Aplicação de Redes Lógicas Tensoriais para Análise e Resolução Estratégica de Sudoku 4x4

Antonio Mileysson França Bragança
Engenharia da Computação
Instituto de Computação – Universidade Federal do Amazonas (UFAM)

22 de julho de 2025

Resumo

Este relatório detalha a concepção e implementação de um sistema de Inteligência Artificial para a análise de quebra-cabeças de Sudoku 4x4, utilizando o paradigma neuro-simbólico através de Redes Lógicas Tensoriais (LTN). O trabalho explora uma progressão de tarefas com complexidade crescente: (1) a classificação de satisfatibilidade de tabuleiros completos, (2) a análise e ranqueamento de movimentos em tabuleiros incompletos, e (3) a recomendação de heurísticas de resolução com base no estado do jogo. Implementados em LTNtorch, os modelos demonstram a eficácia da integração de conhecimento de domínio, expresso em lógica de primeira ordem, com a capacidade de aprendizado de padrões das redes neurais. Os resultados indicam alta performance na validação de regras lógicas e na classificação de estratégias, validando a abordagem neuro-simbólica como uma metodologia robusta para problemas que exigem raciocínio estruturado.

Sumário

1 Introdução

A interseção entre o aprendizado de máquina e o raciocínio simbólico representa uma das fronteiras mais promissoras da Inteligência Artificial (IA). Enquanto modelos de aprendizado profundo (*Deep Learning*) se destacam em tarefas de percepção e reconhecimento de padrões a partir de grandes volumes de dados, eles frequentemente operam como "caixas-pretas", carecendo de interpretabilidade e robustez em domínios que exigem raciocínio lógico explícito. Por outro lado, sistemas simbólicos clássicos, embora transparentes, são frágeis e não possuem a capacidade de aprender com dados brutos.

O Sudoku, apesar de sua aparente simplicidade, encapsula a essência de um Problema de Satisfação de Restrições (CSP), exigindo lógica, dedução e aplicação de estratégias. Ele serve, portanto, como um excelente domínio de teste para arquiteturas de IA que buscam unir aprendizado e raciocínio.

Este trabalho propõe a utilização de Redes Lógicas Tensoriais (LTN) para abordar o problema do Sudoku 4x4. LTN é um framework de IA neuro-simbólica que permite a representação de conhecimento em lógica de primeira ordem diretamente em um ambiente de aprendizado profundo. O objetivo deste estudo é demonstrar, através de três modelos de complexidade crescente, como essa abordagem pode ser usada para:

1. Validar a conformidade de uma solução com as regras do jogo.
2. Orientar a tomada de decisão em estados intermediários do jogo.
3. Classificar a estratégia de resolução mais adequada para um determinado cenário.

Este relatório detalha o referencial teórico, a descrição formal do problema, a implementação de cada modelo e os resultados obtidos, destacando o potencial da abordagem neuro-simbólica.

2 Referencial Teórico

2.1 Inteligência Artificial Neuro-Simbólica

A IA Neuro-Simbólica (*Neuro-Symbolic AI*) é um campo que busca a sinergia entre as redes neurais e a lógica simbólica. A premissa fundamental é que a combinação das duas abordagens pode levar a sistemas de IA mais robustos, explicáveis e generalizáveis. As redes neurais são utilizadas para tarefas de baixo nível, como percepção e aprendizado de representações a partir de dados, enquanto os componentes simbólicos são empregados para o raciocínio de alto nível, manipulação de conhecimento e garantia de consistência lógica.

2.2 Redes Lógicas Tensoriais (LTN)

As Redes Lógicas Tensoriais são uma manifestação concreta da IA neuro-simbólica, proposta por Serafini e Garcez (2016) [1]. Elas fornecem uma linguagem e um método para traduzir axiomas da lógica de primeira ordem em uma rede computacional diferenciável, permitindo que o aprendizado ocorra através de gradiente descendente. Os seus principais componentes são:

- **Termos e Predicados:** Termos são variáveis ou constantes lógicas que são "aterradas" (*grounded*) em tensores do PyTorch. Predicados são funções parametrizadas (tipicamente, redes neurais) que mapeiam os tensores de entrada para um valor de verdade no intervalo contínuo $[0, 1]$, representando o grau de satisfação de uma propriedade.
- **Conectivos Lógicos:** Operadores como conjunção ($\wedge$), disjunção ($\vee$) e implicação ($\rightarrow$) são implementados utilizando normas-t e normas-s da lógica fuzzy, que são funções diferenciáveis.
- **Quantificadores:** Os quantificadores universal ($\forall$, "para todo") e existencial ($\exists$, "existe") são implementados através de operadores de agregação diferenciáveis (e.g., `AggregMin` para $\forall$, `AggregPMean` para $\exists$), que calculam o nível de satisfação de uma fórmula sobre um conjunto de dados.

Essa estrutura permite que um sistema aprenda a partir de dados enquanto adere a um conjunto de restrições lógicas predefinidas.

3 Descrição do Problema

O domínio do problema é o quebra-cabeça de Sudoku 4x4, um CSP onde o objetivo é preencher um grid 4x4 com os dígitos $\{1, 2, 3, 4\}$, de modo que cada linha, coluna e sub-bloco 2x2 contenha cada dígito exatamente uma vez. Este trabalho aborda três problemas distintos neste domínio:

1. **Problema 1: Validação de Soluções:** Dado um tabuleiro 4x4 completamente preenchido, classificá-lo como "válido"(satisfaz todas as restrições) ou "inválido"(viola ao menos uma restrição). Esta é uma tarefa de classificação binária baseada em satisfatibilidade lógica.
2. **Problema 2: Análise de Movimentos:** Dado um tabuleiro incompleto, avaliar todos os movimentos possíveis e ranqueá-los de acordo com seu potencial de levar a uma solução válida. Esta é uma tarefa de pontuação e ranqueamento para suporte à decisão.

3. **Problema 3: Recomendação de Estratégia:** Dado um tabuleiro incompleto, determinar qual heurística de resolução humana (ex: *Naked Single*, *Hidden Single*, *Locked Candidate*) é a mais apropriada para ser aplicada. Esta é uma tarefa de classificação multiclasse.

4 Implementação

O sistema foi desenvolvido em Python utilizando `PyTorch`, `LTNtorch`, `NumPy` e `Pandas`, dentro de um ambiente Google Colab. Como referência metodológica, foi adotado o sistema proposto por Lia Morra *et al.*[2], o qual serviu de base para a estrutura e abordagem utilizadas.

4.1 Geração de Dados

Para cada um dos três problemas, datasets foram gerados proceduralmente.

- **Dataset 1:** Tabuleiros válidos foram criados usando um algoritmo de backtracking, e tabuleiros inválidos foram gerados introduzindo erros deliberados (e.g., duplicatas).
- **Dataset 2:** Tabuleiros incompletos foram gerados para a análise de movimentos.
- **Dataset 3:** Um dataset mais complexo foi criado para o recomendador de heurísticas, onde cada tabuleiro incompleto foi rotulado com a heurística mais simples aplicável, detectada por um algoritmo de verificação.

4.2 Arquitetura do Modelo 1: `SudokuLTN` (Validador)

O primeiro modelo, `SudokuLTN`, foi projetado como um classificador binário para o problema de satisfatibilidade. Seu objetivo é determinar se um tabuleiro 4x4 completamente preenchido adere a todas as restrições do Sudoku. A principal característica desta arquitetura é a sua construção neuro-simbólica, que traduz as regras de primeira ordem do jogo em uma estrutura de rede neural diferenciável.

Representação de Entrada e Arquitetura dos Predicados

O modelo recebe como entrada um tensor de dimensões 4×4 representando o tabuleiro, com seus valores numéricos normalizados para o intervalo $[0, 1]$. Internamente, este tensor é tratado como três domínios de discurso distintos: um conjunto de 4 tensores representando as linhas, um conjunto de 4 tensores para as colunas e um conjunto de 4 tensores para os blocos 2x2.

A capacidade de aprendizado do modelo está concentrada em três predicados lógicos:

`row_pred`, `col_pred` e `block_pred`. A arquitetura destes predicados foi projetada para eficiência de parâmetros, utilizando uma camada de extração de características compartilhada (`shared_layer`), implementada como um Perceptron de Múltiplas Camadas (MLP),

seguida por três camadas lineares de saída distintas, uma para cada predicado. Esta abordagem permite que o modelo aprenda características comuns a todas as unidades (linhas, colunas e blocos), especializando-se apenas na camada final. A função de ativação *Sigmoid* ao final de cada caminho garante que a saída do predicado seja um valor de verdade no intervalo $[0, 1]$.

Formulação Lógica e Processo de Aprendizagem

O núcleo do modelo é a codificação do seguinte axioma lógico que define uma solução de Sudoku válida:

$$\forall r \in \text{linhas}, P_{\text{linha}}(r) \land \forall c \in \text{colunas}, P_{\text{coluna}}(c) \land \forall b \in \text{blocos}, P_{\text{bloco}}(b)$$

É fundamental notar que a semântica de um predicado (e.g., o que constitui uma "linha válida") não é pré-programada. Em vez disso, ela é *aprendida* a partir dos dados durante o treinamento. Ao ser exposto a exemplos de tabuleiros rotulados como válidos (1.0) e inválidos (0.0), o otimizador (Adam) ajusta os pesos dos MLPs dos predicados através de backpropagation. O objetivo é minimizar uma função de perda, como o Erro Quadrático Médio (MSE), entre a saída do modelo e o rótulo verdadeiro. Como resultado, o predicado `row_pred`, por exemplo, aprende a mapear vetores de entrada que representam linhas com valores únicos para um valor de verdade próximo a 1.0, e vetores com valores duplicados para um valor próximo a 0.0.

Agregação Lógica e Saída

A instanciação da fórmula lógica no framework LTN utiliza operadores específicos da lógica fuzzy para garantir a diferenciabilidade:

- **Quantificador Universal ($\forall$):** Implementado com o agregador `AggregMin`. Esta escolha impõe uma interpretação rigorosa da lógica: para que a afirmação "todas as linhas são válidas"seja verdadeira, a verdade da linha com o menor grau de satisfação (o "elo mais fraco") deve ser alta. Isso modela efetivamente uma conjunção sobre todos os elementos do domínio.
- **Conjunção ($\land$):** O conectivo "E"que une as três cláusulas principais (sobre linhas, colunas e blocos) é implementado com `AndMin`.

Durante o *forward pass*, o modelo avalia o grau de verdade de cada uma das 4 linhas, 4 colunas e 4 blocos usando os respectivos predicados. Em seguida, o operador de quantificação agrega esses 12 valores de verdade em três pontuações escalares (uma para cada tipo de unidade). Finalmente, o conectivo de conjunção combina essas três pontuações em um único valor de verdade escalar final no intervalo $[0, 1]$. Este escalar representa a confiança do modelo na validade do tabuleiro como um todo e é o resultado final da computação.

4.3 Arquitetura do Modelo 2: `LTNMoveAnalyzer` (Analisador)

A transição da validação de tabuleiros completos para a análise de tabuleiros incompletos representa uma mudança fundamental no problema, passando de uma classificação estática para uma tarefa de suporte à decisão dinâmica. Para abordar esta nova complexidade, foi desenvolvida uma arquitetura híbrida, `LTNMoveAnalyzer`, que combina a eficiência da lógica simbólica determinística com a capacidade de avaliação de padrões do modelo neuro-simbólico treinado (**Modelo 1**).

Estrutura Híbrida do Sistema

O sistema é composto por dois módulos principais que operam em sequência:

1. **Componente Simbólico-Heurístico (`SudokuHeuristicAnalyzer`):** Este módulo atua como um filtro de pré-processamento rápido e de baixa complexidade computacional. Sua função é aplicar um conjunto de regras lógicas rígidas para identificar estados de jogo que são trivialmente insolúveis. Por exemplo, ele verifica se um dígito que ainda precisa ser posicionado no tabuleiro não possui nenhuma célula vazia onde possa ser legalmente inserido. Ao detectar tais contradições, o módulo encerra a análise prematuramente, otimizando o processo ao evitar avaliações desnecessárias pelo componente neural, que é computacionalmente mais intensivo.

2. **Componente Neuro-Simbólico (`LTNClassifier`):** Se o tabuleiro passar pela filtragem heurística, este módulo é ativado. Ele não retreina, mas sim *reutiliza* o **Modelo 1** (`SudokuLTN`) como uma **função de avaliação de estados** (*state evaluation function*). Sua finalidade é quantificar a "qualidade"ou o "potencial"de um estado de jogo, mesmo que incompleto.

Mecanismo de Avaliação de Movimentos

O processo de análise de movimentos implementa uma busca *lookahead* de um passo. Para cada célula vazia no tabuleiro, o sistema enumera todos os movimentos legalmente possíveis (i.e., colocar um dígito que não viole as regras básicas do Sudoku). Para cada movimento candidato (r, c, n), onde n é o dígito a ser colocado na célula (r, c), o sistema executa os seguintes passos:

1. **Geração de Estado Hipotético:** Um novo estado do tabuleiro é criado em memória, aplicando-se o movimento candidato.

2. **Tratamento da Incerteza:** O **Modelo 1** foi treinado em tabuleiros completos. Para avalizar um estado parcial, as células que permanecem vazias (valor 0) são preenchidas com um valor numérico neutro (`NEUTRAL_FILL_VALUE` = 2.5). Após a normalização para o intervalo $[0, 1]$, este valor se torna 0.5. Esta escolha é deliberada: em um contexto

de lógica fuzzy, 0.5 representa a máxima incerteza ou entropia (nem verdadeiro, nem falso), instruindo o modelo a focar seu julgamento na integridade estrutural das células já preenchidas e no impacto do novo movimento, em vez de penalizar o estado pela simples presença de células vazias.

3. **Geração de Pontuação:** O estado hipotético e normalizado é então passado para o **Modelo 1**. A saída escalar do modelo, um valor no intervalo $[0, 1]$, é interpretada não mais como uma classificação binária, mas como uma pontuação de potencial para aquele movimento.

Saída e Interpretação

Após avaliar todos os movimentos candidatos, o sistema produz como saída uma lista de todos os movimentos válidos, ordenados de forma decrescente com base em suas respectivas pontuações de potencial. Esta lista funciona como uma política de ação (*ranked policy*) que pode ser usada para guiar um jogador humano ou um agente autônomo, sugerindo qual movimento tem a maior probabilidade de manter o tabuleiro em um caminho que leve a uma solução válida.

4.4 Arquitetura do Modelo 3: `LTNHeuristicRecommender4x4`

O terceiro e mais avançado modelo, `LTNHeuristicRecommender4x4`, representa uma mudança de paradigma em relação aos modelos anteriores. O seu objetivo não é validar uma solução ou avaliar um movimento, mas sim realizar uma tarefa de meta-raciocínio: classificar o estado atual do jogo para *recomendar a estratégia de resolução (heurística) mais apropriada*. Esta abordagem visa emular um aspecto da cognição humana, onde um especialista seleciona uma tática com base no reconhecimento de padrões de alto nível.

Engenharia de Features para Representação de Estado

Dada a natureza abstrata da tarefa, uma representação de entrada rica é fundamental. O modelo utiliza um tensor de features multi-canal de dimensões $3 \times 4 \times 4$, onde cada canal fornece uma perspectiva distinta sobre o estado do tabuleiro:

- **Canal 1: Valores das Células.** Contém os valores numéricos do tabuleiro, normalizados para o intervalo $[0, 1]$. Este canal fornece a informação fundamental sobre o estado do jogo.
- **Canal 2: Contagem de Candidatos.** Para cada célula vazia, este canal armazena a contagem de dígitos legalmente possíveis, normalizada pelo número total de dígitos (4). Este é um canal crítico, pois heurísticas como *Naked Single* são diretamente definidas por uma contagem de candidatos igual a 1, enquanto outras dependem da distribuição dessas contagens.

- **Canal 3: Ocupação de Células.** Um mapa binário que indica se uma célula está preenchida (1) ou vazia (0). Este canal fornece um contexto espacial explícito.

Esta representação multi-canal permite que as camadas convolutivas ou lineares subsequentes aprendam correlações complexas entre os valores, as possibilidades e a estrutura espacial do quebra-cabeça.

Arquitetura Neuro-Simbólica

A arquitetura do modelo pode ser decomposta em três estágios funcionais:

1. Predicados como Funções de Aterramento de Conceitos. O modelo define três predicados LTN especializados: `NakedSinglePotential`, `HiddenSinglePotential` e `LockedCandidatePotential`. Cada predicado é um MLP treinado para funcionar como uma função de aterramento de conceito (*concept grounding function*). Sua tarefa é aprender a mapear as features de baixo nível (do tensor de entrada) para um valor de verdade em $[0, 1]$ que representa a existência do conceito lógico correspondente. Notavelmente, os predicados operam em domínios de discurso diferentes, refletindo a natureza de cada heurística:

- `NakedSinglePotential` opera sobre as features de células individuais.
- `HiddenSinglePotential` e `LockedCandidatePotential` operam sobre features agregadas de unidades inteiras (linhas, colunas ou blocos), pois essas heurísticas são propriedades de um conjunto de células, não de uma única célula isolada.

2. Agregação Lógica e Geração do Vetor Semântico. O quantificador existencial ($\exists$), implementado com o agregador `AggregPMean`, é aplicado para cada um dos três predicados. Ele agrega os valores de verdade sobre o domínio relevante (e.g., todas as células para o `NakedSinglePotential`) para produzir uma única pontuação de confiança para cada heurística. O resultado deste estágio é um **vetor de características heurísticas** de 3 dimensões. Este vetor é uma representação de alto nível e semanticamente rica do tabuleiro, que não descreve mais os valores das células, mas sim o potencial estratégico do estado atual.

3. Cabeça de Classificação e Tratamento da Heurística Implícita. O vetor de características heurísticas de 3 dimensões é então alimentado em uma camada de saída linear (*output_layer*) que funciona como uma cabeça de classificação (*classification head*). Esta camada mapeia o espaço semântico de 3 dimensões para um espaço de logits de 4 dimensões, correspondente às quatro classes de estratégias:

1. Naked Single
2. Hidden Single

3. Locked Candidate

4. Backtracking (Força Bruta)

É crucial notar que a quarta classe, *Backtracking*, é tratada como um caso implícito. O modelo aprende a predizer esta classe quando as pontuações de potencial para as três heurísticas explicitamente modeladas são baixas. A camada linear aprende essa relação de "caso contrário"durante o treinamento.

Processo de Aprendizagem

A arquitetura completa, do processamento de entrada à cabeça de classificação, é treinada de ponta a ponta (*end-to-end*). A função de perda utilizada é a `CrossEntropyLoss`, padrão para problemas de classificação multiclasse. O processo de otimização ajusta simultaneamente tanto os pesos dos MLPs dos predicados (ensinando-os a detectar os padrões corretos) quanto os pesos da camada de classificação final (ensinando-a a tomar a melhor decisão com base nas evidências fornecidas pelos predicados).

5 Resultados

A avaliação de cada um dos três modelos foi conduzida utilizando os datasets e os procedimentos de teste descritos na seção de implementação. Os resultados quantitativos e qualitativos são apresentados a seguir.

5.1 Resultados do Modelo 1: Validador de Tabuleiros

Desempenho no Treinamento

O treinamento do modelo `SudokuLTN` foi realizado por 70 épocas. O progresso da função de perda (MSE) para os conjuntos de treino e validação foi monitorado, com os resultados sumariados na Tabela 1. O modelo demonstrou uma convergência estável, com a perda de validação diminuindo consistentemente e se estabilizando em um valor baixo, indicando que não houve sobreajuste (*overfitting*). A melhor perda de validação registrada foi de **0.13835**, evidenciando a capacidade do modelo de aprender um mapeamento robusto entre os estados do tabuleiro e sua validade lógica.

Análise Qualitativa da Validação

Após o treinamento, o modelo foi avaliado em um conjunto de cinco tabuleiros de teste não vistos, contendo tanto exemplos válidos quanto inválidos com diferentes tipos de violações. O modelo classificou corretamente todos os casos, conforme demonstrado nos exemplos abaixo:

Tabela 1: Progresso da Função de Perda (MSE) durante o Treinamento do Modelo 1.

Época	Perda de Treino	Perda de Validação
5	0.22813	0.22336
20	0.16234	0.16303
40	0.14091	0.14102
45	0.13963	0.13943
65	0.13882	0.13934
Melhor Perda de Validação: 0.13835		

- **Tabuleiros Válidos:** Foram corretamente identificados como ‘Válido¡.
- **Tabuleiros Inválidos:** O modelo demonstrou ser robusto na detecção de falhas, identificando corretamente tabuleiros com erros sutis (uma duplicata em uma linha) e erros grosseiros (múltiplas duplicatas em linhas e colunas), classificando-os todos como ‘Inválido¡.

Este resultado confirma que o modelo não apenas memorizou os dados de treino, mas generalizou com sucesso as regras lógicas do Sudoku para classificar novas instâncias com precisão.

5.2 Resultados do Modelo 2: Analisador de Movimentos

A performance do sistema híbrido `LTNMoveAnalyzer` foi avaliada qualitativamente em dois cenários distintos.

Cenário 1: Tabuleiro com Solução Possível

Para o primeiro tabuleiro de entrada, que era complexo mas potencialmente solucionável, o sistema operou conforme o esperado:

1. **Classificação Inicial:** O componente heurístico não encontrou contradições, e o sistema avançou para a análise neuro-simbólica, classificando-o como ‘(2) Solução Possível‘.
2. **Análise de Movimentos:** O modelo gerou uma lista de todos os movimentos válidos, ranqueados por uma pontuação de potencial. Observou-se que o modelo foi capaz de discriminar entre os movimentos, atribuindo pontuações distintas (e.g., o melhor movimento obteve ‘0.0067‘, enquanto o mais arriscado obteve ‘0.0008‘). Embora os valores absolutos das pontuações sejam baixos – o que é esperado para um tabuleiro com muitas células vazias e, portanto, distante de um estado final "válido--, a sua capacidade de ranqueamento relativo foi claramente demonstrada.

Cenário 2: Tabuleiro Sem Solução

Para o segundo tabuleiro, que continha uma violação de regra fundamental (duas ocorrências do dígito ’3’ na terceira linha preenchida), o resultado demonstrou a eficiência da arquitetura

híbrida:

- **Classificação Inicial:** O componente simbólico-heurístico `SudokuHeuristicAnalyzer` identificou a contradição imediatamente, classificando o tabuleiro como '(1) Sem Solução' antes de invocar o modelo neural.

Este caso valida a utilidade do filtro heurístico, que previne a execução de análises computacionalmente mais custosas em estados de jogo trivialmente insolúveis.

5.3 Resultados do Modelo 3: Recomendador de Heurísticas

Desempenho no Treinamento

O treinamento do modelo `LTNHeuristicRecommender4x4` apresentou uma dinâmica de aprendizado notável, resumida na Tabela 2. Após uma fase inicial de ajuste (Época 5), a acurácia do modelo saltou para **80.65%** e se manteve estável nesse patamar até o final do treinamento. Este platô sugere que o modelo rapidamente aprendeu a distinguir as classes mais proeminentes, mas pode requerer um dataset mais diverso ou uma arquitetura mais complexa para resolver os casos de fronteira mais ambíguos. A perda continuou a diminuir gradualmente, indicando um refinamento contínuo dos pesos do modelo.

Tabela 2: Progresso de Perda e Acurácia durante o Treinamento do Modelo 3.

Época	Perda Média	Acurácia
5	1.3355	0.00%
10	0.9055	80.65%
20	0.6839	80.65%
30	0.6446	80.65%

Exemplo de Inferência

O modelo treinado foi testado em um tabuleiro não visto para avaliar sua capacidade de recomendação. Para o tabuleiro de teste apresentado, o modelo calculou as seguintes probabilidades para cada estratégia:

```
Naked Single:      3.38%
Hidden Single:    80.32%
Locked Candidate:  7.29%
Backtracking:      9.01%
```

O resultado demonstra uma inferência de alta confiança. O modelo recomendou a heurística **Hidden Single** com uma probabilidade de 80.32%, um valor significativamente maior que as demais opções. Isso indica que o modelo não apenas realizou uma classificação, mas também desenvolveu uma forte convicção em sua decisão, tornando a recomendação prática e acionável.

6 Conclusão

Este trabalho demonstrou com sucesso a viabilidade e a eficácia da aplicação de Redes Lógicas Tensoriais (LTN) como uma ferramenta de IA neuro-simbólica para a análise de problemas de raciocínio lógico, utilizando o Sudoku 4x4 como um domínio de teste. Através de uma abordagem incremental, foram desenvolvidos três modelos que abordaram tarefas de complexidade cognitiva crescente, validando a capacidade do framework de integrar aprendizado de máquina com conhecimento simbólico.

Os resultados obtidos confirmam que a injeção de conhecimento de domínio através da lógica de primeira ordem confere vantagens significativas. A abordagem neuro-simbólica resultou em modelos que são, ao mesmo tempo, eficientes em dados – dado o forte viés indutivo fornecido pela estrutura lógica – e inerentemente mais interpretáveis do que abordagens puramente de aprendizado profundo. A capacidade de inspecionar a função de cada predicado e entender como as evidências são agregadas representa um passo importante em direção a uma IA mais transparente e confiável.

Apesar do sucesso, reconhece-se as limitações deste estudo. O domínio 4x4 é um ambiente controlado, e a performance do Modelo 3, embora alta, estabilizou-se em um platô que sugere que a distinção entre heurísticas mais complexas requer representações de features ainda mais ricas ou datasets mais extensivos.

7 Referências

Referências

[1] L. Serafini and A. d'Avila Garcez, "Logic Tensor Networks: Deep Learning and Logical Reasoning from Data and Knowledge,"*arXiv preprint arXiv:1606.04422*, 2016.

[2] L. Morra, A. Azzari, L. Bergamasco, M. Braga, L. Capogrosso, F. Delrio, G. Di Giacomo, S. Eiraudo, G. Ghione, R. Giudice, A. Koudounas, L. Piano, D. Rege Cambrin, M. Risso, M. Rondina, A. Russo, M. Russo, F. Taioli, L. Vaiani, C. Vercellino,
Designing Logic Tensor Networks for Visual Sudoku Puzzle Classification,
Proceedings of the 17th International Workshop on Neural-Symbolic Learning and Reasoning (NeSy 2023), CEUR Workshop Proceedings, Vol. 3432, Siena, Italy, July 2023.
Available at: `https://hdl.handle.net/11583/2978475`