PROJECTO DE CONTROLADORES DIFUSOS COM APRENDIZAGEM RÁPIDA

Nuno Gonçalves

Instituto de Sistemas e Robótica / Departamento de Engenharia Electrotécnica
Faculdade de Ciências e Tecnologia - Universidade de Coimbra, Portugal
Pinhal de Marrocos - Polo II - 3030 Coimbra
nunogon@isr.uc.pt

1. INTRODUÇÃO

A forma de conhecimento utilizada pela humanidade para o desenvolvimento da matemática e de outras ciências exactas foi, durante séculos, até ao final da primeira metade do século XX, um conhecimento dualista, isto é, baseado numa bipolarização da verdade - sim/não, verdadeiro/falso, 0/1. Na matemática não havia, e ainda hoje não há em muitas áreas, a noção do "em parte". É neste contexto, e devido à necessidade de uma mudança no paradigma do conhecimento que surge o designado conhecimento difuso e a lógica difusa (*fuzzy logic*). Utilizada actualmente em inúmeras aplicações práticas industriais e não só, a lógica difusa é um meio eficaz de englobar conhecimento pericial empírico partindo de uma sistematização do uso da linguagem corrente, bem como conhecimento proveniente da observação, isto é, permite aproveitar observações efectuadas para "treinar" os modelos da realidade.

Nas últimas décadas tem surgido um conjunto vasto de teorias e aplicações práticas que tornam a lógica difusa num dos temas mais investigado na actualidade, sendo aplicável a maioria da teoria clássica da lógica; a lógica difusa é só por si um campo teórico vasto. Uma das aplicações mais importantes é ao nível do controlo, demonstrando capacidades importantes de adaptação e aprendizagem.

Um outro campo de investigação muito relavante, que foi primeiramente compreendido pela avançada civilização tecnológica do Japão, e que foi rapidamente introduzida nas aplicações industriais e de controlo são as **redes neuronais**. São estruturas de neurónios artificiais em rede baseadas nas estruturas do cérebro humano e que, por serem genético-biologicamente inspiradas têm tido sucesso tanto ao nível da investigação como ao nível do desenvolvimento (I&D).

Como veremos adiante, é possível, por um lado, aliar as capacidade linguísticas e de interpretabilidade da lógica difusa com as excelentes capacidades de aprendizagem das redes neuronais.

Este trabalho surge no âmbito da disciplina **Controlo Inteligente** do Mestrado em Sistemas e Automação, da Faculdade de Ciências e Tecnologia da Universidade de Coimbra. Resulta de um estudo da publicação "*Designing fuzzy controllers by rapid learning*" de Jianwei Zhang e Alois Knoll [Zha99]. Este trabalho pretende ser um resumo explicado do *paper* citado bem como apresentar um exemplo prático onde foram aplicados os métodos propostos no original.

Na secção 2 serão apresentados alguns conceitos introdutórios sobre o controlo difuso para melhor compreensão do seguinte. A secção 3 apresentará as funções do tipo B-spline, propostas no *paper* original para o desenvolvimento das funções de pertença dos controladores. Nesta secção serão também introduzidos os conceitos de *core support points* e de *knots*. Seguir-se-á na secção 4 a implementação das funções do tipo B-spline usando a ferramenta Matlab e a discussão dos resultados. Na secção 5 serão apresentados os passos necessários para a construção de um controlador difuso. Na secção 6 será discutida a metodologia proposta para aprendizagem dos parâmetros do controlador e será apresentado um caso prático na secção 7. Finalmente, terminará com uma breve conclusão e reflexão crítica na secção 8. Em apêndice será incluído o código completo em Matlab das funções usadas para a construção do controlador difuso.

2. CONCEITOS INTRODUTÓRIOS

A forma e altura dos conjuntos difusos utilizados para modelar os termos linguísticos não são relevantes para todas as possíveis aplicações da lógica difusa. Contudo, se um sistema de lógica difusa for aplicado em problemas de modelação e controlo, os parâmetros que descrevem os conjuntos difusos são usados directa e indirectamente no processo de inferência e na desfuzificação. Por este motivo, vale a pena estudar com maior detalhe as características dos conjuntos difusos tanto na parte IF como na parte THEN, isto é, tanto na parte do antecedente como no consequente.

parte IF

O espaço de cada entrada é dividido em regiões que se intersectam (*aliasing*), pois representam o conhecimento vago dos termos linguísticos. A parte do antecedente de uma regra é normalmente modelada usando a conjunção e/ou disjunção de várias condições. Temos, contudo, habitualmente a conjunção como meio de agregação das regras:

IF (x_1 is $A^1_{i_1}$) **and** (x_2 is $A^2_{i_2}$) **and** ... (x_n is $A^n_{i_n}$)

onde x_j é a entrada j e $A^j_{i_j}$ é o i-ésimo termo linguístico definido no espaço da entrada x_j.

A operação de conjunção ou agregação é usada nos casos em que há mais do que uma condição (caso geral) e é implementada por um operador do tipo **t-norm**. Um operador **t-norm** é um mapeamento entre dois valores de pertença e um valor de pertença (saída) - T:[0,1]×[0,1]→[0,1] - que respeita as propriedades de comutatividade, associatividade e monotonicidade. Além disso, deve respeitar a condição de fronteira T(a,1)=a. Os operadores do tipo t-norm mais utilizados na agregação/conjunção de condições são o **min** (tal que $T(a,b) = min(a,b)$) e o **produto** (tal que $T(a,b) = a \times b$).

Em relação à forma (*shape*) das funções de pertença utilizadas para o modelo dos termos linguísticos, têm sido utilizadas muitas mas, a maioria tem a particularidade de necessitar de mais parâmetros além da posição no intervalo do espaço das entradas, designados por nós (*knots*). Uma vez que os únicos parâmetros intrínsecos resultantes da divisão do espaço das entradas em vários sub-intervalos são os nós (fronteiras dos sub-intervalos), a selecção e sintonização daqueles parâmetros adicionais não é natural nem intuitiva. Um exemplo bem conhecido dum parâmetro deste tipo é o desvio padrão de uma função de Gauss.

parte THEN

Há vários tipos de controladores difusos. Os mais conhecidos são o de Mamdani e o Takagi-Sugeno-Kang (TSK).

- Controlador tipo **Mamdani** - é baseado na ideia do uso directo de regras simbólicas para tarefas de controlo diversas. Uma regra deste tipo tem a forma:

  IF (x_1 is $A^1_{i_1}$) **and** ... (x_n is $A^n_{i_n}$)THEN y is B_k

  onde B_k é um conjunto difuso com as mesmas propriedades do que os usados na parte IF. A agregação do valor total para a saída, resultante das várias regras disparadas, pode ser calculada através de vários processos. Os mais usados são o operador máximo ou adição simples.

- Controlador tipo **Takagi-Sugeno-Kang** - é um tipo de controladores muito usado para **aproximação de funções** e **aprendizagem supervisionada**. Uma regra difusa usando um modelo TSK de ordem 1 tem a forma:

  IF (x_1 is $A^1_{i_1}$) **and** ... (x_n is $A^n_{i_n}$)
  THEN $y = f(x_1, ..., x_n) =$
  $= a^i_0 + a^i_1 x_1 + ... + a^i_n x_n$

  onde a^i_0, a^i_1, ..., a^i_n são os coeficientes de um modelo linear local. A forma mais comum de optimizar estes coeficientes é através dum método de optimização pelos mínimos quadrados usando dados de treino.

  Uma das desvantagens deste tipo de controladores é o facto de não utilizar todas as potencialidades da lógica difusa pois os coeficientes da função linear não são obtidos de forma intuitiva, isto é, não são directamente retirados do discurso pericial. Isto colola ainda questões de interpretabilidade, isto é, um controlador TSK de ordem igual ou superior a 1 atribui à saída um valor crespo que resulta de um polinómio. Estes controladores não são portanto directamente interpretáveis. Já os controladores TSK de ordem 0 atribuem à saída um valor crespo definido (*singleton*) cuja interpretabilidade é já aceitável.

  A questão da interpretabilidade é relevante na medida em que está na base da filosofia da lógica difusa. Contudo, em muitos casos o princípio da possibilidade de interpretação linguística está a ser ultrapassado pela necessidade de generalizar os modelos com vista à capacidade de aproximação de funções [Pai99].

O *paper* em análise propõe um controlador em que a parte IF (antecedente) é construída de forma sistemática através da divisão do espaço das entradas e em que as acções de controlo da parte THEN (consequente) podem ser adaptadas através de um processo de aprendizagem, baseado num controlador tipo TSK de ordem 0. O modelo proposto para os termos linguísticos é baseado em funções tipo B-spline. Na secção seguinte será explicada esta classe de funções.

3. CLASSE DE FUNÇÕES B-SPLINE

O desenvolvimento da base de regras difusas de um modelo é um problema de interpolação. Vários métodos (Lagrande, Newton, ...) podem ser usados para interpolar os dados disponíveis mas, na sua maioria há uma dependência entre a quantidade dos dados de interpolação e a ordem das funções de aproximação.

Na classe de funções B-spline, proposta para o modelo dos termos linguísticos, a ordem das funções de base é independente dos dados de interpolação, o que se traduz numa vantagem importante para as aplicações difusas.

Expressão analítica

Seja x uma entrada e seja o espaço da entrada x o intervalo $[x_0, x_p]$. As funções da classe B-spline podem ser unicamente parametrizadas através dos extremos (pontos fronteira) dos sub-intervalos decorrentes da divisão do espaço de entrada. Estes parâmetros são designados por *knots* (nós, em português).

Sejam, os *knots* dados por $x_0, x_1, ..., x_p$ e seja k a ordem das funções B-spline.

As funções B-spline, normalizadas são dadas pela seguinte expressão analítica:

$$X_{i,k}(x) = \begin{cases} \begin{cases} 1 & para \;\; x_i \leq x < x_{i+1} \\ 0 & outros \;\; casos \end{cases} & para \;\; k = 1 \\ \frac{x-x_i}{x_{i+k-1}-x_i} X_{i,k-1}(x) + \frac{x_{i+k}-x}{x_{i+k}-x_{i+1}} X_{i+1,k-1}(x) & \text{para } k > 1 \end{cases} \tag{1}$$

com $i = 0, 1, ..., m-k$, isto é, para p intervalos entre os *knots* e para funções de ordem k, há $m-k$ funções B-spline e portanto termos linguísticos.

Pontos principais de suporte e nós

Os pontos principais de suporte (*core support points* - CSP) são os pontos com mais elevado valor de pertença de um conjunto difuso. Para as funções da classe B-spline, há apenas um pico, pelo que esse é o *core support point* dessa função de pertença.

Se tivermos uma função tipo B-spline a representar um conjunto difuso, esta função é definida pelos extremos do suporte[1] do conjunto difuso, isto é, pelos *knots*.

Os *knots* podem dividir-se em dois conjuntos: os *knots* internos - **Iknots**, isto é, aqueles que pertencem ao espaço das entradas (o intervalo $[x_0, x_m]$) e os *knots* externos - **Eknots**, isto é, aqueles que não pertencem ao espaço das entradas. Os *Eknots* são usados para modelar os termos linguísticos marginais.

O número de *Iknots* usados é $m-(k \bmod 2)$, onde m é o número de termos linguísticos e k é a ordem das funções de aproximação. Usualmente temos $k \leq m$.

Para calcular os *Iknots* usamos as seguintes regras:

- k é par - $Iknot_i = CSP_i$ com $i = 0, 1, ..., m$
- k é ímpar - $Iknot_i = CSP_i + \frac{CSP_{i+1}-CSP_i}{2}$ com $i = 1, ..., m-1$

Quanto ao cálculo dos *Eknots* podemos usar a regra: em cada um dos extremos do intervalo (espaço das entradas) há $(k+1)$ div 2 *Eknots* que podem ser calculados espelhando os correspondentes *Iknots* (os mais próximos do extremo) em relação ao próprio extremo (x_0 ou x_m).

Ao todo esta formulação tem $m + k$ *knots*.

[1] O suporte de um conjunto difuso é o intervalo no qual o valor da pertença é diferente de zero.

4. UM CASO PRÁTICO

O uso da ferramenta Matlab para o teste do exposto até esta parte do trabalho foi muito importante na própria compreensão do seu conteúdo, pois os conceitos de *core support point* e de *knot* não são intuitivos e surgem algumas dúvidas quanto à sua implementação quando abordado de forma mais superficial.

Consideremos que temos uma função $f(x)$ que queremos aproximar usando um controlador difuso e que queremos usar cinco (5) termos linguísticos reais no intervalo de entrada. Usamos mais dois termos linguísticos virtuais (para as partes marginais do intervalo). Este esquema será o usado na parte final do trabalho pelo que faz todo o sentido começar por este exemplo.

Seja o espaço da entrada x o intervalo [0,1]. Para desenvolver as cinco funções B-spline para os termos linguísticos mais as duas funções B-spline para os termos linguísticos virtuais, temos 5 *core support points* reais, no intervalo [0,1] e mais dois CSPs fora do intervalo.

Os nossos CSPs são, portanto, os primeiros parâmetros a fixar e terão, então, os valores:

- CSPs reais: 0.00; 0.25; 0.50; 0.75 e 1.00
- CSPs virtuais: -0.25 e 1.25

Consideremos ainda que queremos usar interpoladores de terceira ordem, isto é, $k = 3$. Uma vez que temos sete intervalos, o valor de m é 6.

Uma vez que $k = 3$ é um valor ímpar, os *Iknots* são calculados como os pontos médios entre os CSPs, pelos que os seus valores são:

- $Iknot_0 = -0.125$
- $Iknot_1 = 0.125$
- $Iknot_2 = 0.375$
- $Iknot_3 = 0.625$
- $Iknot_4 = 0.875$
- $Iknot_5 = 1.125$

por seu lado, os *Eknots* são os $(k+1)$ div $2 = 2$ *Iknots* mais próximos de cada extremo espelhado em relação ao extremo. São dados por:

- $Eknot_0 = CSP_0 - (Iknot_0 - CSP_0) = -0.375$
- $Eknot_1 = CSP_0 - (Iknot_1 - CSP_0) = -0.625$
- $Eknot_2 = CSP_1 + (CSP_1 - Iknot_5) = 1.375$
- $Eknot_3 = CSP_1 + (CSP_1 - Iknot_4) = 1.625$

Desta forma, os pontos extremos dos sub-intervalos - *knots* pertencem ao conjunto {-0.625; -0.375; -0.125; 0.125; 0.375; 0.625; 0.875; 1.125; 1.375; 1.625} e correspondem aos únicos parâmetros usados para o cálculo das 7 funções B-spline que representam os termos linguísticos (5 reais e 2 virtuais).

Para calcular agora as funções B-spline, temos nove intervalos, $p = 9$, isto é, $x_0 = -0.625$ e $x_p = x_9 = 1.625$.

Como exemplo, a função $X_{3,3}(x)$, isto é, a função de pertença do quarto termo linguístico de ordem 3 tem a forma:

$$X_{3,3}(x) = \begin{cases} 8x^2 - 0.125 & 0.125 \leq x < 0.375 \\ -16x^2 + 10x + 2 & 0.375 \leq x < 0.625 \\ 8x^2 - 14x + 6.125 & 0.625 \leq x < 0.875 \end{cases} \tag{2}$$

De seguida apresenta-se o código em Matlab para calcular o valor das funções de classe B-spline em qualquer ponto, dados apenas como parâmetros os *knots*, a ordem da função e o sub-intervalo.

```
function out = bspline(i,k,x,knots);

%i    --> termo linguístico
%k    --> ordem da função
%x    --> valor onde deve ser calculada
%         a funcao
%knots --> valores em que se subdividem o
%         intervalo - p.ex. knots=0:1/8:1

if (k==1)
   if (x>=knots(i+1)) & (x<knots(i+1+1))
      out=1;
   else
      out=0;
   end
else
   %calcula o valor recursivo de X_i_(k-1)
   out1=bspline(i,k-1,x,knots);
   out2=bspline(i+1,k-1,x,knots);

   out=(x-knots(i+1))/(knots(i+k-1+1)-
       knots(i+1))*out1+(knots(i+k+1)-x)/
       (knots(i+k+1)-knots(i+1+1))*out2;
end
```

As figuras 1 a 4 mostram os resultados obtidos para as funções de classe B-spline, do primeiro termo linguístico (a vermelho, traço contínuo) ao sexto termo linguístico (a azul, traço descontínuo), de ordem 1 à ordem 4 ($k = 1, ..., 4$).

5. O CONTROLADOR

Os sistemas podem classificar-se quanto ao número de entradas e saídas como SISO, SIMO, MISO e

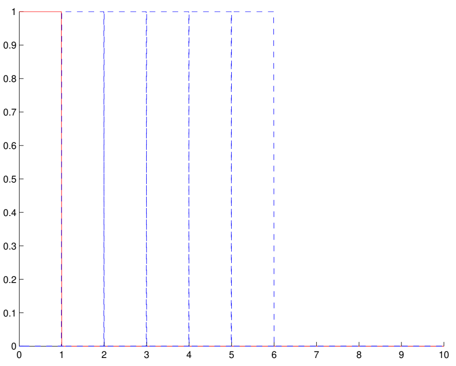

Figura 1: Funções B-spline de ordem 1

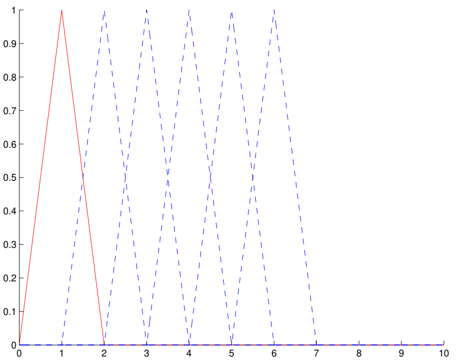

Figura 2: Funções B-pline de ordem 2

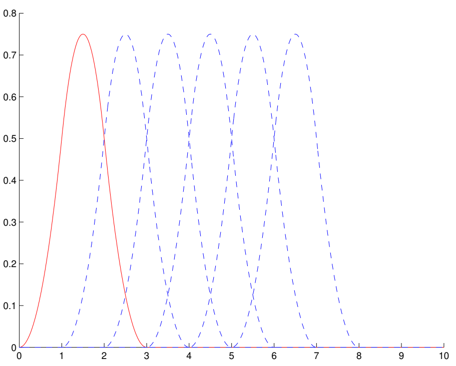

Figura 3: Funções B-spline de ordem 3

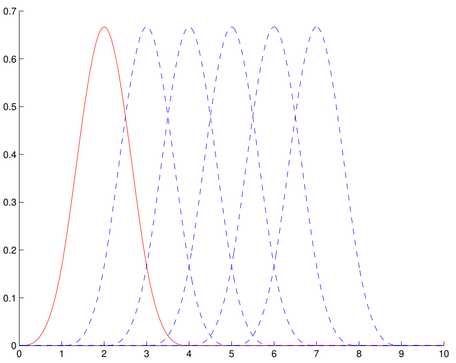

Figura 4: Funções B-spline de ordem 4

MIMO quando têm uma única (S) entrada (I) ou várias (M) e/ou uma única saída (O) ou várias. A forma geral é MIMO, isto é, várias entradas e várias saídas. Contudo, um sistema MIMO pode ser, no caso do controlo, modelado como vários problemas com várias entradas e uma única saída (MISO), tantos quantas as saídas.

No contexto de sistemas MISO, o *paper* analisado apresenta um conjunto de condições para que a saída de um controlador difuso seja equivalente à saída de uma hiper-superfície B-spline geral, isto é, o uso de funções da classe B-spline em controladores difusos é válido. Estas condições são:

- As funções de pertença para as entradas são funções da classe B-spline.
- Os funções de pertença para as saídas são *singletons*, isto é, à saída é atribuído um valor "crespo" em vez de um valor de pertença de um conjunto difuso de saída.
- Para resolver as conjunções de condições é usado o operador "produto".
- O método de desfuzificação é o "centróide", isto é, a saída real é o "centro de massa" dos valores atribuídos à saída por cada uma das regras disparadas.
- É usado um termo linguístico virtual em cada um dos extremos do intervalo do espaço das entradas.
- A base da regra é estendida para os termos linguísticos virtuais copiando os valores da saída da vizinhança mais próxima.[2]

Para um sistema de n entradas e uma saída, a forma geral das regras é:

$$\text{IF } (x_1 \text{ is } A^1_{i_1 k_1}) \textbf{ and } \ldots\ (x_n \text{ is } A^n_{i_n k_n}) \\ \text{THEN } y = Y_{i_1 i_2 \ldots i_n}$$

onde

- x_j é a j-ésima entrada $(j = 1, ..., n)$.
- k_j é a ordem da função B-spline usada para a entrada x_j.
- $X^j_{i_j k_j}$ é o i-ésimo termo linguístico de x_j definido por uma função de pertença B-spline.
- $i_j = 1, ..., m_j$ representa o número de subdivisões do espaço da entrada x_j.
- $Y_{i_1 i_2 \ldots i_n}$ é o vértice de controlo, também chamado ponto de *De Boor*. É o valor atribuído à saída por esta regra se ela disparar, isto é, se o antecedente for verdadeiro.

Usando estas condições, e o método do centróide para desfuzificação, temos que a saída final é dada pela expressão:

$$y = \sum_{i_1=1}^{m_1} \ldots \sum_{i_n=1}^{m_n} \left(Y_{i_1 i_2 \ldots i_n} \prod_{j=1}^{n} X^j_{i_j k_j}(x_j) \right) \tag{3}$$

Construção do controlador

A construção do controlador requer um conjunto de passos que serão descritos neste parágrafo. Posteriormente, explicar-se-ão alguns dos passos mais detalhadamente. A letra M significa que o passo deve ser feito manualmente e a letra A automaticamente.

1. Seleccionar as entradas (M/A)
2. Escolher a ordem das funções de classe B-spline que se pretende usar para cada uma das entradas independentemente. (M/A)
3. Determinar os *core support points* - CSPs partindo o espaço das entradas. (M/A)
4. Calcular as funções de pertença B-spline dos termos linguísticos reais e virtuais para todas as entradas. (A)
5. Inicializar os vértices de controlo para a saída. (A)
6. Adaptar os vértices de controlo. (A)
7. Se os resultados são satisfatórios, terminar. (A)
8. Senão,
   - Modificar os CSPs para as entradas (M/A), ir para o passo $< 4 >$
   - Refinar a granularidade e usar mais dados de treino (M/A), ir para o passo $< 3 >$
   - Aumentar a ordem das funções B-spline. (M/A), ir para o passo $< 3 >$
   - Apagar algumas entradas e/ou adicionar outras. (M/A), ir para o passo $< 2 >$.

Informação pericial

Os passos $< 3 >$ e $< 6 >$ são de grande importância na performance do controlador difuso. Tantos os *core support points* como os *control vertices* podem ser retirados, pelo menos numa primeira aproximação, da informação obtida por pessoas experientes, peritos.

[2] No teste prático efectuado para a aproximação de uma função, verifiquei que se houver um valor da saída próprio para os termos linguísticos virtuais, os resultados são melhores.

Quanto aos *core support points*, estes devem, de um modo geral, ser colocados em pontos onde a saída tem extremos. Se esta informação estiver disponível, normalmente proveniente de informação pericial, é possível melhorar a rapidez de aprendizagem do controlador. Se, por outro lado, a informação não estiver disponível, a melhor opção é colocar os CSPs uniformemente distribuído no intervalo do espaço da entrada correspondente.

Os vértices de controlo (*singletons* - $Y_{i_1 i_2 \ldots i_n}$) podem também ter uma primeira aproximação proveniente de informação pericial, quando disponível. Quando esta informação não existir, será razoável inicializar os seus valores a zero e posteriormente fazer a sua adaptação.

Na secção seguinte serão mostrados os métodos propostos para a adaptação dos CSPs e dos vértices de controlo.

Pesagem e aceleração

A importância das regras pode não ser uniforme e, para isso é possível atribuir um peso específico a cada uma das regras. A parte IF fica inalterada mas é necessário alterar o método de cálculo da saída (*singleton*). O cálculo do centróide passa agora a ser uma média ponderada e não aritmética.

O uso das funções de classe B-spline além de sistematizar as funções de pertença, pode conduzir ao aumento da performance do método, pois para cada vector de entrada $\mathbf{x}$, há exactamente k termos linguísticos, isto é, o número de regras que disparam é exactamente k, sendo então apenas necessário calcular o valor de pertença para k funções B-spline. Para um sistema com n entradas, o número de regras disparadas é k^n, um número muito inferior a m^n, pelo que o esforço computacional necessário é muito menor.

6. ADAPTAÇÃO E APRENDIZAGEM DOS PARÂMETROS

É nesta secção que pode ser de alguma utilidade considerar o nosso problema como um problema neuro-difuso. Na verdade, podemos considerar o controlador difuso como um sistema difuso representado por uma arquitectura em rede, e aproveitar as capacidades de aprendizagem e adaptação das redes neuronais. De seguida serão apresentados os métodos para "aprender" os vértices de controlo (parâmetros da saída) e adaptar o posicionamento dos CSPs (parâmetros da entrada). Para maior detalhe sobre os sistemas neuro-difusos, ver p.ex. [Pai99].

Aprendizagem dos vértices de controlo

O método usado para a optimização dos parâmetros de um controlador difuso com a forma descrita na secção anterior é o método dos gradientes decrescentes. Assumamos que $\{(\mathbf{X},y_d)\}$ é um conjunto de dados de treino, onde $\mathbf{X} = (x_1, x_2, ..., x_n)$ é o vector de treino de entrada e y_d é a saída desejada para a mesma entrada.

A métrica do erro usada é a dos mínimos quadrados, dada pela expressão:

$$E = \frac{(y_r - y_d)^2}{2} \tag{4}$$

onde y_r é a saída real obtida no processo de treino através da equação 3. Os parâmetros que queremos calcular são os $Y_{i_1 i_2 \ldots i_n}$ que minimizam o erro usando a métrica indicada.

Usando o método dos gradientes descendentes e um coeficiente de aprendizagem ϵ, tal que $0 < \epsilon \leq 1$, podemos actualizar os parâmetros em cada iteração de treino da seguinte forma:

$$\begin{aligned} Y^{novo}_{i_1 i_2 \ldots i_n} &= Y_{i_1 i_2 \ldots i_n} + \Delta = & (5) \\ &= Y_{i_1 i_2 \ldots i_n} - \epsilon(y_r - y_d) \prod_{j=1}^{n} X^j_{i_j k_j}(x_j) \end{aligned}$$

Este método garante que o algoritmo de aprendizagem converge para o mínimo global da função de erro uma vez que a segunda derivada do erro em função de $Y_{i_1 i_2 \ldots i_n}$ é constante.

Adaptação dos CSPs

Inserir mais uma função B-spline significa aumentar o número de CSPs e de *knots*, ou seja, aumentar a dimensão do problema. Assim, é conveniente optimizar a utilização dos CSPs de forma a manter a dimensão do problema tão pequena quanto possível e manter a granularidade e a sensibilidade do modelo tão boas quanto possível.

O algoritmo proposto para optimizar os CSPs é uma adaptação de um algoritmo de aprendizagem dos parâmetros de uma rede neuronal auto-organizada.

Ao aplicar um novo vector de entrada de treino $\mathbf{x}$, podemos procurar na sua vizinhança o CSP mais próximo de ambos os lados, a que chamamos $\mathbf{x_l}$ e $\mathbf{x_r}$. Os valores da saída correspondente a estes CSPs são designados por y_l e y_r respectivamente.

O algoritmo estabelece que se o valor da saída y_d referente ao vector de entrada x for muito superior ao valor das saídas dos CSPs vizinhos, então os CSPs necessitam de ser adaptados. É usado um patamar para distinguir os casos - $\theta > 0$.

Sejam $\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_m}$ os CSPs da sequência de dados de entrada e $\mathbf{X_{1,k}}, \mathbf{X_{2,k}}, ..., \mathbf{X_{m,k}}$ os valores de pertença dos termos linguísticos (funções B-spline de ordem k).

O algoritmo de adaptação pode ser descrito da seguinte forma:

- Aplicar um novo vector de entrada de treino e correspondente saída - ($\mathbf{x}$,y_d).
- Encontrar os dois CSPs vizinhos mais próximos ($\mathbf{x_l}$ e $\mathbf{x_r}$) de forma que:

$$\mathbf{x_i} \leq \mathbf{x_l} \leq \mathbf{x} \leq \mathbf{x_r} \leq \mathbf{x_j}$$

  para i=1,...,l-1 e j=r+1,...,m.

  Calcular y_l e y_r.

- – Se $y_l \leq y_r$
    Se $y_d \leq y_l - \theta$ Então modificar $\mathbf{x_l}$ tal que:

$$\mathbf{x_l} = \mathbf{x_l} + (\mathbf{x} - \mathbf{x_l}) \times \mathbf{X_{l,k}}(\mathbf{x})$$

    Se $y_d \geq y_r + \theta$ Então modificar $\mathbf{x_r}$ tal que:

$$\mathbf{x_r} = \mathbf{x_r} + (\mathbf{x} - \mathbf{x_r}) \times \mathbf{X_{r,k}}(\mathbf{x})$$

  – Se $y_l > y_r$
    Se $y_d \geq y_l + \theta$ Então modificar $\mathbf{x_l}$ tal que:

$$\mathbf{x_l} = \mathbf{x_l} + (\mathbf{x} - \mathbf{x_l}) \times \mathbf{X_{l,k}}(\mathbf{x})$$

    Se $y_d \leq y_r - \theta$ Então modificar $\mathbf{x_r}$ tal que:

$$\mathbf{x_r} = \mathbf{x_r} + (\mathbf{x} - \mathbf{x_r}) \times \mathbf{X_{r,k}}(\mathbf{x})$$

- Optimizar os vértices de controlo da variável de saída.[3]
- Continua para < 1 > enquanto houver dados de treino.
- Termina.

7. APROXIMAÇÃO DE FUNÇÕES - UM CASO PRÁTICO

Para a melhor compreensão dos conteúdos propostos no *paper* analisado, desenvolvi com a ajuda da ferramenta Matlab o controlador difuso proposto para aproximação de funções - uma das aplicações mais imediatas do controlo difuso, especialmente quando usado um modelo TSK.

De forma a poder comparar de algum modo os meus resultados para os validar, resolvi usar o algoritmo para aproximar a função unidimensional também usada no *paper*, isto é, a função $sin(2\pi x^2)$, no intervalo [0,1].

[3] Na verdade, este passo, descrito no parágrafo anterior, pode não ser usado em conjunção com este algoritmo de adaptação dos CSPs.

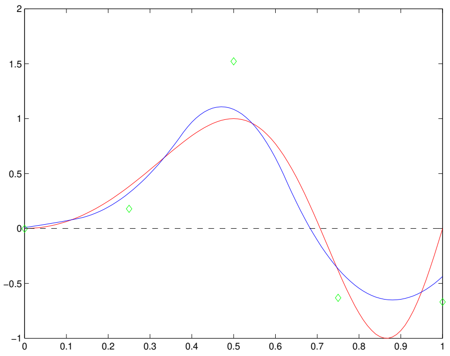

Figura 5: Aproximação da função $sin(2\pi x^2)$ sem adaptação dos CSPs ao fim de 2000 épocas

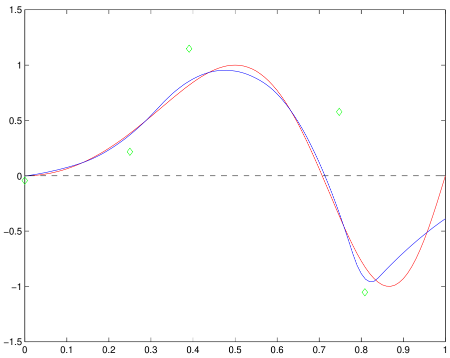

Figura 6: Aproximação da função $sin(2\pi x^2)$ com adaptação dos CSPs ao fim de 2000 épocas

Na secção 4 apresentei já uma das partes fundamentais do processo de construção do controlador para este caso específico: a escolha dos CSPs e dos *knots*. Temos, então 7 termos linguísticos, dos quais dois são virtuais, funções B-spline de terceira ordem e, portanto, 10 *knots*.

O código de implementação do algoritmo é apresentado em anexo. As figuras 5 e 6 mostram respectivamente os resultados do algoritmo sem e com adaptação dos CSPs, para um conjunto de 2000 vectores de treino, isto é, um processo de aprendizagem com 2000 épocas.

Como se depreende dos resultados, o processo de adaptação dos *core support points* é importante pois aumenta a velocidade de convergência dos parâmetros aprendidos. Além disso, é possível perceber que o uso deste método para aproximação de funções dá resultados bastante bons.

8. CONCLUSÃO

O algoritmo proposto no *paper* analisado para o desenho de controladores difusos usando, por um lado um conjunto de funções de classe B-spline e por outro lado um modelo TSK para desfuzificação parece apresentar bons resultados na sua generalidade e, segundo resultados experimentais apresentados, parece ser de elevada rapidez na convergência.

Há vários aspectos que devem ser analisados. Irei primeiramente discutir alguns destes tópicos e posteriormente dar a minha opinião pessoal em relação ao *paper*.

- As **funções da classe B-spline** são fáceis de implementar, tanto a nível analítico como computacional, sendo o esforço computacional necessário aceitável. Estas funções, dada a ordem desejada, apenas dependem dos limites do suporte da função, parâmetros a que chamamos *knots*.

  Estas funções são usadas para modelar os termos linguísticos. Representam as suas funções de pertença.

- A saída do modelo é do tipo TSK de ordem 0, isto é, à saída é atribuído um valor crespo (real) se o antecedente for verdadeiro.

- O facto de termos um modo sistemático para calcular tanto as funções de pertença para os termos linguísticos como para calcular os *singletons*, permite-nos implementar a base de regras de forma automática e com um esforço computacional aceitável.

- Os vértices de controlo, também chamados pontos de *De Boor*, são os valores dos *singletons*, isto é, os valores atribuídos ao consequente. O *paper* propõe um algoritmo de aprendizagem para ajustar os valores dos vértices de controlo.

  É colocada a tónica na aprendizagem supervisionada, através de dados de treino. Este método é particularmente eficiente para o caso de aproximação de funções. Contudo, é possível implementar métodos de aprendizagem não supervisionada, através de funções de avaliação.

- Os *core support points* são os pontos onde as funções de pertença atingem os seus máximos globais. O posicionamento destes pontos é muito importante para a performance do controlador. O *paper* propõe um método para cálculo da configuração óptima de CSPs, com dados de treino.

- Ambos estes métodos de aprendizagem dos vértices de controlo e dos CSPs podem ser englobados num único algoritmo de forma a serem optimizados simultaneamente. Os resultados obtidos são muito bons.

- A colocação inicial tanto dos CSPs como dos vértices de controlo é também muito importante, tanto para o desempenho final do controlador como para a velocidade de aprendizagem. Os valores iniciais podem ser ambos retirados da informação pericial, caso esteja disponível. Intuitivamente os CSPs devem ser colocados nos pontos onde a saída tiver um extremo e os vértices de controlo devem ter valores semelhante aos próprios extremos.

- O facto de a construção da parte IF do controlador estar sistematizada com funções B-spline de ordem k, torna possível calcular o valor da saída para exactamente apenas k *singleton*, pois é o número de regras que vão disparar. Esta característica permite acelerar consideravelmente o modelo.

- Em algumas aplicações o modelo TSK para a saída não é adequado. Nestes casos é possível usar um modelo do tipo *Mamdani*, através de técnicas de *clustering*, agrupando os *singletons* num conjunto de conjuntos difusos.

- Se considerarmos o controlador difuso como um sistema difuso com uma arquitectura em rede, podemos utilizar inúmeras metodologias aplicáveis à aprendizagem dos parâmetros de entrada e saída. Estes sistemas são classificados de neuro-difusos.

Quanto à minha opinião pessoal em relação ao *paper*, penso que está de um modo geral bem escrito e acessível àqueles cujos conhecimentos na área do Controlo Difuso são meramente académicos e não muito aprofundados. Penso que aprendi bastante com a análise deste *paper*, essencialmente na parte da implementação prática do mesmo. Na verdade, as dúvidas de fundo apenas surgem quando é necessário implementar o que se lê.

REFERENCES

[Dou00] Dourado, António. Apontamentos da disciplina de Controlo Inteligente, 2000.

[Kas96] Kasabov, Nikolai. *Foundations of neural networks, fuzzy systems, and knowledge engineering.* MIT Press, 1996.

[Pai99] Paiva, Rui Pedro Carvalho e. Identificação Neuro-Difusa, Aspectos de Interpretabilidade. Master's thesis, DEI-FCT-University of Coimbra, 1999.

[Ram98] Ramos, José Victor Martins. Projecto e Aplicação de Controladores Difusos em Tempo-Real. Master's thesis, DEI-FCT-University of Coimbra, 1998.

[Zha99] Zhang, Jianwei e Knoll, Alois. Designing fuzzy controllers by rapid learning. *Fuzzy Sets and Systems*, (101):287–301, 1999.

CÓDIGO DA IMPLEMENTAÇÃO EM MATLAB DO CONTROLADOR DIFUSO

```
%ponto1
n_random=2000;
%Training data
%vector aleatório
%funcão --> f(x)=sin(2*pi*x*x)
td=zeros(n_random,2);
td(:,1)=random('unif',0,1,n_random,1);
pi=3.141592;
for kk=1:n_random,
   td(kk,2)=sin(2*pi*td(kk,1)*td(kk,1));
end
%ponto 2
n_order=10;
%ponto 3
m=5;
CSP=0:0.25:1;
for ii=1:(m-1),
   Iknots(ii)=CSP(ii)+(CSP(ii+1)-CSP(ii))/2;
end
num=floor((n_order+1)/2)+1;
for jj=1:num,
   Eknots_inf(jj)=CSP(1)-Iknots(num-jj+1);
end
for jj=1:num,
   Eknots_sup(jj)=2*CSP(m)-Iknots(m-1-jj+1);
end
knots=[Eknots_inf';Iknots';Eknots_sup'];
%ponto 5
%control vertices (de Boor points)
Y(1)=0;
Y(2)=0;
Y(3)=0;
Y(4)=0;
Y(5)=0;
Y(6)=0;
Y(7)=0;
%ponto 6
epsilon=0.4;
for sample=1:n_random,
   sample
   x1=td(sample,1);
   yd=td(sample,2);
   for c=1:5,
      if (x1>CSP(c))
         xl=CSP(c);
         l=c-1;
      end
   end
   for c=5:-1:1,
      if (x1<CSP(c))
         xr=CSP(c);
         r=c-1;
      end
   end
   yl=yreal(xl,n_order,knots,5,Y);
   yr=yreal(xr,n_order,knots,5,Y);
   theta=0.2;
```

```
   if (yl<=yr)
      if (yd<=yl-theta)
         xl=xl+(x1-xl)*bspline(l,n_order,x1,knots);
      end
      if (yd>=yr+theta)
         xr=xr+(x1-xr)*bspline(r,n_order,x1,knots);
      end
   else
      if (yd>=yl+theta)
         xl=xl+(x1-xl)*bspline(l,n_order,x1,knots);
      end
      if (yd<=yr-theta)
         xr=xr+(x1-xr)*bspline(r,n_order,x1,knots);
      end
   end
   CSP(l+1)=xl;
   CSP(r+1)=xr;
   for ii=1:(m-1),
      Iknots(ii)=CSP(ii)+(CSP(ii+1)-CSP(ii))/2;
   end
   num=floor((n_order+1)/2)+1;
   for jj=1:num,
      Eknots_inf(jj)=CSP(1)-Iknots(num-jj+1);
   end
   for jj=1:num,
      Eknots_sup(jj)=2*CSP(m)-Iknots(m-1-jj+1);
   end
   knots=[Eknots_inf';Iknots';Eknots_sup'];
   yr=0;
   for intervalo=1:7,
      pertenca=bspline(intervalo-1,3,x1,knots);
      yr=yr+Y(intervalo)*pertenca;
   end
   for intervalo=1:7,
      pertenca=bspline(intervalo-1,3,x1,knots);
      Y(intervalo)=Y(intervalo)-epsilon*(yr-yd)*pertenca;
   end
end

%Verificação dos resultados
graph=zeros(101,2);
xx=0:0.01:1;
for sample=1:101,
   x1=xx(sample);
   yr=0;
   for intervalo=1:7,
      pertenca=bspline(intervalo-1,3,x1,knots);
      yr=yr+Y(intervalo)*pertenca;
   end
   graph(sample,2)=x1;
   graph(sample,1)=yr;
end
xxx=xx.*xx;
plot(xx,sin(2*pi*xxx),'r');
hold on;
plot(graph(:,2),graph(:,1),'b');
plot(CSP,Y(2:6),'dg');
plot(xx,zeros(size(xx)),'k--');
hold off;
```