Uso de redes neurais para reconhecimento de padrões

Conceitos de redes neurais e inteligência artificial aplicados ao desenvolvimento de aplicações para identificação de padrões

Guilherme de Cleva Farto
guilherme.farto@gmail.com - http://lattes.cnpq.br/4645110528610249
Mestrando em Computação Aplicada (Engenharia de Software) pela Universidade Tecnológica Federal do Paraná (UTFPR), campus Cornélio Procópio, especialista em Engenharia de Componentes Java e graduado em Bacharelado em Ciência da Computação. Atualmente é analista de soluções Java no segmento Agroindústria da TOTVS e professor de cursos de graduação e pós-graduação. Autor de palestras e treinamentos nos temas de Java, Google Android, Metaprogramação e Programação Reflexiva, Cloud Computing e Computação Física.

Gabriel Souza da Silva
gabriel-s-s-@hotmail.com
Graduando em Bacharelado em Ciência da Computação pela Fundação Educacional do Município de Assis (FEMA), tem desenvolvido pesquisa na área de redes neurais e identificação de padrões.

[Porque esse artigo é útil:]
Atualmente, na área da Inteligência Artificial (IA), as Redes Neurais Artificiais (RNAs) estão em evidência, pois apresentam com eficiência uma cópia matemática do cérebro, que é dado como a fonte de inteligência em um ser vivo. Estas redes são capazes de aprender sobre um determinado ambiente para, subsequentemente, tomar decisões a respeito do mesmo, exatamente como um ser inteligente faria. Este artigo tem como objetivo explorar os fundamentos de RNAs e, a partir dessa contextualização, implementar uma rede neural artificial para reconhecer padrões de sequencias de DNA.

Um computador funciona de acordo com o ambiente em que vivemos, abstraindo do comportamento de seus elementos para o seu desenvolvimento. Ou seja, também será sempre dependente de uma entidade para movimentar os objetos de seu mundo, seguindo o conceito de tarefa e execução. Este ambiente está em constante evolução, completamente diferente do que os computadores habitam, já que estes podem fazer apenas as operações que são programados para fazerem. Fica claro observar que os computadores precisam da estrutura do nosso ambiente e não apenas de seu comportamento (Delgado, 2006).

Ludwig Jr. e Costa (2007) relatam que estudos que tinham como objetivo ampliar as estratégias de redução de problemas no controle de sistemas que possuíam características não lineares ficaram cada vez mais frequentes. Da mesma maneira, a busca por estruturas de máquinas inteligentes capazes de substituir o ser humano em determinadas tarefas também tem sido ressaltada. Muitos dos objetivos que buscavam esta metodologia foram alcançados pelo desenvolvimento das técnicas que copiam o funcionamento do cérebro humano (Ludwig Jr. e Costa, 2007).

Para encontrar um novo tipo de máquina, as pesquisas neste seguimento basearam-se na estruturação e no funcionamento do principal elemento do sistema nervoso de um ser vivo, o neurônio. O mesmo é composto pelo corpo de neurônio (soma), que coleta e combina os dados que recebe; pelos dendritos, que esperam os estímulos propagados por outros neurônios; e pelo axônio, que possui a tarefa de transmitir os estímulos para outras células (Tatibana e Kaetsu, 2009). A **Figura 1** apresenta um neurônio biológico com as características descritas.

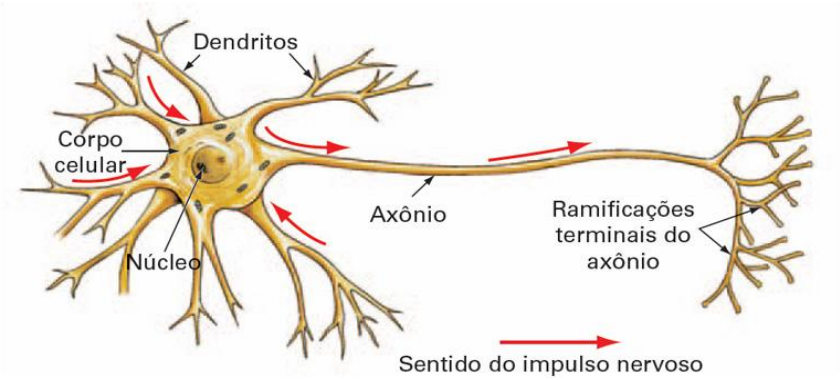

Figura 1. Neurônio biológico

De acordo com Purves et al. (2010), os neurônios são processadores organizados em camadas e interligados pelas sinapses, que são as principais responsáveis pela comunicação intercelular. Esta transmissão sináptica é feita por meio de processos químicos e elétricos, onde a informação codificada por potenciais de ação é transmitida sequencialmente nos contatos sinápticos para a célula seguinte.

Uma base para compreender a estruturação em circuitos dos neurônios foi instituída a partir das pesquisas a respeito da organização celular e dos membros moleculares dos neurônios, assim, diversas de suas funções foram reveladas, abrindo um aglomerado de possibilidades nesta área (PURVES et al., 2010). Posteriormente, surge a iniciativa de criar uma Rede Neural Artificial (RNA). A estrutura de uma RNA é composta por um neurônio de entrada com a função de receber dados, um neurônio oculto para auxiliar o funcionamento da rede e um neurônio de saída para transmitir os dados processados para seu exterior ou para um neurônio adjacente (Barreto, 2002).

Algumas definições de RNAs foram bem aceitas, destacando-se, entre elas, a de Haykin (2001) que determinava que redes neurais artificiais são processadores maciçamente paralelamente distribuídos compostos de simples unidades de processamento, que possuem tendência natural de gravar conhecimento experimental e o disponibilizar para utilização posterior ou imediata.

Já de acordo com o pensamento de Pádua (2000), uma RNA é um sistema formado por unidades de processamento simples distribuídas em paralelo capaz de efetuar funções matemáticas simples sobre dados recebidos em sua entrada. Este modelo foi desenvolvido em 1943 e é chamado pelas inicias de seus autores: MCP (McCulloch e Pitts).

De maneira resumida, uma rede neural artificial adquire conhecimento a partir do ambiente em que está sendo trabalho por meio de métodos de aprendizagem, de pesos de conexão entre os neurônios e de pesos sinápticos. Os pesos têm a capacidade de guardar as informações adquiridas e, dessa forma, todas as operações da RNA se baseiam nelas. Segundo Maia (2012), o *Perceptron* e a *Adaline* estão entre as Redes Neurais mais conhecidas, focando-se em executar operações a partir da ponderação de suas entradas.

As RNAs são utilizadas nos dias de hoje para uma grande quantidade de tarefas como, por exemplo, o reconhecimento de voz, o controle de manipuladores robóticos, a análise de aplicações financeiras, o desenvolvimento de sistemas de controle e otimização, o reconhecimento de padrões em imagens, a visão artificial, entre outros tópicos de interesse (Demuth e Beale, 1996). Além das citadas, outra tarefa que pode ser implementada com base em RNAs é o reconhecimento de padrões, como este artigo descreve por meio da plataforma Java.

Caracterização das RNAs

Segundo Silva et al. (2010), Redes Neurais Artificiais são ferramentas computacionais que copiam o modo como o sistema nervoso de um ser vivo funciona. A partir deste conceito, Braga (1998) relata que, para extrapolar os sistemas computacionais baseados em regras, a computação neural é a principal alternativa, já que é um modo não algorítmico de computação.

Uma RNA possui um método chamado de aprendizagem que é capaz de manipular um conjunto de dados que lhe é fornecido. O resultado deste método é a criação de uma fonte de conhecimento que pode ser utilizada a qualquer instante pela rede (Jesus, 2013).

Após apresentar esta função que tem a capacidade de aprender características sobre um determinado ambiente, diversas aplicabilidades para as RNAs podem ser definidas como, por exemplo, a previsão de ações no mercado financeiro, o reconhecimento de faces em visão computacional, a classificação de padrões de escrita e fala, o controle de trens de grande velocidade, a identificação de anomalias em imagens médicas, o controle de aparelhos eletrônicos e a avaliação de imagens captadas por satélites. (Silva et al., 2010).

Pode-se afirmar que, a aprendizagem é a etapa mais importante de uma RNA. Para um esclarecimento maior, Braga et al. (1998) descreve mais detalhadamente esta metodologia, definido que, inicialmente, um aglomerado de amostras é inserido na rede e este, por sua vez, retira as características que representam a informação fornecida. Futuramente, estas características são utilizadas para obter o resultado de um problema proposto, operando sobre entradas reais e válidas.

Modelo Perceptron

Em 1958, o psicólogo americano Frank Rosenblatt desenvolveu o primeiro e mais simples modelo de uma RNA, o *Perceptron,* que inicialmente era apenas uma abstração. Entretanto, mais tarde viria a se tornar uma base para muitos desenvolvimentos nesta área. Possui um neurônio com pesos ajustáveis, tendo como principal propósito a classificação de padrões linearmente separáveis (Haykin, 2001; Silva et al., 2010).

De acordo com Haykin (2001), o *Perceptron* consegue identificar apenas dois tipos de padrões quando estruturado com um neurônio, mas, quando um número maior de neurônios é adicionado na camada de saída, é possível utilizar um número maior de classes de reconhecimento de padrões, levando sempre em consideração que as classes sejam linearmente separáveis.

Em um *Perceptron* com duas ou mais camadas, o fluxo de transmissão de dados é determinado pela arquitetura *feedforward*, que sempre vai da camada inicial para a final (SILVA et al., 2010).

A **Figura 2** ilustra um *Perceptron*.

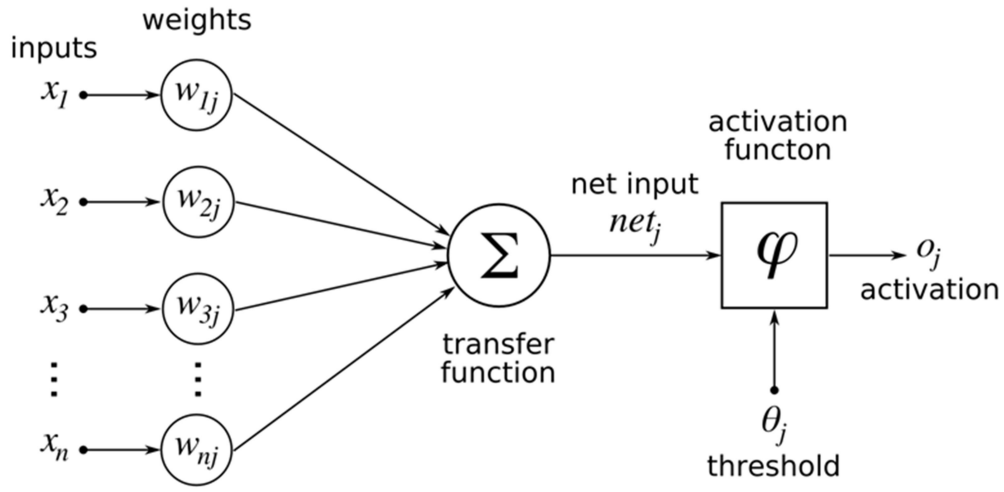

Figura 2.*Perceptron*

Cardon e Muller (1994) observaram que a simplicidade de desenvolvimento é a maior vantagem do *Perceptron* em relação as outras redes, já que possui pouquíssimos parâmetros que precisam de ajuste, além das entradas não necessitarem de um pré-processamento complexo.

Encog Framework

O Encog é uma ferramenta de aprendizagem de máquina extremamente avançada que pode ser utilizada para trabalhar com uma grande quantidade de algoritmos sofisticados. É voltado para a área de Inteligência Artificial em Java, suportando diversos tipos de aplicabilidades relacionados a essa área, além, no contexto deste artigo, das RNAs. Sua especialidade são os algoritmos de aprendizagem de máquina, mas também pode ser aplicado para algoritmos genéticos (Heaton, 2011).

Algoritmo *Perceptron* em Java – Treinamento

Uma sequência de DNA é um conjunto de letras que representam uma cadeia de DNA, composto pelas letras A, C, G e T, respectivamente, Adenina, Citosina, Guanina e Timina. A **Listagem 1** apresenta dez sequências de DNA em vetores de String. Os cinco primeiros vetores representam o primeiro padrão e os cinco últimos o segundo padrão.

Listagem 1. Sequências de DNA

```
String[] padrao1a = { "a", "a", "a", "a", "a", "a", "a", "a", "a", "a",
    "a", "a", "a", "a", "a", "a", "t", "a", "a", "a", "a", "a",
    "a", "a", "a", "t", "a", "a", "a", "a", "a", "a", "a", "a",
    "a", "a", "a", "a", "a", "a", "a", "a", "a", "a", "t", "a",
    "t", "a", "t", "a", "a", "a", "a", "a", "a", "a", "a", "a",
    "a", "a", "a", "a", "a", "a", "a", "a", "a", "a", "a", "a",
    "a", "a", "a", "a", "a", "a", "a", "a", "a", "a", "a", "a",
    "t", "a", "a", "a", "a", "a", "a", "a", "a", "a", "t", "t",
    "a", "a", "a", "a", "a", "a" };
```

```
String[] padrao1b = { "a", "a", "a", "a", "a", "a", "a", "a", "a", "t",
    "a", "a", "a", "t", "a", "t", "t", "t", "a", "t", "a", "a",
    "a", "t", "a", "a", "a", "t", "a", "a", "a", "a", "a", "a",
    "a", "a", "a", "a", "a", "a", "t", "t", "t", "t", "a", "t",
    "a", "t", "a", "a", "a", "a", "a", "a", "a", "a", "a", "a",
    "a", "a", "a", "a", "t", "a", "a", "a", "a", "a", "t", "a",
    "t", "a", "a", "a", "a", "a", "a", "a", "a", "t", "a", "t",
    "a", "a", "a", "a", "a", "a", "a", "t", "a", "a", "a", "a",
    "a", "a", "a", "a", "t", "t" };

String[] padrao1c = { "a", "t", "t", "a", "a", "a", "a", "a", "a", "a",
    "a", "a", "a", "a", "a", "a", "a", "a", "a", "t", "a", "a",
    "a", "a", "a", "a", "t", "a", "a", "a", "a", "a", "t", "t",
    "a", "a", "a", "a", "a", "t", "a", "a", "a", "a", "a", "a",
    "a", "a", "a", "a", "a", "a", "a", "a", "a", "a", "a", "a",
    "t", "a", "a", "t", "t", "a", "a", "a", "a", "a", "a", "t",
    "a", "a", "a", "a", "a", "t", "a", "a", "t", "a", "a", "a",
    "a", "t", "a", "a", "a", "a", "a", "a", "a", "a", "a", "a",
    "a", "a", "t", "a", "a", "t" };

String[] padrao1d = { "a", "a", "a", "a", "t", "a", "a", "a", "t", "a",
    "a", "a", "a", "a", "a", "a", "a", "a", "a", "a", "t", "t",
    "a", "a", "a", "a", "t", "a", "t", "a", "a", "a", "a", "a",
    "a", "a", "a", "a", "a", "t", "a", "a", "a", "a", "a", "a",
    "a", "t", "a", "a", "a", "t", "t", "a", "a", "a", "a", "a",
    "a", "t", "a", "a", "a", "a", "t", "a", "a", "a", "a", "a",
    "a", "t", "a", "a", "t", "a", "a", "a", "a", "a", "a", "a",
    "a", "a", "t", "a", "t", "a", "a", "a", "t", "a", "a", "a",
    "a", "a", "a", "a", "a", "a" };

String[] padrao1e = { "a", "a", "a", "t", "a", "a", "a", "a", "a", "t",
    "a", "a", "a", "a", "t", "a", "a", "a", "t", "t", "a", "a",
    "t", "a", "a", "a", "a", "a", "a", "a", "a", "a", "a", "a",
    "t", "a", "t", "a", "a", "a", "a", "t", "a", "a", "a", "a",
    "a", "a", "a", "a", "t", "a", "a", "a", "a", "a", "a", "a",
    "a", "t", "t", "a", "a", "a", "a", "t", "a", "a", "a", "a",
    "a", "a", "a", "a", "a", "a", "a", "a", "a", "a", "t", "a",
    "a", "a", "a", "a", "a", "a", "a", "t", "a", "t", "a", "t",
    "a", "a", "a", "a", "t", "t" };

String[] padrao2a = { "t", "g", "t", "g", "g", "g", "t", "t", "t", "g",
    "a", "g", "a", "g", "g", "g", "t", "a", "g", "g", "g", "t",
    "g", "g", "g", "t", "g", "g", "g", "g", "a", "t", "t", "a",
    "t", "t", "t", "g", "g", "g", "g", "g", "a", "t", "g", "g",
    "g", "t", "g", "g", "g", "t", "g", "a", "g", "g", "g", "t",
    "t", "a", "g", "g", "g", "g", "g", "g", "g", "g", "t", "g",
    "g", "g", "t", "t", "a", "t", "g", "g", "t", "g", "t", "t",
    "t", "g", "g", "t", "g", "g", "t", "g", "g", "g", "g", "t",
    "g", "g", "t", "t", "t", "t" };

String[] padrao2b = { "g", "a", "t", "g", "g", "g", "t", "t", "t", "t",
    "g", "g", "t", "g", "g", "t", "a", "t", "g", "t", "a", "t",
    "g", "t", "t", "g", "t", "t", "g", "g", "g", "t", "g", "g",
    "t", "g", "g", "g", "t", "g", "g", "g", "g", "t", "t", "g",
    "g", "g", "g", "t", "t", "g", "a", "t", "g", "g", "g", "g",
    "g", "g", "g", "g", "g", "g", "g", "g", "t", "g", "g", "t",
    "g", "g", "t", "g", "g", "t", "t", "g", "g", "t", "t", "a",
    "g", "g", "t", "t", "t", "g", "g", "g", "t", "t", "t", "g",
```

```
        "t", "t", "a", "g", "t", "g" };

    String[] padrao2c = { "g", "g", "g", "g", "a", "g", "g", "a", "g", "t",
        "t", "t", "g", "t", "g", "a", "g", "g", "g", "g", "g", "g",
        "t", "t", "g", "g", "g", "g", "g", "t", "a", "g", "t", "t",
        "g", "g", "g", "g", "g", "a", "g", "t", "g", "g", "g", "g",
        "t", "g", "g", "g", "t", "g", "t", "g", "t", "g", "t", "t",
        "g", "a", "t", "g", "g", "g", "g", "g", "g", "g", "t", "g",
        "g", "t", "g", "g", "g", "a", "t", "g", "g", "t", "a", "g",
        "g", "t", "t", "g", "g", "t", "g", "a", "t", "g", "t", "g",
        "a", "a", "g", "t", "g", "g" };

    String[] padrao2d = { "g", "t", "g", "g", "t", "g", "g", "t", "t", "t",
        "g", "g", "a", "t", "g", "g", "a", "g", "t", "g", "g", "g",
        "t", "g", "g", "g", "a", "g", "t", "t", "t", "a", "t", "g",
        "g", "t", "g", "g", "g", "g", "a", "g", "g", "a", "g", "t",
        "g", "g", "g", "g", "t", "g", "g", "t", "g", "g", "g", "g",
        "t", "g", "g", "g", "g", "g", "t", "a", "t", "g", "g", "g",
        "g", "g", "g", "a", "t", "t", "g", "t", "g", "g", "g", "a",
        "g", "g", "t", "g", "g", "g", "t", "g", "g", "t", "g", "g",
        "a", "g", "t", "t", "g", "g" };

    String[] padrao2e = { "a", "g", "g", "g", "g", "g", "t", "a", "g", "g",
        "g", "g", "g", "t", "g", "g", "g", "g", "g", "t", "g", "g",
        "g", "g", "a", "g", "a", "g", "g", "a", "g", "g", "g", "g",
        "g", "g", "g", "t", "g", "g", "t", "g", "g", "g", "t", "g",
        "t", "a", "g", "a", "g", "g", "t", "g", "g", "t", "t", "g",
        "g", "t", "a", "g", "g", "g", "t", "g", "g", "g", "g", "g",
        "g", "t", "t", "t", "g", "t", "g", "g", "g", "t", "g", "a",
        "g", "g", "g", "t", "t", "a", "g", "g", "g", "g", "g", "g",
        "g", "t", "t", "t", "a", "g" };
```

Para trabalhar com esse tipo de amostragem com uma RNA em Java, foi necessária a conversão dessas cadeias para valores numéricos. Na **Listagem 2**, é demonstrada a função criada para fazer esta conversão, onde o caractere **a** passa para o valor 0.25, o **t** para 0.50, o **c** para 0.75 e o **g** para 1.00.

Listagem 2. Conversão das sequências de DNA

```
public static double[] converter(String[] a) {
    for (int i = 0; i < a.length; i++) {
        if (a[i] == "a") {
            b[i] = 0.25;
        } else if (a[i] == "t") {
            b[i] = 0.50;
        } else if (a[i] == "c") {
            b[i] = 0.75;
        } else if (a[i] == "g") {
            b[i] = 1.00;
        }
    }

    return b;
}
```

O resultado da conversão pode ser visto na **Listagem 3**.

Listagem 3. Resultado da conversão das sequências de DNA

```
double[] vetorAmostra1a = new double[] { 0.25, 0.25, 0.25, 0.25, 0.25,
    0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25,
    0.25, 0.5, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.5,
    0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25,
    0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.5, 0.25, 0.5,
    0.25, 0.5, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25,
    0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25,
    0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25,
    0.25, 0.25, 0.25, 0.25, 0.25, 0.5, 0.25, 0.25, 0.25, 0.25,
    0.25, 0.25, 0.25, 0.25, 0.25, 0.5, 0.5, 0.25, 0.25, 0.25, 0.25,
    0.25, 0.25 };

double[] vetorAmostra1b = new double[] { 0.25, 0.25, 0.25, 0.25, 0.25,
    0.25, 0.25, 0.25, 0.25, 0.5, 0.25, 0.25, 0.25, 0.5, 0.25, 0.5,
    0.5, 0.5, 0.25, 0.5, 0.25, 0.25, 0.25, 0.5, 0.25, 0.25, 0.25,
    0.5, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25,
    0.25, 0.25, 0.25, 0.5, 0.5, 0.5, 0.5, 0.25, 0.5, 0.25, 0.5,
    0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25,
    0.25, 0.25, 0.25, 0.25, 0.5, 0.25, 0.25, 0.25, 0.25, 0.25, 0.5,
    0.25, 0.5, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.5,
    0.25, 0.5, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.5, 0.25,
    0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.5, 0.5 };

double[] vetorAmostra1c = new double[] { 0.25, 0.5, 0.5, 0.25, 0.25, 0.25,
    0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25,
    0.25, 0.25, 0.5, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.5, 0.25,
    0.25, 0.25, 0.25, 0.25, 0.5, 0.5, 0.25, 0.25, 0.25, 0.25, 0.25,
    0.5, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25,
    0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.5, 0.25, 0.25,
    0.5, 0.5, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.5, 0.25, 0.25,
    0.25, 0.25, 0.25, 0.5, 0.25, 0.25, 0.5, 0.25, 0.25, 0.25, 0.25,
    0.5, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25,
    0.25, 0.25, 0.5, 0.25, 0.25, 0.5 };

double[] vetorAmostra1d = new double[] { 0.5, 1.0, 0.5, 1.0, 1.0, 1.0,
    0.5, 0.5, 0.5, 1.0, 0.25, 1.0, 0.25, 1.0, 1.0, 1.0, 0.5, 0.25,
    1.0, 1.0, 1.0, 0.5, 1.0, 1.0, 1.0, 0.5, 1.0, 1.0, 1.0, 1.0, 0.25,
    0.5, 0.5, 0.25, 0.5, 0.5, 0.5, 1.0, 1.0, 1.0, 1.0, 1.0, 0.25,0.5,
    1.0, 1.0, 1.0, 0.5, 1.0, 1.0, 1.0, 0.5, 1.0, 0.25, 1.0, 1.0, 1.0,
    0.5, 0.5, 0.25, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.5, 1.0,
    1.0, 1.0, 0.5, 0.5, 0.25, 0.5, 1.0, 1.0, 0.5, 1.0, 0.5, 0.5, 0.5,
    1.0, 1.0, 0.5, 1.0, 1.0, 0.5, 1.0, 1.0, 1.0, 1.0, 0.5, 1.0, 1.0,
    0.5, 0.5, 0.5, 0.5 };

double[] vetorAmostra1e = new double[] { 0.25, 0.25, 0.25, 0.5, 0.25,
    0.25, 0.25, 0.25, 0.25, 0.5, 0.25, 0.25, 0.25, 0.25, 0.5, 0.25,
    0.25, 0.25,0.5, 0.5, 0.25, 0.25, 0.5, 0.25, 0.25, 0.25, 0.25,
    0.25, 0.25, 0.25,0.25, 0.25, 0.25, 0.25, 0.5, 0.25, 0.5, 0.25,
    0.25, 0.25, 0.25, 0.5, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25,
    0.25, 0.25, 0.5, 0.25, 0.25, 0.25,0.25, 0.25, 0.25, 0.25, 0.25,
    0.5, 0.5, 0.25, 0.25, 0.25, 0.25, 0.5, 0.25, 0.25, 0.25, 0.25,
    0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25,0.25, 0.25, 0.5,
    0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.5, 0.25, 0.5,
    0.25, 0.5, 0.25, 0.25, 0.25, 0.25, 0.5, 0.5 };

double[] vetorAmostra2a = new double[] { 0.5, 1.0, 0.5, 1.0, 1.0, 1.0,
```

```
    0.5, 0.5, 0.5, 1.0, 0.25, 1.0, 0.25, 1.0, 1.0, 1.0, 0.5, 0.25,
    1.0, 1.0, 1.0, 0.5, 1.0, 1.0, 1.0, 0.5, 1.0, 1.0, 1.0, 1.0,
    0.25, 0.5, 0.5, 0.25, 0.5, 0.5, 0.5, 1.0, 1.0, 1.0, 1.0, 1.0,
    0.25, 0.5, 1.0, 1.0, 1.0, 0.5, 1.0, 1.0, 1.0, 0.5, 1.0, 0.25,
    1.0, 1.0, 1.0, 0.5, 0.5, 0.25, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
    1.0, 1.0, 0.5, 1.0, 1.0, 1.0, 0.5, 0.5, 0.25, 0.5, 1.0, 1.0,
    0.5, 1.0, 0.5, 0.5, 0.5, 1.0, 1.0, 0.5, 1.0, 1.0, 0.5, 1.0,
    1.0, 1.0, 1.0, 0.5, 1.0, 1.0, 0.5, 0.5, 0.5, 0.5 };

double[] vetorAmostra2b = new double[] { 1.0, 0.25, 0.5, 1.0, 1.0, 1.0,
    0.5, 0.5, 0.5, 0.5, 1.0, 1.0, 0.5, 1.0, 1.0, 0.5, 0.25, 0.5,
    1.0, 0.5, 0.25, 0.5, 1.0, 0.5, 0.5, 1.0, 0.5, 0.5, 1.0, 1.0,
    1.0, 0.5, 1.0, 1.0, 0.5, 1.0, 1.0, 1.0, 0.5, 1.0, 1.0, 1.0,
    1.0, 0.5, 0.5, 1.0, 1.0, 1.0, 1.0, 0.5, 0.5, 1.0, 0.25, 0.5,
    1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
    0.5, 1.0, 1.0, 0.5, 1.0, 1.0, 0.5, 1.0, 1.0, 0.5, 0.5, 1.0,
    1.0, 0.5, 0.5, 0.25, 1.0, 1.0, 0.5, 0.5, 0.5, 1.0, 1.0, 1.0,
    0.5, 0.5, 0.5, 1.0, 0.5, 0.5, 0.25, 1.0, 0.5, 1.0 };

double[] vetorAmostra2c = new double[] { 1.0, 1.0, 1.0, 1.0, 0.25, 1.0,
    1.0, 0.25, 1.0, 0.5, 0.5, 0.5, 1.0, 0.5, 1.0, 0.25, 1.0, 1.0,
    1.0, 1.0, 1.0, 1.0, 0.5, 0.5, 1.0, 1.0, 1.0, 1.0, 1.0, 0.5,
    0.25, 1.0, 0.5, 0.5, 1.0, 1.0, 1.0, 1.0, 1.0, 0.25, 1.0, 0.5,
    1.0, 1.0, 1.0, 1.0, 0.5, 1.0, 1.0, 1.0, 0.5, 1.0, 0.5, 1.0,
    0.5, 1.0, 0.5, 0.5, 1.0, 0.25, 0.5, 1.0, 1.0, 1.0, 1.0, 1.0,
    1.0, 1.0, 0.5, 1.0, 1.0, 0.5, 1.0, 1.0, 1.0, 0.25, 0.5, 1.0,
    1.0, 0.5, 0.25, 1.0, 1.0, 0.5, 0.5, 1.0, 1.0, 0.5, 1.0, 0.25,
    0.5, 1.0, 0.5, 1.0, 0.25, 0.25, 1.0, 0.5, 1.0, 1.0 };

double[] vetorAmostra2d = new double[] { 1.0, 0.5, 1.0, 1.0, 0.5, 1.0,
    1.0, 0.5, 0.5, 0.5, 1.0, 1.0, 0.25, 0.5, 1.0, 1.0, 0.25, 1.0,
    0.5, 1.0, 1.0, 1.0, 0.5, 1.0, 1.0, 1.0, 0.25, 1.0, 0.5, 0.5, 0.5,
    0.25, 0.5, 1.0, 1.0, 0.5, 1.0, 1.0, 1.0, 1.0, 0.25, 1.0, 1.0,
    0.25, 1.0, 0.5, 1.0, 1.0, 1.0, 1.0, 0.5, 1.0, 1.0, 0.5, 1.0,
    1.0, 1.0, 1.0, 0.5, 1.0, 1.0, 1.0, 1.0, 1.0, 0.5, 0.25, 0.5,
    1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.25, 0.5, 0.5, 1.0, 0.5, 1.0,
    1.0, 1.0, 0.25, 1.0, 1.0, 0.5, 1.0, 1.0, 1.0, 0.5, 1.0, 1.0,
    0.5, 1.0, 1.0, 0.25, 1.0, 0.5, 0.5, 1.0, 1.0 };

double[] vetorAmostra2e = new double[] { 0.25, 1.0, 1.0, 1.0, 1.0, 1.0,
    0.5, 0.25, 1.0, 1.0, 1.0, 1.0, 1.0, 0.5, 1.0, 1.0, 1.0, 1.0, 1.0,
    0.5, 1.0, 1.0, 1.0, 1.0, 0.25, 1.0, 0.25, 1.0, 1.0, 0.25, 1.0,
    1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.5, 1.0, 1.0, 0.5, 1.0, 1.0, 1.0,
    0.5, 1.0, 0.5, 0.25, 1.0, 0.25, 1.0, 1.0, 0.5, 1.0, 1.0, 0.5,
    0.5, 1.0, 1.0, 0.5, 0.25, 1.0, 1.0, 1.0, 0.5, 1.0, 1.0, 1.0, 1.0,
    1.0, 1.0, 0.5, 0.5, 0.5, 1.0, 0.5, 1.0, 1.0, 1.0, 0.5, 1.0, 0.25,
    1.0, 1.0, 1.0, 0.5, 0.5, 0.25, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
    0.5, 0.5, 0.5, 0.25, 1.0 };
```

Na **Listagem 4** é representado o algoritmo utilizado para o treinamento da Rede Neural. Possui como principal objetivo treinar a RNA com as amostras obtidas após a conversão das sequencias de DNA. Os vetores numéricos das sequencias foram colocados em uma matriz para servir de entrada para a rede, onde, cada vetor representa uma coluna desta matriz. Um vetor que possui as saídas desejadas para cada uma das amostras foi criado, e, para armazenar os pesos, outro arranjo unidimensional também foi criado. O vetor de pesos foi inicializado com valores randômicos da divisão de 1.0 por números de 1 a 100, assim os pesos sempre alcançarão um número baixo.

Em um arco de repetição é feita a somatória da multiplicação de cada peso pela sua respectiva amostra, esta somatória passa por uma função de ativação, que testa se a mesma é maior ou igual a zero, caso a resposta seja verdadeira, o retorno da função será no valor 1, caso contrário o retorno será -1.

O resultado é conferido com o vetor de saída desejada, se os dois se igualarem, o treinamento para a amostra é finalizado, caso contrário, uma nova linha de aprendizado é definida, multiplicando a anterior pela saída desejada para a amostra, menos o retorno da função de ativação. Os pesos são atualizados somando o seu valor anterior ao novo aprendizado e multiplicando pelas suas respectivas amostras.

Listagem 4. Implementação do algoritmo *Perceptron* em Java – Treinamento

```
package dna;

import java.util.Random;

public class PerceptronTreino {

    static int A = 8;

    private static String Vetor = "";

    private static String VetorPesos = "";

    public static void main(String[] args) {
        Random random = new Random(System.currentTimeMillis());

        double[][] amostrasTreinamento = new double[100][A];

        int l = 0;
        int c = 0;

        for (; c < A; c++) {
            for (; l < 100; l++) {
                amostrasTreinamento[l][c] = vetorAmostra1a[l];
                amostrasTreinamento[l][c + 1] = vetorAmostra1b[l];
                amostrasTreinamento[l][c + 2] = vetorAmostra1c[l];
                amostrasTreinamento[l][c + 3] = vetorAmostra1d[l];
                amostrasTreinamento[l][c + 4] = vetorAmostra2a[l];
                amostrasTreinamento[l][c + 5] = vetorAmostra2b[l];
                amostrasTreinamento[l][c + 6] = vetorAmostra2c[l];
                amostrasTreinamento[l][c + 7] = vetorAmostra2d[l];
            }
        }

        for (int i = 0; i < amostrasTreinamento.length; i++) {
            for (int k = 0; k < A; k++) {
                System.out.println("[" + i + "]" + "[" + k + "]" +
                    amostrasTreinamento[i][k]);
            }

        }

        double[] saidasDesejadas = new double[] { 1.0, 1.0, 1.0, 1.0, -1.0,
                                                  -1.0, -1.0, -1.0};
        double[] w = new double[100];
```

```
        for (int i = 0; i < w.length; i++) {
            w[i] = 1.0 / (random.nextInt(100) + 1.0);
        }


        double linhaAprendizado = 0.05;
        int epocas = 0;
        String erro;

        do {
            erro = "inexiste";

            for (int k = 0; k < A; k++) {
                double wsoma = 0.0;

                for (int i = 0; i < 100; i++) {
                    wsoma += w[i] * amostrasTreinamento[i][k];
                }

                int x = funcaoTransicao(wsoma);

                // Teste para verificar se o resultado do treino
                // condiz com o resultado desejado
                if (x != saidasDesejadas[k]) {
                    double novoAprendizado = linhaAprendizado * (saidasDesejadas[k] - x);

                    for (int i = 0; i < w.length; i++) {
                        w[i] = w[i] + novoAprendizado * amostrasTreinamento[i][k];
                    }

                    erro = "existe";
                }
            }

            epocas++;
        } while (erro.equals("existe") && epocas < 1000);

        for (int i = 0; i < w.length; i++) {
            System.out.println("Peso treinado = " + i + w[i]);

            VetorPesos += w[i] + (((i + 1) != w.length) ? ", " : "");
        }

        System.out.println(VetorPesos);
        System.out.println("Épocas - " + epocas);
    }

    private static int funcaoTransicao(double wsoma) {
        return (wsoma >= 0.0) ? 1 : -1;
    }
}
```

Após obter o completo treinamento da rede, passa-se para a etapa de simulação, conforme explanado na próxima seção.

Algoritmo *Perceptron* em Java – Simulação

O principal elemento que será utilizado do algoritmo de treinamento para fazer a simulação da RNA é o vetor de pesos, pois ele representa o que a rede aprendeu a respeito das amostras inicialmente apresentadas a ela. A **Listagem 5** apresenta alguns vetores de pesos obtidos após treinamentos da Rede Neural. Simulações foram feitas com cada um dos vetores para determinar sua eficiência. Por meio do relacionamento do resultado obtido com o resultado esperado da simulação, pode-se constatar que o primeiro vetor de pesos é regular, enquanto o segundo foi avaliado como péssimo e o terceiro como ótimo.

Listagem 5. Vetores de pesos treinados

```
// Avaliação de Pesos: Regular
double[] w = new double[] { 0.01492537313432836, -0.05887096774193553, 0.07380952380952381,
-0.13888888888888895, -0.11296296296296301, -0.12916666666666674, 0.060000000000000005,
0.07777777777777778, 0.06123595505617978, 0.02439024390243902, 0.10064102564102564,
-0.10833333333333339, 0.2916666666666667, -0.03461538461538464, -0.11153846153846159,
0.011764705882352948, 0.33563829787234045, 0.23765822784810128, -0.1341269841269842,
0.014084507042253502, -0.01249999999999999, 0.175, -0.13823529411764712,
0.01785714285714285, -0.06875000000000006, 0.16176470588235298, -0.08529411764705888,
0.07692307692307693, -0.12222222222222229, -0.13387096774193555, 0.0855263157894737,
0.06265822784810127, 0.016949152542372878, 0.0903846153846154, 0.0625,
0.020408163265306124, 0.3333333333333336, -0.11296296296296301, -0.05652173913043481,
-0.1214285714285715, -0.03412698412698415, 0.05000000000000001, 0.2053030303030303,
0.17702702702702708, 0.09, -0.03684210526315793, 0.049999999999999996, 0.1136986301369863,
0.016393442622950817, -0.0833333333333334, -0.08305084745762717, 0.015151515151515152,
-0.06424731182795704, 0.13992537313432835, -0.13275862068965522, -0.13648648648648654,
-0.09117647058823533, 0.017241379310344827, 0.013157894736842105, 0.08530927835051547,
-0.13611111111111118, -0.1341269841269842, -0.026744186046511666, -0.13717948717948725,
-0.13888888888888895, -0.12619047619047624, 0.04285714285714284, -0.12297297297297302,
0.11724137931034481, -0.08989898989898995, -0.035294117647058844, -0.09444444444444447,
0.06923076923076923, 0.014492753623188408, 0.08586956521739131, 0.06923076923076923,
-0.009090909090909101, -0.09117647058823533, 0.023255813953488372, 0.3333333333333333,
0.12692307692307692, 0.18575268817204302, 0.16123595505617977, -0.12777777777777782,
-0.0743589743589744, 0.1, -0.08305084745762717, -0.13611111111111118, 0.041666666666666664,
-0.03333333333333336, -0.0772727272727273, -0.0882352941176471, 0.08225806451612903,
0.16785714285714287, -0.067741935483871, -0.08750000000000005, 0.11499999999999999,
0.010000000000000002, 0.19545454545454546, 0.1108695652173913 };

// Avaliação de Pesos: Péssima
double[] w = new double[] { -0.05613207547169812, -0.1636363636363637, -
0.02236842105263158, -0.13653846153846158, 0.051923076923076926, -0.16130136986301377,
-0.05459183673469389, 0.09087301587301588, -0.059848484848484866, 0.27499999999999997,
0.0869047619047619, -0.06401098901098903, 0.04166666666666667, 0.042241379310344836,
-0.13928571428571435, 0.12045454545454545, 0.31666666666666665, 0.09224137931034482,
-0.16323529411764712, -0.029545454545454555, -0.16217948717948724, -0.06413043478260871,
-0.04051724137931036, 0.043867924528301884, -0.15576923076923083, 0.3000000000000001,
-0.06388888888888891, -0.06469072164948454, -0.15539215686274516, -0.051744186046511674,
0.15064102564102563, -0.06479591836734694, 0.035869565217391305, 0.08765822784810126,
-0.05459183673469389, -0.06458333333333335, -0.06388888888888888, -0.14868421052631584,
-0.16323529411764712, -0.012654320987654358, -0.06350574712643677, 0.03541666666666668,
0.1, 0.03992537313432836, -0.03245614035087719, -0.062179487179487195, 0.0857142857142857,
0.07763157894736841, 0.012500000000000011, -0.0840909090909091, -0.06337209302325589,
-0.05961538461538462, -0.06050724637681162, -0.013372093023255816, -0.05775862068965517,
-0.16413043478260878, -0.02236842105263158, 0.03833333333333334, -0.06111111111111111,
0.1422413793103448, -0.058606557377049194, -0.1517441860465117, -0.05912698412698415,
-0.14469696969696974, -0.15459183673469393, -0.1583333333333334, -0.15539215686274516,
```

```
-0.1596153846153847, 0.1472222222222222, -0.15648148148148153, -0.06007462686567168,
-0.019444444444444445, -0.065, 0.06785714285714282, -0.012804878048780485,
0.09087301587301588, -0.03333333333333334, -0.09807692307692312, 0.9250000000000002,
0.04139344262295081, 0.0871951219512195, 0.04038461538461539, 0.06851851851851853,
-0.0035714285714285796, -0.03796296296296297, -0.05613207547169812, -0.1623417721518988,
-0.047972972972972976, 0.25833333333333347, 0.09194915254237288, -0.06469072164948456,
-0.164795918367347, 0.06333333333333334, 0.06851851851851853, -0.011301369863013729,
-0.011666666666666665, 0.0019230769230769232, 0.06847826086956521, 0.03662790697674419,
0.35833333333333334 };

// Avaliação de Pesos: Ótima
double[] w = new double[] { -0.20372340425531915, 0.1583333333333333, 0.07272727272727272,
-0.31424731182795695, 0.14999999999999997, -0.31499999999999995, -0.0621794871794872,
0.22631578947368425, -0.0105072463768116, 0.27564102564102566, 0.30833333333333374,
-0.15613207547169833, 0.06041666666666667, 0.11282051282051281, -0.3057692307692308,
0.3035714285714286, 0.5520270270270272, 0.22222222222222224, -0.25416666666666665,
0.1878205128205128, -0.023076923076923092, -0.059375000000000004, -0.09469696969696986,
0.24999999999999997, -0.15648148148148155, 0.027027027027027042, 0.0378205128205128,
0.060869565217391314, -0.17500000000000013, -0.04807692307692318, 0.11612903225806451,
0.024390243902439032, 0.11666666666666668, 0.11265822784810128, -0.019444444444444473,
-0.16401098901098907, -0.1972222222222223, -0.2535714285714286, -0.14642857142857152,
0.0634615384615384, -0.011111111111111155, 0.07500000000000001, 0.38333333333333325,
0.24038461538461542, 0.06408450704225352, -0.03611111111111115, 0.061111111111111116,
0.08333333333333336, -0.04999999999999998, -0.07500000000000004, 0.044230769230769185,
-0.20681818181818185, 0.06886792452830184, 0.05441176470588235, -0.15775862068965535,
-0.24166666666666672, -0.16469072164948473, -0.06130136986301373, 0.016666666666666663,
0.06075268817204301, -0.09166666666666681, -0.14736842105263165, 0.040151515151515105,
-0.22500000000000012, -0.23500000000000007, -0.2327586206896551, -0.09558823529411772,
-0.26617647058823535, 0.16162790697674423, -0.03765432098765438, -0.0882352941176471,
-0.16295180722891583, 0.04999999999999992, -0.09736842105263169, 0.041666666666666595,
0.33970588235294125, -0.00887096774193552, -0.25372340425531903, -0.0864864864864865,
0.21388888888888888, 0.1663934426229508, 0.32272727272727275, 0.011627906976744179,
0.06111111111111109, 0.03563829787234041, -0.03653846153846156, -0.08409090909090913,
-0.14375000000000016, 0.0749999999999999, 0.13690476190476192, -0.006481481481481477,
-0.11401098901098913, 0.21666666666666654, 0.018867924528301896, 0.1482558139534883,
0.06234567901234564, 0.20357142857142857, -0.014690721649484609, 0.2055555555555556,
0.13851351351351354 };
```

O terceiro vetor de pesos se mostrou eficiente após alguns testes e, portanto, foi escolhido para fazer parte do algoritmo final de simulação. Para a simulação, um vetor numérico de uma sequência de DNA precisa ser submetido a um método que realiza a somatória da multiplicação dos pesos pelas amostras. Esta soma é dada como entrada para a função de transição que verifica se o resultado é 1 (Padrão 1) ou -1 (Padrão 2), obtendo assim a saída da RNA. Na **Listagem 6** é possível ver estas duas funções.

Listagem 6. Escopo principaldo algoritmo *Perceptron* em Java – Simulação

```
// ...
double wsoma = 0.0;

for (int k = 0; k < vetorAmostra.length; k++) {
    wsoma += w[k] * vetorAmostra[k];
}

// Função de verificação
int x = funcaoTransicao(wsoma);
```

```
if (x == -1) {
    System.out.println("Amostra compatível com o padrão 2 (-1).");
} else if (x == 1) {
    System.out.println("Amostra compatível com o padrão 1 (1).");
}

private static int funcaoTransicao(double wsoma) {
    return (wsoma >= 0.0) ? 1 : -1;
}
```

Na **Listagem 7** é apresentado o vetor de uma das sequências de DNA obtido na função de conversão já demonstrada neste artigo.

Listagem 7. Vetor da sequencia de DNA

```
double[] vetorAmostra = new double[] {
    0.25, 0.25, 0.25, 0.5,  0.25, 0.25, 0.25, 0.25, 0.25, 0.5 ,
    0.25, 0.25, 0.25, 0.25, 0.5 , 0.25, 0.25, 0.25, 0.5 , 0.5 ,
    0.25, 0.25, 0.5,  0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25,
    0.25, 0.25, 0.25, 0.25, 0.5 , 0.25, 0.5 , 0.25, 0.25, 0.25,
    0.25, 0.5 , 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25,
    0.5 , 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.5 ,
    0.5 , 0.25, 0.25, 0.25, 0.25, 0.5,  0.25, 0.25, 0.25, 0.25,
    0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25,
    0.5 , 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.5 ,
    0.25, 0.5 , 0.25, 0.5 , 0.25, 0.25, 0.25, 0.25, 0.5 , 0.5 }; // Padrão 1
```

Na **Figura 3**, encontra-se o resultado da simulação com a sequência apresentada na **Listagem 7**.

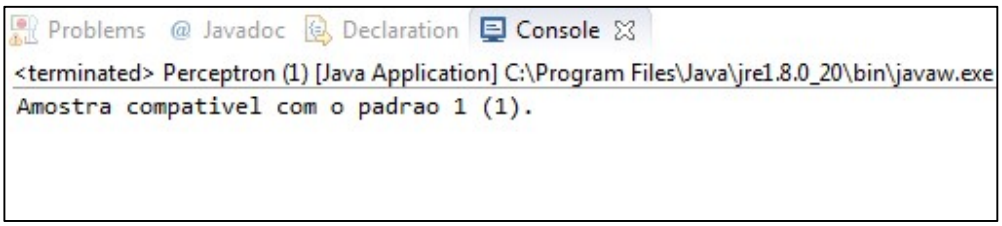

Figura 3. Resultado da primeira simulação exibida no console da IDE

Para consolidar a confiança na RNA, uma segunda simulação foi realizada com base em uma nova sequência. A amostra está localizada na **Listagem 8** e o resultado pode ser visto na **Figura 4**.

Listagem 8. Segunda amostra de sequência de DNA para simulação

```
double[] vetorAmostra = new double[] {
    1.0 , 1.0, 1.0, 1.0, 0.25, 1.0 , 1.0, 0.25, 1.0, 0.5 ,
    0.5 , 0.5, 1.0, 0.5, 1.0 , 0.25, 1.0, 1.0 , 1.0, 1.0 ,
    1.0 , 1.0, 0.5, 0.5, 1.0 , 1.0 , 1.0, 1.0 , 1.0, 0.5 ,
    0.25, 1.0, 0.5, 0.5, 1.0 , 1.0 , 1.0, 1.0 , 1.0, 0.25,
    1.0 , 0.5, 1.0, 1.0, 1.0 , 1.0 , 0.5, 1.0 , 1.0, 1.0 ,
    0.5 , 1.0, 0.5, 1.0, 0.5 , 1.0 , 0.5, 0.5 , 1.0, 0.25,
    0.5 , 1.0, 1.0, 1.0, 1.0 , 1.0 , 1.0, 1.0 , 0.5, 1.0 ,
    1.0 , 0.5, 1.0, 1.0, 1.0 , 0.25, 0.5, 1.0 , 1.0, 0.5 ,
    0.25, 1.0, 1.0, 0.5, 0.5 , 1.0 , 1.0, 0.5 , 1.0, 0.25,
```

```
0.5 , 1.0, 0.5, 1.0, 0.25, 0.25, 1.0, 0.5 , 1.0, 1.0 }; // Padrão 2
```

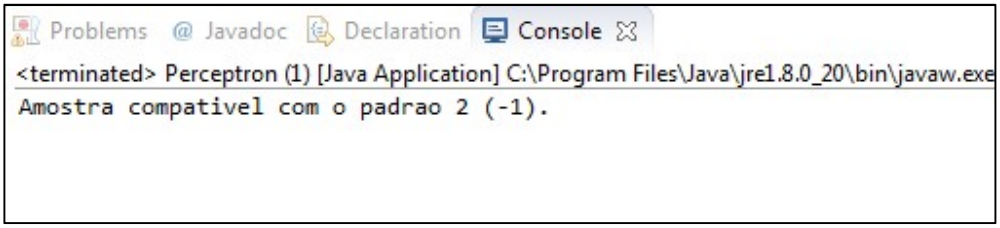

Figura 4. Resultado da simulação com uma amostra correspondente ao padrão 2

Conclusão

Este projeto teve como finalidade trabalhar com aplicações voltadas para o reconhecimento de padrões e estruturadas como uma Rede Neural Artificial. Foi possível alcançar este objetivo com os algoritmos desenvolvidos na linguagem de programação Java e com a IDE de desenvolvimento Eclipse.

Os algoritmos foram capazes de cumprir com seu propósito, separando corretamente os padrões apresentados a eles, além de resultar em seu devido reconhecimento. Demonstrou-se também uma simplicidade estrutural e extrema eficiência e rapidez ao fazer o reconhecimento, alcançando um treinamento em apenas cinco épocas e identificando as amostras em questão de milissegundos.

Conclui-se que este artigo cumpriu com seu propósito, apresentando os conceitos de Redes Neurais Artificiais, bem como métodos para implementação em Java, resultando em um material claro e objetivo para os interessados. Como trabalho futuro, pretende-se ampliar as bases de treinamento com padrões mais complexos e maior variedade. Pretende-se extrair o máximo de características das amostras para evitar falhas, já que o próximo passo é aplicar esta metodologia em componentes do mundo real.

Links e Referências

1. BARRETO, J. B. Introdução às Redes Neurais Artificiais. 2002. 57p. Dissertação - Departamento de Informática e de Estatística - Laboratório de Conexionismo e Ciências Cognitivas UFSC, Santa Catarina, Florianópolis, 2002.

2. BRAGA, Antônio de Pádua; CARVALHO, A. P. L. F.; LUDEMIR, Teresa Bernarda. Fundamentos de redes neurais artificiais. XI Escola brasileira de computação, 1998.

3. CARDON, André; MÜLLER, Daniel Nehme. Introdução Às Redes Neurais Artificiais. 1994. 31p. Curso de Pós-Graduação em Ciência da Computação - Instituto de Informática - Universidade Federal do Rio Grande do Sul, Rio Grande do Sul, Porto Alegre, 1994.

4. DELGADO, Cynthia. Finding the Evolution in Medicine. Artigo em NIH Record (National Institute of Health). Disponível em: <http://nihrecord.nih.gov/newsletters/2006/07_28_2006/story03.htm>. Acesso em: 20abr. 2015.

5. DEMUTH, H.; BEALE, M. Neural Network Toolbox User'sGuide. Versão 4.0.The Mathworks Inc, 1998.

6. HAYKIN, S. Redes Neurais: Princípios e prática. 2. ed. Tradução de Paulo Martins Angel. Porto Alegre: Editora Bookman, 2001.

7. HEATON, J. Programming Neural Networks with Encog3 in Java. 2nd ed. Heaton Research, Chesterfield. 2011.

8. JESUS, Willian Moldenhauer. Uso de redes neurais artificiais auto regressivas para estimar a capacidade de refrigeração de compressores através de dados de regime transiente. 2013. 58 p.. Trabalho de Conclusão de Curso (Graduação em Engenharia da Mobilidade) - Universidade Federal de Santa Catarina, Campus Joinville. Florianópolis, SC, 2013.

9. LEE, Alexander. Digit Recognition. Disponível em: <http://alexander-lee.net/digit-recognition/>. Acesso em: 01 out. 2015.

10. LUDWIG JR., O; COSTA, Eduard Montgomery M. Redes Neurais: Fundamentos e Aplicações com Programas em C. Rio de Janeiro: Editora Ciência Moderna Ltda. 2007.

11. MAIA, Renato Dourado. Inteligência Computacional – Redes Neurais – Adaline. 2012. 36p. Faculdade de Ciências e Tecnologia de Montes Claros - Fundação Educacional Montes Claros, Minas Gerais, Montes Claros, 2012.

12. PURVES, Daleet al. Neurociências. 4. ed. Porto Alegre: Editora Artmed, 2010.

13. PADUA, A. et al. Redes Neurais Artificiais-Teoria e Aplicações. Rio de Janeiro: Editora LTC, 2000.

14. SILVA, Ivan Nunes da; SPATTI, Danilo Hernane; FLAUZINO, Rogério Andrade. Redes Neurais Artificiais para engenharia e ciências aplicadas – São Paulo: Artliber, 2010.

15. TATIBANA, Cassia Yuri; KAETSU, Deisi Yuri. Uma introdução às redes neurais. Departamento de Informática (DIN) da Universidade Estadual de Maringá (UEM)**.** Disponível em: http://www.din.uem.br/ia/neurais/#links. Acesso em: 12 out. 2014.