

## 1. Lógica Fuzzy

A aplicação da vida real escolhida para a aplicar a lógica fuzzy foi a de um sistema SIMO (Single Input – Multiple Outputs) para um controle preciso de temperatura de um ambiente, mistura, tanque, etc. O sistema consiste em um sensor de temperatura instalado no ambiente que se quer controlar e de um aquecedor e um arrefecedor (independentes) que atuarão no sistema, para manter a temperatura próxima a um valor desejado.

Foi considerado que tanto o aquecedor, quanto o arrefecedor possuem três estados de funcionamento cada: **desligado**, aquece (resfria) **intermediário**, aquece (resfria) **máximo**, que serão acionados à partir das inferências que o fuzzy irá fazer em cima das regras que possuir.

Implementação:

À partir da temperatura do ambiente  $T_a$ , em comparação a uma temperatura desejada  $T_d$ , obtém-se um erro:

$$E = T_d - T_a$$

Esse erro é mapeado para a variável linguística sensação, que pode assumir os seguintes valores:

- Muito Quente (mqente)
- Quente (quente)
- Ok (ok)
- Frio (frio)
- Muito Frio (mfrio)

A figura abaixo mostra esse mapeamento:

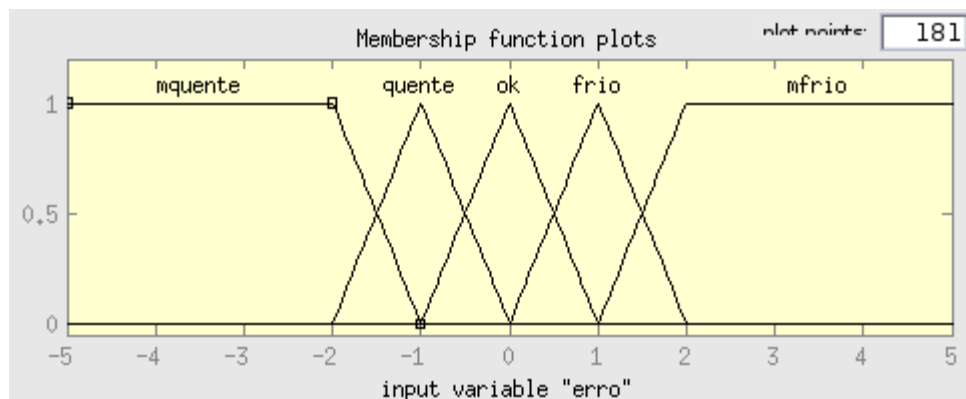


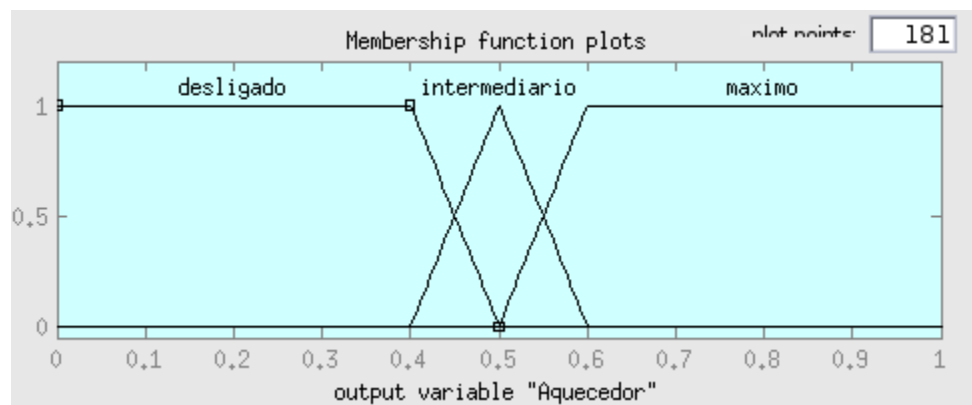
Figura 1: Mapeamento dos valores de Erro para a variável linguística sensação

E à partir dessa sensação, as seguintes regras foram constituídas para a tomada de decisões:

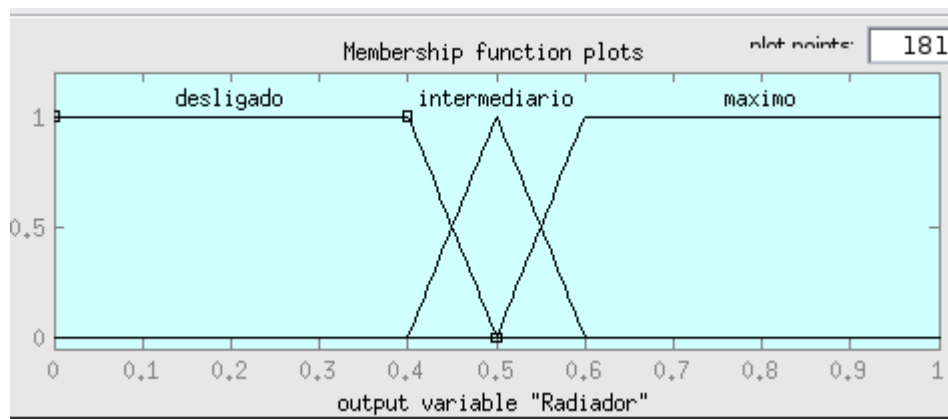
1. If (erro is mquente) then (Aquecedor is desligado)(Radiador is maximo) (1)
2. If (erro is quente) then (Aquecedor is desligado)(Radiador is intermediario) (1)
3. If (erro is frio) then (Aquecedor is intermediario)(Radiador is desligado) (1)
4. If (erro is mfrio) then (Aquecedor is maximo)(Radiador is desligado) (1)
5. If (erro is ok) then (Aquecedor is desligado)(Radiador is desligado) (1)

*Figura 2: Regras para a tomada de decisões*

E as seguintes saídas foram mapeadas tanto para o aquecedor quanto para o arrefecedor.

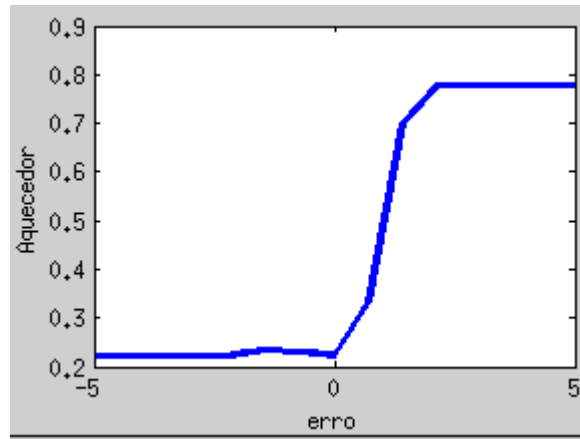


*Figura 3: Aquecedor*

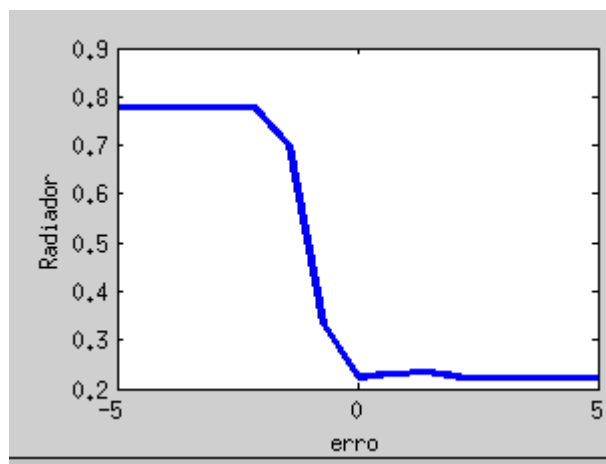


*Figura 4: Arrefecedor*

As superfícies geradas indicando quais ações de controle tomar para estabilizar a temperatura foram essas:



*Figura 5: Ações de controle para o funcionamento do aquecedor*



*Figura 6: Ações de controle para o funcionamento do arrefecedor*

E averiguando as duas superfícies geradas, constata-se que o mecanismo Fuzzy tem o seu funcionamento de maneira adequada, ligando o arrefecedor (aquecedor) ao máximo, quando o erro é muito negativo (positivo) - implicando em uma temperatura do ambiente bem acima (abaixo) da temperatura desejada e desligando os atuadores, quando o erro é próximo de zero. Vale lembrar que para maior eficiência, quando o aquecedor encontra-se em pleno funcionamento, o arrefecedor está desligado e vice versa.

## 2. Perceptron

O perceptron pedido no enunciado para treinar uma rede neural a implementar as funções “E” e “OU” de três entradas foi implementado usando a estratégia de “Regra Delta” ordenada.

O bias ( $x_0$ ) foi considerado como sendo 1 e os pesos iniciais (incluindo  $w_0$  do bias – 4 no total) foram atribuídos de forma randômica, para números no intervalo de -0.5 até 0.5.

A taxa de aprendizado escolhida para a RNA foi de  $\eta=0.20$ .

O Código é executado no matlab usando a function perceptron.m que retorna 7 variáveis de saída que são:

- $x$  – variável de todas as combinações possíveis de 3 entradas para binários (8 possibilidades)
- $and$  – variável da saída da RNA ao final de execução para a função 'AND'
- $or$  – variável da saída da RNA ao final de execução para a função 'OR'
- $w_{and}$  – pesos finais para o treinamento da rede para execução da função 'AND'
- $w_{or}$  – pesos finais para o treinamento da rede para execução da função 'OR'
- $w_{i\_and}$  – pesos iniciais para o treinamento da rede para execução da função 'AND'
- $w_{i\_or}$  – pesos iniciais para o treinamento da rede para execução da função 'OR'.

A implementação deste perceptron necessitou de 115 linhas e abaixo vou colocar os trechos mais importantes do código, que exemplificam o que foi feito.

```
% rna and - treinamento da rede
e = 0; % e - esperado
while e ~= 1 % enquanto nao se alcançam os valores esperados
    e = 1;
    index = randperm(8);
    for i = 1:8
        % selecao de uma entrada aleatoria - por causa do randperm ali em cima
        x_i = horzcat([1], x(index(i),1:3)); % x0, x1, x2, x3 - concatenacao horizontal necessaria, para incluir bias
        y_d = y_d_and(index(i)); % y desejado para a operacao AND
        net = 0;
        for j = 1:4
            net = net + w(j)*x_i(j); % calculo da rede
        end
        if net > 0
            y = 1;
        else
            y = 0;
        end
        error = y_d - y;
        if error ~= 0
            for j = 1:4
                w(j) = w(j) + n*error*x_i(j);
            end
            e = 0;
        end
    end
end
end
```

Figura 7: Treinamento para operação AND (Para a operação OR é feita a mesma coisa, só muda a variável  $y\_d\_and$  para  $y\_d\_or$ )

```

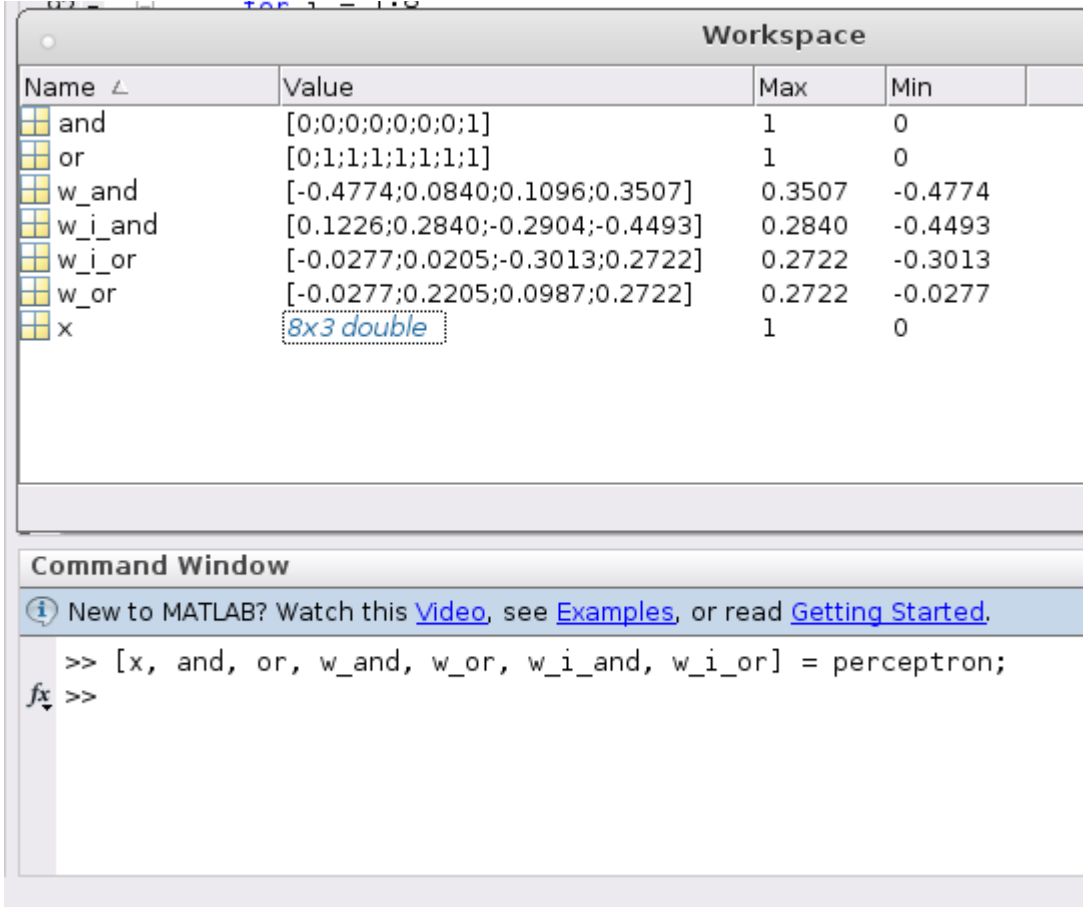
% apos treinamento da rede - averiguacao do funcionamento
for i = 1:8
    x_i = horzcat([1], x(i,1:3));
    net = 0;
    for j = 1:4
        net = net + w(j)*x_i(j); % calculo da rede
    end
    if net > 0
        y = 1;
    else
        y = 0;
    end
    and(i) = y;
end
w_and = w;

```

Figura 8: , Teste da RNA após o treinamento para as mesmas entradas e agora, com a saída sendo armazenada em `and` e os pesos em `w_and`, para uso posterior. (Para a operação `or` é exatamente o mesmo procedimento, trocando-se `and` por `or`)

### Execuções do perceptron:

#### Execução:



The screenshot shows the MATLAB interface with the Workspace and Command Window. The Workspace window displays the following variables:

Name	Value	Max	Min
and	[0;0;0;0;0;0;0;1]	1	0
or	[0;1;1;1;1;1;1;1]	1	0
w_and	[-0.4774;0.0840;0.1096;0.3507]	0.3507	-0.4774
w_i_and	[0.1226;0.2840;-0.2904;-0.4493]	0.2840	-0.4493
w_i_or	[-0.0277;0.0205;-0.3013;0.2722]	0.2722	-0.3013
w_or	[-0.0277;0.2205;0.0987;0.2722]	0.2722	-0.0277
x	8x3 double	1	0

The Command Window shows the following command and output:

```

>> [x, and, or, w_and, w_or, w_i_and, w_i_or] = perceptron;
fx >>

```

### 3. WEKA

#### Parte 1 – Árvores de Decisão

Para execução do algoritmo com opções padrões, os seguintes resultados foram obtidos:

- Tamanho da árvore: 207
- Tx. de Acerto p/ “Percentage Split 66%”: 92.1995%
- Tx. de Acerto p/ “Percentage Split 33%”: 91.5667%

Habilitando “reducedErrorPruning”, os seguintes resultados foram obtidos:

- Tamanho da árvore: 161
- Tx. de Acerto p/ “Percentage Split 66%”: 92.4552%
- Tx. de Acerto p/ “Percentage Split 33%”: 89.0042%

Uma leve melhor ocorreu na “Percentage Split 66%” ao mesmo tempo que a “Percentage Split 33%” teve uma leve piora. Porém, essas mudanças não causam tanto impacto. Entretanto, o tamanho da árvore sofreu uma grande redução por conta de ser feita uma poda onde cada nó é substituído pela sua classe mais popular, mantendo as mudanças que não afetem a eficácia da predição.

#### Parte 2 – Bayes Ingênuo

Para execução do algoritmo com opções padrões, os seguintes resultados foram obtidos:

- Tx. de Acerto p/ “Percentage Split 66%”: 78.0051%
- Tx. de Acerto p/ “Percentage Split 33%”: 78.0409%

Habilitando “UseSupervisedDiscretization”, os seguintes resultados foram obtidos:

- Tx. de Acerto p/ “Percentage Split 66%”: 90.3453%
- Tx. de Acerto p/ “Percentage Split 33%”: 89.9124%

Uma melhora significativa ocorreu ao usar a supervisão discretizada. Isso ocorreu, pois a dimensão do problema foi reduzida e como o Naive Bayes é sensível à dimensão dos problemas, com um problema de dimensão menor, ele apresentou desempenhos melhores.

### Parte 3 – Redes Neurais

A tabela abaixo apresenta os resultados dos treinamentos das redes.

Tempo de Treino	n	Unidades na Camada Oculta					
		5		10		20	
100	0.1	1.8 s	89.3223 %	3.39 s	88.2353 %	6.25 s	88.8107 %
	0.3	1.8 s	85.8696 %	3.35 s	89.5141 %	6.27 s	88.8107 %
300	0.1	5.27 s	90.7928 %	9.78 s	89.1944 %	18.7 s	89.6419 %
	0.3	5.27 s	89.3223 %	9.74 s	90.0895 %	18.69 s	89.5141 %
500	0.8	8.86 s	93.0307 %	16.27 s	87.3402 %	31.34 s	88.4271 %
	1	8.95 s	85.422 %	16.31 s	87.4041 %	31.86 s	90.601 %

*Tabela 1: Treinamento das RNAs para  $n = 0.1$ ,  $n = 0.3$  e  $n = 0.1$ ,  $n = 0.8$ . Tempo de treino: 100, 300 e 500. Camadas Ocultas: 5, 10 e 20*

Para tempo de treino igual a 100, a melhor RNA foi a de  $n = 0.3$ , 10 camadas. Porém, esse valor é insignificamente maior que os obtidos nos outros treinos. Analisando os resultados obtidos para tempo de treino igual a 300, a melhor RNA foi a de  $n=0.1$ , 5 camadas. A RNA  $n=0.3$ , 10 camadas foi a segunda melhor, um pouco melhor que para  $n=0.1$ . Mas tais melhoras são insignificantes.

Após isso, realizou-se outros experimentos com tempo de treino 500 e taxas de aprendizado de 0.8 e 1. A melhor rna observada foi a de 0.8, 5 camadas. Essa melhora foi aproximadamente 3% superior às obtidas anteriormente.

Comparando essa melhora de 3% em relação às obtidas anteriormente - que não passavam de 1% - poderia-se concluir que aumentando o tempo de treino e a taxa de aprendizado, os resultados tenderiam a se aprimorar sempre, entretanto, essa condição não foi válida para  $n = 1$ .

#### Referências:

[1] Stuart. J. Russel and Peter Norvig. Artificial Intelligence: A modern approach. Pearson Education, 2003.

[1] <http://www.cs.waikato.ac.nz/ml/weka>, acessado em 07/12/2015