␣A";
188       }
          if (this.ultimaResposta == 1.0) {
190         resposta = "Valvula␣B";
          }
192
          return resposta;
```

```
194     }

196     private double getErro(ArquivoDadosTreinamento arquivoTreinamento) {
          FileReader arq;
198       BufferedReader lerArq;
          String linha;
200       String[] vetor;
          int i;
202       int numDadosTreinamento;
          double entrada1;
204       double entrada2;
          double entrada3;
206       double entrada4;
          double saidaEsperada;
208       double erro;

210       try {
            erro = 0D;
212         numDadosTreinamento = 0;
            arq = new FileReader(arquivoTreinamento.getCaminhoCompleto());
214         lerArq = new BufferedReader(arq);

216         linha = lerArq.readLine();
            if (linha.contains("x1")) {
218           linha = lerArq.readLine();
            }
220
            while (linha != null) {
222           numDadosTreinamento++;
              vetor = linha.split("\\s+");
224           i = 0;

226           if (vetor[0].equals("")) {
                i = 1;
228           }

230           entrada1 = Double.parseDouble(vetor[i++]
                .replace(",", "."));
232           entrada2 = Double.parseDouble(vetor[i++]
```

```
            .replace(",", "."));
234         entrada3 = Double.parseDouble(vetor[i++]
            .replace(",", "."));
236         entrada4 = Double.parseDouble(vetor[i++]
            .replace(",", "."));
238         saidaEsperada = Double.parseDouble(vetor[i]
            .replace(",", "."));
240
            this.ultimaResposta = (entrada1 * this.peso1) +
242           (entrada2 * this.peso2) + (entrada3 * this.peso3) +
              (entrada4 * this.peso4) + (-1.0 * this.peso0);
244         erro = erro + Math.pow((saidaEsperada - this.ultimaResposta),
                        2.0);
246
            linha = lerArq.readLine();
248       }

250       arq.close();
        } catch(IOException | NumberFormatException e) {
252       return 0D;
        }
254
        erro = erro / numDadosTreinamento;
256
        return erro;
258   }

260   private double funcaoDegrauBipolar(double valor) {
        if (valor < 0.0) {
262       return -1.0;
        }
264
        if (valor >= 0.0) {
266       return 1.0;
        }
268
        return 1.0;
270   }
```

```
272     private void imprimirSituacao(ArquivoDadosTreinamento
          arquivoTreinamento) {
274       Comunicador.addLog("EPOCA " + this.numEpocasTreinamento);
          Comunicador.addLog("Erro: " + Double
276         .toString(getErro(arquivoTreinamento)).replace(".", ","));
          Comunicador.addLog("Pesos: " + Double
278         .toString(this.peso0).replace(".", ",") + "; " +
          Double.toString(this.peso1).replace(".", ",")
280         + "; " + Double.toString(this.peso2).replace(".", ",") +
          "; " + Double.toString(this.peso3).replace(".", ",")
282         + "; " + Double.toString(this.peso4).replace(".", ","));
      }
284
  }
```

ANEXO A - CONJUNTO DE TREINAMENTO

x1	x2	x3	x4	d
0.4329	-1.3719	0.7022	-0.8535	1.0000
0.3024	0.2286	0.8630	2.7909	-1.0000
0.1349	-0.6445	1.0530	0.5687	-1.0000
0.3374	-1.7163	0.3670	-0.6283	-1.0000
1.1434	-0.0485	0.6637	1.2606	1.0000
1.3749	-0.5071	0.4464	1.3009	1.0000
0.7221	-0.7587	0.7681	-0.5592	1.0000
0.4403	-0.8072	0.5154	-0.3129	1.0000
-0.5231	0.3548	0.2538	1.5776	-1.0000
0.3255	-2.0000	0.7112	-1.1209	1.0000
0.5824	1.3915	-0.2291	4.1735	-1.0000
0.1340	0.6081	0.4450	3.2230	-1.0000
0.1480	-0.2988	0.4778	0.8649	1.0000
0.7359	0.1869	-0.0872	2.3584	1.0000
0.7115	-1.1469	0.3394	0.9573	-1.0000
0.8251	-1.2840	0.8452	1.2382	-1.0000
0.1569	0.3712	0.8825	1.7633	1.0000
0.0033	0.6835	0.5389	2.8249	-1.0000
0.4243	0.8313	0.2634	3.5855	-1.0000
1.0490	0.1326	0.9138	1.9792	1.0000
1.4276	0.5331	-0.0145	3.7286	1.0000
0.5971	1.4865	0.2904	4.6069	-1.0000
0.8475	2.1479	0.3179	5.8235	-1.0000
1.3967	-0.4171	0.6443	1.3927	1.0000
0.0044	1.5378	0.6099	4.7755	-1.0000
0.2201	-0.5668	0.0515	0.7829	1.0000
0.6300	-1.2480	0.8591	0.8093	-1.0000
-0.2479	0.8960	0.0547	1.7381	1.0000
-0.3088	-0.0929	0.8659	1.5483	-1.0000
-0.5180	1.4974	0.5453	2.3993	1.0000
0.6833	0.8266	0.0829	2.8864	1.0000
0.4353	-1.4066	0.4207	-0.4879	1.0000
-0.1069	-3.2329	0.1856	-2.4572	-1.0000
0.4662	0.6261	0.7304	3.4370	-1.0000
0.8298	-1.4089	0.3119	1.3235	-1.0000