UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DIRETORIA DE PESQUISA E PÓS GRADUAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

JÔNATAS TRABUCO BELOTTI

IMPLEMENTAÇÃO PROJETO PRÁTICO 5.10: MLP PARA SISTEMAS VARIANTES NO TEMPO

RELATÓRIO

PONTA GROSSA

2017

JÔNATAS TRABUCO BELOTTI

IMPLEMENTAÇÃO PROJETO PRÁTICO 5.10: MLP PARA SISTEMAS VARIANTES NO TEMPO

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

O *Perceptron* de Múltiplas Camadas (PMC, ou MLP do inglês *Multilayer Perceptron*) é constituído por um conjunto de neurônios artificiais dispostos em varias camadas, de modo que o sinal de entrada se propaga para frente através da rede, camada por camada. Dentre suas camadas existe a camada de entrada que recebe os sinais de entrada, a camada de saída que entrega o resultado obtido pela rede e no meio dessas podem existir quantas camadas forem necessárias. Essas camadas são chamadas intermediárias ou ocultas (HAYKIN, 2001). Note que uma MLP é constituida de pelo menos 2 camadas neurais, sendo uma camada de saída e pelo menos 1 camada escondida (SILVA; SPATTI; FLAUZINO, 2010).

As MLPs são consideradas uma das arquiteturas mais versáteis quanto a aplicabilidade, sendo utilizadas em diversas áreas do conhecimento. Dentre as utilizações da MLP estão: aproximação universal de funções, reconhecimento de padrões, identificação e controle de processos, previsão de series temporais e otimização de sistemas (SILVA; SPATTI; FLAUZINO, 2010).

Esse relatório tem como objetivo descrever o desenvolvimento do Projeto Prático 5.10 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 *Perceptron* Multicamadas para ser usada como preditora de um sistema variante no tempo com o objetivo de prever o preço de determinada mercadoria.

1.1 ESTUDO DE CASO

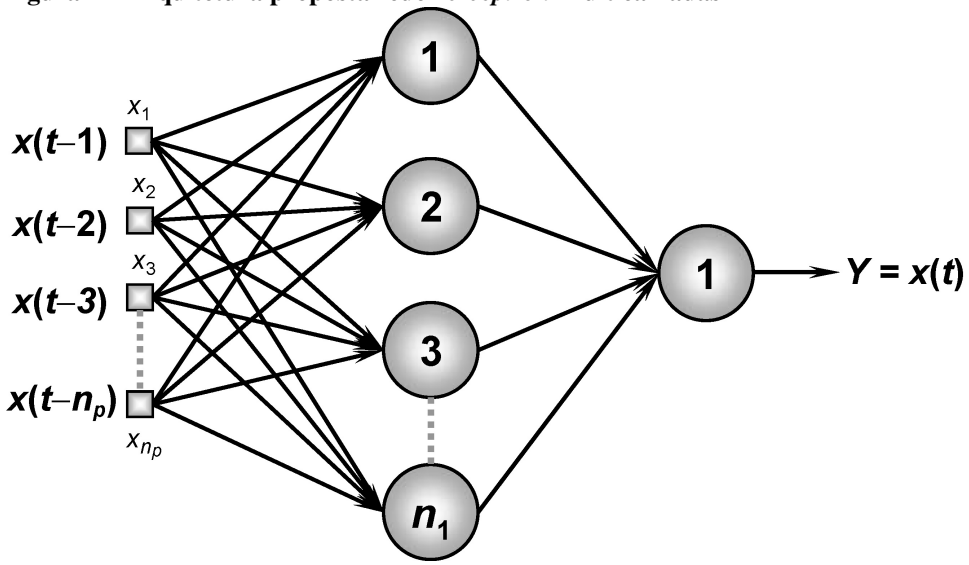
O projeto prático 5.10 do livro Redes neurais artificias para engenharia e ciências aplicadas de Silva, Spatti e Flauzino (2010), mostra que existe variação no preço de determinada mercadoria disposta para ser comercializada no mercado de ações.

O objetivo do projeto é o desenvolvimento, treinamento e teste de uma MLP que seja capaz de prever as variações no preço dessa mercadoria com base no histórico de preços. É determinada que a arquitetura da rede deve fazer uso da topologia *time delay neural network*. A Figura 1 mostra detalhadamente a arquitetura que a MLP deve ter para a realização do projeto.

Analisando a Figura 1 verifica-se que a quantidade de entradas que a MLP deve possuir não é definida, tal valor será definido durante a execução do projeto. A rede possui ainda penas uma camada escondida, sendo que a quantidade de neurônios nessa camada escondida deverá ser determinada durante a execução do projeto. Note ainda que a camada de saída da rede possui apenas 1 neurônio.

Para que se possa determinar a quantidade de neurônio presentes na camada escondida e a quantidade de entradas que a rede deve ter serão testadas 3 topologias diferentes. As topologias candidatas passiveis de serem aplicadas no mapeamento deste problema são:

Figura 1 – Arquitetura proposta rede *Perceptron* Multicamadas



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

- TDNN1 - 5 entradas, com 10 neurônios na camada escondida
- TDNN2 - 10 entradas, com 15 neurônios na camada escondida
- TDNN3 - 15 entradas, com 25 neurônios na camada escondida

2 DESENVOLVIMENTO DO PROJETO

A rede MLP proposta na Seção 1.1 foi desenvolvida na linguagem Java, a classe MLP é a responsável por implementar o funcionamento da rede, seu código fonte está disponível no Apêndice A. Com o intuito de facilitar o acesso a rede desenvolvida todo o código fonte, juntamente com o programa já compilado e os arquivos de treinamento e de teste estão disponíveis em um repositório do *GitHub* [1] que pode ser acessado pelo link <https://github.com/jonatastbelotti/MLPSistemaVarianteTempo>.

As seções 2.1 e 2.2 apresentam as discussões a respeito dos treinamentos e testes realizados na MLP.

2.1 TREINAMENTO

A rede neural Perceptron Multicamadas foi treinada utilizando o algoritmo de aprendizagem *backpropagation* com *momentum*. Para tanto os pesos sinápticos iniciais foram gerados de forma aleatória com valores entre 0 e 1. Foram utilizadas uma taxa de aprendizagem $\eta = 0,1$; fator *momentum* $\alpha = 0,8$ e precisão de $\varepsilon = 0,5 \times 10^{-6}$. O conjunto de dados utilizado para o treinamento da MLP está disponível no Anexo A.

Foram realizados 3 treinamentos para cada uma das redes *TDNN* candidatas apresentadas na Seção 1.1. A Tabela 1 apresenta o número de épocas de cada treinamento e o respectivo erro médio $\mathrm{E_M}$ para cada treinamento de cada rede candidata.

Tabela 1 – Resultados dos treinamentos

Treinamento	**TDNN 1**		**TDNN 2**		**TDNN 3**	
	$\mathbf{E_M}$	**Épocas**	$\mathbf{E_M}$	**Épocas**	$\mathbf{E_M}$	**Épocas**
1º (T1)	$0,012718$	9665	$0,010994$	3888	$0,010819$	3723
2º (T2)	$0,012850$	9206	$0,009872$	9108	$0,008463$	7456
3º (T3)	$0,015251$	5839	$0,008631$	7176	$0,011229$	2781

Fonte: Autoria própria.

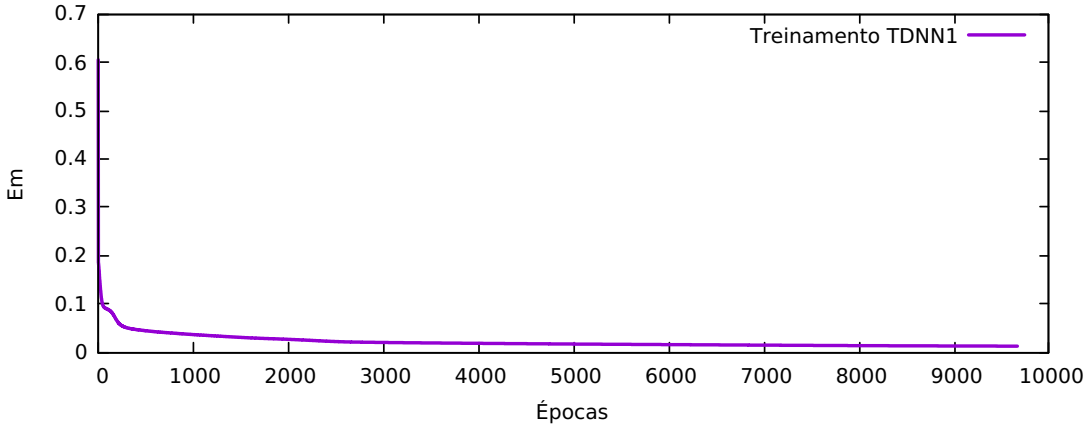
Analisando os dados da Tabela 1 nota-se que os melhores treinamentos de cada rede candidata são:

- TDNN1 - Treinamento T(1), atingindo $\mathrm{E_M} = 0,012718$ ao fim de 9665 épocas;
- TDNN2 - Treinamento T(3), atingindo $\mathrm{E_M} = 0,008631$ ao fim de 5839 épocas;
- TDNN3 - Treinamento T(2), atingindo $\mathrm{E_M} = 0,008463$ ao fim de 7456 épocas;

[1] No repositório do *GitHub* o caminho para acessar a classe MLP.java é 'MLPSistemasVariantesTempo/src/Modelo/MLP.java'

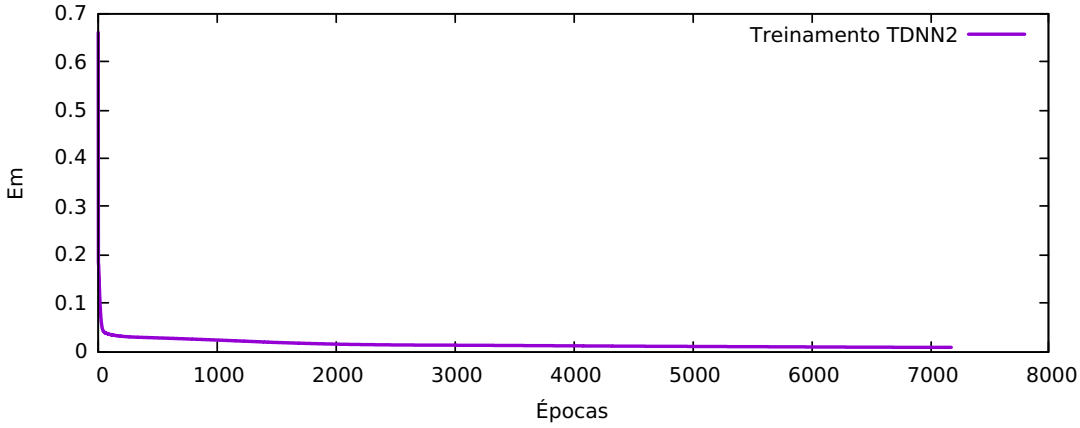
Com o objetivo de analisar os melhores treinamentos dentre as redes candidatas os gráficos 1, 2 e 3 apresentam o o erro médio em relação de cada época de treinamento para o melhor treinamento de cada rede candidata.

Gráfico 1 – E_M treinamento TDNN1



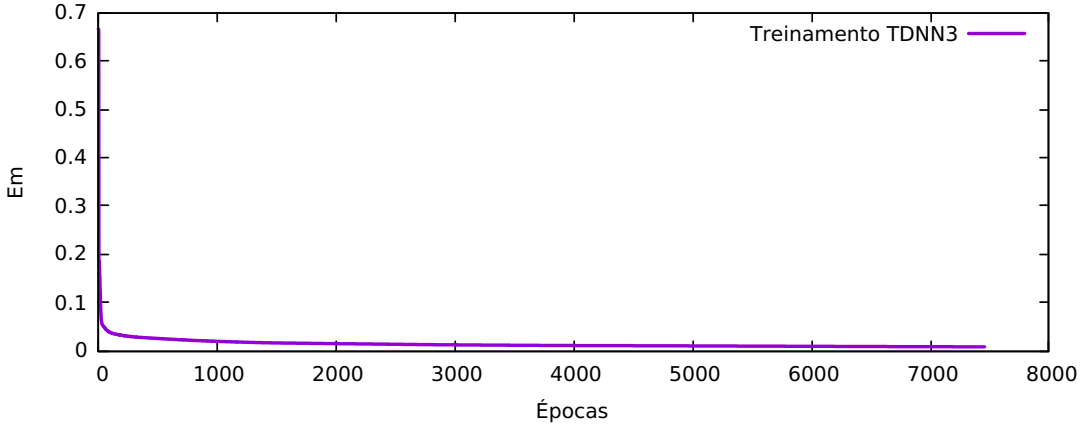
Fonte: Autoria própria.

Gráfico 2 – E_M treinamento TDNN2



Fonte: Autoria própria.

Gráfico 3 – E_M treinamento TDNN3



Fonte: Autoria própria.

Pelos gráficos 1, 2 e 3 nota-se que o comportamento do $\mathrm{E_M}$ é semelhante para os 3 treinamentos. Inicialmente o $\mathrm{E_M}$ é maior, o que gera um valor de gradiente maior que por sua vez ajusta os pesos sinápticos com um valor maior. Como a cada iteração os pesos sinápticos são ajustados visando minimizar o $\mathrm{E_M}$ e o ajuste dos pesos sinápticos se dá em função do valor gradiente calculado em razão do $\mathrm{E_M}$, a cada iteração o $\mathrm{E_M}$ é menor fazendo com que o ajuste nos pesos sinápticos seja menor e consequentemente fazendo com que a diminuição do $\mathrm{E_M}$ para a próxima iteração também seja menor.

Todos os treinamentos realizados para cada rede candidata obtiveram resultados diferentes, isso se deve aos seguintes fatores:

- Inicialização dos pesos sinápticos – para cada treinamento de cada rede candidata foram gerados pesos sinápticos de forma aleatória com valores entre 0 e 1, assim todos os treinamentos foram realizados partindo de matrizes de pesos sinápticos diferentes.
- Ordem do preditor ($\mathrm{n_p}$) e número de neurônios – foram testadas 3 possibilidades para a ordem do preditor e a quantidade de neurônios na camada escondida (Seção 1.1), gerando assim 3 topologias candidatas. Topologias diferentes geram resultados diferentes, foram testadas 3 topologias com o objetivo de verificar qual delas se adequá melhor ao problema específico.

De posse dos dados da Tabela 1 e dos gráficos 1, 2 e 3 é possível constatar que o menor erro médio foi obtido pelo treinamento T2 da rede candidata TDNN3, $\mathrm{n_p} = 0,008463$.

2.2 TESTE

Para cada treinamento realizado em cada topologia candidata apresentado na Seção 2.1 foi efetuado um teste com o objetivo de validar as topologias candidatas em relação aos valores previstos pelas topologias em comparação com os valores desejados. Para o teste foi utilizado um conjunto de 20 amostras disponíveis na Tabela 2. O conjunto de teste também esta disponível no Anexo B.

A Tabela 2 também apresenta os resultados obtidos nos testes realizados na topologia TDNN1 e TDNN2. Nela também é possível verificar o Erro Relativo Médio (E.R.M.) juntamente com a variância obtidos por cada treinamento das topologias TDNN1 e TDNN2.

Por sua vez, a Tabela 3 apresenta os resultados dos testes realizados nos 3 treinamentos da topologia candidata TDNN3.

Analisando os dados das tabelas 2 e 3 verifica-se que o menor erro relativo médio dentre todos os treinamentos realizados nas 3 topologias foi $7,5385\%$, obtido no treinamento T(1) da topologia TDNN3. Por sua vez o maior erro relativo médio dentre todos os treinamentos realizados nas 3 topologias foi $23,4979\%$, obtido no treinamento T(2) da topologia TDNN1.

Tabela 2 – Resultados dos testes

Valores		**TDNN 1**			**TDNN 2**		
Amostras	**x(t)**	**(T1)**	**(T2)**	**(T3)**	**(T1)**	**(T2)**	**(T3)**
$\mathbf{t = 101}$	$\mathbf{0.4173}$	$0,418977$	$0,432502$	$0,420124$	$0,424318$	$0,429526$	$0,427589$
$\mathbf{t = 102}$	$\mathbf{0.0062}$	$0,008855$	$0,007579$	$0,005535$	$0,005603$	$0,002150$	$0,010613$
$\mathbf{t = 103}$	$\mathbf{0.3387}$	$0,371766$	$0,354102$	$0,381037$	$0,360712$	$0,352588$	$0,348029$
$\mathbf{t = 104}$	$\mathbf{0.1886}$	$0,164434$	$0,174134$	$0,155159$	$0,169172$	$0,145559$	$0,164645$
$\mathbf{t = 105}$	$\mathbf{0.7418}$	$0,720219$	$0,716613$	$0,704831$	$0,748859$	$0,742605$	$0,745211$
$\mathbf{t = 106}$	$\mathbf{0.3138}$	$0,277783$	$0,293661$	$0,263251$	$0,298566$	$0,311550$	$0,310256$
$\mathbf{t = 107}$	$\mathbf{0.4466}$	$0,445375$	$0,446080$	$0,438734$	$0,443213$	$0,447732$	$0,444803$
$\mathbf{t = 108}$	$\mathbf{0.0835}$	$0,074448$	$0,072425$	$0,083967$	$0,077594$	$0,082386$	$0,082160$
$\mathbf{t = 109}$	$\mathbf{0.1930}$	$0,248380$	$0,226364$	$0,236434$	$0,206201$	$0,199540$	$0,199357$
$\mathbf{t = 110}$	$\mathbf{0.3807}$	$0,328418$	$0,307122$	$0,366619$	$0,413383$	$0,417628$	$0,415959$
$\mathbf{t = 111}$	$\mathbf{0.5438}$	$0,613396$	$0,651033$	$0,557366$	$0,517604$	$0,533525$	$0,524578$
$\mathbf{t = 112}$	$\mathbf{0.5897}$	$0,502975$	$0,480258$	$0,526549$	$0,579897$	$0,585623$	$0,594158$
$\mathbf{t = 113}$	$\mathbf{0.3536}$	$0,397828$	$0,425064$	$0,365572$	$0,325576$	$0,334955$	$0,344193$
$\mathbf{t = 114}$	$\mathbf{0.2210}$	$0,171160$	$0,160816$	$0,182206$	$0,258723$	$0,250284$	$0,260785$
$\mathbf{t = 115}$	$\mathbf{0.0631}$	$0,119827$	$0,121117$	$0,118111$	$0,058527$	$0,058132$	$0,057847$
$\mathbf{t = 116}$	$\mathbf{0.4499}$	$0,426123$	$0,397304$	$0,443344$	$0,531708$	$0,479143$	$0,487885$
$\mathbf{t = 117}$	$\mathbf{0.2564}$	$0,392078$	$0,436904$	$0,372514$	$0,185875$	$0,170061$	$0,181735$
$\mathbf{t = 118}$	$\mathbf{0.7642}$	$0,685099$	$0,680070$	$0,660815$	$0,781883$	$0,776356$	$0,769333$
$\mathbf{t = 119}$	$\mathbf{0.1411}$	$0,265978$	$0,269032$	$0,246983$	$0,055796$	$0,058580$	$0,060430$
$\mathbf{t = 120}$	$\mathbf{0.3626}$	$0,321217$	$0,324087$	$0,302573$	$0,368229$	$0,375945$	$0,379133$
E. R. M. $(\%)$		$22,7333$	$23,4979$	$18,2224$	$10,2954$	$12,1793$	$11,9249$
Variância $(\%)$		$686,9928$	$754,1994$	$575,5429$	$184,3271$	$359,8478$	$374,7971$

Fonte: Autoria própria.

Ainda analisando os dados das tabelas 2 e 3 é possível notar que os valores previstos pelas topologias candidatas foram próximos dos valores desejados, tendo muitas vezes se diferenciado a partir da 3^{a} casa decimal, como no teste $t = 107$ do treinamento T(3) da topologia candidata TDNN3 onde o valor previsto pela rede foi $0,448360$ enquanto o valor desejado era $0,4466$, tendo uma diferença de $0,00176$.

Já em relação a variância do erro relativo médio, o maior valor obtido foi $754,1994\%$ no treinamento T(2) da topologia TDNN1 e o menor valor foi $46,1201\%$ no treinamento T(1) da topologia TDNN3.

Considerando os melhores treinamentos de cada topologia candidata apresentados na Seção 2.1 os gráficos 4, 5 e 6 apresentam a comparação dos valores desejados com os valores previstos pelas redes.

Analisando os gráficos 4, 5 e 6 nota-se que os mesmos comprovam as análises das tabelas 2 e 3, confirmando que os valores previstos pelas redes foram próximos aos valores desejados.

Ainda com base nos gráficos 4, 5 e 6 verifica-se que as curvas dos valores previstos e valores desejados são quase sobrepostas nos 3 gráficos. Note que nos gráficos 4 e 5 até a previsão $t = 8$ as curvas de valores previstos e valores desejados são praticamente sobrepostos, sendo que

Tabela 3 – Continuação resultados dos testes

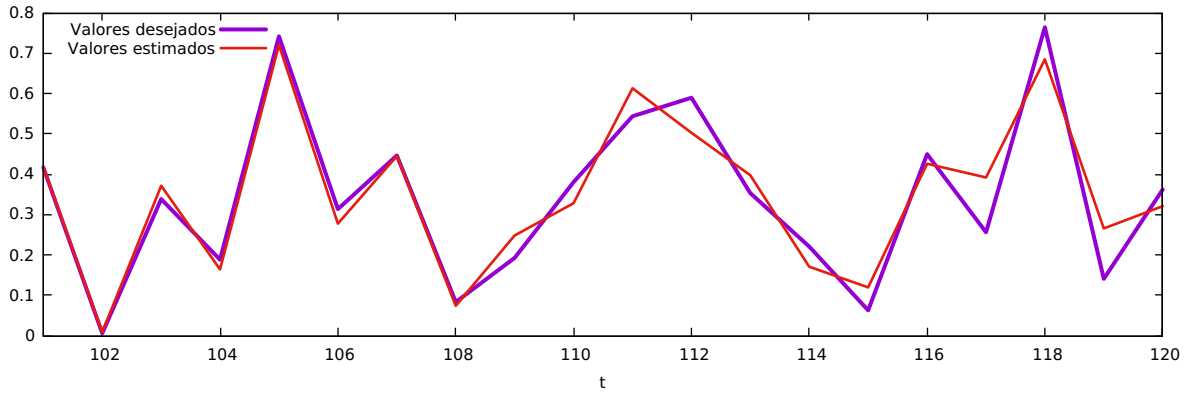
Valores		**TDNN 3**		
Amostras	**x(t)**	**(T1)**	**(T2)**	**(T3)**
$\mathbf{t = 101}$	$\mathbf{0.4173}$	$0,410293$	$0,425906$	$0,421526$
$\mathbf{t = 102}$	$\mathbf{0.0062}$	$0,005250$	$0,011938$	$0,008122$
$\mathbf{t = 103}$	$\mathbf{0.3387}$	$0,358699$	$0,348261$	$0,356002$
$\mathbf{t = 104}$	$\mathbf{0.1886}$	$0,164081$	$0,166110$	$0,157600$
$\mathbf{t = 105}$	$\mathbf{0.7418}$	$0,739856$	$0,735151$	$0,740295$
$\mathbf{t = 106}$	$\mathbf{0.3138}$	$0,285857$	$0,305316$	$0,293661$
$\mathbf{t = 107}$	$\mathbf{0.4466}$	$0,451262$	$0,446171$	$0,448360$
$\mathbf{t = 108}$	$\mathbf{0.0835}$	$0,069489$	$0,075136$	$0,078458$
$\mathbf{t = 109}$	$\mathbf{0.1930}$	$0,216210$	$0,201082$	$0,200419$
$\mathbf{t = 110}$	$\mathbf{0.3807}$	$0,375205$	$0,387858$	$0,391664$
$\mathbf{t = 111}$	$\mathbf{0.5438}$	$0,556521$	$0,523676$	$0,534216$
$\mathbf{t = 112}$	$\mathbf{0.5897}$	$0,544014$	$0,585152$	$0,575542$
$\mathbf{t = 113}$	$\mathbf{0.3536}$	$0,362288$	$0,337637$	$0,335654$
$\mathbf{t = 114}$	$\mathbf{0.2210}$	$0,198494$	$0,242368$	$0,240653$
$\mathbf{t = 115}$	$\mathbf{0.0631}$	$0,080307$	$0,065769$	$0,064553$
$\mathbf{t = 116}$	$\mathbf{0.4499}$	$0,430503$	$0,487188$	$0,492330$
$\mathbf{t = 117}$	$\mathbf{0.2564}$	$0,275968$	$0,214847$	$0,205326$
$\mathbf{t = 118}$	$\mathbf{0.7642}$	$0,746982$	$0,789984$	$0,791961$
$\mathbf{t = 119}$	$\mathbf{0.1411}$	$0,151407$	$0,079070$	$0,085115$
$\mathbf{t = 120}$	$\mathbf{0.3626}$	$0,352079$	$0,395980$	$0,407131$
E. R. M. $(\%)$		$7,5385$	$11,6527$	$8,9353$
Variância $(\%)$		$46,1201$	$454,6472$	$110,8064$

Fonte: Autoria própria.

a partir de previsão $t = 9$ os valores previstos são próximos dos valores ótimos, mas as curvas não são mais sobrepostas. Por sua vez, no Gráfico 6 as curvas são sobrepostas até a previsão $t = 116$ e a partir da previsão $t = 117$ os valores previstos e desejados continuam próximos mas as curvas não são mais sobrepostas.

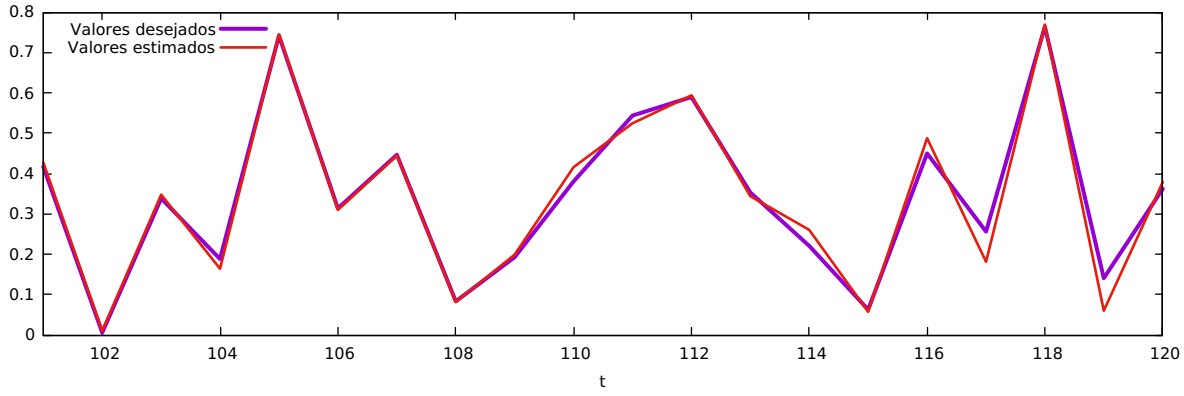
Como mostrado na Seção 2.1, levando em consideração apenas o erro médio a melhor rede foi a topologia TDNN3 com o treinamento T2. Em contra partida, levando em conta apenas os resultados dos testes a melhor rede foi a topologia TDNN3 com o treinamento T1, pois foi a que obteve o menor erro relativo médio.

Gráfico 4 – Resultado teste TDNN1



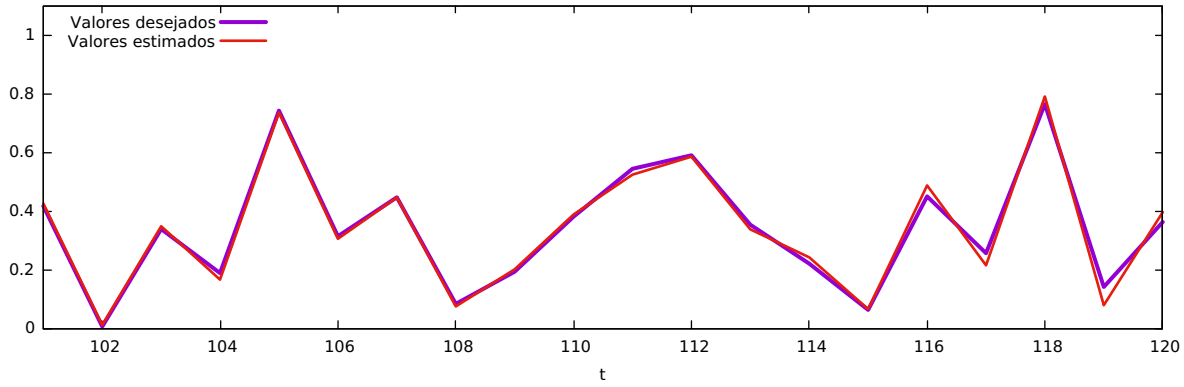
Fonte: Autoria própria.

Gráfico 5 – Resultado teste TDNN2



Fonte: Autoria própria.

Gráfico 6 – Resultado teste TDNN3



Fonte: Autoria própria.

3 CONCLUSÃO

Foi desenvolvida uma rede neural Perceptron multicamadas com apenas 1 camada escondida. Foram testadas 3 topologias candidatas TDNN1, TDNN2 e TDNN3, para cada topologia candidata foram realizados 3 testes com o objetivo de determinar qual a melhor topologia para ser utilizada no problema.

Conclui-se que a melhor rede para ser utilizada no problema da previsão do custo de mercadorias é a topologia TDNN3 com a configuração de treinamento obtida pelo treinamento T1, pois a mesma obteve o menor erro médio relativo dentre todos os testes de todas as topologias.

REFERÊNCIAS

HAYKIN, Simon. **Redes Neurais: principios e prática**. 2. ed. Porto Alegre: Bookman, 2001. ISBN 978-85-7307-718-6.

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.

APÊNDICE A - IMPLEMENTAÇÃO DA CLASSE MLP EM JAVA

```
1  package Modelo;
2
   import Controle.Comunicador;
4  import Recursos.Arquivo;
   import java.io.BufferedReader;
6  import java.io.FileNotFoundException;
   import java.io.FileReader;
8  import java.io.IOException;
   import java.util.Random;
10
   /**
12  *
    * @author Jônatas Trabuco Belotti [jonatas.t.belotti@hotmail.com]
14  */
   public class MLP {
16
     public static final int NUM_ENTRADAS = 15;
18   private final int NUM_NEU_CAMADA_ESCONDIDA = 25;
     public static final int NUM_NEU_CAMADA_SAIDA = 1;
20   private final double TAXA_APRENDIZAGEM = 0.1;
     private final double PRECISAO = 0.5 * Math.pow(10D, -6D);
22   private final double FATOR_MOMENTUM = 0.8;
     private final double BETA = 1.0;
24   private final int LIMITE_NUM_EPOCAS = 50000;

26   private int numEpocas;
     private double[] entradas;
28   private double[][] pesosCamadaEscondida;
     private double[][] pesosCamadaEscondidaProximo;
30   private double[][] pesosCamadaEscondidaAnterior;
     private double[][] pesosCamadaSaida;
32   private double[][] pesosCamadaSaidaProximo;
     private double[][] pesosCamadaSaidaAnterior;
34   private double[] potencialCamadaEscondida;
     private double[] saidaCamadaEscondida;
36   private double[] potencialCamadaSaida;
     private double[] saidaCamadaSaida;
38   private double[] saidaEsperada;
     private double[] gradienteCamadaSaida;
```

```
40  private double[] gradienteCamadaEscondida;

42  public MLP() {
      Random random;
44
      entradas = new double[NUM_ENTRADAS + 1];
46    pesosCamadaEscondida = new double[NUM_NEU_CAMADA_ESCONDIDA][
        NUM_ENTRADAS + 1];
      pesosCamadaEscondidaAnterior = new double[
        NUM_NEU_CAMADA_ESCONDIDA][NUM_ENTRADAS + 1];
48    pesosCamadaEscondidaProximo = new double[
        NUM_NEU_CAMADA_ESCONDIDA][NUM_ENTRADAS + 1];
      pesosCamadaSaida = new double[NUM_NEU_CAMADA_SAIDA][
        NUM_NEU_CAMADA_ESCONDIDA + 1];
50    pesosCamadaSaidaAnterior = new double[NUM_NEU_CAMADA_SAIDA][
        NUM_NEU_CAMADA_ESCONDIDA + 1];
      pesosCamadaSaidaProximo = new double[NUM_NEU_CAMADA_SAIDA][
        NUM_NEU_CAMADA_ESCONDIDA + 1];
52    potencialCamadaEscondida = new double[NUM_NEU_CAMADA_ESCONDIDA
        + 1];
      saidaCamadaEscondida = new double[NUM_NEU_CAMADA_ESCONDIDA +
        1];
54    potencialCamadaSaida = new double[NUM_NEU_CAMADA_SAIDA];
      saidaCamadaSaida = new double[NUM_NEU_CAMADA_SAIDA];
56    saidaEsperada = new double[NUM_NEU_CAMADA_SAIDA];
      gradienteCamadaSaida = new double[NUM_NEU_CAMADA_SAIDA];
58    gradienteCamadaEscondida = new double[NUM_NEU_CAMADA_ESCONDIDA
        ];

60    //Iniciando pesos sinapticos
      random = new Random();
62
      for (int i = 0; i < NUM_NEU_CAMADA_ESCONDIDA; i++) {
64      for (int j = 0; j < NUM_ENTRADAS + 1; j++) {
          pesosCamadaEscondida[i][j] = random.nextDouble();
66      }
      }
68
      for (int i = 0; i < NUM_NEU_CAMADA_SAIDA; i++) {
70      for (int j = 0; j < NUM_NEU_CAMADA_ESCONDIDA + 1; j++) {
          pesosCamadaSaida[i][j] = random.nextDouble();
72      }
```

```
    }
74  }

76  public boolean treinar(Arquivo arquivoTreinamento) {
      FileReader arq;
78    BufferedReader lerArq;
      String linha;
80    double erroAtual;
      double erroAnterior;
82    long tempInicial;

84    copiarMatriz(pesosCamadaEscondida, pesosCamadaEscondidaAnterior
        );
      copiarMatriz(pesosCamadaSaida, pesosCamadaSaidaAnterior);
86
      tempInicial = System.currentTimeMillis();
88    numEpocas = 0;
      erroAtual = erroQuadraticoMedio(arquivoTreinamento);
90
      Comunicador.iniciarLog("Início treinamento da MLP");
92    Comunicador.addLog(String.format("Erro inicial: %.6f",
        erroAtual).replace(".", ","));
      imprimirPesos();
94    Comunicador.addLog("Época Eqm");

96    try {
        do {
98        this.numEpocas++;
          erroAnterior = erroAtual;
100       arq = new FileReader(arquivoTreinamento.getCaminhoCompleto
            ());
          lerArq = new BufferedReader(arq);
102
          linha = lerArq.readLine();
104       if (linha.contains("x") || linha.contains("f")) {
            linha = lerArq.readLine();
106       }

108       //Iniciando janela de entradas
          entradas[0] = -1D;
110       for (int i = entradas.length - 1; i > 0; i--) {
            entradas[i] = separarEntrada(linha);
```

```
112             linha = lerArq.readLine();
            }
114
            while (linha != null) {
116             saidaEsperada[0] = separarEntrada(linha);//Proxima
                    entrada é a saída experada

118             calcularSaidas();

120             ajustarPesos();

122             linha = lerArq.readLine();
                ajustarJanela();
124             entradas[1] = saidaEsperada[0];
            }
126
            arq.close();
128         erroAtual = erroQuadraticoMedio(arquivoTreinamento);
            Comunicador.addLog(String.format("%d   %.6f", numEpocas,
                erroAtual).replace(".", ","));
130       } while (Math.abs(erroAtual - erroAnterior) > PRECISAO &&
             numEpocas < LIMITE_NUM_EPOCAS);

132       Comunicador.addLog(String.format("Fim do treinamento. (%.2fs)
             ", (double) (System.currentTimeMillis() - tempInicial) /
             1000D));
          imprimirPesos();
134     } catch (FileNotFoundException ex) {
          return false;
136     } catch (IOException ex) {
          return false;
138     }

140     return true;
    }
142
    public void testar(Arquivo arquivoTreinamento, Arquivo
        arquivoTeste) {
144     FileReader arq;
        BufferedReader lerArq;
146     String linha;
        int numAmostras;
```

```
148     double erroMedio;
        double variancia;
150     double erro;

152     numAmostras = 0;
        erroMedio = 0D;
154     variancia = 0D;

156     Comunicador.iniciarLog("Início teste da MLP");
        Comunicador.addLog("x  -- y");
158
        try {
160       //Iniciando janela de entradas
          arq = new FileReader(arquivoTreinamento.getCaminhoCompleto())
             ;
162       lerArq = new BufferedReader(arq);

164       linha = lerArq.readLine();
          if (linha.contains("x") || linha.contains("f")) {
166         linha = lerArq.readLine();
          }
168
          entradas[0] = -1D;
170       for (int i = entradas.length - 1; i > 0; i--) {
            entradas[i] = separarEntrada(linha);
172         linha = lerArq.readLine();
          }
174
          while (linha != null) {
176         ajustarJanela();
            entradas[1] = separarEntrada(linha);
178         linha = lerArq.readLine();
          }
180
          //Abrindo arquivos com dados de teste
182       arq = new FileReader(arquivoTeste.getCaminhoCompleto());
          lerArq = new BufferedReader(arq);
184
          linha = lerArq.readLine();
186       if (linha.contains("x") || linha.contains("f")) {
            linha = lerArq.readLine();
188       }
```

```
190         while (linha != null) {
              saidaEsperada[0] = separarEntrada(linha);//Proxima entrada
                 é a saída experada
192
              numAmostras++;
194
              calcularSaidas();
196
              erro = (100D / saidaEsperada[0]) * Math.abs(saidaEsperada
                 [0] - saidaCamadaSaida[0]);
198           erroMedio += erro;
              variancia += Math.pow(erro, 2D);
200           Comunicador.addLog(String.format("%.6f %.6f %.2f%%",
                 saidaEsperada[0], saidaCamadaSaida[0], erro));

202           ajustarJanela();
              entradas[1] = saidaCamadaSaida[0];
204           linha = lerArq.readLine();
            }
206
            //Calculando erro relativo médio
208         erroMedio = erroMedio / (double) numAmostras;

210         //Calculando variância
            variancia = variancia - ((double) numAmostras * Math.pow(
               erroMedio, 2D));
212         variancia = variancia / ((double) (numAmostras - 1));

214         Comunicador.addLog("Fim do teste");
            Comunicador.addLog(String.format("Erro relativo médio: %.6f%%
               ", erroMedio));
216         Comunicador.addLog(String.format("Variância: %.6f%%",
               variancia));

218         arq.close();
          } catch (FileNotFoundException ex) {
220       } catch (IOException ex) {
          }
222     }

224     private double erroQuadraticoMedio(Arquivo arquivo) {
```

```
        FileReader arq;
226     BufferedReader lerArq;
        String linha;
228     int numAmostras;
        double erroMedio;
230     double erro;

232     erroMedio = 0D;
        numAmostras = 0;
234
        try {
236       arq = new FileReader(arquivo.getCaminhoCompleto());
          lerArq = new BufferedReader(arq);
238
          linha = lerArq.readLine();
240       if (linha.contains("x") || linha.contains("f")) {
            linha = lerArq.readLine();
242       }

244       //Iniciando janela de entradas
          entradas[0] = -1D;
246       for (int i = entradas.length - 1; i > 0; i--) {
            entradas[i] = separarEntrada(linha);
248         linha = lerArq.readLine();
          }
250
          while (linha != null) {
252         numAmostras++;
            saidaEsperada[0] = separarEntrada(linha);//Proxima entrada
                é a saída experada
254
            calcularSaidas();
256
            //Calculando erro
258         erro = 0D;
            for (int i = 0; i < saidaCamadaSaida.length; i++) {
260           erro = erro + Math.abs(saidaEsperada[i] -
                  saidaCamadaSaida[i]);
            }
262         erroMedio = erroMedio + erro;

264         ajustarJanela();
```

```
            entradas[1] = saidaEsperada[0];
266         linha = lerArq.readLine();
          }
268
          arq.close();
270       erroMedio = erroMedio / (double) numAmostras;

272     } catch (FileNotFoundException ex) {
        } catch (IOException ex) {
274     }

276     return erroMedio;
      }
278
      private double separarEntrada(String linha) {
280     String[] vetor;
        int i;
282
        vetor = linha.split("\\s+");
284     i = 0;

286     if (vetor[0].equals("")) {
          i = 1;
288     }

290     return Double.parseDouble(vetor[i].replace(",", "."));
      }
292
      private void ajustarJanela() {
294     for (int i = entradas.length - 1; i > 1; i--) {
          entradas[i] = entradas[i - 1];
296     }
      }
298
      private void calcularSaidas() {
300     double valorParcial;

302     //Calculando saidas da camada escondida
        saidaCamadaEscondida[0] = -1D;
304     potencialCamadaEscondida[0] = -1D;

306     for (int i = 1; i < saidaCamadaEscondida.length; i++) {
```

```
        valorParcial = 0D;
308
        for (int j = 0; j < entradas.length; j++) {
310       valorParcial += entradas[j] * pesosCamadaEscondida[i - 1][j
              ];
        }
312
        potencialCamadaEscondida[i] = valorParcial;
314     saidaCamadaEscondida[i] = funcaoLogistica(valorParcial);
      }
316
      //Calculando saida da camada de saída
318   for (int i = 0; i < saidaCamadaSaida.length; i++) {
        valorParcial = 0D;
320
        for (int j = 0; j < saidaCamadaEscondida.length; j++) {
322       valorParcial += saidaCamadaEscondida[j] * pesosCamadaSaida[
              i][j];
        }
324
        potencialCamadaSaida[i] = valorParcial;
326     saidaCamadaSaida[i] = funcaoLogistica(valorParcial);
      }
328 }

330 private void ajustarPesos() {
      //Ajustando pesos sinapticos da camada de saida
332   for (int i = 0; i < gradienteCamadaSaida.length; i++) {
        gradienteCamadaSaida[i] = (saidaEsperada[i] -
            saidaCamadaSaida[i]) * funcaoLogisticaDerivada(
            potencialCamadaSaida[i]);
334
        for (int j = 0; j < NUM_NEU_CAMADA_ESCONDIDA + 1; j++) {
336       pesosCamadaSaidaProximo[i][j] = pesosCamadaSaida[i][j]
                  + (FATOR_MOMENTUM * (pesosCamadaSaida[i][j] -
                      pesosCamadaSaidaAnterior[i][j]))
338               + (TAXA_APRENDIZAGEM * gradienteCamadaSaida[i] *
                      saidaCamadaEscondida[j]);
        }
340   }

342   //Ajustando pesos sinapticos da camada escondida
```

```
      for (int i = 0; i < gradienteCamadaEscondida.length; i++) {
344     gradienteCamadaEscondida[i] = 0D;
        for (int j = 0; j < NUM_NEU_CAMADA_SAIDA; j++) {
346       gradienteCamadaEscondida[i] += gradienteCamadaSaida[j] *
              pesosCamadaSaida[j][i + 1];
        }
348     gradienteCamadaEscondida[i] *= funcaoLogisticaDerivada(
            potencialCamadaEscondida[i + 1]);

350     for (int j = 0; j < NUM_ENTRADAS + 1; j++) {
          pesosCamadaEscondidaProximo[i][j] = pesosCamadaEscondida[i
              ][j]
352             + (FATOR_MOMENTUM * (pesosCamadaEscondida[i][j] -
                    pesosCamadaEscondidaAnterior[i][j]))
                + (TAXA_APRENDIZAGEM * gradienteCamadaEscondida[i]
                    * entradas[j]);
354     }
      }
356
      //Copiando pesos
358   copiarMatriz(pesosCamadaEscondida, pesosCamadaEscondidaAnterior
        );
      copiarMatriz(pesosCamadaEscondidaProximo, pesosCamadaEscondida)
        ;
360   copiarMatriz(pesosCamadaSaida, pesosCamadaSaidaAnterior);
      copiarMatriz(pesosCamadaSaidaProximo, pesosCamadaSaida);
362 }

364 private void copiarMatriz(double[][] origem, double[][] destino)
      {
      for (int i = 0; i < origem.length; i++) {
366     for (int j = 0; j < origem[i].length; j++) {
        if (destino.length > i) {
368       if (destino[i].length > j) {
            destino[i][j] = origem[i][j];
370       }
        }
372     }
      }
374 }

376 private double funcaoLogistica(double valor) {
```

```
        return 1D / (1D + Math.pow(Math.E, -1D * BETA * valor));
378   }

380   private double funcaoLogisticaDerivada(double valor) {
        return (BETA * Math.pow(Math.E, -1D * BETA * valor)) / Math.pow
           ((Math.pow(Math.E, -1D * BETA * valor) + 1D), 2D);
382   }

384   private void imprimirPesos() {
        String log;
386
        Comunicador.addLog("Pesos camada escondida:");
388
        for (int i = 0; i < NUM_NEU_CAMADA_ESCONDIDA; i++) {
390       log = "N" + (i + 1) + " =";

392       for (int j = 0; j < NUM_ENTRADAS + 1; j++) {
            log += String.format(" %f", pesosCamadaEscondida[i][j]);
394       }

396       Comunicador.addLog(log);
        }
398
        Comunicador.addLog("Pesos camada de saída:");
400     for (int i = 0; i < NUM_NEU_CAMADA_SAIDA; i++) {
          log = "N" + (i + 1) + " =";
402
          for (int j = 0; j < NUM_NEU_CAMADA_ESCONDIDA + 1; j++) {
404         log += String.format(" %f", pesosCamadaSaida[i][j]);
          }
406
          Comunicador.addLog(log);
408     }
    }
410
}
```

ANEXO A - CONJUNTO DE TREINAMENTO

f(t)
0.1701
0.1023
0.4405
0.3609
0.7192
0.2258
0.3175
0.0127
0.4290
0.0544
0.8000
0.0450
0.4268
0.0112
0.3218
0.2185
0.7240
0.3516
0.4420
0.0984
0.1747
0.3964
0.5114
0.6183
0.3330
0.2398
0.0508
0.4497
0.2178
0.7762
0.1078
0.3773
0.0001
0.3877
0.0821
0.7836

0.1887
0.4483
0.0424
0.2539
0.3164
0.6386
0.4862
0.4068
0.1611
0.1101
0.4372
0.3795
0.7092
0.2400
0.3087
0.0159
0.4330
0.0733
0.7995
0.0262
0.4223
0.0085
0.3303
0.2037
0.7332
0.3328
0.4445
0.0909
0.1838
0.3888
0.5277
0.6042
0.3435
0.2304
0.0568
0.4500
0.2371
0.7705
0.1246
0.3701

0.0006
0.3943
0.0646
0.7878
0.1694
0.4468
0.0372
0.2632
0.3048
0.6516
0.4690
0.4132
0.1523
0.1182
0.4334
0.3978
0.6987
0.2538
0.2998
0.0195
0.4366
0.0924
0.7984
0.0077

ANEXO B - CONJUNTO DE TESTE

x(t)
0.4173
0.0062
0.3387
0.1886
0.7418
0.3138
0.4466
0.0835
0.1930
0.3807
0.5438
0.5897
0.3536
0.2210
0.0631
0.4499
0.2564
0.7642
0.1411
0.3626