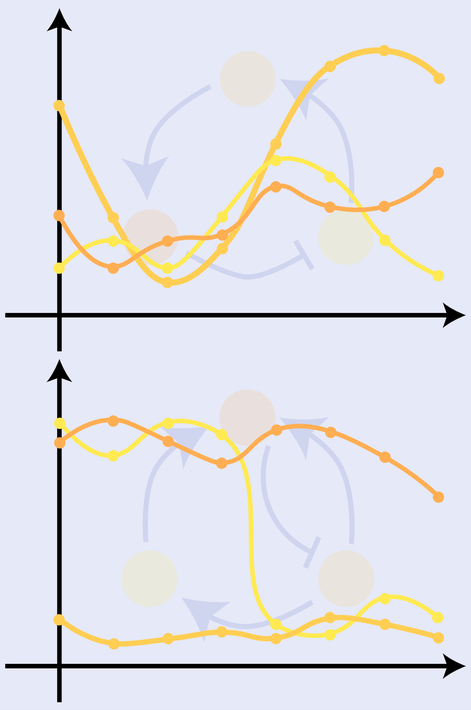
Learning Logic Programs Using Neural Networks by Exploiting Symbolic Invariance

Yin Jun Phua[1, 2], Katsumi Inoue[1, 2]

[1] The Graduate University for Advanced Studies (Sokendai), Japan [2] National Institute of Informatics, Japan

WHAT



WHY

$$p_t \leftarrow r_{t-1}$$
$$q_t \leftarrow p_{t-1}$$
$$r_t \leftarrow \neg q_{t-1}$$

$$p_t \leftarrow q_{t-1} \wedge r_{t-1}$$
$$q_t \leftarrow r_{t-1}$$
$$r_t \leftarrow \neg p_{t-1}$$

Learning from Interpretation Transition

- **Learning from Interpretation Transition (LFIT)** learns explainable rules by observing the state transitions of a dynamical system.
- Given a set of transitions, the LFIT algorithm **outputs a normal logic program** (NLP) which completely represents the given transitions.
- The **NLP gives explanation as to why** a variable is activated in a particular timestep based on the state of the previous timestep.
- LFIT has applications in robotics, biology and many other fields.
- The LFIT algorithm has mainly been implemented using the symbolic method and the neural network method.

$$\begin{array}{c} \\ a \\ b \end{array}\!\!\begin{array}{|cccc|cccc|} \{a\} & \{b\} & \{\neg a\} & \{\neg b\} & \{a,b\} & \{\neg a,b\} & \{a,\neg b\} & \{\neg a,\neg b\} \\ \hline 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{array}$$

For a system with 2 variables, the logic program matrix can be split up into something like the left. Each part represents the **output nodes that are getting reused**. The reuse is across **variable heads** and **body length**. So for example, output node o can represent $a_t \leftarrow a_{t-1}, a_t \leftarrow a_{t-1} \wedge b_{t-1}, b_t \leftarrow a_{t-1}, b_t \leftarrow a_{t-1} \wedge b_{t-1}$. However by doing this there will be **imbalance in the labels**. There is also the case where **all the output nodes are labeled as o**. Therefore rules that are **subsumed are substituted with a non-zero μ** to aid in training.

$$\begin{array}{c} \\ a \\ b \end{array}\!\!\begin{array}{|cccc|cccc|} \{a\} & \{b\} & \{\neg a\} & \{\neg b\} & \{a,b\} & \{\neg a,b\} & \{a,\neg b\} & \{\neg a,\neg b\} \\ \hline 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0\mu & 0 & 0\mu & 0 \end{array}$$

∂LFIT+

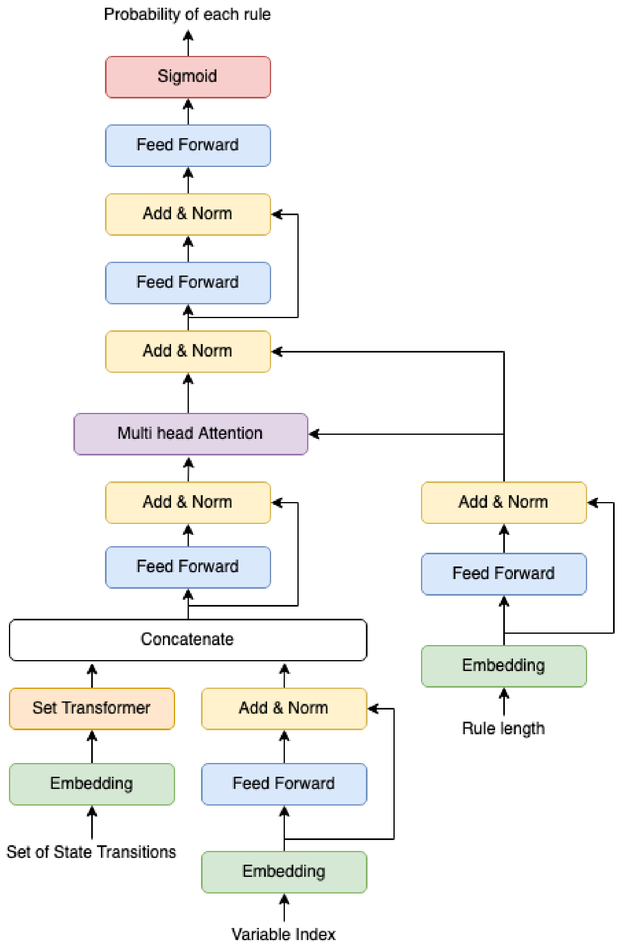
- ∂LFIT had issues with **inefficient use of training data**, **missing some of the input data** and **poor scalability**.
- To solve the inefficient use of training data, we use **set transformer** for the input sequence order invariance.
- To address the scalability issue, we **reuse the same output node** to predict several different rules. An output node can predict **rules with different head**, or **rules with different lengths**.
- We also apply **weights to the labels** to counteract several labels dominating the dataset.
- We also apply **label smoothing by subsumption** to overcome labels being too sparse.

Symbolic Invariance

- There are many **invariances or symmetries** in the symbolic world. For example substituting the literal a and b usually does not change the semantics.
- However in neural networks, **different permutations yield different meanings**, and thus different results.
- One invariance that we seek to exploit is the **ordering of the input sequence**. In LFIT, the input is a **set of state transitions**. The ordering of state transitions within the set itself does not convey any meaning to LFIT.
- ∂LFIT for example treats the input $\{(pq,r), (r,pq)\}$ as different to $\{(r,pq), (pq,r)\}$. Therefore **both permutations have to be supplied** as training data in order to train ∂LFIT.
- We employ the **set transformer** model in order to guarantee this invariance in ∂LFIT+.
- Set transformer uses the **attention mechanism** in order to guarantee that the ordering does not affect its output.

Index for 3-variable System

$l = 1$	1: p.	$l = 3$	19: $p \wedge q \wedge r$.	25: $p \wedge \neg q \wedge \neg r$.
	2: q.		20: $\neg p \wedge q \wedge r$.	26: $\neg p \wedge \neg q \wedge \neg r$.
	3: r.		21: $p \wedge \neg q \wedge r$.	
	4: $\neg p$.		22: $\neg p \wedge \neg q \wedge r$.	
	5: $\neg q$.		23: $p \wedge q \wedge \neg r$.	
	6: $\neg r$.		24: $\neg p \wedge q \wedge \neg r$.	
$l = 2$	7: $p \wedge q$.		13: $p \wedge \neg q$.	
	8: $p \wedge r$.		14: $p \wedge \neg r$.	
	9: $q \wedge r$.		15: $q \wedge \neg r$.	
	10: $\neg p \wedge q$.		16: $\neg p \wedge \neg q$.	
	11: $\neg p \wedge r$.		17: $\neg p \wedge \neg r$.	
	12: $\neg q \wedge r$.		18: $\neg q \wedge \neg r$.	

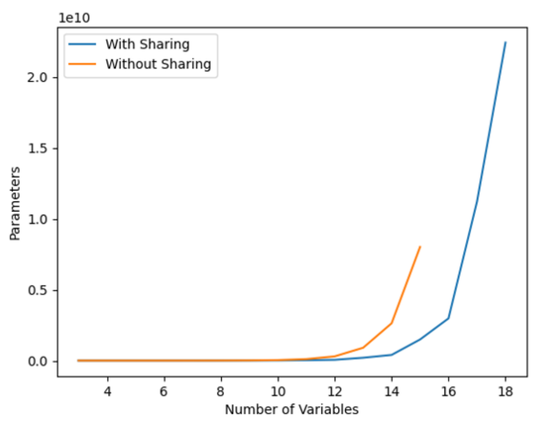


- The model architecture is pictured on the left.
- The model takes a **set of state transitions**, the **variable index** that will be the head of the predicted rules, and the **rule length** for the body of the rules as inputs.
- All the inputs are **embedded** into an embedding space.
- The output of the set transformer and the variable index embedding are **concatenated and provided as key and values** for the multihead attention. The rule length embedding is the **query** for the multihead attention.
- The output of the attention is then fed through several layers of feed forward network, and finally a **sigmoid activation** is applied to obtain the probability for each rule.
- The model performs **inference for each variable** in the system and for **all possible rule lengths**, all the outputs are then **assembled to produce a logic program**.

Results

Boolean Network	δLFIT	δLFIT+^{3}	δLFIT+^{5}	δLFIT+^{7}
3-node (a)	0.095	0.271	0.271	0.271
3-node (b)	0.054	0.188	0.208	0.208
Raf	0.253	0.188	0.208	0.208
5-node	0.142	-	0.278	0.325
7-node	-	-	-	0.223
WNT5A [22]	-	-	-	0.194

Boolean Network	δLFIT+^{5}	δLFIT+$^5_{\neg T}$
3-node (a)	0.271	0.313
3-node (b)	0.208	0.292
Raf	0.208	0.271
5-node	0.278	0.375



References

1. Inoue, K., Ribeiro, T., Sakama, C.: Learning from Interpretation Transition. Machine Learning 94(1), 51-79 (2014)
2. Ribeiro, T., Inoue, K.: Learning prime implicant conditions from interpretation transition. In: ILP 2015, pp. 108-125. Springer (2015)
3. Gentet, E., Tourret, S., Inoue, K.: Learning from interpretation transition using feed-forward neural network. In: Proceedings of ILP 2016, CEUR Proc. 1865. pp. 27-33 (2016)
4. Yin Jun Phua and Katsumi Inoue. Learning logic programs from noisy state transition data. In Inductive Logic Programming - 29th International Conference, ILP 2019 (2019)
5. Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer. CoRR, abs/1810.00825 (2018)