

**FACULDADE DE TECNOLOGIA DO IPIRANGA**

**CURSO DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS**

MARIO THOMAZ APRA

APLICAÇÃO DE REDES NEURAIS PARA PREVISÃO DE PREÇO DE FECHAMENTO 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 PARA PREVISÃO DE PREÇO DE FECHAMENTO 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 peça bem importante no começo da faculdade pelo incentivo.

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

Agradeço а todos os professores por me proporcionar о conhecimento não apenas racional, mas а manifestação do caráter е afetividade da educação por todo o processo da formação, por tanto que se dedicaram а mim, não somente por terem me ensinado com maestria, mas por terem me ensinado um 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.

*“Não sou obrigado a vencer, mas tenho o dever de ser verdadeiro. Não sou obrigado a ter sucesso, mas tenho o dever de corresponder à luz que tenho.“*

*(Abraham Lincoln*)

RESUMO

Uma rede neural que é capaz de prever os preços das ações pode ser muito útil para qualquer pessoa que queira investir em ações. Hoje em dia é necessário muito trabalho para extrair os dados da BM&FBOVESPA, normalizá-los, criar uma rede, configurá-la, treiná-la e verificar se ela é eficiente. Assim, este trabalho tem como objetivo criar e validar redes neurais aplicadas à predição de preços no mercado financeiro. Além disso, construir um software que facilite todas essas etapas para que até uma pessoa sem conhecimento em programação possa criar uma rede neural eficiente. Caracterizando-se como uma pesquisa qualitativa, documental com o método indutivo. O resultado encontrado foi ...

Palavras-chave: Inteligência Artificial, Rede Neural Artificial, predição, mercado financeiro.

ABSTRACT

A neural network that can predict stock prices can be very useful for anyone who wants to invest in stock market. Nowadays it is necessary a lot of work to extract the data from BM&FBOVESPA, normalize them, build a network, configure it, train it and them check if it is efficient. In that way, this research has the propose to build and validate neural network to predict stock market prices. Besides that, building a software that facilitates all these steps so that, even a person without programming knowledge, can create an efficient neural network. It Is characterized as a qualitative and documentary research, using the inductive method. The founded result was....

Keywords: Artificial Intelligence, Artificial Neural Network, prediction, stock market.

LISTA DE FIGURAS

[Figura 1. Modelo de um Neurônio Artificial 21](#_Toc479836922)

[Figura 2. Funções de ativação 22](#_Toc479836923)

[Figura 3. ActivationLOG 24](#_Toc479836924)

[Figura 4. ActivationSigmoid 24](#_Toc479836925)

[Figura 5. ActivationTANH. 25](#_Toc479836926)

[Figura 6. Comparação entre Elliott (contínua) e Sigmoid (pontilhada) 25](#_Toc479836927)

[Figura 7. Comparação Symmetric Elliott (contínua) e TANH (pontilhada). 26](#_Toc479836928)

[Figura 8. Multilayer perceptron (MLP) 27](#_Toc479836929)

[Figura 9. TLFN focada. 27](#_Toc479836930)

[Figura 10. *Recurrent* *Network* 28](#_Toc479836931)

[Figura 11. Custo de entendimento 29](#_Toc479836932)

[Figura 12. Hierarquia do aprendizado 30](#_Toc479836933)

[Figura 13. Redes com e sem *dropout* 31](#_Toc479836934)

[Figura 14. *Overfitting* com e sem *dropout* 31](#_Toc479836935)

[Figura 15. Maiores altas em 5 dias 35](#_Toc479836936)

[Figura 16. Maiores quedas em 5 dias 35](#_Toc479836937)

[Figura 17. *Product Backlog* 43](#_Toc479836938)

[Figura 18. *Sprint Backlog* 1 – Gerenciar RNA 44](#_Toc479836939)

[Figura 19. *Sprint Backlog* 2 – Treinar RNA 44](#_Toc479836940)

[Figura 20. *Sprint Backlog* 3 – Gerar Relatório 44](#_Toc479836941)

[Figura 21. *Sprint Backlog 4 -* Importar dados BMF 45](#_Toc479836942)

[Figura 22. Atores 45](#_Toc479836943)

[Figura 23. Formato importação BMF 47](#_Toc479836944)

[Figura 24. I001 – Gerenciar Rede Neural 48](#_Toc479836945)

[Figura 25. Formato importação RNA 50](#_Toc479836946)

[Figura 26. I002 - Treinar Rede Neural 51](#_Toc479836947)

[Figura 27. I003 - Gerar Relatório 53](#_Toc479836948)

[Figura 28. Diagrama de Classes de negócio 54](#_Toc479836949)

[Figura 29. Diagrama de implantação 55](#_Toc479836950)

[Figura 30. Diagrama de classes de projeto – Create NN 56](#_Toc479836951)

[Figura 31. Diagrama de sequência - Create NN 56](#_Toc479836952)

[Figura 32. Diagrama de classes de projeto - Choose NN 57](#_Toc479836953)

[Figura 33. Diagrama de sequência - Choose NN 57](#_Toc479836954)

[Figura 34. Diagrama de classes de projeto - Import BMF Data 57](#_Toc479836955)

[Figura 35. Diagrama de sequência - Import BMF Data 58](#_Toc479836956)

[Figura 36. Diagrama de classes de projeto - Comparative Chart 58](#_Toc479836957)

[Figura 37. Diagrama de sequência - Comparative Chart 59](#_Toc479836958)

[Figura 38. Diagrama de classes de projeto - Train NN 59](#_Toc479836959)

[Figura 39. Diagrama de sequência - Train NN 60](#_Toc479836960)

[Figura 40. Topologia da RNA utilizada 61](#_Toc479836961)

[Figura 41. Código implementação topologia 62](#_Toc479836962)

[Figura 42. Código *Factory* função de ativação 64](#_Toc479836963)

[Figura 43. Ajuste *Date Interval* 65](#_Toc479836964)

[Figura 44. Código *mixed data* 65](#_Toc479836965)

[Figura 45. Código *calculated data* 66](#_Toc479836966)

[Figura 46. Código vetor de pessoa 67](#_Toc479836967)

[Figura 47. Código *normalize data* 68](#_Toc479836968)

[Figura 48. Código *get max and min* 69](#_Toc479836969)

[Figura 49. Código *getNormalizedValue* 70](#_Toc479836970)

[Figura 50 Código exemplo Javadoc 72](#_Toc479836971)

[Figura 51. Resultado javadoc do método *train* 72](#_Toc479836972)

[Figura 52. Exemplo de *inner class* 76](#_Toc479836973)

[Figura 53. Diferença entre *inner class* e *lambda* 76](#_Toc479836974)

[Figura 54. Implementação do *main* com *lamda* 77](#_Toc479836975)

[Figura 55. Diferenças entre *for* 77](#_Toc479836976)

[Figura 56. Código *lambda* e *method* *reference* 78](#_Toc479836977)

[Figura 57. Código *compareByVolume* 78](#_Toc479836978)

[Figura 58. Código de ordenação 79](#_Toc479836979)

[Figura 59. Código exemplo *stream* – pegando dados 79](#_Toc479836980)

[Figura 60. Exemplo código *stream* - sobrescrita *toString* 80](#_Toc479836981)

[Figura 61. Saída do código de exemplo da *stream* 80](#_Toc479836982)

[Figura 62. Código exemplo *stream* 81](#_Toc479836983)

[Figura 63. Ilustração código stream 82](#_Toc479836984)

[Figura 64. Código *stream* com paralelismo 83](#_Toc479836985)

[Figura 65. Código exemplo paralelismo 84](#_Toc479836986)

[Figura 66. Saída código exemplo paralelismo 84](#_Toc479836987)

[Figura 67. Código updateMaxMin 85](#_Toc479836988)

LISTA DE TABELAS

[Tabela 1. Comparativo Elliott 25](#_Toc479552310)

[Tabela 2. Preços PRIO3 “puros” 33](#_Toc479552311)

[Tabela 3. PRIO3 normalizado 33](#_Toc479552312)

[Tabela 4. Comparativo softwares 37](#_Toc479552313)

[Tabela 5. Notas softwares 37](#_Toc479552314)

LISTA DE EQUAÇÕES

[Equação 1. Neurônio Artificial 21](#_Toc479552315)

[Equação 2. ActivationLOG 22](#_Toc479552316)

[Equação 3. ActivationSigmoid 23](#_Toc479552317)

[Equação 4. ActivationTANH 24](#_Toc479552318)

[Equação 5. Elliott 24](#_Toc479552319)

[Equação 6. Symmetric Elliott 25](#_Toc479552320)

[Equação 7. Normalização 32](#_Toc479552321)

[Equação 8. Normalização com margem 32](#_Toc479552322)

[Equação 9. Normalização exemplo 33](#_Toc479552323)

LISTA DE ABREVIATUAS E SIGLAS

RNA - Rede Neural Artificial

LOG - Logaritmo

TANH - Tangente Hiperbólica

MLP - *Multlayer 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](#_Toc479700254)

[1.1 Objetivos 18](#_Toc479700255)

[1.2 Justificativa 18](#_Toc479700256)

[1.3 Métodos e Tecnologias 18](#_Toc479700257)

[1.4 Organização do Trabalho 19](#_Toc479700258)

[2. REDES NEURAIS ARTIFICIAIS 20](#_Toc479700259)

[2.1 Motivação para utilizar Redes Neurais na Predição. 21](#_Toc479700260)

[2.2 Funções de Ativação 21](#_Toc479700261)

[2.2.1 ActivationBiPolar 22](#_Toc479700262)

[2.2.2 Activation Competitive 22](#_Toc479700263)

[2.2.3 ActivationLinear 22](#_Toc479700264)

[2.2.4 ActivationLOG 22](#_Toc479700265)

[2.2.5 ActivationSigmoid 23](#_Toc479700266)

[2.2.6 ActivationSoftMax 23](#_Toc479700267)

[2.2.7 ActivationTANH 24](#_Toc479700268)

[2.2.8 Elliott 24](#_Toc479700269)

[2.2.9 Symmetric Elliott 25](#_Toc479700270)

[2.3 Topologias da Rede Neural 25](#_Toc479700271)

[2.3.1 Feed-Forward 26](#_Toc479700272)

[2.3.2 Recurrent Network 27](#_Toc479700273)

[2.4 Aprendizagem 28](#_Toc479700274)

[2.4.1 Backpropagation 31](#_Toc479700275)

[2.4.2 ResilientPropagation 31](#_Toc479700276)

[2.4.3 ManhattanPropagation 31](#_Toc479700277)

[2.4.4 QuickPropagation 31](#_Toc479700278)

[2.4.5 ScaledConjugateGradient 32](#_Toc479700279)

[2.5 Normalização 32](#_Toc479700280)

[3. Benchmark 35](#_Toc479700281)

[3.1 Excel + VBA 35](#_Toc479700282)

[3.2 Weka 35](#_Toc479700283)

[3.3 Neuroph Studio 36](#_Toc479700284)

[3.4 Matlab 36](#_Toc479700285)

[3.5 NeuroFURG 36](#_Toc479700286)

[3.6 Comparativo 37](#_Toc479700287)

[4. REQUISITOS DO SISTEMA DE SOFTWARE 39](#_Toc479700288)

[4.1 Identificação dos requisitos 39](#_Toc479700289)

[4.1.1 Prioridades dos requisitos 39](#_Toc479700290)

[4.1.2 Requisitos Funcionais 40](#_Toc479700291)

[4.1.3 Requisitos Não-Funcionais 41](#_Toc479700292)

[4.1.4 Regras de Negócio 42](#_Toc479700293)

[4.1.5 Product e Sprint Backlog 42](#_Toc479700294)

[4.2 Modelagem dos requisitos funcionais 44](#_Toc479700295)

[4.2.1 Atores 44](#_Toc479700296)

[4.2.2 Diagrama de Caso de uso 44](#_Toc479700297)

[4.2.3 Especificação do Caso de Uso 46](#_Toc479700298)

[5. Desenvolvimento do projeto 53](#_Toc479700299)

[5.1 Análise 53](#_Toc479700300)

[5.1.1 Diagrama de Classes de Análise (Visão de Negócio) 53](#_Toc479700301)

[5.2 Projeto 54](#_Toc479700302)

[5.2.1 Arquitetura do Sistema 54](#_Toc479700303)

[5.2.2 Diagrama de Classes de Projeto por Caso de Uso 55](#_Toc479700304)

[5.3 Detalhes de implementação. 60](#_Toc479700305)

[5.3.1 Topologia 60](#_Toc479700306)

[5.3.2 Função de ativação 62](#_Toc479700307)

[5.3.3 Date Interval 63](#_Toc479700308)

[5.3.4 Normalizador de dados 67](#_Toc479700309)

[5.3.5 Javadoc 70](#_Toc479700310)

[5.3.6 Recursos do Java 8 74](#_Toc479700311)

[5.3.7 Ajustes de implementação 84](#_Toc479700312)

[6. Resultados Obtidos 86](#_Toc479700313)

[6.1 Criar RNA 86](#_Toc479700314)

[6.2 Escolher RNA 86](#_Toc479700315)

[6.3 Treinar RNA 86](#_Toc479700316)

[6.4 Gerar Relatório 86](#_Toc479700317)

[7. Considerações Finais 87](#_Toc479700318)

[8. Bibliografia 88](#_Toc479700319)

# 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 a qual 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)

A empresa que trabalha com IA voltada para o mercado que mais chamou atenção foi a Numerai, que é uma empresa criada no final de 2015 como sendo um torneio para predição do mercado financeiro para cientistas de dados e desde lá, já recebeu 14 bilhões de preços previstos. O mais interessante é que eles pegam os modelos que possuem maior acurácia e usam em seu fundo de investimento real e em troca, remunera os melhores contribuintes com Bitcoin e, além disso, é gratuito e livre para qualquer pessoa submeter sua rede para eles. (NUMERAI, 2016)

Voltando um pouco para a origem da AI, um questionamento muito importante no começo da sua evolução é: um computador pode pensar? Essa foi uma pergunta com a qual Alan Turing (1950), “avô” 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 essa pergunta, uma motivação de se trabalhar com redes neurais é a diferença que um cérebro humano processa uma informação em relação a um computador convencional. O cérebro é altamente complexo, não linear e processa as informações paralelamente 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 AI ocorreu em maio de 1997, na qual *Deep Blue*, 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. Este feito foi importante não só para vencer uma partida de xadrez, mas a arquitetura usada na construção desta brilhante máquina foi aplicada na análise de risco para o mercado financeiro, na dinâmica molecular para descobrir e desenvolver novos remédios, no *data mining* e, principalmente, para desenvolver e entender os limites da programação e 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 *algorithmic trading* para aumentar a liquidez de um papel, como no trabalho do Hendershott, Jones e Menkveld (2011) ou para operar em mercados, como no do Chabou, Chiquioine, *et al* (2014) até o uso da AI para tomar as 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).

Vários trabalhos acadêmicos foram realizados demonstrando a eficácia do uso de AI e redes neurais para predição dos preços das ações, um dos primeiros feito por Kimoto, Asakawa, *et al.* (1990), ou realizado por Thomaz e Vellasco (2005), ou por Refenes, Zapranis e Francis (1994), entre outros. Os trabalhos elencados não abordavam a criação de uma rede em si, somente sua aplicação. Desta forma, essa pesquisa tem como diferencial o desenvolvimento de um software.

## Objetivos

O objetivo deste trabalho é criar e validar redes neurais aplicadas à predição de preços no mercado financeiro.

## Justificativa

Ao tentar reproduzir trabalhos já feitos na área de redes neurais aplicadas ao mercado financeiro, perde-se muito tempo com toda a parte *background* e necessita-se de um bom conhecimento em programação para criar uma rede.

Em todos os trabalhos pesquisados, as etapas de extração dos dados, normalização, configuração, treinamento e validação a rede e, por fim, gerar um relatório com os resultados, não era o objetivo e era uma etapa que todos os pesquisadores perdiam certo tempo. Nos trabalhos, várias ferramentas foram utilizadas, como o Excel e o Neuroph Studio no trabalho do Krieger (2012), Matlab e Eviews no trabalho do Oliveira (2007), entre outros.

## Métodos e Tecnologias

O trabalho é apresentado por meio de levantamentos bibliográficos, tipo de pesquisa documental, qualitativa e com o método indutivo. O software está escrito na linguagem Java 8, com a biblioteca YahooFinance API para extração dos dados e Encog Framework para a criação da rede neural. O versionamento do sistema é feito pelo GitHub com o ambiente de desenvolvimento (IDE) Eclipse.

## Organização do Trabalho

Este trabalho é composto por 7 capítulos, incluindo esta introdução. O Capítulo 2 introduz os conceitos gerais de redes neurais. O Capítulo 3 apresenta algumas ferramentas para a criação de redes neurais, analisando suas contribuições e o que deixa 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 os resultados obtidos. O Capítulo 7 apresenta as considerações finais.

# REDES NEURAIS ARTIFICIAIS

Rede Neural 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, podemos dizer que uma Rede Neural Artificial é uma combinação de neurônios artificiais. Para entender uma RNA, é imprescindível saber como funciona um neurônio. 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*.”

A Figura 1 ilustra uma representação de um neurônio artificial:

Figura 1. Modelo de um Neurônio Artificial



Fonte: (VON ZUBEN e ATTUX, 2010).

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



Equação 1. Neurônio Artificial

Onde *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).

## Motivação para utilizar Redes Neurais na Predição.

O principal benefício do uso de uma RNA (Rede Neural Artificial) 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, onde diversos fatores podem afetar o preço de uma ação, como fatores psicológicos, notícias, divulgação de resultados, regulamentação, entre outros, esta 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 em um intervalo de [0,1] ou [-1,1] (HAYKIN, 2009). Ela tem a função de manter o neurônio ativa quando as entradas fornecidas forem corretas e mantê-lo 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). Esta última é a mais utilizada e é uma mistura entre as funções lineares e não lineares (HAYKIN, 2009), que serão exemplificadas na Figura 2.

Figura 2. Funções de ativação



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

A seguir é apresentado algumas das funções de ativação que estão implementadas no software. As funções que possuírem derivada pode ser utilizado com a aprendizagem do tipo *propagation*. (HEATON, 2011).

### ActivationBiPolar

Utilizada em redes que requerem valores binários (*true* ou *false*). A ideia básica é que se o valor for positivo o resultado é 1, senão é -1 (HEATON, 2011). Não possui derivada. Está ilustrada na Figura 2 (c).

Este tipo de função de ativação foi utilizado primeiramente por McCulloch e Pits (1943), onde a saída do neurônio assumia o valor de 1 ou 0 (no lugar do -1). Em seu trabalho foi referenciado com o termo de *all-or-none property* (propriedade de tudo ou nada).

### Activation Competitive

É utilizada para forçar um seleto grupo de neurônios para ganhar. O vencedor é o grupo com a maior saída. Para isso, percorre todos os parâmetros e acha os vencedores, e atribui aos perdedores o valor zero. Não possui derivada.

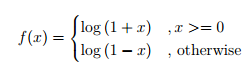
Este tipo de função de ativação pode ser usado tanto para redes competitivas quanto para mapas auto-organizáveis (HEATON, 2011).

### ActivationLinear

É utilizada em tipos específicos de redes, que não possuem função de ativação, como por exemplo, mapas auto-organizáveis. Este tipo está representado na Figura 2 (a), na qual o valor que é passado para x é igual a f(x). Por essa razão, na primeira (entrada) e na última (saída), será sempre utilizado esse tipo de função.

Pode resultar em números negativos. Como sua derivada é constante, também pode ser utilizada no treinamento do tipo *propagation* (HEATON, 2011).

### ActivationLOG



Equação 2. ActivationLOG

Figura 3. ActivationLOG



Fonte: (HEATON, 2011)

Possui uma curva similar com a tangente hiperbólica, que será apresentada posteriormente. Este tipo de função de ativação é útil para prevenir saturação, portanto é uma escolha possível quando o treinamento não obtém sucesso com a tangente hiperbólica.

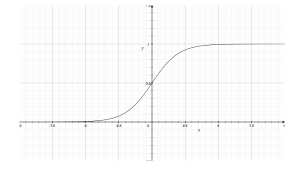
Ela pode fazer com que a etapa de treinamento demore significativamente. Pode resultar em números negativos e possui derivada. (HEATON, 2011)

### ActivationSigmoid



Equação 3. ActivationSigmoid

Figura 4. ActivationSigmoid



Fonte: (HEATON, 2011)

Só produz números positivos. É uma escolha bem comum para redes *feed-forward* e algumas redes recorrentes simples. Se requerido números negativos, a tangente hiperbólica pode ser uma solução melhor. Possui derivada (HEATON, 2011).

### ActivationSoftMax

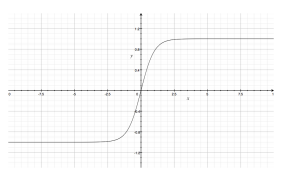
Só vai produzir saídas iguais a uma. Geralmente é utilizada nas camadas de saída para problemas que envolvem classificação, mas também pode ser usada nas camadas escondidas. (HEATON, 2011)

### ActivationTANH



Equação 4. ActivationTANH

Figura 5. ActivationTANH.



Fonte: (HEATON, 2011)

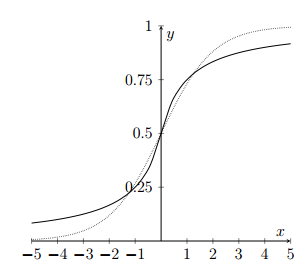
Utiliza funções de tangente hiperbólica. É, provavelmente, a função mais comum, por trabalhar com valores positivos e negativos. Possui derivada (HEATON, 2011).

### Elliott



Equação 5. Elliott

Figura 6. Comparação entre Elliott (contínua) e Sigmoid (pontilhada)



Fonte: (WINRICH, 2015)

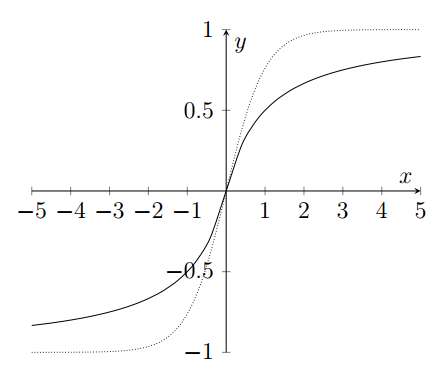
Foi originalmente proposta por David L. Elliott em 1993 como uma alternativa computacionalmente mais eficiente comparada a sigmoide e a tangente hiperbólica. Também produz valores entre 0 e 1. Possui derivada (WINRICH, 2015).

### Symmetric Elliott



Equação 6. Symmetric Elliott

Figura 7. Comparação Symmetric Elliott (contínua) e TANH (pontilhada).

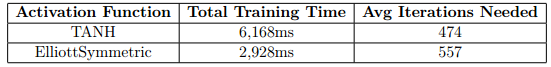


Fonte: (WINRICH, 2015)

A ideia é a mesma da Elliott, mas para substituir a TANH, também produz valor entre -1 e 1 e possui derivada.

Conforme a Tabela 1 apesar de requerer mais iterações, o tempo total de treinamento é menos do que a metade na média dos treinos se comparada a TANH.

Tabela 1. Comparativo Elliott



Fonte: (WINRICH, 2015)

## Topologias da Rede Neural

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*). Estão presentes 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)

Figura 8. Multilayer perceptron (MLP)



Fonte: (HAYKIN, 2009)

De acordo com a Figura 8, percebe-se que os dados têm um fluxo que vai da camada de entrada, passando pelas camadas escondidas (se tiver) 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 para a mesma.

#### 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 este 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 pode ser adicionada nos pesos sinápticos ou na entrada da rede (HAYKIN, 2009).

Um exemplo para este caso, que é abordado no trabalho, é a TLFN focada (*focused time lagged feed-forward network*), onde os atrasos de tempo são incorporados somente na camada de entrada.

Figura 9. TLFN focada.



Fonte: (HAYKIN, 1998)

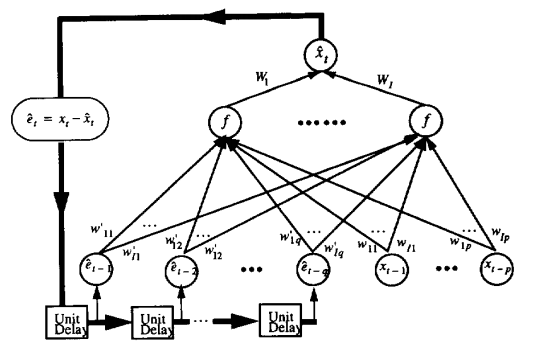
A **Erro! Fonte de referência não encontrada.** ilustra uma TLFN focada, onde as entradas são apresentadas à camada de entrada neurônio por neurônio, ou seja, com um *delay.* No instante *n* é apresentado ao primeiro neurônio, no n*-1* no segundo neurônio e assim por diante, até atingir o número *n-p* de atraso.

### 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 sua performance (HAYKIN, 2009).

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

Figura 10. *Recurrent* *Network*



Fonte: (CONNOR, MARTIN e ATLAS, 1994).

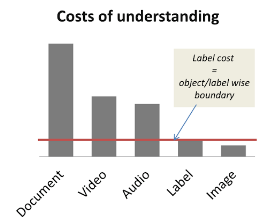
A Figura 10 ilustra uma rede recorrente, e nota-se a diferença dita anteriormente: no fluxo de dados tem uma parte cíclica.

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 (HEATON, 2011). Ela é um dos fatores mais importantes na boa formação da RNA. O gargalo para a ampla utilização dos sistemas inteligentes é a aquisição de conhecimento, que ainda é artesanal e subjetiva. (MONARD e BARANAUSKAS, 2003). A Figura 11 mostra o custo de aprendizado para vários tipos de dados diferentes:

Figura 11. Custo de entendimento



Fonte: (CHERMAN, TSOUMAKAS e MONARD, 2016).

Figura 12. Hierarquia do aprendizado



Fonte: (MONARD e BARANAUSKAS, 2003).

Monard e Baranauskas (2003) explicam muito bem a Figura 12, suas tipologias e vários paradigmas de aprendizado. Neste trabalho será explicado somente o aprendizado supervisionado, que será utilizado.

“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 disponível os seguintes dados para formar a entrada: preço de abertura, preço máximo do dia, preço mínimo do dia, preço de fechamento (cotação) e volume.

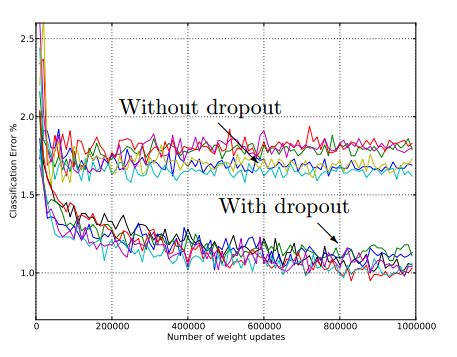
Figura 13. Redes com e sem *dropout*



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

Além disso, existe mais uma entrada muito importante: 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 ver na Figura 13.

Figura 14. *Overfitting* com e sem *dropout*



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

A Figura 14 mostra um gráfico comparando a diferença entre uma rede que faz o uso do *dropout* e outra sem, em diferentes arquiteturas computacionais. Dá para notar que a rede fica com *overfitting* antes das primeiras 100.000 nas primeiras atualizações dos números dos neurônios, já na rede que usa o *dropout* ela perde a eficiência na casa dos 800.000.

A seguir é apresentado os principais algoritmos de treinamento abordados pelo software desenvolvido:

### Backpropagation

O livro de Chauvin e Rumelhart (1995) aborda muito bem 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 da camada de saída. 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*).

### ResilientPropagation

Este algoritmo foi criado com a finalidade de corrigir o problema da magnitude da derivada parcial de cada peso (usado para descobrir a função erro do passo 1 descrito anteriormente) no *Backpropagation*. Além disso, tem o problema que a taxa de aprendizagem é o mesmo valor para a rede toda. (RIEDMILLER e BRAUN, 1993)

O *ResilientPropagation* utiliza um valor de atualização especial (similar à taxa de aprendizagem) em cada conexão que é automaticamente determinada (ao contrário da forma anterior). (HEATON RESERACH, 2014)

### ManhattanPropagation

Para corrigir os problemas já descritos do *Backpropagation*, este algoritmo usa a derivada parcial para somente indicar o sinal para a atualização da matriz dos pesos sinápticos, já o valor (módulo) é obtido com uma constante qualquer. Em geral, deve-se começar com uma constante alta e ir diminuindo conforme a necessidade (HEATON RESERACH, 2014).

### QuickPropagation

Algoritmo desenvolvido por meio de um estudo empírico com o objetivo de aumentar a rapidez do treinamento, em comparação ao *Backpropagation*. (FAHLMAN, 1988)

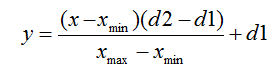
### ScaledConjugateGradient

Assim como o algoritmo anterior, este foi criado para aumentar a rapidez do treinamento, em comparação ao *Backpropagation*. (MØLLER, 1993)

## Normalização

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

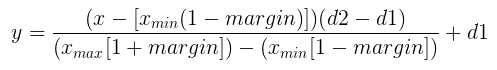
É possível normalizar de várias formas, desde regra de três até com o uso de equações. A ideia básica é verificar qual é o valor máximo possível, o mínimo, e ver a razão do numero entre eles, como mostra na Equação 7.



Equação 7. 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. Suponhamos que temos n dados de entrada do limite [0,10], então xmin = 0 e xmax = 10. Reduziremos os dados ao limite [0,1], então d1 = 0 e d2 = 1.

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



Equação 8. Normalização com margem

Como mostra a Equação 8, a margem é adicionada somente nos 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.

A Tabela 2 exemplifica os dados utilizados pelo software no período de 13 dias úteis para a ação de PRIO3. A tabela 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 em verde representam os valores máximos da amostra, e os valores em vermelho representam os valores mínimos da amostra. Esses valores serão úteis para a normalização dos dados, conforme já mostrado nas equações anteriores.

Tabela 2. 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 dos preços fica da seguinte maneira:



Equação 9. Normalização exemplo

Com isso, a Tabela 3 mostra os todos os dados normalizados conforme as condições anteriores. Para o volume, usa-se a mesma ideia, porém os valores para *xmax* e *xmin* serão 57400 e 0, respectivamente.

Tabela 3. PRIO3 normalizado



Fonte: Autor (2017)

O valor padrão (*default*) no software é 100%, ou seja, uma ação pode alterar em 100% seus preços limites (dobrar seu preço máximo ou “zerar”) sem que isso interfira na rede. Este é um ambiente totalmente possível, como mostra a Figura 15 e a Figura 16.

Figura 15. Maiores altas em 5 dias



Fonte: (BARCHART, 2016)

Figura 16. Maiores quedas em 5 dias



Fonte: (BARCHART, 2016)

# Benchmark

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 deles. 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)

O VBA é uma linguagem de programação que auxilia e customiza os aplicativos do Office. Com ele é possível automatizar tarefas repetitivas e integrar entre aplicações do Office. (MICROSOFT, 2009)

Com o Excel é possível criar campos para escolher o período de data em que quer 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. Com os dados devidamente extraídos, descobre os valores limites da amostra e faz a normalização dos valores com possíveis margens no Excel. A parte de treinar a rede é possível com o uso de alguns add-ins como o NeuroXL, ou o NeuroNetwork Add-in. Essas opções não são tão completas, mas é possível criar uma rede, mesmo que bem simples. Aparte de gerar o relatório e executada facilmente com a criação de gráficos no Excel e com cálculo de desvio padrão ou variância usando fórmulas já prontas.

## Weka

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

Com o Weka é possível normalizar os dados, escolhendo somente os limites (BROWNLEE, 2016). Não da para escolher uma margem de forma fácil, só se aplicar a Equação 8 e descobrir quais serão os valores limites, então 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 o algoritmo de aprendizagem não é suportada. Também é possível escolher o período de treinamento, o de validação e configurar alguns avisos caso a rede dê algum tipo de erro (WEKA).

## Neuroph Studio

O Neuroph Studio é um framework leve para Java que auxilia o desenvolvimento de arquiteturas de redes neurais. Este software tem o intuito de simplificar o desenvolvimento de redes neurais (NEUROPH).

Com o uso desse software é possível criar uma RNA bem completa, escolhendo tanto quantos neurônios por camada e quantas camas, como também a função de ativação (chamada por eles de função de transferência) por neurônio e, por fim, o algoritmo de aprendizado (chamada de regra de aprendizado). Além da rede, é possível, como no caso do Weka, fazer a normalização dos valores somente escolhendo os limites, sem a margem. (NEUROPH, 2014)

Ele também tem o recurso de gerar um gráfico com o total de erros por iteração, mas nem o desvio padrão nem a variância 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 *machine learning*, processamento de signos, processamento de imagem, visão computacional, comunicação, computação financeira, robóticas e muito mais (MATHWORKS)

Com ele é possível normalizar os dados de entrada (assim como os dois anteriores, sem a possibilidade de escolher a margem), 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 época (DEMUTH e BEALE, 2003).

## NeuroFURG

O NeuroFURG é uma ferramenta de construção e simulação de RNA com o foco dos modelos de neurônios Perceptron e Adaline (não abordado neste trabalho) para estudantes de computação. (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

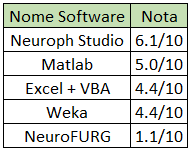
Tabela 4. Comparativo softwares



Fonte: Autor (2017)

Podemos notar que nenhum software é capaz de realizar todas as operações. Enquanto em uns são voltados somente na rede em si, em outros é possível fazer a normalização dos dados e a extração, conforme a **Erro! Fonte de referência não encontrada.**.

Tabela 5. Notas softwares



Fonte: Autor (2017)

As notas da Tabela 5 foram calculadas com a quantidade de etapas que o determinado software faz e pela quantidade de etapas totais. Dividindo o primeiro com o segundo valor, tem-se a porcentagem de etapas que o software consegue atingir, e multiplicando esse valor por 10, chega-se nas notas.

Com base na **Erro! Fonte de referência não encontrada.** e na Tabela 5, o conjunto mais eficiente para realizar todo o procedimento de criar e validar uma RNA para predição de ações no mercado financeiro é com o uso do Excel + VBA com o Neuroph Studio.

# 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. O texto a seguir tem como objetivo relembrar conceitos e padrões de especificação dos requisitos de software.

## Identificação dos requisitos

Por convenção, a referência a requisitos é feita através do nome da subseção onde eles estão descritos, seguidos do identificador do requisito, de acordo com a especificação a seguir:

* **Requisitos funcionais** devem ser identificados por: [RFxxx] – nome e descrição e serem descritos na seção requisitos funcionais.
* **Regras de negócio** devem ser identificadas por: [RNxxx] – nome e descrição e serem descritas na seção requisitos funcionais, subitem regras de negócio.
* **Requisitos não funcionais** devem ser identificados por: [RNFxxx] – nome e descrição e serem descritos na seção requisitos não funcionais.

Os requisitos devem ser identificados com um identificador único. A numeração inicia com o identificador [RF001], [RN001] ou [RNF001] e prossegue sendo incrementada à medida que forem surgindo novos requisitos ou regras.

Por convenção, a referência aos casos de uso é feita através do nome da subseção onde eles estão descritos, seguidos do identificador do caso de uso, de acordo com a especificação a seguir:

Casos de Uso devem ser identificados por: CSUXXX – nome a ser descrito na função modelagem funcional.

Os casos de uso devem ser identificados com um identificador único. A numeração inicia com o identificador CSU001 e prossegue sendo incrementada à medida que forem surgindo novos casos de uso.

### Prioridades dos requisitos

Para estabelecer a prioridade dos requisitos, foram adotadas as denominações “essencial”, “importante” e “desejável”.

* **Essencial** é o requisito sem o qual o sistema não entra em funcionamento. Requisitos essenciais são requisitos imprescindíveis, que têm que ser implementados impreterivelmente.
* **Importante** é o requisito sem o qual o sistema entra em funcionamento, mas de forma não satisfatória. Requisitos importantes devem ser implementados, mas, se não forem, o sistema poderá ser implantado e usado mesmo assim.
* **Desejável** é o requisito que não compromete as funcionalidades básicas do sistema, isto é, o sistema pode funcionar de forma satisfatória sem ele. Requisitos desejáveis podem ser deixados para versões posteriores do sistema, caso não haja tempo hábil para implementá-los na versão que está sendo especificada.

### Requisitos Funcionais

Neste item devem ser 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. A seguir são apresentados exemplos.

**[RF001] – Importar Dados da BMF**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Prioridade**: | X | Essencial | - | Importante | - | Desejável |

**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**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Prioridade**: | X | Essencial | - | Importante | - | Desejável |

**Descrição**: Este requisito permite que o usuário gerencie uma RNA, sendo possível criar uma RNA (sua topologia, qual ação que a rede é “especialista”), salvar essa rede localmente em um formato definido pelo framework Encog (.eg) e recuperará-la.

**[RF003] – Treinar rede neural**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Prioridade**: | X | Essencial | - | Importante | - | Desejável |

**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**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Prioridade**: | X | Essencial | - | Importante | - | Desejável |

**Descrição**: O sistema deverá um gráfico comparando os preços esperados e o calculado, bem como o coeficiente de variação e o desvio padrão.

### Requisitos Não-Funcionais

Neste item devem ser apresentados os requisitos não funcionais, que especificam restrições sobre os serviços ou funções providas pelo sistema. A seguir são apresentados alguns exemplos de requisitos não funcionais.

**Requisitos de Produto:** Conjunto de requisitos para prover conteúdo aos usuários cadastrados.

**[RNF001] – Usabilidade**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Prioridade**: | - | Essencial | - | Importante | X | Desejável |

**Descrição**: O sistema deve prover interface simples e intuitiva, de fácil navegação para facilitar o uso do mesmo por parte dos usuários.

**[RNF002] – Apresentação da Interface Gráfica**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Prioridade**: | X | Essencial | - | Importante | - | Desejável |

**Descrição**: O sistema deve fazer uso, exclusivamente, da língua inglesa para todo e qualquer texto apresentado.

**[RNF003] – Linguagem de programação adotada**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Prioridade**: | X | Essencial | - | Importante | - | Desejável |

**Descrição**: O sistema deve utilizar a linguagem Java versão 1.8.

**[RNF004] – Sistema Operacional alvo**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Prioridade**: | X | Essencial | - | Importante | - | Desejável |

**Descrição**: O sistema deve ser executado em qualquer SO que suporte a máquina virtual do Java (JVM).

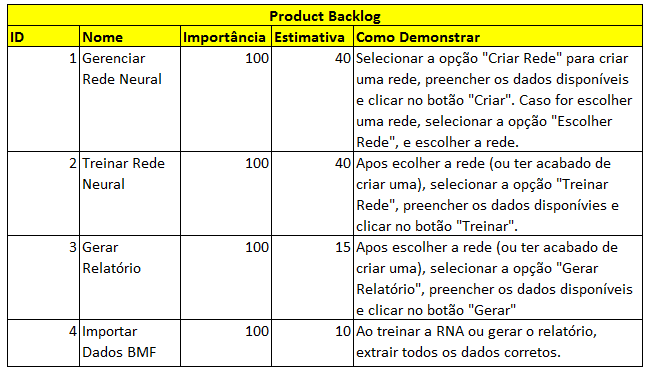
### Regras de Negócio

**[RN001] – Alterar topologia reseta pesos**

**Descrição:** Toda vez que alterar a topologia de uma rede, seus pesos serão alterados para valores aleatórios.

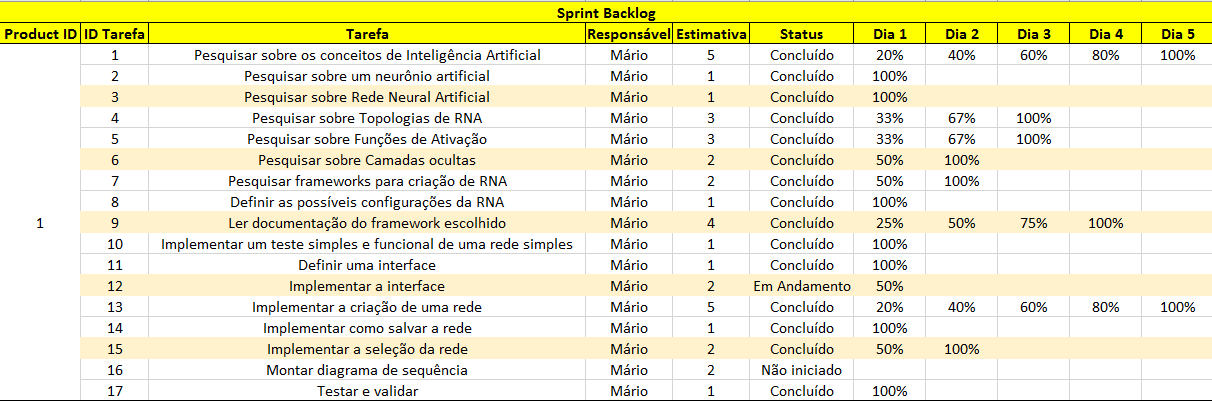
### Product e Sprint Backlog

Figura 17. *Product Backlog*



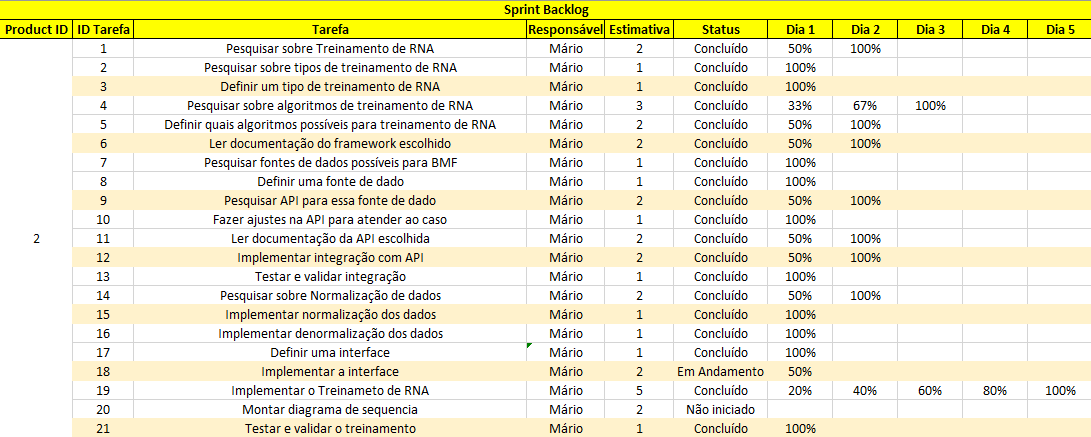
Fonte: Autor (2017)

Figura 18. *Sprint Backlog* 1 – Gerenciar RNA



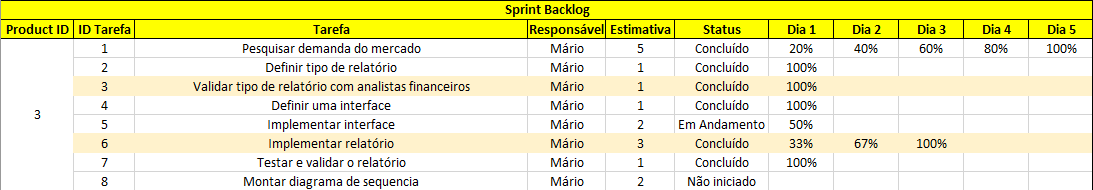
Fonte: Autor (2017)

Figura 19. *Sprint Backlog* 2 – Treinar RNA



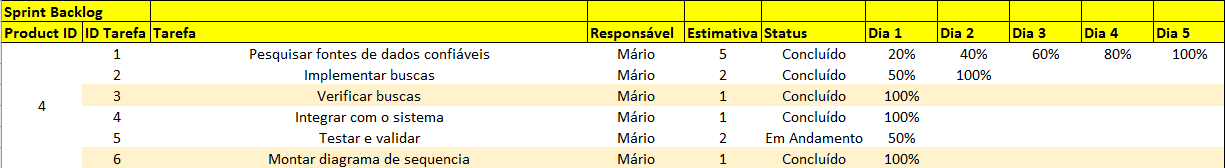
Fonte: Autor (2017)

Figura 20. *Sprint Backlog* 3 – Gerar Relatório



Fonte: Autor (2017)

Figura 21. *Sprint Backlog 4 -* Importar dados BMF



Fonte: Autor (2017)

## Modelagem dos requisitos funcionais

Neste item devem ser descritos 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. A seguir um exemplo de especificação de atores, do diagrama de caso de uso e da especificação de caso de uso.

### Atores

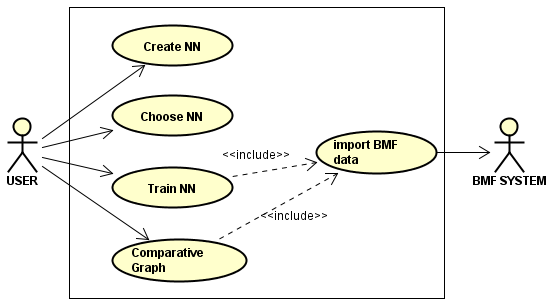
A seguir são apresentados exemplos da especificação de atores.

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

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

### Diagrama de Caso de uso

Figura 22. Atores

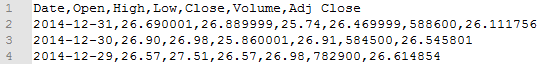


Fonte: Autor (2017)

### Especificação do Caso de Uso

|  |  |
| --- | --- |
| **CSU001 – Importar Dados da BMF** | |
| Sumário: | Importar Dados da BMF |
| Ator Primário: | Sistema BMF |
| Casos de Uso Associados: CSU002 | |
| **Pré-condição:** | |
| **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:** | |
| **Pós-condições:**  **a.** O Sistema armazena os dados. | |
| **Requisitos:** RF001 | |
| **Regras de Negócio:** | |
| **Interface:** | |
| **Formato dos dados de importação:**  Figura 23 | |

Figura 23. Formato importação BMF

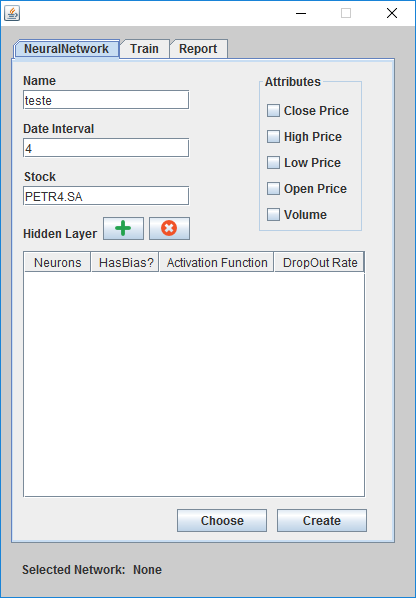


Fonte: Autor (2017)

Como é possível observar na Figura 23, os dados obtidos da BMF vêm em um formato .csv. O sistema é responsável por extrair os dados deste arquivo.

|  |  |
| --- | --- |
| **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 a ser comparadas, a quantidade de data necessária para gerar o próximo dado. 3. O usuário informa os campos necessários. 4. O caso de uso termina. | |
| **Pós-condições:**  **a.** Rede Neural criada. | |
| **Requisitos:** RF002 | |
| **Regras de Negócio:** | |
| **Interface:** I001 – | |

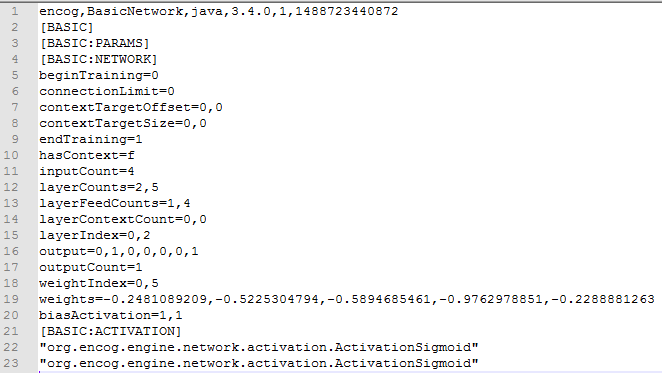
Figura 24. I001 – Gerenciar Rede Neural



Fonte: Autor (2017)

|  |  |
| --- | --- |
| **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. | |
| **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 as novas configurações. 4. O Sistema altera os pesos dos neurônios para um número randômico. (RN001) 5. O caso de uso termina | |
| **Pós-condições:**  **a.** A RNA é carregada em memória**.** | |
| **Requisitos:** RF002. | |
| **Regras de Negócio:** | |
| **Interface:** I001 – Gerenciar Rede Neural | |
| **Formato dos dados de importação:**  Figura 25 | |

Figura 25. Formato importação RNA

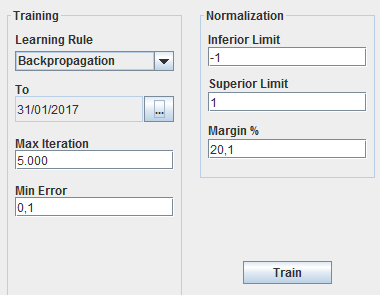


Fonte: Autor (2017)

A Figura 25 mostra o padrão de arquivo utilizado pelo framework da Encog (.eg) para salvar a RNA em um arquivo local.

|  |  |
| --- | --- |
| **CSU004 – Treinar Rede Neural** | |
| Sumário: | Treinar Rede |
| Ator Primário: | Usuário |
| Ator Secundário: | Sistema BMF |
| **Caso de Uso associado:** CSU002. | |
| **Pré-condição:**   1. O Usuário deve ter selecionado uma RNA. (CSU002) 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 4. O caso de uso termina. | |
| **Fluxo de Exceção (3) – Sem Conexão com a internet**   1. O Sistema informa que não está conectado à internet 2. O caso de uso termina. | |
| **Fluxo de Exceção (3) – Sem dados da BMF**   1. O Sistema informa que não foi possível obter os dados da BMF 2. O caso de uso termina. | |
| **Pós-condições:**  **a.** O Sistema treina a RNA. | |
| **Requisitos:** RF003. | |
| **Interface**: I002 | |

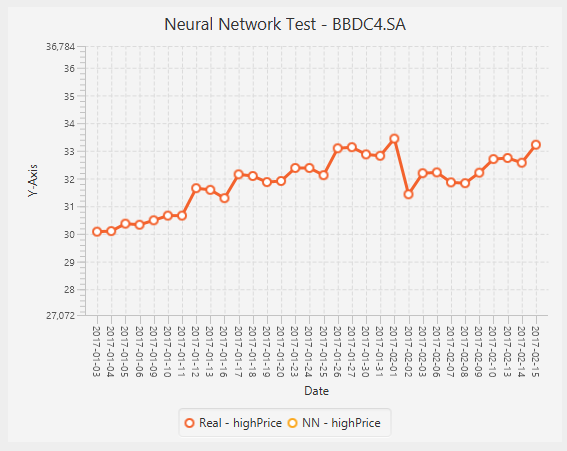
Figura 26. I002 - Treinar Rede Neural



Fonte: Autor (2017)

|  |  |
| --- | --- |
| **CSU005 – Gerar comparativo entre dado calculado pela RNA e o real** | |
| Sumário: | Gerar relatório |
| Ator Primário: | Usuário |
| Ator Secundário: | Sistema BMF |
| **Caso de Uso associado:** CSU002, CSU003. | |
| **Pré-condição:**   1. O Usuário deve ter configurado as variáveis. (CSU002) 2. Deve haver conexão com a internet. | |
| **Fluxo Principal**   1. O Sistema solicita as datas ‘de’ e ‘para’. 2. O Usuário informa os dados. 3. O Sistema envia puxa as informações do Sistema da BMF. 4. O caso de uso termina. | |
| **Fluxo de Exceção (3) – Sem Conexão com a internet**   1. O Sistema informa que não está conectado à internet 2. O caso de uso termina. | |
| **Fluxo de Exceção (3) – Sem dados da BMF**   1. O Sistema informa que não foi possível obter os dados da BMF 2. O caso de uso termina. | |
| **Pós-condições:**  **a.** O Sistema gera o relatório. | |
| **Requisitos:** RF004. | |
| **Interface**: I003 | |

Figura 27. I003 - Gerar Relatório



Fonte: Autor (2017)

# 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 28 é apresentada o diagrama de classes.

Figura 28. 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 29. Diagrama de implantação



Fonte: Autor (2017)

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

Nesta sessão, serão demonstrados todos os diagramas de realização de caso de uso modelados.

Figura 30. Diagrama de classes de projeto – Create NN



Fonte: Autor (2017)

Figura 31. Diagrama de sequência - Create NN



Fonte: Autor (2017)

Figura 32. Diagrama de classes de projeto - Choose NN



Fonte: Autor (2017)

Figura 33. Diagrama de sequência - Choose NN



Fonte: Autor (2017)

Figura 34. Diagrama de classes de projeto - Import BMF Data



Fonte: Autor (2017)

Figura 35. Diagrama de sequência - Import BMF Data



Fonte: Autor (2017)

Figura 36. Diagrama de classes de projeto - Comparative Chart



Fonte: Autor (2017)

Figura 37. Diagrama de sequência - Comparative Chart



Fonte: Autor (2017)

Figura 38. Diagrama de classes de projeto - Train NN



Fonte: Autor (2017)

Figura 39. Diagrama de sequência - Train NN



Fonte: Autor (2017)

## Detalhes de implementação.

Nesta sessão serão detalhados alguns detalhes que mereçam destaque sobre a implementação. Para que o sistema ficasse bem flexível para o usuário foram necessários alguns ajustes no algoritmo para que atendesse as 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 mesmo gerou. Como foi visto 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 foi utilizado essas topologias de modo dinâmico, com a quantidade de neurônios na primeira camada sendo igual à quantidade de atributos multiplicada pelo *date interval* e na última camada somente a quantidade de atributos, sendo um neurônio para cada atributo escolhido pelo usuário.

Figura 40. Topologia da RNA utilizada



Fonte: Autor (2017)

A Figura 40 apresenta um esboço de como a rede utilizada está implementada de uma forma simples e fácil de entender, 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.

Um detalhe interessante mostrado na Figura 40 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 vermelho 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 utilizadas 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 40. Já quando utiliza os próprios dados, ela se comporta como uma rede recorrente.

Figura 41. Código implementação topologia



Fonte: Autor (2017)

Já no início do código da Figura 41 aparece um recurso muito utilizado no projeto, mas que será abordado em outra parte: o Javadoc. 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.

### 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 a função de ativação por neurônio, por causa do modo que foi implementado 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 usando a CPU + GPU.

Para o projeto, nas camadas de entrada e saída foram utilizadas somente a função *ActivationLinear,* explicada no item 2.2.3, para que não haja interferência entre um dia e outro.

Figura 42. Código *Factory* função de ativação



Fonte: Autor (2017)

A Figura 42 apresenta o código utilizado na criação das funções de ativação, utilizado o padrão de projeto *Factory*. Este padrão é responsável por encapsular, simplificar e flexibilizar a criação e objetos concretos. Na Figura 41 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.

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 é 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. Na Figura 24. I001 – Gerenciar Rede Neural,Figura 24 é possível ver esse atributo representado como *Date Interval*. Para implementar isso, foi necessário que os dados de entrada, para criar o relatório, fosse composto de dados puros (somente dados reais), dados mistos (dados reais e dados gerados) e somente dados gerados.

Figura 43. Ajuste *Date Interval*



Fonte: Autor (2017)

Na Figura 43 é possível visualizar a explicação do parágrafo anterior. Cada retângulo representa um conjunto de dados de uma ação em um dia (qual ação é, 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.

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.

Figura 44. Código *mixed data*



Fonte: Autor (2017)

Como visto anteriormente, o primeiro laço mostrado na Figura 44 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 45. Código *calculated data*



Fonte: Autor (2017)

Na Figura 45 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. 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*”.

Figura 46. Código vetor de pessoa



Fonte: Autor (2017)

Na Figura 46 é possível visualizar o que foi explicado anteriormente. Neste caso, como não foi criado um novo objeto do tipo *Person*, o que será guardado no vetor é a referência da memória para um mesmo objeto, logo, ao tentar mostrar na tela, ele mostrará o mesmo texto.

### 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. Foi visto na Figura 45, ao final do código foi chamado um normalizador (*normalizer*) para realizar essa tradução entre a “linguagem” da rede e da ação.

Figura 47. Código *normalize data*



Fonte: Autor (2017)

A Figura 47 mostra 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 8, já abordada anteriormente. Nota-se, também, uma nova forma de fazer um loop no Java 8: o *forEach*, que será explicado posteriormente.

Figura 48. Código *get max and min*



Fonte: Autor (2017)

A Figura 48 mostra como é feito para obter os valores máximos e mínimos. Apesar de não ser a forma mais eficiente de obter esses valores extremos, dado que a complexidade de fazer a ordenação é de O (log (n)), enquanto procurar elemento a elemento pelo maior é linear, foi implementado dessa forma para utilizar os recursos do Java 8, já que o desempenho não é um fator essencial.

Figura 49. Código *getNormalizedValue*



Fonte: Autor (2017)

A Figura 49 mostra como foi implementado a Equação 8. Um detalhe que foi utilizado é para 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.

### Javadoc

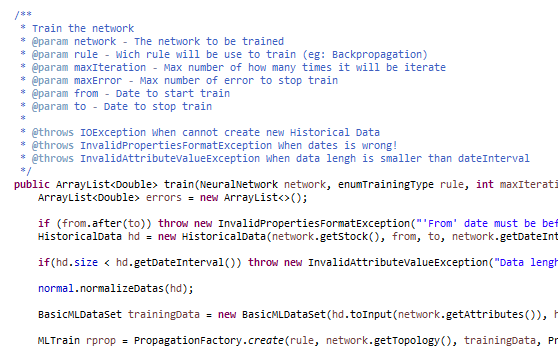
O javadoc é um recurso pouco explorado pelos desenvolvedores, mas muito útil. Ele é responsável por criar uma documentação completa sobre as classes, os métodos e os atributos de um projeto. É com base nele que a documentação mantida pela Oracle sobre o java está disponível aos usuários, como a documentação do javaSE, JDK, entre outros. Pode ser entendido também como uma espécie de contrato, ou manual, entre o usuário do código e o implementador.

Ao contrário do comentário de várias linhas (/\* ... \*/), o javadoc precisa de dois asteriscos no começo. A documentação é escrita em HTML e necessita estar declarada antes de uma classe, parâmetro, construtor ou método. É composta de duas partes: uma descrição e um bloco de *tags*.

Há algumas boas práticas para se escrever uma documentação, como o limite de 80 caracteres por linha, separar os parágrafos com a *tag* <p>, utilizar a terceira pessoa do singuar, entre outras.

As principais tags são: @author, @version, @param, @return, @exception, @see, @since e @deprecated.

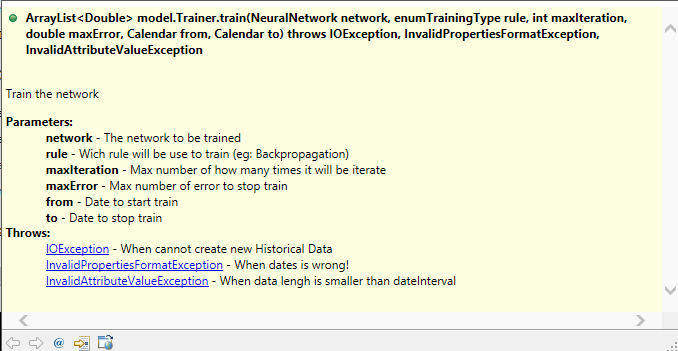
Figura 50 Código exemplo Javadoc



Fonte: Autor (2017)

A Figura 50 mostra um exemplo de uma documentação para o método *train*. É possível observar que a primeira linha explica resumidamente qual é a função deste método, e nas demais linhas, aparece a *tag* “param”, que como o próprio nome diz, explica sobre os parâmetros do método. Nas últimas linhas, a *tag* “throws” mostra quais são os tipos de erro que esse método pode gerar, explicando em que caso que isso ocorre.

Figura 51. Resultado javadoc do método *train*



Fonte: Autor (2017)

A partir do momento em que a documentação está pronta, é possível observar, ao colocar o mouse em cima do método, que em algumas IDE’s mostra a documentação desse trecho do código, em um formato padronizado, como mostra a Figura 51.

#### *Tag* @author

Ela é utilizada somente em classes e interfaces, é obrigatória e serve para indicar quem é o autor. Ela pode vir sozinha, acompanhada de várias outras *tags* @author ou sem nenhuma.

Não é uma *tag* crítica, já que não é incluída ao gerar uma especificação da API, e só é vista quando está com o código fonte.

#### *Tag* @version

Ela também só é utilizada em classes e interfaces, também é obrigatória e indica a partir de qual versão do projeto o recurso a ser documentado foi disponibilizada.

É bastante útil em casos onde o controle de versão é um fator crítico, e indica em que versão o código em questão está. O documento pode contar várias *tags* de @version se fizer sentido.

#### *Tag* @param

A *tag* @param só é utilizada para métodos e construtores e é composta pelo nome do parâmetro (e não o tipo de dado) seguida de uma descrição. É uma boa prática elencar os vários atributos na mesma ordem que está o método, mesmo isso não sendo obrigatório.

Por convenção, os nomes dos parâmetros são escritos em caixa baixa. O tipo de dado é escrito em minúsculo, para indicar um objeto, ao invés de uma classe. A descrição come

#### *Tag* @return

A *tag* @return é utilizada somente em métodos onde o retorno é diferente de *void*, e indica qual é o retorno do método, e deve ser utilizada mesmo se o conteúdo for redundante.

Ter essa *tag* explícita faz com que se encontre o retorno do método mais rapidamente. É uma boa prática escrever qual é o intervalo de valores possíveis do retorno, se fizer sentido. Sempre que possível, encontre um caso não redundante, ou mais específico para usar.

#### *Tag* @exception (@throws)

A *tag @*exception, ou @throws (a partir da Javadoc 1.2), indica qual é o tipo de *exception* que o método pode lançar, bem como uma descrição de quando isso ocorre.

É uma boa prática escrever múltiplas *tags* deste tipo em ordem alfabética. A ideia desta *tag* é indicar ao usuário do método quais são as exceções que ele deve tratar ao utilizar. Além disso, é uma boa prática não documentar exceções *uncheckeds*, ou seja, aqueles erros irrecuperáveis, como por exemplo um *NullPointerException*.

#### *Tag* @see

A *tag* @*see* é utilizada para fazer um link com alguma outra referência. Ela é valida para qualquer tipo de comentário, seja em um campo, um método, um construtor, uma interface, uma classe, um pacote, ou um *overview*.

Há três maneiras de se utilizar esta tag. A primeira é adicionar somente um texto, que indique um livro, ou outra informação que não tenha uma URL, exemplo @*see* “*Artificial Intelligence: a modern approach*”. A segunda é indicar a referência como se fosse em um HTML: @*see*<ahref=”URL#value”>texto</a>, neste caso ao clicar na palavra “texto”, será redirecionado para a *url* “URL” na sessão “*value*”. O último caso, que é o mais comum, é referenciar um item de uma classe : @*see* *String*#*equals*(*Object*) *equalsText*. Neste caso a classe é a “*String*”, no método “*equals*” e o que vai aparecer é o texto “*equalsText*”.

#### *Tag* @since

A *tag* @since é muito útil para projetos que são amplamente utilizados e que sofrem atualizações constantes. Ela mostra desde quando o trecho do código em questão está disponível.

Um exemplo de utilização é nas novas funcionalidades do Java 8, neste caso elas estarão com a *tag* @since 1.8. É usual para as documentações da API do Android, para que o desenvolvedor saiba exatamente a partir de qual versão o recurso funcionará.

#### *Tag* @deprecated

A *tag* @deprecated é utilizada para alertar aos usuários que o código foi descontinuado e o que utilizar no lugar. A primeira parte indica o que ficará no lugar e as subsequentes detalham o motivo.

Em IDE’s modernas, quando percebe o uso de códigos que foram descontinuados, geralmente aparece um aviso para alertar o desenvolvedor.

### Recursos do Java 8

Nesta sessão será apresentado alguns recursos que vieram com o Java 8 e outros que se tornaram mais fáceis de utilizar.

#### Functional Programming

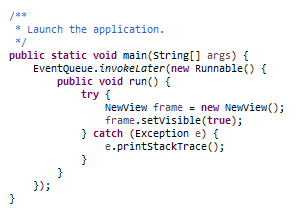
Antes de entrar em detalhes sobre os novos recursos, é preciso entender alguns conceitos e o primeiro deles é sobre a programação funcional. Programação funcional é um paradigma de programação na qual todo o programa é escrito como um conjunto de funções matemáticas, evitando estados ou dados mutáveis.

Para exemplificar de uma maneira mais fácil: se no paradigma de orientação a objeto uma função só pode receber objetos como parâmetro, nesse paradigma é possível aceitar outros métodos como parâmetro, ou até retornar um outro método.

#### Inner Class (Classes anônimas)

As classes anônimas são uma forma de contornar o que foi dito no parágrafo anterior. Elas são classes internas (escritas dentro de outra classe) que são utilizadas para herdar o comportamento, ou implementar uma função, em apenas um local exclusivo.

Figura 52. Exemplo de *inner class*



Fonte: Autor (2017)

A Figura 52 mostra um exemplo de *inner class*, onde é utilizada para criar uma classe somente nesse trecho do código para implementar a interface *Runnable*, isto é, implementar somente o método *run*() da classe *Thread*.

#### Lambda

O lambda é uma sintaxe mais fácil para utilizar *inner class*. 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.

Figura 53. Diferença entre *inner class* e *lambda*

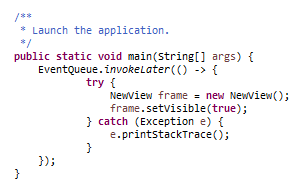


Fonte: Autor (2017)

Na Figura 53 é possível ver o comportamento do clique do botão (neste caso o que adiciona uma linha na tabela de camadas da rede neural, exemplificada na Figura 24. I001 – Gerenciar Rede Neural) implementado de duas formas diferentes.

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.

Figura 54. Implementação do *main* com *lamda*



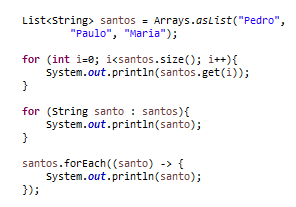
Fonte: Autor (2017)

Na parte da esquerda da função lambda, é passado os parâmetros da função, já na direita é a implementação propriamente dita. A Figura 54 é outra forma de implementar o método mostrado anteriormente, e nota-se que quando o método implementado não possui parâmetros, usa-se o ‘ () ’.

#### *ForEach*

O método *forEach* foi criado na interface *Iterable*<T> para realizar o for dentro de uma coleção, por exemplo. Ao invés de utilizarmos o for tradicional, ou até mesmo o *for* *each* convencional, temos essa outra opção.

Figura 55. Diferenças entre *for*



Fonte: Autor (2017)

Na Figura 55 observa-se a diferença na sintaxe entre os tipos de laço for existente. A diferença maior é somente na sintaxe, pois sobre o desempenho, para esse exemplo praticamente é igual. Existe, claro, uma diferença no desempenho para o uso do método *forEach* (até porque não teriam criado se não tivesse) que é na utilização com paralelismo, mas será abordado posteriormente.

#### *Method* *Reference*

O uso de expressos lambdas foi explicado anteriormente, mas em alguns casos, não o método não faz nada do que chamar outro método existente. Nesses casos, o uso de *method* *reference* permite que faça isso de maneira mais simples e fácil de entender.

Figura 56. Código *lambda* e *method* *reference*



Fonte: Autor (2017)

Na Figura 56 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 57. Código *compareByVolume*



Fonte: Autor (2017)

Se no caso anterior a sintaxe não pareceu tão convidativa, será apresentado um caso que a diferença é mais clara. Por se tratar de um objeto e não um tipo primitivo, para organizar uma lista de dados é preciso dizer à função *sort* como se ordena esse tipo específico. A Figura 57 mostra o método que especifica como se compara um dado, comparando o atributo volume dos dois parâmetros.

Figura 58. Código de ordenação



Fonte: Autor (2017)

Depois de especificado como se ordena um dado, é preciso agora ordená-lo. Existe algumas maneiras de fazer isso, e na Figura 58 é mostrado as três formas que foras explicadas.

#### *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)

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

Figura 59. Código exemplo *stream* – pegando dados



Fonte: Autor (2017)

Primeiro passo é extrair os dados da Petrobrás entre esse período. Na Figura 59 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 60. Exemplo código *stream* - sobrescrita *toString*



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 60. A ideia é bem simples, mostrar todos os valores, sem se preocupar se existe ou não.

Figura 61. Saída do código de exemplo da *stream*



Fonte: Autor (2017)

A Figura 61 apresenta o resultado do código mostrado na Figura 59. É possível notar que somente os atributos que eu declarei 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*.

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.

Figura 62. Código exemplo *stream*



Fonte: Autor (2017)

Com a Figura 62 percebemos 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.

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*.

Figura 63. Ilustração código stream



Fonte: Autor(2017)

A Figura 63 ilustra a sequencia dos passos realizados, explicada nos parágrafos anteriores de uma forma mais simples de entender.

#### *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?

Figura 64. Código *stream* com paralelismo



Fonte: Autor (2017)

De fato, a resposta das duas perguntas acima é não. Como mostra a Figura 64, só foi necessário 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 65. Código exemplo paralelismo



Fonte: Autor (2017)

Para mostrar as diferenças citadas no parágrafo anterior, a Figura 65 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 66. Saída código exemplo paralelismo



Fonte: Autor (2017)

A Figura 66 mostra 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.

É importante deixar claro que o paralelismo implementado utiliza um mecanismo de ­*fork-join­ ­*para o *pool* de *thread* e, neste caso, se uma das instruções for uma tarefa que tenha um tempo de execução longa, ela irá bloquear todas as *threads* do *pool*, consequentemente, irá todas as outras tarefas que utilizam *parallel streams*. Por exemplo, em uma das ações é uma execução rápida na CPU e outra é um cálculo complexo de um número primo com mais de 10.000 dígitos. Enquanto esta última não acabar, a primeira soma não será desbloqueada, portanto, é uma boa prática utilizar para tarefas que o tempo de execução tende a ser similar.

Figura 67. Código updateMaxMin



Fonte: Autor (2017)

A Figura 67 apresenta o mesmo código da Figura 48, 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.

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, pega-se 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 implementa uma *Stream*, ou seja, pode-se encadear quantas operações do tipo intermediárias quiser, desde que faça sentido para a aplicação. Exemplo 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

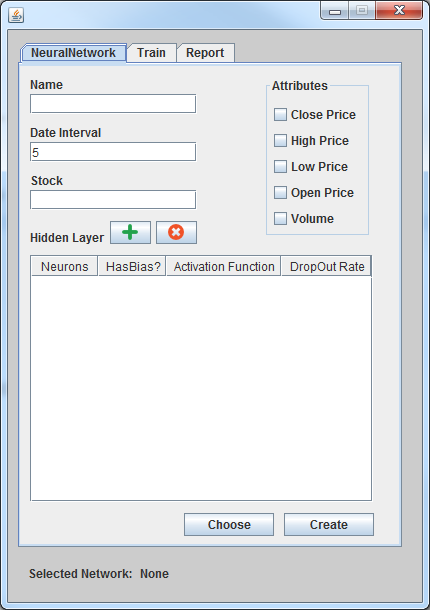
Foi obtido como resultado deste trabalho, um software para criação de RNA com o foco na previsão de dados relevantes para o mercado financeiro.

A seguir será apresentada a sequência de passos que podem ser tomados pelo usuário do sistema.

## Criar RNA

Ao abrir o sistema, a primeira tela será a de escolha da RNA.

Figura 68. Interface 1 - *initial*



Fonte: Autor (2017)

É possível perceber que na Figura 68 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.

Figura 69. Interface 1 - *error*



Fonte: Autor (2017)

Na Figura 69 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 70. Interface 1 – *success*



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 mostra a Figura 70. 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 71. Interface 1 - *add hidden layer*



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 mostra a Figura 71.

Figura 72. Interface 1 - *activation functions*



Fonte: Autor (2017)

Na Figura 72 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 apagada a linha selecionada, e se não tiver nenhuma selecionada, apaga a última linha.

Figura 73. Interface 1 – *overwrite*



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. É importante notar que, apesar de um dos requisitos não funcionais especificar 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 74. Interface 1 – *don’t overwrite*



Fonte: Autor (2017)

Caso o usuário clique no botão de negação, “Não” no caso, a mensagem deixará de ser exibida e aparecerá uma outra mensagem, como mostra a Figura 72. 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.

Figura 75. Interface 1 *- NN don't exists*



Fonte: Autor (2017)

A Figura 73 apresenta 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 76. Interface 1 - *NN loaded*



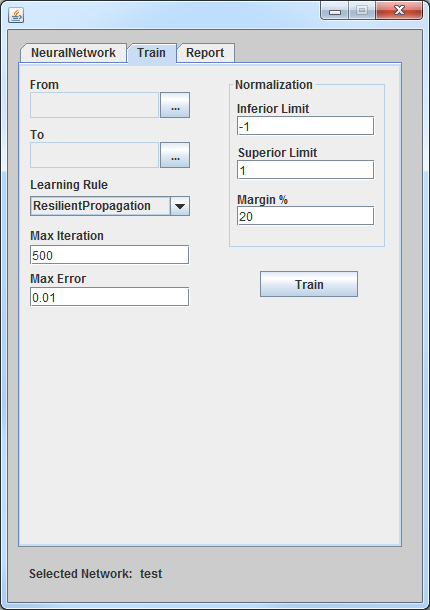
Fonte: Autor (2017)

A Figura 74 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.

## 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*”.

Figura 77. Interface 2 – initial



Fonte: Autor (2017)

A Figura 77 mostra 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.

Figura 78. Interface 2 - *null dates*



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 78.

Figura 79. Interface 2 - *null NN*



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 79, solicitando ao usuário escolher uma RNA antes de prosseguir.

Figura 80. Interface 2 - *Learning Rules*



Fonte: Autor (2017)

A Figura 80 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 81. Interface 2 – *dates*



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 mostra a Figura 81. 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.

## Gerar Relatório

# Considerações Finais

# Bibliografia

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.

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**. 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.

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

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

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

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.

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.

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.

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.

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**. 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**. 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.