

IV-Applications

Neuro Symbolic AI

Logik für Erklärbare KI: Technische Einführung in das ENEXA Projekt

Maria Schnödt-Fuchs, Alex Goeßmann

Funded by the
European Union

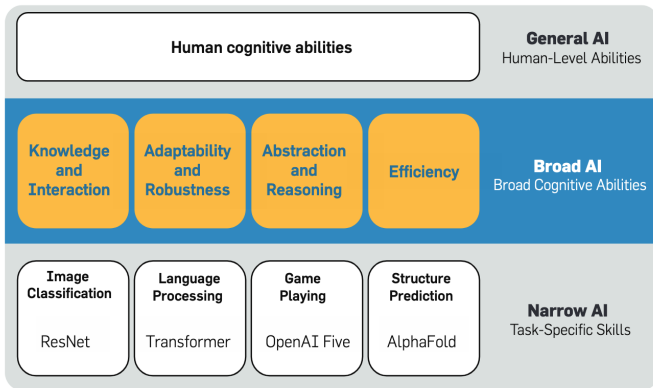


15.+16. July, 2024

Communications of the ACM, April 2022

"Europe's Opportunity for a Broad AI:

The most promising approach to a broad AI is a neuro-symbolic AI, that is, a bilateral AI that combines methods from symbolic and sub-symbolic AI."

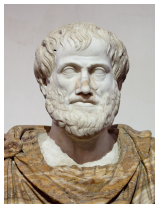


How to represent knowledge?

→ **Logical Syntax**

How to interpret knowledge?

→ **Logical Semantics**



Aristotle

Symbolic AI: Reasoning based on Logic

- ▶ Data and models represented by logical syntax
- ▶ Learning and inference based on logical semantics

Advantage: Model explainability in its purest form (ante-hoc)

Sub-symbolic AI: Computations in Neural Architectures

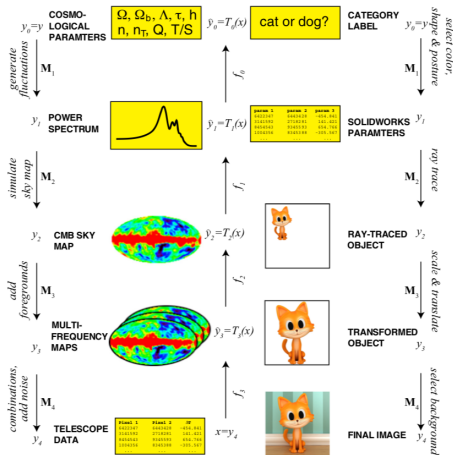
- ▶ Expressivity of Deep Networks: Effective representation of Data
- ▶ Differentiable parametrization

Problem: Black-box when not designed otherwise

Why deep and not shallow networks?

- ▶ **Physical explanations:** Deep neural networks appear naturally in use cases
- ▶ **Mathematical explanations:** Deep neural networks have astonishing approximation properties

Generation processes by deep layers



Lin, Tegmark, Rolnick: Why does deep and cheap learning work so well?
Journal of Statistical Physics, 2017

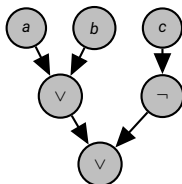
Combining the paradigms: Hierarchical structure of Logical Formulas

Logical formulas have an iterative decomposition structure into subformulas f_1, f_2 until they are at atomic level:

$$f = f_1 \circ f_2 \quad (1)$$

Example: Variables a or b implied by another variable c

$$f = (c \Rightarrow a \vee b) = (a \vee b \vee \neg c)$$



Challenge: Overparametrization by Logical Formulas

A system with d binary variables



has a number of states

$$\#\{x_1, \dots, x_n \in \{0, 1\}\} = 2^d.$$

Whereas the number of logical formulas is

$$2^{(2^d)}.$$

For $d = 10$ binary variables we have

$$2^{(2^{10})} > 10^{300},$$

whereas the number of atoms in the known universe is 10^{80} .

Two learning tasks

Given data we learn a Markov Logic Network by:

- ▶ **Structure Learning**: Find the formulas f to be activated.
- ▶ **Weight Estimation**: Find the optimal weights w^f to the formulas.

While Weight Estimation is efficiently solvable, Structure Learning faces

- ▶ enormous search spaces: $2^{(2^d)}$ formulas to d atoms
- ▶ computational demand to evaluate single formulas

We ease these problems by a tensor network decompositions of

- ▶ hypothesis formulas, by **formula selecting neurons**
- ▶ log-likelihood losses and its gradients

We take an ensemble perspective:

- ▶ Each formula has limited expressivity, thus we have to use collections
- ▶ We learn in a greedy way, that is choose best formula in each step

Learning an additional formula to a given distribution

Given a hypothesis set \mathcal{F} and a current distribution \mathbb{P} we want to add the best formula $f \in \mathcal{F}$. We approach this by

- ▶ Finding an efficient representation of the formulas as a tensor network
- ▶ Contraction with the gradient of the likelihood
- ▶ Search for the maximum likelihood ascent among the formulas

Trick: Selection Variables

Add argument to the formulas representing the selection choice among the hypothesis set \mathcal{F} . Corresponding random variables are called **selection variables** L .

Given a set of p formulas $\{f_l : l \in [p]\}$, we define the formula selecting map as

$$\mathcal{F} : \prod_{k \in [d]} [2] \times [p] \rightarrow [2]$$

defined for $l \in [p]$ by

$$\mathcal{F}(i_0, \dots, i_{d-1}, l) = f_l(i_0, \dots, i_{d-1}).$$

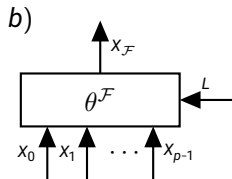
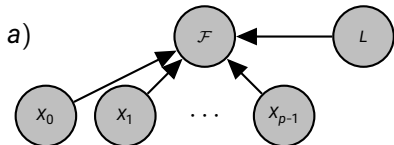
Given a set of p formulas $\{f_l : l \in [p]\}$, we define the formula selecting map as

$$\mathcal{F} : \bigtimes_{k \in [d]} [2] \times [p] \rightarrow [2]$$

defined for $l \in [p]$ by

$$\mathcal{F}(i_0, \dots, i_{d-1}, l) = f_l(i_0, \dots, i_{d-1}).$$

We depict formula selecting maps by



- a) Introduction of selection variables to the graphical model
- b) Tensor Core with selection variable stored in an incoming leg

We provide two building blocks by the

- ▶ Choice of connections (\sim support of the weights):

Variable selecting maps

- ▶ Choice of activations (\sim value of the weights):

Connective selecting maps

and combine both in a

- ▶ Symbolic neuron: Choice of a connective and variables passed into arguments
- ▶ Symbolic architecture: Collection of neurons with layerwise dependencies on each other

Definition (Variable selecting map)

Given a set of variables enumerated by $[p]$, we call the map

$$\mathcal{F}_V : \left(\prod_{l \in [p]} [2] \right) \times [p] \rightarrow [2] \quad (2)$$

defined coordinatewise by

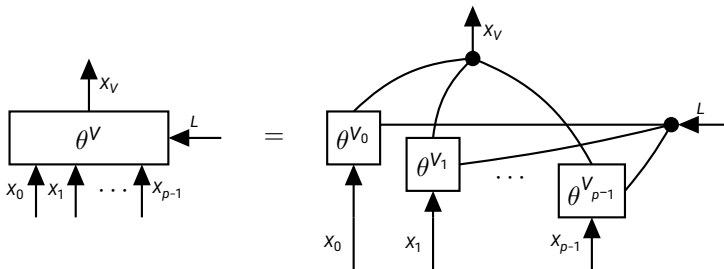
$$\mathcal{F}_V(i_0, \dots, i_{p-1}, l) = i_l \quad (3)$$

the corresponding **variable selecting map**.

The one-hot encoding of the variable selection map has a decomposition

$$\theta^V = \sum_{l \in [p]} \theta^{x_l} \otimes \mathbf{e}_l.$$

We capture this to define cores $\theta^{V_l} \in \mathbb{R}^2 \otimes \mathbb{R}^p \otimes \mathbb{R}^2$ and get a decomposition



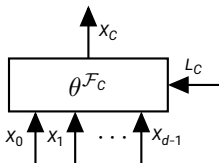
Definition (Connective selecting map)

Let $\{\circ_0, \dots, \circ_{p_C-1}\}$ be a set of connectives with d arguments. The associated **connective selecting map** is the map

$$\mathcal{F}_C : \prod_{k \in [d]} [2] \times [p_C] \rightarrow [2]$$

defined for each $l_C \in [p_C]$ by

$$\mathcal{F}_C(i_0, \dots, i_{d-1}, l_C) = \circ_{l_C}(i_0, \dots, i_{d-1}).$$



Definition (Symbolic neuron)

Given an order $n \in \mathbb{N}$ let there be a connective selector L_o selecting connectives of same order and let $\mathcal{F}_{V,0}, \dots, \mathcal{F}_{V,n-1}$ be a collection of variable selectors. The corresponding symbolic neuron is the map

$$\sigma : \left(\prod_{k \in [d]} [2] \right) \times [p_C] \times \left(\prod_{s \in [n]} [p_s] \right) \rightarrow [2]$$

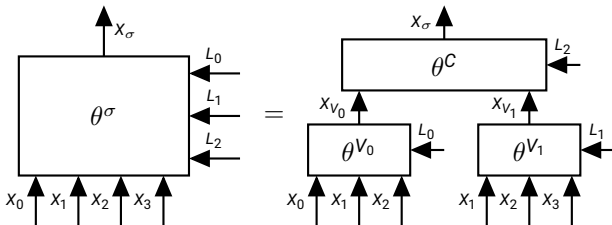
defined for $i_0, \dots, i_{d-1} \in \prod_{k \in [d]} [2]$, $l_C \in [p_C]$ and $l_0 \dots l_{n-1} \in \prod_{s \in [n]} [p_s]$ by

$$\begin{aligned} & \sigma(i_0, \dots, i_{d-1}, l_C, l_0 \dots l_{n-1}) \\ &= \mathcal{F}_C(\mathcal{F}_{V,0}(i_0, \dots, i_{d-1}, l_0), \dots, \mathcal{F}_{V,n-1}(i_0, \dots, i_{d-1}, l_{n-1}), l_C). \end{aligned}$$

Let us specify a neuron $\sigma = f_1 \circ_2 f_3$ by candidates

$$f_1 : [X_0, X_1, X_2], \quad \circ_2 : [\wedge, \vee, \Rightarrow], \quad f_3 : [X_1, X_2, X_3].$$

We can decompose the encoding of a symbolic neuron by



Where the variable selector cores θ^{V_0} and θ^{V_1} can be further decomposed into leg cores.

Architectures are collections of neurons

$$\mathcal{H} = \{\sigma_0, \dots, \sigma_{n-1}\}$$

which can depend on each other, i.e. a neuron can take another neuron as argument.

We require **acyclicity** of the possible dependencies

- ▶ for well-definedness of the resulting formulas (avoid circular dependencies)
- ▶ resulting in a feed-forward architecture of neurons based on the dependency order

In this way, we can parametrize formulas with arbitrary complexity.

We extend the nested lists encoding $S(f)$ defined for propositional formulas f to also encode

- ▶ Logical neurons σ
- ▶ Formula selecting maps \mathcal{H}

Strategy: **Choices captured in further nested lists**

- ▶ Replace connectives by list of candidate connectives
- ▶ Replace direct subformula specifications by lists of variables (e.g. atomic variables or other neurons)

Example: Wet street

Following the wet street example, we can define a neuron by

$$S(\sigma) = [["imp", "eq"], ["Wet", "Sprinkler"], ["Street"]]$$

from which the formulas

["imp", "Wet", "Street"]

["eq", "Wet", "Street"]

["imp", "Sprinkler", "Street"]

["eq", "Sprinkler", "Street"]

can be chosen. Combining this neuron with further neurons, e.g. by the architecture

$$S(\mathcal{H}) = \{ \text{"neur1": } [["imp", "eq"], ["neur2"], ["Street"]], \\ \text{"neur2": } [["Inot", "id"], ["Wet", "Sprinkler"]] \}$$

the expressivity increases. In this case, the further neuron provides the flexibility of the first atoms to be replaced by its negation.

Having a probability tensor \mathbb{P} as a current model, we want to find $f \in \mathcal{F}$ solving

$$\operatorname{argmax}_{f \in \mathcal{F}} \max_{w \in \mathbb{R}} \mathcal{L}_D (\mathbb{P} + w \cdot f) \ .$$

Intractability of direct optimization

The likelihood involves partition functions and is not linear in f . Therefore we cannot make use of the tensor network representation of \mathcal{F} .

Extending a distribution \mathbb{P} by $f \in \mathcal{F}$ we have

$$\begin{aligned}\frac{\partial}{\partial \mathbf{w}} \mathcal{L}_D (\mathbb{P} + \mathbf{w} \cdot f) \big|_{\mathbf{w}=0} &= \mathbb{E}_{\mathbb{P}^D} [f] - \mathbb{E}_{\mathbb{P}} [f] \\ &= \mathcal{C} \left(\{\mathbb{P}^D, f\}, \emptyset \right) - \mathcal{C} \left(\{\mathbb{P}, f\}, \emptyset \right)\end{aligned}$$

We notice, that the partial derivative is linear in f and therefore, that we can express the gradient for all $f \in \mathcal{F}$ leaving the selection variable open

$$\frac{\partial}{\partial \mathbf{w}} \mathcal{L}_D \left(\mathbb{P} + \sum_{l \in [p]} w^{f_l} \cdot f_l \right) \bigg|_{\mathbf{w}=0} = \mathcal{C} \left(\{\mathbb{P}^D, f\}, \{L\} \right) - \mathcal{C} \left(\{\mathbb{P}, f\}, \{L\} \right) .$$

Likelihood gradient ascent

The problem of likelihood gradient ascent is solved by

$$\operatorname{argmax}_{l \in [p]} \mathcal{C} \left(\{\mathbb{P}^D, f\}, \{L\} \right) - \mathcal{C} \left(\{\mathbb{P}, f\}, \{L\} \right)$$

This is the search for the maximal coordinate of a tensor in a network representation.

Targeting space consumption: **Tensor Network Decomposition**

- ▶ Avoid the creation of high-dimensional tensors
- ▶ Markov Logic Networks stored in local cores

Targeting runtimes: **Dynamic Programming**

- ▶ Avoid the repetition of local contractions
- ▶ Formula Selecting Architectures evaluate exponentially many formulas batchwise

Application in Inductive Logic Programming

Formula Selecting Architectures make use of the redundancies of propositional logics and provide a way to operate in large sets of formulas.

DATEV is an abbreviation of data processing
(in german: DATEn-Verarbeitung)

Traditional Approach

- ▶ Logic hard-coded in programs
- ▶ Processing by coded formulas
- ▶ User has full control

Data-driven Approach

- ▶ Learn logic from data
- ▶ Uncertainty tolerant processing
- ▶ User desires control

Advantages of Neuro-Symbolic Methods

- ▶ Exploit expert knowledge when learning models or designing them declaratively
- ▶ Maintain control through intrinsic model interpretability
- ▶ Minimize user interaction by uncertainty assessments