# Implementação de um neurônio artificial para resolução das portas lógicas E e OU

**Gabriel Félix Ramos (31407), Joyce Mayara Almeida (30952)**

Instituto de Matemática e Computação (IMC) – Universidade Federal de Itajubá (UNIFEI)
Caixa Postal 50 – 37500 903 – Itajubá – MG – Brazil

gabriel.felixramos@gmail.com, joycemayara@unifei.edu.br

***Resumo.*** *Este documento descreve a implementação das portas lógicas E e OU solucionadas por uma neurônio artificial na Linguagem C. Ainda, mostra informações sobre o problema que o neurônio irá aprender, especificando seus padrões de entrada e saída, e fornece detalhes e justificativas da implementação realizada, com grande enfoque em como o modelo funciona para a aplicação em questão.*

## 1. Visão introdutória de um neurônio artificial e descrição do problema

"Modelos neurais, também conhecidos como processamento paralelo distribuído (PDP, do inglês Parallel Distributes Processing) ou sistemas conexionistas, retiram a ênfase do uso explícito de símbolos para solucionar problemas" (LUGER, 2013).

Em vez disso, esses modelos afirmam que a inteligência surge em sistemas de componentes simples, interativos (ou neurônios biológicos ou artificiais), por meio de um processo de aprendizado, ou de adaptação, pelo qual as conexões entre os componentes são ajustadas (LUGER, 2013). O processamento nesses sistemas é distribuído por meio e conjuntos ou camadas de neurônios (LUGER, 2013).

Para construir uma rede neural, o projetista deve criar um esquema para codificar parões do mundo em quantidades numéricas da rede (LUGER, 2013).

Em sistemas conexionistas, o processamento é paralelo e distribuído sem manipulação de símbolos como símbolos; Os padrões de um domínio são codificados com vetores numéricos e as conexões entre os componentes (neurônios) são também representadas por valores numéricos; A transformação de padrões é o resultado de operações numéricas, normalmente multiplicações de matrizes (LUGER, 2013).

"Os algoritmos e as arquiteturas que implementam essas técnicas são normalmente treinados ou condicionados, em vez de serem explicitamente programados" (LUGER, 2013).

A base das redes neurais é o neurônio artificial, que consiste em: sinais de

entrada, dados que podem vir do ambiente ou da ativação de outros neurônios; um conjunto de pesos com valor real, que descrevem as forças de conexão e armazenam o aprendizado da rede (obtido pelo treinamento do neurônio); Um nível de ativação, calculado pela soma ponderada das entradas com seus pesos; Uma função de limiar, que calcula o estado final, ou de saída, do neurônio, determinando o quanto o nível de ativação do neurônio está abaixo ou acima de um valor de limiar (LUGER, 2013; MEDEIROS, 2018). Há também o bias, um valor fixo que pode ser entendido como uma espécie de ajuste fino, que não se multiplica com entrada nenhuma (MEDEIROS, 2018).

O primeiro exemplo de computação neural é o neurônio de McCulloch-Pitts: As entradas de um neurônio de McCulloch-Pitts são excitatórias (+1) ou inibitórias (– 1) (LUGER, 2013). A função de ativação multiplica cada entrada por seu peso correspondente e soma os resultados; se a soma for maior ou igual a zero, o neurônio retorna 1, caso contrário, ele retorna – 1 (LUGER, 2013).. McCulloch e Pitts mostraram como esses neurônios poderiam ser construídos de forma a calcular qualquer função lógica, demonstrando que os sistemas desses neurônios fornecem um modelo computacional completo (LUGER, 2013).

Neste trabalho, foi implementado um neurônio para resolver a porta lógica OU e para resolver a porta lógica E. Primeiramente o neurônio precisa ser treinado para um tipo de operação lógica específica à ser escolhida pelo usuário, cuja escolha é lida do teclado.

O objetivo é, dado um conjunto de exemplos, determinar um vetor de pesos que faça o neurônio em questão, para calcular a porta lógica E ou OU, produzir a saída correta (1 ou 0) para todos os exemplos (isto é, cada conjunto de entradas) fornecido. A Figura 1 contém o algoritmo para execução do neurônio.

```
início
        atribuir pesos aleatórios
        laço
                para cada exemplo fazer
                        executar exemplo
                        se saida incorreta então fazer
                                modificar pesos (w)
                        fim se
                fim para
        repetir até (todas saídas corretas) ou (terminar N épocas)
fim
```

**Figura 1. Algoritmo da execução do neurônio.**

O conjunto de entradas e saídas esperadas pela execução do neurônio estão descritas nas tabelas a seguir (Tabela 1 e Tabela 2):

**Tabela 1. Entrada e saídas da resolução da porta E**

| Entradas | | | Saída Desejada (E) |
|---|---|---|---|
| x0 | x1 | x2 (Bias) | |
| 0 | 0 | -1 | 0 |
| 0 | 1 | -1 | 0 |
| 1 | 0 | -1 | 0 |
| 1 | 1 | -1 | 1 |

**Tabela 2. Entrada e saídas da resolução da porta OU**

| Entradas | | | Saída Desejada (OU) |
|---|---|---|---|
| x0 | x1 | x2 (Bias) | |
| 0 | 0 | -1 | 0 |
| 0 | 1 | -1 | 1 |
| 1 | 0 | -1 | 1 |
| 1 | 1 | -1 | 1 |

## 2. Descrição da implementação e funcionamento do modelo

Na implementação do neurônio, foi utilizada a linguagem de programação C. O desenvolvimento foi feito usando-se o editor de textos Sublime 3 e a IDE Netbeans.

O conjunto de entradas do neurônio foi definido em uma matriz com quatro linhas e três colunas, na qual cada linha da matriz representa uma entrada diferente. Cada linha de entrada é composta por três dados, sendo os dois primeiros 0 ou 1, e o último dígito especificando o bias, sempre com o valor de -1.

Para capturar cada linha de entrada, foi utilizado um loop for, no qual um vetor auxiliar armazena temporariamente cada entrada (cada linha da matriz) e repassa seus valores (por referência) para as funções que calculam o nível de ativação do neurônio, a saída do neurônio, e o erro individual (gerado por cada entrada).

A execução do neurônio se dá pelo somatório de todas as suas entradas multiplicadas por seus pesos. Essa soma ponderada resulta no nível de ativação do neurônio, que é calculado por uma função cujos parâmetros são o vetor da entrada e o vetor de pesos, e o retorno da função é o próprio nível de ativação calculado (LUGER, 2013). Considerando o conjunto de entrada é formado pelos dados x0, x1 e x2 (que

representa o bias), e o conjunto de pesos é formado pelos dados w0, w1, w2, tem-se o cálculo do nível de ativação do neurônio:

Nível de ativação = x0* w0 + x1 * w1 + x2 * w2.

Após o cálculo do nível de ativação, esse valor é passado a função que implementa a função de ativação (ou de limiar) do neurônio, que objetiva definir a ativação do neurônio. Ela calcula o estado final (ou a saída) do neurônio, determinando o quanto o nível de ativação do neurônio está abaixo ou acima de um valor de limiar (LUGER, 2013). A função é do tipo função de limiar abrupto simples, em que uma ativação acima de limiar resulta em um valor de saída 1, ou no caso contrário, de 0. "A função de limiar (hard-limiter) representa uma função de decisão abrupta. Descreve a propriedade tudo-ou-nada de um neurônio de McCulloch-Pitts" (ENGEL, [20–?]). Assim, dados os valores de entrada, os pesos e um limiar, o neurônio calcula seu valor de saída (ou saída intermediária) como 1, se o nível de ativação for maior ou igual ao limiar, e 0 se o nível de ativação for menor que o limiar.

Após o cálculo da saída do neurônio, a função de ativação verifica se a saída calculada é igual à saída desejada (esperada). O neurônio usa uma forma simples de aprendizado supervisionado: após tentar resolver uma ocorrência do problema, é fornecido a ele o resultado correto (como se um professor fornecesse a resposta certa); o neurônio, então, modifica seus pesos de modo a reduzir o erro (LUGER, 2013). Desse modo, se ocorre uma diferença entre o valor da saída calculada e o valor de saída desejado (passado por parâmetro para a função), essa diferença forma o erro individual. O valor desse erro é retornado pela função de ativação. Depois do retorno, o valor é analisado: Se o erro for zero, as saídas (desejada e calculada) são iguais e o neurônio é executado para seu próximo conjunto de entradas; se o erro individual for diferente de zero, uma função é chamada para que o neurônio, então, modifique seus pesos (e os aplique na execução do próximo conjunto de entradas).

A função de modificação dos pesos recebe como parâmetros o vetor de pesos, a taxa de aprendizagem (uma constante de valor pequeno, chamada também de ritmo de aprendizagem), o erro individual e o vetor que contém o conjunto de entradas. A função calcula um novo conjunto de pesos através de uma fórmula. A fórmula para ajustes dos pesos do neurônio artificial deve ser aplicada à todos os pesos do conjunto de entradas caso o resultado calculado da execução do neurônio seja diferente do resultado esperado; Esta fórmula é específica para o treinamento do neurônio artificial (ZAMBIASI, 2018). A fórmula pode ser descrita simplificadamente como:

Novo peso = peso atual + (taxa de aprendizado * erro individual * entrada atual).

Tendo feito ou não os ajustes dos pesos, segue-se para a execução do próximo exemplo (próximo conjunto de entradas e saída esperada correspondente) (ZAMBIASI, 2018). Quando terminar a execução de todos os exemplos (todos os conjuntos de entrada) é completada uma época (ZAMBIASI, 2018).

O algoritmo deve continuar até que não exista mais erros entre a saída calculada e a saída desejada para todos os quatro conjuntos de entrada (e consequentemente, não seja mais necessário ajustar os pesos) (ZAMBIASI, 2018). Se assim ocorrer, diz-se que o neurônio está treinado. Para capturar a não ocorrência de erros em todos os quatro conjuntos de entrada, foi utilizado uma variável contador, que é incrementada cada vez em que a execução da entrada não apresentou erro. Assim, essa variável será igual a quatro (4) se não houver erro para todas as entradas do neurônio (Nesse caso, o aprendizado do neurônio está completo e pode-se interromper o loop que executa cada conjunto de entradas); se a execução de uma das entradas resultar em erro individual, a variável contadora é reiniciada para zero (0). Também foi definido um limite de épocas, a ser especificado pelo usuário pelo teclado, caso o algoritmo não consiga convergir, ou seja, não consiga treinar: Isso previne que o algoritmo fique indefinidamente executando, treinando sem nunca conseguir convergir para o resultado correto (ZAMBIASI, 2018).

Adicionalmente, durante a execução do código, a função “gettimeofday” da biblioteca “sys/time.h” foi utilizada para se obter informações quanto ao tempo de execução em milissegundos necessário para a resolução de cada porta lógica. Para a resolução de ambas as portas, foram escolhidos os pesos 0.7, -0.4 e 0.3 e o limite de 35 épocas, e os tempos de execução em milissegundos foram: 10773.97600 milisegundos e 6789.21000 milisegundos. Essa execução foi realizada no computador com as seguintes configurações: Acer Aspire E1-421-0 BR899, processador AMD E1-1200 APU, memória de 4GB, disco rí gido de 400GB, e sistema operacional Ubuntu LTS versão 16.04.

A implementação se encontra disponível no seguinte link: <https://github.com/joycemayaraa/neural> .

## 3. Referências


ENGEL, Paulo Martins. Modelos de neurônios artificiais – Fundamentos da lógica de limiar. [20–?]. Disponível em: <http://www.inf.ufrgs.br/~engel/data/media/file/cmp121/modelos.pdf >. Acesso em: 12 nov 2018.

LUGER, Gerge F; Aprendizado de máquina: conexionista. In: Inteligência Artificial. 6ª. ed. São Paulo: Pearson Education do Brasil, 2013. P. 375–386.

MEDEIROS, Luciano Frontino de; Introdução a redes neurais artificiais. In: Inteligência Artificial Aplicada: uma abordagem introdutória. 1ª. ed. Curitiba: InterSaberes, 2018. P. 127 – 147.

ZAMBIASI, Saulo Popov. Neurônio Artificial – Implementação. 2018. Disponível em: <https://www.gsigma.ufsc.br/~popov/aulas/rna/neuronio_implementacao/ >. Acesso em: 12 nov 2018.