Silk: Uma biblioteca Java para uso de Machine Learning

Flavio Daniel Tuyarot Barci[1], Sandro Roberto Fernandes[2]

[1]Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais – Campus Juiz de Fora

[2]Núcleo de Informática - Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais – Campus Juiz de Fora

flavio.barci@gmail.com, sandro.fernandes@ifsudestemg.edu.br

Abstract. *The purpose of this article is to present the Silk library, which implements Machine Learning algorithms in Java. This project is one of the bases for the development of the Weave tool, an image classification program based on Texture Descriptors. To develop this project, we used the library development methodology as well as tools like Git, Maven and GitHub. The neural network was validated and was able to simulate the XOR function with 97.44% accuracy with 2 neurons in the hidden layer.*

Resumo. *O objetivo deste artigo é apresentar a biblioteca Silk, que implementa algoritmos de Machine Learning em Java. Esse projeto é uma das bases para o desenvolvimento da ferramenta Weave, um programa de classificação de imagens baseadas em Descritores de Textura. Para desenvolver esse projeto, utilizou-se da metodologia de desenvolvimento de bibliotecas assim como as ferramentas Git, Maven e GitHub. A rede neural foi validada e conseguiu simular a função XOR com 97.44% de precisão com 2 neurônios na camada escondida.*

1. Introdução

Machine Learning (ML) é o estudo científico de algoritmos e modelos estatísticos que computadores podem utilizar para realizar uma tarefa sem instruções explícitas do que deve ser feito. ML é objeto de estudo e pesquisa e uma de suas aplicações é voltada para o processamento de dados com o intuito de obter informações que auxiliem na tomada de decisões [Rashid, T. 2016].

A motivação para o desenvolvimento deu-se a partir da ferramenta *Weave,* que atualmente ainda está em fases iniciais de desenvolvimento. Durante o levantamento de requisitos desse *software*, percebeu-se a necessidade de uma biblioteca em Java que implementasse os algoritmos de ML para a classificação de Descritores de Textura junto com alguns requisitos específicos para o funcionamento total do *Weave*. A biblioteca encontrada e testada apresentava problemas de

usabilidade para os objetivos definidos e também impediam que o *Weave* fosse registrado e patenteado. Com a falta de uma biblioteca que atendesse as especificidades necessárias, iniciou-se o desenvolvimento da biblioteca *Silk*. O objetivo principal deste trabalho visou desenvolver uma biblioteca em Java que facilite o uso de uma técnica de Machine Learning chamada Rede Neural Artificial (RNA) na produção de software.

Uma Rede Neural Artificial (RNA) é um algoritmo computacional que, uma vez treinada, espera-se que seus resultados se aproximem da função objeto. Uma RNA contém os seguintes elementos básicos: uma entrada, neurônios e uma saída. Esses neurônios podem ser organizados em camadas, onde a saída de uma camada é a entrada da outra camada, e esse processo é denominado *feedforward*. A saída de cada neurônio pode ser ele mesmo, ou transformado por uma função de ativação, e esta função serve para ajudar a aproximar as equações ao resultado. Antes de passar pela função de ativação, é adicionado um *bias* (peso) que ajuda a aproximar a função [Rojas, R. 2013].

Dado isso, uma RNA pode ser utilizada para diversas aplicações onde se sabem as entradas e as saídas, mas não a função. Dentro do contexto do *Weave*, a rede neural seria utilizada para classificar imagens a partir dos valores numéricos encontrados pelos seus Descritores de Textura, possibilitando automações de diversos processos antes possíveis apenas por ação humana, como por exemplo detecção de tumores malignos em imagens de mamografias.

2. Revisão Sistemática da Literatura

Para o desenvolvimento desta, três questões primordiais foram foco de pesquisa durante a revisão sistemática:

1) É possível ter uma biblioteca ou similar para trabalhar com RNA em alguma linguagem de programação / IDE?

2) Em caso afirmativo, é possível criar uma biblioteca para a linguagem de programação Java?

3) Como seriam as "boas práticas" para criar uma biblioteca na Linguagem Java?

Foram elaboradas três *strings* de busca, na qual foi gerada uma lista com os artigos de maior relevância para as questões a partir da busca nos Periódicos da Capes. A primeira *string* de busca foi elaborada da seguinte forma:

(library OR libraries) AND (artificial OR neural OR neuron OR network OR networks) AND (machine OR learning OR learn) AND (artificial OR intelligence)

A segunda *string* de busca adicionou o termo *java,* além dos demais termos:

(java) AND (library OR libraries) AND (artificial OR neural OR neuron OR network OR networks) AND (machine OR learning OR learn) AND (artificial OR intelligence)

E finalmente a terceira *string* de busca fica da seguinte forma:

(good OR practices OR practice) AND (library) AND (java) AND (project patterns) AND (methodology)

As *strings* foram utilizadas no portal Periódicos Capes e retornaram, após os devidos refinamentos, 10 artigos para a primeira *string*, 2 artigos para a segunda *string*, 62 artigos para a terceira *string*.

Após os materiais passarem pelo processo de seleção dos estudos, foi feita uma lista dos artigos de relevância para a pesquisa. De cada material pertencente a essa lista foram extraídas o titulo do material, o autor do materal e o resumo do material.

3. Metodologia

O levantamento de requisitos para o *Silk* iniciou a partir das necessidades do *Weave*. Para não limitar o uso da biblioteca apenas a uma ferramenta, optou-se por definir requisitos que permitissem seu uso em outros softwares.

No contexto de Machine Learning, mais especificamente nas Redes Neurais Artificias, existem diversos algoritmos para diversos casos de uso. O primeiro algoritmo idealizado na área é o de *feedforward,* que, como citado anteriormente, é o processo de alimentar os neurônios da camada seguinte com as informações da camada anterior. Porém existem outras formas de execução dessa rede. E, além da execução, também estão os algoritmos para treinamento, das quais a que comumente é usada chama-se *backpropagation* [Rojas, R. 2013].

Durante a revisão sistemática, evidenciou-se que não haviam bibliotecas em Java que atendiam as necessidades do projeto *Weave*. A partir disso, elaborou-se um plano de desenvolvimento, visando implementar os algoritmos de *Machine Learning* necessários para o *Weave*, junto com boas práticas de programação para a utilização da biblioteca em qualquer outro *software*.

Como existem diversos modos de se elaborar uma rede neural, foi avaliado qual seria a necessidade com relação a complexidade a primeiro momento e decidiu-se que o primeiro algoritmo a ser implementado pelo *Silk* para uso no *Weave* seria uma Rede Neural Artificial *Feedforward* com *Backpropagation* para seu treinamento [Rojas, R. 2013] . Foi também decidido que o *Stochastic Gradient Descent* seria utilizado como método para otimização do aprendizado da rede. Tomou-se essa decisão pois o algoritmo atendia *a priori* as necessidades do *Weave* e é o que oferece menor complexidade de implementação quando comparado a outros tipos de redes neurais.

Para o fluxo de desenvolvimento do *Silk*, optou-se por seguir um modelo cíclico, onde cada parte do software seria planejado, desenvolvido e validado antes de ser consolidado dentro da versão. Este fluxo de trabalho garante qualidade do software e facilita ajustes de acordo com as necessidades, que podem mudar durante o desenvolvimento além de facilitar também experimentos [Yurkovich, T et al. 2017].

Para melhor entender este processo, podem-se visualizar todas as etapas do fluxo na Figura 01 abaixo:

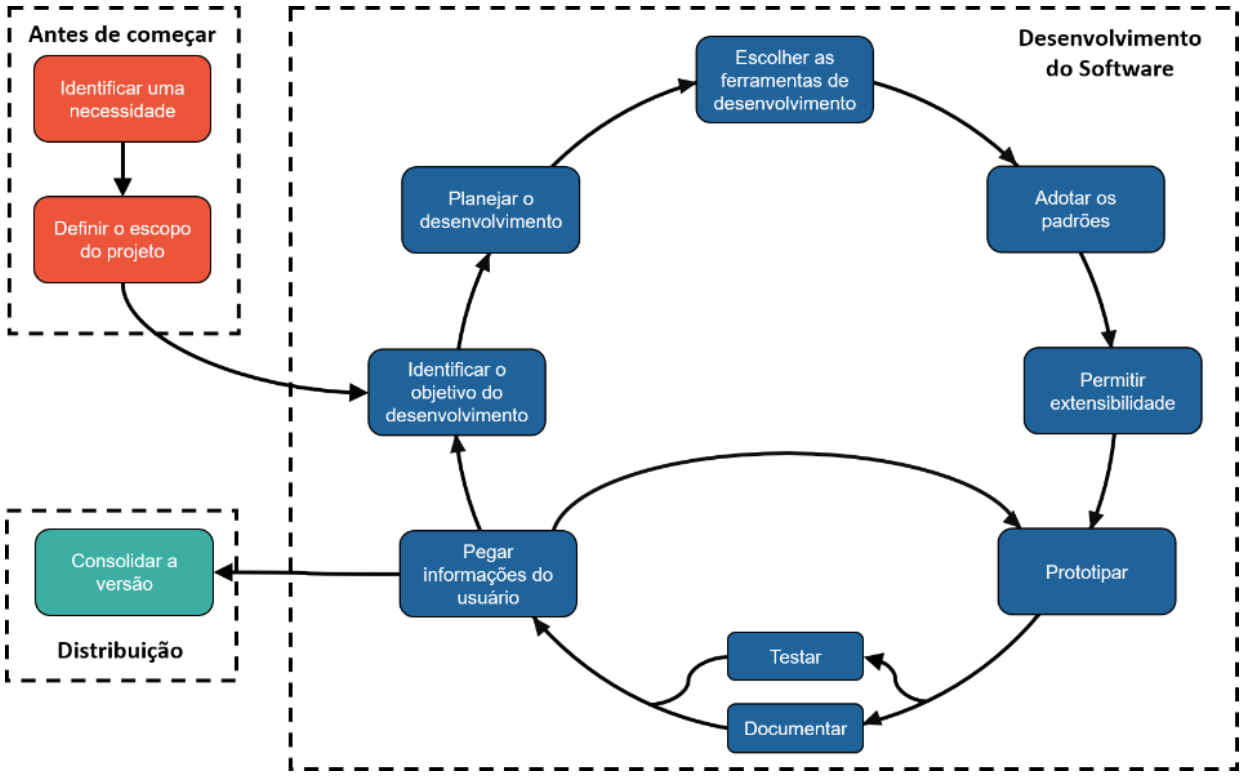

Figura 1 - Fluxo de desenvolvimento. Fonte: O Autor, adaptado de *A Padawan Programmer's Guide to Developing Software Libraries*

Para exemplificar como este processo foi crucial para o desenvolvimento, houve uma iteração em que o escopo era o uso da *Classe BigDecimal* para armazenar o valor das variáveis no lugar da Classe *Double* do *Java*. Depois de vários testes na etapa final do ciclo, chegou-se à conclusão de que o resultado foi uma perda da performance da rede numa magnitude de 100 vezes, sem melhora significativa nos resultados que estavam sendo obtidos por ela. Então no final do ciclo, optou-se por um *rollback* para a versão anterior. O fluxo de desenvolvimento então, facilitou a detecção de problemas e correção deles, de uma maneira eficiente e pronta.

4. Ferramentas

As ferramentas que fizeram parte do desenvolvimento do *Silk*, foram escolhidas visando garantir a qualidade do código, e agilizar o processo de desenvolvimento. A seguir listamos as ferramentas utilizadas.

Utilizou-se a biblioteca SLF4J, com o Logback sendo o *backend*, como Framework de Logs. Esta ferramenta facilita o gerenciamento de logs e o torna independente do código, podendo ser configurado o formato e destino sem necessidade de mudança na estrutura do projeto. O Framework também implementa níveis de severidade nos logs e domínio de onde o log foi criado. A escolha desta biblioteca deu-se por ser apenas um *Facade,* isto é, se for necessario mudar a biblioteca de *backend,* não há alteração no código. Atualmente o Silk utiliza a versão *1.7.25* do SLF4J e *1.2.3* do Logback. Um exemplo do arquivo de configuração pode ser visto na figura 2 abaixo:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
        <layout class="ch.qos.logback.classic.PatternLayout">
            <Pattern>
                [%date] %highlight(%level) %thread - %msg%n
            </Pattern>
        </layout>
    </appender>
    <logger name="com.funnel" level="debug" additivity="false">
        <appender-ref ref="CONSOLE"/>
    </logger>
    <root level="error">
        <appender-ref ref="CONSOLE"/>
    </root>
</configuration>
```

Figura 2 – Configuração do LSF4J com Logback, Font: O Autor

Um *linter* é uma ferramenta que avalia a formatação geral do código. É um modo de obrigar que o código esteja dentro dos padrões definidos antes mesmo de poder compilar. O *linter* usado no *Silk* é o *Checkstyle*, na versão *8.37,* pois é o que fornece a maior gama de suporte à ambientes de desenvolvimento. Na figura 3 pode-se ver o resultado ao executar a ferramenta:

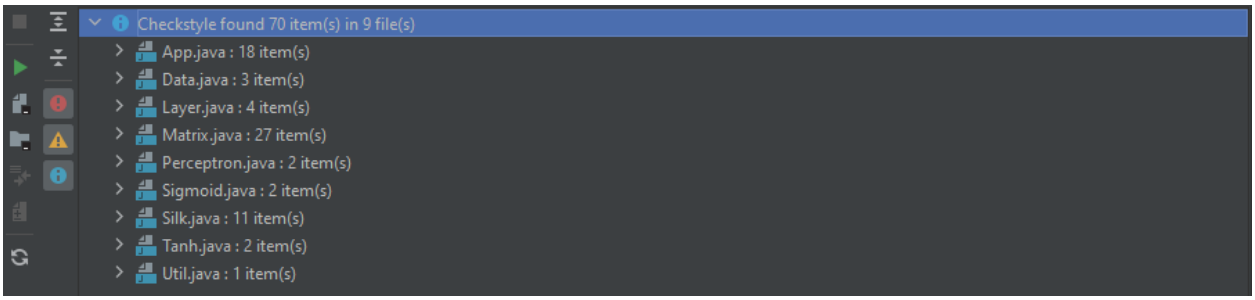

Figura 3 – Tela de resultado da execução do Checkstyle. Fonte: O Autor.

O Maven é uma ferramenta que automatiza a compilação e gerenciamento de dependências, mais comumente usados em projetos *Java*. Ele funciona com base em fluxos de operações, nas quais podem ser adicionado execução de testes e de *linter* antes da execução da compilação. Esta ferramenta foi escolhida por proporcionar todas essas facilidades em um arquivo único de configuração simples. Atualmente o *Silk* utiliza a versão *4.0.0* do Maven. Abaixo mostra-e um exemplo do arquivo de declaração de execução do Maven, na figura 3:

```
1   <?xml version="1.0" encoding="UTF-8"?>
2
3   <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4            xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
5       <modelVersion>4.0.0</modelVersion>
6       <groupId>com.funnel.silk</groupId>
7       <artifactId>silk</artifactId>
8       <version>1.0-SNAPSHOT</version>
9       <properties...>
12      <dependencies...>
41      <build...>
109     <distributionManagement...>
116 </project>
```

Figura 4 – Arquivo pom.xml, com as configurações do *Maven.* Fonte: O Autor.

É importante manter controle sobre as mudanças feitas no código para ter a certeza de que versão e mudanças estão sendo executadas. Para isso é usado uma ferramenta de controle de versões chamada Git. Com ela, pode-se ver o histórico de modificações e voltar atrás em qualquer modificação feita. É uma ferramenta *open source* e descentralizada. A versão utilizada no *Silk* é a *2.28*. Pode-se ver a árvore de modificações feitas no código fonte na figura 5, abaixo:

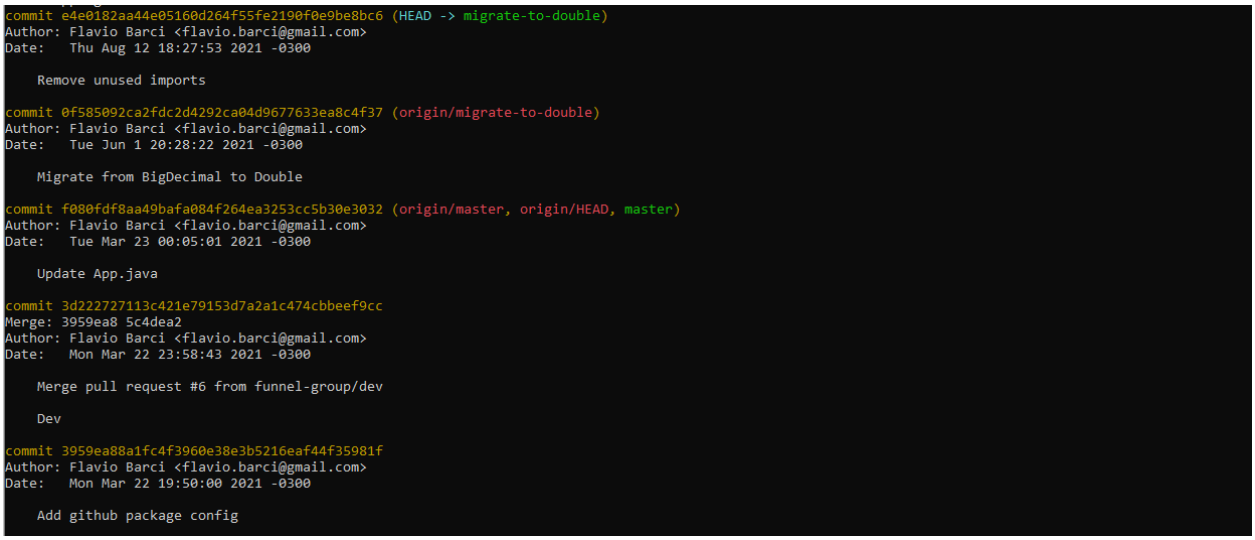

Figura 5 – Árvore de modificações do código do *Silk*. Fonte: O Autor

Utilizou-se o GitHub para a hospedagem do código fonte. O GitHub é uma plataforma que implementa um servidor git onde podemos manter uma versão remota do repositório, onde fica a versão mais estável fora de desenvolvimento. A plataforma fornece recursos para compilação na nuvem, e foi configurado para toda vez que houver uma versão nova, sejam gerados os respectivos binários. Isto facilita a consolidação e distribuição da versão. A tela do projeto no GitHub pode ser vista na figura 6:

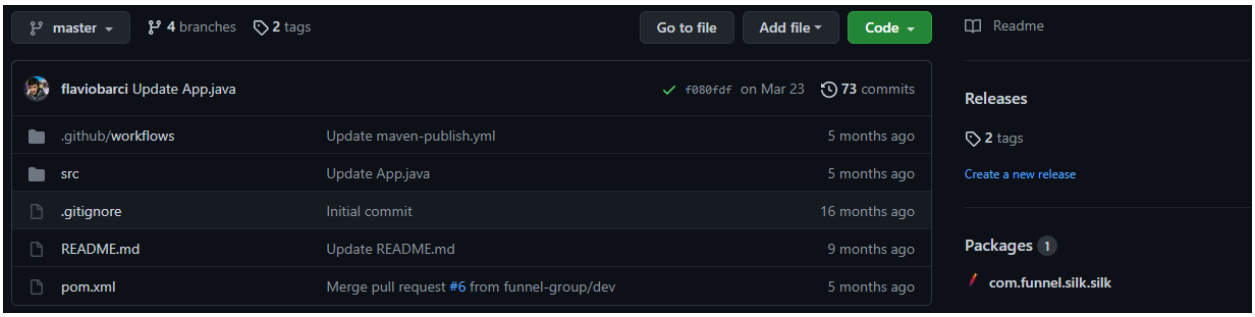

Figura 6 – Tela principal do projeto no GitHub. Fonte: O Autor

5. Biblioteca *Silk*

A biblioteca *Silk* foi projetada de forma que se permite configurar o número de camadas, o número de neurônios em cada camada e as funções de ativação de cada camada. Para tal, foram criadas, além da Classe *Silk*, as seguintes classes: Layer, Activation, Matrix, Data e Progress.

A classe Layer representa as camadas da rede neural. Quando se instancia um objeto dessa classe, pode-se configurar qual será a função de ativação dessa camada e quantos neurônios estão

atuando dentro dela. Nesta, os pesos e os *biases* são configurados como uma randomização entre -1 e 1. Esta classe também guarda os valores dos resultados antes e depois de passarem pela função de ativação. Um exemplo de seu código pode ser visto na Figura 7 abaixo:

```
6   public class Layer implements Serializable {
7     private final int neurons;
8     private Matrix weights;
9     private Matrix biases;
10    private Matrix output;
11    private Matrix rawOutput;
12    private Activation function;
13    private final Double rate = 3d;
14
15    public Layer(Activation function, int neurons) {
16      this.function = function;
17      this.neurons = neurons;
18    }
```

Figura 7 - Classe Layer. Fonte: O Autor

A classe Activation é uma interface. Isso permite aos programadores implementarem suas próprias classes de ativação a partir dela. A classe Activation é usada no processo de *feedfoward* dentro da classe *Silk*. Além de permitirmos a implementação, a biblioteca já conta com algumas funções já implementadas, e com possibilidade de novas funções entrarem de acordo com os ciclos de desenvolvimento. As oferecidas pela biblioteca hoje são as seguintes:

- *Rectified Linear Unit*
- *Leaky Rectified Linear Unit*
- *Linear*
- *Sigmoid*
- *Step*
- *Tanh*

Um exemplo do código da Classe Activation pode ser visto na Figura 8 abaixo:

```java
package com.funnel.silk.activation;

public interface Activation {
  Double run(Double input);

  Double derivative(Double input);
}
```

Figura 8 - Classe Activation. Fonte: O Autor

A classe Matrix contém as operações relacionadas a matrizes e lida com a estrutura de dados nela. É capaz de adicionar e remover linhas, colunas e elementos. Realiza multiplicação matricial, multiplicação escalar. Também implementa adição matricial, subtração matricial, produto *haddamard* e transposição. Possuí métodos para aplicação de funções em cada elemento, e potência em cada elemento. Esta possui diversos construtores para facilitar a usabilidade nas classes principais. Uma amostra de seu código pode ser vista na Figura 9 abaixo:

```java
public class Matrix implements Serializable {
  private final Logger logger = LoggerFactory.getLogger(Matrix.class);
  private int rows;
  private int columns;
  private Double[][] data;

  public Matrix(Double[][] data) {...}

  public Matrix(Matrix m) {...}

  public Matrix(Double[] inputs) {...}

  public Matrix(Double[] inputs, Double bias) {...}

  public Matrix(int rows, int columns) {...}

  public Matrix(int rows, int columns, Double element) {...}
```

Figura 9 - Classe Matrix. Fonte: O Autor

Dependendo do tamanho da rede e do problema, pode-se esperar por muito tempo esperando o treinamento da rede ser finalizado. Isso gera um problema em que não se tem certeza em que ponto está o processamento durante a execução. Para solucionar este problema, foi desenvolvida a classe Progress.

A classe Progress é uma interface criada com o objetivo de disponibilizar uma maneira de verificar o progresso do treinamento da rede neural. Apenas é necessário implementar a interface e passá-la como argumento, que ela vai ser chamada a cada iteração de cálculos.

Um exemplo do código da classe Progress pode ser vista na Figura 10 abaixo:

```
1  package com.funnel.silk;
2
3  public interface Progress {
4    void increase(double current, int total);
5  }
```

Figura 10 - Class Progress. Fonte: O Autor

A classe *Silk* é a classe principal classe da biblioteca e reúne todo o fluxo de uso da rede neural. Cada instância desta classe é um contexto de rede neural, com suas próprias configurações e treinamento definidos.

Ao instanciar esta classe, é informado o número de entradas de dados que a rede neural irá suportar configurando assim a primeira camada da rede. Uma vez isto feito, deve-se ir adicionando as camadas da rede neural na ordem que se deseja, sendo que a ordem em que são adicionadas irá ditar quais serão as ordens de entrada e saída das respectivas camadas. Após essa ação não é possível alterar a rede. A classe *Silk* conta com um método para a adição de camadas que aceita instâncias do objeto Layer para este objetivo.

Uma das funções necessárias, determinada pelo levantamento de requisitos e necessário para o *Weave*, é a função de salvar a rede em um arquivo para uso posterior. Essa função possui um método que armazena todos os pesos e *biases* da rede neural em um arquivo. Isto ajuda na consolidação da usabilidade da biblioteca, eliminando a necessidade de ter que treinar a rede toda vez que utilizar o *Weave*. Essa função é particularmente útil quando se treina uma rede por um longo período de tempo, e não se deseja perder o resultado.

Um exemplo do código da classe *Silk* pode ser visto na Figura 11 abaixo:

```java
package com.funnel.silk;

import ...

public class Silk implements Serializable {
  private final Logger logger = LoggerFactory.getLogger(Silk.class);
  private final ArrayList<Layer> layers;
  private Matrix cost;
  private final int numberOfInputs;
  private Matrix inputs;

  /**
   * New neural network instance.
   *
   * @param inputs Number of inputs accepted by the neural network
   */
  public Silk(int inputs) {
    this.numberOfInputs = inputs;
    this.layers = new ArrayList<>();
  }
```

Figura 11 - Classe Silk Fonte: O Autor

6. Uso da Biblioteca

Para exemplificar o uso da biblioteca *Silk*, utilizamos uma rede neural simples, com dois neurônios na camada escondida e um neurônio de saída. O modelo deste uso pode ser visto na Figura 12 abaixo:

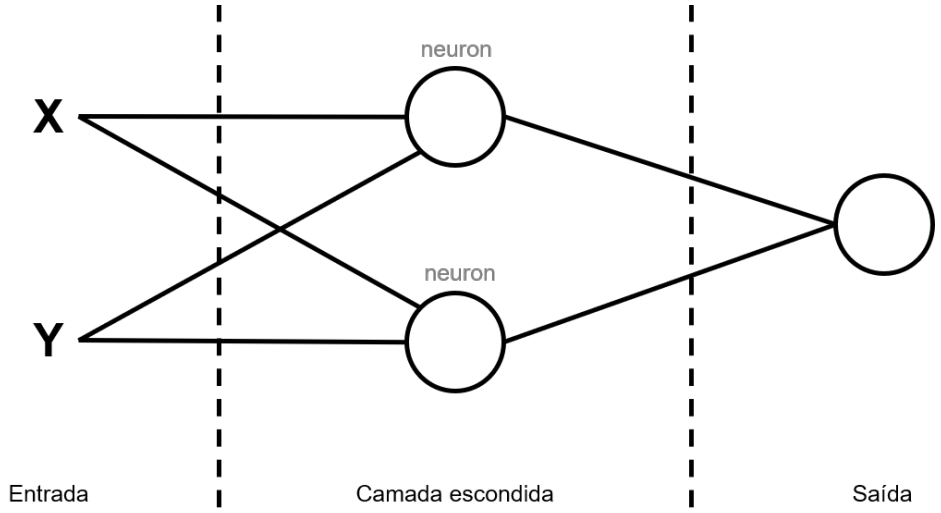

Figura 12 - Modelo de rede. Fonte: O Autor

O código para criar uma rede equivalente à modelada na Figura 12 com a biblioteca Silk pode ser vista na Figura 13 logo abaixo:

```
Silk nn = new Silk( inputs: 2);
nn.add(new Layer(new Sigmoid(), neurons: 2));
nn.add(new Layer(new Sigmoid(), neurons: 1));
```

Figura 13 – Criação da rede. Fonte: O Autor

Após a criação da rede, a próxima etapa é carregar os dados que serão utilizados. O Silk conta com uma classe chamada Data, que se encarrega de carregar os dados. Esta classe possui suporte para dois tipos de entrada de dados. Uma entrada por meio do uso de uma *String*, com seus valores separados por vírgulas e uma entrada utilizando arquivos no padrão CSV. Na Figura 14 abaixo há o uso de uma entrada de dados utilizando um arquivo no formato CSV.

```
Data XOR =
    new Data(
        new File( pathname: "path/to/file.csv"),
        batch: 4);
```

Figura 14 - Carregamento dos dados. Fonte: O Autor

O construtor da classe Data possui a opção de fornecer o tamanho do *batch* que será utilizado para a otimização por meio do *Stochastic Gradient Descent*. Uma vez que se possuí um objeto pronto da classe Data, agora pode-se utilizá-la no método *train* da classe Silk, como visto na Figura 15 abaixo.

```
19          nn.train(XOR, epochs: 10000);
```

Figura 15 - Treinamento da rede. Fonte: O Autor

O método *train* recebe como argumentos um objeto da classe Data, e o número de *epochs* que se deseja treinar a rede, sendo uma *epoch* uma passagem completa por todo o conjunto de dados de treinamento.

7. Software de Exemplo

Para a validação da rede neural, escolheu-se testar a função XOR. Sua escolha foi devido ser um problema simples, poucos dados de entrada e não é linearmente separável [Rashid, T. 2016]. Este teste trata-se de treinar uma RNA para imitar o comportamento da função XOR, que pode ser observada na tabela verdade a seguir (Tabela 1):

Tabela 1 - Tabela verdade da função XOR.

A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

Dado este comportamento, não é possível fazer uma função linear que imite o comportamento definido na tabela, o que pode ser visto no gráfico a seguir na figura 16:

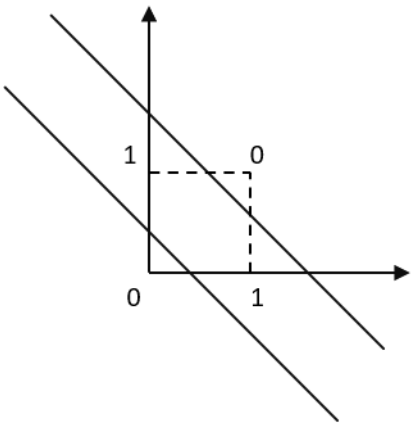

Figura 26 - Gráfico XOR. Fonte: O Autor

O resultado do problema do XOR valida a rede neural exatamente por ser um problema não linearmente separável. Para tal teste, foi desenvolvido um software de exemplo que utiliza a biblioteca *Silk* e carrega a tabela verdade no formato CSV. O treinamento utilizou o método *predict*, da classe *Silk*. Isto pode ser visto na Figura 17 abaixo:

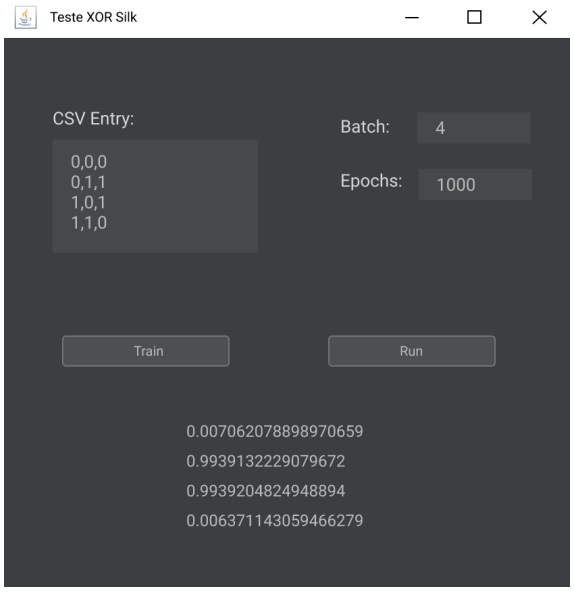

Figura 17 – Interface do aplicativo para teste da Biblioteca Silk. Fonte: O Autor

Após o treinamento da RNA observou-se que foi possível obter resultados satisfatórios com o uso da Biblioteca *Silk*. Estes resultados são discutidos a seguir.

8. Resultados e Discussões

O desenvolvimento do *Silk* baseou-se em boas práticas de orientação a objetos e padrões de projeto. Para avaliar a qualidade do código em si, optou-se por utilizar das métricas de *Chidamber Kemerer* [Shyam R. Chidamber, Chris F. Kemerer, 1993] , que são baseadas em uma sólida teoria de medição dos valores de baixo acoplamento e alta coesão, como visto em *Design Patterns* [Erich Gamma et al, 1993].

Essas métricas são a WMC (número de métodos dentro de dada classe), DIT (Profundidade da árvore de heranças), NOC (Número de classes filhas), CBO (Acoplamento entre objetos), RFC (Respostas para uma classe) e LCOM (Falta de coesão dos métodos).

Atualmente o repositório do Silk conta com 76 *commits*, 1562 linhas e 17 classes. Assim, os seguintes valores de métricas de *Chidamber Kemerer* gerados para o código do Silk podem ser vistos na Figura 18 abaixo:

class	CBO	DIT	LCOM	NOC	RFC	WMC
com.funnel.silk.activation.Activation					2	
com.funnel.silk.activation.Function	6		3		6	1
com.funnel.silk.activation.LeakyRectifiedLinearUnit	2	1	2	0	3	4
com.funnel.silk.activation.Linear	2	1	2	0	2	2
com.funnel.silk.activation.RectifiedLinearUnit	2	1	2	0	4	3
com.funnel.silk.activation.Sigmoid	4	1	1	0	3	2
com.funnel.silk.activation.Step	2	1	2	0	3	3
com.funnel.silk.activation.Tanh	2	1	1	0	4	2
com.funnel.silk.App	5	1	1	0	12	1
com.funnel.silk.Data	4	1	1	0	19	19
com.funnel.silk.Layer	4	1	2	0	21	17
com.funnel.silk.Matrix	7	1	1	0	42	92
com.funnel.silk.Perceptron	1	1	1	0	5	8
com.funnel.silk.Progress					1	
com.funnel.silk.Silk	6	1	3	0	65	38
com.funnel.silk.Util	1	1	1	0	2	1
Total						193
Average	3.43	1.00	1.64	0.00	12.12	13.79

Figura 18 - Métricas Chidamber Kemerer

De acordo com *Chidamber & Kemerer object-oriented metrics suite* [Aivosto Oy, 2013], para um código ser considerado de alta qualidade, os números devem se aproximar dos vistos na tabela 2 abaixo:

Tabela 2 – Parâmetros da *Chidamber & Kemerer object-oriented metrics suite*.

CBO	1.25
LCOM	78.34
RFC	43.84
NOC	0.35
DIT	0.97
WMC	11.10

Atualmente o repositório do Silk conta com 76 *commits*, 1562 linhas e 13 classes.

De acordo com *Encog 3.3: Quick Start Guide* [Jeff Heaton 2014] os valores vistos na Figura 19 são os resultados da biblioteca Encog para o teste feito com XOR, o mesmo teste que foi executado para o Silk, e os resultados podem ser encontrados na Figura 17.

```
Neural Network Results:
0.0,0.0, actual=0.037434516460193114,ideal=0.0
1.0,0.0, actual=0.8642455025347225,ideal=1.0
0.0,1.0, actual=0.8950073477748369,ideal=1.0
1.0,1.0, actual=0.0844306876871185,ideal=0.0
BUILD SUCCESSFUL
Total time: 4.401 secs
[jheaton@jeffdev encog-java-examples]$
```

Figura 19 – Resultados do teste XOR do encog. Fonte: Encog 3.3: Quick Start Guide.

Para fins de comparação, calculando a precisão total obtida pela rede neural criada pela Silk, obtemos 97.44% de precisão. Enquanto a precisão vista na documentação do Encog é de 63.73%.

9. Trabalhos Futuros

Uma das próximas etapas a serem desenvolvidas é a adoção de paralelismo nas operações da rede. Atualmente a rede do *Silk* conta com muitas operações a serem executadas, porém todas sequencialmente. A maioria destas operações não dependem uma das outras, possibilitando com que sejam executadas em paralelo. Outra possibilidade a ser explorada é o envio destes cálculos para a unidade de processamento gráfico (GPU) onde seriam executadas com maior velocidade, criando a possibilidade de uso de redes maiores com menor tempo de treinamento.

Espera-se continuar adicionando novas funções de ativação que a biblioteca disponibiliza pronta para o uso.

Ademais, espera-se expandir mais as possibilidades de configuração da rede neural, fazendo com o *learning rate* seja configurável, assim como o espectro de randomização dos pesos e dos *biases.* Deixando isso a cargo do usuário, permite resultados mais variados e precisos de acordo com o problema sendo estudado.

Finalmente, pretende-se explorar a possibilidade de implementar outros algoritmos de RNA para poder abranger casos de uso mais diversos, como por exemplo redes neurais recorrentes ou redes neurais convolucionais. Além disso, investigar sobre a adição de algoritmos de otimização de aprendizado das redes, e que as mesmas sejam configuráveis para escolha do uso.

10. Conclusões

O *Silk* já está sendo integrado no *Weave*, e está sendo de extrema importância para o seu desenvolvimento, visto que atende os requisitos levantados para o *Weave*. O Silk ainda está em fase de testes, mas já apresenta resultados promissores para o que se propõe. Estes resultados, quando comparados a resultados da aproximação de XOR do ENCOG, se mostram bem promissores.

Além disso, como foi decidido implementar uma solução genérica, esta ferramenta poderá ser utilizada em diversos projetos.

Referências

Gamma, E., Helm, R., Johnson, R., Vlissides, J., & Patterns, D. (1995). Elements of reusable object-oriented software (Vol. 99). Reading, Massachusetts: Addison-Wesley.

Aivosto Oy, (2013) Chidamber & Kemerer object-oriented metrics suite, http://people.scs.carleton.ca/~jeanpier/sharedF14/T1/extra%20stuff/about%20metrics/Chidamber%20&%20Kemerer%20object-oriented%20metrics%20suite.pdf

Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. IEEE Transactions on software engineering, 20(6), 476-493.

Heaton, J. (2015). Encog: library of interchangeable machine learning models for Java and C#. J. Mach. Learn. Res., 16, 1243-1247.

Heaton, J. (2014). Encog 3.3: Quick Start Guide.

Yurkovich, J. T., Yurkovich, B. J., Dräger, A., Palsson, B. O., & King, Z. A. (2017). A padawan programmer's guide to developing software libraries. Cell Systems, 5(5), 431-437.

Rojas, R. (2013). Neural networks: a systematic introduction. Springer Science & Business Media.

Rashid, T. (2016). Make your own neural network (p. 222). CreateSpace Independent Publishing Platform.