Instituto de Ciências Matemáticas de São Carlos

ISSN - 0103-2569

PROTÓTIPO DE UM SISTEMA HÍBRIDO DE RACIOCÍNIO BASEADO EM CASOS

CLAUDIA REGINA MILARÉ
ANDRÉ C. P. L. F. DE CARVALHO

Nº 73

RELATÓRIOS TÉCNICOS DO ICMSC

São Carlos
Abr./1998

Protótipo de um Sistema Híbrido de Raciocínio Baseado em Casos

Claudia Regina Milaré
André Carlos Ponce Leon Ferreira de Carvalho

Universidade de São Paulo
Instituto de Ciências Matemáticas de São Carlos
Departamento de Ciências de Computação e Estatística
Caixa Postal 668, 13560-970 - São Carlos, SP
e-mail:{claudia, andre}@icmsc.sc.usp.br

Resumo

Muitos pesquisadores atualmente utilizam duas ou mais técnicas (paradigmas) para solucionar problemas. Sistemas que envolvem duas ou mais técnicas na resolução de problemas são denominados de Sistemas Híbridos. Com a utilização de sistemas híbridos pode-se extrair as vantagens de cada uma das técnicas envolvidas e obter melhores soluções para problemas enfrentados.

Redes Neurais (RN) e Raciocínio Baseado em Casos (RBC) são dois paradigmas diferentes de Inteligência Artificial que apresentam características interessantes.
Neste trabalho é descrito um protótipo de um sistema de Raciocínio Baseado em Casos aplicado ao domínio culinário. Este protótipo integra uma Rede Neural que auxilia o sistema de Raciocínio Baseado em Casos em algumas de suas fases.

São Carlos, abril de 1998.

Sumário

1 Introdução

Atualmente, há grande interesse dos pesquisadores em combinar mais de um paradigma para solucionar problemas. Sistemas com esta característica são denominados Sistemas Híbridos [Goonatilake 95]. O que se espera ao integrar várias técnicas é obter um resultado mais eficiente na resolução de problemas, combinando o que cada técnica envolvida tem de mais eficiente.

A área de Redes Neurais (RN) representa o paradigma conexionista de Inteligência Artificial. Redes Neurais são baseadas em sistemas neurais biológicos, utilizadas para desenvolver sistemas computacionais que adquirem conhecimento através da experiência. Dentre suas características mais notáveis podemos citar a rapidez com que o conhecimento armazenado na rede pode ser acessado e utilizado; e ainda sua capacidade de generalização [Fausett 94].

Raciocínio Baseado em Casos (RBC) é um paradigma simbólico de Inteligência Artificial que utiliza experiências passadas (casos) para resolver problemas correntes [Kolodner 93]. Geralmente, um sistema de RBC deve ter mecanismos para:

- Representação de Casos
- Indexação de Casos
- Recuperação de Casos
- Reuso de Casos
- Revisão/Reparo de Casos

Raciocínio Baseado em Casos e Redes Neurais têm sido combinados em algumas pesquisas [Malek 95, Reategui 94, Reategui 95]. Este relatório descreve um protótipo de um sistema de Raciocínio Baseado em Casos implementado na linguagem de programação lógica Prolog.

2 O Modelo ART1

O conhecimento a respeito de determinado domínio do mundo real pode se modificar ao longo do tempo. O mesmo acontece com os padrões de entrada de uma Rede Neural que podem se modificar com o passar do tempo para acomodar mudanças ocorridas. Assim, para um número de aplicações, o desempenho da rede pode ir decaindo com o tempo uma vez que os pesos fixados na fase de treinamento não acompanham estas mudanças. Para adaptar novos padrões de entrada indefinidamente, um algoritmo de aprendizado de Redes Neurais deve ser *plástico* [Heins 95].

Para solucionar este problema, poderia-se retreinar a rede com os novos padrões de entrada, mas com isto a informação antiga seria perdida. Para preservar o conhecimento previamente aprendido, um algoritmo de aprendizado de Redes Neurais deve ser não somente *plástico*, mas também *estável* [Heins 95]. Outra solução seria retreinar a rede frequentemente com os novos e antigos padrões. Isto resolveria o problema, mas está longe de ser uma solução viável.

O conflito entre estabilidade e plasticidade é chamado dilema *estabilidade-plasticidade* [Carpenter 87], e envolve questões como:

- *Como um sistema de aprendizado pode ser projetado para se adaptar indefinidamente em resposta à eventos significativos e ainda continuar indiferente a eventos irrelevantes?*

- *Como pode um sistema de aprendizado preservar seu conhecimento previamente aprendido enquanto continua a aprender conhecimentos novos?*

- *O que previne um novo conhecimento de se sobrepor a conhecimentos prévios?*

Como foi descrito anteriormente, nas aplicações do mundo real é comum ocorrerem mudanças inesperadas nos dados de entrada dos sistemas. Em sistemas de Raciocínio Baseado em Casos não é diferente, ainda mais por serem sistemas que geralmente envolvem aplicações cujo domínio é *aberto*, ou seja, domínios que não foram ainda vastamente estudados.

Para a finalidade que se destina a utilização de Redes Neurais neste trabalho, é necessário um modelo de rede incremental, no qual não seja preciso retreinar a rede quando mudanças nos padrões de entrada ocorrerem e ainda que preserve o conhecimento aprendido previamente [Heins 95].

Uma família de Redes Neurais chamada ART (*Adaptive Resonance Theory*) foi desenvolvida para solucionar o dilema estabilidade-plasticidade [Carpenter 87]. O primeiro modelo desenvolvido da família ART foi a rede ART1 [Grossberg 76]. Este modelo aceita como entrada para a rede somente números binários e envolve duas camadas de neurônios. A camada de entrada, também conhecida como F_1, para o processamento dos dados de entrada; e a camada de saída, também conhecida como F_2, contendo os neurônios de *cluster,* ver Figura 1.

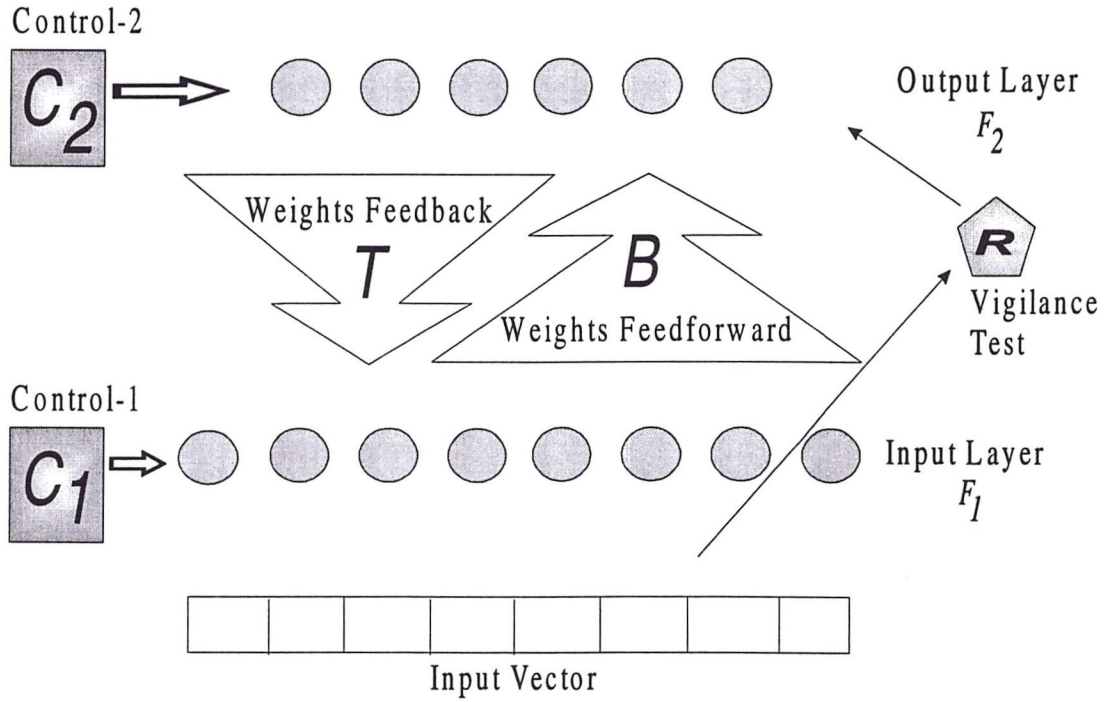

Figura 1: Arquitetura básica da rede ART1.

Estas duas camadas estão conectadas através de dois vetores de pesos. Um vetor de pesos *feedforward*, da camada de entrada para a camada de saída; e outro vetor de pesos *feedback*, da camada de saída para a camada de entrada.

Para cada camada da rede ART há uma unidade externa de controle. Estas unidades controlam o fluxo de dados através das camadas em cada fase de um ciclo de treinamento. Elas são representadas pelas letras C_1 e C_2.

Entre a camada de entrada e a camada de saída há também uma unidade de *reset*, representada pela letra *R*, que é responsável por comparar as entradas da rede com um *threshold* de vigilância (ρ). O *threshold* de vigilância determina se uma nova classe de padrão deverá ser criada para um padrão de entrada.

3 Implementação

Este protótipo foi desenvolvido na linguagem de programação lógica Prolog, por ser uma linguagem de desenvolvimento rápido de protótipos.

Como foi descrito na Seção 2, a rede neural ART1 possui dois vetores de pesos: o vetor de pesos *bottom-up* e o vetor de pesos *top-down*. Nesta implementação os vetores de pesos *bottom-up* e *top-down* são representados por uma lista. O número de elementos destas listas dependem do número de neurônios da camada de saída. Assim, se a camada de saída contiver 20 neurônios, cada lista de pesos, *bottom-up* e *top-down*, terá também 20 elementos. Os elementos de cada uma destas listas são listas cujos elementos são números reais para os pesos *bottom-up* e 0 ou 1 para os pesos *top-down*. Os valores dos pesos *bottom-up* representam a conexão dos neurônios da camada de entrada para camada de saída. A conexão da camada de saída para a camada de entrada é feita através dos pesos *top-down*.

Os padrões de entrada também estão representados por uma lista. Cada elemento desta lista corresponde a um padrão de entrada. Como foi utilizado dados do domínio da culinária para testar este protótipo, cada elemento da lista **s** representa uma receita culinária. Cada padrão de entrada possui o nome da receita, uma lista indicando a presença (representado pelo valor 1) ou ausência (representado pelo valor 0) de determinados ingredientes, o modo de preparo da receita, as calorias da receita e o tempo gastao no preparo da receita. Os elementos da lista de padrões tem o seguinte formato:

```
[nome_da_receita, [lista correspondentes aos ingredientes
da receita], modo_de_preparo, calorias, tempo_de_preparo]
```

Os neurônios de saída estão também representados em uma lista, **y**. Estes neurônios, como foi descrito na Seção 2, representam os *clusters* em que os padrões serão agrupados pela Rede Neural. Inicialmente, o valor de cada neurônio de saída é 0.

A seguir serão descritos os procedimentos desenvolvidos nesta implementação. Os procedimentos de níveis mais alto serão descritos primeiramente.

Procedimento 3.1 `inicializa/0`

```
inicializa:-
     pt(PT),
     pb(PB),
     abre_arquivo(Arq),
     abre_base(Arq2),
     limpa,
     assert(p_top(PT)),
     assert(p_botton(PB)),
     assert(cont_padrao(0)),
     assert(contador(1)),
     principal(Arq, Arq2),
     close(Arq),
     close(Arq2).
```

O procedimento `inicializa/0` instancia PT e PB com os pesos *top-down* e *bottom-up* respectivamente através dos procedimentos `pt/1` e `pb/1`. Em seguida, são abertos dois arquivos com os procedimentos `abre_arquivo/1` e `abre_base/1`. Estes procedimentos terão sucesso se conseguirem abrir os arquivos corretamente, fornecendo os apontadores em `Arq` e `Arq2` respectivamente. A seguir, o procedimento `limpa/0` limpa a memória. O comando `assert` carrega a memória com informações que serão posteriormente utilizadas, como por exemplo; as listas de pesos *top-down* e *bottom-up*, número correspondente ao número do padrão que será apresentado à rede primeiramente e o número 1 correspondente ao primeiro ciclo que a Rede Neural executará. O procedimento `principal/2` é chamado e finalmente termina com o fechamento dos arquivos através do comando `close`.

Procedimento 3.2 `pt/1`

```
pt([[lista_de_pesos], ..., [lista_de_pesos]]).
```

O procedimento `pt/1` possui somente um argumento que é a lista de pesos *top-down*.

Procedimento 3.3 `pb/1`

```
pb([[lista_de_pesos], ..., [lista_de_pesos]]).
```

O procedimento `pb/1` também possui somente um argumento, a lista de pesos *bottom-up*.

Procedimento 3.4 `abre_arquivo/1`

```
abre_arquivo(Arq) :-
     nl, nl,
     write('Arquivo onde serao gravados os resultados: '),
     read(Arq), nl,
     open(Arq, append),
     tell(Arq),
     outpos(0),
     write('----- Resultados do treinamento --------'),
     nl, nl.
```

Os resultados referentes ao treinamento da rede serão armazenados em um arquivo, cujo nome deverá ser fornecido pelo usuário. O procedimento `abre_arquivo/1` é responsável pela abertura deste arquivo.

Procedimento 3.5 `abre_base/1`

```
abre_base(Arq2) :-
     tell(user),nl,
     write('Base de Casos: '),
     read(Arq2), nl, nl,
     open(Arq2, append),
     tell(Arq2),
     outpos(0).
```

O procedimento `abre_base/1` abre um arquivo onde será gravada a base de dados do sistema, ou seja, os casos.

Procedimento 3.6 `limpa/0`

```
limpa :-
     abolish(contador/1),
     abolish(cont_padrao/1),
     abolish(p_top/1),
     abolish(p_botton/1),
     abolish(indice/1),
     abolish(receita/1),
     abolish(preparo/1),
     abolish(calorias/1),
     abolish(tempo/1),
     abolish(casos/1),
     abolish(x1/1).
```

Este procedimento `limpa/0`, como foi descrito anteriormente, limpa a memória, evitando possíveis erros que possam ocorrer devido algum lixo existente na memória.

Procedimento 3.7 `principal/2`

```
principal(Arq, Arq2):-
     one num_ciclo(Num_Ciclo),
     tell(user),
     write('Aguarde... a rede esta sendo treinada!'),
     nl, nl,
     write('Digite o grau de semelhanca entre os casos: '),
     read(Ro), nl,
     tell(Arq),
     write('Ro = '), write(Ro), nl,
     l(L),
     write('L = '), write(L), nl, nl,
     write('Ciclos = '), write(Num_Ciclo), nl,
     num_padrao(Num_Padrao),
     write('Padroes = '),
     write(Num_Padrao), nl, nl, nl,
     repeat,
     tell(Arq),
     one retract(contador(Cont1)),
     one abolish(indice/1),
```

```
        one abolish(x1/1),
        one pega_padrao(S),
        one passo4(S, NormaS),
        one passo5(S, X),
        one retract(p_botton(PB)),
        one passo6(PB, X, Y),
        one assert(p_botton(PB)),
        one passo7(Ro, Y, S, Reset, Flag, Flag_),
        one atualiza_antes(Arq, Arq2, S, Cont1, Flag_),
        one (Cont2 is Cont1 + 1),
        one assert(contador(Cont2)),
        one (Cont1 = Num_Ciclo),
        one tell(user),
        one write('O treinamento terminou!'), nl,
        repeat,
        one interface(S1, Ro_),
        one classifica(Arq2, S1, Ro_),
        one tell(user), nl, nl,
        one write('Deseja continuar?'),
        one read(Parar), nl, nl,
        one (Parar = nao), !.
```

O procedimento `principal/2` possui dois argumentos: os apontadores para os arquivos que foram abertos anteriormente. Este procedimento é responsável por chamar todos os demais procedimentos.

Procedimento 3.8 `num_ciclo/1`

O procedimento num_ciclo/1 possui um argumento: o número de ciclos de treinamento da Rede Neural.

Procedimento 3.9 `l/1`

O procedimento l/1 possui um argumento: o valor do parâmetro **1** da rede ART1.

Procedimento 3.10 `num_padrao/1`

O procedimento `num_padrao/1` possui um argumento: o número de padrões com que a Rede Neural será treinada.

Procedimento 3.11 `pega_padrao/1`

```
pega_padrao(S) :-
        retract(cont_padrao(N1)),
        randon(N1, N2),
        s(Padroes),
        n_esimo(N2, AuxS, Padroes),
        n_esimo(1, Receita, AuxS),
        n_esimo(2, S, AuxS),
        n_esimo(3, Preparo, AuxS),
```

```
        n_esimo(4, Calorias, AuxS),
        n_esimo(5, Tempo, AuxS),
        assert(receita(Receita),
        assert(calorias(Calorias),
        assert(tempo(Tempo),
        assert(preparo(Preparo)).
```

O procedimento `pega_padrão/1` é sempre bem sucedido se o único argumento que ele possui for unificado com um padrão a ser apresentado à Rede Neural à cada ciclo de treinamento. Primeiramente, este procedimento instancia `N1` com a posição (na lista de padrões) do padrão apresentado à Rede Neural anteriormente. Em seguida chama o procedimento `randon/2` que tem por finalidade indicar qual a posição do próximo padrão (referente à lista de padrões) a ser apresentado à Rede Neural. Logo após, a variável `Padrões` é instanciada com a lista de padrões através do procedimento `s/1`. O padrão que será apresentado à rede é extraído da lista de padrões que foi instanciada em `Padrões` através da primeira chamada ao procedimento `n_esimo/3`. A seguir, é extraído do padrão, ou seja, da receita culinária que está representada em uma lista, seu primeiro elemento (que corresponde ao nome da receita culinária) através da segunda chamada ao procedimento `n_esimo/3`; seu segundo elemento (que corresponde a uma lista representado os ingredientes da receita culinária) através da terceira chamada ao procedimento `n_esimo/3`; e o terceiro, quarto e quinto elemento (correspondendo a modo de preparo, calorias e tempo de preparo respectivamente) através da quarta, quinta e sexta chamada ao procedimento `n_esimo/3` respectivamente. Finalmente, o nome da receita culinária, as calorias, o tempo de preparo e o modo de preparo são carregados para a memória através do comando `assert`.

Procedimento 3.12 `randon/2`

```
randon(N1, N2) :-
        num_padrao(Num_Padrao),
        N1 == Num_Padrao,
        N2 is 1,
        assert(cont_padrao(N2)).

randon(N1, N2) :-
        N2 is N1 + 1,
        assert(cont_padrao(N2)).
```

O procedimento `randon/2` encontra a posição do próximo padrão (referente à lista de padrões) a ser apresentado à Rede Neural. O primeiro argumento, argumento de entrada, refere-se à posição do padrão anterior apresentado à rede. O segundo argumento, argumento de saída, refere-se à posição do próximo padrão a ser apresentado à rede. Se o argumento de entrada refere-se à posição do último padrão da lista de padrões, o procedimento `randon/2` pega a posição do primeiro elemento da lista de padrões, ou seja, posição 1.

Procedimento 3.13 `s/1`

```
s([[padrão], [padrão], ..., [padrão]]).
```

O procedimento `s/1` possui somente um argumento, a lista de padrões.

Procedimento 3.14 `n_esimo/3`

```
n_esimo(1, Elem, [Elem|_]).

n_esimo(N, Elem, [_|Cauda]) :-
          n_esimo(M, Elem, Cauda),
          N is M+1.
```

O procedimento `n_esimo/3` extrai o n-ésimo elemento de uma lista. O primeiro argumento do procedimento `n_esimo` pode ser a posição do elemento na lista ou então uma variável. O segundo argumento pode ser o elemento que se quer extrair da lista ou então uma variável que será instanciada com o elemento extraído da lista. Finalmente, o terceiro argumento é uma lista da qual se extrairá o n-ésimo elemento.

Os procedimentos a seguir são procedimentos utilizados no treinamento da Rede Neural.

Procedimento 3.15 `passo4/2`

```
passo4(Lista, Norma):-
    norma(Lista, Norma).
```

O procedimento `passo4/2` determina a norma de um vetor através da chamada ao procedimento `norma/2`.

Procedimento 3.16 `norma/2`

```
norma([],0).

norma([Elem|Cauda],S):-
    norma(Cauda,S1),
    S is S1 + Elem.
```

O procedimento `norma/2` calcula a norma de um vetor. O primeiro argumento deste procedimento é uma lista e o segundo argumento é o resultado da norma.

Procedimento 3.17 `passo5/2`

```
passo5(S, X) :-
    x(S,X).
```

O procedimento `passo5/2` envia um sinal de entrada para a camada de entrada através do procedimento `x/2`.

Procedimento 3.18 `x/2`

```
x(X,X).
```

O procedimento `x/2` instancia X (segundo argumento do procedimento) com o primeiro argumento.

Procedimento 3.19 `passo6/2`

```
passo6(PB, X, Y):-
    y6(PB, X, Y).
```

O procedimento `passo6/3` determina a ativação de cada neurônio de saída para uma determinada entrada através da chamada ao procedimento `y6/3`. O primeiro argumento corresponde aos pesos *bottom-up*. O segundo argumento é uma lista correspondente ao padrão de entrada atual e o terceiro argumento é uma lista com a ativação dos neurônios da camada de saída.

Procedimento 3.20 `y6/3`

```
y6([PB], X, [CaudaY]):-
    auxy(PB, X, Aux),
    soma(Aux, CaudaY).

y6([PB|CaudaPB], X, [Y|CaudaY]):-
    auxy(PB, X, Aux),
    soma(Aux, Y),
    y6(CaudaPB, X, CaudaY).
```

O procedimento `y6/3` calcula a ativação de cada neurônio de saída para as entradas da Rede Neural com o auxílio dos procedimentos `auxy/3` e `soma/2`.

Procedimento 3.21 `auxy/3`

```
auxy([PB], [X], [CaudaAux]):-
    CaudaAux is PB * X.

auxy([PB|CaudaPB], [X|CaudaX], [Aux|CaudaAux]):-
    Aux is PB * X,
    auxy(CaudaPB, CaudaX, CaudaAux).
```

O procedimento `auxy/3` auxilia no cálculo da ativação dos nerônios de saída.

Procedimento 3.22 `soma/2`

```
soma([],0).

soma([Elem|Cauda],X):- soma(Cauda,X1),
    X is X1 + Elem.
```

O procedimento `soma/2` calcula a soma dos elementos de uma lista. O primeiro argumento deste procedimento é uma lista e o segundo argumento é o resultado da soma dos elementos.

Procedimento 3.23 `passo7/6`

```
passo7(Ro, Y, S, Reset, Flag, Flag_) :-
    passo8(Y, Maiory),
    one(n_esimo(Indice, Maiory, Y)),
    one(retract(p_top(PT))),
    one(n_esimo(Indice, T, PT)),
    one(passo9(T, S, X1)),
    one(passo11(Ro,Maiory,Y,S,X1,Ynew,Reset,Flag,Flag_)),
    one(assert(p_top(PT))),
    passo71(Ro, Ynew, S, Reset, Flag, Flag_).
```

O procedimento `passo7/6` é responsável por executar os procedimentos `passo8/2`, `passo9/3` e `passo11/9` até encontrar um *cluster* (neurônio de saída) que possa inserir o padrão de entrada corrente ou se não encontrar alocar um novo neurônio de saída para este padrão.

Procedimento 3.24 `passo8/2`

```
passo8(X, MaiorY) :-
    maiory(X, MaiorY).
```

O procedimento `passo8/2` determina o maior elemento de uma lista através da chamada ao procedimento `maiory/2`.

Procedimento 3.25 `maiory/2`

```
maiory([X],X).

maiory([X,Y|Cauda], MaiorY) :-
    X >= Y,
    !,
    maiory([X|Cauda], MaiorY).

maiory([X,Y|Cauda], MaiorY) :-
    maiory([Y|Cauda], MaiorY).
```

O procedimento `maiory/2` encontra o maior elemento de uma lista. O primeiro argumento deste procedimento é uma lista de números e o segundo o maior elemento desta lista.

Procedimento 3.26 `passo9/3`

```
passo9(PT, S, X):-
    y9(PT, S, X).
```

O procedimento `passo9/3` recomputa a ativação da camada de entrada através da chamada ao procedimento `y9/3`.

Procedimento 3.27 `y9/3`

```
y9([PT], [S], [CaudaX]) :-
    CaudaX is PT * S.

y9([PT|CaudaPT], [S|CaudaS], [X|CaudaX]) :-
    X is PT * S,
    y9(CaudaPT, CaudaS, CaudaX).
```

O procedimento `y9/3` calcula a nova ativação da camada de entrada. O primeiro argumento deste procedimento é uma lista (no qual cada elemento desta lista é uma lista) correspondente aos pesos *top-down*. O segundo argumento é uma lista correspondente ao padrão de entrada atual e o terceiro argumento é o resultado da nova ativação da camada de entrada.

Procedimento 3.28 `passo11/9`

```
passo11(Ro,Maiory,Y,S,X1,Ynew,Reset,Flag,Flag_):-
    norma(S, NormaS),
    norma(X1, NormaX),
    teste(Ro, Maiory, Y, NormaS, X1, NormaX, Ynew,
          Reset, Flag, Flag_).
```

O procedimento `passo11/9` verifica se o neurônio de saída escolhido é semelhante o suficiente (compara com o parâmetro de vigilância, ρ) para alocar o padrão de entrada corrente.

Procedimento 3.29 `teste/10`

```
teste(Ro,Maiory,Y,NormaS,X1,NormaX,Ynew,0,Flag,Flag_):-
    Reset is (NormaX/NormaS),
    Reset >= Ro,
    n_esimo(Indice, Maiory, Y),
    abolish(indice/1),
    assert(indice(Indice)),
    assert(x1(X1)),
    Flag is 0.

teste(Ro,Maiory,Y,NormaS,X1,NormaX,Ynew,1,Flag,Flag_):-
    Reset is (NormaX/NormaS),
    Reset < Ro,
    n_esimo(Indice, Maiory, Y),
    retirar(Maiory, Y, CaudaY),
    insere(-1, Indice, CaudaY, Ynew),
    verifica(Ynew, Flag).
```

O procedimento `teste/10` é que fará o teste que compara a ativação do neurônio escolhido com o valor do parâmetro de vigilância.

Procedimento 3.30 `retirar/3`

```
retirar(Elem, [Elem|Cauda], Cauda).

retirar(Elem, [Elem1|Cauda], [Elem1|Cauda1]) :-
    retirar(Elem, Cauda, Cauda1).
```

O procedimento `retirar/3` retira uma ocorrência de um elemento de uma lista. O primeiro argumento deste procedimento corresponde a um elemento, o segundo argumento corresponde à lista que será retirado o elemento e o terceiro argumento corresponde à lista sem o elemento.

Procedimento 3.31 `insere/4`

```
insere(Elem, 1, Lista, [Elem|Lista]):-!.

insere(Elem, Posicao, [X|Cauda], [X|Resp]):-
    Pos2 is Posicao - 1,
    insere(Elem, Pos2, Cauda, Resp).
```

O procedimento `insere/4` insere um elemento em uma lista dada sua posição. O primeiro argumento deste procedimento corresponde ao elemento a ser inserido, o segundo à posição em que o elemento será inserido, o terceiro à lista sem o elemento e finalmente, o quarto argumento corresponde à lista resultante da inserção do elemento.

Procedimento 3.32 `verifica/2`

```
verifica(Y, Flag):-
    retirar_todas(-1, Y, L),
    aciona_flag(Flag, L), !.
```

O procedimento `verifica/2` verifica se há ainda algum neurônios de saída que não tenha sido inibido (com valor igual a –1). `Verifica/2` faz isto através da chamada aos procedimentos `retirar_todas/3` e `aciona_flag/2`. O procedimento `retirar_todas/3` retira todos os neurônios de saída com valor igual a –1. O procedimento `aciona_flag/2` seta o valor de `Flag` com 1 se todos os neurônios de saída estão inibidos e com 0 caso contrário.

Procedimento 3.33 `retirar_todas/3`

```
retirar_todas(_, [], []).

retirar_todas(Elem, [Elem|Cauda], L) :-
    retirar_todas(Elem, Cauda, L).

retirar_todas(Elem,[Elem1|Cauda],[Elem1|Cauda1]) :-
    Elem \== Elem1,
    retirar_todas(Elem, Cauda, Cauda1).
```

O procedimento `retirar_todas/3` retira todas as ocorrências de um elemento de uma lista. O primeiro argumento deste procedimento corresponde ao elemento que se deseja retirar da lista. O segundo argumento corresponde à lista de onde as ocorrências do elemento serão retiradas e o terceiro argumento corresponde à lista sem qualquer ocorrência do elemento.

Procedimento 3.34 `aciona_flag/2`

```
aciona_flag(1,[]) :- !.

aciona_flag(0,X)  :- !.
```

O procedimento `aciona_flag/2` seta o valor de `Flag` com 1 se todos os neurônios de saída estão inibidos e com 0 caso contrário.

Procedimento 3.35 `passo71/6`

```
passo71(Ro, Y, S, 0, 0, Flag_) :-
    Flag_ is 0,!.

passo71(Ro, Y, S, 1, 1, Flag_) :-
    Flag_ is 1,!.

passo71(Ro, Y, S, 1, Flag, Flag_) :-
    passo8(Y, Maiory),
    one(n_esimo(Indice, Maiory, Y)),
    one(retract(p_top(PT))),
    one(n_esimo(Indice, T, PT)),
    one(passo9(T, S, X1)),
    one(passo11(Ro,Maiory,Y,S,X1,Ynew,Reset,Flag1,Flag_)),
    one(assert(p_top(PT))),
    passo71(Ro, Ynew, S, Reset, Flag1, Flag_),!.
```

O Procedimento `passo71/6` é chamado pelo procedimento `passo7/6` para auxiliar na seleção do neurônio de saída que poderá alocar o padrão de entrada corrente.

Procedimento 3.36 `atualiza_antes/5`

```
atualiza_antes(Arq, Arq2, S, Cont1, 0) :-
    one(retract(p_top(PT))),
    one(retract(indice(Indice))),
    one(gravar(Arq, Arq2, S, Cont1, Indice)),
    one(retract(p_botton(PB))),
    one(retract(x1(X1))),
    one(passo12(PT, PB, Indice, X1, PTnew, PBnew)),
    one(assert(p_top(PTnew))),
    one(assert(p_botton(PBnew))),!.

atualiza_antes(Arq, Arq2, S, Cont1, 1) :-
    one tell(Arq),
    one(write('Nao consegui classificar! ')),
    one(gravar(Arq, Arq2, S, Cont1, Indice)).
```

O procedimento `atualiza_antes/5` chama os procedimentos para atualizar os pesos da Rede Neural na fase de treinamento. Este procedimento é também responsável por chamar os procedimentos para: gravar os padrões associados aos respectivos *clusters* (neurônios de saída) em um arquivo (Base de Casos) e informações referentes ao treinamento em outro arquivo.

Procedimento 3.37 `gravar/5`

```
gravar(Arq, Arq2, S, Cont1, Indice) :-
    one pode_gravar(Cont1), !,
    one tell(Arq),
    one retract(receita(Receita)),
    one(write('Receita: ')),
    one(write(Receita)),nl,
    one(write('Padrao: ')),
    one(write(S)), nl,
    one(write('Cluster: ')),
    one(write(Indice)),nl,nl,
    one retract(preparo(Preparo)),
    one retract(calorias(Calorias)),
    one retract(tempo(Tempo)),
    one armazena(Arq2, Indice, Receita, S, Preparo,
                 Calorias, Tempo),
    one tell(Arq).

gravar(Arq, Arq2, S, Cont1, Indice) :- !.
```

O procedimento `gravar/5` chama o procedimento `pode_gravar/1` que verifica se é a última vez que cada padrão será apresentado à rede. Se for, este procedimento grava algumas informações referentes ao treinamento da rede em um arquivo e chama o procedimento `armazena/7` que grava informações em outro arquivo (Base de Casos).

Procedimento 3.38 `pode_gravar/1`

```
pode_gravar(Cont1) :-
    one num_padrao(Padroes),
    one num_ciclo(Ciclos),
    one Aux is (Ciclos - Padroes),
    one Cont1 > Aux.
```

O procedimento `pode_gravar/1` verifica se é a última vez que cada padrão será apresentado à rede.

Procedimento 3.39 `armazena/7`

```
armazena(Arq2, Indice, Receita, S, Preparo,
         Calorias, Tempo) :-
    one tell(Arq2),
    one write('caso('), write(Indice), write(','),
    one write(Receita), write(','),
    one write(S), write(','), write(''''),
    one write(Preparo), write(''''), write(','),
    one write(Calorias), write(','),
    one write(Tempo), write(','),
    one write(').'), nl.
```

O procedimento `armazena/7` grava em um arquivo (Base de Casos) as informações referentes aos padrões (casos). Estas informações são: ingredientes, modo de preparo, calorias, tempo de preparo e um índice referente ao *cluster* que este padrão foi associado.

Procedimento 3.40 `passo12/6`

```
passo12(PT, PB, Indice, X, PTnew, PBnew) :-
    n_esimo(Indice, Told, PT),
    retirar(Told, PT, ListaPT),
    insere(X, Indice, ListaPT, PTnew),
    norma(X, NormaX),
    bu(X, NormaX, Bnew),
    n_esimo(Indice, Bold, PB),
    retirar(Bold, PB, ListaPB),
    insere(Bnew, Indice, ListaPB, PBnew).
```

O procedimento `passo12/6` faz a atualização dos pesos *top-down* e chama o procedimento `bu/3` que atualiza os pesos *botton-up*.

Procedimento 3.41 `bu/3`

```
bu([X], NormaX, [CaudaBnew]) :-
    l(L),
    CaudaBnew is ((L*X)/(L - 1 + NormaX)).

bu([X|CaudaX], NormaX, [Bnew|CaudaBnew]) :-
    l(L),
    Bnew is ((L*X)/(L - 1 + NormaX)),
    bu(CaudaX, NormaX, CaudaBnew).
```

O procedimento `bu/3` atualiza os pesos *bottom-up*.

Procedimento 3.42 `interface/2`

```
interface(S, Ro):-
    tell(user),nl,nl,
    write('Entre com um caso novo:'),
    read(S),nl,
    one(write('Digite o grau de semelhanca: ')),
    one(read(Ro)).
```

Depois que o treinamento da Rede Neural terminou, o procedimento `interface/2` pede ao usuário um caso novo que instancia com o seu primeiro argumento e o grau de semelhança que instancia com o segundo argumento.

Procedimento 3.43 `classifica/3`

```
classifica(Arq2, S, Ro):-
    one(write('Padrao: ')),
    one(write(S)), nl,
    one(passo4(S, NormaS)),
    one(passo5(S, X)),
    one(retract(p_botton(PB))),
    one(passo6(PB, X, Y)),
    one(assert(p_botton(PB))),
    one(passo7(Ro, Y, S, Reset, Flag, Flag_)),
    one(conseguiu_classificar(Arq2, S, Ro, Flag_)).
```

O procedimento `classifica/3` tenta associar o caso novo a algum dos *clusters* formados anteriormente pela rede. `Classifica/3` chama alguns procedimentos da Rede Neural e em seguida o procedimento consegui_classificar/4 que verifica se a rede consegui associar o caso novo.

Procedimento 3.44 `consegui_classificar/4`

```
conseguiu_classificar(Arq2, S, Ro, 0) :-
    one(recuperacao(Arq2, S)),
    one(aprende(Aprende)),
    one(atualiza_depois(Arq2, Ro, S, Aprende)), !.

conseguiu_classificar(Arq2, S, Ro, 0) :-
    one(retract(indice(Indice))),
    one findall([I, Receita, Padrao, Preparo,
                 Calorias, Tempo],
                 casos(caso(I, Receita, Padrao,
                 Preparo, Calorias, Tempo)),
                 Casos_Recup),
    one separar_indice(Casos_Recup, Ind),
    one assert(indice(Indice)),
    one maiory(Ind, Maior),
    one num_cluster(Num_Cluster),
    one intervalo(Maior, Indice, Num_Cluster, Result),
    one Result == sim ->
    one decide(R),
    one cria_classe(Arq2, S, Ro, Indice, R).

conseguiu_classificar(Arq2, S, Ro, 1) :-
    one abaixa_ro(Arq2, S, Ro).
```

O procedimento `consegui_classificar/4` verifica se a rede consegui associar o novo caso. Se a rede não conseguiu classificar o novo caso, pode-se criar um novo *cluster* ou diminuir o valor do parâmetro de vigilância.

Procedimento 3.45 `recuperação/2`

```
recuperacao(Arq2, S):-
    one(retract(indice(Indice))),
    one abolish(casos/1),
    one close(Arq2),
    one open(Arq2, read),
    one le_base(Arq2),
    one close(Arq2),
    one open(Arq2, append),
    one see(user),
    one findall([Indice, Receita, Padrao, Preparo,
          Calorias, Tempo],
          casos(caso(Indice, Receita, Padrao,
          Preparo, Calorias, Tempo)),
          Casos_Recup),
    one assert(indice(Indice)),
    one separar_nome(Casos_Recup, Nome),
    one separar_padrao(Casos_Recup, Ing),
    one separar_preparo(Casos_Recup, Prep),
    one separar_calorias(Casos_Recup, Cal),
    one separar_tempo(Casos_Recup, Temp),
    one tell(user), nl, nl,
    one write('O sistema recuperou os casos: '),
    one write(Nome),nl,
    escolha_ingred(Ing, S, Res_Ing),
```

```
one mostra_user(Nome, Res_Ing, Temp, Cal,
                Selecionada),
one n_esimo(Pos, Selecionada, Nome),
one tell(user),nl, nl.
```

Se a rede conseguiu associar o novo caso, então o procedimento `recuperação/2` recupera os casos do *cluster* associado e mostra ao usuário.

Procedimento 3.46 `le_base/1`

```
le_base(Arq2) :-
    see(Arq2),
    repeat,
    read(Casos),
    assertz(casos(Casos)),
    at_end_of_file.
```

O procedimento `le_base/1` lê a Base de Casos.

Procedimento 3.47 `separar_nome/3`

```
separar_nome([Cauda], [Cauda1]):-
    n_esimo(2,Cauda1,Cauda).

separar_nome([Casos_Recup|Cauda], [Nome|Cauda1]):-
    n_esimo(2,Nome,Casos_Recup),
    separar_nome(Cauda, Cauda1).
```

O procedimento `separar_nome/3` extrai os nomes das receitas dos casos do *cluster* selecionado.

Procedimento 3.48 `separar_padrao/3`

```
separar_padrao([Cauda], [Cauda1]):-
    n_esimo(3,Cauda1,Cauda).

separar_padrao([Casos_Recup|Cauda], [Padrao|Cauda1]):-
    n_esimo(3,Padrao,Casos_Recup),
    separar_padrao(Cauda, Cauda1).
```

O procedimento `separar_padrao/3` extrai as listas com os ingredientes das receitas dos casos do *cluster* selecionado.

Procedimento 3.49 `separar_preparo/3`

```
separar_preparo([Cauda], [Cauda1]):-
    n_esimo(4,Cauda1,Cauda).

separar_preparo([Casos_Recup|Cauda],[Preparo|Cauda1]):-
    n_esimo(4,Preparo,Casos_Recup),
    separar_preparo(Cauda, Cauda1).
```

O procedimento `separar_preparo/3` extrai o modo de preparo das receitas dos casos do *cluster* selecionado.

Procedimento 3.50 `separar_calorias/3`

```
separar_calorias([Cauda], [Cauda1]):-
    n_esimo(5,Cauda1,Cauda).

separar_padrao([Casos_Recup|Cauda], [Calorias|Cauda1]):-
    n_esimo(5,Calorias,Casos_Recup),
    separar_calorias(Cauda, Cauda1).
```

O procedimento `separar_calorias/3` extrai as calorias das receitas dos casos do *cluster* selecionado.

Procedimento 3.51 `separar_tempo/3`

```
separar_tempo([Cauda], [Cauda1]):-
    n_esimo(3,Cauda1,Cauda).

separar_tempo([Casos_Recup|Cauda], [Tempo|Cauda1]):-
    n_esimo(6,Tempo,Casos_Recup),
    separar_tempo(Cauda, Cauda1).
```

O procedimento `separar_tempo/3` extrai os tempos de preparo das receitas dos casos do *cluster* selecionado.

Procedimento 3.52 `escolha_ingred/3`

```
escolha_ingred([Cauda], S, [Iguais]):-
    ingredientes(Cauda, S, Iguais).

escolha_ingred([Padrao|Cauda], S, [Res_Ing|Iguais]):-
    ingredientes(Padrao, S, Res_Ing),
    escolha_ingred(Cauda, S, Iguais).
```

O procedimento `escolha_ing/3` compara os ingredientes do caso novo com os ingredientes dos casos do *cluster* selecionado através da chamada ao procedimento `ingredientes/3`. O argumento de saída deste parâmetro é uma lista com os números dos ingredientes iguais entre o caso novo e os casos do *cluster* selecionado.

Procedimento 3.53 `ingredientes/3`

```
ingredientes([], [], 0).

ingredientes([L1|CaudaL1], [L2|CaudaL2], L3) :-
    L1 == L2,
    ingredientes(CaudaL1, CaudaL2, SomaAux), !,
    L3 is SomaAux + 1.

ingredientes([L1|CaudaL1], [L2|CaudaL2], L3) :-
    ingredientes(CaudaL1, CaudaL2, L3).
```

O procedimento `ingredientes/3` compara os elementos de duas listas e retorna o número de elementos iguais.

Procedimento 3.54 `mostra_user/3`

```
mostra_user(Nome, Ing, Selecionada):-
    nl, nl,
    one write('A tabela abaixo mostra as receitas
              recuperadas.'),
    nl, nl, nl,
    one write('Receitas Ingredientes Tempo Calorias'),nl,
    one imprimir(Nome, Ing, Tempo, Calorias, 2),
    nl, nl,
    one write('Escolha uma delas, digitando seu nome: '),
    one read(Selecionada).
```

O procedimento `mostra_user/3` mostra ao usuário as informações (nome da receita, ingredientes, tempo de preparo e calorias) das receitas do cluster selecionado ao usuário através da chamada ao procedimento `imprimir/5` e ainda pede ao usuário para selecionar uma das receitas.

Procedimento 3.55 `imprimir/5`

```
imprimir([Nome|C1],[Ing|C2],[Tempo|C3],[Calorias|C4],I):-
    J is I + 3,
    imprimir(Nome, Ing, Tempo, Calorias, J),
    imprx(C1, C2, C3, C4, J).

imprimir(Nome, Ing, Tempo, Calorias, I) :-
    tab(I),
    len(Nome, T1),
    Aux1 is 20 - T1,
    write(Nome),tab(Aux1),
    len(Ing, T2),
    Aux2 is 15 - T2,
    write(Ing), tab(Aux2),
    len(Tempo, T3),
    Aux3 is 8 - T3,
    write(Tempo), tab(Aux3),
    write(Calorias), nl.
```

O procedimento `imprimir/5` imprime na tela as informações, em forma de tabela, referentes aos casos do *cluster* selecionado.

Procedimento 3.56 `imprx/5`

```
imprx([], [], [], [], _).

imprx([Nome|C1],[Ing|C2],[Tempo|C3],[Calorias|C4],I):-
    imprimir(Nome, Ing, Tempo, Calorias, I),
    imprx(C1, C2, C3, C4, I).
```

O procedimento `imprx/5` auxilia o procedimento `imprimir/5` na imprime impressão das informações referentes aos casos do *cluster* selecionado.

Procedimento 3.57 `aprende/1`

```
aprende(Aprende) :-
    tell(user),
    nl, nl,
    write('Deseja aprender este novo caso? '),
    read(Aprende).
```

Depois que o usuário selecionou uma receita, o procedimento `aprende/1` pergunta ao usuário se ele deseja aprender este caso.

Procedimento 3.58 `atualiza_depois/4`

```
atualiza_depois(Arq2, Ro, S, sim) :-
    one(retract(p_top(PT))),
    one(retract(indice(Indice))),
    one(retract(p_botton(PB))),
    one(retract(x1(X1))),
    one(passo12(PT, PB, Indice, X1, PTnew, PBnew)),
    one(assert(p_top(PTnew))),
    one(assert(p_botton(PBnew))),
    one(tell(user)),
    one(write('Qual o nome da receita? ')),
    one(read(Receita)),nl, nl,
    one(write('E o modo de preparo? ')),
    one(read(Prep)),nl, nl,
    one(write('E quanto aos ingredientes?')),
    one(read(S1)), nl,nl,
    one(write('Calorias:')),
    one(read(Cal)), nl,nl,
    one(write('Tempo de Preparo:')),
    one(read(Temp)), nl,nl,
    one(armazena(Arq2,Indice,Receita,S1,Prep,Cal,Temp)).

atualiza_depois(Arq2, Ro, S, nao) :- !.
```

O predicado `atualiza_depois/4` atualiza os pesos da Rede Neural na fase de uso.

Procedimento 3.59 `separar_indice/3`

```
separar_indice([Cauda], [Cauda1]):-
    n_esimo(1,Cauda1,Cauda).

separar_indice([Casos_Recup|Cauda], [Ind|Cauda1]):-
    n_esimo(1,Ind,Casos_Recup),
    separar_indice(Cauda, Cauda1).
```

O procedimento `separar_indice/3` extrai os índices (número do cluster associado ao caso) das receitas dos casos do *cluster* selecionado.

Procedimento 3.60 `num_cluster/1`

O procedimento `num_cluster/1` possui um argumento: o número de clusters (neurônio s de saída) que a rede possui. Este valor é utilizado para verificar se há a possibilidade de criar um *cluster* novo.

Procedimento 3.61 `intervalo/4`

```
intervalo(Maior, Indice, Num_Cluster, Result) :-
    one Indice > Maior,
    one Indice =< Num_Cluster,
    one Result = 'sim', !.

intervalo(Maior, Indice, Num_Cluster, Result) :-
    one Result = 'nao'.
```

O procedimento `intervalo/4` verifica se há possibilidade de criar um *cluster* novo, ou seja, verifica se há algum neurônio de saída que ainda não foi utilizado para alocar um grupo de casos.

Procedimento 3.62 `decide/1`

```
decide(R) :-
    nl, nl,
    one write('Nao consigo recuperar nenhum caso!'),
    nl, nl,
    one write('Voce deseja: '), nl,nl,
    one write('1. Criar novo cluster?'), nl,
    one write('2. Mudar a semelhanca entre os
    casos?'), nl, nl,
    one write('Voce decide!  '),
    one read(R), nl, nl.
```

Se há possibilidade de criação de um novo *cluster*, o sistema pergunta ao usuário, através do procedimento `decide/1`, se ele deseja criar um novo *cluster* ou diminuir o parâmetro de vigilância para tentar classificar o caso em algum *cluster* já existente.

Procedimento 3.63 `cria_classe/5`

```
cria_classe(Arq2, S, Ro, Indice, 2) :-
    one abaixa_ro(Arq2, S, Ro, _), !.

cria_classe(Arq2, S, Ro, Indice, 1) :-
    one atualiza_depois(Arq2, Ro, S, sim).
```

Se o usuário desejar diminuir o parâmetro de vigilância, o procedimento `cria_classe/5` faz uma chamada ao procedimento `abaixa_ro/4` para diminuir o valor do parâmetro de vigilância e tentar associar o caso novo. Caso o usuário decida por criar um novo cluster, o procedimento `atualiza_depois/4` é chamado.

Procedimento 3.64 `abaixa_ro/4`

```
abaixa_ro(Arq2, S, Ro, _) :-
    one(tell(user)), nl, nl,
    one(write('Deseja mesmo mudar o grau de semelhanca? ')),
    one(read(Resposta)), nl,
    one(resposta(Resposta, Ro, Ro_)),
    one Resposta \== 'nao' ->
    one classifica(Arq2, S, Ro_)
    ; write('Infelizmente nao foi possivel classificar
              este caso!').
```

O procedimento `abaixa_ro/4` diminui o valor do parâmetro de vigilância e tenta classificar o caso novamente.

Procedimento 3.65 `resposta/3`

```
resposta(nao, Ro, Ro_):- Ro_ is Ro, !.

resposta(sim, Ro, Ro_):-
    nl, nl,
    write('Digite o grau de semelhanca desejado: '),
    read(Ro_),
    nl, nl.
```

O procedimento `resposta/3` ao usuário que digite o valor do parâmetro de vigilância.

Procedimento 3.66 `abaixa_ro/3`

```
abaixa_ro(Arq2, S, Ro) :-
    one(tell(user)), nl, nl,
    one(write('Nao consegui classificar o caso! ')),
    nl, nl,
    one(write('Diminui semelhanca entre este caso e os
         casos armazenados? ')),
    one(read(Resposta)), nl,
    one(resposta(Resposta, Ro, Ro_)),
```

```
one Resposta \== 'nao' ->
one classifica(Arq2, S, Ro_)
; write('Infelizmente nao foi possivel classificar
          este caso!').
```

Quando não há mais neurônios de saída disponíveis, o procedimento abaixa_ro/3 pergunta ao usuário se ele deseja diminuir o valor do parâmetro de vigilância. Se o usuário decidir por diminuir, o procedimento tenta associar o caso.

4 Exemplos de Execução

Nesta seção são apresentados alguns exemplos de execução. No primeiro exemplo é solicitado ao usuário para que entre com um novo caso, ou seja, forneça os ingredientes principais do novo caso. O usuário então entra com um vetor representando o novo caso. Neste caso os ingredientes principais do novo caso são: óleo, maçã, açúcar e farinha, representados pelo valor 1 do vetor. Em seguida o sistema solicita ao usuário novamente para que entre com um valor entre 0.1 e 1.0 (o grau de semelhança), ou seja, o quão parecidos devem ser os casos armazenados com o novo caso. O usuário digita 0.8. O sistema primeiro mostra ao usuário os nomes das receitas que conseguiu recuperar e depois as receitas com outras informações (ingredientes, calorias e tempo de preparo). Logo após pede ao usuário que selecione uma destas receitas. É mostrado então ao usuário o modo de preparo da receita que este selecionou. Finalmente, o sistema pergunta ao usuário se deseja aprender esta receita. Como o usuário não deseja aprender esta receita, o sistema pergunta ao usuário se este deseja apresentar mais algum caso novo ao sistema e caso o usuário deseja o sistema recomeça solicitando um novo caso, ou seja, uma nova receita. A seguir é mostrado o primeiro exemplo de execução.

```
Entre com um novo caso: |:
[0,0,1,1,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0].

Digite o grau de semelhanca requerido: |: 0.8.

O sistema recuperou os casos: |: [bolo_de_cenoura,
                                  bolo_de_beterraba,
                                  bolo_de_chocolate].

A tabela abaixo mostra as receitas recuperadas.

Receitas            Ingredientes  Calorias  Tempo de Preparo
bolo_de_cenoura            3        6235         60 min
bolo_de_beterraba          3        6183         60 min
bolo_de_chocolate          3        4636         40 min

Selecione uma receita digitando seu nome: |: bolo_de_chocolate.

Modo de preparo: Bater no liquidificador ate fazer bolhas:
3 ovos, 1 xicara de oleo, 1/2 de acucar, 1 de chocolate, 1 de
agua fervendo e 1 colher de fermento. Acrescentar 2 xicaras de
farinha e misturar. Assar em fogo baixo por 40 minutos.

Deseja aprender este novo caso? |: nao.

Deseja apresentar um novo caso? |: sim.
```

No segundo exemplo de execução quando o sistema não consegue recuperar nenhum caso parecido com o novo caso (os ingredientes do novo caso são leite e batata). O sistema pergunta ao usuário se deseja criar um novo *cluster* para este caso ou modificar o grau de semelhança entre os casos armazenados e o novo caso. O usuário

opta por criar um novo *cluster*. O sistema então pede o nome da receita, os ingredientes que compõem esta receita, o modo de preparo, o total de calorias e o tempo de preparo. Finalmente o sistema pergunta ao usuário se este deseja continuar a apresentar um novo caso.

```
Entre com um novo caso: |:
[0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0].

Digite o grau de semelhanca requerido: |: 0.8.

Nao consigo recuperar nenhum caso!

Voce deseja:

1.Criar novo cluster?
2.Modificar a semelhanca entre os casos?

Digite sua opcao: |: 1.

Nome da receita: |: pure_de_batata.

Ingredientes: |: [0,0,0,0,1,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0].

Modo de preparo: |: Frite uma cebola no oleo ou manteiga.
Acrescente: 1/2 kg de batata cozida e amassada, 1 copo de leite
e sal a gosto. Deixe cozinhar por 10 minutos, mexendo.

Tempo de preparo (em minutos): 10.

Calorias: 1398.

Deseja apresentar um novo caso? |: nao.

yes

|?-
```

---

5 Conclusão

Neste trabalho foi descrito um protótipo de um Sistema de Raciocínio Baseado em Casos. Este protótipo integra Raciocínio Baseado em Casos e Redes Neurais. Um modelo de Redes Neurais, chamado ART1, é responsável por agrupar os casos que inicialmente irão compôr a Base de Casos em *clusters* e posteriormente encontrar o *cluster* mais semelhante para um caso corrente, *matching* parcial. Os casos são organizados sequencialmente na Base de Casos, indexados pelo *cluster* correspondente. Quando um caso corrente não se assemelha a nenhum *cluster* definido anteriormente, a rede pode aprender este novo *cluster* sem detrimento dos *clusters* previamente definidos. Quando isto ocorre diz-se que o sistema de Raciocínio Baseado em Casos aprendeu.

6 Referências

[Aamodt 94] Aamodt, A.; Plaza, E.; *Case-Based Reasoning: Foundational Issues, Methodological Variations and System Approaches.* AI Communications, vol. 7, n. 1, pp. 39-59, Março, 1994.

[Barletta 91] Barletta, R.; *An Introduction to Case-Based Reasoning.* AI Expert, pp. 43-49, 1991.

[Beale 90] Beale, R.; Jackson, T.; *Neural Computing: An Introduction.* Adam Hilger, 1990.

[Kolodner 87] Kolodner, J. L.; *Extendig Problem Solver Capabilities Through Case-Based Inference.* Proceedings of the Fourth International Workshop on Machine Learning, pp 167-178, 1987.

[Kolodner 92] Kolodner, J. L.; *An Introduction to Case-Based Reasoning.* Artificial Intelligence Review. 6, pp. 3-34, 1992.

[Kolodner 93] Kolodner, J. L.; *Cased-Based Reasoning.* Kaufmann, 1993.

[Carpenter 87] Carpenter, G.; Grossberg, S.; *A Massively Parallel Architecture for a Self Organizing Neural Pattern Recognition Machine.* Computer Vision, Graphics, and Image Processing, vol. 37, pp. 54-115, 1987.

[Fausett 94] Fausett, L.; *Fundamentals of Neural Networks. Architectures, Algorithms, and Applications.* Prentice Hall International Editions, 1994.

[Grossberg 76] Grossberg, S.; *Adaptive Pattern Classification and Universal Recoding, II: Feedback, Expectation, Olfacation, and Illusions.* Biological Cybernectics, vol. 23, pp. 187-202, 1976.

[Hammond 89] Hammond, K. J.; *Case-Based Planning Viewing Planning as a Memory Task.* Academic Press, Boston, 1989.

[Heins 95] Heins, L. G.; D. R. Tauritz, D. R.; *Adaptative Resonance Theory (ART): An Introduction.* Maio/Junho, 1995.

[Lopes 95] Lopes, A. A.; *Raciocínio Baseado em Casos.* Dissertação apresentada ao Instituto de Ciências Matemáticas de São Carlos, Julho, 1995.

[Malek 95] Malek, M.; *A Connectionist Indexing Approach for CBR Systems.* In Aamodt, A.; Veloso, M. (Eds.), Preceedings of First International Conference on Case-Based Reasoning Research and Development. Sesimbra, Portugal, pp. 520-527, Outubro, 1995.

[Nakatani 94] Nakatani, Y.; Israel, D.; *Tuning Rules by Cases.* Lecture Notes in Artificial Intelligence, vol. 837, pp. 313-324, 1994.

[Reategui 94] Reategui, E.; Campbell, J. A.; *A Classification System for Credit Card Transactions*. n Keane, M.; Haton, J. pp.; Manago, M. (Eds.), Preceedings of European Workshop on Case-Based Reasoning. Chantilly, França, pp. 167-174, 1994.

[Reategui 95] Reategui, E.; Campbell, J. A.; Borghetti, S.; *Using a Neural Network to Learn General Knowledge in a Case-Based System*. In Aamodt, A.; Veloso, M. (Eds.), Preceedings of First International Conference on Case-Based Reasoning Research and Development. Sesimbra, Portugal, pp. 528-537, Outubro, 1995.