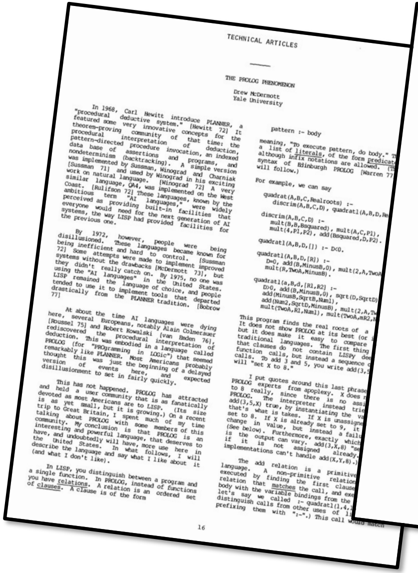
Computational Intelligence

LMU Munich
winter term 2024/2025

Thomas Gabor
Claudia Linnhoff-Popien

reading exercise

2024-11-28



TECHNICAL ARTICLES

THE PROLOG PHENOMENON

Drew McDermott
Yale University

In 1968, Carl Hewitt introduce PLANNER, a "procedural deductive system." [Hewitt 72] It featured some very innovative concepts for the theorem-proving community of that time: the procedural interpretation of deduction, pattern-directed procedure invocation, an indexed data base of assertions and programs, and nondeterminism (backtracking). A simple version was implemented by Sussman, Winograd and Charniak [Sussman 71] and used by Winograd in his exciting work on natural language. [Winograd 72] A very similar language, QA4, was implemented on the West Coast. [Rulifson 72] These languages, known by the ambitious term "AI languages," were widely perceived as providing built-in facilities that everyone would need for the next generation of AI systems, the way LISP had provided facilities for the previous one.

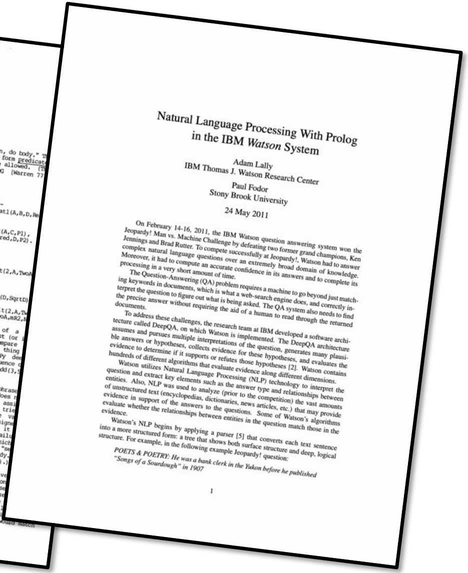
By 1972, however, people were being disillusioned. These languages became known for being inefficient and hard to control. [Sussman 72] Some attempts were made to implement improved systems without the drawbacks [McDermott 73], but they didn't really catch on. By 1975, no one was using the "AI languages" in the United States. LISP remained the language of choice, and people tended to use it to implement tools that departed drastically from the PLANNER tradition. [Bobrow 77]

At about the time AI languages were dying here, several Europeans, notably Alain Colmerauer [Roussel 75] and Robert Kowalski [van Emden 76], rediscovered the procedural interpretation of deduction. This was embodied in a language called PROLOG (for "PROgramming in LOGic") that seemed remarkably like PLANNER. Most Americans probably thought this was just the beginning of a delayed version of events here, and expected disillusionment to set in fairly quickly.

This has not happened. PROLOG has attracted and held a user community that is as fanatically devoted as most Americans are to LISP. (Its size is as yet small, but it is growing.) On a recent trip to Great Britain, I spent much of my time talking about PROLOG with some members of this community. My conclusion is that PROLOG is an interesting and powerful language, that deserves to have, and undoubtedly will have, more use here in the United States. In what follows, I will describe the language and say what I like about it (and what I don't like).

In LISP, you distinguish between a program and a single function. In PROLOG, instead of functions you have relations. A relation is an ordered set of clauses. A clause is of the form

16



Natural Language Processing With Prolog in the IBM *Watson* System

Adam Lally
IBM Thomas J. Watson Research Center

Paul Fodor
Stony Brook University

24 May 2011

On February 14-16, 2011, the IBM Watson question answering system won the Jeopardy! Man vs. Machine Challenge by defeating two former grand champions, Ken Jennings and Brad Rutter. To compete successfully at Jeopardy!, Watson had to answer complex natural language questions over an extremely broad domain of knowledge. Moreover, it had to compute an accurate confidence in its answers and to complete its processing in a very short amount of time.

The Question-Answering (QA) problem requires a machine to go beyond just matching keywords in documents, which is what a web-search engine does, and correctly interpret the question to figure out what is being asked. The QA system also needs to find the precise answer without requiring the aid of a human to read through the returned documents.

To address these challenges, the research team at IBM developed a software architecture called DeepQA, on which Watson is implemented. The DeepQA architecture assumes and pursues multiple interpretations of the question, generates many plausible answers or hypotheses, collects evidence for these hypotheses, and evaluates the evidence to determine if it supports or refutes those hypotheses [2]. Watson contains hundreds of different algorithms that evaluate evidence along different dimensions.

Watson utilizes Natural Language Processing (NLP) technology to interpret the question and extract key elements such as the answer type and relationships between entities. Also, NLP was used to analyze (prior to the competition) the vast amounts of unstructured text (encyclopedias, dictionaries, news articles, etc.) that may provide evidence in support of the answers to the questions. Some of Watson's algorithms evaluate whether the relationships between entities in the question match those in the evidence.

Watson's NLP begins by applying a parser [5] that converts each text sentence into a more structured form: a tree that shows both surface structure and deep, logical structure. For example, in the following example Jeopardy! question:

POETS & POETRY: He was a bank clerk in the Yukon before he published "Songs of a Sourdough" in 1907

1

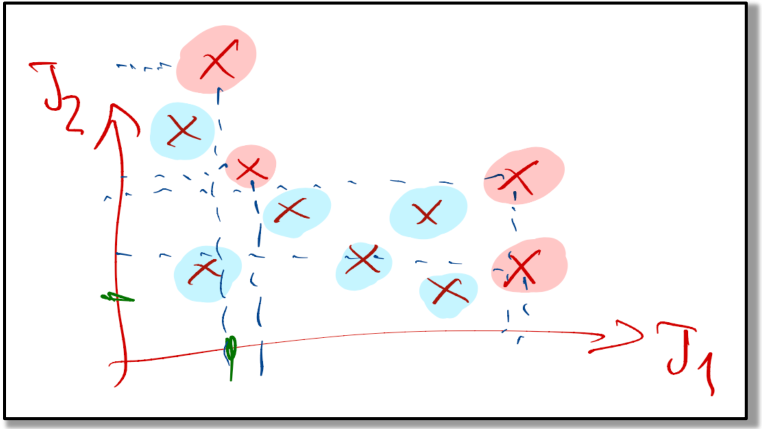
"early exam"

2025-02-06

18:30h

on main campus

Multi-Objective Optimization



Definition 6 (multi-objective optimization)**.** Let $\mathcal{E} = (\mathcal{X}, \mathcal{T}, \tau, E, \langle X_u \rangle_{0 \leq u \leq t})$ be an optimization process. $\mathcal{E}$ is a multi-objective optimization process iff the target space $\mathcal{T}$ has the form $\mathcal{T} = \mathcal{T}_0 \times \cdots \times \mathcal{T}_{N-1}$ for some $N \in \mathbb{N}$ with $\leq_i$ being a total order on $\mathcal{T}_i$ for any $i \in [0; N-1] \subset \mathbb{N}$. Unless stated otherwise, we assume that no single total order on $\mathcal{T}$ is available. However, we can construct a partial order $\preceq$ so that

$$(x_0, ..., x_{N-1}) \preceq (x'_0, ..., x'_{N-1}) \iff \forall i \in [0; N-1] \subset \mathbb{N} \; : \; x_i \leq x'_i,$$

which is sufficient to adapt many standard optimization algorithms.

Definition 6 (multi-objective optimization)**.** Let $\mathcal{E} = (\mathcal{X}, \mathcal{T}, \tau, E, \langle X_u \rangle_{0 \leq u \leq t})$ be an optimization process. $\mathcal{E}$ is a multi-objective optimization process iff the target space $\mathcal{T}$ has the form $\mathcal{T} = \mathcal{T}_0 \times \cdots \times \mathcal{T}_{N-1}$ for some $N \in \mathbb{N}$ with $\leq_i$ being a total order on $\mathcal{T}_i$ for any $i \in [0; N-1] \subset \mathbb{N}$. Unless stated otherwise, we assume that no single total order on $\mathcal{T}$ is available. However, we can construct a partial order $\preceq$ so that

$$(x_0, ..., x_{N-1}) \preceq (x'_0, ..., x'_{N-1}) \iff \forall i \in [0; N-1] \subset \mathbb{N} \; : \; x_i \leq x'_i,$$

which is sufficient to adapt many standard optimization algorithms.

Definition 7 (Pareto front for optimization)**.** Let $\mathcal{E} = (\mathcal{X}, \mathcal{T}, \tau, E, \langle X_u \rangle_{0 \leq u \leq t})$ be a multi-objective optimization process with $\preceq$ being a partial order on the multi-objective target space $\mathcal{T}$.

- A solution candidate x Pareto-dominates a solution candidate x' (assuming minimization) iff $x \preceq x'$.
- A solution candidate x is Pareto-optimal if there exists no other solution candidate $x' \in \mathcal{X}$ so that $x' \preceq x$.
- The set of all Pareto-optimal solution candidates in $\mathcal{X}$ is called the Pareto front of $\mathcal{X}$ (w.r.t. $\preceq$).

Example: Multi-Objective Evolutionary Algorithm

The Goal Class Hierarchy

Goal Class 5: State Values	
Goal Class 4: Rewards and Costs	
Goal Class 3: Goal Direction	
Goal Class 2: Goal Valuation	Goal Class 2.5: Multiple Goal Valuations
Goal Class 1: Goal Predicate	Goal Class 1.5: Multiple Goal Predicates
Goal Class 0: No Goals	

Goal Class 3: Goal Direction

“I know which way it’s getting better!”

Gradients of Target Functions

Derivation

Algorithm 5 (gradient descent)**.** Let $\mathcal{D} = (\mathcal{X}, \mathcal{T}, \tau, e, \langle x_u \rangle_{0 \leq u \leq t})$ be an optimization process. Let $\mathcal{T}$ be continuous ($\mathcal{T} = \mathbb{R}$, e.g.) and let $\tau' : \mathcal{X} \to \mathcal{T}$ be the first derivative of τ. The process $\mathcal{D}$ continues via gradient descent (with update rate $\alpha \in \mathbb{R}^+$) if e is of the form

$$e(\langle x_u \rangle_{0 \leq u \leq t}, \tau) = x_{t+1} = x_t - \alpha \cdot \tau'(x_t).$$

The learning rate α can also be given as a function, usually $\alpha : \mathbb{N} \to \mathbb{R}$ so that $e(\langle x_u \rangle_{0 \leq u \leq t}, \tau) = x_{t+1} = x_t - \alpha(t) \cdot \tau'(x_t)$.
If τ is stochastic, this process is called stochastic gradient descent (SGD).

Algorithm 5 (gradient descent)**.** Let $\mathcal{D} = (\mathcal{X}, \mathcal{T}, \tau, e, \langle x_u \rangle_{0 \le u \le t})$ be an optimization process. Let $\mathcal{T}$ be continuous ($\mathcal{T} = \mathbb{R}$, e.g.) and let $\tau' : \mathcal{X} \to \mathcal{T}$ be the first derivative of τ. The process $\mathcal{D}$ continues via gradient descent (with update rate $\alpha \in \mathbb{R}^+$) if e is of the form

$$e(\langle x_u \rangle_{0 \le u \le t}, \tau) = x_{t+1} = x_t - \alpha \cdot \tau'(x_t).$$

The learning rate α can also be given as a function, usually $\alpha : \mathbb{N} \to \mathbb{R}$ so that $e(\langle x_u \rangle_{0 \le u \le t}, \tau) = x_{t+1} = x_t - \alpha(t) \cdot \tau'(x_t)$.
If τ is stochastic, this process is called stochastic gradient descent (SGD).

Algorithm 6 (gradient descent (policy))**.** Let π_θ be a policy π that depends on vector of continuous parameters $\theta \in \Theta$ such that usually $\Theta = \mathbb{R}^N$ for some N. Let $\tau : \Theta \to \mathcal{T}$ be a target function on the parameters θ of a policy π_θ. Let $\mathcal{T}$ be continuous ($\mathcal{T} = \mathbb{R}$, e.g.) and let $\tau' : \Theta \to \mathcal{T}$ be the first derivative of τ, i.e., $\tau'(\theta) = \frac{\partial\, \tau(\theta)}{\partial\, \theta}$. If $\mathcal{D} = (\Theta, \mathcal{T}, \tau, e, \langle x_u \rangle_{0 \le u \le t})$ is an optimization process that continues via gradient descent, $\mathcal{D}$ is a process of policy optimization via gradient descent.

Many Variants of Gradient Descent

source: https://optimization.cbe.cornell.edu/index.php?title=AdaGrad

Algorithm 1: AdaGrad general algorithm

η: Stepsize ;
$f(x)$: Stochastic objective function ;
x_1: Initial parameter vector;
for $t = 1$ *to* T **do**
  Evaluate $f_t(x_t)$;
  Get and save g_t ;
  $G_t \leftarrow \sum_{\tau=1}^{t} g_\tau g_\tau^\top$;
  $x_{t+1} \leftarrow x_t - \eta G_t^{-1/2} g_t$;
end
return x_t

source: https://arxiv.org/pdf/1412.6980.pdf%5D

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)