IV-Applications
Neuro Symbolic AI
Foundations of Neuro-Symbolic AI

Alex Goessmann

University of Applied Science Würzburg-Schweinfurt

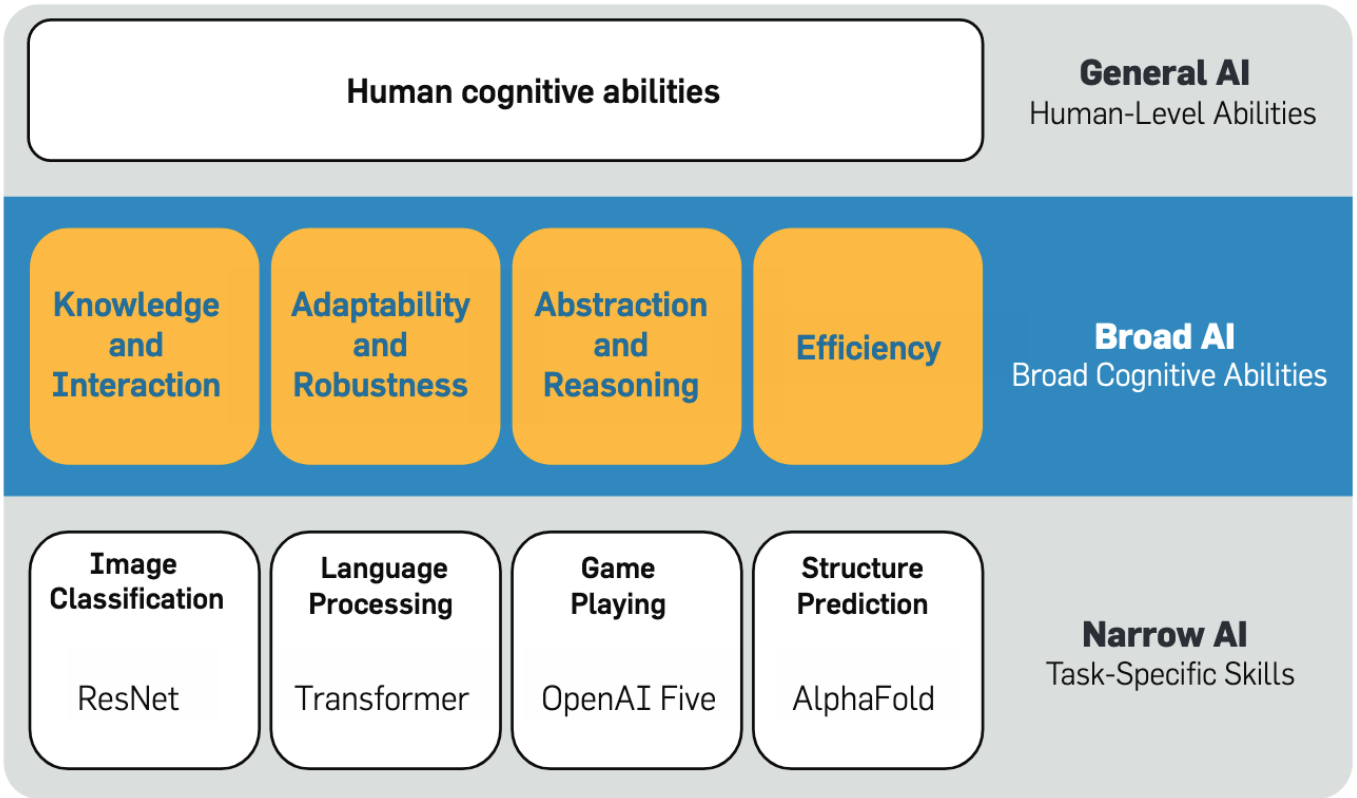
Summer Term 2026

Hochreiter: Towards a Broad AI

Communications of the ACM, April 2022

"Europe's Opportunity for a Broad AI:

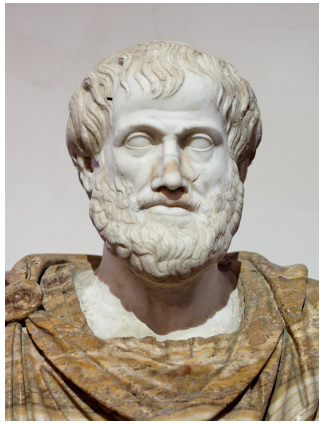
The most promising approach to a broad AI is a neuro-symbolic AI, that is, a bilateral AI that combines methods from symbolic and sub-symbolic AI."



The Symbolic Paradigm

How to represent knowledge?
$\rightarrow$ **Logical Syntax**

How to interpret knowledge?
$\rightarrow$ **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)

The Neural Paradigm

Sub-symbolic AI: Computations in Neural Architectures

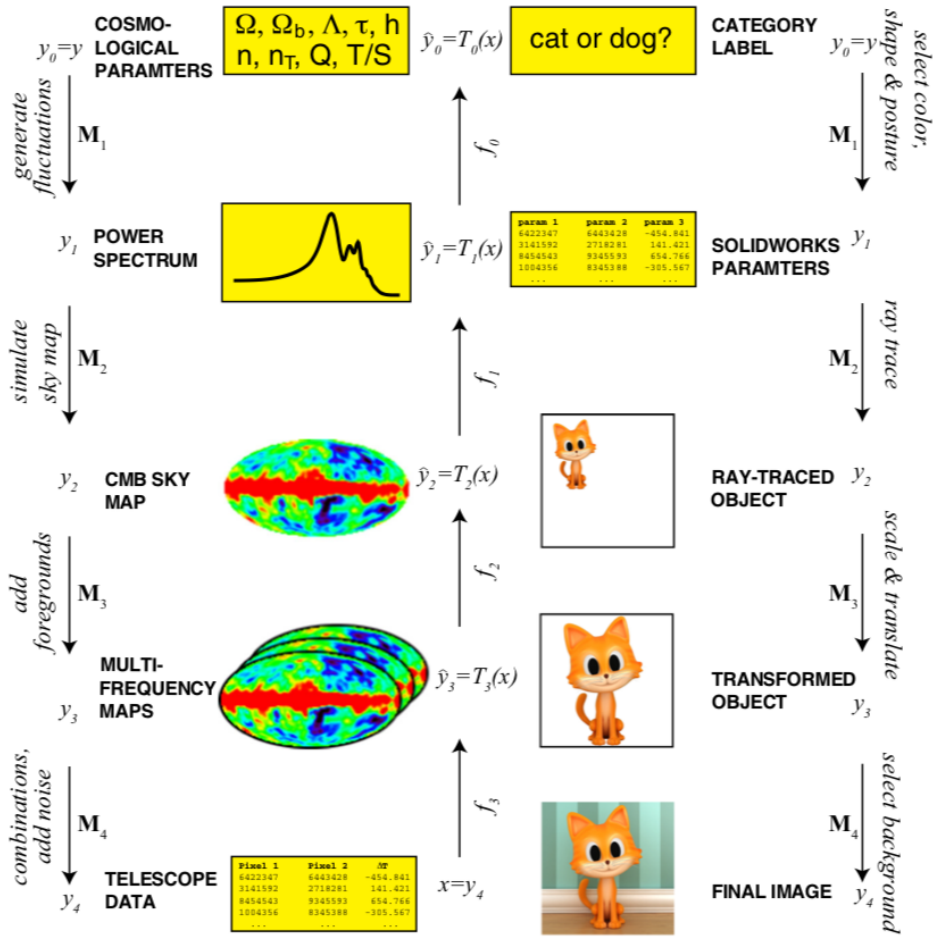
- Expressivity of Deep Networks: Effective representation of Data
- Differentiable parametrization

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

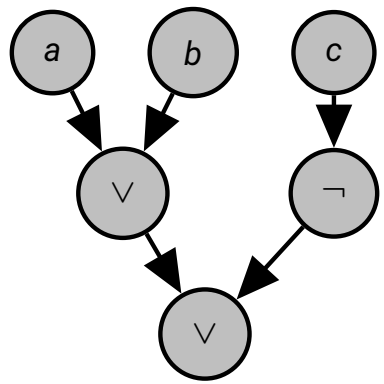
Combining the paradigms: Hierarchical structure of Logical Formulas

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

$$f = f_1 \circ f_2 \tag{1}$$

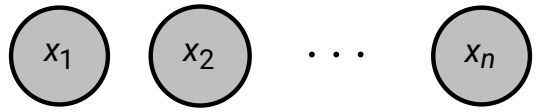
Example: Variables a or b implied by another variable c

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



Challenge: Overparametrization by Logical Formulas

A system with d binary variables



has a number of states

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

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}.

Learning Markov Logic Networks

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 θ_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

Structure Learning

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

Representing multiple 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

$$t : \bigtimes_{k \in [d]} [2] \times [p] \to [2]$$

defined for $l \in [p]$ by

$$t(x_0, \ldots, x_{d-1}, l) = f_l(x_0, \ldots, x_{d-1}) \,.$$

Representation as a Tensor Network

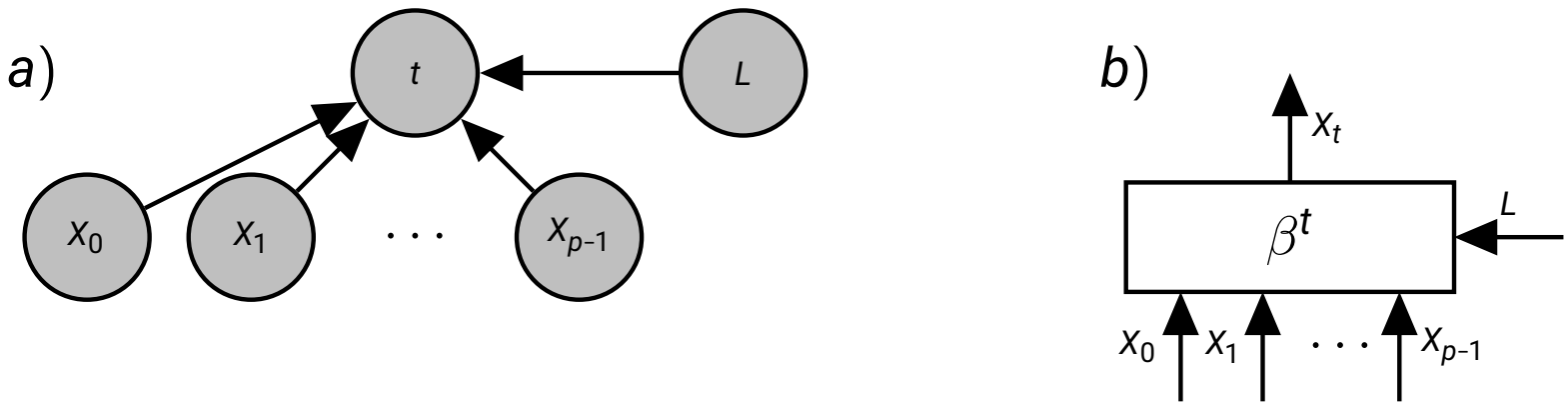
Given a set of p formulas $\{f_l : l \in [p]\}$, we define the formula selecting map as

$$t : \bigtimes_{k \in [d]} [2] \times [p] \to [2]$$

defined for $l \in [p]$ by

$$t(x_0, \ldots, x_{d-1}, l) = f_l(x_0, \ldots, x_{d-1}) \,.$$

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

Design of Formula selecting maps

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

Variable selecting maps

Definition (Variable selecting map)

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

$$t_V : \left(\bigtimes_{l \in [p]} [2] \right) \times [p] \to [2] \tag{2}$$

defined coordinatewise by

$$t_V(x_0, \ldots, x_{p-1}, l) = x_l \tag{3}$$

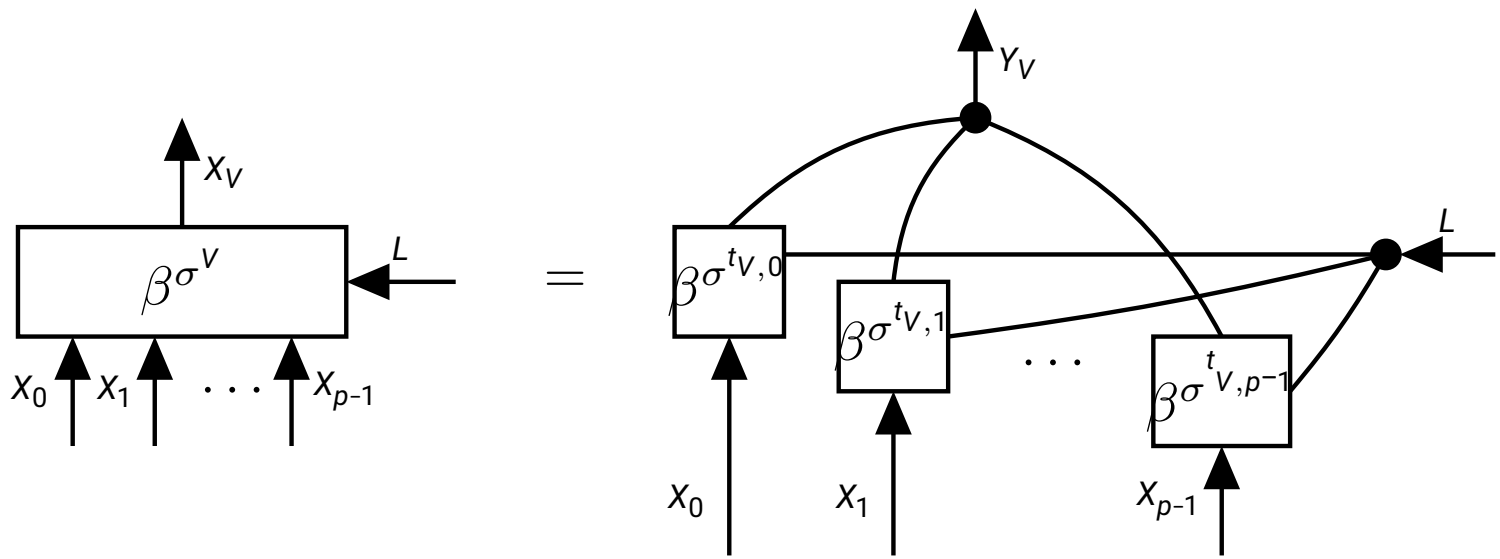
the corresponding variable selecting map.

Tensor Network representation

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

$$\beta^{\sigma^V} = \sum_{I \in [p]} \beta^{f^I} \otimes \epsilon_I \,.$$

We capture this to define cores $\beta^{\sigma^{t_{V,I}}} \in \mathbb{R}^2 \otimes \mathbb{R}^p \otimes \mathbb{R}^2$ and get a decomposition



Connective selecting maps

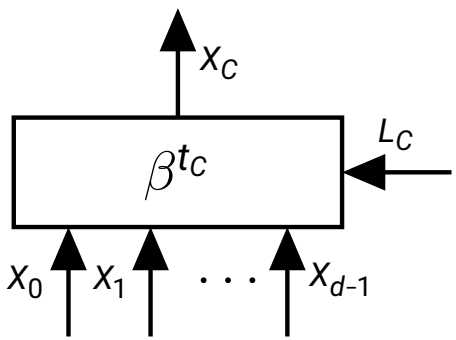
Definition (Connective selecting map)

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

$$t_C : \bigtimes_{k\in[d]} [2] \times [p_C] \to [2]$$

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

$$t_C(x_0, \ldots, x_{d-1}, l_C) = \circ_{l_C}(x_0, \ldots, x_{d-1}) .$$



Symbolic Neurons

Definition (Symbolic neuron)

Given an order $p \in \mathbb{N}$ let there be a connective selector L selecting connectives of same order and let $t_{V,0}, \ldots, t_{V,p-1}$ be a collection of variable selectors. The corresponding symbolic neuron is the map

$$\mathcal{N} : \left(\bigtimes_{k \in [d]} [2] \right) \times [p_C] \times \left(\bigtimes_{s \in [n]} [p_s] \right) \to [2]$$

defined for $x_0, \ldots, x_{d-1} \in \bigtimes_{k \in [d]} [2]$, $I_C \in [p_C]$ and $I_0, \ldots, I_{n-1} \in \bigtimes_{s \in [n]} [p_s]$ by

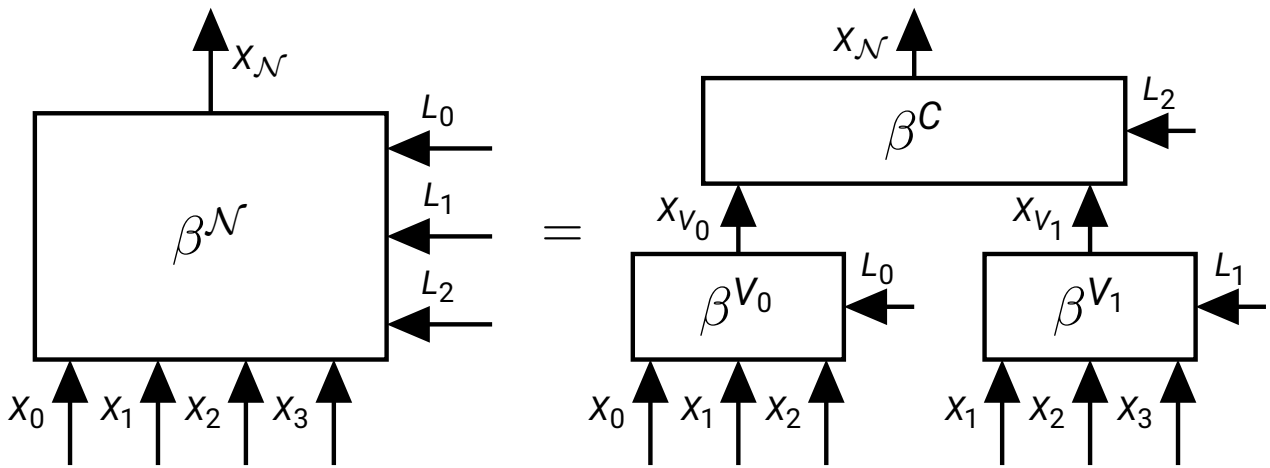
$$\begin{aligned} &\mathcal{N}(x_0, \ldots, x_{d-1}, I_C, I_0, \ldots, I_{n-1}) \\ &\quad = t_C(t_{V,0}(x_0, \ldots, x_{d-1}, I_0), \ldots, t_{V,p-1}(x_0, \ldots, x_{d-1}, I_{p-1}), I_C)\,. \end{aligned}$$

Decomposition of Symbolic Neurons

Let us specify a neuron $\mathcal{N} = f_1 \circ_2 f_3$ by candidates

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

We can decompose the encoding of a symbolic neuron by



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

Architectures

Architectures are collections of neurons

$$\mathcal{A} = \{\mathcal{N}_0, \ldots, \mathcal{N}_{n-1}\}$$

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.

Architecture specification in tnreason

We extend the nested lists encoding σ^f defined for propositional formulas f to also encode

- Logical neurons $\mathcal{N}$
- Formula selecting maps $\mathcal{A}$

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

$$\sigma^{\mathcal{N}} = [[\textit{"imp"},\textit{"eq"}],[\textit{"Wet"},\textit{"Sprinkler"}],[\textit{"Street"}]]$$

from which the formulas

$$[\textit{"imp"}, \textit{"Wet"}, \textit{"Street"}]$$
$$[\textit{"eq"}, \textit{"Wet"}, \textit{"Street"}]$$
$$[\textit{"imp"}, \textit{"Sprinkler"}, \textit{"Street"}]$$
$$[\textit{"eq"}, \textit{"Sprinkler"}, \textit{"Street"}]$$

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

$$\sigma^{\mathcal{A}} = \{\ \textit{"neur1"}: [[\textit{"imp"},\textit{"eq"}],[\textit{"neur2"}],[\textit{"Street"}]]\ ,$$
$$\textit{"neur2"}: [[\textit{"Inot"},\textit{"id"}],[\textit{"Wet"},\textit{"Sprinkler"}]]\ \}$$

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

Optimization of the Likelihood

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

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

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 t.

Gradient Ascent Approach

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

$$\begin{aligned}\frac{\partial}{\partial\theta}\mathcal{L}_D\left(\mathbb{P}+\theta\cdot f\right)|_{\theta=0} &= \mathbb{E}_{\mathbb{P}^D}\left[f\right]-\mathbb{E}_{\mathbb{P}}\left[f\right]\\ &= \Big\langle\{\mathbb{P}^D, f\}\Big\rangle_{[\varnothing]}-\langle\{\mathbb{P}, f\}\rangle_{[\varnothing]}\end{aligned}$$

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

$$\frac{\partial}{\partial\theta}\mathcal{L}_D\left(\mathbb{P}+\sum_{I\in[p]}\theta_{f_I}\cdot f_I\right)\Big|_{\theta=0} = \Big\langle\mathbb{P}^D, f\Big\rangle_{[L]}-\langle\mathbb{P}, f\rangle_{[L]}\ .$$

Gradient Ascent Approach

Likelihood gradient ascent

The problem of likelihood gradient ascent is solved by

$$\operatorname{argmax}_{I \in [p]} \left\langle \mathbb{P}^D, f \right\rangle_{[L]} - \langle \mathbb{P}, f \rangle_{[L]}$$

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

Algorithmic Efficiency

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.

Neuro-Symbolic AI for DATEV

DATEV is an abbrevation 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