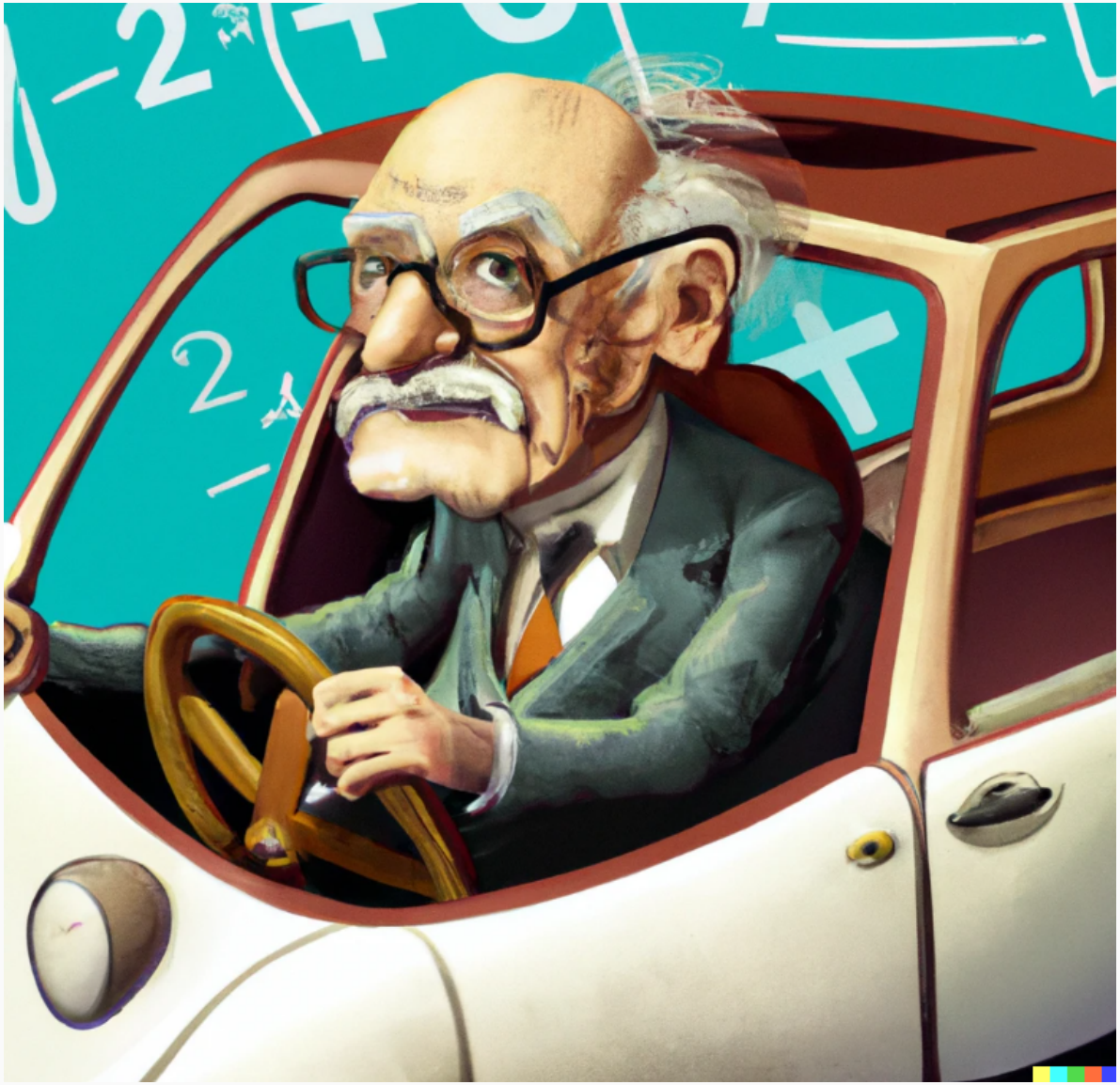
Połączenie sieci neuronowych i modeli symbolicznych w przetwarzaniu informacji

Mikołaj Małkiński
m.malkinski@mini.pw.edu.pl

5 października 2022

Szkoła Doktorska Politechniki Warszawskiej
Wydział Matematyki i Nauk Informacyjnych

Sieci neuronowe vs modele symboliczne



Rysunek 1: Rysunek wygenerowany przez DALL·E 2: https://openai.com/dall-e-2/ na podstawie tekstu: "An old mathematician driving a sports car, realistic picture".

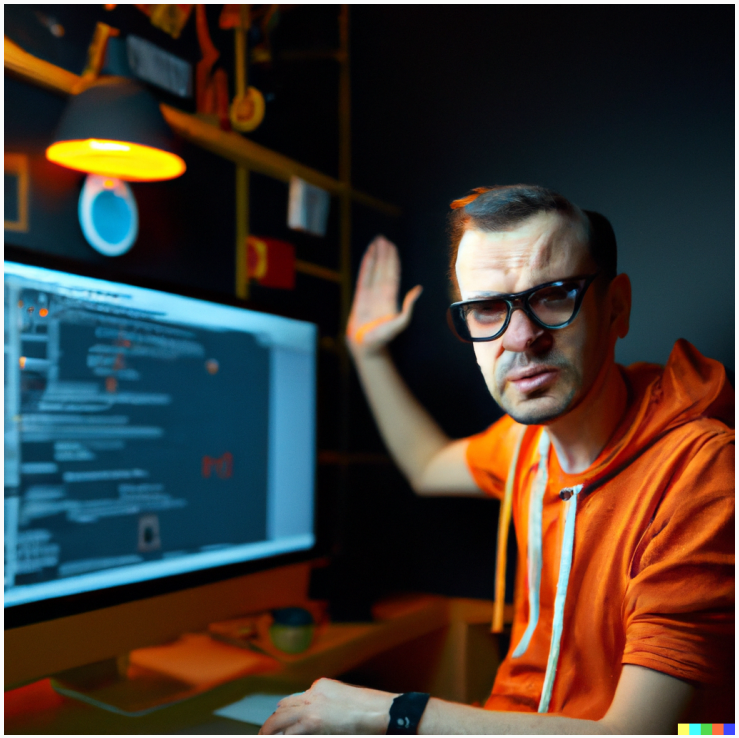
Teoria podwójnego procesu

Tabela 1: Teoria podwójnego procesu sugeruje, że ludzkie rozumowanie składa się z dwóch procesów – *System 1* i *System 2* (*niejawny* i *jawny*).

System 1	System 2
Szybki	Wolny
Automatyczny	Świadomy
Impulsy	Przemyślenia
Przyzwyczajenia	Planowanie
Poglądy	Rozumowanie

Sieci neuronowe – mocne strony

(a) "A frustrated student in a glass building during a discrete maths exam, digital art"



(b) "Angry programmer fixing an operating system bug, digital art"

Rysunek 2: Rysunki wygenerowane przez DALL·E 2: https://openai.com/dall-e-2/

Modele symboliczne – mocne strony

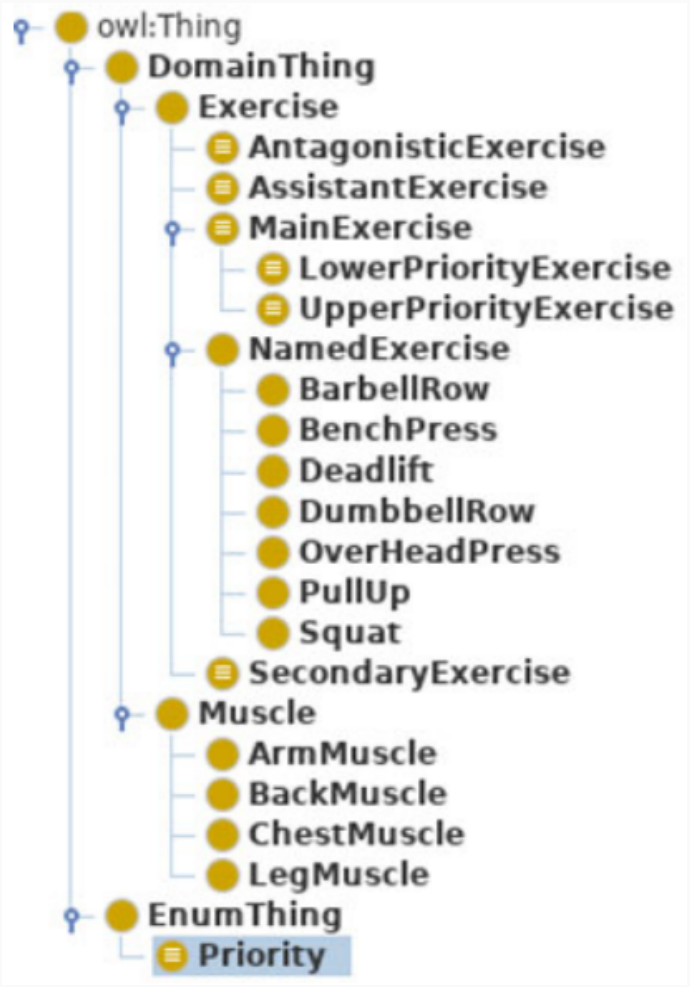
(a) "Oil pastel linux", wygenerowe przez https://openai.com/dall-e-2/

```python
def incr_list(l: list):
    """Return list with elements incremented by 1.
    >>> incr_list([1, 2, 3])
    [2, 3, 4]
    >>> incr_list([5, 3, 5, 2, 3, 3, 9, 0, 123])
    [6, 4, 6, 3, 4, 4, 10, 1, 124]
    """
    return [i + 1 for i in l]
```

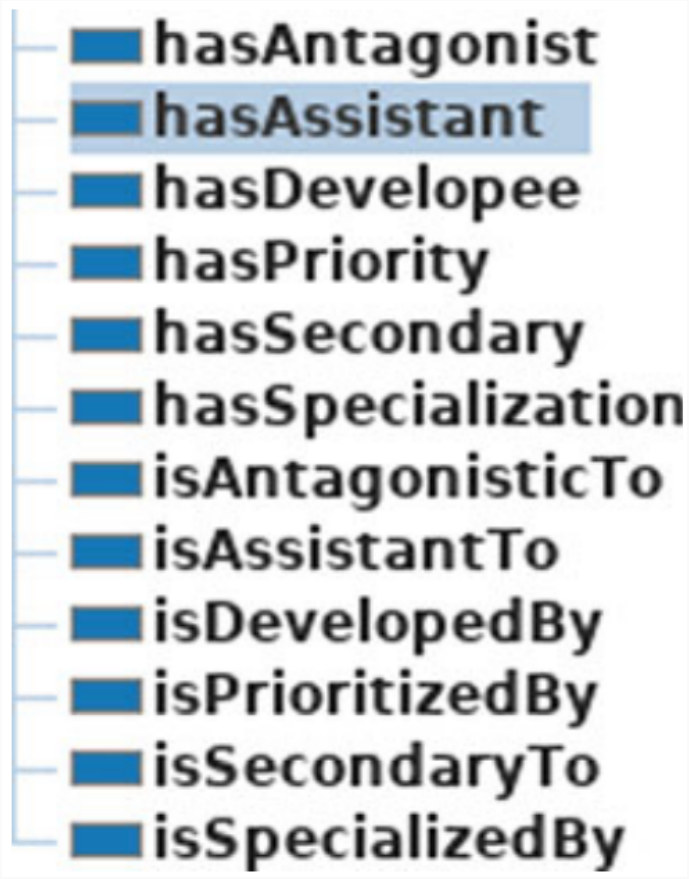
(b) Skrypt zwiększający wartości elementów listy o 1. Rysunek z [1].

Rysunek 3: Mocne strony szeroko-pojętych modeli symbolicznych.

Ontologia jako model symboliczny



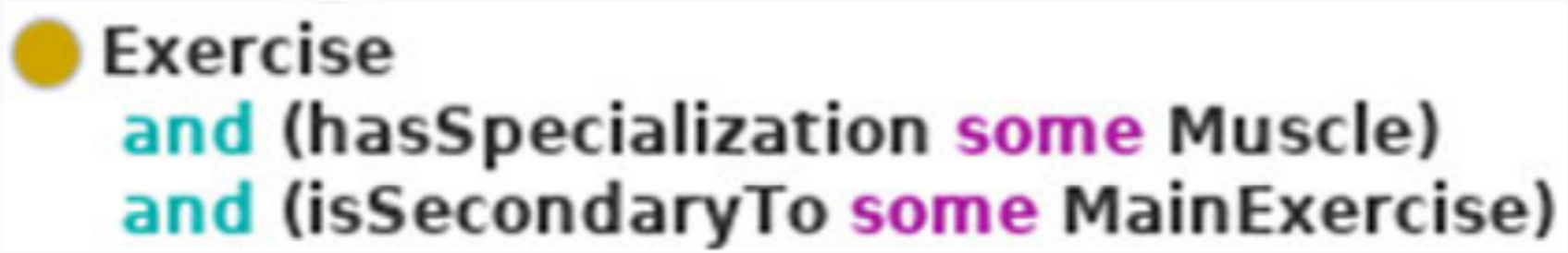
(a) Hierarchia klas



(b) Relacje

Rysunek 4: Ontologia ćwiczeń [4].

Ontologia jako model symboliczny



Rysunek 5: Definicja klasy `SecondaryExercise` [4].

Ontologia jako model symboliczny

```
PREFIX sf: <http://www.stayfit.oof/ontologies/exercise#>
SELECT ?exercise
WHERE {
    ?exercise sf:isAssistantTo sf:ClassicBenchPress
}
```

Rysunek 6: Zapytanie SPARQL wyszukujące pomocnicze ćwiczenia do określonego ćwiczenia [4].

Sieci neuronowe vs modele symboliczne

Tabela 2: Porównanie sieci neuronowych i modeli symbolicznych. Zaadaptowane z https://www.deepmind.com/blog/learning-explanatory-rules-from-noisy-data

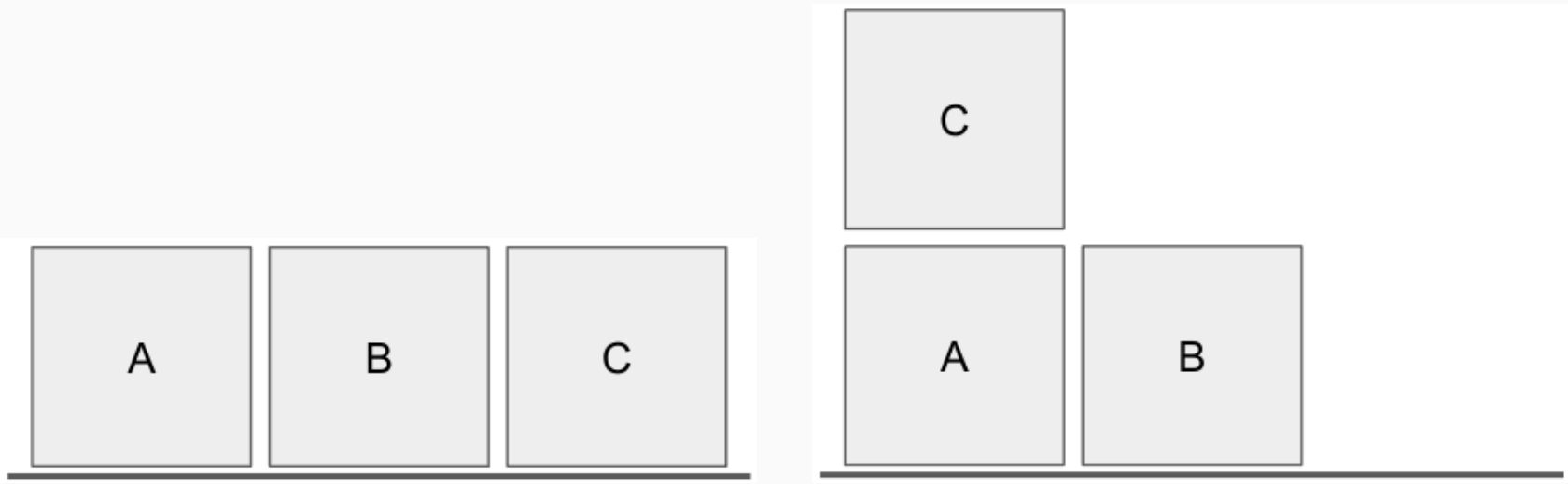
	Sieci neuronowe	Modele symboliczne
Odporność na szum	✓	×
Nieustrukturyzowane dane	✓	×
Wydajne uczenie	×	✓
Interpretowalność	×	✓

Świat kwadratów (ang. *Blocks world*) [5, 3]

Świat kwadratów

1. Świat składa się ze zbioru kwadratów umieszczonych w przestrzeni 2D.
2. Możemy przemieścić kwadrat jeśli nie znajduje się na nim żaden inny.
3. Zadanie polega na przekształceniu świata źródłowego w świat docelowy przemieszczając kwadraty.

Świat kwadratów



(a) Świat źródłowy

(b) Świat docelowy

Rysunek 7: Przykładowe zadanie ze świata kwadratów.

Świat kwadratów – techniczny opis

1. Świat składa się z dwóch rodzajów obiektów: kwadrat i ziemia.
2. Każdy obiekt ma 4 atrybuty: `world_id`, `object_id`, `coord_x`, `coord_y`.
3. Ziemia ma stały koordynat: $(0, 0)$.
4. Obiekt leżący bezpośrednio na ziemi ma $\texttt{object\_id} = \texttt{coord\_x}$.
5. Podstawowe reguły można sformułować porównując numeryczne atrybuty par obiektów. Przykładowo dla koordynatu x:

$$\texttt{Left}(u, v) := u.\texttt{coord\_x} < v.\texttt{coord\_x}$$

$$\texttt{SameX}(u, v) := u.\texttt{coord\_x} = v.\texttt{coord\_x}$$

$$\texttt{Right}(u, v) := u.\texttt{coord\_x} > v.\texttt{coord\_x}$$

Świat kwadratów – techniczny opis

Na podstawie wiedzy eksperckiej możemy zdefiniować dodatkowe reguły, przykładowo:

$$\begin{aligned}
\texttt{IsGround}(u) &\leftarrow \forall_v\ \texttt{Above}(v, u) \\
\texttt{Clear}(u) &\leftarrow \forall_v\ \neg\left(\texttt{Above}(u, v) \wedge \texttt{SameX}(u, v)\right) \\
\texttt{Moveable}(u) &\leftarrow \neg\texttt{IsGround}(u) \wedge \texttt{Clear}(u) \\
\texttt{Placeable}(u) &\leftarrow \texttt{IsGround}(u) \vee \texttt{Clear}(u)
\end{aligned}$$

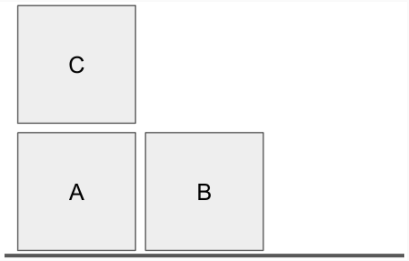
Świat kwadratów – techniczny opis

1. Zadanie składa się z dwóch światów: źródłowy i docelowy, każdy posiadający ziemię i m kwadratów.
2. Operujący agent wykonuje sekwencję akcji $\texttt{Move}(u, v)$, która przemieszcza obiekt u na obiekt v, gdzie $\texttt{Moveable}(u) \wedge \texttt{Placeable}(v)$.
3. Celem agenta jest przekształcenie świata źródłowego w docelowy.
4. Łącznie, dostępnych jest $m \times (m + 1)$ unikalnych akcji.

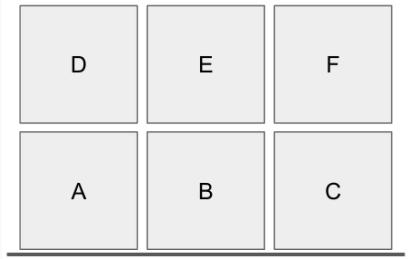
Świat kwadratów – główne wyzwania

Wyzwanie 1. Uniwersalność reguł, generalizacja do światów z większą liczbą kwadratów.

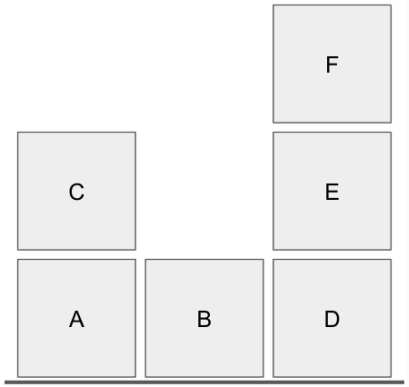
(a) Zbiór treningowy ($m = 3$), źródło.



(b) Zbiór testowy ($m = 3$), cel.



(c) Zbiór testowy ($m = 6$), źródło.



(d) Zbiór testowy ($m = 6$), cel.

Rysunek 8: Światy różniące się liczbą kwadratów.

Świat kwadratów – główne wyzwania

Wyzwanie 2. Przetwarzanie różnorodnej liczby obiektów i kwantyfikatorów.

Przykład: relacja przechodnia (ang. *transitivity*) wymaga spojrzenia na 3 obiekty:

$$r(u, w) \leftarrow \exists v \; r(u, v) \wedge r(v, w) \tag{1}$$

Świat kwadratów – główne wyzwania

Wyzwanie 3. Skalowalność do reguł o dużej złożoności.

Przykład: Reguła $\texttt{ShouldMove}(u)$ określająca czy dany obiekt powinien zostać przemieszczony:

$$\begin{aligned}
\texttt{SameXAbove}(u,v) &\leftarrow \texttt{SameWorldID}(u,v) \wedge \texttt{SameX}(u,v) \wedge \texttt{Above}(u,v) \\
\texttt{Match}(u,v) &\leftarrow \neg\texttt{SameWorldID}(u,v) \wedge \texttt{SameID}(u,v) \\
&\qquad \wedge \texttt{SameX}(u,v) \wedge \texttt{SameY}(u,v) \\
\texttt{Matched}(u) &\leftarrow \exists_v\, \texttt{Match}(u,v) \\
\texttt{UnmatchedBelow}(u) &\leftarrow \exists_v\, \texttt{SameXAbove}(u,v) \wedge \neg\texttt{Matched}(v) \\
\texttt{InitialWorld}(u) &\leftarrow \forall_v\, \neg\texttt{SmallerWorldID}(v,u) \\
\texttt{ShouldMove}(u) &\leftarrow \texttt{InitialWorld}(u) \wedge \texttt{Moveable}(u) \\
&\qquad \wedge \texttt{UnmatchedBelow}(u)
\end{aligned}$$

Świat kwadratów – główne wyzwania

Wyzwanie 4. Minimalizacja wiedzy eksperckiej, formułowanie reguł na podstawie surowych danych.

Świat kwadratów – główne wyzwania

Podsumowanie:

1. Uniwersalność reguł, generalizacja do światów z większą liczbą kwadratów.
2. Przetwarzanie różnorodnej liczby obiektów i kwantyfikatorów.
3. Skalowalność do reguł o dużej złożoności.
4. Minimalizacja wiedzy eksperckiej, formułowanie reguł na podstawie surowych danych.

Neural Logic Machines (NLM) [2]

NLM – wejście / wyjście

1. **Wejście**: Zbiór bazowych predykatów utwierdzonych na zbiorze obiektów (przesłanki). Porównanie numerycznych atrybutów wszystkich par obiektów.
2. **Akcja**: Sekwencyjne stosowanie reguł logiki pierwszego rzędu (elementy zbioru w połączeniu z kwantyfikatorami)
3. **Wyjście**: Wnioski o atrybutach obiektów. MLP + `Softmax` zastosowane do wyjściowych predykatów wszystkich par obiektów.
4. **Metoda treningu**: uczenie ze wzmocnieniem (ang. *Reinforcement Learning*).

Przykład: Na postawie przesłanek `IsGround(u)` i `Clear(u)`, NLM jest w stanie wnioskować wartość `IsMoveable(u)`.

NLM – reprezentacja przesłanek

- Zbiór obiektów $\mathcal{U} = \{u_1, u_2, \ldots, u_m\}$
- Predykat $p(x_1, x_2, \ldots, x_r)$ o krotności r (ang. *arity*)
- Predykat p utwierdzony na zbiorze obiektów $\mathcal{U}$ daje tensor $p^{\mathcal{U}}$ o wymiarach $[m^{\underline{r}}]$
- $[m^{\underline{r}}] := [m, m-1, m-2, \ldots, m-r+1]$
- Każdy element $p^{\mathcal{U}}(u_{i_1}, u_{i_2}, \ldots, u_{i_r})$ mówi czy p jest Prawdą dla obiektów $x_1 = u_{i_1}, x_2 = u_{i_2}, \ldots, x_r = u_{i_r}$
- Utwierdzone obiekty x_i wzajemnie się wykluczają: $i_j \neq i_k$ dla wszystkich j i k

NLM – reprezentacja przesłanek

Niech $C^{(r)}$ będzie liczbą predykatów o krotności r. Tensory predykatów o tej samej krotności są łączone tworząc większy tensor o wymiarach:

$$[m^{\underline{r}}, C^{(r)}] := [m, m-1, m-2, \ldots, m-r+1, C^{(r)}]$$

Przykładowe wymiary tensorów grup predykatów:

- $[m, C^{(1)}]$ – jednoargumentowe (atrybuty obiektów)
- $[m, m-1, C^{(2)}]$ – binarne (relacje pomiędzy parami obiektów)
- $[m, m-1, m-2, C^{(3)}]$ – trójkowe (relacje pomiędzy trójkami)

NLM – reprezentacja przesłanek

Przykład 1:

- Obiekty: $\mathcal{U} = \{u, v, w\}$ $(m = 3)$
- Predykaty jednoargumentowe: $\{\texttt{IsGround}(x), \texttt{Clear}(x)\}$ $(r = 1,\ C^{(1)} = 2)$

Tensor grupy predykatów o wymiarach $[m, C^{(1)}] = [3, 2]$:

$$\begin{bmatrix} \texttt{IsGround}(u) & \texttt{Clear}(u) \\ \texttt{IsGround}(v) & \texttt{Clear}(v) \\ \texttt{IsGround}(w) & \texttt{Clear}(w) \end{bmatrix}$$

NLM – reprezentacja przesłanek

Przykład 2:

- Obiekty: $\mathcal{U} = \{u, v, w\}$ $(m = 3)$
- Predykaty binarne: $\{\texttt{Above}(x, y)\}$ $(r = 2,\ C^{(2)} = 1)$

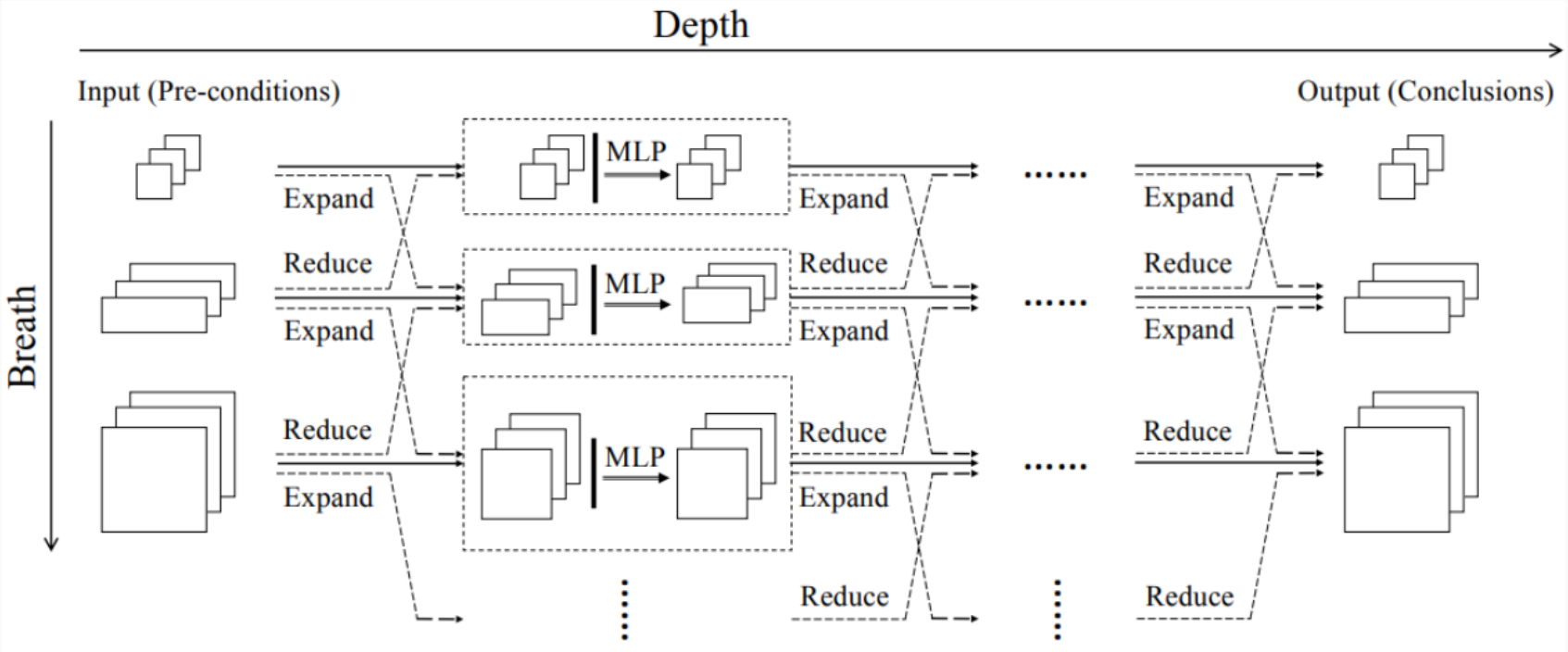
Tensor grupy predykatów o wymiarach $[m, m-1, C^{(2)}] = [3, 2, 1]$ (ostatni wymiar został spłaszczony dla czytelności):

$$\begin{bmatrix} \texttt{Above}(u, v) & \texttt{Above}(v, u) \\ \texttt{Above}(u, w) & \texttt{Above}(w, u) \\ \texttt{Above}(v, w) & \texttt{Above}(w, v) \end{bmatrix}$$

NLM – architektura

1. Szerokość B (ang. *breadth*) – hiperparametr określający maksymalną krotność predykatów w NLM.
2. Głębokość D (ang. *depth*) – liczba warstw modelu.
3. Każda warstwa ma $B + 1$ modułów obliczeniowych, operujących na predykatach o krotnościach $r \in [0, B]$.
4. Tensory przybierają wartości $\in [0, 1]$ – probabilistyczna interpretacja reguł.

NLM – architektura



Rysunek 9: Wysoko-poziomowa architektura NLM [2], $B = 2$.

NLM – architektura

Rysunek 10: Implementacja reguł logicznych jako moduły neuronowe [2]: a) Moduł neuronowy odpowiadający operatorom binarnym (`AND`, `OR`, `NOT`); b) Kwantyfikatory łączące predykaty o różnych krotnościach.

NLM – przetwarzanie informacji

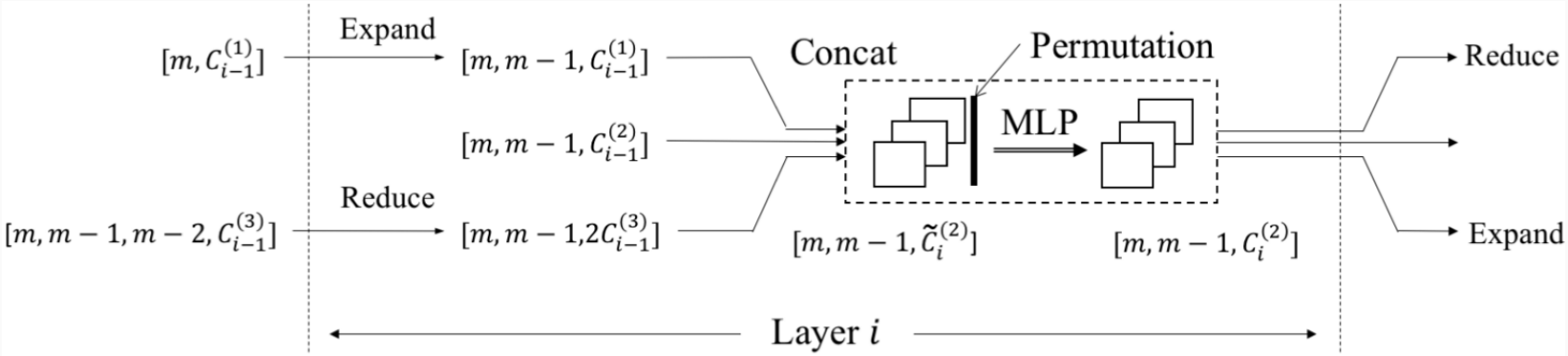
- $\mathcal{O}_i = \left\{ O_i^{(0)}, O_i^{(1)}, \ldots, O_i^{(B)} \right\}$ – wyjście warstwy i.
- $O_i^{(r)}$ – wyjście modułu o krotności r w warstwie i.
- $\mathcal{O}_0 = \left\{ O_0^{(0)}, O_0^{(1)}, \ldots, O_0^{(B)} \right\}$ – bazowe predykaty (przesłanki).
- $\mathcal{O}_D = \left\{ O_D^{(0)}, O_D^{(1)}, \ldots, O_D^{(B)} \right\}$ – wyjściowe predykaty (wnioski).
- Warstwy od 1 do D operują sekwencyjnie przyjmując $\mathcal{O}_{i-1}$ jako wejście i zwracając $\mathcal{O}_i$ jako wyjście.
- Moduły obliczeniowe operują równolegle.

NLM – moduł przetwarzania informacji

Rozważmy moduł r w warstwie i, aby pokazać jak obliczyć $O_i^{(r)}$.
Proces składa się z dwóch operacji:

- przetwarzanie między-modułowe
- przetwarzanie wewnątrz-modułowe

NLM – moduł przetwarzania informacji



Rysunek 11: NLM – moduł $r = 2$ w warstwie i. Rysunek z [2].

NLM – przetwarzanie między-modułowe

Cel: połączyć informację z predykatów o różnej krotności.

Przykłady motywacyjne:

1. $\texttt{Moveable}(x) \leftarrow \neg\texttt{IsGround}(x) \wedge \texttt{Clear}(x)$
2. $\texttt{ValidMove}(x, y) \leftarrow \texttt{Moveable}(x) \wedge \texttt{Placeable}(y)$
3. $\texttt{Clear}(x) \leftarrow \forall_y \neg\texttt{On}(y, x)$

NLM – przetwarzanie między-modułowe

Pod uwage brane są wyjścia z poprzedniej warstwy $i-1$ z bezpośrednio sąsiadujących modułów ($r-1$, r i $r+1$). Początkowo, ich wymiary to:

- $O_{i-1}^{(r-1)}$: $[m^{\underline{r-1}}, C_{i-1}^{(r-1)}]$
- $O_{i-1}^{(r)}$: $[m^{\underline{r}}, C_{i-1}^{(r)}]$
- $O_{i-1}^{(r+1)}$: $[m^{\underline{r+1}}, C_{i-1}^{(r+1)}]$

NLM – przetwarzanie między-modułowe

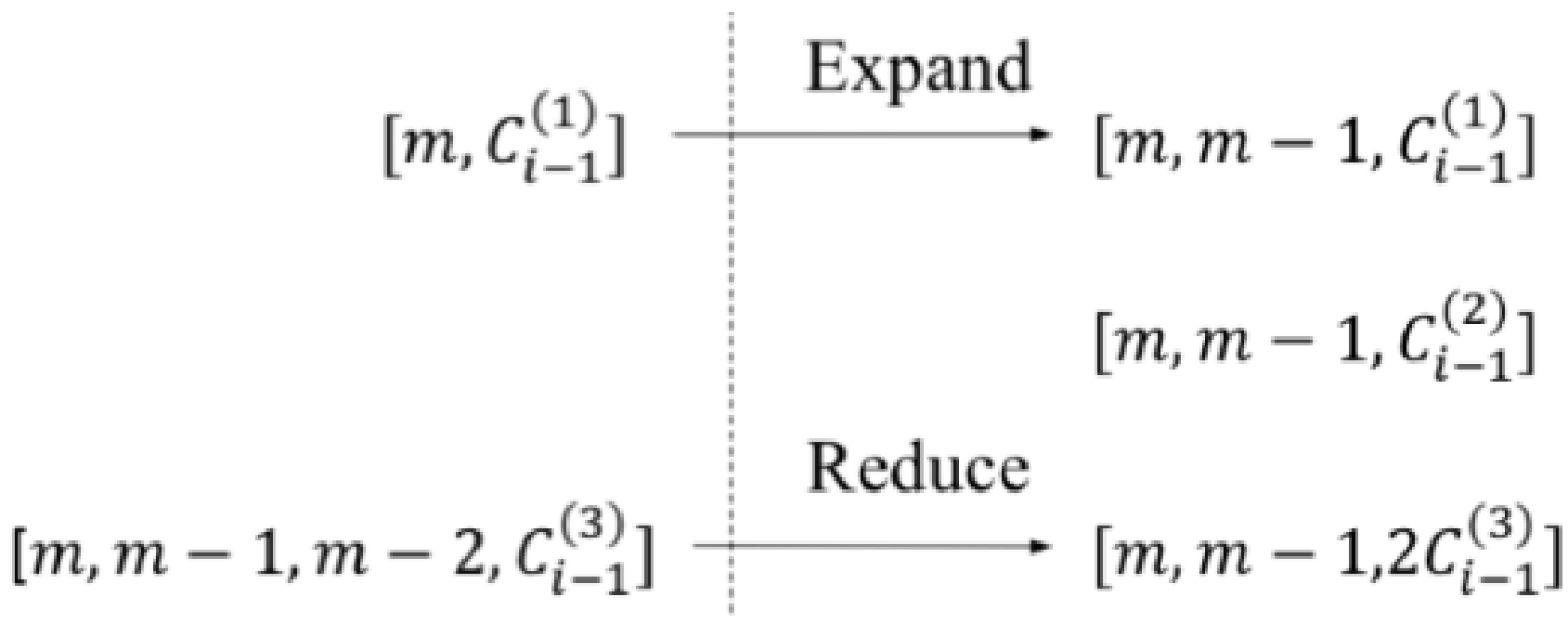
Rozszerzenie – tensor $O_{i-1}^{(r-1)}$ jest duplikowany $m - r + 1$ razy i łączony w nowym wymiarze dając tensor o wymiarach:

$$[m^{\underline{r-1}}, m - r + 1, C_{i-1}^{(r-1)}] = [m^{\underline{r}}, C_{i-1}^{(r-1)}]$$

Redukcja – tensor $O_{i-1}^{(r+1)}$ jest agregowany funkcjami Min i Max oraz łączony w ostatnim wymiarze dając tensor o wymiarach:

$$[m^{\underline{r}}, 2C_{i-1}^{(r+1)}]$$

NLM – przetwarzanie między-modułowe



Rysunek 12: NLM – wejście do warstwy i. Przykład dla $r = 2$. Zaadaptowane z [2].

NLM – przetwarzanie między-modułowe

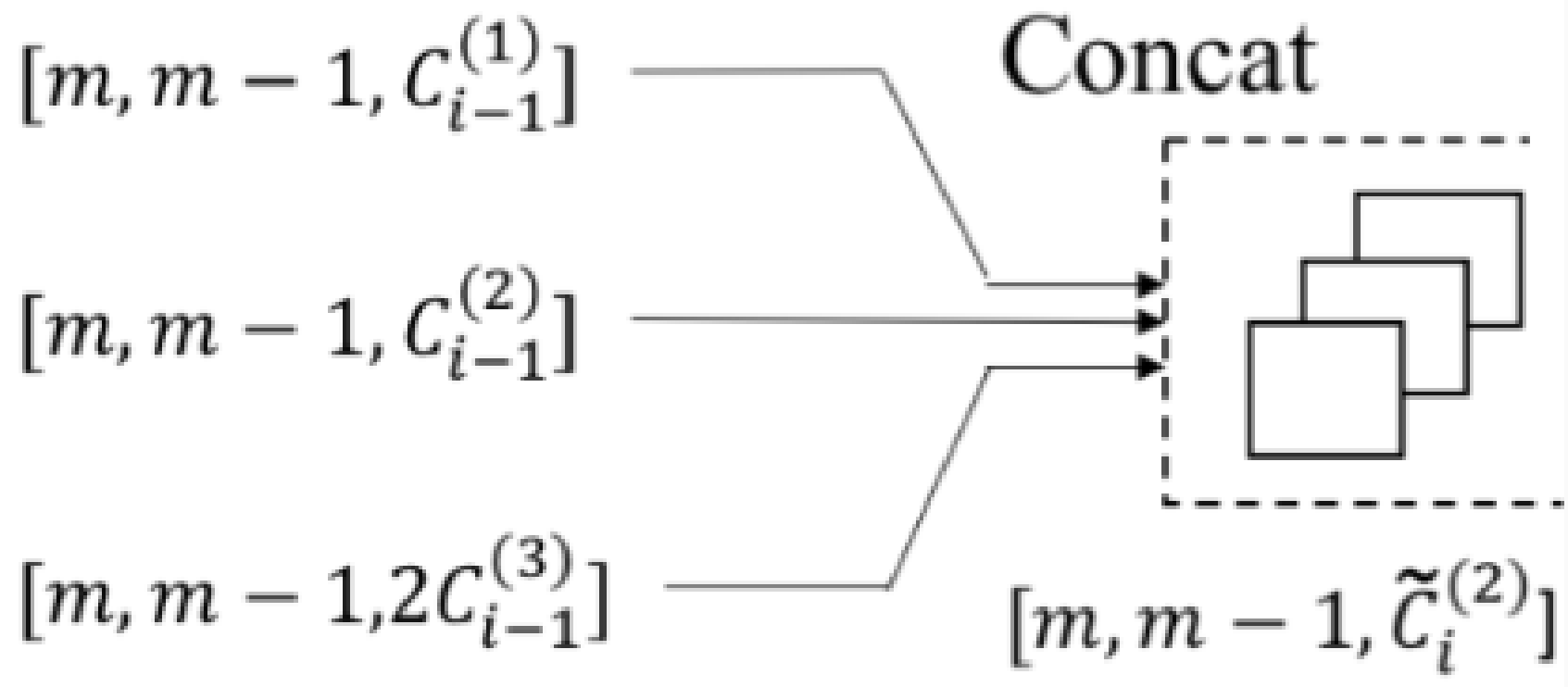
Następnie, tensory są łączone:

$$I_i^{(r)} = \text{Concat}\left(\text{Expand}\left(O_{i-1}^{(r-1)}\right), O_{i-1}^{(r)}, \text{Reduce}\left(O_{i-1}^{(r+1)}\right)\right)$$

dając tensor $I_i^{(r)}$ o wymiarach $[m^r, \widetilde{C}_{i-1}^{(r)}]$, gdzie:

$$\widetilde{C}_{i-1}^{(r)} := C_{i-1}^{(r-1)} + C_{i-1}^{(r)} + 2C_{i-1}^{(r+1)}$$

NLM – przetwarzanie między-modułowe



Rysunek 13: NLM – łączenie reprezentacji predykatów o sąsiadujących krotnościach po odpowiednim przekształceniu. Zaadaptowane z [2].

NLM – przetwarzanie wewnątrz-modułowe

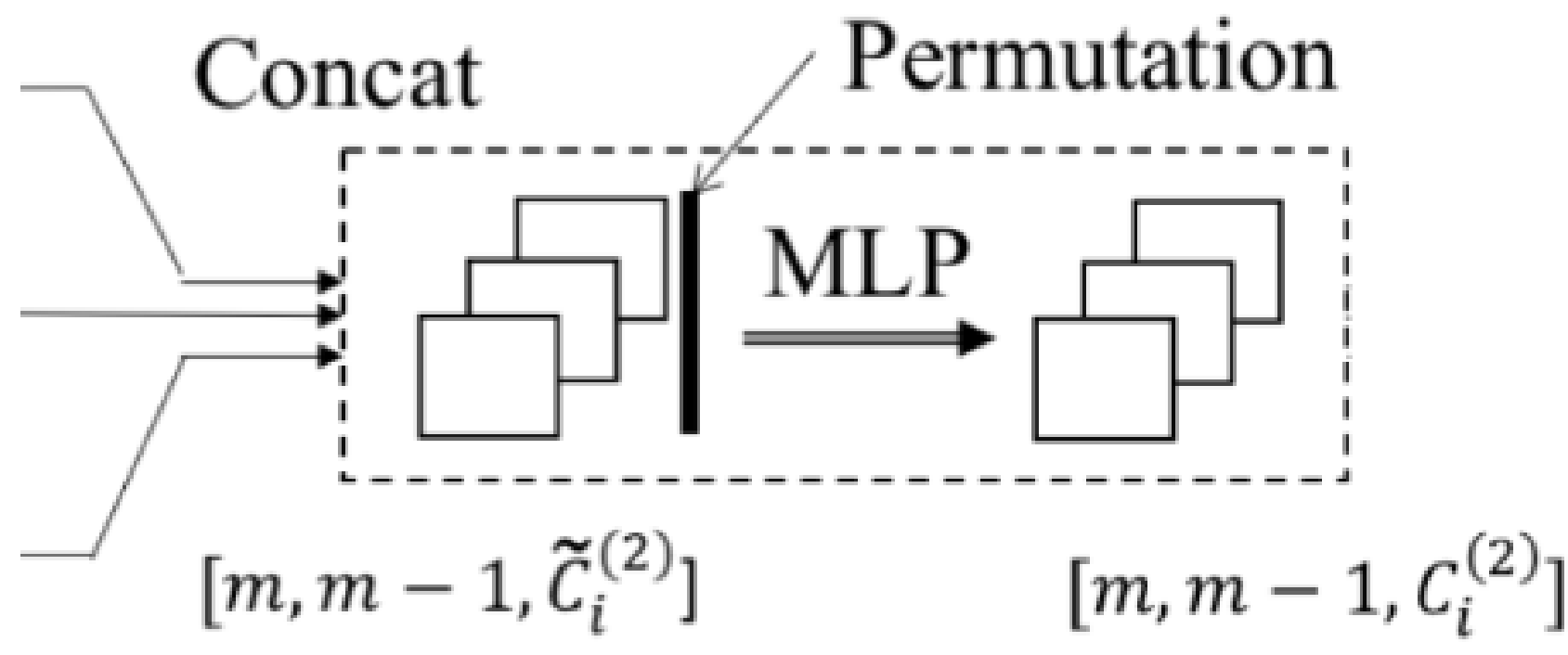
Przetwarzanie wewnątrz-modułowe przyjmuje na wejściu $I_i^{(r)}$. Tensor jest permutowany oraz przekształcany przez MLP z sigmoidalną funkcją aktywacji:

$$O_i^{(r)} = \sigma\left(\text{MLP}\left(\text{Permute}\left(I_i^{(r)}\right); \theta_i^{(r)}\right)\right)$$

gdzie $\theta_i^{(r)}$ to parametry modelu, wymiary $O_i^{(r)}$ to $[m^r, C_i^{(r)}]$, a $C_i^{(r)}$ to hiperparametr określający liczbę predykatów wyjściowych o krotności r w warstwie i.

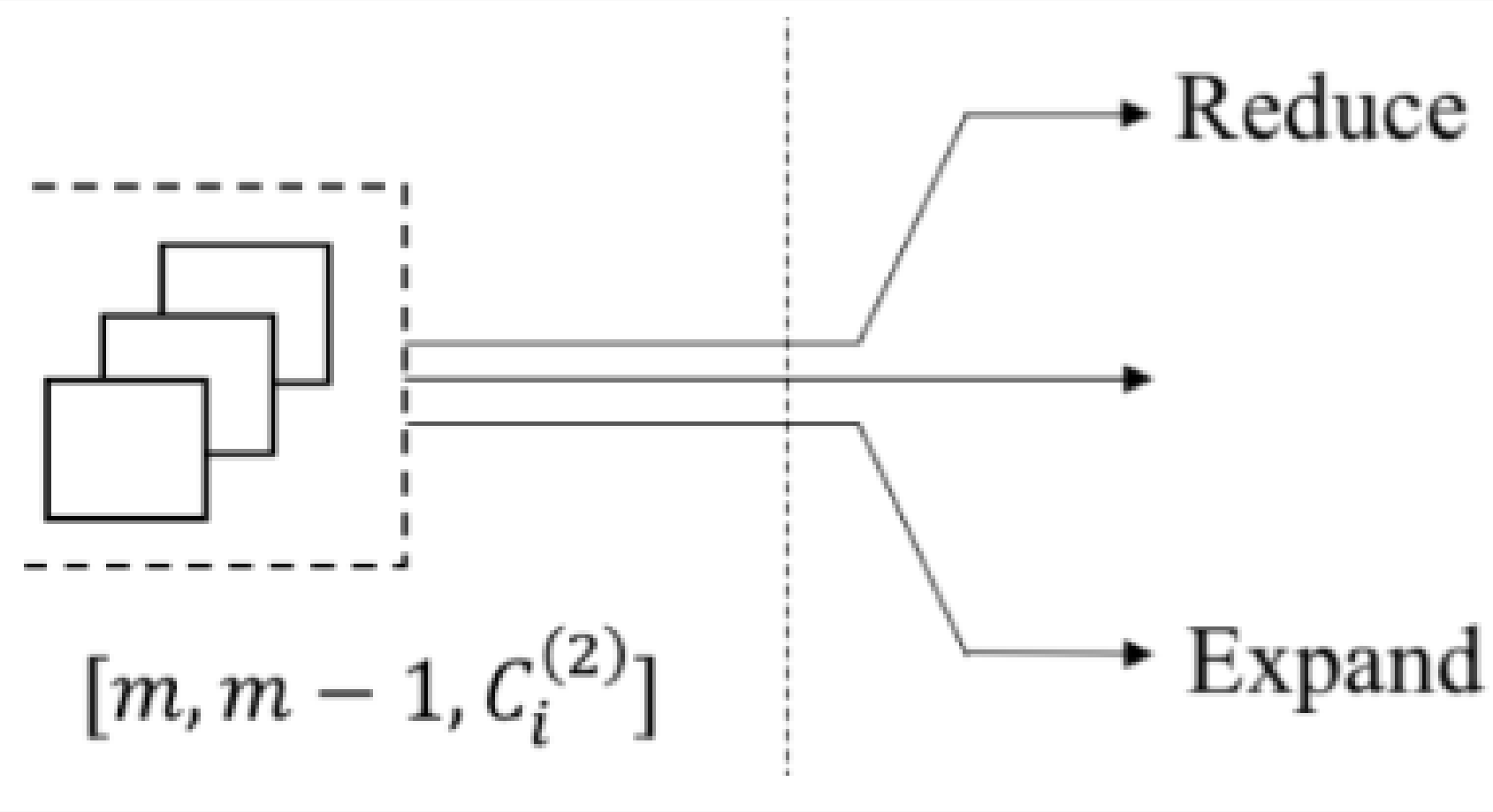
Moduł wyznacza nowy predykat na podstawie wszystkich permutacji utwierdzeń predykatów wejściowych.

NLM – przetwarzanie wewnątrz-modułowe



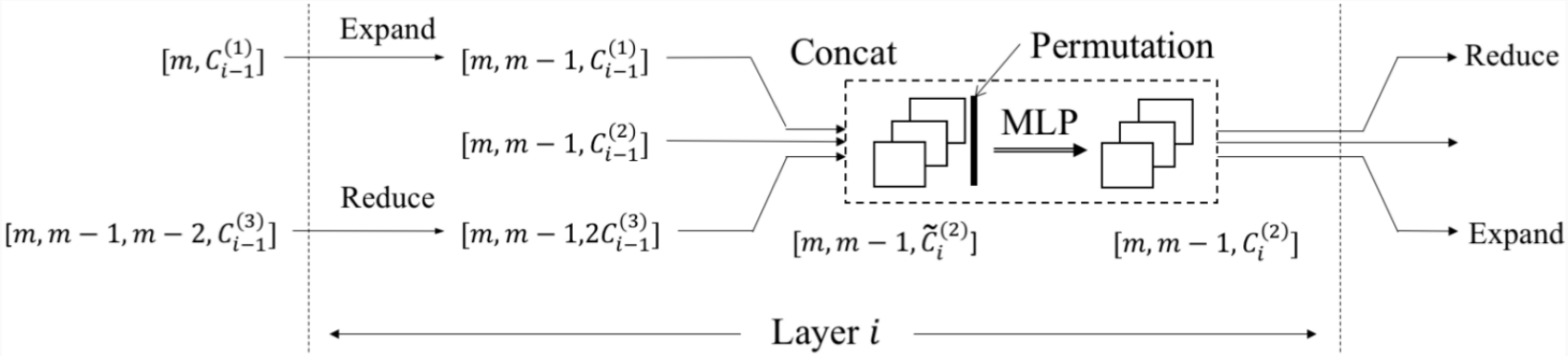
Rysunek 14: NLM – generowanie nowych predykatów o krotności r za pomocą MLP z sigmoidalną funkcją aktywacji. Zaadaptowane z [2].

NLM – przetwarzanie wewnątrz-modułowe



Rysunek 15: NLM – wygenerowane predykaty przekazywane są do kolejnej warstwy. Zaadaptowane z [2].

NLM – moduł przetwarzania informacji



Rysunek 16: NLM – moduł $r = 2$ w warstwie i. Rysunek z [2].

NLM – sortowanie

1. Zadanie polega na posortowaniu m-elementowej tablicy liczb całkowitych.
2. Wejście: liczbowe relacje na indeksach (czy $i < j$) oraz wartościach (czy $arr[i] < arr[j]$).
3. Algorytm może zamieniać pozycję dwóch elementów.
4. Możliwych jest $m \times (m - 1)$ akcji.

NLM – znajdywanie najkrótszej ścieżki w grafie

1. Zadanie polega na znalezieniu najkrótszej ścieżki w nieskierowanym grafie pomiędzy dwoma węzłami.
2. Wejście: macierz sąsiedztwa.
3. Algorytm kolejno wybiera następny węzeł na ścieżce.

NLM – wyniki

Task	MemNN		NLM (Ours)	
	$m = 10$	$m = 50$	$m = 10$	$m = 50$
`BlocksWorld`	0% / N/A	0% / N/A	100% / 12	100% / 84
`Sorting`	100% / 22	90% / 986.6	100% / 8	100% / 45
`Path`	45% / 13.3	12% / 42.7	100% / 4	100% / 4

Rysunek 17: Wyniki dwóch modeli w zadaniach uporządkowania świata kwadratów (`BlocksWorld`), sortowania (`Sorting`) i znajdywania najkrótszej ścieżki (`Path`). Oba modele były trenowane na $m \leq 12$. Użyte metryki to skuteczność łamana na średnią liczbę kroków. Tabela z [2].

NLM – kolejne kroki

1. Głębokość modelu jest hiperparametrem – czy możemy automatycznie znaleźć optymalną złożoność predykatów w zależności od problemu?
2. Algorytm wymagał zaawansowanych metod optymalizacji (połączeń rezydualnych, *curriculum learning*).
3. Eksperymenty głównie polegały na danych symbolicznych.
4. Predykaty stworzone przez model nie są interpretowalne.

“A debate between connectionists and formalists, oil painting”
https://openai.com/dall-e-2/

Bibliografia i


Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al.
Evaluating large language models trained on code.
arXiv preprint arXiv:2107.03374, 2021.


Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou.
Neural logic machines.
In *International Conference on Learning Representations*, 2019.


Naresh Gupta and Dana S Nau.
On the complexity of blocks-world planning.
Artificial Intelligence, 56(2-3):223–254, 1992.

Bibliografia ii

Givi-Giorgi Mamatsashvili, Konrad Ponichtera, Mikołaj Małkiński, Maria Ganzha, and Marcin Paprzycki.
Semantic-based system for exercise programming and dietary advice.
In *Advances in Bioinformatics, Multimedia, and Electronics Circuits and Signals*, pages 105–120. Springer, 2020.

Nils J Nilsson.
***Principles of artificial intelligence*.**
Springer Science & Business Media, 1982.