

**FACULDADE DE TECNOLOGIA DO IPIRANGA**

**CURSO DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS**

MARIO THOMAZ APRA

**APLICAÇÃO DE REDES NEURAIS ARTIFICIAIS PARA PREVISÃO DE PREÇOS DAS AÇÕES NO MERCADO FINANCEIRO**

SÃO PAULO

2017



**FACULDADE DE TECNOLOGIA DO IPIRANGA**

**CURSO DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS**

MARIO THOMAZ APRA

**APLICAÇÃO DE REDES NEURAIS ARTIFICIAIS PARA PREVISÃO DE PREÇOS DAS AÇÕES NO MERCADO FINANCEIRO**

Trabalho de Conclusão de Curso apresentado à Faculdade de Tecnologia do Ipiranga, como requisito parcial para a obtenção do grau de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientadora: Prof.ª Dr.ª Andréia Cristina G. Machion.

SÃO PAULO

2017

AGRADECIMENTOS

Aos meus parentes, pelo amor, incentivo e apoio incondicional. À minha tia Simone que foi uma pessoa bem importante no começo da faculdade pelo incentivo.

Agradeço a meus companheiros de maratonas: David Cabrera, Fabio Santana, Felipe Conceição, Lucas Ferreira e Neftales Antunes; aos grandes companheiros do curso, em especial ao Cauê Polimanti e à Juliana Aboud.

Agradeço а todos os professores por me proporcionar conhecimento não apenas racional, mas também а manifestação do caráter е da afetividade da educação durante todo o processo da minha formação. Agradeço por tudo que fizeram por mim, não somente por terem me ensinado com maestria, mas por terem me ensinado algo a mais. А palavra mestre, nunca fará justiça aos professores dedicados.

Agradeço aos meus eternos mestres, artistas-educadores do espaço Quixote: Andreia de Almeida, Casper, Fabio Prior, Gabriel Draetta, Laura Nogueira, Luiz dos Santos, Marcelo Bolognesi, Ronaldo Crispin, Zuza Gonçalves e a todos os companheiros, em especial aos companheiros mais próximos da Trupe Quixotesca.

Agradeço aos melhores professores da faculdade, que sem eles o curso com certeza não seria o mesmo, por elevarem o nível da faculdade e pela excelência: Ana Claudia, Ana Paula, Andreia Machion, Antonio Guardado, Carlos Menezes, Marcio Fontoura, Marcio Rodrigues, Rodrigo Bossini e Simone Telles.

Por fim, a todos que direta ou indiretamente fizeram parte da minha formação, о meu muito obrigado.

*“The isolated man does not develop any intellectual power. It is necessary for him to be immersed in an environment of other men, whose techniques he absorbs during the first 20 years of his life. He may then perhaps do a little research of his own and make a very few discoveries which are passed on to other men.”*

*(Alan Mathison Turing*)

RESUMO

A análise de tendências dos preços das ações no mercado financeiro exige um número relevante de variáveis. Muitos estudos têm sido realizados a fim de se obter modelos que forneçam recomendações de compra e venda de ações, de modo a maximizar os lucros. Considerando que a Bovespa disponibiliza dados referentes às trocas de mensagens entre as partes que constituem o mercado financeiro, as redes neurais artificiais podem ser uma técnica adequada para situações com diversas variáveis e padrões a serem descobertos durante essa troca de mensagens. Para isso, é necessário extrair os dados da BM&FBOVESPA, normalizá-los, criar uma rede neural artificial que modele esses dados e sua dinâmica, configurá-la, treiná-la e verificar se ela é eficiente. Assim, este trabalho tem como objetivo construir um software que facilite todas essas etapas para que até uma pessoa sem conhecimento em programação possa criar uma rede neural artificial eficiente. O resultado obtido foi um programa que utiliza redes neurais artificiais que, sob determinadas condições, produzem níveis admissíveis de predição e previsão, que pode ser utilizado como ferramenta para auxílio na tomada de decisão de um investidor.

Palavras-chave: Inteligência Artificial. Mercado financeiro. Predição. Previsão. Redes neurais artificiais

ABSTRACT

The analysis of stock price trends in the financial market requires a relevant number of variables that can influence. Many studies have been conducted in order to obtain models that provide recommendations for buying and selling stocks in order to maximize profits. Considering that the Bovespa provides data regarding the exchange of messages between the parties that constitute the financial market, artificial neural networks may be a suitable technique for situations with several variables and patterns to be discovered during this exchange of messages. For this, it is necessary to extract data from BMF&BOVESPA, normalize them, create an artificial neural network that models this data and its dynamics, configure it, train it and verify if this network is efficient. Thus, this work aims build a software that facilitates all these steps so that even a person without programming knowledge can create an efficient artificial neural network. The result was a software that uses artificial neural networks that, under certain conditions, produce allowable levels of forecasting and prediction, that can be used as a tool to aid in the decision making of an investor.

Keywords: Artificial Intelligence. Artificial neural networks. Forecasting. Prediction. Stock market..

LISTA DE FIGURAS

[Figura 1. Modelo de um Neurônio Artificial 21](#_Toc489064457)

[Figura 2. Funções de ativação 25](#_Toc489064458)

[Figura 3. Multilayer perceptron (MLP) 26](#_Toc489064459)

[Figura 4. TLFN focada. 27](#_Toc489064460)

[Figura 5. Recurrent Network 28](#_Toc489064461)

[Figura 6. Hierarquia do aprendizado 29](#_Toc489064462)

[Figura 7. Redes com e sem *dropout* 30](#_Toc489064463)

[Figura 8. Maiores altas em 5 dias 34](#_Toc489064464)

[Figura 9. Maiores quedas em 5 dias 35](#_Toc489064465)

[Figura 10. Diagrama de caso de uso 42](#_Toc489064466)

[Figura 11. Diagrama de Classes de negócio 50](#_Toc489064467)

[Figura 12. Diagrama de implantação 51](#_Toc489064468)

[Figura 13. Diagrama de classes de projeto – *Create NN* 52](#_Toc489064469)

[Figura 14. Diagrama de sequência - *Create NN* 52](#_Toc489064470)

[Figura 15. Diagrama de classes de projeto - *Choose NN* 53](#_Toc489064471)

[Figura 16. Diagrama de sequência - Choose NN 53](#_Toc489064472)

[Figura 17. Diagrama de classes de projeto *- Import BMF Data* 54](#_Toc489064473)

[Figura 18. Diagrama de sequência *- Import BMF Data* 54](#_Toc489064474)

[Figura 19. Diagrama de classes de projeto - *Comparative Chart* 55](#_Toc489064475)

[Figura 20. Diagrama de sequência - *Comparative Chart* 55](#_Toc489064476)

[Figura 21. Diagrama de classes de projeto *- Train NN* 56](#_Toc489064477)

[Figura 22. Diagrama de sequência - *Train NN* 57](#_Toc489064478)

[Figura 23. Topologia da RNA utilizada 59](#_Toc489064479)

[Figura 24. Código implementação topologia 60](#_Toc489064480)

[Figura 25. Código *Factory* função de ativação 61](#_Toc489064481)

[Figura 26. Ajuste *Date Interval* 63](#_Toc489064482)

[Figura 27. Código *mixed data* 64](#_Toc489064483)

[Figura 28. Código *calculated data* 65](#_Toc489064484)

[Figura 29. Código *normalize data* 66](#_Toc489064485)

[Figura 30. Código *get max and min* 67](#_Toc489064486)

[Figura 31. Código *getNormalizedValue* 69](#_Toc489064487)

[Figura 32. Diferença entre *inner class* e *lambda* 70](#_Toc489064488)

[Figura 33. Código *lambda* e *method* *reference* 71](#_Toc489064489)

[Figura 34. Código *compareByVolume* 72](#_Toc489064490)

[Figura 35. Código de ordenação 72](#_Toc489064491)

[Figura 36. Código exemplo *stream* – pegando dados 73](#_Toc489064492)

[Figura 37. Exemplo código *stream* - sobrescrita *toString* 74](#_Toc489064493)

[Figura 38. Saída do código de exemplo da *stream* 75](#_Toc489064494)

[Figura 39. Código exemplo *stream* 76](#_Toc489064495)

[Figura 40. Ilustração código stream 78](#_Toc489064496)

[Figura 41. Código *stream* com paralelismo 79](#_Toc489064497)

[Figura 42. Código exemplo paralelismo 80](#_Toc489064498)

[Figura 43. Saída código exemplo paralelismo 80](#_Toc489064499)

[Figura 44. Código updateMaxMin 82](#_Toc489064500)

[Figura 45. Interface 1 - *initial* 84](#_Toc489064501)

[Figura 46. Interface 1 - *error* 85](#_Toc489064502)

[Figura 47. Interface 1 – *success* 86](#_Toc489064503)

[Figura 48. Interface 1 - *add hidden layer* 87](#_Toc489064504)

[Figura 49. Interface 1 - *activation functions* 88](#_Toc489064505)

[Figura 50. Interface 1 – *overwrite* 89](#_Toc489064506)

[Figura 51. Interface 1 – *don’t overwrite* 90](#_Toc489064507)

[Figura 52. Interface 1 *- NN don't exists* 92](#_Toc489064508)

[Figura 53. Interface 1 - *NN loaded* 93](#_Toc489064509)

[Figura 54. Interface 2 – initial 94](#_Toc489064510)

[Figura 55. Interface 2 - *null dates* 95](#_Toc489064511)

[Figura 56. Interface 2 - *null NN* 96](#_Toc489064512)

[Figura 57. Interface 2 - *Learning Rules* 97](#_Toc489064513)

[Figura 58. Interface 2 – *dates* 98](#_Toc489064514)

[Figura 59. Interface 2 - *from after to* 99](#_Toc489064515)

[Figura 60. Interface 2 - *don't have data form selected date* 100](#_Toc489064516)

[Figura 61. Interface 2 – *trained* 101](#_Toc489064517)

[Figura 62. Interface 3 – *initial* 102](#_Toc489064518)

[Figura 63. Interface 3 - *null NN* 103](#_Toc489064519)

[Figura 64. Interface 3 - *report created* 104](#_Toc489064520)

[Figura 65. Interface 3 - *report with 2 attributes* 105](#_Toc489064521)

[Figura 66. Under fitting 106](#_Toc489064522)

[Figura 67. Over fitting - Trainning 107](#_Toc489064523)

[Figura 68. Over fitting - Report 108](#_Toc489064524)

[Figura 69. Topologia aleatória 109](#_Toc489064525)

[Figura 70. Configuração RNA 1 110](#_Toc489064526)

[Figura 71. Resultado RNA 1 111](#_Toc489064527)

[Figura 72. Resultado RNA 2 112](#_Toc489064528)

[Figura 73. Resultado RNA 3 113](#_Toc489064529)

[Figura 74. Resultado RNA 4 114](#_Toc489064530)

[Figura 75. Resultado RNA 5 115](#_Toc489064531)

LISTA DE TABELAS

[Tabela 1. Preços PRIO3 “puros” 33](#_Toc489064532)

[Tabela 2. PRIO3 normalizado 34](#_Toc489064533)

[Tabela 3. Comparativo entre os softwares descritos 39](#_Toc489064534)

LISTA DE EQUAÇÕES

[Equação 1. Neurônio Artificial 22](#_Toc489064535)

[Equação 2. Normalização 31](#_Toc489064536)

[Equação 3. Normalização com margem 32](#_Toc489064537)

[Equação 4. Normalização exemplo 33](#_Toc489064538)

LISTA DE ABREVIATUAS E SIGLAS

RNA - Rede Neural Artificial

LOG - Logaritmo

TANH - Tangente Hiperbólica

MLP - *Multilayer Perceptron*

TLFN - *Time Lagged Feed-forward Network*

IDE - *Integrated Development Environment*

API - *Application Programming Interface*

AI - *Artificial Intelligence*

IBM - *International Business Machines*

SUMÁRIO

[1 INTRODUÇÃO 17](#_Toc489064539)

[1.1 Objetivos 19](#_Toc489064540)

[1.2 Justificativa 19](#_Toc489064541)

[1.3 Tecnologias e Ferramentas de Apoio 20](#_Toc489064542)

[1.4 Organização do Trabalho 20](#_Toc489064543)

[2 REDES NEURAIS ARTIFICIAIS 21](#_Toc489064544)

[2.1 Motivação para utilizar Redes Neurais na Predição. 23](#_Toc489064545)

[2.2 Funções de Ativação 23](#_Toc489064546)

[2.3 Topologias das Redes Neurais 25](#_Toc489064547)

[2.3.1 Feed-Forward 25](#_Toc489064548)

[2.3.2 Recurrent Network 27](#_Toc489064549)

[2.4 Aprendizagem 28](#_Toc489064550)

[2.4.1 Backpropagation 30](#_Toc489064551)

[2.5 Normalização 31](#_Toc489064552)

[3 FERRAMENTAS PARA CRIAÇÃO DE RNA 36](#_Toc489064553)

[3.1 Excel + VBA 36](#_Toc489064554)

[3.2 Weka 37](#_Toc489064555)

[3.3 Neuroph Studio 37](#_Toc489064556)

[3.4 Matlab 38](#_Toc489064557)

[3.5 NeuroFURG 38](#_Toc489064558)

[3.6 Comparativo 39](#_Toc489064559)

[4 REQUISITOS DO SISTEMA DE SOFTWARE 40](#_Toc489064560)

[4.1 Identificação dos requisitos 40](#_Toc489064561)

[4.1.1 Requisitos Funcionais 40](#_Toc489064562)

[4.2 Modelagem dos requisitos funcionais 41](#_Toc489064563)

[4.2.1 Atores 41](#_Toc489064564)

[4.2.2 Diagrama de caso de uso 41](#_Toc489064565)

[4.2.3 Especificação dos Casos de Uso 42](#_Toc489064566)

[5 Desenvolvimento do projeto 48](#_Toc489064567)

[5.1 Análise 48](#_Toc489064568)

[5.1.1 Diagrama de Classes de Análise (Visão de Negócio) 48](#_Toc489064569)

[5.2 Projeto 50](#_Toc489064570)

[5.2.1 Arquitetura do Sistema 50](#_Toc489064571)

[5.2.2 Diagrama de Classes de Projeto por Caso de Uso 51](#_Toc489064572)

[6 Detalhes de implementação. 58](#_Toc489064573)

[6.1 Topologia 58](#_Toc489064574)

[6.2 Função de ativação 61](#_Toc489064575)

[6.3 Date Interval 62](#_Toc489064576)

[6.4 Normalizador de dados 66](#_Toc489064577)

[6.5 Recursos do Java 8 70](#_Toc489064578)

[6.5.1 Lambda 70](#_Toc489064579)

[6.5.2 Method Reference 71](#_Toc489064580)

[6.5.3 Stream 73](#_Toc489064581)

[7 Resultados Obtidos 83](#_Toc489064582)

[7.1 Passos do sistema 84](#_Toc489064583)

[7.1.1 Criar RNA 84](#_Toc489064584)

[7.1.2 Escolher RNA 91](#_Toc489064585)

[7.1.3 Treinar RNA 93](#_Toc489064586)

[7.1.4 Gerar Relatório 102](#_Toc489064587)

[7.2 Estresse de uma RNA 106](#_Toc489064588)

[7.2.1 Baixo treinamento (under fitting) 106](#_Toc489064589)

[7.2.2 Alto treinamento (over fitting) 107](#_Toc489064590)

[7.2.3 Rede eficiente para prever PETR4 em maio de 2017 109](#_Toc489064591)

[8 Considerações Finais 116](#_Toc489064592)

[8.1 Trabalhos futuros 117](#_Toc489064593)

[8.1.1 Gráfico 117](#_Toc489064594)

[8.1.2 Comparativo entre outras RNA 117](#_Toc489064595)

[8.1.3 Base de dados 118](#_Toc489064596)

[8.1.4 Outros relatórios 118](#_Toc489064597)

[8.1.5 Layout 119](#_Toc489064598)

[Bibliografia 120](#_Toc489064599)

# INTRODUÇÃO

As ações são títulos representativos do capital social de uma companhia aberta, que é negociada em bolsa de valores. Seu preço é fruto das condições de oferta e demanda que refletem a economia do país, específicas da empresa e de seu setor econômico (FORTUNA, 2008).

Tentar prever os preços das ações na bolsa de valores é algo que todo mundo que conhece um pouco deste mundo já sonhou algum dia. O segundo estágio para escolher um investimento começa com crenças relevantes sobre o desempenho futuro, tais como possíveis descontos ou retornos sobre os investimentos. Um exemplo é vender uma ação a um preço maior do que foi pago por ela e, neste caso, nota-se a importância de prever os preços futuros (MARKOWITZ, 1952).

O estudo de Inteligência Artificial (IA) tem em uma de suas razões o entendimento do ser humano, mas não como filosofia ou psicologia, que também se concentra na inteligência, mas em como construir entidades inteligentes. Outra razão é a importância desta construção por si só. É notório que ninguém consegue prever o futuro em detalhes, mas um computador com a capacidade da inteligência humana pode ter um grande impacto no dia-a-dia e no futuro da civilização (RUSSELL e NORVIG, 2005).

Existe uma empresa que trabalha com IA aplicada ao mercado financeiro, criada no final de 2015 como um torneio para a predição e previsão de dados do mercado. Ela utiliza as melhores redes neurais artificiais para os próprios fundos de investimentos e com os ganhos, usa uma parte para pagar as pessoas que obtiveram o menor erro em relação aos dados reais com Bitcoins. Além disso, é gratuito e livre para qualquer pessoa submeter os resultados obtidos pelas suas redes (NUMERAI, 2016).

Sobre a origem da IA, um questionamento muito importante no começo da sua evolução é: um computador pode pensar? Essa foi uma pergunta com a qual Alan Turing (1950), precursor da IA, começou sua publicação. Ele faz uma comparação com a objeção de Lady Lovelace, que diz que uma máquina só pode fazer o que dizem para ela fazer. Esta pergunta continua sendo atual, é muito difícil responder se um computador pode pensar ou não.

Para tentar responder a essa pergunta, uma motivação para se trabalhar com redes neurais é a diferença entre o processamento realizado por um cérebro humano e por um computador convencional. O cérebro é altamente complexo, não linear e processa as informações de forma paralela e muito rapidamente. Há alguns casos em que o cérebro humano é muito superior, em capacidade de processamento e entendimento, do que um computador, como é o caso da visão, ou o reconhecimento de um de um rosto familiar dentre vários rostos desconhecidos em aproximadamente 150 *ms* (HAYKIN, 1998).

Por outro lado, uma das conquistas mais simbólicas e famosas do uso da aplicação da IA ocorreu em maio de 1997, na qual *Deep Blue*, um computador da IBM, conquistou um feito inédito que foi ganhar de um dos melhores jogadores de xadrez da história, Garry Kasparov, com um placar de duas vitórias, uma derrota e três empates para a máquina. Esse feito foi importante não só por vencer uma partida de xadrez, mas pela arquitetura usada na construção dessa brilhante máquina que foi aplicada à análise de riscos para o mercado financeiro, à dinâmica molecular para descobrir e desenvolver novos remédios, à mineração de dados e, principalmente, ao desenvolvimento e entendimento sobre os limites da programação e da arquitetura paralela de uma forma massiva (IBM, 2011).

Não é de hoje que alguns fundos de investimentos usam de alguma forma o poder da computação para auxiliar a atuar no mercado financeiro. Esse poder vai desde o uso de negociação algorítmicapara operar em mercados, como em Chabou, Chiquioine, *et al* (2014) , o uso da IA para tomada de decisões, inclusive passando por momentos difíceis nessa área, como o recente *Brexit* (Saída do Reino Unido da União Europeia),e conseguindo ótimos retornos, como no caso da Kathleen e Komaki (2016), e impactando na liquidez dos investimentos, como mostrado pelo trabalho de Hendershott, Jones e Menkveld (2011).

Vários trabalhos acadêmicos foram realizados demonstrando a eficácia do uso de IA, mais especificamente das redes neurais artificiais (RNA) para predição dos preços das ações. Um dos primeiros, feito por Kimoto, Asakawa, *et al.* (1990), que pode predizer tendências no índice de Tóquio, ou o realizado por Thomaz e Vellasco (2005) que desenvolveu um modelo para vários indicativos financeiros, ou o trabalho de Refenes, Zapranis e Francis (1994) que criou uma RNA para prever medias moveis. O trabalho de Moreira (2013) mostrou como a técnica de mineração de dados se mostra eficaz para a predição de valores ao longo do dia. Os trabalhos elencados não abordavam a criação de uma rede, somente sua aplicação. Desta forma, a partir da fundamentação desenvolvida, este projeto propõe o desenvolvimento de um software para auxiliar nas tarefas da criação da rede neural, extração dos dados, normalização, treinamento e validação da rede.

## Objetivos

O objetivo deste trabalho é desenvolver um software para auxiliar nas tarefas da criação da rede neural, extração dos dados, normalização, treinamento e validação da rede.

## Justificativa

Uma das tarefas da análise de dados do mercado financeiro é identificar padrões nos preços das ações, que podem servir para predição e previsão de preços. Entretanto, os históricos de cotações são naturalmente dispostos em séries temporais, que necessitam de grande esforço tanto para extração das informações quanto para sua modelagem e tudo isso requer bons conhecimentos de programação e de manipulação de softwares.

Acredita-se que um software que incorpore facilidades de manuseio e opções de alto nível de abstração para que um usuário obtenha seus resultados com a precisão desejada, seja por meio da escolha da base de dados a ser analisada, seja pela configuração do algoritmo a ser utilizado (no caso, uma RNA), seja de grande utilidade para que o usuário consiga ter predições confiáveis para atingir seus objetivos.

## Tecnologias e Ferramentas de Apoio

O software está escrito na linguagem Java 8 (ORACLE, 2014), com a biblioteca YahooFinance API (STRICKX, 2014) para extração dos dados e Encog Framework (RESEARCH, 2013) para a criação da RNA.

## Organização do Trabalho

Este trabalho é composto por oito capítulos, incluindo esta introdução. O Capítulo 2 introduz os conceitos gerais sobre RNAs. O Capítulo 3 apresenta algumas ferramentas para a criação de RNAs, analisando suas contribuições e o que deixam a desejar para esta tarefa específica. O Capítulo 4 apresenta os requisitos do software desenvolvido. O Capítulo 5 apresenta o desenvolvimento do projeto. O Capítulo 6 apresenta alguns detalhes técnicos da implementação do sistema. O Capítulo 7 apresenta os resultados obtidos. O Capítulo 8 apresenta as considerações finais.

# REDES NEURAIS ARTIFICIAIS

Rede Neural Artificial pode ser definida como um processador massivamente distribuído e paralelo, feito de unidades de processamentos simples que tem uma propensão natural em armazenar conhecimentos empíricos e tornar isso disponível para o uso. Lembra um cérebro em dois aspectos (HAYKIN, 2009):

1. O conhecimento é adquirido pela rede a partir do ambiente em que se encontra durante um processo de aprendizagem
2. Os pesos sinápticos, ou forças de interconexão, são usados para armazenar os conhecimentos obtidos.

Assim, pode-se dizer que uma RNA é uma combinação de neurônios artificiais, portanto, para entender uma RNA, é imprescindível saber como funciona um neurônio artificial (Figura 1). Segundo (TAFNER, 1998, p. 1)

O neurônio artificial é uma estrutura lógico-matemática que procura simular a forma, o comportamento e as funções de um neurônio biológico. Assim sendo, os dendritos foram substituídos por *entradas*, cujas ligações com o corpo celular artificial são realizadas através de elementos chamados de *peso* (simulando as sinapses). Os estímulos captados pelas entradas são processados pela *função de soma*, e o limiar de disparo do neurônio biológico foi substituído pela *função de transferência*.

Figura 1. Modelo de um Neurônio Artificial



Fonte: (VON ZUBEN e ATTUX, 2010).

A Equação 1 mostra a fórmula de um neurônio artificial com o bias (VON ZUBEN e ATTUX, 2010)

Equação 1. Neurônio Artificial



Na Equação 1, yk é a saída do neurônio, f representa a função de ativação, dentro dos parênteses um somatório dos sinais de entrada (xj) com os pesos sinápticos (wkj) com o bias (bk). O bias tem o efeito de deslocar a função linear da somatória dos pesos, tanto para cima (quando positivo) quanto para baixo (quando negativo), transformando a saída do neurônio (HAYKIN, 2009).

## Motivação para utilizar Redes Neurais na Predição.

O principal benefício do uso de uma RNA é a capacidade de generalização, que é a capacidade de uma rede aprender, isto é, produzir uma saída adequada mesmo para entradas não encontradas durante o treinamento (HAYKIN, 2009). Um exemplo prático: uma alteração grande no volume financeiro negociado geralmente indica um aumento na volatilidade das ações.

Outro ponto importante é a adaptabilidade, que é a capacidade de uma RNA alterar seus pesos sinápticos, devido a modificações no meio ambiente, em tempo real (HAYKIN, 2009). No contexto complexo do mercado financeiro, no qual diversos fatores podem afetar o preço de uma ação, como fatores psicológicos, notícias, divulgação de resultados, regulamentação, entre outros, essa capacidade é essencial.

## Funções de Ativação

A função de ativação restringe o valor recebido pela função soma, limitando a amplitude da saída de um neurônio a um intervalo de [0, 1] ou [-1, 1] (HAYKIN, 2009). Ela tem a função de manter o neurônio ativado quando as entradas fornecidas forem corretas ou mantê-los inativo quando forem incorretas (RUSSELL e NORVIG, 2005). Existem vários tipos de funções de ativação, dentre elas a função linear (a), função rampa (b), função *step* (c) e função sigmoide (d). Essa última é a mais utilizada e é uma mistura entre funções lineares e não lineares (HAYKIN, 2009), que são exemplificadas na

Figura 2.

Figura 2. Funções de ativação



Fonte: (DO CARMO ROQUE e DE MELLO, 2009)

O software disponibiliza várias funções de ativação: Bi Polar, Competitive, Linear, LOG, Sigmoid, Soft Max, TanH (HEATON, 2011), Elliott, Symmetric Elliott (WINRICH, 2015)

## Topologias das Redes Neurais

Existem dois tipos de topologias para uma RNA: *feed-forward* (alinhadas à frente) e *recurrent network* (redes recorrentes) (RUSSELL e NORVIG, 2005).

### Feed-Forward

As redes *feed-forward* podem conter uma (*single-layer*) ou mais camadas (*multilayer*). Existe uma camada de entrada e uma de saída (nesta ordem), e as demais camadas que estão entre elas, também chamadas de ocultas. Adicionando essas camadas ocultas, a rede tem a possibilidade de extrair estatísticas mais robustas da camada de entrada. (HAYKIN, 2009)

A Figura 3, mostra que os dados têm um fluxo que vai da camada de entrada, passando pelas camadas escondidas (se houver) e por fim vai para a camada de saída. Além disso, após passar por uma camada, o dado nunca volta para uma camada anterior nem permanece na atual.

Figura 3. Multilayer perceptron (MLP)



Fonte: (HAYKIN, 2009)

#### Incorporação do Tempo em Feed-Forward

Dentre algumas aplicações que necessitam a incorporação do tempo, destaca-se a previsão e modelagem de séries temporais. Para que uma rede tenha esse dinamismo, é necessário que possua memória, podendo ser de curto ou longo prazo. Para fazer isso, uma das formas possíveis é adicionar atrasos de tempo, que podem ser adicionados nos pesos sinápticos ou na entrada da rede (HAYKIN, 2009).

Um exemplo para este caso é a *focused time lagged feed-forward network* (TLFN focada), na qual os atrasos de tempo são incorporados somente na camada de entrada.

Na Figura 4 é ilustrada uma TLFN focada, na qual as entradas são apresentadas à camada de entrada neurônio por neurônio, ou seja, com um *delay.* O primeiro neurônio alimentado com o dado do instante n, o segundo com o dado do n*-1* e assim por diante, até atingir todos os neurônios da camada (p).

Figura 4. TLFN focada.



Fonte: (HAYKIN, 1998)

### Recurrent Network

Outra forma de dar dinamismo e incorporar o fator tempo a uma rede é com o uso de uma rede recorrente, que é o caso deste trabalho. A diferença entre a rede recorrente e a *feed-forward* é que ao menos um neurônio é alimentado por outro que se encontra na mesma camada ou em uma camada posterior. Este mecanismo gera um ciclo de *feedback*, que tem um profundo impacto na capacidade de aprendizagem da rede e em seu desempenho (HAYKIN, 2009).

Um exemplo de uso para redes recorrentes é para a previsão de séries temporais, nas quais o resultado de um período é fonte de dado para outro período (HAYKIN, 2009).

Na Figura 5 é ilustrada uma rede recorrente, e nota-se a diferença dita anteriormente: o fluxo de dados tem uma parte cíclica.

Figura 5. Recurrent Network



Fonte: (HAYKIN, 2009)

Não existem regras para a escolha da topologia da rede na questão de quantos neurônios ou quantas camadas, somente indicações. Um bom método para a escolha é por meio da "tentativa e erro", baseando-se em trabalhos anteriores (MANTOVANI, 2011).

## Aprendizagem

A aprendizagem consiste no ajuste dos pesos das sinapses para obter as saídas desejadas. Ela é um dos fatores mais importantes na boa formação da RNA. (HEATON, 2011).

Monard e Baranauskas (2003) explicam a **Erro! Autoreferência de indicador não válida.**, suas tipologias e vários paradigmas de aprendizado. Neste trabalho será explicado somente o aprendizado supervisionado, que será utilizado.

Figura 6. Hierarquia do aprendizado



Fonte: (MONARD e BARANAUSKAS, 2003).

Um ponto importante a ser considerado é a escolha de atributos com boa capacidade preditiva. Não importa qual método seja empregado, os conceitos que podem ser aprendidos estão à mercê dos dados e da qualidade dos atributos. Por exemplo, para a tarefa de determinar se uma pessoa está ou não com gripe, pode-se escolher atributos com baixo poder preditivo, tais como (cor-do-cabelo, cor-do-olho, modelo-do-carro, número-de-filhos) ou atributos com alto poder preditivo, tais como (temperatura, resistência-da-pele, exame do-pulmão). Para esta tarefa específica, no segundo caso, melhores previsões em exemplos não-rotulados provavelmente ocorrerão do que com o primeiro conjunto de atributos. (MONARD e BARANAUSKAS, 2003, p.43).

Usando os pontos levantados pela citação anterior e conversas com especialistas no mercado financeiro, foi decidido deixar disponíveis os seguintes dados para formar a entrada: preço de abertura, preço máximo do dia, preço mínimo do dia, preço de fechamento (cotação) e volume.

Além disso, existe um parâmetro importante a ser considerado: o *dropout rate* que é utilizado para evitar que a rede decore ao invés de aprender, conhecido como *overfitting*. Com o uso do *dropout*, alguns neurônios são desativados, como se fossem removidos temporariamente (daí o nome *drop*), como se pode verificar comparando-se as redes ilustradas nas Figura 7 (a) e (b).

Figura 7. Redes com e sem *dropout*



Fonte: (SRIVASTAVA, HINTON, *et al.*, 2014)

A seguir são apresentados os principais algoritmos de treinamento que o software desenvolvido suporta.

### Backpropagation

Chauvin e Rumelhart (1995) abordam o funcionamento e algumas aplicações do algoritmo. Uma breve explicação pode ser feita da seguinte forma:

1. Após os dados chegarem à saída, são analisados os erros dessa camada. É determinado o erro de acordo com a contribuição de cada neurônio da camada de saída, usando como base seu conhecimento (peso).
2. Esses pesos são ajustados para minimizar o erro da saída.
3. Este processo continua de forma regressiva pelas camadas da rede (por isso o nome *Backpropagation*).

Este foi um dos primeiros algoritmos desenvolvido para realizar o treinamento de uma rede neural. Naturalmente, depois de muitos estudos realizados, foi possível alcançar melhores resultados a partir de novos algoritmos baseados neste. Para o desenvolvimento do software, além do *Backpropagation*, estará disponível os seguintes algoritmos: *Resilient Propagation* (RIEDMILLER e BRAUN, 1993), *Manhattan Propagation* (HEATON RESERACH, 2014), *Quick Propagation* (FAHLMAN, 1988),  *Scaled Conjugate Gradient* (MØLLER, 1993).

## Normalização

Apesar de uma RNA ser bem “inteligente”, ela não consegue processar o dado de qualquer forma. Ela aceita receber números de ponto flutuante (*float*), de acordo com a função de ativação (entre -1 e 1, ou 0 e 1) (HEATON, 2011).

A ideia básica é verificar qual é o valor máximo possível, o mínimo e ver a razão do número entre eles, como mostra na Equação 2.

Equação 2. Normalização

Neste caso, o *x* é o valor a ser normalizado, *xmax* e *xmin* são os valores extremos da entrada, e *d*1e *d*2 são os valores extremos da saída. Considere que existem n dados de entrada no intervalo [0,10], então xmin = 0 e xmax = 10. Esses dados devem ser reduzidos ao intervalo [0,1], então d1 = 0 e d2 = 1.

Para a implementação do software, além desses limites, foi acrescentado um valor de margem, a fim de evitar que um possível resultado não conhecido venha a interferir na rede como um todo.

Como mostra a Equação 3 a margem é adicionada somente aos valores extremos. É possível observar que para fazer este procedimento, precisa-se percorrer toda a entrada dos dados e descobrir quais são os valores extremos.

Equação 3. Normalização com margem

Para a Tabela 1, foi utilizada a ação de Petro Rio (PRIO3) que teve uma alta de cerca de 32% em 13 dias, a fim de exemplificar como os dados são utilizados pelo software. Ela é separada entre os valores dos preços e em quantidades de ações negociadas (volume). Essa separação é importante para que os significados de cada valor sejam preservados e que o limite tenha um sentido.

As marcas verdes representam os valores máximos da amostra, enquanto que as vermelhas representam os valores mínimos. Esses valores são úteis para a normalização dos dados, conforme mostrado anteriormente.

Tabela 1. Preços PRIO3 “puros”



Fonte: Autor (2017)

Para uma margem de 100%, ou seja, os valores extremos dos preços serão 0 e 36 e dos volumes, 0 e 57400 e para uma entrada entre 0 e 1 a Equação 4Equação 3 mostra o que foi explicado.

Equação 4. Normalização exemplo



Na Tabela 2 mostra os dados normalizados conforme as condições anteriores. Para o volume, usa-se a mesma ideia, porém os valores para *xmax* e *xmin* são 57400 e 0, respectivamente.

Tabela 2. PRIO3 normalizado



Fonte: Autor (2017)

O sugerido pelo software, ao treinar uma rede é 20%, ou seja, uma ação pode alterar em 20% seus preços limites (dobrar seu preço máximo ou “zerar”) sem que isso interfira na rede. Este é um ambiente totalmente possível, como mostram a Figura 8 e a Figura 9.

Figura 8. Maiores altas em 5 dias



Fonte: (BARCHART, 2016)

Figura 9. Maiores quedas em 5 dias



Fonte: (BARCHART, 2016)

# FERRAMENTAS PARA CRIAÇÃO DE RNA

Neste capítulo apresentam-se alguns softwares disponíveis que contribuem na criação de uma RNA que faz predição dos preços de ações, bem como a comparação entre eles. A comparação é feita com base na quantidade de etapas que o software consegue realizar em relação ao total de etapas necessárias e desejáveis.

## Excel + VBA

O Excel é um dos mais famosos editores de planilhas no mercado. Com ele é possível descobrir as ideias ocultas nos dados com a estruturação, formatação, organização e análise, mostrar os dados da melhor forma com o uso de gráficos, tendências e padrões em evidência e compartilhar e colaborar o arquivo com outras pessoas (MICROSOFT, 2016).

A linguagem de programação VBA permite customizar os aplicativos do Office. Com ela é possível automatizar tarefas repetitivas e integrar as aplicações do Office (MICROSOFT, 2009).

No caso da análise de ações, com o Excel é possível criar campos para escolher o período para o qual se deseja extrair os dados, escolher os papeis e os campos. Após essa operação, com o uso do VBA faz-se a extração dos dados de alguma fonte, como por exemplo, do site do Yahoo. De posse dos dados, descobrem-se os valores limites da amostra e faz-se a normalização dos valores com possíveis margens no Excel. A etapa de treinamento da rede é possível com o uso de alguns módulos de extensão (*add-ins*) como o NeuroXL (FRANZ AG, 1998), ou o NeuroNetwork Add-in (ADELAIDE, 2010). Essas opções não são completas, mas é possível criar uma rede, mesmo que bem simples. A etapa de geração do relatório pode ser executada facilmente com a criação de gráficos no Excel e com cálculos de medidas estatísticas relevantes disponíveis no software Excel.

## Weka

Weka é uma poderosa coleção de algoritmos de *machine learning* no contexto de *data mining*. O nome faz a referência a um pássaro que não voa, encontrado somente na Nova Zelândia, já que foi desenvolvido na Universidade de Waikato localizado neste país (UNIVERSITY OF WAIKATO, 2010).

Com este software é possível normalizar os dados, escolhendo-se apenas os limites (BROWNLEE, 2016). Não é possível escolher uma margem de forma direta, somente com aplicação da Equação 3, sendo assim, essa capacidade será desconsiderada.

Para a criação da rede, podem-se escolher quantas camadas a rede terá, bem como quantos neurônios por camada. A escolha da função de ativação, bem como do algoritmo de aprendizagem não é suportada. Também é possível escolher os períodos de treinamento e de validação e configurar alguns avisos caso a rede dê algum tipo de erro (WEKA, 2010).

## Neuroph Studio

O Neuroph Studio é um framework leve para Java que auxilia o desenvolvimento de arquiteturas de redes neurais, com o intuito de simplificar esse desenvolvimento. (NEUROPH, 2008).

Com o uso deste software, é possível criar uma RNA bem completa, sendo possível escolher quantos neurônios por camada, quantas camadas, a função de ativação (chamada função de transferência) por neurônio e o algoritmo de aprendizado (chamado regra de aprendizado). Como no caso do Weka, só é possível fazer a normalização dos valores somente escolhendo os limites, sem a margem (NEUROPH, 2014).

Como resultado, tem-se um gráfico com o total de erros por iteração, mas medidas estatísticas como desvio padrão e variância não são nativas (NEUROPH, 2014).

## Matlab

O Matlab é uma ferramenta voltada para otimização ou resolução de problemas científicos ou da engenharia. Ele possui uma linguagem de programação própria e está presente em sistemas de segurança automotiva, veículos espaciais interplanetários, dispositivos de monitoramento de saúde, redes de internet de celulares, entre outros. É usado para algoritmos de *machine learning*, processamento de imagem, visão computacional, comunicação, computação financeira, robóticas e muito mais (MATHWORKS, 2007).

A normalização é realizada somente com os limites, sem as margens. É possível definir quantas camadas e quantos neurônios por camada, escolher a função de ativação por camada e o algoritmo de treinamento. Além disso, ele também gera um gráfico com o erro por tempo de treinamento (época) (DEMUTH e BEALE, 2003).

## NeuroFURG

O NeuroFURG é uma ferramenta de construção e simulação de RNAs com o foco nos modelos de neurônios Perceptron e Adaline (esse último não abordado neste trabalho) (NEUROFURG, 2011).

Esta ferramenta é muito simples e não tem a funcionalidade de adicionar camadas ocultas, ou escolher quantos neurônios por camada, ou até mesmo a função de ativação, nem o algoritmo de aprendizagem. Com ele só dá para fazer uma rede com dois neurônios, e sem muita complexidade. É possível gerar um gráfico com o erro (NEUROFURG, 2011).

## Comparativo

Nesta seção será apresentado o comparativo entre os softwares supracitados. Pode-se notar que pela Tabela 3 nenhum software é capaz de realizar todas as operações. Enquanto alguns são voltados somente às redes em si, outros permitem fazer a normalização dos dados e a extração.

Tabela 3. Comparativo entre os softwares descritos



Fonte: Autor (2017)

# REQUISITOS DO SISTEMA DE SOFTWARE

Este capítulo tem como objetivo especificar os requisitos funcionais, não funcionais e a regras de negócio, bem como apresentar o protótipo de telas e o cronograma de atividades do desenvolvimento do software.

## Identificação dos requisitos

### Requisitos Funcionais

Nesta seção são descritos os requisitos funcionais que especificam ações que um sistema deve ser capaz de executar, ou seja, os objetivos do sistema, incluindo prioridade e regras de negócio. (SOMMERVILLE, 2010)

**[RF001] – Importar dados da BMF**

**Descrição**: Este requisito permite que o sistema importe os preços de fechamento e os volumes ajustados por proventos da Bolsa de Valores do Brasil de uma ação específica e em um intervalo de datas específico.

**[RF002] – Gerenciar rede neural**

**Descrição**: Este requisito permite que o usuário gerencie uma RNA, sendo possível criar uma RNA (configurar a topologia, qual ação será prevista, quantos dias precisa para prever o próximo) salvar essa rede localmente em um formato definido pelo framework Encog (.eg) e recuperara-la.

**[RF003] – Treinar rede neural**

**Descrição**: Este requisito permite que o usuário configure o ambiente de treinamento de uma RNA.

**[RF004] – Gerar comparativo entre os dados calculados pela rede e o real**

**Descrição**: O sistema deve gerar gráfico comparando os preços esperados e os calculados.

## Modelagem dos requisitos funcionais

Aqui se descrevem os requisitos a serem atendidos funcionalmente pelo sistema de uma forma simples, possibilitando a compreensão do comportamento do sistema pela perspectiva do usuário. Devem ser descritos os atores e o diagrama de caso de uso.

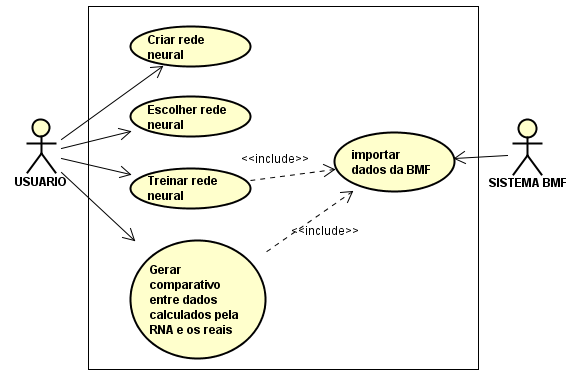
### Atores

**USUARIO:** Representa alguém que utiliza o sistema.

**SISTEMA BMF:** Representa a integração com o WebService da BMF.

### Diagrama de caso de uso

Figura 10. Diagrama de caso de uso



Fonte: Autor (2017)

### Especificação dos Casos de Uso

Nesta seção são apresentadas as especificações dos casos de uso. A especificação dos casos de uso e uma descrição textual que identifica uma interação entre os usuários e o sistema que complementam o caso de uso. (SOMMERVILLE, 2010)

|  |  |
| --- | --- |
| **CSU001 – Importar Dados da BMF** | |
| Sumário: | Importar Dados da BMF |
| Ator Primário: | Sistema BMF |
| Casos de Uso Associados: CSU002 | |
| **Pré-condição:**   1. Conexão com a internet | |
| **Fluxo Principal**   1. O caso de uso inicia quando o sistema necessita das informações da BMF. 2. O sistema envia para a BMF o código da ação e o intervalo de datas. 3. O sistema da BMF retorna a data, o preço de abertura, o máximo, o mínimo, o de fechamento, o volume e o preço de fechamento ajustado por proventos. 4. O sistema extrai as informações da BMF. 5. O caso de uso termina. | |
| **Fluxo de Exceção (3) – Sem conexão com a internet**   1. O sistema informa que o usuário não possui uma conexão com a internet 2. O caso de uso termina. | |
| **Fluxo de Exceção (3) – BMF não possui dados**   1. O sistema informa que a BMF ainda não possui os dados 2. O caso de uso termina. | |
| **Pós-condições:**  a**.** O Sistema armazena os dados. | |
| **Requisitos:** RF001 | |

|  |  |
| --- | --- |
| **CSU002 – Criar Rede Neural** | |
| Sumário: | Criar Rede Neural |
| Ator Primário: | Usuário |
| Casos de Uso Associados: | |
| **Pré-condição:** | |
| **Fluxo Principal**   1. O caso de uso inicia quando o usuário deseja criar uma RNA. 2. O Sistema solicita a quantidade de camadas, de neurônios por camada, função de ativação por camada, se tem ou não bias na camada, o *dropout* da camada, ação que a rede sabe prever, a quantidade de data necessária para gerar o próximo dado e o nome da rede. 3. O usuário informa os campos necessários. 4. O sistema avisa que a RNA foi criada com sucesso. 5. O caso de uso termina. | |
| **Fluxo alternativo (3) – RNA já existe**   1. O sistema informa que já existe uma rede com o nome digitado e pergunta se o usuário quer sobrescrever. 2. O usuário informa se quer sobrescrever. | |
| **Fluxo alternativo (3.1) – Usuário quer sobrescrever**   1. O usuário informa que quer sobrescrever 2. O sistema sobrescreve a RNA existente 3. O caso de uso termina | |
| **Fluxo alternativo (3.2) – Usuário não quer sobrescrever**   1. O usuário informa que não quer sobrescrever a RNA existente 2. O sistema exibe uma mensagem confirmando a opção do usuário, sem alterar a RNA existente. 3. O caso de uso termina | |
| **Fluxo de Exceção (3) – Informações incorretas**   1. O sistema informa que as informações preenchidas estão incorretas 2. O caso de uso termina. | |
| **Pós-condições:**  a**.** Rede Neural criada. | |
| **Requisitos:** RF002 | |

|  |  |
| --- | --- |
| **CSU003 – Escolher Rede Neural** | |
| Sumário: | Alterar RNA |
| Ator Primário: | Usuário |
| Casos de Uso Associados: CSU002 | |
| **Pré-condição:**  a**.** A RNA deverá ter sido criada anteriormente. (CSU002) | |
| **Fluxo Principal**   1. O caso de uso inicia quando o usuário deseja selecionar uma RNA 2. O Sistema solicita o nome da RNA. 3. O Usuário informa o nome da RNA. 4. O sistema pega as informações da RNA 5. O sistema exibe uma mensagem informando que a RNA foi carregada com sucesso. 6. O caso de uso termina | |
| **Fluxo de exceção (4): Rede não encontrada**   1. O sistema informa que não encontrou nenhuma RNA com o nome digitado 2. O caso de uso termina | |
| **Pós-condições:**  a**.** A RNA é carregada em memória**.** | |
| **Requisitos:** RF002. | |

|  |  |
| --- | --- |
| **CSU004 – Treinar Rede Neural** | |
| Sumário: | Treinar Rede |
| Ator Primário: | Usuário |
| Ator Secundário: | Sistema BMF |
| **Caso de Uso associado:** CSU001, CSU003. | |
| **Pré-condição:**   1. O Usuário deve ter selecionado uma RNA. (CSU003) 2. Deve haver conexão com a internet. | |
| **Fluxo Principal**   1. O Sistema solicita as datas ‘de’ e ‘para’, as regras de normalização (limite inferior e superior, e a margem), o algoritmo de aprendizagem, o número máximo de iterações e o erro máximo. 2. O Usuário informa os dados. 3. O Sistema envia puxa as informações do Sistema da BMF (CSU001) 4. O sistema elabora um gráfico da quantidade de iterações versus o erro 5. O sistema exibe uma mensagem avisando que a RNA foi treinada 6. O caso de uso termina. | |
| **Fluxo de Exceção (2) – Datas inválidas**   1. O Sistema informa que as datas escolhidas são inválidas 2. O caso de uso termina. | |
| **Pós-condições:**  a**.** A RNA é treinada. | |
| **Requisitos:** RF003. | |

|  |  |
| --- | --- |
| **CSU005 – Gerar comparativo entre dados calculados pela RNA e os reais** | |
| Sumário: | Gerar relatório |
| Ator Primário: | Usuário |
| Ator Secundário: | Sistema BMF |
| **Caso de Uso associado:** CSU001, CSU003. | |
| **Pré-condição:**   1. O Usuário deve ter selecionado uma RNA. (CSU003) 2. Deve haver conexão com a internet. | |
| **Fluxo Principal**   1. O Sistema solicita as datas ‘de’ e ‘para’ e as informações de normalização (limite inferior e superior, e a margem) 2. O Usuário informa os dados. 3. O Sistema envia puxa as informações do Sistema da BMF. (CSU001) 4. O caso de uso termina. | |
| **Fluxo de Exceção (2) – Datas inválidas**   1. O Sistema informa que as datas escolhidas são inválidas 2. O caso de uso termina. | |
| **Pós-condições:**  a**.** O Sistema gera o gráfico. | |
| **Requisitos:** RF004. | |

# Desenvolvimento do projeto

Este capítulo tem como objetivo analisar, detalhar e propor uma solução geral do sistema, sob o ponto de vista de negócio, de acordo com os requisitos levantados e validados no Capítulo anterior. Além disso, é apresentado o refinamento da proposta de solução geral do sistema, apresentando a solução técnica, incluindo a visão de projeto e implementação, a arquitetura e a tecnologia utilizada.

## Análise

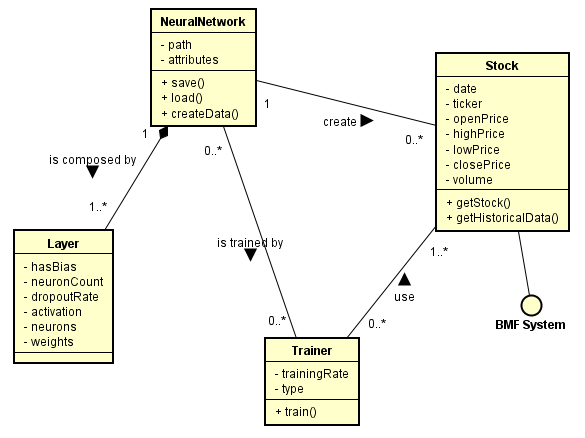
Neste item deve ser apresentado o modelo do domínio, visão de negócio, que representa um primeiro modelo conceitual do diagrama de classes. Posteriormente, esse diagrama deve ser validado, refinado e complementado para compor o diagrama de classes de projeto.

### Diagrama de Classes de Análise (Visão de Negócio)

O diagrama de classes deve possuir todas as classes identificadas do sistema, deve conter os atributos e métodos de cada classe, e os relacionamentos entre elas. Na

Figura 11 é apresentado o diagrama de classes.

Figura 11. Diagrama de Classes de negócio



Fonte: Autor (2017)

## Projeto

Neste item deve ser apresentada a arquitetura de infraestrutura do sistema, demonstrando o tipo de arquitetura que será utilizada a configuração de hardware, de rede, de software, padrões de projeto, componentes específicos e componentes externos a serem utilizados, bem como o dimensionamento mínimo de conexões.

### Arquitetura do Sistema

Para a representação da arquitetura de infraestrutura será apresentado o diagrama de implantação.

Figura 12. Diagrama de implantação



Fonte: Autor (2017)

### Diagrama de Classes de Projeto por Caso de Uso

Nesta sessão, são mostrados todos os diagramas de realização de caso de uso modelados.

Figura 13. Diagrama de classes de projeto – *Create NN*



Fonte: Autor (2017)

Figura 14. Diagrama de sequência - *Create NN*



Fonte: Autor (2017)

Figura 15. Diagrama de classes de projeto - *Choose NN*



Fonte: Autor (2017)

Figura 16. Diagrama de sequência - Choose NN



Fonte: Autor (2017)

Figura 17. Diagrama de classes de projeto *- Import BMF Data*



Fonte: Autor (2017)

Figura 18. Diagrama de sequência *- Import BMF Data*



Fonte: Autor (2017)

Figura 19. Diagrama de classes de projeto - *Comparative Chart*



Fonte: Autor (2017)

Figura 20. Diagrama de sequência - *Comparative Chart*



Fonte: Autor (2017)

Figura 21. Diagrama de classes de projeto *- Train NN*



Fonte: Autor (2017)

Figura 22. Diagrama de sequência - *Train NN*



Fonte: Autor (2017)

# Detalhes de implementação.

Neste capítulo descrevem-se detalhes que merecem destaque sobre a implementação. Para que o sistema ficasse bem flexível para o usuário foram necessários alguns ajustes no algoritmo para que atendesse às exigências e especificações do mercado financeiro.

## Topologia

A rede neural deste trabalho deve ser capaz de prever preços futuros e, para que isso ocorra, ela deve usar os dados que ela mesma gerou. Como foi apresentado no item 2.3.1.1 e no item 2.3.2*,* a rede recorrente e a *feed-forward* com tempo incorporadoé utilizada em casos onde o tempo é um fator determinante. Por causa desses fatores, nesse trabalho foram utilizadas essas topologias de modo dinâmico, com a quantidade de neurônios na primeira camada sendo igual à quantidade de atributos multiplicada pelo *date interval* (que será explicado na seção 6.3)e na última camada somente a quantidade de atributos, sendo um neurônio para cada atributo escolhido pelo usuário.

Na Figura 23 é apresentado um esboço de como a rede utilizada está implementada, utilizando dois atributos e duas camadas ocultas, com 3 neurônios cada e com o *date interval* igual a 2. Cada círculo representa a abstração de um neurônio, e cada seta é uma ligação, ou sinapse. As figuras em vermelho representam o primeiro atributo, por exemplo, o preço de fechamento. As figuras em azul representam o segundo atributo, por exemplo, o preço máximo. As figuras pontilhadas representam as camadas ocultas.

Figura 23. Topologia da RNA utilizada



Fonte: Autor (2017)

Um detalhe interessante mostrado na Figura 23 é que, ao contrário de todas as ligações, onde cada neurônio se liga com todos os neurônios da camada seguinte, a sinapse entre a camada de saída se dá da seguinte forma: um neurônio só se liga com o outro neurônio que represente o mesmo atributo, já que a entrada do neurônio da camada de entrada vermelha é o preço de fechamento (para esse exemplo) e a saída do neurônio vermelho da camada de saída também é o preço de fechamento. O mesmo vale para o par em azul.

Também é possível notar como o fator tempo (*delay*) foi incorporado à rede. Os dados da camada de saída serão utilizados pelos da camada de entrada com um atraso. A quantidade de atraso máxima é igual ao *date interval*, ou seja, a primeira saída será utilizada na próxima iteração, a segunda na segunda, e assim por diante.

A topologia é dinâmica, pois enquanto a rede utiliza dados verdadeiros (externos) ela se comporta como *feed-forward*, como dá para ver no destaque em amarelo da Figura 23. Já quando utilizam os próprios dados, ela se comporta como uma rede recorrente.

Já no início do código da Figura 24 aparece um recurso muito utilizado no projeto, a fim de melhorar a qualidade do código, o Javadoc. Ele é responsável por criar uma documentação completa por meio de meta-dados, chamados de ‘tags’. (ORACLE, 2014) Conforme foi falado nos parágrafos anteriores, a camada de entrada será com a função de ativação *Linear*, com a quantidade de neurônios igual ao de atributos multiplicado pelo intervalo de dados.

Figura 24. Código implementação topologia



Fonte: Autor (2017)

## Função de ativação

Enquanto é possível ter a função de ativação por neurônio, como foi mostrado no item 3.6, no projeto implementado não foi possível fazer por neurônio, mas sim por camada.

Isso se deu já que o framework utilizado (Encog) não tinha suporte à função de ativação por neurônio, por causa do modo que foi implementada a abstração de um neurônio: ao invés de virar uma classe, como representado nos diagramas da sessão anterior, os neurônios eram abstraídos somente como pesos, por questão de desempenho na hora de realizar cálculos.

Para o projeto, nas camadas de entrada e saída foram utilizadas somente a função ***Erro! Fonte de referência não encontrada.****,* para que não haja interferência entre um dia e outro.

Na Figura 25 é apresentado o código utilizado na criação das funções de ativação, utilizado o padrão de projeto *Factory*, com o uso do *Generics* para a parte de transformação do enum para um objeto. Este padrão é responsável por encapsular, simplificar e flexibilizar a criação de objetos. Na Figura 24 o trecho de criação de uma camada utiliza a *factory* desta última figura. Nota-se a facilidade em criar vários tipos de funções de ativações em uma mesma linha de código.

Figura 25. Código *Factory* função de ativação



Fonte: Autor (2017)

Outro ponto de destaque é a utilização de “enum”, que serve para restringir e padronizar as constantes que são abordadas pelo código.

## Date Interval

Uma característica da rede aplicada ao mercado financeiro é poder escolher qual o período de dados passados serão necessários para gerar um dia específico, a fim de diminuir os ruídos em um período de tempo muito pequeno. É necessário que os dados de entrada, para criar o relatório, fossem composto de dados puros (somente dados reais), dados mistos (dados reais e dados gerados) e somente dados gerados.

Na Figura 26 é possível visualizar a explicação do parágrafo anterior. Cada retângulo representa um conjunto de dados de uma ação em um dia (qual é a ação, o dia real e seus preços). Os retângulos em laranja mostram os dados obtidos da BMF, ou seja, os dados reais. Já os retângulos verdes mostram os dados calculados pela rede neural. O contorno em preto representa o conjunto de dado utilizado para gerar o próximo e seu tamanho é dado pelo *date interval* já explicado.

Figura 26. Ajuste *Date Interval*



Fonte: Autor (2017)

Utilizando i = *date interval*, percebemos que para gerar os dados de D+1 são necessários os dados de D+(1 - i) até D+0. Já para D+2, são necessários os dados de D-(2 - i) até D1. Desta forma percebemos um padrão: para calcular os dados de D+n, são necessários os dados de D+(n - i), até D+(n-1) sempre com o tamanho dos dados igual a i. Também podemos notar que são necessários i passos para que a fonte de dados seja formada só por elementos calculados.

Como visto anteriormente, o primeiro laço mostrado na Figura 27 é interagido até o valor de *dateInterval*, dentro dele terá um laço para adicionar os dados reais, outro para adicionar os dados calculados e um laço para calcular os valores de cada atributo e adicionar isso ao vetor contendo todos os dados.

Figura 27. Código *mixed data*



Fonte: Autor (2017)

Na Figura 28 é possível ver o trecho do código que trata os dados calculados pela rede. Nota-se na primeira parte a inserção dos valores padrões, isso porque não existe ponteiro explícito em Java e, caso não criasse um “new Data”, o vetor seria completado com o mesmo dado, o que será explicado no próximo parágrafo.

Figura 28. Código *calculated data*



Fonte: Autor (2017)

Além disso, nota-se a criação do *input* com base nos dados calculados e guardados no vetor *datas*, e depois a utilização desses dados para criar os valores do próximo dia. Ao final desse processo, os valores são retornados ao normal, por meio do método “*denormalizeDatas*”.

## Normalizador de dados

Para que uma rede entenda os dados de uma ação, é necessário que esses dados estejam em um formato aceitável. Esse assunto já foi abordado na sessão 2.5. Na Figura 28, ao final do código é chamado um normalizador (*normalizer*) para realizar essa tradução entre a “linguagem” da rede e da ação.

Na Figura 29 é mostrado como é feito para normalizar toda a série histórica. Antes de tudo, os valores máximos e mínimos são atualizados, depois para cada dado da série, e para cada atributo, normaliza o valor utilizando a Equação 3 já abordada anteriormente. Nota-se, também, uma nova forma de fazer um loop no Java 8: o *forEach*, que será explicado posteriormente.

Figura 29. Código *normalize data*



Fonte: Autor (2017)

Figura 30. Código *get max and min*



Fonte: Autor (2017)

Na Figura 31 é mostrado como foi implementado a Equação 3. Quando a margem é igual a 1 (100%), o algoritmo faz um ajuste para que o valor fique igual a 1,001 (100,1%), pois se isso não ocorrer, o valor mínimo do ajuste, ao multiplicar o valor mínimo da série por (1 – *margin*), será 0 e poluirá o range verdadeiro.

Figura 31. Código *getNormalizedValue*



Fonte: Autor (2017)

## Recursos do Java 8

Nesta sessão será apresentado alguns recursos do Java 8 que foram utilizados para o desenvolvimento deste trabalho.

### Lambda

O lambda é uma sintaxe mais fácil de utilizar em comparação à *inner class,* que era utilizada nas versões antigas. Ela abstrai o conceito de criar uma classe e depois criar um método, onde a ideia é somente ter um comportamento. Um exemplo muito utilizado é no comando dos botões usando o Swing do Java.

Na Figura 32 é possível ver o comportamento do clique do botão, neste caso o que adiciona uma linha na tabela de camadas da rede neural, implementado de duas formas diferentes.

Figura 32. Diferença entre *inner class* e *lambda*



Fonte: Autor (2017)

No primeiro caso é criada uma *inner class* só para realizar duas operações simples. Nota-se a facilidade de entendimento que o lambda proporciona, e a diminuição de instruções para fazer a mesma coisa.

### Method Reference

O *Method Reference* é outro recurso vindo com o Java 8, que auxilia a escrever códigos mais enxutos. Ele vem substituir as expressões lambdas nos casos em que só se chama um método, sem fazer qualquer outro ajuste.

Na Figura 33 contém um trecho que mostra na tela o nome de cada santo. Ambos os códigos fazem a mesma coisa, mas no segundo caso é muito mais fácil entender o que está acontecendo.

Figura 33. Código *lambda* e *method* *reference*



Fonte: Autor (2017)

Se no caso anterior a sintaxe não pareceu tão convidativa, será apresentado um caso que a diferença é mais clara. Em uma parte do código deste trabalho, foi necessário criar uma ordenação personalizada, por se tratar de um objeto e não um tipo primitivo. Para isso é preciso dizer à função sort como se ordena esse tipo específico. Na Figura 35 é mostrado o método que especifica como se compara um dado, comparando o atributo volume dos dois parâmetros.

Figura 34. Código *compareByVolume*



Fonte: Autor (2017)

Depois de especificado como se ordena um dado, é preciso agora ordená-lo. Existem algumas maneiras de fazer isso, e na Figura 35 é mostrado elas. Nota-se a facilidade de compreensão que o uso do *method reference* proporciona.

Figura 35. Código de ordenação



Fonte: Autor (2017)

### Stream

*Stream* é uma nova abstração para processar dados de uma forma mais amigável. Uma definição rápida de *stream* é: uma sequencia de elementos de um recurso que suporta operações agregadas (ORACLE, 2014). Existem dois tipos de *Stream*: linear e em paralelo, que se diferem pelo tipo de processamento.

#### Linear Stream

Se for necessário, por exemplo, calcular a média dos preços de fechamento da Petrobrás, mas somente nas datas que tiveram um volume expressivo, digamos acima 60.000.000 ações negociadas, entre 12 de janeiro e 24 de janeiro de 2017, como fazer? Primeiro é necessário filtrar todos os dias que tiveram volume maior que 60 milhões, depois extrair somente o preço de fechamento e, por fim, fazer uma média desses valores.

Como se pode perceber, apesar de ser uma tarefa simples, renderia várias linhas de código, mas como é possível fazer todas as operações usando o *Stream*?

Primeiro passo é extrair os dados da Petrobrás entre esse período. Na Figura 36 é mostrado o código de como extrair esses dados utilizando partes do código deste trabalho para facilitar essa extração, e depois mostrando na tela os resultados obtidos.

Figura 36. Código exemplo *stream* – pegando dados



Fonte: Autor (2017)

Para que a última linha da figura anterior funcione, é necessário fazer uma sobrescrita do método “toString”, que é mostrado na Figura 38. A ideia é bem simples, mostrar todos os valores, sem se preocupar se existe ou não.

Figura 37. Exemplo código *stream* - sobrescrita *toString*



Fonte: Autor (2017)

Na Figura 38 é apresentado o resultado do código mostrado na Figura 36. É possível notar que somente os atributos declarados anteriormente estão sendo extraídos: *HighPrice, ClosePrice* e Volume. Apesar de no exemplo não precisar do preço máximo, foi atribuído a este *HistoricalData* esse atributo também, para futuramente mostrar uma outra funcionalidade do *Stream*.

Figura 38. Saída do código de exemplo da *stream*



Fonte: Autor (2017)

Além disso, foi proposto ações com volume maior do que 60.000.000 ações para facilitar, pois desta lista, somente três datas superaram isso. Desta forma, calculando manualmente a média temos que a média dos preços de fechamento deverá ser (13,59 + 13,16 + 13,66 / 3 = 13,47).

Agora que os dados foram apresentados, é necessário fazer o filtro pelo volume, pegar somente os preços de fechamento e ainda fazer uma média sobre isso.

Com a Figura 39 percebe-se a facilidade lógica e semântica que se ganha com o uso do *Stream*. É possível notar o padrão de projeto *Builder* e o *Fluent Interface*, responsáveis por essa leitura mais “fluente” e um código muito menos verboso.

Figura 39. Código exemplo *stream*



Fonte: Autor (2017)

O primeiro método chamado é o *stream()*, que retorna um *stream* contendo os elementos da coleção passada, no caso um *ArrayList* de *Data*.

O segundo método (*.filter*) , como o próprio nome sugere, faz um filtro, selecionando somente os valores que atendam a condição. Por baixo dos panos, é feito um *forEach*, como foi explicado anteriormente, percorrendo linearmente toda a *stream*, procurando a parte da direita da instrução que, necessariamente, deve retornar um *boolean*, mas isso tudo foi abstraído.

O terceiro método (*.mapToDouble*) serve para transformar esse conjunto de dados que foi filtrado em um *DoubleStream*, que é uma sequencia do tipo primitivo *double*, com alguns recursos extras, como suporte a operações agregadas sequenciais e paralelas, que será explicado posteriormente.

Após isso, no quarto esse método, é feita a operação de média com os valores encontrados. Esse método retorna um *OptionalDouble*, que é uma abstração que basicamente diz se foi encontrado um valor ou não, pois é possível não encontrar nenhum valor no *.filter* acima e, com isso, é impossível fazer uma média com a divisão por zero.

Por fim, é chamado o método *getAsDouble,* que tenta fazer a conversão de um *OptionalDouble* para um Double, podendo lançar uma *exception* quando o resultado é vazio, mas é possível verificar se existe com o método *.isPresent*.

Na Figura 40 é ilustrado a sequência dos passos realizados, explicada nos parágrafos anteriores de uma forma mais simples de entender.

Figura 40. Ilustração código stream



Fonte: Autor (2017)

#### *Parallel Stream*

Como foi dito anteriormente, uma das maiores sacadas da nova atualização é a comodidade em utilizar o paralelismo. Uma das formas encontradas foi utilizar a nova abstração do *Stream* e implementar uma *Parallel Stream*, que é uma diferenciação por implementar conceitos de paralelismo por baixo dos panos, como concorrência de *threads*, *threadPool*, *executorService*, sincronismo, entre outros.

Desta forma, o paralelismo é implementado de forma implícita, isto é, quando o programador não explicita o paralelismo por meio de comandos. Dada a explicação, como ficaria a implementação de um Parallel Stream, será que muita coisa muda? É muito mais complexo?

De fato, a resposta das duas perguntas acima é não. Como é mostrado na Figura 41, só foi necessária a troca do primeiro método, ou seja, ao invés de chamar um *stream* sequencial, chama-se um *parallel stream*. No caso exemplificado não é tão fácil notar a diferença, visto que o resultado será o mesmo e os dados a serem processados são bem pequenos, então não terá uma melhora tão grande no desempenho.

Figura 41. Código *stream* com paralelismo



Fonte: Autor (2017)

Para mostrar as diferenças citadas no parágrafo anterior, a Figura 43 mostra um código simples de popular uma lista com valores de 1 a 10, e em seguida lista-los, mas de duas formas: a primeira sequencialmente e a segunda com o paralelismo..

Figura 42. Código exemplo paralelismo



Fonte: Autor (2017)

Na Figura 43 é mostrado a saída dos três laços. É possível perceber que no primeiro caso, como era de se esperar, os números são exibidos de forma sequencial, e nos dois outros não. Além disso, nos dois últimos casos a lista não é exibida na mesma ordem, mesmo sendo chamada da mesma forma.

Figura 43. Saída código exemplo paralelismo



Fonte: Autor (2017)

Na Figura 44 é apresentado o mesmo código da Figura 30, mas com o uso do *Parallel Stream*. Para que o *Stream* possa ser usado em mais de uma operação final (será visto no próximo item), é necessário criar um *SummaryStatistics*, que é um objeto que já contém os valores máximo, mínimo, média e soma de um *Stream*. Em seu construtor, ele faz uma varredura linear por toda a coleção e encontra esses valores e armazena em seus atributos, sendo muito útil quando se quer pegar mais de uma dessas informações, ao invés de ter que refazer todos os passos e aumentar, deste modo, a complexidade do código.

Figura 44. Código updateMaxMin



Fonte: Autor (2017)

Na primeira linha é convertido a lista dos dados em um *parallelStream* chamado *stream*. Depois disso, vem um laço para cada atributo, que o usuário selecionou para a rede cobrir, converte para um *DoubleStream* e depois para *SummaryStatistics*. Com isso, se pega o máximo e mínimo de cada atributo e armazena em uma coleção com o máximo de todos os atributos (exceto o volume), e em outra coleção com o mínimo de cada atributo, e, por fim, retorna o máximo dos máximos, e o mínimo dos mínimos. O volume é diferente por se tratar de outro tipo de dado, ele não é, que nem os outros, referente ao preço de uma ação, portanto sua ‘base’ será completamente diferente.

#### Tipos de operações do *Stream*

O *Stream* possui dois tipos de operações: Intermediárias e Terminais. As operações intermediárias são as que são implementadas no padrão *Builder* e retorna uma classe que implemente uma *Stream*, ou seja, pode-se encadear quantas operações do tipo intermediárias quiser, desde que faça sentido para a aplicação. Exemplos dessas operações são: filter(), mapToInt(), mapToDouble().

O outro tipo, terminal, são as operações que retornam um valor ou um objeto. Após ser invocada, não é possível realizar outras operações, nem intermediárias nem finais. Exemplos de operações desse tipo são o forEach(), sum(), min(), max(), findFirst().

# Resultados Obtidos

Foi obtido como resultado deste trabalho, um software para criação de RNA com o foco na previsão de dados relevantes para o mercado financeiro. Neste capítulo é apresentado telas do sistema implementado.

## Passos do sistema

Nas seções deste capítulo são apresentadas as sequências de passos que podem ser tomados pelo usuário do sistema.

### Criar RNA

Ao abrir o sistema, a primeira tela é a de escolha da RNA. É possível perceber que na Figura 45 todos os campos estão vazios, exceto o *Date Interval*, que é uma sugestão. No rodapé da imagem, é apresentado qual é a RNA escolhida no momento, como o programa acabou de ser iniciado ainda não tem nenhuma RNA escolhida. Se o ponteiro do mouse ficar em cima da aba *NeuralNetwork* por alguns segundos, aparecerá uma dica do que se trata a aba.

Figura 45. Interface 1 - *initial*



Fonte: Autor (2017)

Na Figura 47 é possível notar no centro da tela o erro que aparece se for clicado no botão Create. Como não tem o nome, nem o Stock, nem os atributos, um erro é mostrado para que o usuário complete os campos com dados válidos.

Figura 46. Interface 1 - *error*



Fonte: Autor (2017)

Se o usuário colocar dados válidos e clicar no botão *Create*, é apresentada uma mensagem dizendo que a rede foi criada com sucesso, como é mostrado na Figura 47. Mesmo sem adicionar nenhuma camada oculta é possível criar a rede e, no caso, só terá duas camadas verdadeiras: a inicial (ou de entrada) e a final (ou de saída).

Figura 47. Interface 1 – *success*



Fonte: Autor (2017)

Se o usuário clicar no botão “+”, será adicionado uma linha na tabela das camadas ocultas, com uma sugestão de camada: três neurônios, contendo bias, função de ativação ElliottSymmetric e com 0,3 de dropOutRate, como é mostrado na Figura 49.

Figura 48. Interface 1 - *add hidden layer*



Fonte: Autor (2017)

Na Figura 49 é possível ver o que acontece ao clicar na função de ativação, são mostradas todas as funções de ativação que o sistema suporta como foi explicado anteriormente no capítulo 2.2. Se clicar no botão “X”, é apagado a linha selecionada, e se não tiver nenhuma selecionada, apaga a última linha.

Figura 49. Interface 1 - *activation functions*



Fonte: Autor (2017)

Se o botão Create for clicado novamente, é apresentada uma mensagem alertando ao usuário que já existe uma rede criada com o mesmo nome, e se ele deseja sobrescrever a rede existente, como é mostrado na Figura 51. É importante notar que, apesar de um dos requisitos não funcionais especificarem que todo o texto deverá ser em inglês, neste caso os botões são apresentados no idioma do sistema operacional em que o usuário está executando o programa.

Figura 50. Interface 1 – *overwrite*



Fonte: Autor (2017)

Caso o usuário clique no botão de negação, “Não” no caso, a mensagem deixará de ser exibida e aparecerá outra mensagem, como é mostrado na Figura 51.

Figura 51. Interface 1 – *don’t overwrite*



Fonte: Autor (2017)

Caso o usuário clique no de afirmação, “Sim” no caso, é feita uma sobrescrita na RNA já criada, mesmo que nenhum detalhe for mudado. Isso acontece, pois cada vez que uma rede é criada (neste caso recriada), os neurônios são criados com pesos aleatórios. Essa parte é importante no caso de a rede sofrer algum tipo de *overtraining* durante o treinamento e ter sua capacidade de predição comprometida. Se acontecer, basta manter a topologia criada e forçar uma sobrescrita da rede, para alterar os pesos dos neurônios.

Também é possível notar que após a rede criada, por padrão, o programa já a seleciona e no rodapé da tela é mostrado ao usuário que a rede selecionada é essa já criada.

### Escolher RNA

Nesta etapa existem somente duas coisas que podem acontecer, ou a rede não existe ou existe.

Na Figura 52 é apresentado o primeiro caso: se não existe. Quando o programa tenta procurar por uma rede com um nome de uma rede inexistente, apresenta uma mensagem alertando que não pode ser carregada, e que o arquivo não existe. Repare que foram desmarcados todos os atributos e apagado qual é o *Stock* da rede, para que se note o preenchimento dos campos corretos

.

Figura 52. Interface 1 *- NN don't exists*



Fonte: Autor (2017)

A Figura 54 mostra o que acontece quando o usuário tenta escolher uma rede existente. Além de aparecer a mensagem informando que a rede foi carregada, os dados dessa rede são preenchidos na tela, para que o processo de identificação das características da rede seja facilitado.

Figura 53. Interface 1 - *NN loaded*



Fonte: Autor (2017)

### Treinar RNA

Passado o passo de criar e escolher uma RNA, o usuário pode querer treinar sua rede. Para isso basta clicar na aba “*Train*”.

Na Figura 54 é mostrado o que aparece ao clicar pela primeira vez na aba *Train* durante a execução do programa. Nesta etapa, são vários os erros que podem acontecer, e eles serão apresentados. São apresentados alguns valores de sugestão, para que um usuário sem conhecimento em redes neurais possa interagir com o sistema e perceber como uma RNA se comporta. Se o ponteiro do mouse ficar em cima da aba *Train* por alguns segundos, aparecerá uma dica do que se trata a aba.

Figura 54. Interface 2 – initial



Fonte: Autor (2017)

Se clicar no botão Train sem ter escolhido as datas, é apresentada uma mensagem solicitando ao usuário escolher as datas, como mostra a Figura 56.

Figura 55. Interface 2 - *null dates*



Fonte: Autor (2017)

Caso o usuário tenha acessado a essa aba já no início do programa, sem ter escolhido uma RNA ainda, o erro apresentado é o mostrado pela Figura 56, solicitando ao usuário escolher uma RNA antes de prosseguir.

Figura 56. Interface 2 - *null NN*



Fonte: Autor (2017)

A Figura 58 mostra todas as opções de algoritmos de aprendizagem disponível no programa. Esses algoritmos já foram tratados em outra parte deste trabalho, quando foi apresentado sobre aprendizagem e seus algoritmos, presente na sessão 2.4.

Figura 57. Interface 2 - *Learning Rules*



Fonte: Autor (2017)

Quando o botão “...”, presente ao lado de cada data, é clicado, é apresentado ao usuário uma interface intuitiva de um calendário, conforme é mostrado na Figura 58. Nessa interface, é possível navegar com facilidade entre os meses, os anos, e as datas. Nota-se que os textos são apresentados conforme o idioma do sistema operacional em que o programa está sendo executado

Figura 58. Interface 2 – *dates*



Fonte: Autor (2017)

Caso o usuário se confunda e tente inserir a data de partida como depois da data final, é apresentada uma mensagem alertando-o disso, como mostra a Figura 63. Não foram invertidas automaticamente as datas, contornando o erro, portanto, pois se o usuário realmente queria essa data, mas de um ano anterior, e apenas se confundiu na hora de selecionar o ano, a rede não treine com dados que não desejados.

Figura 59. Interface 2 - *from after to*



Fonte: Autor (2017)

Na Figura 60 é apresentado o erro que aparece quando o usuário tenta selecionar datas que ainda não existem dados disponíveis, no caso foi selecionado em 2017 dados de negociações feitas em 2018.

Figura 60. Interface 2 - *don't have data form selected date*



Fonte: Autor (2017)

Se o usuário digitar todos os valores corretos e clicar no botão Train, a rede é treinada e aparece uma mensagem informando isso, como mostra a Figura 62. Além disso, é apresentado um gráfico que mostra como foi a evolução do erro desta RNA para o intervalo de data por cada interação (epoch).

Figura 61. Interface 2 – *trained*



Fonte: Autor (2017)

É possível notar, como era o esperado, que a rede tende a diminuir seu erro a cada iteração, pois é quando o algoritmo de aprendizagem age e vai corrigindo aos poucos.

### Gerar Relatório

Depois de selecionado a RNA, é possível gerar um gráfico comparando os dados calculados pela rede e os reais, mesmo sem ter treinado essa rede antes.

Na Figura 62 é mostrado a tela em questão, quando aberta assim que o aplicativo inicie, ou seja, sem ter selecionado uma RNA. Se o ponteiro do mouse ficar em cima da aba *Report* por alguns segundos, aparecerá uma dica do que se trata a aba, assim como as outras telas.

Figura 62. Interface 3 – *initial*



Fonte: Autor (2017)

Assim como no caso do treinamento, se o usuário tentar criar um relatório sem ter selecionado a RNA, será apresentado o erro mostrado na Figura 64, alertando-o que é necessário escolher uma RNA antes.

Figura 63. Interface 3 - *null NN*



Fonte: Autor (2017)

Também como no caso do treinamento, todos os erros que envolvem as datas aparecem aqui também, inclusive com a mesma mensagem e mesmo título. Sendo assim, será apresentado somente o relatório com todos os passos válidos, sem qualquer erro.

Na Figura 64 é mostrado o que acontece quando os dados estão corretos e o botão *Create Report* recebe um clique. O sistema gera o gráfico comparativo entre os dados reais e os dados criados pela rede neural.

Figura 64. Interface 3 - *report created*



Fonte: Autor (2017)

A Figura 66 mostra o mesmo relatório, mas com a rede configurada para mais de um atributo. Nota-se que é criada uma linha para cada atributo real, e uma linha para cada atributo calculado pela Rede, para que seja possível comparar atributo a atributo.

Figura 65. Interface 3 - *report with 2 attributes*



Fonte: Autor (2017)

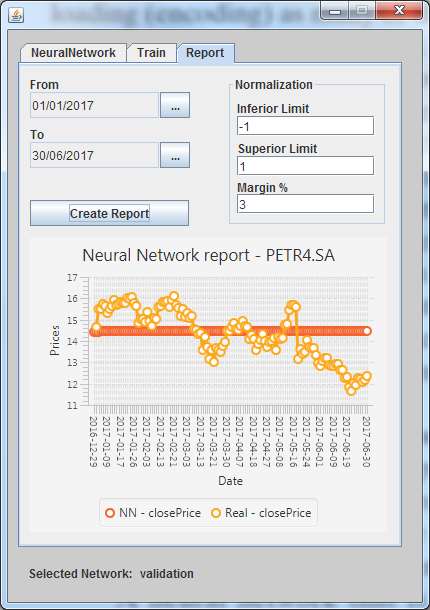
## Estresse de uma RNA

Nesta seção são apresentadas algumas telas do sistema mostrando dois conceitos de aprendizagem de máquina e depois uma sequencia de telas tentando criar uma rede capaz de prever o preço das ações de Petrobras (PETR4) no mês de maio de 2017.

### Baixo treinamento (under fitting)

O baixo treinamento ocorre quando uma rede não consegue absorver o conteúdo a ser aprendido. É possível notar este comportamento na Figura 66. Foi necessário criar uma rede e já testá-la, sem passar pela etapa de treinamento.

Figura 66. Under fitting



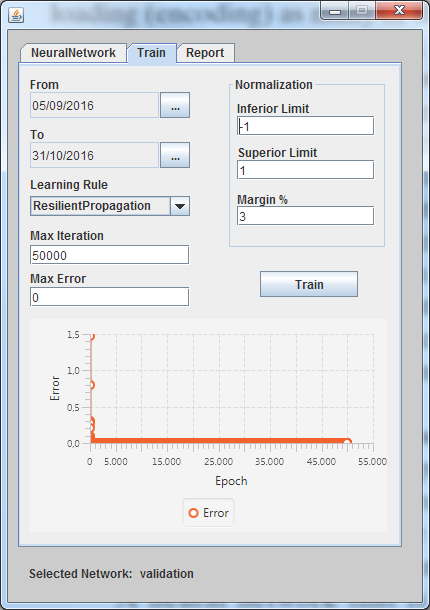
Fonte: Autor (2017)

É possível notar que a rede previu (linha reta) um preço constante, sem qualquer alteração, somente propagando o dado de entrada por todos os dias.

### Alto treinamento (over fitting)

O alto treinamento ocorre quando uma rede passa a memorizar o conteúdo ao invés de generalizar os dados. Ela perde a capacidade de generalizar conteúdos e padrões similares. Na Figura 67 nota-se a configuração utilizada para treinamento: datas curtas (anteriores ao dia de previsão), erro sendo 0 (quase impossível) e quantidade de iteração como 50.000.

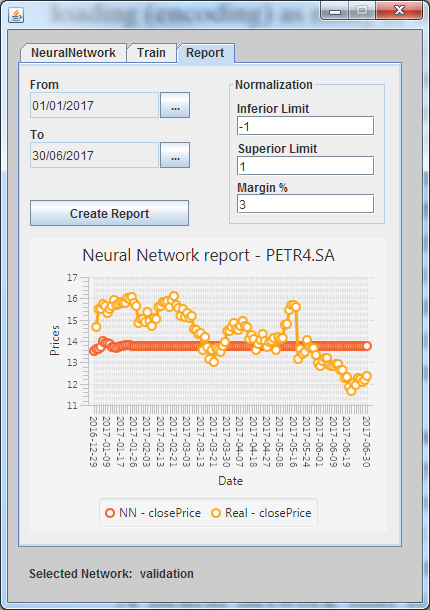
Figura 67. Over fitting - Trainning



Fonte: Autor (2017)

Para fins de comparação, foram utilizados os mesmos dados da Figura 66. É possível notar que na Figura 68 no primeiro mês a rede teve um comportamento provavelmente similar ao período que foi treinada (inclusive a movimentação se permaneceu por somente 1 mês, assim como o período de treinamento) e depois disso já não sabia o que fazer e apresentou um comportamento constante.

Figura 68. Over fitting - Report



Fonte: Autor (2017)

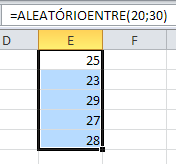
### Rede eficiente para prever PETR4 em maio de 2017

O mês de maio de 2017 foi escolhido, pois no meio do mês, no dia 17, teve a delação premiada de um dos donos da JBS contra o presidente. Isso contribuiu para que os investidores ficassem preocupados com o risco de uma instabilidade política e venderam suas ações, e causando uma queda bruta no preço das ações. A ação escolhida foi a PETR4, pois é uma das ações mais líquidas negociadas na bolsa do Brasil, e tende a refletir bastante o ambiente macroeconômico.

Para o treinamento será utilizado desde Agosto de 2012 até Abril de 2017, pois além desse período terem acontecido várias quedas bruscas no preço das ações, também simula os dados que um investidor no começo de maio teria como fonte. Além disso, serão necessárias 2000 iterações, para tentar atingir um erro máximo de 0, utilizando o algoritmo *Resilient Propagation* para a aprendizagem.

Para a criação da primeira topologia, foi utilizada uma fórmula de números aleatórios entre 20 e 30, que é minha idade, como é apresentado na Figura 69. A quantidade de neurônios foi pelo número de letras que o símbolo PETR4 possui, ou seja, cinco.

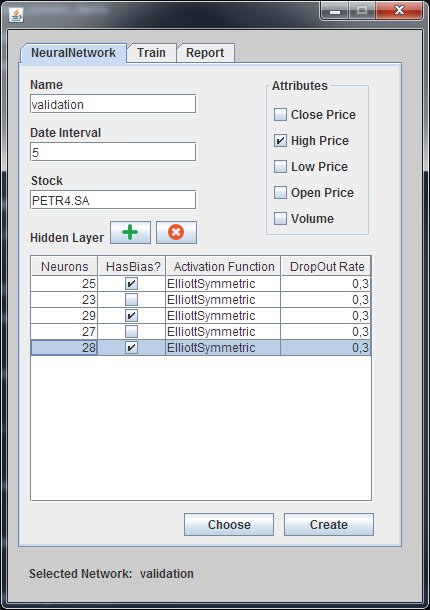
Figura 69. Topologia aleatória



Fonte: Autor (2017)

Para simplificar o problema com muitas variáveis, todas as funções de ativação serão a ElliotSymmetric, com o DropoutRate de 0,3. Para ver se a rede teria BIAS, foi adotada uma forma simples de que as camadas ímpares teriam bias e as pares não. Na Figura 70 é mostrado como ficou a configuração.

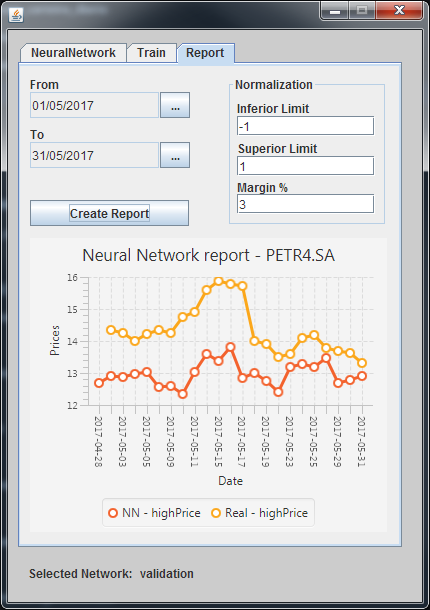
Figura 70. Configuração RNA 1



Fonte: Autor (2017)

A RNA criada foi capaz de prever uma alta, seguida de uma queda praticamente no mesmo período que aconteceu de verdade. Apesar de não ter acertado a amplitude, pode acertar a tendência, o que já se mostra bastante eficaz, como é mostrado na Figura 71.

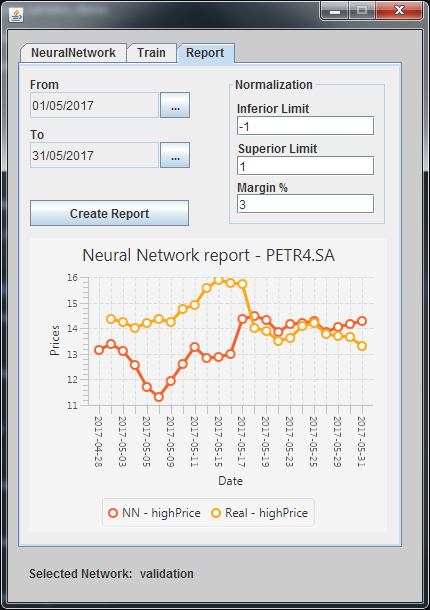
Figura 71. Resultado RNA 1

. 

Fonte: Autor (2017)

Colocando todos os valores dos dropout para zero, a rede passa a ter um comportamento oposto do que aconteceu de verdade, respeitando as demais condições. É possível perceber pela Figura 72 que no momento que o preço da ação subiu, a rede previu que cairia, e o inverso também se mostrou verdadeiro.

Figura 72. Resultado RNA 2

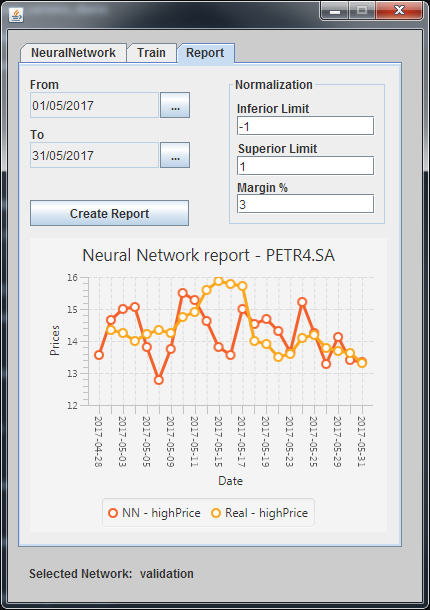


Fonte: Autor (2017)

Para a terceira configuração, todos os *Dropout Rates* ficaram em um. Como resultado, a rede se mostrou pouco eficaz, prevendo preços com bastante distorção, inclusive entre eles. Com a

Figura 73 fica claro perceber isso.

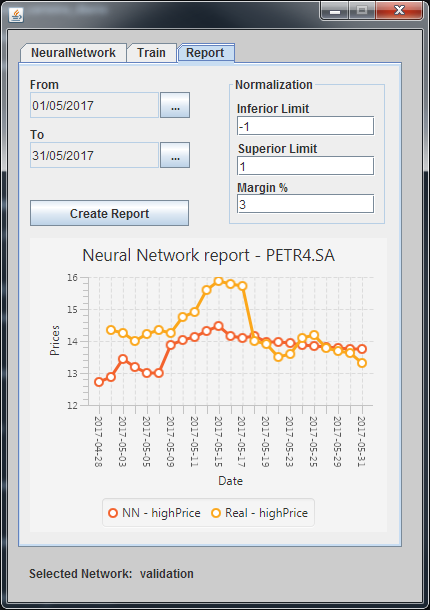
Figura 73. Resultado RNA 3



Fonte: Autor (2017)

Por outro lado, mesmo treinando a série de dados por 15000 vezes, (7.5 vezes a mais que as anteriores), ela não sofreu o efeito do *over fitting*, mas também não mostrou resultados tão satisfatórios, como nota-se na Figura 74.

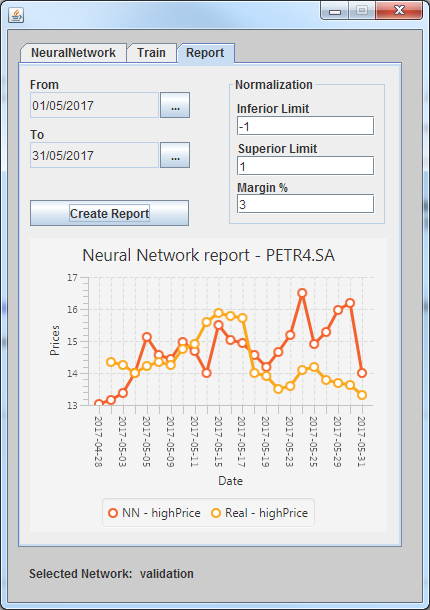
Figura 74. Resultado RNA 4



Fonte: Autor (2017)

Aumentar o número de neurônios por camada para mais de 80, ou seja, aproximadamente quatro vezes mais, fez com que o treinamento demorasse cerca de 200 vezes mais. Isso mostra que a complexidade do treinamento é exponencial, reforçando o que (SCHMIDHUBER, 1992) diz. Mesmo com tudo isso, a eficiência da rede não se comprovou na mesma proporção, até mesmo piorou, como é possível notar na Figura 75.

Figura 75. Resultado RNA 5



Fonte: Autor (2017)

# Considerações Finais

O objetivo deste trabalho foi criar um software capaz de criar uma RNA com o foco no mercado de ações, abrangendo desde a criação da topologia da RNA, passando pela extração de dados das bolsas de valores, até a criação de um relatório que mostre a eficácia da IA à predição.

O primeiro passo do trabalho foi identificar, por meio de estudos e pesquisa, os principais conceitos sobre IA, topologias diversas e as capacidades e limitações de uma RNA. Paralelamente a isso, também foi necessário a pesquisa do mercado financeiro, desde conceitos fundamentais de economia e matemática financeira, até estratégias de investimento.

O segundo passo foi necessário identificar quais conceitos de uma RNA poderiam ser utilizados para usuários do mercado financeiro, bem como descobrir como isso é feito atualmente. Foi constatado que em nenhum dos softwares disponíveis tem abrange esses dois temas. Com isso, elaborar um comparativo entre os softwares disponíveis mais importantes, fez sentido tanto para elencar quais são os melhores, quanto para descobrir o que seria desejável um sistema nesta área. Para validar as configurações utilizadas foi conversado com analistas de uma gestora de fundos de investimentos sobre quais campos seriam necessários, qual seria uma visualização importante ter, entre outros detalhes.

Após a validação dos especialistas, o próximo foi encontrar um framework em Java que auxilie na criação da RNA, pois era onde o conhecimento deveria ser mais profundo. A linguagem foi Java para poder usar e testar os novos recursos do Java 8. O Framework utilizado foi o Encog por vários motivos: código aberto, suporte a multithread, uso de CPU + GPU, muito bem documentado e com alguns exemplos.

Com o programa já programado e funcionando, a capacidade em prever preços se mostrou bastante eficaz, mas a maior limitação é a configuração da rede, devido a complexidade. Foi importante ver na prática os conceitos como a RNA tende a melhorar suas predições com o treinamento e como o treinamento em excesso compromete a capacidade da rede.

## Trabalhos futuros

Em relação a trabalhos futuros, este trabalho tem algumas das melhorias que poderiam ser feitas:

### Gráfico

Foi utilizado o Java FX para fazer os gráficos, mas se provou um framework muito verboso e que para aplicações simples e básicas são necessárias varias linhas de código e muitas configurações.

Utilizar alguma biblioteca mais potente, ou de alguma forma gerar uma página da web e utilizar bibliotecas em JS que fazem esse trabalho como, por exemplo, o AmCharts.

Outra melhoria interessante é deixar o volume como um eixo secundário no gráfico, para não poluir.

### Comparativo entre outras RNA

Uma maneira para se comparar desempenho de várias RNAs que fazem a mesma coisa seria uma implementação muito importante para o trabalho, para testar mais rapidamente como cada ajuste interfere no comportamento da rede.

Além disso, um sistema de elencar as redes mais eficazes por algum tipo classificação ou de ordenação pode ser uma forma de encontrar as redes mais adequadas para cada tipo de arquivo.

### Base de dados

Atualmente o sistema de base de dados é voltado somente para o mercado financeiro e há uma dependência com o YahooFinance e se, por algum motivo, eles descontinuarem o serviço, a aplicação precisará de alguns ajustes. Ter mais de uma fonte de dados pode ser um recurso bastante útil como, por exemplo, o Google Finance.

Outros tipos de dados temporais também podem se mostrar úteis, como dados mais gerais sobre países, meteorologia, ou qualquer outro tipo. Para os dados de países, principalmente dos estados unidos, recomendo o FRED (<https://fred.stlouisfed.org/>). Por lá é possível obter desde evolução do desemprego, até evolução da dívida com escolaridade em universidades públicas.

### Outros relatórios

Além do comparativo entre o que a rede previu e o que foi realizado, é possível ter outras visões. Uma visão bem interessante seria gerenciar uma carteira fictícia, no qual se a rede acha que o preço vai subir em D+1, poderia ter uma regra de comprar um pouco em D+0, e se realmente subir em D+1 vender essas ações. Com isso é possível ter uma “cota”, como o que acontece com fundos de investimentos para essa rede. E a cada dia, ela vai atualizando sua base de aprendizagem com novos dados.

Outra visão, mais fácil de fazer, é avaliar o acerto das tendências. Se a rede prevê uma tendência de alta de 5%, por exemplo, e sobe só 3%, apesar de ter errado o valor, acertou a tendência e isso já é muito valorizado no mercado financeiro.

### Layout

Como o foco do trabalho foi nos algoritmos e na construção da parte “por baixo dos panos”, a parte visual e de interface poderiam ser melhoradas. Apesar de o sistema possuir poucas iterações com o usuário, telas simples e auto explicáveis, o esquema de cores, bordas e posicionamento não são muito agradáveis.

Outra funcionalidade que poderia ter é buscar as redes por meio de uma interface gráfica, e não somente pelo nome da rede. Com isso, já seria possível armazenar a rede em qualquer lugar, e usá-la sem maiores problemas, inclusive se a rede estivesse na “nuvem”, daria para programar alguma forma de consumir essa rede pela internet.

# Bibliografia

ADELAIDE, U. O. **Neural Network Excel Add-in**, 2010.

BARCHART. Stocks - Performance. **Barchart**, 2016. Disponivel em: <http://www.barchart.com/stocks/performance/5day.php>. Acesso em: 10 Outubro 2016.

BROWNLEE, J. How to Normalize and Standardize Your Machine Learning Data in Weka. **Machine Learning Mastery**, 2016. Disponivel em: <http://machinelearningmastery.com/normalize-standardize-machine-learning-data-weka/>. Acesso em: 2 November 2016.

CHABOUD, A. P. et al. Rise of the machines: Algorithmic trading in the foreign exchange market. **The Journal of Finance**, v. 69, n. 5, p. 2045-2084, 2014.

CHAUVIN, Y.; RUMELHART, D. E. **Backpropagation:** theory, architectures, and applications. New Jersey: Psychology Press, 1995.

CHERMAN, E. A.; TSOUMAKAS, G.; MONARD, M. C. Active Learning Algorithms for Multi-label Data. **Artificial Intelligence Applications and Innovations: 12th IFIP WG 12.5 International Conference and Workshops**, Greece, 16-18 September 2016. 267-279.

CONNOR, J. T.; MARTIN, D.; ATLAS, L. E. Recurrent Neural Networks and Robust Time Series Prediction. **IEEE TRANSACTIONS ON NEURAL NETWORKS,** v. 5, n. 2, p. 240-254, Março 1994.

DEMUTH, H.; BEALE, M. **Neural Network Toolbox:** for use with Matlab. 4. ed. Natick: MathWorks, 2003. Disponivel em: <http://www.dsc.ufcg.edu.br/~hmg/disciplinas/graduacao/rn-2016.1/RN-5b%20-%20Redes%20Neurais%20no%20Matlab.pdf>.

DO CARMO ROQUE, R.; DE MELLO, F. L. Estudo sobre a empregabilidade da previsão do índice BOVESPA usando Redes Neurais Artificiais, 2009.

FAHLMAN, S. E. An empirical study of learning speed in back-propagation networks. **Carnegie Mellon University: Research Showcase**, 1988.

FORTUNA, E. **Mercado Financeiro:** produtos e serviços. 11ª. ed. Rio de Janeiro: Qualitymark Editora Ltda, 2008.

FRANZ AG, O. L. **Neuro XL**, 1998.

HAYKIN, S. **FEEDFORWARD NEURAL NETWORKS:** AN INTRODUCTION. [S.l.]: [s.n.], 1998.

HAYKIN, S. **Neural networks and learning machines**. Third Edition. ed. Upper Saddle River, NJ, USA: Pearson, 2009.

HEATON RESERACH. Javadoc Encog 3.3, 2014. Disponivel em: <http://heatonresearch-site.s3-website-us-east-1.amazonaws.com/javadoc/encog-3.3/org/encog/package-summary.html>. Acesso em: 1 Outubro 2016.

HEATON, J. **Programming Neural Networks with Encog3 in Java**. St. Louis: Heaton Research, 2011.

HENDERSHOTT, T.; JONES, C. M.; MENKVELD, A. J. Does algorithmic trading improve liquidity? **The Journal of Finance**, v. 66, n. 1, p. 1-33, 2011.

IBM. IBM 100 - Deep Blue. **IBM - Icons of Progress**, 2011. Disponivel em: <http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/>. Acesso em: 22 Outubro 2016.

KATHLEEN, C.; KOMAKI, I. How This Hedge Fund Robot Outsmarted Its Human Master. **Bloomberg News**, 2016. Disponivel em: <http://www.bloomberg.com/news/articles/2016-08-21/hedge-fund-robot-outsmarts-human-master-as-ai-passes-brexit-test>. Acesso em: 22 Outubro 2016.

KIMOTO, T. et al. Stock Market Prediction System with Modular Neural Networks. **1990 IJCNN International Joint Conference on**, Junho 1990. 1-6.

KRIEGER, P. E. Uso de redes neurais artificiais para predição da bolsa de valores, Itajaí, 2012. 91 folhas. Trabalho de conclusão de curso (Graduação em Ciência da Computação) - Centro de Ciências Tecnológicas da Terra e do Mar, Universidade do Vale do Itajaí, Itajaí, 2012.

MANTOVANI, W. A. **Utilização de Redes Neurais Recorrentes na Caracterização de Cargas Não Lineares em Sistemas Elétricos**. Ilha Solteira. 2011. Dissertação (Mestrado em Engenharia Elétrica) - Universidade Estadual Paulista, Ilha Solteira.

MARKOWITZ, H. Portfolio Selection. **The Journal of Finance**, 7, n. 1, Março 1952. 77-91.

MATHWORKS. Matlab. **MathWorks Products**, 2007. Disponivel em: <https://www.mathworks.com/products/matlab>. Acesso em: 2 Novembro 2016.

MCCULLOCH, W. S.; PITTS, W. H. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, v. 5, n. 4, p. 115-133, 1943.

MICROSOFT. Getting Started with VBA in Office 2010. **Documentation Office 2010**, 2009. Disponivel em: <https://msdn.microsoft.com/en-us/library/office/ee814735(v=office.14).aspx>. Acesso em: 31 Outubro 2016.

MICROSOFT. Produtos Office: Excel. **Produtos Office**, 2016. Disponivel em: <https://products.office.com/pt-BR/excel#>. Acesso em: 31 Outubro 2016.

MØLLER, M. F. A scaled conjugate gradient algorithm for fast supervised learning. **Neural networks**, v. 6, n. 4, p. 525-533, 1993.

MONARD, M. C.; BARANAUSKAS, J. A. **Sistemas Inteligentes:** Fundamentos e Aplicações. [S.l.]: Manole Ltda, 2003.

MOREIRA, R. B. T. **Construção do livro de ofertas a partir de dados de alta frequência e um algoritmo de predição de valores baseado em técnicas de agrupamento e regressão linear**. Universidade de São Paulo. São Paulo. 2013.

NEUROFURG. Uma ferramenta de apoio ao ensino de Redes Neurais Artificiais. **NeuroFURG**, 2011. Disponivel em: <http://neurofurg.sourceforge.net/>. Acesso em: 2 Novembro 2016.

NEUROPH. Neuroph v2.92. **Neuroph**, 2008. Disponivel em: <http://neuroph.sourceforge.net/>. Acesso em: 2 Novembro 2016.

NEUROPH. Neuroph Studio JavaDoc. **Neuroph Studio**, 2014. Disponivel em: <http://neuroph.sourceforge.net/javadoc/index.html>. Acesso em: 2 Novembro 2016.

NUMERAI. About: Numerai. **Numerai web site**, 2016. Disponivel em: <https://numer.ai/about>. Acesso em: 16 November 2016.

OF, W. S. R. G. A. T. U. O. A. S. **Neural Network Excel Add-in**, 2010.

OLIVEIRA, M. A. **Aplicação de redes neurais artificiais na análise de séries temporais econômico-financeiras**. São Paulo, p. 316 folhas. 2007. Tese (Doutorado em Administração) - Faculdade de Economia, Administração e Contabilidade da Universidade de São Paulo.

ORACLE. Articles. **Oracle Tech Network Articles**, 2014. Disponivel em: <http://www.oracle.com/technetwork/articles/java/ma14-java-se-8-streams-2177646.html>. Acesso em: 11 Abril 2017.

ORACLE. **Java 8**, 2014.

REFENES, A. N.; ZAPRANIS, A.; FRANCIS, G. Stock performance modeling using neural networks: A comparative study with regression models. **Neural Networks**, Oxford, 7, n. 2, 1994. 375-388.

RESEARCH, H. **Encog**, 2013.

RIEDMILLER, M.; BRAUN, H. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. **Neural Networks, 1993., IEEE International Conference On.**, 1993. 586-591.

RUSSELL, S. J.; NORVIG, P. **Artificial Intelligence:** a modern approach. 3ª. ed. New Jersey: Prentice Hall, 2005.

SCHMIDHUBER, J. A FIXED SIZE STORAGE O (n^3) TIME COMPLEXITY LEARNING ALGORITHM FOR FULLY RECURRENT CONTINUALLY RUNNING NETWORKS. **Neural Computation**, München, p. 243-248, 1992.

SOMMERVILLE, I. **Software engineering**. 9th. ed. Pearson: [s.n.], 2010.

SRIVASTAVA, N. et al. Dropout: a simple way to prevent neural networks from overfitting. **Journal of Machine Learning Research**, v. 15, n. 1, p. 1929-1958, 2014.

STRICKX, S. **Yahoofinance-api**, 2014.

TAFNER, M. A. Redes Neurais Artificiais: Aprendizado e Plasticidade, 1998. Disponivel em: <http://www.cerebromente.org.br/n05/tecnologia/rna.htm#neuronio>. Acesso em: 25 Setembro 2016.

THOMAZ, C. E.; VELLASCO, M. M. B. R. Análise de Tendências de Mercado por Redes Neurais Artificiais. **7º CBRN – Congresso Brasileiro de Redes Neurais**, Natal, 2005.

TURING, A. M. Computing machinery and intelligence. **Mind**, v. 59, n. 236, p. 433-460, 1950.

UNIVERSITY OF WAIKATO. Weka 3: Data Mining Software in Java. **Weka**, 2010. Disponivel em: <http://www.cs.waikato.ac.nz/ml/weka/documentation.html>. Acesso em: 2 Novembro 2016.

VON ZUBEN, F. J.; ATTUX, R. R. F. Redes Neurais Artificiais – Parte I, 2010. Disponivel em: <ftp://ftp.dca.fee.unicamp.br/pub/docs/vonzuben/ia004\_1s10/notas\_de\_aula/topico1\_IA004\_1s10\_Parte1.pdf>. Acesso em: 25 Setembro 2016.

WEKA. Class MultilayerPerceptron. **Weka Documentation**, 2010. Disponivel em: <http://weka.sourceforge.net/doc.stable/weka/classifiers/functions/MultilayerPerceptron.html>. Acesso em: 2 Novembro 2016.

WINRICH, B. **Training and Source Code Generation for Artificial Neural Networks**. Rhode Island. 2015. Tese de Doutorado (Master of Science in Computer Science). University of Rhode Island.