UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

CONTROLE INTELIGENTE

Aplicação de perceptrons monocamada em circuitos digitais

VITÓRIA
2013

GUILHERME TEBALDI MEIRA

Aplicação de perceptrons monocamada em circuitos digitais

Relatório

Relatório apresentado ao professor Edson Ferreira para avaliação parcial na disciplina de Controle Inteligente, no nono período de Engenharia de Computação do Centro Tecnológico da Universidade Federal do Espírito Santo.

Sumário

Índice de figuras

1-Introdução

Ao longo das disciplinas do curso de Engenharia, são apresentadas técnicas para construção de modelos matemáticos de sistemas do mundo real que permitem seu estudo e solução de problemas. Esses modelos são construídos de forma a aproximar de forma aceitável o sistema físico, adotando simplificações onde for conveniente. Entretanto, por vezes o sistema em estudo não possui um modelo, por apresentar muitas não-linearidades ou simplesmente por seu funcionamento interno ser segredo industrial.

Nessas situações, faz-se necessário lançar mão de técnicas que sejam capazes de se apropriarem de conhecimento a partir de informações de entrada e saída do sistema em estudo e replicá-lo.

As redes neurais são estruturas de dados capazes de realizar a apropriação de conhecimento a partir de um modelo simplificado dos neurônios do cérebro humano. O conhecimento é armazenado nos pesos das sinapses entre os neurônios.

Antes de serem colocadas em operação, as redes neurais devem passar por um estágio de treinamento, onde são apresentadas a conjuntos de dados de entrada e suas respectivas saídas esperadas e, através de algoritmos de treinamento, ajustam seus pesos de forma a representar esse conhecimento.

A rede neural mais simples é o perceptron monocamada, composto apenas de uma camada de entrada e um único neurônio de saída. Apesar de possuir várias limitações conhecidas, esse tipo de rede neural é muito simples de ser implementado computacionalmente, possui um algoritmo simples de treinamento e pode ser aplicado em diversas situações.

Neste trabalho, apresentamos a aplicação do perceptron monocamada para a construção de circuitos digitais, bem como uma implementação construída na plataforma Java.

2-O perceptron monocamada

O perceptron monocamada é um algoritmo de classificação que mapeia diversas entradas em uma saída booleana:

$$f(x) = \begin{cases} 1, se\ w.x + b > 0 \\ 0, caso\ contrário \end{cases}$$

Onde **w** é um vetor de valores reais representando o peso das sinapses e **b** é o *bias*.

Um perceptron monocamada é a forma mais simples de uma rede neural, apresentando uma camada de neurônios de entrada e um único neurônio de saída:

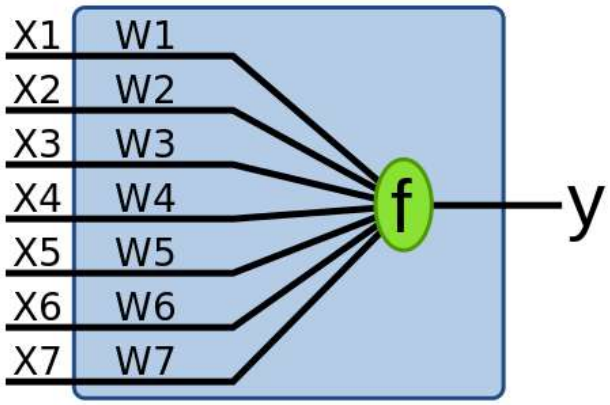


Figura 1-Perceptron monocamada

Antes de começar a operar, o perceptron precisa passar por uma etapa de treinamento, de forma a ajustar os pesos de suas conexões e o *bias*. Para redes neurais mais complexas, existem diversos algoritmos de treinamento, entretanto, para o perceptron monocamada um algoritmo muito simples pode ser empregado:

1. Seja $D = \{(x_1, d_1), \dots, (x_s, d_s)\}$ o conjunto de dados para treinamento, sendo x_j um vetor n-dimensional de entradas e d_j a saída desejada.
2. Seja $y = f(z)$ a saída do perceptron para um vetor z de entrada.
3. Seja b o bias, $w_i(t)$ o valor do peso da i-ésima conexão no instante t e α (entre 0 e 1) a taxa de aprendizado.
4. Inicialize os pesos e a limiar que ativa o neurônio (*threshould*). Os pesos podem ser ajustados para zero ou um valor aleatório pequeno.
5. Para cada exemplo j no conjunto de treino D, faça:
    a. Calcule a saída: $y_j(t) = f\left[w(t).x_j\right] = f[b + w_1(t).x_{j,1} + \cdots + w_n(t).x_{j,n}$
    b. Atualize os pesos: $w_i(t+1) = w_i(t) + \alpha\left(d_j - y_j(t)\right).x_{j,i}$
6. Repita o passo 5 até que o erro da iteração $\frac{1}{s}.\sum_j^s[d_j - y_j(t)]$ seja menor que um limiar ɣ ou o número máximo de iterações seja atingido.

Para o caso de um perceptron de duas entradas, esse algoritmo pode ser visto graficamente como a busca por uma reta que divida as entradas em dois conjuntos:

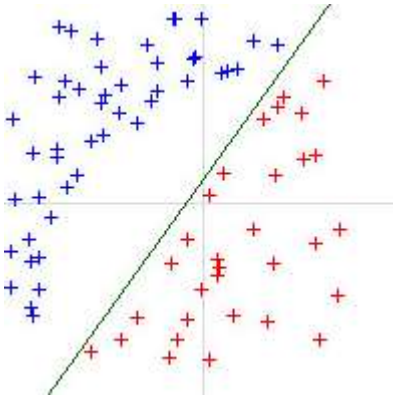


Figura 2-Treinamento do perceptron

Essa visualização gráfica permite perceber uma limitação séria na aplicação dos perceptrons monocamadas: ele só é capaz de classificar as entradas corretamente se existir uma reta que divida o universo de entradas possíveis em dois conjuntos. Matematicamente, isso equivale a dizer que o conjunto de treinamento deve ser **linearmente separável**.

Apesar dessa limitação, o perceptron pode ser aplicado em diversas situações, entretanto, como será mostrado adiante, essa é uma limitação importante quando se está lidando com circuitos digitais.

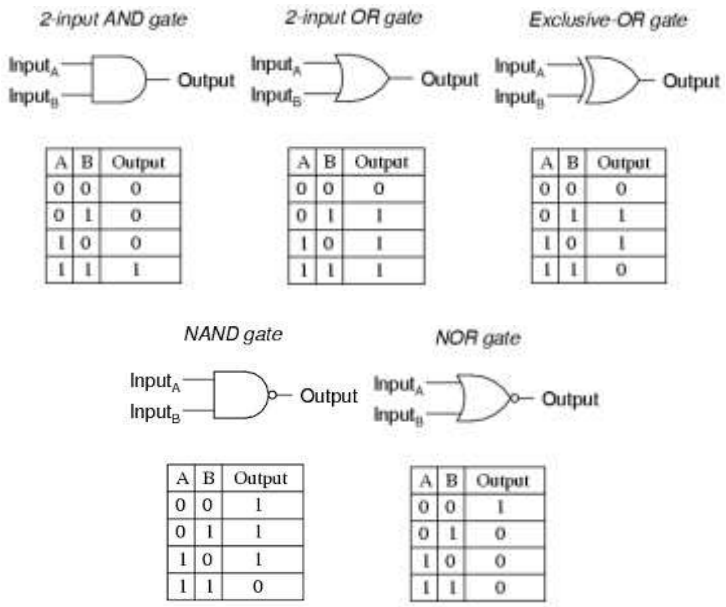
3-Perceptrons aplicados a circuitos digitais

Através do algoritmo mostrado na seção anterior, o perceptron é capaz de aprender e reproduzir um grande número de funções.

Assim, é natural pensar na aplicação de perceptrons para a reprodução de funções lógicas, que são o princípio de funcionamento de qualquer circuito digital.

As funções lógicas são implementadas em circuitos através de **portas lógicas**, que geram suas saídas de acordo com uma **tabela verdade**.

A seguir, mostramos as tabelas verdade de algumas das funções lógicas mais comuns:



A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

Figura 3-Tabelas verdade de funções lógicas comuns

Algumas dessas funções podem ser facilmente representadas por um perceptron, já que é possível encontrar uma reta que separe as entradas que geram saída 1 das que geram saída 0:

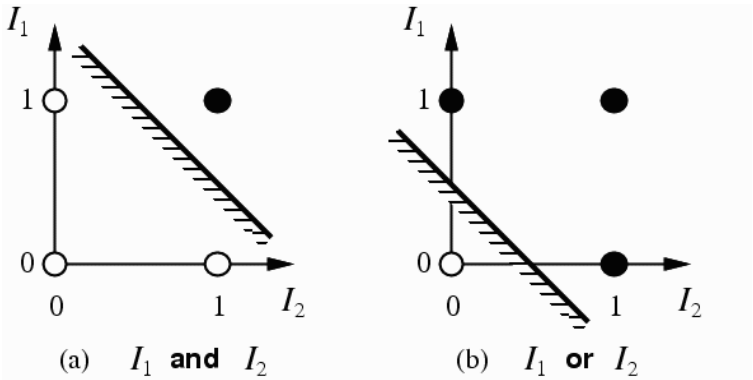


Figura 4-Funções lógicas representadas por um perceptron

Entretanto, no caso da função XOR isso não é possível:

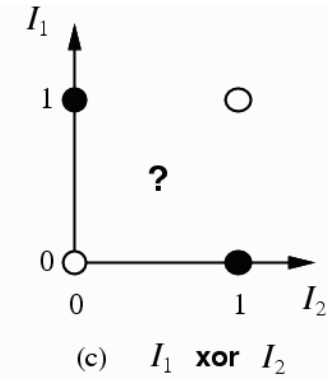


Figura 5-Função XOR em um perceptron

Apesar de ser uma limitação séria do perceptron, ela não impede sua aplicação em circuitos digitais, graças a um fato conhecido da Eletrônica Digital: **toda** tabela verdade pode ser implementada por uma camada de portas AND seguida de uma porta OR.

Como já foi visto acima, um perceptron pode implementar as portas AND e OR, dessa forma, um conjunto de perceptrons pode implementar **qualquer circuito digital**.

4-Somador

Como prova de conceito, vamos implementar um somador de 8 bits substituindo todas as portas lógicas por perceptrons treinados para implementá-las.

Um somador de 8 bits pode ser construído a partir do cascateamento de 8 **somadores completos (full adder)** de 1 bit.

Um somador completo é um circuito digital com três entradas e duas saídas. As duas primeiras entradas representam os dígitos dos dois números a serem somados e a terceira entrada é o chamado **carry-in**, ou **vem-um**, e representa um *overflow* na soma do dígito anterior.

As saídas do somador completo são o valor da soma em si e o **carry-out**, ou **vai-um**, que indica o *overflow* no somador atual e deve ser conectada ao carry-in do próximo estágio.

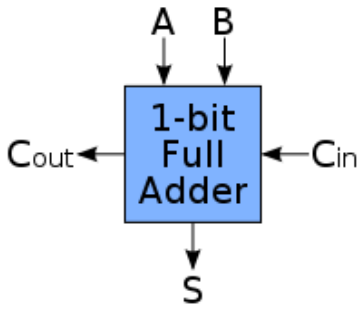


Figura 6-Somador completo

Uma implementação comum do somador completo é a seguinte:

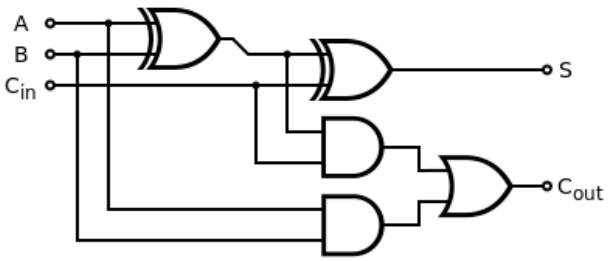


Figura 7-Implementação do somador completo

Entretanto, essa implementação utiliza portas XOR, que o nosso perceptron não é capaz de representar.

Assim, devemos implementar o somador olhando para sua tabela verdade e construindo um circuito com uma camada de portas AND seguida de uma porta OR.

A tabela verdade do somador completo é a seguinte:

Inputs			Outputs	
A	B	C_{in}	C_{out}	S
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0
0	0	1	0	1
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1

Figura 8-Tabela verdade do somador completo

Essa tabela pode ser implementada da seguinte forma:

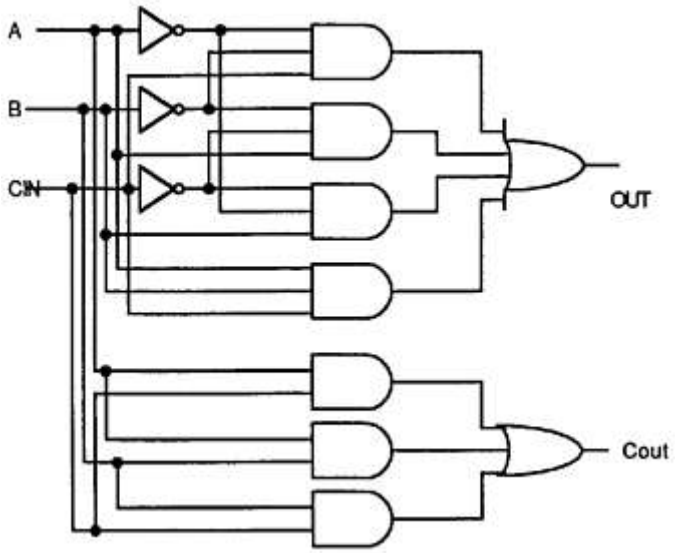


Figura 9-Implementação da tabela verdade do somador completo

Com essa arquitetura, temos um somador completo utilizando somente portas AND e OR, que podem ser substituídas pelos perceptrons.

5-Implementação

Para a implementação computacional deste circuito, foi utilizada a plataforma Java.

Java é uma linguagem desenvolvida pela Oracle que executa sobre a Java Virtual Machine (JVM). Ela foi escolhida para este projeto por ser multiplataforma (o mesmo binário executa em qualquer ambiente que possua uma JVM) e por oferecer nativamente ferramentas para construção de interfaces gráficas com o usuário (GUIs).

Naturalmente, para executar este projeto, é necessário possuir uma instalação da JVM, que pode ser obtida gratuitamente em www.java.com.

O projeto começa com a implementação do perceptron. A classe `Perceptron` implementa um perceptron monocamada com qualquer número de entradas. A classe ainda fornece o método `train`, que realiza o treinamento da rede através de dados de entrada, utilizando o algoritmo apresentado anteriormente.

Um nível acima na abstração, temos as classes `AndGate`, `OrGate` e `NotGate`. Cada objeto dessas classes instancia seu próprio perceptron e o treina oferecendo como entrada a tabela verdade que corresponde à porta lógica desejada. Essas classes podem implementar portas AND, OR e NOT de tamanho arbitrário, gerando sob demanda a tabela verdade no tamanho que for preciso.

No próximo nível de abstração, temos a classe `TruthTableImplementer`, que implementa qualquer tabela verdade utilizando uma camada de portas AND ligadas a uma porta OR. Como as classes que implementam essas portas foram feitas para lidarem com qualquer número de entradas, esta classe também pode implementar tabelas verdade de tamanho arbitrário.

No último nível, temos as classes `Adder` e `Carry`, que utilizam a classe `TruthTableImplementer` para implementar as tabelas verdade dos circuitos de soma e de carry.

Com 8 objetos da classe `Adder` e 7 da classe `Carry`, é possível obter o somador de 8 bits.

Essa arquitetura modularizada permite que partes do sistema possam ser alteradas sem afetar as demais. Por exemplo, as portas AND, OR e NOT poderiam ser implementadas utilizando as operações lógicas da própria linguagem Java em vez de perceptrons e todo o restante da arquitetura poderia ser mantido intocado que o sistema continuaria funcional.

Através da interface do programa é possível alterar as entradas do somador e visualizar em tempo real a resposta.

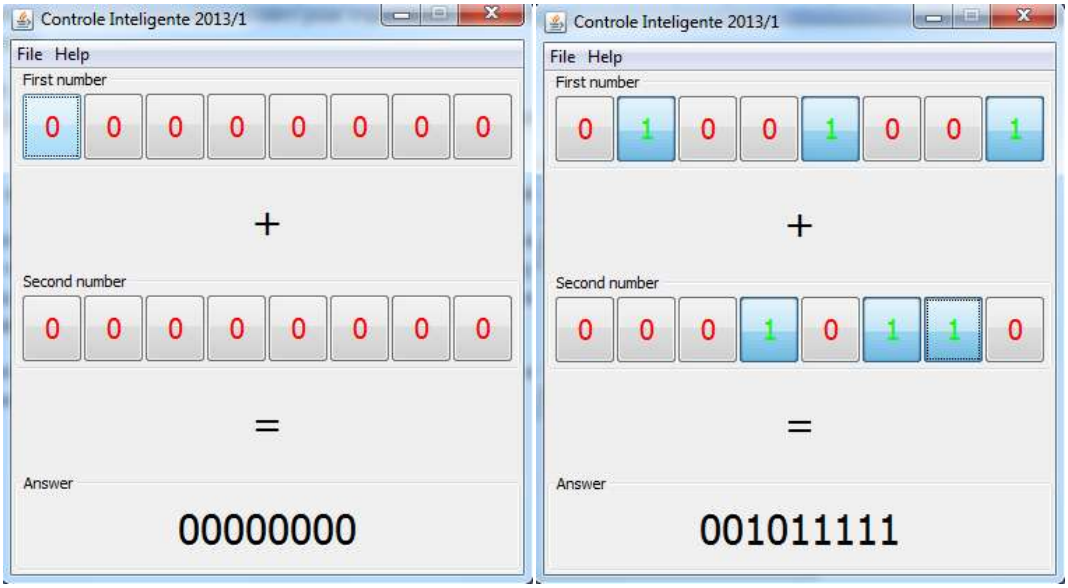


Figura 10-Interface do programa

Todo o código fonte do programa está disponível em http://github.com/guimeira/ci20131.

O código foi todo desenvolvido e comentado em inglês, de forma a facilitar a utilização por qualquer pessoa.

6-Conclusão

Este trabalho apresentou uma aplicação simples de um perceptron monocamada. Apesar da limitação desse tipo de rede neural em relação à porta XOR, mostramos que é possível implementar qualquer circuito digital utilizando esse recurso.

Uma vez terminado o treinamento do perceptron, a tabela verdade pode ser descartada e ele passa a funcionar de forma idêntica à porta lógica desejada, apenas utilizando os pesos obtidos.

É evidente que esse tipo de abordagem é computacionalmente muito menos eficiente que a realização da soma utilizando as instruções do hardware dedicadas a isso. Este projeto serve apenas como uma prova de conceito.

Porém, o poder das redes neurais vai muito além disso. Com redes mais complexas é possível copiar o comportamento de sistemas muito mais sofisticados que as portas lógicas apresentadas aqui.

Além disso, neste trabalho apresentamos um caso em que o treinamento do perceptron foi feito de forma exaustiva, ou seja, todas as entradas possíveis foram apresentadas a ele. Entretanto, esse mesmo perceptron poderia ser utilizado para o caso onde não seja possível gerar todas as entradas possíveis. Com as entradas disponíveis pode ser feito o treinamento e, então, o perceptron tentará obter uma aproximação quando for apresentado a uma entrada desconhecida.