Relatório do Trabalho prático de Inteligência Artificial

Vinícius P. Medeiros, Odalisio L. da S. Neto , Luiza Paula M. Leão, Rodrigo G. Printes, Alexandre M. Uchôa, Guilherme Matheus de A. Lima, José Marcos C. Neto, Salomão C. Cruz, Kennedy Coelho Nolasco, André Luiz Miranda Brandão, Victor G. Cavalcante, Daniel Santiago da Silva

[1] Instituto de Computação - Universidade Federal do Amazonas (UFAM)
Caixa Postal - 69080-900 - Manaus - AM - Brasil

{vpms,oldsn,lpml,rgp,alexandre.uchoa,gmdal,jmcn,scc,kcn,andre.brandao,
dsds2}@icomp.ufam.edu.br

1. Introdução

Os sistemas neuro-simbólicos são a tentativa de programar computadores de forma que eles tenham um funcionamento parecido com o cérebro humano. Eles utilizam modelos matemáticos e algoritmos de aprendizado de máquina para processar as informações e tomar decisões parecidas com o cérebro humano.

Sistemas neuro-simbólicos são modelos computacionais que integram características de redes neurais e lógica simbólica para representar e processar conhecimento de forma mais expressiva e interpretável. Esses sistemas buscam combinar a capacidade de aprendizado e generalização das redes neurais com a capacidade de representação e manipulação simbólica da lógica. Isso permite que sejam aplicados em uma ampla variedade de tarefas, desde o processamento de linguagem natural até a tomada de decisões em sistemas de controle e robótica.

Extração de conhecimento de redes neurais é o processo de extrair informações úteis e interpretações de uma rede neural treinada. Embora as redes neurais possam ser poderosas em termos de desempenho em muitas aplicações, elas são frequentemente consideradas caixas pretas, pois seus modelos não fornecem uma explicação clara dos mecanismos de raciocínio subjacentes. A extração de conhecimento de redes neurais é uma abordagem para superar essa limitação, permitindo que os usuários compreendam melhor o processo de tomada de decisão da rede neural e usem essa compreensão para aprimorar ainda mais o modelo.

Este trabalho será dividido em três problemas relacionados: **O primeiro**, será implementar uma solução para um problema lógico, utilizando meta-redes para conectar os conceitos das premissas. **O segundo** será estender o modelo desenvolvido pelo primeiro problema e treiná-lo para outros casos. **O terceiro,** será extrair da rede treinada no segundo problema regras que descrevem quais características de um trem que vai para o leste e compará-las com o resultado da rede LTN.

2. Referencial Teórico

2.1. Sistemas Neuro Simbólicos

Os sistemas neuro-simbólicos de aprendizado são modelos de aprendizado de máquina que integram elementos de processamento simbólico e processamento neural. Esses sistemas combinam as vantagens do processamento simbólico, que é capaz de lidar com informação estruturada e interpretável, com as vantagens do processamento neural, que é capaz de lidar com dados brutos e complexos.

Os sistemas neuro-simbólicos de aprendizado são modelos de inteligência artificial que combinam a capacidade de aprendizado das redes neurais artificiais com a capacidade de raciocínio simbólico dos sistemas baseados em lógica. Essa combinação de diferentes abordagens de aprendizado de máquina tem como objetivo superar as limitações de cada uma das abordagens individualmente e aproveitar suas vantagens complementares.

Esses sistemas são usados em diversas áreas, como processamento de linguagem natural, visão computacional, robótica e diagnóstico médico. Eles são muito úteis em tarefas que envolvem a interpretação e o raciocínio sobre informações complexas, que requerem a combinação de dados brutos com conhecimento prévio e regras lógicas.

2.1.2. Ciclo de Aprendizado Neuro-Simbólicos

O ciclo de aprendizagem neuro-simbólico é o processo pelo qual os sistemas neuro-simbólicos adquirem e integram novos conhecimentos e habilidades. Ele é composto de quatro etapas principais:

(i) **Aquisição de dados:** A primeira etapa é a aquisição de dados, onde os sistemas neuro-simbólicos coletam informações relevantes a partir de exemplos, observações ou outras fontes.

(ii) **Análise de dados:** Na segunda etapa, os dados coletados são analisados e processados para identificar padrões e relações relevantes.

(iii) **Integração de conhecimento:** Na terceira etapa, o conhecimento adquirido é integrado com o conhecimento existente, o que pode resultar na atualização ou na adição de novas regras, relações ou conceitos.

(iv) **Validação e verificação:** Por fim, o conhecimento integrado é validado e verificado a fim de garantir sua precisão e consistência.

Esse ciclo de aprendizagem neuro-simbólico é iterativo e permite que os sistemas neuro-simbólicos aprendam e evoluam ao longo do tempo. À medida que os sistemas adquirem mais conhecimento, eles se tornam mais precisos e capazes de tomar decisões informadas em situações complexas.

Em resumo, o ciclo de aprendizagem neuro-simbólico é uma combinação de processos de aprendizado de máquina e conhecimento explícito que permite que os sistemas neuro-simbólicos adquiram e integrem novos conhecimentos e habilidades.

2.2 O sistema C-ILP

C-ILP é um modelo computacional altamente paralelo baseado em uma rede neural artificial de feedforward. Ele é uma abordagem neuro-simbólica desenvolvida por *D'avila Garcez (1999)* e integra a aprendizagem indutiva a partir de exemplos e background knowledge com a aprendizagem dedutiva usando a programação lógica. Nesta abordagem um Algoritmo de Tradução mapeia um programa lógico geral $\mathscr{P}$ em uma rede neural de camada oculta única $\mathscr{N}$, de modo que $\mathscr{N}$ calcula o ponto fixo mínimo de $\mathscr{P}$. Além disso, $\mathscr{N}$ pode ser treinada por exemplos usando backpropagation, tendo $\mathscr{P}$ como background knowledge. O conhecimento adquirido pelo treinamento pode então ser extraído, fechando o ciclo de aprendizagem de um sistema neuro-simbólico.

No Algoritmo de Tradução C-ILP, cada cláusula (r_l) de $\mathscr{P}$ é mapeada da camada de entrada para a camada de saída de $\mathscr{N}$ por meio de um neurônio (N_l) na única camada oculta de $\mathscr{N}$. De maneira intuitiva, o Algoritmo de Tradução de $\mathscr{P}$ para $\mathscr{N}$ precisa implementar as seguintes condições: (C1) O potencial de entrada de um neurônio oculto (N_l) só pode exceder o limiar de N_l (θ_l), ativando N_l, quando todos os antecedentes positivos de r_l são atribuídos o valor verdadeiro, enquanto todos os antecedentes negativos de r_l são atribuídos o valor falso; e, (C2) O potencial de entrada de um neurônio de saída (A) só pode exceder o limiar de A (θ_A), ativando A, quando pelo menos um neurônio oculto N_l conectado a A está ativado.

Uma cláusula geral é uma regra da forma $L_1, \ldots, L_k \rightarrow A$, onde A é um átomo e L_i ($1 \leq i \leq k$) é um literal (um átomo ou a negação de um átomo). Um programa lógico geral é um conjunto finito de cláusulas gerais.

Considere como exemplo o seguinte programa lógico $\mathscr{P} = \{B; B \wedge C \wedge \sim D \rightarrow A; E \wedge F \rightarrow A\}$.

O Algoritmo de Tradução mapeia o programa acima para uma rede $\mathscr{N}$, essa rede pode ser vista na figura abaixo:

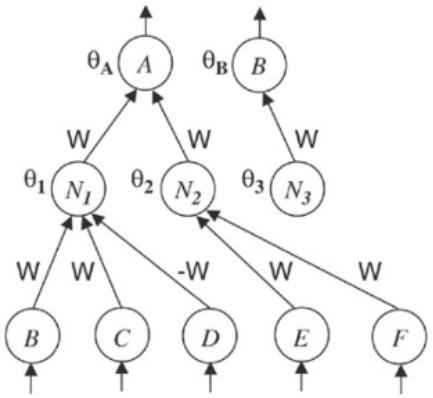

Figura 1: Rede $\mathscr{N}$ resultante do Algoritmo de Tradução

Na rede $\mathscr{N}$, os pesos (W) e os limiares (θ) estão dispostos de tal maneira que ambas as condições (C1) e (C2) sejam satisfeitas. Repare que a rede N da figura acima não está totalmente conectada, para fazer com que ela fique totalmente conectada, todas as ligações que não estão na figura devem receber um peso inicial de 0.

Na figura 1, cada entrada e saída da rede $\mathscr{N}$ estão associadas com um átomo do programa lógico $\mathscr{P}$, note também que cada neurônio N_l da camada oculta corresponde à uma cláusula r_l de $\mathscr{P}$. Para calcular uma semântica de ponto fixo de $\mathscr{P}$, o neurônio de saída B deve alimentar o neurônio de entrada B, de modo que $\mathscr{N}$ seja usado para iterar $T_{\mathscr{P}}$, o operador de ponto fixo de $\mathscr{P}$. $\mathscr{N}$ eventualmente convergirá para um estado estável que é idêntico ao modelo estável de $\mathscr{P}$.

Os passos para realizar o Algoritmo de Tradução C-ILP podem ser vistos a seguir:

(1) Calcular o valor de W de tal forma que o seguinte seja satisfeito:

$$W \geq \frac{2}{\beta} \cdot \frac{ln(1+A_{min}) - ln(1-A_{min})}{MAX_{\mathscr{P}}(\vec{k},\vec{\mu})(A_{min}-1)+A_{min}+1}$$

(2) Para cada cláusula r_l de $\mathscr{P}$ na forma $L_1, \ldots, L_k \rightarrow A (k \geq 0)$:

(a) Crie neurônios de entrada $L_1, \ldots, L_k$ e o neurônio de saída A em $\mathscr{N}$ (se ainda não existirem);

(b) Adicione um neurônio N_l à camada oculta de $\mathscr{N}$;

(c) Conecte cada neurônio L_i ($1 \leq i \leq k$) na camada de entrada ao neurônio N_l na camada oculta. Se L_i for um literal positivo, defina o peso da conexão como W; caso contrário, defina o peso da conexão como $-W$;

(d) Conecte o neurônio N_l na camada oculta ao neurônio A na camada de saída e defina o peso da conexão como W;

(e) Defina o limiar (θ_l) do neurônio N_l na camada oculta como:

$$\theta_l = \frac{(1+A_{min})(k_l-1)}{2} W$$

(f)Defina o limiar(θ_A) do neurônio A na camada de saída como:

$$\theta_A = \frac{(1+A_{min})(1-\mu_l)}{2} W$$

(3) Defina *g(x)* como a função de ativação dos neurônios na camada de entrada de $\mathcal{N}$. Dessa forma, a ativação dos neurônios na camada de entrada, dada por cada vetor de entrada *i*, representará uma interpretação para $\mathcal{P}$.

(4) Defina *h(x)* como a função de ativação dos neurônios nas camadas oculta e de saída de $\mathcal{N}$. Dessa forma, um algoritmo de aprendizado de descida de gradiente, como o backpropagation, pode ser aplicado a $\mathcal{N}$ de forma eficiente.

Os significados de cada variável podem ser vistos em Garcez et al.(1999).

2.3 Extração de conhecimento

Embora as redes neurais tenham mostrado um desempenho muito bom em muitos domínios de aplicação, uma de suas principais desvantagens reside na incapacidade de fornecer uma explicação para os mecanismos de raciocínio subjacentes. As decisões que uma rede neural toma devem ser explicáveis. A habilidade de explicar como uma decisão de classificação é tomada pode levar a novas percepções e gerar um entendimento mais profundo do problema em consideração. A capacidade de explicação das redes neurais pode ser alcançada pela extração de conhecimento simbólico.

Existem três tipos principais de algoritmos de extração de conhecimento:

- Algoritmos decomposicionais
- Algoritmos pedagógicos
- Algoritmos ecléticos

Os algoritmos de extração de conhecimento decomposicionais tentam extrair conhecimento usando a estrutura interna do modelo - ANN (Artificial Neural Networks), SVM (Support Vector Machines) ou qualquer outro modelo caixa-preta. Isso significa que os algoritmos decomposicionais projetados para extração de conhecimento de uma família específica de redes neurais não são aplicáveis a outros tipos de classificadores. Por outro lado, tais métodos geralmente têm maior desempenho (em termos de acurácia e precisão) e geralmente devem ser executados mais rapidamente.

Abordagens de extração de conhecimento pedagógicas não fazem suposições em relação à estrutura interna do modelo sendo processado. Geralmente, esses algoritmos usam o modelo treinado como uma caixa-preta para amostrar o espaço de entrada e obter saídas densas. Isso permite que o algoritmo construa uma representação densa da fronteira de decisão e capture-a na forma de regras M de N ou árvores de decisão. É fácil observar que tais algoritmos são independentes do modelo, ou seja, podem ser aplicados não apenas às redes neurais treinadas, mas também a outros tipos de classificadores.

Abordagens ecléticas são uma combinação de abordagens decomposicionais e pedagógicas. Acredita-se que abordagens decomposicionais sejam capazes de produzir conhecimento mais preciso. Por outro lado, métodos pedagógicos são agnósticos em relação ao modelo e podem ser aplicados a diferentes tipos de classificadores (portáteis). Como os métodos decomposicionais acessam a estrutura interna de uma rede neural, eles têm muito mais poder na precisão e na abrangência do conhecimento extraído, e, no geral, tal algoritmo pode ser ajustado em um nível mais refinado do que um método pedagógico, que trata a rede neural como uma caixa preta/oráculo que prevê valores de classe para entradas dadas. A consistência do algoritmo, ou seja, a capacidade de produzir o mesmo resultado em várias execuções, é um ponto fraco para todos os algoritmos, mas atualmente os métodos decomposicionais são os mais fracos nesse aspecto.

A velocidade de execução dos algoritmos é algo discutível, pois os algoritmos pedagógicos precisam gerar entradas adicionais para localização precisa e forma do hiperplano de classificação, mas isso pode ser feito de maneira inteligente, enquanto os algoritmos decomposicionais lidarão com a completa complexidade interna (número de parâmetros treináveis - pesos ou neurônios) da rede neural. A avaliação da variedade da representação do conhecimento é novamente algo discutível, mas acredita-se que uma abordagem decomposicional dá uma vantagem em termos de facilidade na construção de um algoritmo que possa produzir um ou outro tipo de conhecimento. Em termos de escalabilidade, todos os tipos de algoritmos são igualmente paralelizáveis. No que diz respeito à complexidade do algoritmo, no momento, a contagem dos valores de saída do neurônio é o que consome a maior parte do tempo dentro das abordagens decomposicionais. Por outro lado, a geração inteligente de dados de entrada adicionais pode reduzir drasticamente a quantidade de cálculos. Portanto, um algoritmo tentará apenas gerar regras explicando as saídas em termos de entradas. Portanto, os algoritmos decomposicionais são geralmente considerados mais complexos computacionalmente.

As abordagens pedagógicas não são eficazes quando o tamanho da rede neural aumenta, como em aplicações do mundo real. Para superar essa limitação, os métodos de decomposição, em geral, aplicam buscas guiadas heurísticamente ao processo de extração. O método "Subset", por exemplo, tenta

procurar subconjuntos de pesos de cada neurônio nas camadas ocultas e de saída de N, de modo que o potencial de entrada dos neurônios exceda seu limite. Cada subconjunto que satisfaz a condição acima é escrito como uma regra. Um dos métodos de decomposição mais interessantes é a técnica M de N . Baseado no método Subset, ele usa clusterização e poda de pesos para facilitar a extração de regras. Ele também gera um número menor de regras, aproveitando a representação M de N, na qual $m(A_1, ..., A_n) \rightarrow A$ indica que se m de $(A_1, ..., A_n)$ são verdadeiros, então A é verdadeiro, onde $m \leq n$.

Os métodos de decomposição, em geral, usam mecanismos de poda de pesos antes da extração. No entanto, não há garantia de que uma rede podada seja equivalente à original. Essa é a razão pela qual esses métodos geralmente exigem o retreinamento da rede. Durante o retreinamento, algumas restrições devem ser impostas ao processo de aprendizado, por exemplo, permitindo apenas que os limites, mas não os pesos, mudem para que a rede mantenha sua estrutura podada "bem-comportada". Neste ponto, não há garantia de que o retreinamento será bem-sucedido sob tais restrições. Outros métodos de extração usam funções de penalidade durante o treinamento para tentar manter a estrutura "bem-comportada" inicial da rede e, assim, facilitar a extração. Tais métodos estão destinados a restringir a capacidade de aprendizado da rede, pois não seriam aplicáveis a uma rede treinada com um algoritmo de aprendizado "off the shelf". Mesmo que evitemos o uso de funções de penalidade e clusterização e poda de pesos, a simples tarefa de decompor a rede em sub-redes menores, das quais as regras são extraídas e, em seguida, montadas, deve ser realizada com cuidado. Isso ocorre porque, em geral, o efeito coletivo da rede é diferente do efeito da sobreposição de suas partes.

3. Descrição do Problema

No exemplo do trem de Michalski. A meta é classificar quais os trens que vão para leste e os que vão para oeste. Abaixo, figura 2, temos os exemplos dos modelos dos trens, 5 indo para o oeste e 5 indo para o leste.

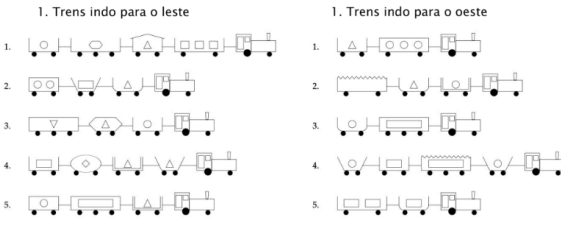

Figura 2 : Exemplos dos modelos de trem

Para cada trem temos os seguintes atributos:

1. Quantidade de vagões (valor entre 3 a 5);
2. Quantidade de cargas diferentes que pode levar (valor entre 1 a 4);
3. Para cada vagão do trem:
   a. A quantidade de eixo com rodas (valor entre 2 e 3);
   b. O comprimento (valor curto ou longo);
   c. O formato da carroceria do vagão, e pode ser
      i. Retângulo-fechado;
      ii. Retângulo-aberto;
      iii. Duplo retângulo-aberto;
      iv. Elipse;
      v. Hexágono;
      vi. Topo dentado;
      vii. Topo triangular-fechado;
   d. Quantidade de cargas no vagão (0 a 3);
   e. o formado de carga (círculo, hexágono, retângulo ou triângulo).

Para o trabalho foram propostas 10 variáveis booleanos (proposicionais) que descrevem se qualquer par de tipos de carga estão ou não em vagões adjacentes do trem (já que cada carro carrega um único tipo de carga). Temos as seguintes relações com respeito aos vagões de um trem, cujo valor lógico varia entre -1 (Falso) e 1 (Verdadeiro).

1. existem um retângulo próximo de um retângulo (V ou F)
2. existe um retângulo próximo de um triângulo (V ou F)
3. existe um retângulo próximo de um hexágono (V ou F)
4. existe um retângulo próximo de um círculo (V ou F)
5. existe um triângulo próximo de um triângulo (V ou F)
6. existe um triângulo próximo de um hexágono (V ou F)
7. existe um triângulo próximo de um círculo (V ou F)
8. existe um hexágono próximo de um hexágono (V ou F)
9. existe um hexágono próximo de um círculo (V ou F)
10. existe um círculo próximo de um círculo (V ou F)

Há um único atributo de classe que define a direção de um trem: leste ou oeste. Observe que para atributos com múltiplos valores deve-se assinalar valores numéricos na ordem em que surgem Por exemplo, o tipo de carga deve ser 1 para denotar círculo, 2 para hexágono, 3 para retângulo, e assim por diante. Os neurônios correspondentes devem usar função de ativação linear, i.e. h(x) = x.

Além de classificar corretamente se dado trem está indo para o leste ou oeste. O algoritmo deve extrair as regras de decisão da rede treinada, ou seja, irá definir

quais características de um trem que o levam a ir para o leste ou oeste. Após extrair as regras uma comparação deve ser feita com o algoritmo LTN.

Um pipeline de referência foi seguido para implementar o algoritmo de extração de regras da rede treinada:

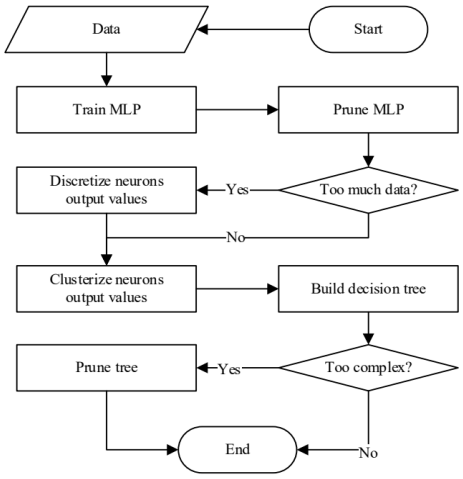

Figura 3 : Árvore de decisão do algoritmo de extração de regras

4. Descrição do Modelo

As técnicas utilizadas neste estudo foram baseadas no capítulo 10.6 do livro "Neural-symbolic cognitive reasoning" [5]. O trabalho foi realizado em Python, usando a biblioteca Keras, e as implementações podem ser encontradas no repositório [6]. A seção se divide em duas partes: a primeira descreve a implementação de uma Perceptron multicamadas e uma rede mais profunda composta por 11 sub-redes que representam regras específicas. A segunda parte descreve a expansão do modelo para outros tipos de casos.

4.1 Implementação 1

Na implementação 1, era necessário propor uma solução com base no modelo de aprendizado relacional contendo meta-redes para conectar conceitos das premissas. Este modelo deve conter 11 redes, uma para cada um dos seguintes conceitos conforme definição das páginas 136 e 137:

1. *num_cars(t, nc)*, em que t ∈ [1..10] e nc ∈ [3..5].
2. *num_loads(t, nl)* em que t ∈ [1..10] e nl ∈ [1..4].
3. num_wheels(t, c, w) em que t ∈ [1..10] e c ∈ [1..4] e w ∈ [2..3].
4. *length(t, c, l)* em que t ∈ [1..10] e c ∈ [1..4] e l ∈ [-1..1] (-1 denota curto e 1 longo).
5. *shape(t, c, s)* em que t ∈ [1..10] e c ∈ [1..4] e s ∈ [1..10] (um número para cada forma).
6. *num_cars_loads(t, c, ncl)* em que t ∈ [1..10] e c ∈ [1..4] e ncl ∈ [0..3].
7. *load_shape(t, c, ls)* em que t ∈ [1..10] e c ∈ [1..4] e ls ∈ [1..4].
8. *next_crc(t, c, x)* em que t ∈ [1..10] e c ∈ [1..4] e x ∈ [-1..1], em que o vagão c do trem t tem um vagão adjacente com cargas em círculo.
9. *next_hex(t, c, x)* em que t ∈ [1..10] e c ∈ [1..4] e x ∈ [-1..1], em que o vagão c do trem t tem um vagão adjacente com cargas em hexágono.
10. *next_rec(t, c, x)* em que *t* ∈ [1..10] e *c* ∈ [1..4] e x ∈ [-1..1], em que o vagão c do trem t tem um vagão adjacente com cargas em retângulo.
11. *next_tri(t, c, x)* em que *t* ∈ [1..10] e *c* ∈ [1..4] e *x* ∈ [-1..1], em que o vagão *c* do trem *t* tem um vagão adjacente com cargas em triângulo.

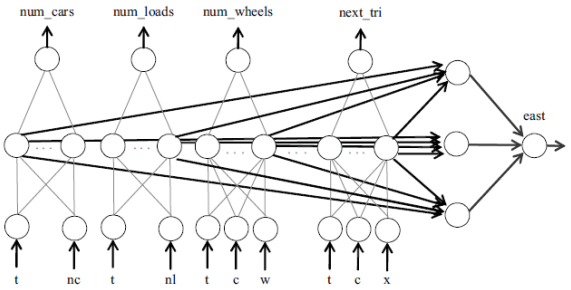

Figura 4 : Representação da meta-rede implementada por d'Avila Garces

A estrutura desenvolvida se baseia de acordo com o proposto no livro d'Avila Garces [5]. A taxa de aprendizado foi 0,01 o número de épocas foi 1000, e utilizou-se o otimizador *Adam*. Cada meta-rede possui 20 neurônios na camada escondida com função de ativação *Relu* e a camada de saída com um único neurônio com função de ativação *Sigmoid*.

A meta-rede foi treinada usando 36 exemplos e validada usando os 4 exemplos restantes. A última rede, que é responsável por classificar se cada vagão está indo para o *leste (1)* ou *oeste (0)*, foi conectada às camadas ocultas das 11 redes que representam regras específicas. A rede principal foi treinada usando os métodos de "leaving-one-out" e "cross-validation" em relação a cada trem. Os dados com 10 instâncias foram expandidos para 40 porque cada trem pode ter até 4 vagões.

Com intuito de comparar os resultados da implementação da questão 2, foi feito outro teste com a implementação 1 usando os dados dos 18 trens, ou seja, juntou-se às amostras da questão 1 as da questão 2.

4.2 Implementação 2

Na implementação 2, era pedido para estender o modelo para demais tipos de casos como mostrado na Figura 5, descrevendo os novos predicados como na questão anterior e treinar o modelo como no exemplo do livro.

Os novos predicados descritos para segunda implementação foram 3, sendo eles:

1. *roof_car(t, c, rc)* em que $t \in [1..18]$, $c \in [1..4]$ e $rc \in [-1..1]$, em que o vagão *c* do trem *t* tem o teto fechado (*rc* = 1) ou não (rc = -1).
2. *num_jagged_top(t, nt)* em que $t \in [1..18]$, $nt \in [1..4]$, em que em um trem t possui nt vagões com teto irregular.
3. *num_cars_dif(t, ncd)*, em que $t \in [1..18]$, $ncd \in [1..4]$, onde ncd representa o número de vagões diferentes em um trem t.

Perceba que o valores do parâmetro t pertencem a um intervalo maior de valores, devido a inserção de mais oito amostras para realizar o treinamento e validação do modelo. Para efeitos de comparação com o resultado da primeira questão, realizou-se dois testes, sendo eles: Primeiro, foi feito treino com os 18 trens, o segundo treino foi feito apenas com os mesmos trens que foram passados para modelo da primeira implementação.

A estrutura geral da meta-rede é semelhante à da implementação 1, diferindo apenas pela inserção de 3 meta-redes devido aos novos predicados e, em relação ao treinamento, a diferença foi na taxa de aprendizado que, para esta implementação, foi de 0,01.

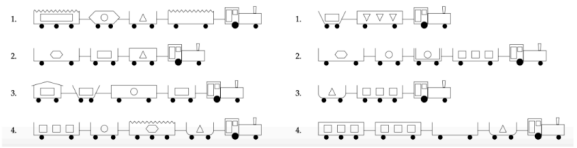

Figura 5 : Exemplos para estender o modelo

4.3 Implementação 3

Para a implementação 3, pediu-se para extrair da rede treinada na implementação 2, as regras que descrevem quais características de um trem que vai para o leste e comparar com a implementação do modelo LTN da turma de 2021.

Para esse problema, utilizamos uma implementação do algoritmo de construção de árvore de decisão chamada TREPAN, que é descrito em [11]. Esse algoritmo utiliza uma rede neural treinada e um conjunto de dados de treinamento como entrada. Então, é produzida uma árvore de decisão que fornece uma aproximação da função que representa a rede treinada em questão. A vantagem do TREPAN em relação a outros algoritmos semelhantes é que ele desenha novas amostras enquanto cresce a árvore com base em critérios mínimos de amostra em cada nó da árvore e usa o modelo treinado para atribuir rótulos a novas amostras.

A forma de extrair as regras são baseadas nas formas de operação em decision tree do MLJAR que se baseia em percorrer a árvore de decisão até as folhas, extraindo as features e colocando em uma regra para cada caminho da árvore.

5. Resultados e Análises

5.1 - Resultados da Implementação 1

Os resultados obtidos na implementação 1 estão representados na tabela 1. Observa-se que o modelo desenvolvido aprendeu bem os casos passados como treinamento e só teve um caso onde houve um erro na predição da direção do trem, que no caso foi para trem 4.

A partir do gráfico da figura 6, verifica-se que o modelo aprende muito rápido os dados de treinamento antes mesmo de chegar a 100 épocas. Por outro lado, a validação se estabilizou a partir de 200 épocas de treinamento.

	Train	Cars Accuracy	Output of flat network	Desired output	Class
0	0	1.0	1.00	1.0	east
1	1	1.0	1.00	1.0	east
2	2	1.0	1.00	1.0	east
3	3	1.0	1.00	1.0	east
4	4	1.0	0.26	1.0	east
5	5	1.0	0.02	0.0	west
6	6	1.0	0.01	0.0	west
7	7	1.0	0.00	0.0	west
8	8	1.0	0.01	0.0	west
9	9	1.0	0.00	0.0	west

Tabela 1 : Resultados para o treinamento com meta-redes

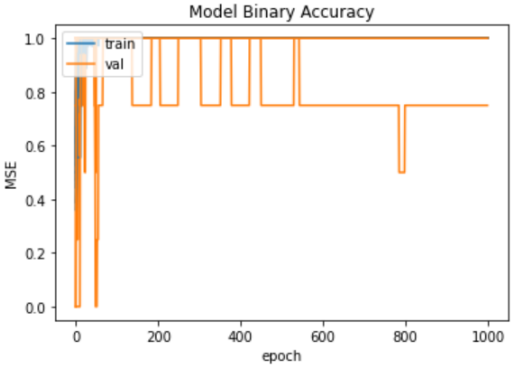

Figura 6: Acurácia binária do modelo para os testes com predicados originais

5.2 - Resultados da Implementação 2

Para implementação do modelo com novos predicados e com mais dados para treinamento, percebe-se que a adição dos novos predicados não contribuiu para a melhora da classificação do modelo. Em alguns casos, o modelo não chegou a aprender bem os dados de treino, como pode ser observado na tabela 2. Levando em conta o teste do trem 16, por exemplo, temos que o modelo apresentou uma acurácia de 52,9% para o treino, praticamente chutando para qual direção o trem estava indo. Outra coisa interessante a se observar é que o modelo errou mais. Para os primeiros 10 trens, houveram 3 erros, enquanto que para a primeira implementação houve apenas 1 dentre as 10 instâncias.

	Train	Cars Accuracy	Output of flat network	Desired output	Class
0	0	0.529412	0.48	1.0	east
1	1	1.000000	1.00	1.0	east
2	2	0.764706	1.00	1.0	east
3	3	1.000000	1.00	1.0	east
4	4	0.764706	0.77	1.0	east
5	5	1.000000	0.00	0.0	west
6	6	1.000000	0.23	0.0	west
7	7	0.985294	0.67	0.0	west
8	8	0.764706	0.34	0.0	west
9	9	0.529412	0.53	0.0	west
10	10	1.000000	0.00	1.0	east
11	11	1.000000	0.39	1.0	east
12	12	1.000000	0.00	1.0	east
13	13	0.823529	0.26	1.0	east
14	14	0.764706	0.34	0.0	west
15	15	1.000000	0.19	0.0	west
16	16	0.529412	0.53	0.0	west
17	17	1.000000	0.01	0.0	west

Tabela 2 : Resultados para o treinamento com meta-redes estendidas para 18 trens

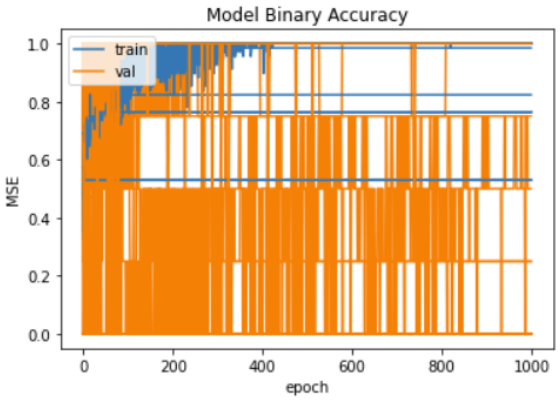

Figura 7: Acurácia binária do modelo para os testes estendidos para 18 trens e com os novos predicados

5.3 - Resultados de comparação dos novos predicados e trens

O teste adicional, que verifica o desempenho das meta-redes com novos predicados para o mesmo conjunto de dados da implementação 1, mostrou que os novos predicados não afetam negativamente o modelo de classificação. Observando primeiramente o resultado da figura 8, pode-se ter duas sugestões da piora do modelo: uma é em relação à escolha de novos predicados, e a outra sendo os próprios dados adicionais.

Analisando os resultados da tabela 3 e da tabela 2, nota-se que o modelo da primeira implementação errou 8 casos de teste e o modelo com novos predicados apresentou 9 erros. Desta forma, é nítido que os novos predicados não tiveram uma contribuição decisiva para modelo e que os predicados já utilizados não são eficientes quando o treinamento é feito com os dados adicionais da figura 2.

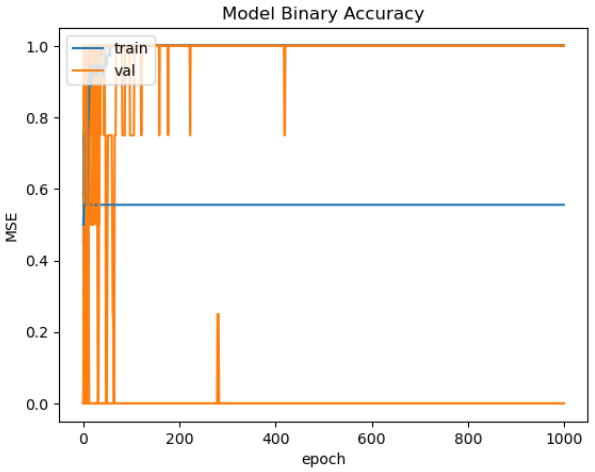

Figura 8: Acurácia binária do modelo para os testes estendidos para 10 trens com novos predicados

	Train	Cars Accuracy	Output of flat network	Desired output	Class
0	0	1.000000	0.99	1.0	east
1	1	1.000000	0.99	1.0	east
2	2	1.000000	1.00	1.0	east
3	3	1.000000	1.00	1.0	east
4	4	1.000000	0.01	1.0	east
5	5	1.000000	0.11	0.0	west
6	6	1.000000	0.01	0.0	west
7	7	0.555556	0.56	0.0	west
8	8	1.000000	0.01	0.0	west
9	9	1.000000	0.01	0.0	west

Tabela 3: Resultados para o treinamento com meta-redes estendidas para 10 trens

As amostras da primeira implementação são facilmente classificadas com as seguintes teorias:

a. Se um trem tem um vagão curto e fechado, então ele vai para o leste, caso contrário, vai para o oeste.
b. Se um trem tem dois vagões, ou tem um vagão com teto irregular, então ele vai para o oeste, caso contrário, vai para o leste.
c. Se um trem tiver mais de dois tipos diferentes de carga, então ele vai para o leste, ao contrário, vai para o oeste.

Tratando-se das amostras da figura 5, a única teoria que classifica todos os casos é:

a. Se um trem tem um vagão curto e fechado, então ele vai para o leste, caso contrário, vai para o oeste

Portanto, os dados adicionais afunilam as regras que podem ser utilizadas para classificar o sentido dos trens. Dessa forma, pode-se justificar o mal desempenho dos modelos quando foram treinados com 18 trens, pois os predicados já existentes e os novos não permitem verificar que apenas a teoria acima já é auto suficiente para realizar a classificação.

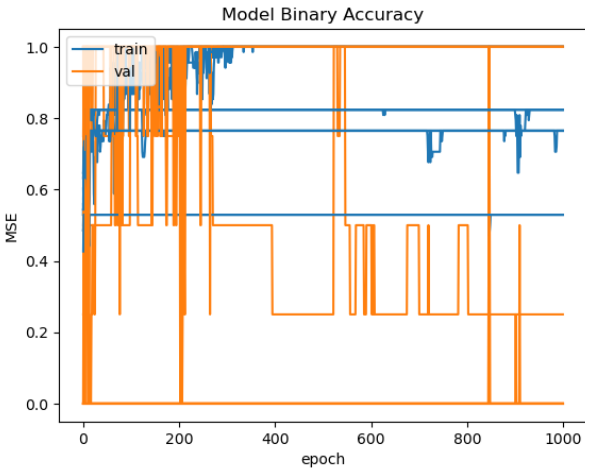

Figura 9: Acurácia binária do modelo para os testes com predicados originais para 18 trens

Train	Cars Accuracy	Output of flat network	Desired output	Class
0	0.529412	0.48	1.0	east
1	1.000000	0.99	1.0	east
2	0.764706	1.00	1.0	east
3	0.529412	0.48	1.0	east
4	1.000000	1.00	1.0	east
5	0.764706	0.56	0.0	west
6	0.764706	0.35	0.0	west
7	0.764706	0.36	0.0	west
8	0.764706	0.35	0.0	west
9	0.529412	0.53	0.0	west
10	0.823529	0.27	1.0	east
11	0.823529	0.27	1.0	east
12	0.823529	0.27	1.0	east
13	0.529412	0.48	1.0	east
14	1.000000	0.00	0.0	west
15	0.764706	0.35	0.0	west
16	0.764706	0.35	0.0	west
17	1.000000	0.00	0.0	west

Tabela 4: Resultados para o treinamento com meta-redes com predicados originais para 18 trens

5.4 - Resultados da Implementação 3

Em decorrência das complexidades advindas dos tipos de modelos em meta-redes e do código exigido para extrair a árvore de decisão correspondente a cada meta-rede, não foi possível obter os resultados desejados.

6. Considerações finais

Comparando as duas implementações, verificou-se que a implementação com 11 meta-redes se saiu melhor para classificar a trajetória dos trens. Uma justificativa para esse resultado é que os novos predicados escolhidos para realizar a classificação contribuíram, na verdade, para confundir o modelo, ou seja, colocou-se mais dados que na verdade trouxeram excesso de informações inúteis para a classificação. Uma possível atividade futura é verificar o desempenho da implementação 1 para o conjunto de dados utilizados na implementação 2, para que assim seja feito uma comparação justa entre as implementações.

Apesar dos resultados da primeira implementação serem encorajadores, vale ressaltar que não fica claro, ao inspecionar a meta-rede, que a classificação pode ser descrita como: "*Se um trem tem um vagão curto e fechado, então ele vai para leste, caso contrário, vai para oeste*". Para chegar em tal conclusão se utiliza extração de conhecimento de redes neurais.

Embora tenham sido identificados alguns desafios durante o processo de transferência das entradas necessárias para o TREPAN, a implementação é capaz

de utilizar os métodos idealizados em [6] com algumas limitações.

Deixamos como trabalho futuro o dimensionamento dos vetores de valores da rede estendida e a compatibilidade dos tipos de modelo em meta-redes. Desse modo, será possível extrair a árvore de decisão a partir dos valores do modelo treinado e comparar os dois modelos treinados.

7. Referências

[1] Neuro-Symbolic Methods for Knowledge Representation and Reasoning por Artur d'Avila Garcez, Krysia Broda, Dov Gabbay, e Luis C. Lamb

[2] Neuro-Symbolic Learning and Reasoning: A Survey and Interpretation" por Kai-Uwe Kühnberger, Sebastian Rudolph e Pei Wang

[3] Hybrid Artificial Intelligence Systems: Third International Workshop editado por Emilio Corchado, Hujun Yin e Vicente Botti

[4] Deep Neuro-Symbolic Reinforcement Learning por Li Zhao, Bingbing Li e Tao Xie

[5] Artur S d'Avila Garcez, Luís C Lamb, and Dov M Gabbay. Neural-symbolic learning systems. Neural-Symbolic Cognitive Reasoning, pages 35–54, 2009.

[6] Mark W. Craven e Jude W. Shavlik. Extracting Tree-Structured Representations of Trained Networks, Advances in Neural Information Processing Systems, vol. 8, pages 24-30, 1996.

[7] Artur d'Avila Garcez, Gerson Zaverucha, The Connectionist Inductive Learning and Logic Programming System, Applied Intelligence vol. 11, pages 59–77, 1999.

[8]A.S. d'Avila Garcez, K. Broda, D.M. Gabbay, Symbolic knowledge extraction from trained neural networks: A sound approach, Artificial Intelligence vol. 125, pages 155-207, 2001.

[9]Andrey Bondarenko, Ludmila Aleksejeva, Methodology for Knowledge Extraction from Trained Artificial Neural Networks, Information Technology and Management Science vol. 21, pages 6–14, 2018.

[10] https://github.com/alexandreuch/IA

[11] CRAVEN, Mark William. Extracting comprehensible models from trained neural networks. The University of Wisconsin-Madison, 1996.