Instituto de Ciências Matemáticas e de Computação

ISSN - 0103-2569

Projeto e Descrição da Implementação Prolog de uma Ferramenta para Extração de Conhecimento de Redes Neurais

Gustavo Enrique de Almeida Prado Alves Batista
Claudia Regina Milaré
Maria Carolina Monard

Nº 54

RELATÓRIOS TÉCNICOS DO ICMC

São Carlos

Março/1997

Projeto e Descrição da Implementação Prolog de uma Ferramenta para Extração de Conhecimento de Redes Neurais[1]

Gustavo Enrique de Almeida Prado Alves Batista
Claudia Regina Milaré
Maria Carolina Monard
Universidade de São Paulo/ILTC
Instituto de Ciências Matemáticas de São Carlos
Departamento de Ciências de Computação e Estatística
Caixa Postal 668, 13560-970 - São Carlos, SP
e-mail:{gbatista, claudia, mcmonard}@icmsc.sc.usp.br

Sumário

Redes Neurais têm sido cada vez mais utilizadas em diversas áreas. Contudo, existem características relevantes que Redes Neurais não possuem, como por exemplo, a capacidade de explicar de uma forma compreensível o processo pelo qual a solução foi obtida.
Existem algumas técnicas para extrair algum conhecimento de Redes Neurais. Uma delas é conhecida como EN. Esta técnica fornece recursos de explicação para responder às seguintes perguntas: *Como?*, *Porque?*, além de prover um mecanismo de *Trace*.
As explicações fornecidas por EN dependem do procedimento adaptado para selecionar os neurônios mais significativos. Neste trabalho propomos outros procedimentos de seleção. Consideramos que pelo menos um desses procedimentos, que se baseia no fato de que a soma das conexões que chegam a um determinado neurônio é mais significativa para a predição de sua ativação do que os valores dos pesos (absolutos ou não) independentes, torna EN mais robusto.
Neste trabalho é apresentada, em detalhes, uma implementação do método de explicação EN juntamente com as modificações propostas para o critério de seleção de pesos. Esta implementação foi realizada em linguagem de programação lógica Prolog. Este trabalho apresenta também exemplos de execução da ferramenta, os quais ilustram as diferenças entre os diversos critérios de seleção implementados.

Março de 1997

[1]Trabalho realizado com auxílio do CNPq e FAPESP.

Sumário

1 Introdução

As aplicações que utilizam Redes Neurais são cada vez mais empregadas no comércio, na ciência e na indústria, o que confirma a relevância deste paradigma. Entre as características que contribuem para sua popularidade, podem ser citadas:

- O modo direto com que Redes Neurais adquirem conhecimento sobre um determinado domínio de problema através de um processo de treinamento.
- A forma compacta (completamente numérica) em que o conhecimento adquirido é armazenado em uma Rede Neural treinada, e a rapidez com que este conhecimento pode ser acessado e usado.
- A robustez de uma solução de Redes Neurais na presença de ruídos nos dados de entrada.

Contudo, Redes Neurais são consideradas como "caixas pretas" por alguns usuários que as utilizam, especialmente quando estes não estão familiarizados com os métodos matemáticos empregados em tais sistemas. Desta forma, a utilização de Redes Neurais pode levar a resultados não desejáveis. Pessoas pouco familiarizadas à computação podem não confiar nos resultados apresentados pelo computador. No melhor dos casos, o usuário sentirá somente uma falta de compreensão de como o resultado está sendo gerado. No pior dos casos, a resposta do computador será aceita sem questionamentos, o que pode levar a desentendimentos e, o pior, os resultados podem ser mal aplicados. Existe também a possibilidade na qual as pessoas não acreditam na veracidade dos resultados e estes são simplesmente ignorados.

Uma forma de contornar este problema é gerar explicações para os usuários de como foi atingida a conclusão. Com isto, Redes Neurais ganhariam maior aceitabilidade e credibilidade entre seus usuários, bem como aumentariam sua utilidade como ferramenta de aprendizado e generalização.

Existem diferentes técnicas para extrair algum conhecimento de Redes Neurais. Rumelhart em [13] estudou as representações internas de Redes Neurais quando desenvolveu o algoritmo backpropagation. Ele utilizou uma rede multi-camadas para armazenar relações entre pessoas de duas famílias, uma inglesa e outra italiana. Neste estudo, ele observou que os pesos das camadas intermediárias resultantes do treinamento representavam informações semânticas importantes do domínio do problema. Observação semelhante foi utilizada em [8], na qual uma Rede Neural é usada para classificar o retorno do som de um cilindro de metal e de uma rocha cilíndrica e em [2], uma rede é usada para aprender a jogar *Bridge*. Análise Sensitiva [10] é outra técnica para extrair conhecimento de uma Rede Neural. Esta técnica consiste em examinar cada uma das entradas e determinar que impacto uma pequena mudança nestas entradas produziria nas saídas. Se uma pequena mudança em uma entrada particular causa uma mudança drástica em uma determinada saída, aquela entrada deve ser considerada um dos principais fatores que influenciam no resultado corrente gerado na saída.

Extrair regras de Redes Neurais também é uma forma de se obter conhecimento. Há vários estudos sobre extração de regras encontrados em [1, 4, 5, 6, 11, 14] e alguns mais recentes sobre técnicas de extração de regras fuzzy em [3, 7, 9].

Este trabalho concentra-se em uma técnica de extração de conhecimento de Redes Neurais conhecida como EN [12], que fornece recursos de explicação para responder às seguintes perguntas sobre uma Rede Neural: ***Porque?, Como? e Trace***. O método EN pode ser muito útil na validação e re-projetamento de Redes Neurais, além de ser independente do domínio da aplicação, o que o torna bastante flexível. Outra vantagem deste método é a facilidade em implementá-lo. Neste trabalho duas alterações no algoritmo original são propostas e implementadas, na linguagem de programação lógica Prolog, visando torná-lo mais robusto.

O trabalho está organizado da seguinte forma: na seção 2 é descrita a facilidade de explicação EN proposta originalmente. Na seção 3 são descritos os métodos de seleção de pesos `ccg` e `soma` propostos neste trabalho. Na seção 4 é descrita, em detalhes, a implementação Prolog do método EN original juntamente com os critérios de seleção propostos. Na seção 5 são apresentados exemplos para ilustrar as diferenças entre as diversas modificações implementadas neste trabalho.

2 A Facilidade de Explicação EN

EN é um método bastante difundido para gerar explições a partir de uma Rede Neural multi-camadas. Este método provê duas facilidades de explicação: *Como?* e *Porque?*, juntamente com uma ferramenta de *Trace* que ajuda a visualizar os neurônios envolvidos em cada explicação.

Uma das características mais interessantes deste método é a independência de qualquer método específico de treinamento de Redes Neurais. A facilidade de explicação EN pode ser aplicada a quase todos os tipos de Redes Neurais, como por exemplo backpropagation, Hopfield, Adaline, etc. Para ser aplicado, este método assume que estão disponíveis os pesos das conexões resultantes do processo de treinamento da Rede Neural.

2.1 Estrutura da Facilidade de Explicação EN

Esta técnica busca satisfazer os seguintes objetivos:

1. O número de neurônios e camadas é na maioria das vezes muito grande e, deve ser feita uma seleção dos caminhos de propagação da rede mais significantes.
2. Em geral, os nós de entrada podem ser descritos segundo alguma natureza, e o método EN deve relacionar estas características com o comportamento de classificação da Rede Neural.

3. As opções de explicação devem incluir (seguindo a terminologia padrão de Sistemas Baseados em Conhecimento):

   - ***Porque?***: relaciona as saídas de uma rede com suas entradas, resultando em uma cadeia de resolução que justifica a alocação de um dado X em uma classe C(X), em termos das contribuições dos nós de entrada para o resultado.
   - ***Como?***: relaciona as entradas de uma rede com suas saídas, resultando em uma cadeia de resolução que explica a alocação da classe C(Y) quando é selecionado como entrada da rede um subconjunto Y de neurônios.
   - ***Trace***: em ambos *Como?* e *Porque?*, gera o subconjunto de neurônios em cada camada envolvidos nas cadeias de resolução.

Deve ser observado que a facilidade de explicação *Como?* e *Porque?* são semanticamente diferentes. A facilidade *Porque?* relaciona as saídas de uma Rede Neural com suas entradas, enquanto que a facilidade *Como?* opera em sentido inverso, isto é, relaciona as entradas de uma Rede Neural com suas saídas.

O mecanismo de explicação proposto depende da combinação dos seguintes conceitos:

1. Pesos maiores contribuem mais, assumindo que as entradas estão normalizadas.
2. Nós de entrada significativos podem ser agrupados, e estes grupos constróem e representam conceitos.
3. Relacionando informações específicas da aplicação para cada neurônio, pode-se associar um significado concreto aos neurônios de entrada.

Deve ser enfatizado que *Porque?, Como?* e o mecanismo *Trace* dependem dos conceitos 1, 2 e 3 juntos, para dar significado aos grupos de neurônios de entrada adequadamente selecionados, independentemente da aplicação. Somente algumas informações, conceitos e regras são específicas da aplicação.

2.2 Funcionamento do Método EN

Considerando uma Rede Neural com N camadas e com $n(l)$ neurônios por camada l, sendo $l = 1, \ldots, N$. A camada $l = 1$ é a camada de entrada, e a camada $l = N$ a camada de saída, com apenas um neurônio para cada $C = n(N)$ classes.

A facilidade de explicação *Como?* busca relacionar um conjunto Y de neurônios de entrada, onde $Y \subset 1, \ldots, n(1)$, a uma classe $C(Y)$. Para isto, no método EN originalmente proposto, são tomados os valores absolutos de todos os pesos ligados aos neurônios do conjunto Y na camada $l = 1$, e destes valores absolutos é selecionada uma fração d dos pesos mais significativos. Esta fração d é chamada *grau de explicação*. Os neurônios ligados aos pesos selecionados da fração d na camada seguinte $l = 2$ serão utilizados na próxima iteração. Este procedimento é repetido até a camada de saída

$l = N$. Os neurônios selecionados na última camada são as classes relacionadas aos neurônios de entrada do conjunto Y.

Neste trabalho é dado ênfase ao critério de seleção dos pesos. No método EN original os pesos são simplesmente selecionados por seus valores absolutos. As modificações implementadas neste trabalho utilizam um processo diferente de seleção, no qual os pesos são agrupados e somados. Este processo será detalhado na seção 3.

O valor do grau de explicação é atribuído pelo usuário do EN. Em casos nos quais este valor é muito pequeno, o produto $d * n(l)$ pode ser zero por motivos de arredondamento inteiro. Nestes casos este valor deve ser modificado para 1.

A facilidade de explicação *Porque?* funciona de forma semelhante, uma vez que o mesmo procedimento é aplicado iniciando da camada de saída $l = N$ até a camada de entrada $l = 1$.

3 Variações no Critério de Seleção dos Pesos

O método EN original considera somente um critério para selecionar os pesos relevantes à explicação solicitada, os pesos são ordenados por seu valor absoluto e uma fração correspondente ao grau de explicação é selecionada para a próxima iteração.

Como já mencionado, neste trabalho são propostos outros critérios para selecionar os pesos a fim de pesquisar o comportamento do método EN em função desses critérios.

Uma primeira modificação, muito simples, denominada "ccg", é a ordenação pelos valores dos pesos. É importante observar que tanto nesta primeira variação proposta, quanto no método EN original, os pesos são ordenados e selecionados como se fossem conexões indepentendes da Rede Neural, estas variações não analisam os pesos como partes integrantes dos neurônios da rede.

Uma outra variação proposta, que consideramos mais robusta por analisar os pesos de um mesmo neurônio como um todo, é denominada "soma". Nesta variação os pesos de cada neurônio são agrupados e somados a cada iteração para logo após realizar a ordenação. Este critério é detalhado a seguir.

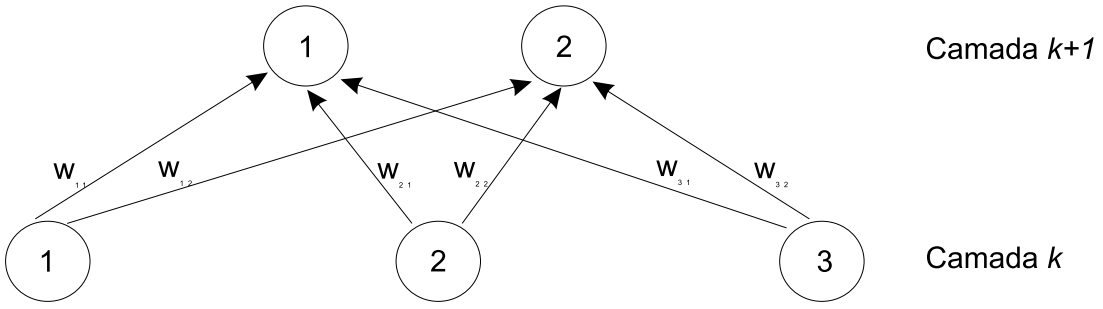

Figura 1: Rede Neural a ser Consultada

A Figura 1 mostra duas camadas de uma Rede Neural qualquer. A camada k possui

três neurônios e a camada $k+1$ dois neurônios, interligados pelas conexões w_{ij}, onde i indica o índice do neurônio fonte e o j o índice do neurônio destino. Para o mecanismo de explicação *Como?* os pesos devem ser agrupados em conjuntos de pesos que chegam aos neurônios da camada $k+1$. No caso específico da Figura 1 os pesos serão separados em dois conjuntos, no qual o primeiro conterá os pesos das conexões entre a camada k e o neurônio 1 da camada $k+1$, ou seja, w_{11}, w_{21}, w_{31}. Por sua vez, o segundo conjunto conterá os pesos das conexões entre a camada k e o neurônio 2 da camada $k+1$, ou seja, w_{12}, w_{22}, w_{32}. Em seguida são somados os pesos dos conjuntos separadamente, resultando em dois totais $(w_{11} + w_{21} + w_{31})$ e $(w_{12} + w_{22} + w_{32})$. Estes totais são ordenados por valor e é selecionada uma fração de neurônios correspondentes ao grau de explicação que farão parte da próxima iteração.

A facilidade de explicação *Porque?* funciona de forma semelhante à facilidade *Como?* descrita anteriormente, com excessão que a facilidade *Porque?* deve relacionar os neurônios de saída com os neurônios de entrada. Desta forma, na facilidade *Porque?*, o método de explicação deve iniciar selecionando os neurônios da última camada formando a cadeia de resolução até a primeira camada.

A modificação "soma" baseia-se no fato de que a soma das conexões que chegam em um determinado neurônio é mais significativa para a predição de sua ativação do que os valores dos pesos (absolutos ou não) independentes.

Por exemplo, em um caso particular pode ocorrer que duas conexões que chegam a um neurônio possuem pesos com valores altos e próximos, mas com sinais contrários. No método EN original, este neurônio é um forte candidato a ser selecionado para a próxima iteração. Já no método EN modificado, utilizando o critério "soma", este neurônio não teria muitas chances de ser selecionado, pois o resultado da soma dos pesos das conexões teria um valor baixo e este neurônio poderia perder para qualquer outro que tivesse maior soma dos pesos.

4 Implementação

O método EN, bem como as alterações propostas, foram implementadas na linguagem de programação lógica Prolog. Esta linguagem foi escolhida por oferecer uma grande facilidade para a criação e rápida implementação de protótipos.

Um procedimento Prolog consiste de um conjunto de cláusulas referentes a uma mesma relação ou predicado. Predicados são notados pelo seu nome e por sua aridade. Por exemplo a notação `abc/4` refere-se ao predicado `abc` e este predicado possui quatro argumentos, ou seja, aridade 4. Esta convenção é utilizada neste texto. Além disso, quando for conveniente, os argumentos dos procedimentos serão descritos da seguinte forma:

$[+] < arg_n >$: o n-ésimo argumento é de entrada;

$[-] < arg_n >$: o n-ésimo argumento é de saída;

[?] $< arg_n >$: o n-ésimo argumento é de entrada/saída.

Cada sistema de treinamento de Redes Neurais tem sua própria estrutura. A conversão de uma estrutura qualquer para a estrutura que esta implementação Prolog utiliza, pode ser realizada por um pré-processador muito simples que transforma as conexões da rede treinada em uma lista na sintaxe Prolog. Cada elemento desta lista corresponde a uma conexão da Rede Neural original. Estes elementos devem seguir o seguinte formato:

```
w(Camada, Fonte, Destino, Peso)
```

onde:

`Camada`: Camada onde o neurônio está localizado.

`Fonte`: Índice do neurônio fonte.

`Destino`: Índice do neurônio destino.

`Peso`: Valor do peso da conexão.

Como Redes Neurais geralmente apresentam muitas conexões, este pré-processador pode ainda eventualmente simplificar, de alguma forma, (por exemplo, podando conexões muito "fracas") a Rede Neural original.

Nesta implementação do método EN, as facilidades de explicação, bem como o critério a ser utilizado para selecionar os pesos relevantes da Rede Neural, estão implementados pelo procedimento de nível mais alto chamado `explica/6` que consiste de uma única cláusula, na qual os primeiros cinco argumentos são de entrada.

Procedimento 4.1 `explica/6`

```
explica(Criterio, Trace, Grau, CP, RN, Expl):-
    condicao_parada(CP, RN, Camada_Parada),
    explica(Criterio, Trace, Grau, CP, RN, Camada_Parada, Expl),
    informe(CP, Expl).
```

Os argumentos de `explica/6` são

[+] $< arg_1 >$: Critério de seleção de pesos

[+] $< arg_2 >$: Liga ou desliga o mecanismo de trace

[+] $< arg_3 >$: Grau de explicação ($0 \leq$ `Grau` ≤ 1).

[+] $< arg_4 >$: Tipo de explicação *Como?* ou *Porque?*

[+] $< arg_5 >$: Lista com os pesos da Rede Neural

[−] $< arg_6 >$: Lista com os índices dos neurônios de resposta

O primeiro argumento indica o critério de seleção de pesos a ser utilizado, este argumento é válido se instanciado com os seguintes átomos

- `pau` para executar o método EN sem alterações,
- `ccg` para executar o método EN por valor não absoluto e,
- `soma` para executar o método EN modificado para somar os pesos.

O segundo argumento diz respeito a impressão, ou não, de informações de trace, este argumento instanciado com o átomo

- `sim` indica que o usuário deseja utilizar o mecanismo de trace e,
- `nao` faz com que o método não apresente as informações de trace.

O quarto argumento é um funtor do tipo

```
porque(Camada, Neuronio)
```

ou

```
como(Camada, Neuronio)
```

o qual indica o tipo de explicação — *Como?* ou *Porque?* — a ser utilizado. Este funtor possui dois argumentos, o segundo argumento é uma lista dos neurônios que devem ser explicados pelo sistema, o primeiro argumento é um valor inteiro representando a camada onde os neurônios a serem explicados se encontram.

O procedimento `explica/6` é sempre bem sucedido com o último parâmetro `Expl` unificado com a explicação correspondente.

No que se segue será utilizada uma Rede Neural muito simples para ilustrar o comportamento de cada procedimento do método de explicação. Esta Rede Neural, esboçada na Figura 2, possui três neurônios na camada de entrada — Camada 1 — e dois neurônios na camada de saída — Camada 2. Os neurônios de cada camada estão identificados por números inteiros[2] e existem conexões ligando todos os neurônios da camada de entrada a todos os neurônios da camada se saída, totalizando 6 conexões.

Serão criadas também duas consultas a serem resolvidas durante as explicações sobre o funcionamento de cada procedimento. A primeira consulta refere-se ao critério de seleção de pesos `pau` utilizando explicação do tipo *Como?*. A segunda consulta envolve o critério de seleção de pesos `ccg` e a explicação do tipo *Porque?*. A primeira consulta pode ser verbalizada da seguinte forma

[2]Na realidade, os neurônios não necessariamente precisam ser indentificados por números inteiros, pode-se utilizar átomos alfanumérios.

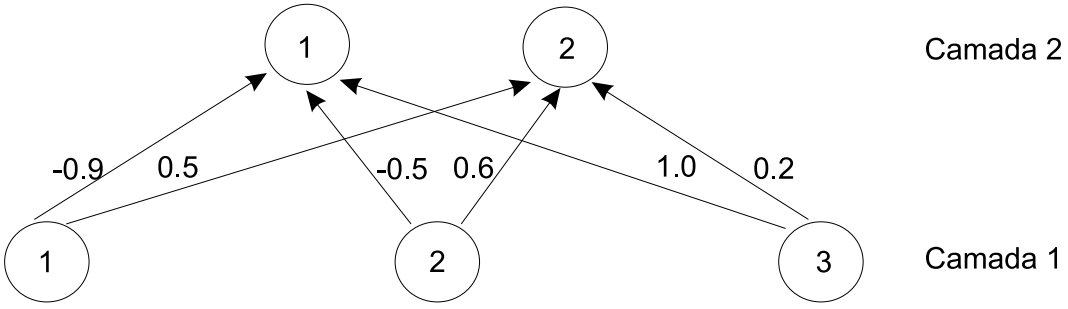

Figura 2: Rede Neural Exemplo

> *Quais os neurônios da camada de saída que estão mais "fortemente" ligados com os neurônios 1 e 3 da camada de entrada? A opção de trace deve estar ligada e o grau de explicação deve ser de 0.5.*

Esta consulta é equivalente a seguinte interrogação Prolog `explica/6` da seguinte forma

```
?- explica(pau, sim, 0.5, como(1, [1,3]), [w(1,1,1,-0.9), w(1,1,2,0.5),
     w(1,2,1,-0.5), w(1,2,2,0.6), w(1,3,1,1), w(1,3,2,0.2)], Expl).
```

Como foi dito anteriormente, `como(1, [1,3])` é um funtor que contém como segundo argumento a lista com os índices dos neurônios a serem explicados, no caso do exemplo a lista contém os elementos 1 e 3. O primeiro parâmetro do funtor `como` é a camada onde se encontram os neurônios a serem explicados, neste caso os neurônios se encontram na camada de entrada — camada 1.

Após a execução do procedimento `explica/6` ser bem sucedida, o último argumento — argumento saída — será instanciado com os índices dos neurônios que estão mais "fortemente" ligados aos neurônios 1 e 3.

A segunda consulta a ser resolvida envolve o critério de seleção de pesos `ccg` e a explicação do tipo *Porque?*. A consulta pode ser verbalizada da seguinte forma

> *Quais os neurônios da camada de entrada que estão mais fortemente ligados com o neurônio 1 da camada de saída. O opção de trace deve estar ligada e o grau de explicação deve ser 0.1*

Esta consulta é equivalente a seguinte interrogação Prolog

```
?- explica(ccg, sim, 0.1, porque(2, [1]), [w(1,1,1,-0.9), w(1,1,2,0.5),
     w(1,2,1,-0.5), w(1,2,2,0.6), w(1,3,1,1), w(1,3,2,0.2)], Expl).
```

Nesta consulta é utilizado o funtor `porque` que possui os mesmos argumentos do funtor `como`, ou seja, a camada e os índices dos neurônios a serem explicados. Neste exemplo, a camada onde se encontram os neurônios a serem explicados é a camada de saída

— camada 2 — e, os índices dos nerônios a serem explicados estão representados pela lista que possui apenas o neurônio de índice 1.

O procedimento `explica/6` — pg. 6 ativa os procedimentos `condicao_parada/4` que determina a última camada a ser tratada e, logo após o procedimento `explica/7` o qual é responsável por todo o processo de explicação. Finalmente, o procedimento `explica/6` ativa o procedimento `informe/2` que informa a explicação encontrada.

Procedimento 4.2 `condicao_parada/4`

```
condicao_parada(porque(_,_), _, 0):-!.
condicao_parada(como(_,_), RN, Maior):-
     total_camadas(RN, Maior).
```

Os argumentos de `condicao_parada/4` são

[+] $< arg_1 >$: Funtor que indica o tipo de explicação *Como?* ou *Porque?*

[+] $< arg_2 >$: Lista de pesos da Rede Neural

[−] $< arg_3 >$: Camada de parada

O procedimento `condicao_parada/4` é responsável por encontrar o número da última camada da Rede Neural a ser analizada pelo método iterativo. As camadas de uma Rede Neural são numeradas em ordem crescente a partir da camada de entrada — camada 1— até a camada de saída. O procedimento `condicao_parada/4` trabalha com duas possibilidades distintas

1. A primeira possibilidade, coberta pelo primeiro predicado do procedimento, ocorre quando é utilizada uma explicação do tipo *Porque?*. As explicações do tipo *Porque?* buscam encontrar os neurônios da camada de entrada que estão mais "fortemente" ligados com certos neurônios da camada de saída. Desta forma o método deve começar selecionado neurônios da camada de saída e a cada iteração selecionar neurônios das camadas intermediárias até atingir a camada de entrada, ou seja, a camada 1. Desta forma, quando o sistema atinge a camada 0 não há mais camadas a serem analisadas e o processo de explicação chegou ao fim.

2. A segunda possibilidade a ser tratada pelo procedimento é quando se está utilizando a explicação do tipo *Como?*. Esta explicação busca encontrar quais neurônios da camada de saída estão ligados a determinados neurônios de entrada através de conexões robustas. Desta forma, o método deve partir da camada de entrada e caminhar através das conexões mais robustas até encontrar a camada de saída. Portanto neste tipo de explicação, quando o sistema atinge a última camada da Rede Neural mais um é um sinal que não há mais camadas a serem analisadas e o processo de explicação chegou ao fim. Entretanto, o valor da última camada de uma Rede Neural varia conforme a arquitetura adotada pelo usuário. Para se

obter este valor é necessário analisar a lista de pesos da Rede Neural e procurar por um peso que liga a última camada intermediária com a camada de saída. Esta análise é realizada pelos procedimentos `total_camadas/2` e `total_camadas/3`.

Procedimento 4.3 `total_camadas/2`

```
total_camadas([w(Camada, _, _, _)|Cauda], Maior):-
     total_camadas(Cauda, Camada, Aux),
     Maior is Aux+1.
```

Os argumentos de `total_camadas/2` são

[+] < arg_1 >: Lista de pesos da Rede Neural

[−] < arg_2 >: Maior valor de camada encontrado na lista de pesos

O procedimento `total_camadas/2` tem o papel de inicializar os parâmetros e executar o procedimento `total_camadas/3`, o qual realmente encontra o maior valor de camada entre as camadas da Rede Neural. Para inicializar o procedimento `total_camadas/3`, o procedimento `total_camadas/2` retira o valor da camada do primeiro peso contido na lista de pesos da Rede Neural — primeiro argumento. Este valor de camada é utilizado para inicializar o segundo argumento do procedimento `total_camadas/3`.

Procedimento 4.4 `total_camadas/3`

```
total_camadas([], MaiorAtual, MaiorAtual).
total_camadas([w(Camada, _, _, _)|Cauda], MaiorAtual, Maior):-
     Camada > MaiorAtual, !,
     total_camadas(Cauda, Camada, Maior).
total_camadas([w(Camada, _, _, _)|Cauda], MaiorAtual, Maior):-
     total_camadas(Cauda, MaiorAtual, Maior).
```

Os argumentos de `total_camadas/3` são

[+] < arg_1 >: Lista de pesos da Rede Neural

[+] < arg_2 >: Maior valor de camada encontrado na lista de pesos durante a execução do procedimento

[−] < arg_3 >: Maior valor de camada encontrado na lista de pesos

O primeiro predicado do procedimento `total_camadas/3` é responsável pela parada de recursão, este predicado é bem sucedido quando o primeiro argumento — lista de pesos da Rede Neural — é uma lista vazia. Neste caso, todos os pesos já foram processados

e o maior valor de camada encontrado até então é o valor da última camada, este valor é instanciado com o terceiro argumento — argumento de saída — e o procedimento encerra sua execução.

O segundo predicado é executado com sucesso quando o valor da camada do peso na cabeça da lista de pesos da Rede Neural — primeiro argumento — é maior que o maior valor de camada encontrado até o momento — segundo argumento. Neste caso, o procedimento é chamado recursivamente, mas agora o valor de camada do peso na cabeça da lista de pesos da Rede Neural é passado como sendo o maior valor de camada encontrado até o momento — segundo argumento.

O terceiro predicado é executado quando o segundo predicado não foi bem sucedido, ou seja, o valor da camada do peso que está na cabeça da lista de pesos da Rede Neural é menor ou igual ao maior valor de camada encontrado até o momento. Neste caso deve-se simplesmente analisar o próximo peso através de uma chamada recursiva ao procedimento, sem alterar o segundo argumento.

Voltando ao procedimento 4.1 `explica/6`, após o cálculo da camada de parada este procedimento executa o procedimento `explica/7`. O procedimento `explica/7` consiste de 6 cláusulas. As duas primeiras cláusulas são responsáveis pela parada de recursão para as explicações *Como?* e *Porque?*, respectivamente. As duas cláusulas seguintes são responsáveis pela implementação das explicações *Como?* e *Porque?* nos critérios "pau" e "ccg", e as duas últimas cláusulas pelo critério "soma". Inicialmente, serão discutidos apenas os predicados referentes aos critérios de seleção de pesos "pau" e "ccg", pois estes predicados possuem procedimentos comuns. Posteriormente, nesta seção, serão discutidos também os dois últimos predicados referentes ao critério de seleção de pesos "soma".

Segue a listagem do procedimento `explica/7` sem os dois últimos predicados, os quais se referem ao critério de seleção de pesos "soma".

Procedimento 4.5 `explica/7`

```
explica(_, _, _, como(Camada, Indice_Neuronios), _, Camada,
          Indice_Neuronios):-!.
explica(_, _, _, porque(Camada, Indice_Neuronios), _, Camada,
          Indice_Neuronios):-!.
explica(Criterio, Trace, Grau, como(Camada, Neuronios), RN,
          Camada_Parada, Expl):-
     metodo(Criterio), !,
     criterio_ordenacao(Criterio, Cri_Ord),
     pesos_camada(como, Camada, Neuronios, RN, Pesos_Camada),
     sortw(Cri_Ord, Pesos_Camada, Pesos_Ordenados_Camada),
     trace(prox, Trace, Camada, Pesos_Ordenados_Camada),
     seleciona_neuronios(Grau, Pesos_Ordenados_Camada,
          Neuronios_Selecionados),
     trace(pesos, Trace, Camada, Neuronios_Selecionados),
     indice_neuronios(como, Neuronios_Selecionados, Indice_Neuronios),
     trace(neuronios, Trace, Camada, Indice_Neuronios),
     Proxima_Camada is Camada + 1,
     explica(Criterio, Trace, Grau, como(Proxima_Camada,
          Indice_Neuronios), RN, Camada_Parada, Expl).
explica(Criterio, Trace, Grau, porque(Camada, Neuronios), RN,
          Camada_Parada, Expl):-
     metodo(Criterio), !,
     criterio_ordenacao(Criterio, Cri_Ord),
     pesos_camada(porque, Camada, Neuronios, RN, Pesos_Camada),
     sortw(Cri_Ord, Pesos_Camada, Pesos_Ordenados_Camada),
     trace(prox, Trace, Camada, Pesos_Ordenados_Camada),
     seleciona_neuronios(Grau, Pesos_Ordenados_Camada,
          Neuronios_Selecionados),
     trace(pesos, Trace, Camada, Neuronios_Selecionados),
     indice_neuronios(porque, Neuronios_Selecionados, Indice_Neuronios),
     trace(neuronios, Trace, Camada, Indice_Neuronios),
     Proxima_Camada is Camada - 1,
     explica(Criterio, Trace, Grau, porque(Proxima_Camada,
          Indice_Neuronios), RN, Camada_Parada, Expl).
```

Os argumentos de `explica/7` são

$[+] < arg_1 >$: Critério de seleção de pesos

$[+] < arg_2 >$: Liga ou desliga o mecanismo de trace

$[+] < arg_3 >$: Grau de explicação ($0 \leq$ `Grau` ≤ 1)

$[+] < arg_4 >$: Tipo de explicação *Como?* ou *Porque?*

$[+] < arg_5 >$: Lista dos pesos da Rede Neural

$[+] < arg_6 >$: Camada de Parada

$[-] < arg_7 >$: Neurônios de resposta

Como já foi dito, os dois primeiros predicados são responsáveis pela parada de recursão do procedimento, para as explicações *Como?* e *Porque?* respectivamente. Como pode ser visto, estes dois predicados são executados com sucesso quando o primeiro argumento do funtor `como` ou `porque`, o qual armazena a camada que está sendo atualmente analisada, é igual a camada de parada, que é armazenada no sexto argumento. Neste caso, o segundo argumento do funtor `como` ou `porque`, o qual armazena a lista de neurônios selecionados na camada atual, é instanciado com a variavel de saída armazenada no sétimo argumento.

O terceiro e quarto predicados são responsáveis por implementar as explicações do tipo `pau` e `ccg`. A primeira cláusula destes predicados chama o procedimento `metodo/1` que verifica se o argumento `Criterio` instancia ou com o átomo `pau` ou com o átomo `ccg`. O procedimento `metodo/1` está definido da seguinte forma

Procedimento 4.6 `metodo/1`

```
metodo(pau).
metodo(ccg).
```

O único argumento de `metodo/1` é

$[+] < arg_1 >$: Critério de seleção de pesos

Após executar o procedimento `metodo/1`, a segunda cláusula do procedimento `explica/7` executa o procedimento `criterio_ordenacao/2`, cuja função é fornecer ao programa o tipo de ordenação de pesos — por valor ou por valor absoluto — que deve ser utilizada levando em consideração o método de explicação que está sendo empregado. Foram implementadas três possibilidades

- Método `pau`, o qual utiliza ordenação de pesos por valor absoluto;
- Método `ccg`, o qual utiliza ordenação de pesos por valor;
- Método `soma`, o qual utiliza ordenação de pesos por valor.

Segue a implementação do procedimento `criterio_ordenacao/2`

Procedimento 4.7 `criterio_ordenacao/2`

```
criterio_ordenacao(pau, va).
criterio_ordenacao(ccg, v).
criterio_ordenacao(soma, v).
```

Os argumentos de `criterio_ordenacao/2` são

[+] < arg_1 >: Critério de seleção de pesos

[−] < arg_2 >: `va` para ordenação de pesos por valor absoluto e, `v` para ordenação de pesos por valor

O procedimento `explica/7` chama o procedimento `criterio_ordenacao/2` passando como primeiro parâmetro a variável `Criterio`, a qual está instanciada com um átomo que identifica o critério de seleção de pesos a ser utilizado. Como segundo parâmetro é utilizado a variável livre `Cri_Ord` que será instanciada com o átomo `va` para ordenação de pesos por valor absoluto, ou com o átomo `v` para odenação de pesos por valor. Posteriormente, o valor da variável `Cri_Ord` será utilizado na chamada do procedimento `sortw/3` responsável pela ordenação de pesos.

No início desta seção foram propostas duas consultas a serem resolvidas no decorrer desta seção. Na primeira chamada desejava-se saber quais neurônios na camada de saída estão fortemente ligados aos neurônios 1 e 3 da camada de entrada, esta é uma explicação do tipo *Como?*. O primeiro passo para resolver esta consulta é separar da lista geral de pesos da Rede Neural somente os pesos que partem dos neurônios 1 e 3 da camada de entrada. Tais pesos são

```
[w(1,1,1,-0.9), w(1,1,2,0.5), w(1,3,1,1), w(1,3,2,0.2)]
```

Na segunda consulta foi formulada uma explicação do tipo *Porque?* onde deseja-se saber quais neurônios da camada de entrada estão mais "fortemente" ligados ao neurônio 1 da camada de saída. Neste caso é necessário fazer o processo inverso da explicação *Como?*. Deve-se separar da lista de pesos da Rede Neural somente os pesos que chegam aos neurônios que devem ser explicados, no caso do segundo exemplo o neurônio a ser explicado é o neurônio de número 1.

Os pesos que chegam ao neurônio 1 da camada de saída são

```
[w(1,1,1,-0.9), w(1,2,1,-0.5), w(1,3,1,1)]
```

É exatamente esta a função do procedimento `pesos_camada/5`, encontrar os pesos que partem — no caso da explicação tipo *Como?* — ou chegam — no caso da explicação do tipo *Porque?* — a um grupo de neurônios em uma certa camada da Rede Neural.

Segue a listagem do procedimento `pesos_camada/5`.

Procedimento 4.8 `pesos_camada/5`

```
pesos_camada(_, _, [], _, []).
pesos_camada(CP, Camada, [Cab|Cauda], L, Lresp):-
     pesos_camada2(CP, Camada, Cab, L, L1),
     pesos_camada(CP, Camada, Cauda, L, L2),
     concatena(L1, L2, Lresp).
```

Os argumentos de `pesos_camada/5` são

[+] $< arg_1 >$: Tipo de explicação, átomos válidos: `como` ou `porque`

[+] $< arg_2 >$: Camada em que os pesos serão selecionados

[+] $< arg_3 >$: Lista com os índices dos neurônios a serem explicados

[+] $< arg_4 >$: Lista com todos os pesos da Rede Neural

[−] $< arg_5 >$: Lista com os pesos que chegam ou partem dos neurônios contidos no terceiro argumento

O primeiro predicado deste procedimento é responsável pela parada de recursão, ele é executado com sucesso quando não existem mais índices de neurônios na lista de neurônios a serem explicados — terceiro argumento. Neste caso, o quinto argumento — argumento de saída — é instanciado com uma lista vazia.

O segundo predicado é responsável por retirar um a um os índices dos neurônios contidos na lista de neurônios a serem explicados — terceiro argumento — e passá-los para o procedimento `pesos_camada2/5`. Para cada índice de neurônio, o procedimento `pesos_camada2/5` retorna uma lista com os pesos relacionados com tal índice. Resta ao procedimento `pesos_camada/5` concatenar todas as listas retornadas pelo procedimento `pesos_camada2/5` para formar a lista de resposta.

Segue a listagem do procedimento `pesos_camada2/5`

Procedimento 4.9 `pesos_camada2/5`

```
pesos_camada2(_, _, _, [], []).
pesos_camada2(como, Cam, N, [w(Cam, N, X, Y)|Cauda], L):-
     !,
     pesos_camada2(como, Cam, N, Cauda, L2),
     insere(w(Cam, N, X, Y), L2, L).
pesos_camada2(como, Cam, N, [C|Cauda], L):-
     pesos_camada2(como, Cam, N, Cauda, L).
pesos_camada2(porque, Cam, N, [w(Cam, X, N, Y)|Cauda], L):-
     !,
     pesos_camada2(porque, Cam, N, Cauda, L2),
     insere(w(Cam, X, N, Y), L2, L).
pesos_camada2(porque, Cam, N, [C|Cauda], L):-
     pesos_camada2(porque, Cam, N, Cauda, L).
```

Os argumentos de `pesos_camada2/5` são semelhantes aos argumentos do procedimento `pesos_camada/5`, com excessão que o terceiro argumento não é uma lista de índices de neurônios, mas um único índice de um neurônio.

[+] $< arg_1 >$: Tipo de explicação, átomos válidos: `como` ou `porque`

[+] $< arg_2 >$: Camada em que os pesos serão selecionados

[+] $< arg_3 >$: Índice de um dos neurônios a serem explicados

[+] $< arg_4 >$: Lista com todos os pesos da Rede Neural

[−] $< arg_5 >$: Lista com os pesos que chegam ou partem do neurônio expecificado no terceiro argumento

O primeiro predicado deste procedimento é responsável pela parada de recursão, ele será executado com sucesso quando todos os pesos da lista de pesos da Rede Neural — quarto argumento — tiverem sido analisados, ou seja, quando o quarto argumento for uma lista vazia. Neste caso, este predicado instancia o quinto argumento — argumento de saída — com uma lista vazia.

O segundo e terceiro predicados são responsáveis por encontrar os pesos que partem de um certo neurônio, desta forma estes predicados são úteis na explicação do tipo *Como?*. O quarto e quinto predicados são responsáveis por encontrar os pesos que chegam a um certo neurônio, estes predicados são utilizados nas explicações do tipo *Porque?*. Para executar de forma correta, estes predicados aceitam como primeiro parâmetro um átomo diferenciador, este átomo é `como` para explicações do tipo *Como?* e `porque` para explicações do tipo *Porque?*

O segundo predicado é bem sucedido quando o peso que está na cabeça da lista no quarto parâmetro é um peso que parte do mesmo neurônio que aquele especificado no terceiro argumento e está na mesma camada que aquela especificada no segundo argumento. Neste caso, o procedimento é chamado recursivamente para encontrar outros pesos que também possuem as mesmas características e por fim cada um destes pesos é inserido em uma lista de resposta através do procedimento `insere/2`.

O terceiro predicado é executado quando o segundo predicado não é bem sucedido, isto ocorre quando o peso que está na cabeça da lista não parte do neurônio especificado no terceiro argumento, ou quando o peso não está na camada especificada no segundo argumento. Neste caso, o procedimento é chamado recursivamente e o peso não é inserido na lista de resposta.

O quarto e quinto predicados funcionam de forma semelhante ao segundo e terceiro predicados, com a diferença que o peso que está na cabeça da lista de pesos da Rede Neural deve chegar ao neurônio especificado no terceiro parâmetro e deve estar localizado na camada especificada no segundo argumento.

Voltando ao procedimento 4.5 `explica/7`, após separar todos os pesos que partem ou chegam a um certo grupo de neurônios em uma camada específica, deve-se ordenar estes pesos. Para isto foi criado um procedimento chamado `sortw/3`, este procedimento pode ordenar os pesos por valor ou por valor absoluto conforme o átomo especificado no primeiro argumento, `v` para valor e `va` para valor absoluto. A ordenação deve ser por ordem decrescente, ou seja, os pesos mais "pesados" devem ficar no início da

lista e os pesos menos significantes devem ficar no fim da lista. Nesta implementação optou-se por utilizar o método de ordenação Quicksort, por ser bastante simples de ser implementado em Prolog, apesar de não ser muito eficiente por haver uma dupla chamada recursiva em seu código.

Segue a implementação do procedimento `sortw/3` e seus procedimentos auxiliares

Procedimento 4.10 `sortw/3`

```
sortw(_, [], []).
sortw(Ind, [X|Cau], ListaOrd):-
     particiona(Ind, X, Cau, Men, Mai),
     sortw(Ind, Men, MenOrd),
     sortw(Ind, Mai, MaiOrd),
     concatena(MenOrd, [X|MaiOrd], ListaOrd).
particiona(_, _, [], [], []).
particiona(Ind, X, [Y|Cau], [Y|Men], Mai):-
     menor(Ind, X, Y),
     !,
     particiona(Ind, X, Cau, Men, Mai).
particiona(Ind, X, [Y|Cau], Men, [Y|Mai]):-
     particiona(Ind, X, Cau, Men, Mai).
menor(va, w(C, _, _, Peso1), w(C, _, _, Peso2)):-
     abs(Peso1) < abs(Peso2), !.
menor(v, w(C, _, _, Peso1), w(C, _, _, Peso2)):-
     Peso1 < Peso2.
```

Os argumentos de `sortw/3` são

$[+] < arg_1 >$: Tipo de ordenação, `v` por valor ou `va` por valor obsoluto

$[+] < arg_2 >$: Lista de pesos a serem ordenados

$[-] < arg_3 >$: Lista de pesos ordenados

Deve ser chamada a atenção sobre o procedimento auxiliar `menor/3`, este procedimento é constituído de dois predicados, o primeiro predicado é utilizado na ordenação por valor absoluto de peso e o segundo predicado na ordenação por valor normal de peso.

O procedimento `explica/7` utiliza o procedimento `sortw/3` para ordenar a lista de pesos criada pelo procedimento `pesos_camada/5`. No caso da primeira consulta criada no início desta seção, os pesos separados pelo procedimento `pesos_camada/5` foram

```
[w(1,1,1,-0.9), w(1,1,2,0.5), w(1,3,1,1), w(1,3,2,0.2)]
```

Esta lista está instanciada com a variável `Pesos_Camada`, a qual é passada como parâmetro ao procedimento `sortw/3` para ordenação. A lista ordenada é instanciada com a variável `Pesos_Ordenados_Camada`, no caso da primeira consulta, os pesos ordenados estão na seguinte ordem

```
[w(1,3,1,1), w(1,1,1,-0.9), w(1,1,2,0.5), w(1,3,2,0.2)]
```

Para a segunda consulta, os pesos selecionados pelo procedimento `pesos_camada/5` foram

```
[w(1,1,1,-0.9), w(1,2,1,-0.5), w(1,3,1,1)]
```

Nesta consulta foi escolhido o critério de seleção de pesos "ccg", que utiliza a ordenação de pesos por valor, desta forma os pesos ordenados estão na seguinte ordem

```
[w(1,3,1,1), w(1,2,1,-0.5), w(1,1,1,-0.9)]
```

Tão logo os pesos foram separados e ordenados, o procedimento `explica/7` chama o procedimento `trace/4`, o qual é responsável por imprimir informações sobre o andamento do processo de explicação. No procedimento `trace/4` o primeiro parâmetro é utilizado para indicar quais informações de trace devem ser impressas. O procedimento `trace/4` é chamado pelo procedimento `explica/7` em três contextos diferentes e, a cada chamada necessita que informações diferentes sejam impressas. Na primeira ocasião que o procedimento `trace/4` é chamado, é passado como primeiro parâmetro o átomo `prox`, o qual indica que devem ser impressas informações sobre os pesos selecionados e ordenados. As demais chamadas ao procedimento `trace/4` serão explicadas no decorrer desta seção.

Segue a implementação do procedimento `trace/4`

Procedimento 4.11 `trace/4`

```
trace(_, nao, _, _):- !. %verde
trace(prox, sim, Camada, Pesos):-
     write('Pesos da Camada: '),
     write(Camada),
     nl,
     write('Pesos: '),
     write(Pesos),
     nl.
trace(pesos, sim, Camada, Pesos):-
     write('Pesos Selecionados na Camada: '),
     write(Camada),
     nl,
     write('Pesos: '),
     write(Pesos),
     nl.
trace(neuronios, sim, Camada, Neuronios):-
     write('Neuronios Selecionados na Camada: '),
     write(Camada),
     nl,
     write('Neuronios: '),
     write(Neuronios),
     nl.
```

Os argumentos de `trace/4` são

[+] $< arg_1 >$: Os átomos `prox`, `pesos` e `neuronios` que indicam quais informações devem ser impressas

[+] $< arg_2 >$: O átomo `sim` habilita a impressão de informações, o átomo `nao` desabilita

[+] $< arg_3 >$: Camada da Rede Neural na qual se encontra o processo de explicação

[+] $< arg_4 >$: Lista de pesos ou neurônios a ser impressos

O próximo procedimento a ser chamado pelo procedimento `explica/7` é o procedimento `seleciona_neuronios/3`. Este procedimento tem como objetivo selecionar os pesos mais "fortes", utilizando o grau de explicação como índice de corte.

O procedimento `seleciona_neuronios/3` foi implementado da seguinte forma

Procedimento 4.12 `seleciona_neuronios/3`

```
seleciona_neuronios(Grau, Pesos_Ordenados_Camada, Neuronios_Selecionados):-
     nrelementos(Pesos_Ordenados_Camada, Nr_Elementos),
     N is (Grau * Nr_Elementos),
     arredonda(N, NL),
     retira_primeiros(NL, Pesos_Ordenados_Camada, Neuronios_Selecionados).
```

Os argumentos de `seleciona_neuronios/3` são

[+] < arg_1 >: Grau de explicação

[+] < arg_2 >: Lista com os pesos separados e ordenados

[−] < arg_3 >: Lista com os pesos mais "fortes" acima do índice de corte

O procedimento `seleciona_neuronios/3` é composto de apenas um predicado, o primeiro procedimento que este procedimento chama é o procedimento `nrelementos/2`. O procedimento `nrelementos/2` é um procedimento bastante conhecido e simples, seu papel é contar quantos elementos existem em uma lista passada como primeiro parâmetro, o número de elementos desta lista é retornado no segundo argumento. Tão logo `seleciona_neuronios/3` encontra o número de pesos contidos na lista, este valor é multiplicado pelo grau de explicação, para se obter então quantos pesos serão selecionados acima do índice de corte. Este valor — armazenado na variável `N` — é arredondado através do procedimento `arredonda/2`, o procedimento `arredonta/2` possui dois argumentos, o primeiro é um número fracionado e o segundo argumento – argumento de saída — retorna o número arredondado sem casas decimais. Uma vez que o método de explicação exige que pelo menos um peso seja selecionado acima do índice de corte, o procedimento `arredonda/2` foi implementado de forma que o número arredondado seja maior ou igual a 1.

O procedimento `arredonda/2` foi implementado de seguinte forma

Procedimento 4.13 `arredonda/2`

```
arredonda(X, 1.0):-
     Z is round(X,0),
     Z < 1, !.
arredonda(X, Z):-
     Z is round(X,0).
```

Os argumentos de `arredonda/2` são

[+] < arg_1 >: Número fracionado

[−] < arg_2 >: Número arredondado maior ou igual a um

O procedimento `seleciona_neuronios/3` armazena na variável `NL` o valor de `N` arredondado pelo procedimento `arredonda/2`. Resta então separar os `NL` pesos mais "fortes" da lista de pesos. Vale lembrar que a lista pesos passada como segundo argumento do procedimento `seleciona_neuronios/3` já teve seus elementos previamente separados pelo procedimento `pesos_camada/5` e ordenados pelo procedimento `sortw/3`. A ordenação ocorreu em ordem decrescente, ou seja, os pesos mais "pesados" se encontram

no início da lista e os menos "pesados"no fim da lista. Basta portanto que sejam selecionados os primeiros `NL` elementos da lista de pesos ordenados, esta tarefa é realizada pelo procedimento `retira_primeiros/3`.

No exemplo desta seção, os pesos da primeira consulta foram ordenados pelo procedimento `sortw/3` e se encontram na seguinte ordem

```
[w(1,3,1,1), w(1,1,1,-0.9), w(1,1,2,0.5), w(1,3,2,0.2)]
```

Na primeira consulta o grau de explicação utilizado é de 0.5, isto significa que 50% dos pesos mais "fortes" serão selecionados para a próxima iteração, desta forma os pesos selecionados são

```
[w(1,3,1,1), w(1,1,1,-0.9)]
```

Para o exemplo da segunda consulta, os pesos ordenados são

```
[w(1,3,1,1), w(1,2,1,-0.5), w(1,1,1,-0.9)]
```

Nesta consulta, o grau de explicação escolhido foi de 0.1, desta forma deve ser selecionado 0.3 peso, normalmente este valor seria arredondado para 0, mas o procedimento `arredonda/2` foi implementado de forma a retornar apenas valores maiores ou iguais a 1. Portanto, será selecionado apenas um peso mais significante da lista de pesos ordenados. A lista resposta do procedimento `seleciona_neuronios/3` para a segunda consulta é

```
[w(1,3,1,1)]
```

O próximo procedimento ativado por `explica/7` é novamente o procedimento `trace/3` chamado com o parâmetro `pesos`. O átomo `pesos` indica que o procedimento `trace/3` deve imprimir a camada e os pesos que foram selecionados acima do índice de corte pelo procedimento `seleciona_neuronios/3`.

Após selecionar os pesos mais "fortes" o procedimento `explica/7` deve analisar estes pesos e extrair os neurônios a que estes pesos estão ligados na camada seguinte. Novamente as explicações *Como?* e *Porque?* devem ser tratadas de forma diferente. As duas consultas podem ajudar a visualizar melhor as diferenças entre as duas explicações. Na primeira consulta, a explicação *Como?* inicia analisando na camada de entrada todos os pesos ligados aos neurônios 1 e 3 —neurônios que o usuário deseja que sejam explicados. Após separar os pesos ligados aos neurônios 1 e 3, o método de explicação, através do procedimento `seleciona_neuronios/3` separa os 50% pesos mais "fortes". Neste exemplo eles são

```
[w(1,3,1,1), w(1,1,1,-0.9)]
```

Deve ser observado que o primeiro peso liga o neurônio 3 da primeira camada ao neurônio 1 da segunda camada. O segundo peso liga o neurônio 1 da primeira ao neurônio 1 da segunda camada. Desta forma, os dois pesos estão ligados ao neurônio 1 da segunda camada e, este índice de neurônio será utilizado na próxima recursão do método de explicação.

Para a explicação *Porque?* o processo é inverso. O programa começa analisando os pesos da camada de saída ligados ao neurônio 1 e, após serem ordenados e selecionados apenas o peso

```
[w(1,3,1,1)]
```

foi escolhido como o peso mais forte. Este peso liga o neurônio 1 da camada de saída ao neurônio 3 da camada seguinte. Portanto deve-se extrair, do peso selecionado, o índice de neurônio 3 para ser utilizado na próxima recursão.

O procedimento que analisa uma lista de pesos e extrai a quais neurônios estes pesos estão ligados é o procedimento `indice_neuronios/3`.

Segue a listagem do procedimento `indice_neuronios/3`.

Procedimento 4.14 `indice_neuronios/3`

```
indice_neuronios(CP, Neuronios_Selecionados, Indice_Neuronios):-
     indice_neuronios(CP, Neuronios_Selecionados, [],
           Indice_Neuronios_Repetidos),
     retira_repetidos(Indice_Neuronios_Repetidos, Indice_Neuronios).
```

Os argumentos de `indice_neuronios/3` são

$[+] < arg_1 >$: Tipo de explicação *Como?* ou *Porque?*

$[+] < arg_2 >$: Lista de pesos selecionados acima do índice de corte

$[-] < arg_3 >$: Lista dos índices dos neurônios selecionados

O procedimento `indice_neuronios/3` recebe como primeiro argumento um átomo que identifica qual tipo de explicação está em uso. O átomo `como` identifica uma explicação do tipo *Como?* e um átomo `porque` identifica uma explicação do tipo *Porque?*. O procedimento `indice_neuronios/3` utiliza um procedimento auxiliar chamado `indice_neuronios/4`, o qual realmente é responsável por extrair os neurônios a que cada peso está ligado. Em seguida, o procedimento `indice_neuronios/3` chama o procedimento `retira_repetidos/2` cuja função é retirar índices de neurônios que estejam repetidos. Um índice de neurônio pode estar contido na lista por mais de uma vez, caso este neurônio esteja ligado a dois ou mais pesos selecionados entre os mais "fortes". Neste caso o procedimento `retira_repetidos/2` recebe como entrada uma lista com números repetidos e como saída retorna uma lista sem número repetidos.

Segue a listagem do procedimento `indice_neuronios/4`

Procedimento 4.15 `indice_neuronios/4`

```
indice_neuronios(_, [], Indice_Neuronios, Indice_Neuronios).
indice_neuronios(como, [w(_, _, Indice, _)|C], C1, Indice_Neuronios):-
     indice_neuronios(como, C, [Indice|C1], Indice_Neuronios).
indice_neuronios(porque, [w(_, Indice, _, _)|C], C1, Indice_Neuronios):-
     indice_neuronios(porque, C, [Indice|C1], Indice_Neuronios).
```

Os argumentos de `indice_neuronios/4` são

[+] $< arg_1 >$: Tipo de explicação *Como?* ou *Porque?*

[+] $< arg_2 >$: Lista de pesos selecionados acima do índice de corte

[+] $< arg_3 >$: Lista dos índices dos neurônios selecionados encontrados até o momento

[−] $< arg_4 >$: Lista dos índices dos neurônios selecionados

O primeiro predicado do procedimento `indice_neuronios/4` é responsável pela parada de recursão. Este predicado é executado com sucesso quando o segundo argumento — lista de pesos acima do índice de corte — é uma lista vazia. Neste caso não existem mais pesos a serem processados e o predicado instancia o quarto argumento — argumento de saída — o valor do terceiro argumento — lista dos índices de neurônios selecionados encontrados até o momento.

O segundo e terceiro predicados são responsáveis por extrair índices dos neurônios para as explicações *Como?* e *Porque?* repectivamente. Estes dois predicados são bastante semelhantes. No primero predicado, cada peso é analisado e é retirado o terceiro argumento do funtor `como`. No segundo predicado, são retirados os índices de neurônios localizados no segundo argumento do funtor `porque`.

O próximo passo tomado pelo procedimento `explica/7` é chamar novamente o procedimento `trace/3`, passando como primeiro argumento o átomo `neuronios`. Nesta ocasicao, serão impressas a camada atualmente sendo analisada e os índices dos neurônios selecionados pelo procedimento `indice_neuronios/4`.

Após a execução do procedimento `trace/3` é calculado o valor da próxima camada da Rede Neural a ser analisada. Como já foi dito no início desta seção, o segundo predicado do procedimento `explica/7` é responsável por explicações do tipo *Como?* e o terceiro predicado é responsvel por explicações do tipo *Porque?*. Para explicações do tipo *Como?*, o valor da variável `proxima_camada` será o valor da camada atual incrementado de 1. Enquanto que para explicações do tipo *Porque?* o valor da variável `proxima_camada` será o valor da camada atual decrementado 1. O valor da próxima camada é calculado e instanciado com a variável `Proxima_Camada`.

Por fim, o procedimento `explica/7` é chamado recursivamente. Nesta nova chamada são alterados os valores dos argumentos do funtor `como` — para explicações do tipo

Como? — ou do funtor `porque` — para explicações do tipo *Porque?*. Como primeiro parâmetro destes funtores é passado o valor da nova camada da Rede Neural a ser analisada, este valor está contido na variável `Proxima_Camada`. Como segundo parâmetro é passada a variável `Indice_Neuronios` que está instanciada como a lista de índices dos neurônios mais "fortes" retornada pelo procedimento `indice_neuronios/3`.

Como dito anteriormente, as duas últimas cláusulas do procedimento `explica/7`, referentes ao critério de seleção de pesos "soma", serão discutidas somente agora, uma vez que estas cláusulas possuem alguns procedimentos adicionais. O funcionamento geral do critério de seleção de pesos "soma" resume-se a agrupar todos os pesos que chegam — explicações *Como?* — ou partem — explicações *Porque?* — de um determinado neurônio e somar o valor destes pesos. Os neurônios com os maiores valores de seus pesos somados são selecionados para a próxima recursão do algoritmo. Os procedimentos responsáveis por dividir e somar os pesos são chamados `divide/3` e `soma/3`, respectivamente. Segue a listagem das duas últimas cláusulas do procedimento `explica/7`.

Procedimento 4.16 `explica/7 - Criterio soma`

```
explica(Criterio, Trace, Grau, como(Camada, Neuronios), RN,
          Camada_Parada, Expl):-
 criterio_ordenacao(Criterio, Cri_Ord),
 pesos_camada(como, Camada, Neuronios, RN, Pesos_Camada),
 divide(como, Pesos_Camada, Pesos_Camada_Divididos),
 soma(como, Pesos_Camada_Divididos, Pesos_Somados),
 sortw(Cri_Ord, Pesos_Somados, Pesos_Ordenados_Camada),
 trace(prox, Trace, Camada, Pesos_Ordenados_Camada),
 seleciona_neuronios(Grau, Pesos_Ordenados_Camada,
          Neuronios_Selecionados),
 trace(pesos, Trace, Camada, Neuronios_Selecionados),
 indice_neuronios(como, Neuronios_Selecionados, Indice_Neuronios),
 trace(neuronios, Trace, Camada, Indice_Neuronios),
 Proxima_Camada is Camada + 1,
 explica(Criterio, Trace, Grau, como(Proxima_Camada,
          Indice_Neuronios), RN, Camada_Parada, Expl).
explica(Criterio, Trace, Grau, porque(Camada, Neuronios), RN,
          Camada_Parada, Expl):-
 criterio_ordenacao(Criterio, Cri_Ord),
 pesos_camada(porque, Camada, Neuronios, RN, Pesos_Camada),
 divide(porque, Pesos_Camada, Pesos_Camada_Divididos),
 soma(porque, Pesos_Camada_Divididos, Pesos_Somados),
 sortw(Cri_Ord, Pesos_Somados, Pesos_Ordenados_Camada),
 trace(prox, Trace, Camada, Pesos_Ordenados_Camada),
 seleciona_neuronios(Grau, Pesos_Ordenados_Camada,
          Neuronios_Selecionados),
 trace(pesos, Trace, Camada, Neuronios_Selecionados),
 indice_neuronios(porque, Neuronios_Selecionados, Indice_Neuronios),
 trace(neuronios, Trace, Camada, Indice_Neuronios),
 Proxima_Camada is Camada - 1,
 explica(Criterio, Trace, Grau, porque(Proxima_Camada,
          Indice_Neuronios), RN, Camada_Parada, Expl).
```

Vale relembrar que os argumentos de `explica/7` são

$[+] < arg_1 >$: Critério de seleção de pesos

$[+] < arg_2 >$: Liga ou desliga o mecanismo de trace

$[+] < arg_3 >$: Grau de explicação ($0 \leq$ `Grau` ≤ 1)

$[+] < arg_4 >$: Tipo de explicação *Como?* ou *Porque?*

$[+] < arg_5 >$: Lista dos pesos da Rede Neural

$[+] < arg_6 >$: Camada de Parada

$[-] < arg_7 >$: Neurônios de resposta

O procedimento `divide/3` é responsável por agrupar, por neurônios, os pesos da camada sendo analisada. Segue a listagem deste procedimento

Procedimento 4.17 `divide/3`

```
divide(_, [], []).
divide(como, [w(C,N1,N2,P)|Cauda], [L1|L]):-!,
  divide2(como, N2, [w(C,N1,N2,P)|Cauda], L1, Sem_Repetidos),
  divide(como, Sem_Repetidos, L).
divide(porque, [w(C,N1,N2,P)|Cauda], [L1|L]):-
  divide2(porque, N1, [w(C,N1,N2,P)|Cauda], L1, Sem_Repetidos),
  divide(porque, Sem_Repetidos, L).
```

Os argumentos de `divide/3` são

$[+] < arg_1 >$: Tipo de explicação, átomos válidos: `como` ou `porque`

$[+] < arg_2 >$: Lista de pesos da Rede Neural

$[-] < arg_3 >$: Lista com os pesos divididos por neurônio

O primeiro predicado do procedimento `divide/3` é responsável pela condição de parada, este determina que quando a lista de pesos da Rede Neural — segundo argumento — estiver vazia então o predicado deve retornar uma lista vazia no terceiro argumento — argumento de saída. O segundo e terceiro predicados são responsáveis pelas explicações *Como?* e *Porque?*, respectivamente. Estes predicados simplesmente retiram o valor do neurônio destino — para explicações do tipo *Como?* — ou fonte — para explicações do tipo *Porque?* — e passam este valor como segundo parâmetro ao procedimento `divide2/5`.

Suponha que a seguinte lista de pesos foi passada como parâmetro ao procedimento `divide/3`

```
[w(1,1,1,0.3), w(1,1,2,-0.2), w(1,1,3,0.7),
 w(1,2,1,0.1), w(1,2,2,-0.7), w(1,2,3,0.2)]
```

Para uma consulta do tipo *Como?*, o procedimento `divide/3` irá retornar a seguinte lista como resposta

```
[[w(1,1,1,0.3), w(1,2,1,0.1)], [w(1,1,2,-0.2),
 w(1,2,2,-0.7)], [w(1,1,3,0.7), w(1,2,3,0.2)]
```

Como já foi dito anteriormente, para explicações do tipo *Como?*, o procedimento `divide/3` divide a lista de pesos da Rede Neural em sublistas, na qual cada sublista contém todos os pesos que chegam a um mesmo neurônio.

Suponha, agora, que a mesma lista fosse passada como parâmetro ao procedimento `divide/3` para uma explicação do tipo *Porque?*, a resposta seria a seguinte lista

```
[[w(1,1,1,0.3), w(1,1,2,-0.2), w(1,1,3,0.7)],
 [w(1,2,1,0.1), w(1,2,2,-0.7), w(1,2,3,0.2)]]
```

Para explicações do tipo *Porque?* o procedimento `divide/3` separa os pesos em sublistas, na qual cada sublista possui os pesos que partem de um mesmo neurônio.

Para separar os pesos corretamente, o procedimento `divide/3` utiliza o procedimento auxiliar `divide2/5`, segue a listagem deste procedimento

Procedimento 4.18 `divide2/5`

```
divide2(_, _, [], [], []).
divide2(como, N, [w(C,N1,N,P)|Cauda], [w(C,N1,N,P)|L], L2):-!,
 divide2(como, N, Cauda, L, L2).
divide2(porque, N, [w(C,N,N2,P)|Cauda], [w(C,N,N2,P)|L], L2):-!,
 divide2(porque, N, Cauda, L, L2).
divide2(CP, N, [Cabeca|Cauda], L, [Cabeca|L2]):-
 divide2(CP, N, Cauda, L, L2).
```

Os argumentos de `divide2/5` são

[+] < arg_1 >: Tipo de explicação, átomos válidos: `como` ou `porque`

[+] < arg_2 >: Índice do neurônio

[+] < arg_3 >: Lista de pesos da Rede Neural

[−] < arg_4 >: Lista de pesos relacionados com o neuônio especificado no segundo argumento

[−] < arg_5 >: Lista de pesos que não possuem relação com o neurônio do segundo argumento

O procedimento `divide2/5` é constituído de quatro predicados, o primeiro é responsável pela condição de parada. O segundo e terceiro predicados são responsáveis pelas explicações *Como?* e *Porque?*, respectivamente. Estes predicados comparam o segundo argumento com o índice do neurônio destino — explicações *Como?* — ou fonte — explicações *Porque?* — do peso que está na cabeça da lista no terceiro argumento. Caso estes valores sejam iguais, o peso é inserido na lista de saída do quarto argumento. O quarto predicado é executado quando todos os anteriores falharem, ele é responsável por inserir o peso que está na cabeça da lista do terceiro argumento na lista de resposta do quinto argumento. Desta forma, aqueles pesos que possuem índice de neurônio igual ao segundo argumento são inseridos na lista do quarto argumento, todos os outros que não possuem o mesmo valor são inseridos na lista do quinto argumento.

Logo após separar os pesos em sublistas o procedimento `explica/7` chama o procedimento `soma/3`, o qual é responsável por somar os valores de todos os pesos em cada sublista. No exemplo anterior, uma lista de pesos foi dividida pelo procedimento `divide/3`, para a explicação *Como?*, esta lista é

```
[[w(1,1,1,0.3), w(1,2,1,0.1)], [w(1,1,2,-0.2), w(1,2,2,-0.7)],
                    [w(1,1,3,0.7), w(1,2,3,0.2)]
```

Após ser dividida, esta lista pode ser passada como parâmetro ao procedimento `soma/3`, criando a seguinte lista como resultado

```
[w(1,0,1,0.4), w(1,0,2,-0.9), w(1,0,3,0.9)]
```

Esta lista não segue a notação de pesos utilizada até agora, pois os funtores `w` não mais representam pesos da Rede Neural, mas sim os valores dos pesos somados para cada neurônio. Para uma explicação *Como?* os argumentos do funtor `w` representa

```
w(Camada, Ignorado, Neuronio, Soma)
```

O primeiro valor o funtor `w` indica a camada que está sendo analisada. O segundo argumento é ignorado, nesta posição é armazenado o valor 0. O terceiro argumento indica o neurônio destino das conexões e, por fim, o quarto argumento indica a soma de todas as conexões que estão chegando a este neurônio.

Nas explicações do tipo *Porque?* o funtor `w` também é utilizado para representar a soma dos pesos que partem de um certo neurônio, entretanto a ordem do segundo e terceiro argumentos é invertida

```
w(Camada, Neuronio, Ignorado, Soma)
```

Segue a listagem do procedimento `soma/3`

Procedimento 4.19 `soma/3`

```
soma(CP, [], []).
soma(CP, [Cabeca|Cauda], [Peso|L]):-
 soma2(CP, Cabeca, 0, Peso),
 soma(CP, Cauda, L).
```

Os argumentos de `soma/3` são

$[+] < arg_1 >$: Tipo de explicação, átomos válidos: `como` ou `porque`

$[+] < arg_2 >$: Lista de pesos da Rede Neural divididos por neurônio

$[-] < arg_3 >$: Lista de funtores `w` representando a soma das conexões

O procedimento `soma/3` é muito simples, pois sua função se resume a retirar uma a uma as sublistas contidas no segundo argumento e passá-las ao procedimento auxiliar `soma2/4`. Segue a listagem do procedimento `soma2/4`

Procedimento 4.20 `soma2/4`

```
soma2(como, [w(C,N1,N2,P)], Soma, w(C,0,N2,Soma2)):-
 Soma2 is Soma + P.
soma2(porque, [w(C,N1,N2,P)], Soma, w(C,N1,0,Soma2)):-
 Soma2 is Soma + P.
soma2(CP, [w(C,N1,N2,P)|Cauda], Soma, Peso):-
 Soma2 is Soma + P,
 soma2(CP, Cauda, Soma2, Peso).
```

Os argumentos de `soma2/4` são

$[+] < arg_1 >$: Tipo de explicação, átomos válidos: `como` ou `porque`

$[+] < arg_2 >$: Lista de pesos da Rede Neural divididos por neurônio

$[+] < arg_3 >$: Variável auxiliar utilizada para armazenar o valor da soma

$[-] < arg_4 >$: Funtor `w` representando a soma das conexões

O procedimento `soma2/4` possui três predicados, o primeiro e segundo predicados são responsáveis pela condição de parada de recursão para as explicações *Como?* e *Porque?* respectivamente. O terceiro predicado retira cada peso da cabeça da lista de pesos divididos por neurônio — segundo argumento — e soma os valores dos pesos, passando este valor para a próxima recursão através do terceiro argumento.

Os funtores `w`, os quais representam os valores dos pesos somados por neurônio, serão analisados pelos procedimentos subseqüentes, da mesma forma como ocorre para as explicações `pau` e `ccg`, pois tais procedimentos são comuns a todos os critérios de seleção de pesos. Vale relembrar que após divididos e somados, os funtores `w` serão ordenados pelo procedimento `sortw/3`, os conjuntos de pesos mais "fortes" serão selecionados pelo procedimento `seleciona_neuronios/3` e, por fim, os funtores serão convertidos para índices de neurônios pelo procedimento `indice_neuronios/3`. Este processo continuará ocorrendo repetidas vezes até que a camada de parada seja atingida e o procedimento encerrado com sucesso, da mesma forma que ocorre com as outras variações do método EN.

Na próxima seção, são apresentados alguns exemplos de execução das variações propostas do método EN.

5 Exemplos de Execução

Nesta seção são apresentados três exemplos de execução da implementação realizada do método EN segundo os diferentes critérios de seleção dos pesos descritos nas seções anteriores, utilizando uma Rede Neural muito simples apresentada na Figura 3.

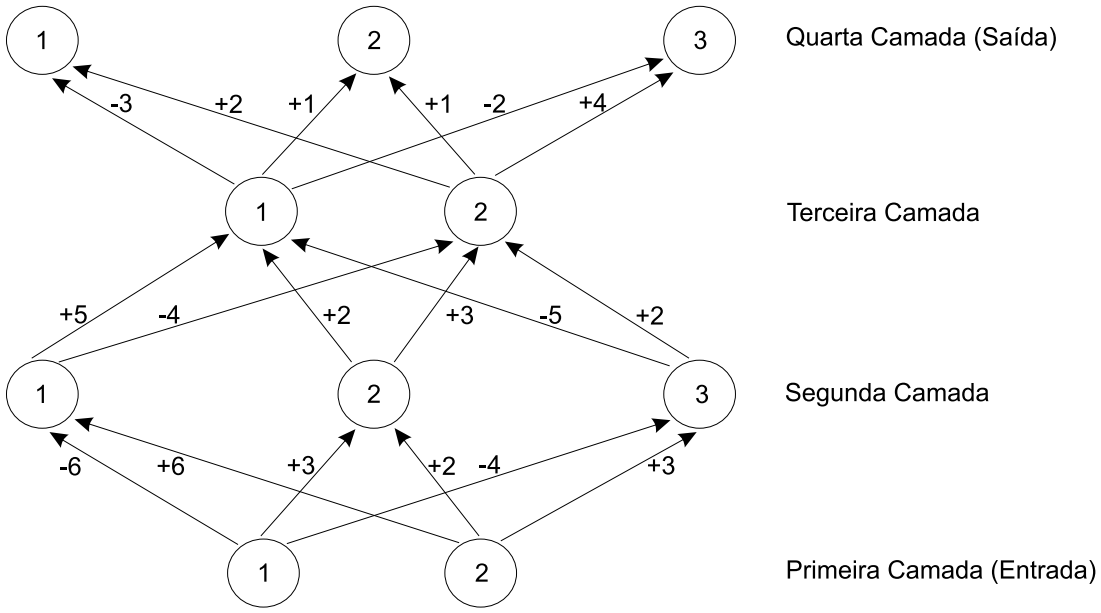

Figura 3: Rede Neural Exemplo

Esta Rede Neural foi pré-processada e transformada em uma lista de pesos correspondente a estrutura de dados descrita na seção 4. O procedimento `redeneural/1` é responsável por instanciar uma variável livre com a estrutura de dados que será utilizada pelo procedimento `explica/6`, responsável pelo mecanismo de explicação.

No primeiro exemplo é mostrado o mecanismo de explicação EN utilizando o critério de seleção de pesos "soma", no qual os pesos são agrupados e somados. Foi escolhido a explicação *Como?*, desta forma o programa deverá relacionar os neurônios da entrada da rede com os neurônios da saída. O mecanismo de *Trace* foi acionado para que o programa informe ao usuário como está ocorrendo o processo de explicação. Foi escolhido um grau de explicação de 0.5 (50%). Os neurônios escolhidos pelo usuário para serem explicados são os neurônios 1 e 2 da primeira camada.

```
?- redeneural(X), explica(soma, sim, 0.5, como(1, [1,2]), X, Expl).

Pesos da Camada: 1
Pesos: [w(1,0,2,5), w(1,0,1,0), w(1,0,3,-1)]

Pesos Selecionados na Camada: 1
Pesos: [w(1,0,2,5), w(1,0,1,0)]

Neuronios Selecionados na Camada: 1
Neuronios: [1,2]

Pesos da Camada: 2
Pesos: [w(2,0,1,7), w(2,0,2,-1)]
```

```
Pesos Selecionados na Camada: 2
Pesos: [w(2,0,1,7)]

Neuronios Selecionados na Camada: 2
Neuronios: [1]

Pesos da Camada: 3
Pesos: [w(3,0,2,1), w(3,0,3,-2), w(3,0,1,-3)]

Pesos Selecionados na Camada: 3
Pesos: [w(3,0,2,1), w(3,0,3,-2)]

Neuronios Selecionados na Camda: 3
Neuronios: [3,2]

Como: [3,2]
```

Como pode ser observado através do mecanismo de *Trace*, o neurônio 3 da segunda camada é desconsiderado pois este neurônio possui uma soma de pesos menor que a soma dos pesos dos outros neurônios da segunda camada. Se fosse empregado o critério original de seleção de pesos ("pau") com o mesmo grau de explicação (0.5), este neurônio seria escolhido pois possui valores absolutos dos pesos maiores do que o neurônio 2 da segunda camada.

A seguir é mostrado o segundo exemplo, onde a mesma explicação *Como?* é requisitada ao programa, mas utilizando o critério de seleção de pesos "pau", ao invés do critério "soma", e sem informações de *Trace*.

```
?- redeneural(X), explica(pau, nao, 0.5, como(1, [1,2]), X, Expl).

Como: [3,1]
```

Por fim, no terceiro exemplo é apresentado o mecanismo de explicação com o critério de seleção de pesos "ccg", o qual ordena os pesos por valores não absolutos. O mesmo grau de explicação dos outros exemplos é utilizado.

```
?- redeneural(X), explica(ccg, nao, 0.5, como(1, [1,2], X, Expl).

Como: [2,1,3]
```

Como pode ser observado as três execuções apresentam resultados diferentes. A primeira execução é considerada por nós a mais robusta, uma vez que o critério empregado nesta execução ("soma") analisa a soma das conexões que chegam a um determinado neurônio. Os outros critérios ("pau" e "ccg") analisam uma Rede Neural considerando que as suas conexões são independentes dos neurônios a que estão ligadas.

6 Conclusão

Neste trabalho foi apresentado o método EN para extração de conhecimento de Redes Neurais, assim como algumas modificações realizadas neste método no processo de seleção dos pesos. O processo de seleção do método EN original consiste em ordenar os pesos por valores absolutos e em seguida selecionar uma fração destes pesos. Na primeira alteração implementada, este critério é modificado para ordenar os pesos por seus valores e, na segunda alteração, os pesos que chegam em cada neurônio são agrupados, somados e ordenados por valor para então serem selecionados os neurônios que farão parte da próxima iteração.

Intuitivamente, o critério de seleção de pesos `soma` é mais robusto, uma vez que considera os pesos positivos como incentivadores e pesos negativos como inibidores do ativamento neural. Através da soma pesos, positivos e negativos, em um neurônio procura-se descobrir se os pesos incentivadores são suficientemente "fortes" para compensar os pesos negativos e ativar o neurônio. Entretanto, estas constatações intuitivas ainda carecem de comprovações empíricas, desta forma como trabalhos futuros pretende-se aplicar esta ferramenta a uma série de Redes Neurais treinadas por conjuntos de dados bem conhecidos, a fim de comparar os diversos critérios implementados para selecionar os neurônios mais significativos, com vistas a procurar por melhores critérios.

Agradecimentos: Ao Prof. André Ponce de Leon Ferreira de Carvalho pelas freqüentes sugestões para a melhoria e desenvolvimento deste trabalho.

Referências

[1] Andrews, R.; Diederich, J.; Tickle, A. B.; *A Survey and Critique of Techniques for Extracting Rules from Trained Artificial Neural Networks.* Neurocomputing Research Centre, Queensland University of Technology, Queensland, Australia, 1995.

[2] Bochereau, L.; Bourgine, P.; *Implèmentation et Extration de Traits Sèmantiques sur un Rèseau Neuro-Mimètique: Exemple de la Première Annonce au Bridge.* In Neuro'Nimes 89, Nimes, France, pp. 125-141, November, 1989.

[3] Carpenter, G. A.; Tan, A.; *Rule Extration: from Neural Architecture to Symbolic Representation.* Connection Science, vol. 7, n. 1, pp. 3-27, 1995.

[4] Fu, L.; *Knowledge-Based Connectionism for Revising Domain Theories.* IEEE Transactions on Systems, Man, and Cybernetics, vol. 23, n. 1, pp. 173-182, January/February, 1993.

[5] Fu, L.; *Rule Generation from Neural Networks.* IEEE Transactions on Systems, Man, and Cybernetics, vol. 24, n. 8, pp. 1114-1124, August, 1994.

[6] Gallant, S. I.; *Connectionist Expert Systems.* Communications of the ACM, vol. 31, n.2, pp. 152-169, February, 1988.

[7] Glorennec, P. Y.; *Un Reseau "Neuro-Flou" Evolutif.* Neuro'Nimes 91, pp. 301-314, November, 1991.

[8] Gorman, R. P.; Sejnowski, T. J.; *Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets.* Neural Networks, vol. 1, pp. 75-89, 1988.

[9] Halgamuge, S. K.; Glesner, M.; *A Fuzzy-Neural Approach for Pattern Classification with the Generation of Rules Based on Supervised Learning.* Neuro'Nimes 92, pp. 165-173, November, 1992.

[10] Klimasauskas, C. C.; *Neural Nets Tell Why.* Dr. Dobb's Journal, pp. 16-24, April, 1991.

[11] McMillan, C.; Mozer, M. I.; Smolensky, P.; *The Connectionist Scientist Game: Rule Extraction and Refinement in a Neural Network.* Technical Report, Departament of Computer Science and Institute of Cognitive Science, University of Colorado, 1991.

[12] Pau, L. F.; Götzche, T.; *Explanation Facility for Neural Networks.* Journal of Intelligent and Robotic Systems, vol. 5, pp. 193-206, 1992.

[13] Rumelhart, D. E.; Hilton, G. E.; Williams, R. J.; *Learning Representations by Back-Propagation Errors.* Nature, vol. 323, pp. 533-536, October, 1986.

[14] Saito, K.; Nakano, R.; *Rule Extraction from Facts and Neural Networks.* In INNC-90 Paris: Proceedings of the International Neural Network Conference, vol. 1, pp. 379-382, July, 1990.