ARMIN B. CREMERS

Armin B. Cremers received his doctoral degree in mathematics and his lectureship qualification in computer science from the University of Karlsruhe (now KIT). He has served on the Computer Science Faculties of the University of Southern California, the University of Dortmund, and, since 1990, the University of Bonn as Head of the Artificial Intelligence / Robotics/ Intelligent Vision Systems Research Groups. In 2002 he became Founding Director of the Bonn-Aachen International Center for Information Technology (B-IT), Emeritus since 2014.

Machine Spaces and General Problem Classes

Machine Spaces and General Problem Classes

Armin B. Cremers

joint work with Jörg Zimmermann

B-IT, University of Bonn, Germany

Overview

1. Foundations
   1.1 Induction Problem: the Standard Learning Framework
   1.2 Possibilities and Limits: Fundamental Results
2. Complexity in Practice
   2.1 Machines: Axioms and Metrics
   2.2 Reference Machines
   2.3 Iterated Boolean Circuits
   2.4 Smart Search in Circuit Space
3. 3 Realms: Organizing Science with HPC at its core
4. Outlook

The General Prediction Problem

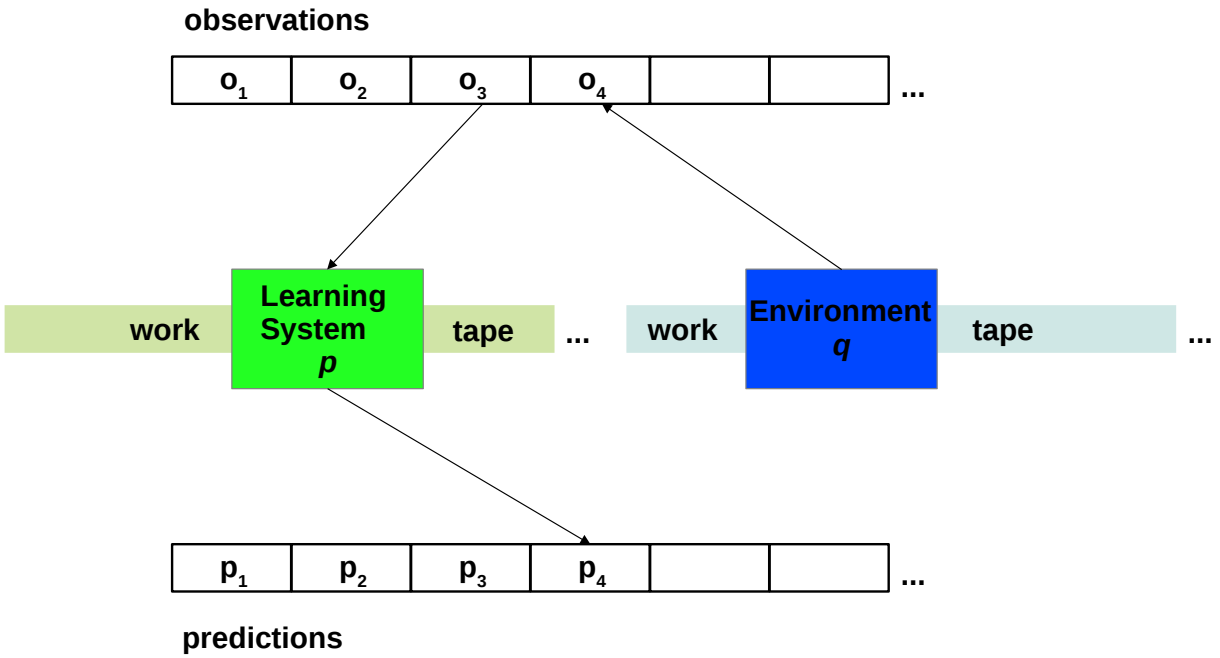
A fundamental problem class is the general prediction problem, which can be illustrated as follows: given a finite sequence of bits, e.g.:

0010010000111111011010101000100010000101

Question: What is the next bit?

The Standard Learning Framework

A predictor observing and predicting an environment:



The Standard Learning Framework

In this standard framework for the general prediction problem, the predictor waits till the environment has produced a new output and vice versa.

The predictor doesn't know how many internal transactions the environment has executed in order to generate an observation (1 second, 1 year, 100 billion years, ...).

Solomonoff Induction

- Bayesian learning in program space.
- Programs are run on a universal Turing machine U.
- Prior $\sim 2^{-|p|}$, $\quad |p| =$ length of program p in bits.
- But posterior distribution on program space is not computable! (the programs stopping to produce output cause trouble).

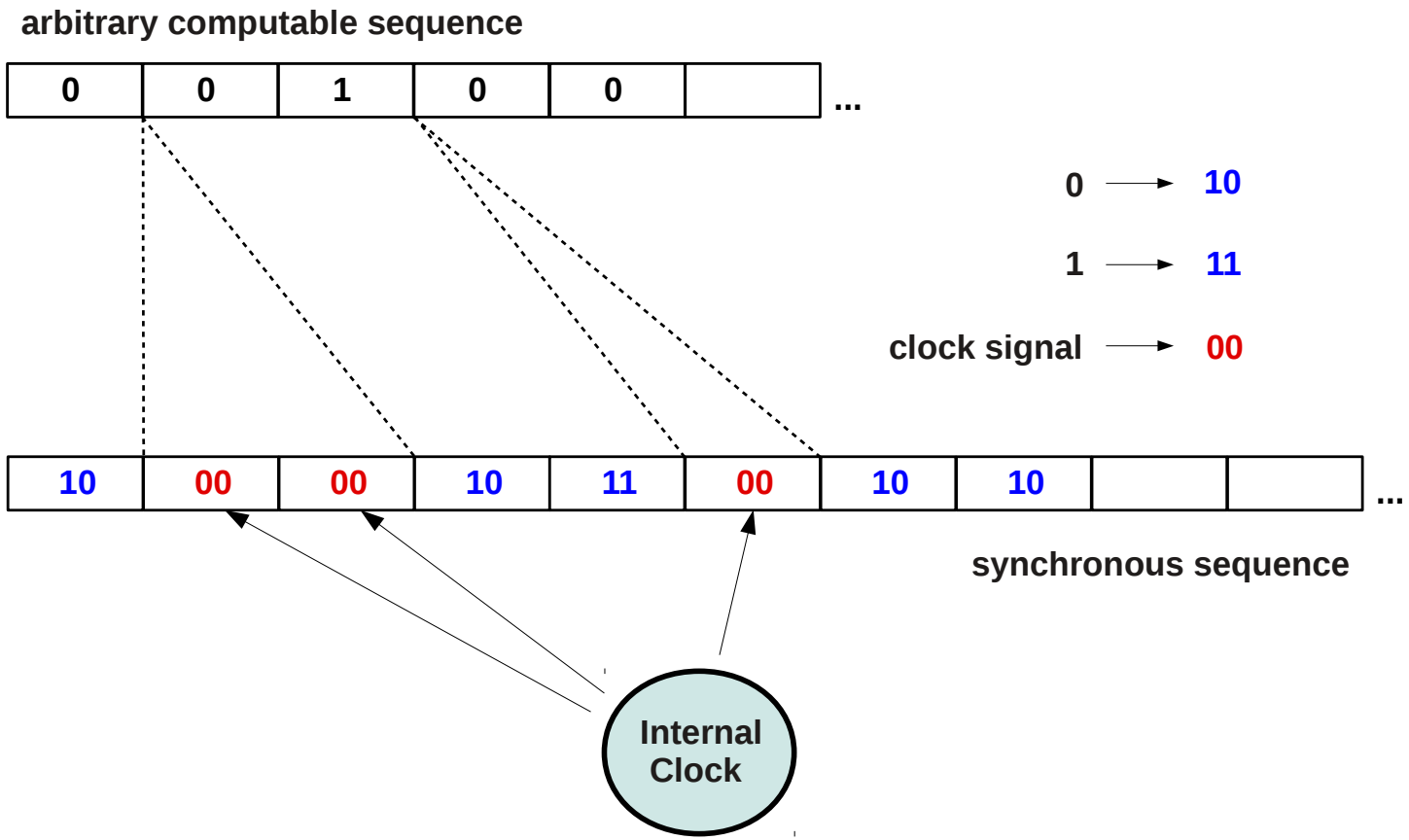
The Synchronous Learning Framework

Observation: in real world learning situations, environment and the predictor are not suspended while the other one is busy.

In a Synchronous Learning Framework, the time scales of the predictor and the environment are coupled.

This can be achieved, for example, by clockification.

Clockification



Fundamental Result for the Synchronous Learning Framework

If the predictor is enhanced by an internal clock:

Effective universal induction is possible!

Hence our research now focuses on efficient universal induction.

The Reference Machine

In Solomonoff Induction, one uses a fixed, but arbitrary universal Turing machine U for the execution of programs p.

This machine U is called the reference machine.

But on finite data x, the choice of a universal reference machine can manipulate the posterior probability of a program consistent with x between ϵ and $1 - \epsilon$ by using complex transition tables.

The Reference Machine

This means that only asymptotic properties are well-defined, in order to get reasonable predictions for finite data sets, we need a:

$\rightsquigarrow$ "natural" reference machine. But how can one define "natural" for machines?

$\rightsquigarrow$ axiomatic investigation of the "Machine Space".

Machine Axioms

The "ontology" of the machine space:

- State space Σ
- Input space I
- Output space O
- Program space P
- *initializer*: a mapping $init$ from $P \times I$ to Σ
- *output operator*: a mapping out from Σ to O

Machine Axioms

A machine wrt. time T and a state space Σ is a mapping M from $\Sigma \times T$ to Σ (denoted by $M_t(s)$).

- Subset $HALT$ of Σ. States in $HALT$ will be used to signal termination of a computation.
- $TERM_M(s)$: denotes the set of time points t with $M_t(s) \in HALT$.

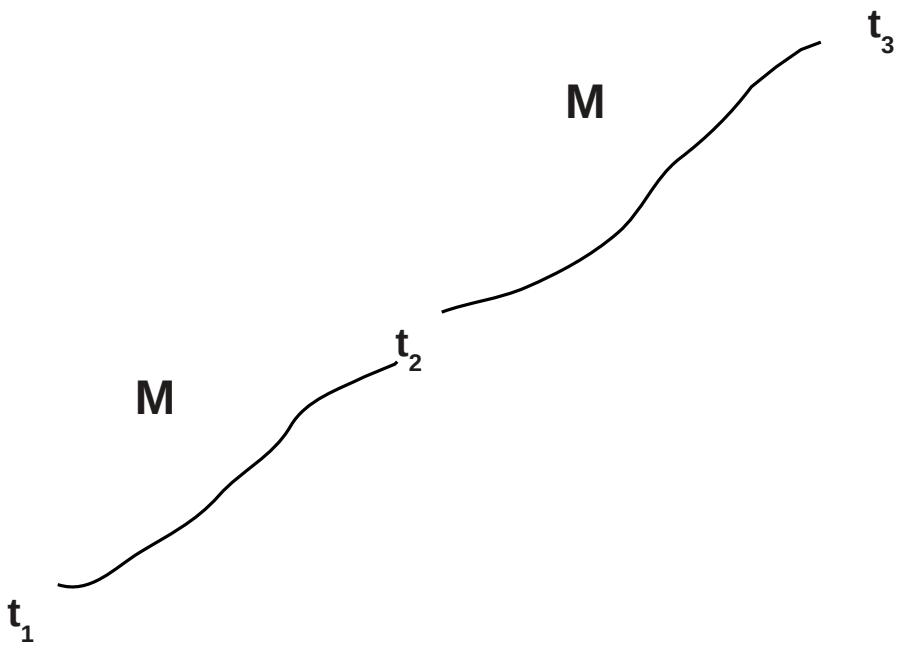
Machine Axioms

(Start) $\forall s \in \Sigma : M_{0^{(t)}}(s) = s \quad$ (i.e., $M_{0^{(t)}} = id_\Sigma$),

(Action) $\forall t_1, t_2 \in T : M_{t_1+t_2} = M_{t_2} \circ M_{t_1}$.

These two axioms state that time is operating on the state space via machine M.

Machine Axioms



The Action Axiom implies that M traces out trajectories in state space and does not jump from START to STOP

Machine Axioms

(Stop) $\forall s \in \Sigma, t_1, t_2 \in T : t_1 \in TERM_M(s)$ and $t_1 \leq t_2 \Rightarrow M_{t_1}(s) = M_{t_2}(s)$.

That is, after reaching a termination state, nothing changes anymore, i.e., termination states are fixpoints of the machine dynamics.

Machine Axioms

(Well-Termination) $\forall s \in \Sigma : TERM_M(s) \neq \emptyset \Rightarrow \exists t_1 \in TERM_M(s)\ \forall t_2 \in TERM_M(s) : t_1 \leq t_2$.

Well-termination requires that if a machine terminates on s, i.e., reaches $HALT$ for some point in time, then there is a first point in time when this happens.

If $TERM_M(s)$ is non-empty, its least element is denoted by t^*.

Implementation

Definition: A function $f : I \to O$ is *implemented* by $p \in P$ on M iff

$$f(x) = out(M_{t^*}(init(p, x))$$

for all $x \in I$.

Functions f which are implementable on a machine M are called "M-computable". $[p]_M$ denotes the (partial) function implemented by p on M.

Measuring Distance: Time Factors

Measuring distance wrt. a finite input set by maximal ratio between running times on machine 1 and machine 2.

Let $time_p^M(x) = min(TERM_M(init(p, x)))$.

Then define a transfer function between machines as follows:

$\tau : T \to T$ is an *admissible time transfer function (attf)* from M_1 to M_2 iff τ is monotone and $\forall p_1 \in P_1 \exists p_2 \in P_2 : [p_1]_{M_1} = [p_2]_{M_2}$ and $\forall x \in I : time_{p_2}^{M_2}(x) \leq \tau(time_{p_1}^{M_1}(x))$.

Transfer functions will be used to measure the "time distance" of two machines in machine space.

M-dependent Computability and Complexity

A machine M defines implicitly a set of functions, the M-computable functions:

$$COMP(M) = \{f | f : I \to O, f \text{ is } M - computable\}$$

But it also defines complexity classes in analogy to the classical complexity classes:

$$TIME_M(g) = \{f | f \in COMP(M), [p]_M = f, time_p^M(x) \leq g(x)\}$$

Metrics on Machine Space

M_1 and M_2 are time-compatible if they operate on the same time structure, input space and output space.

A generalized metric $\Delta^{(t)}$ on machine space is now defined as follows:

$\Delta^{(t)}(M_1, M_2) = \{\tau | \tau \text{ is an attf from } M_1 \text{ to } M_2\}$.

This roughly corresponds to statements like: "Machine A can simulate machine B with a logarithmic factor".

Metrics on Machine Space

One can combine and compare sets of functions much like single functions. Let $\alpha, \beta \subseteq T^T$:

$\alpha \circ \beta := \{\tau_1 \circ \tau_2 | \tau_1 \in \alpha, \tau_2 \in \beta\}$.

$\alpha \leq \beta$ iff $\forall \tau_2 \in \beta \exists \tau_1 \in \alpha : \tau_1 \leq \tau_2$.

Metrics on Machine Space

- By these definitions sets of attfs become a directedly ordered monoid (dom).
- Directed monoids can be used as ranges for generalized metrics, allowing many standard constructions of topology.

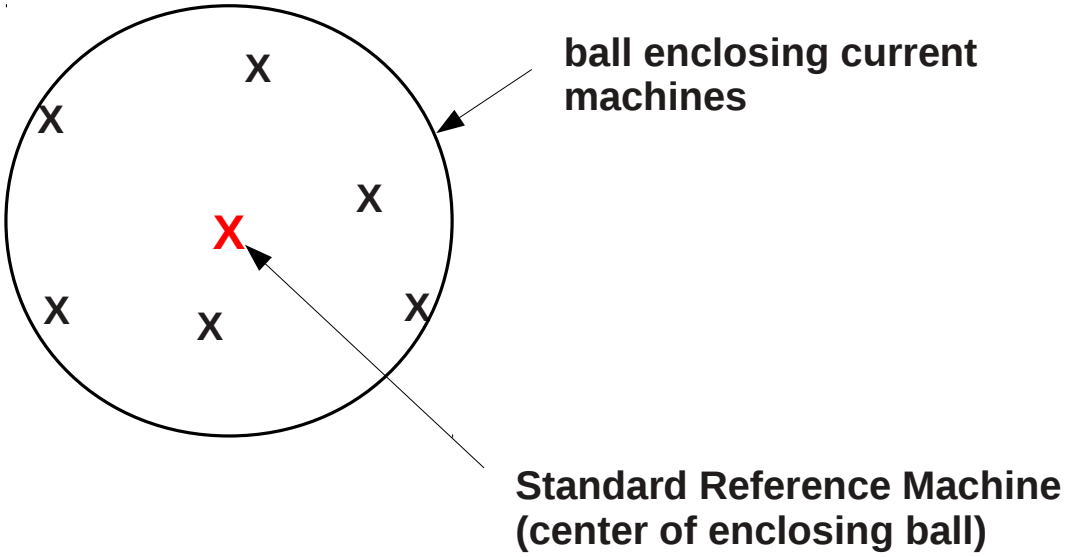
Our metric can be classified as a *dom-valued directed pseudometric*, satisfying the following triangle inequality:

$$\Delta^{(t)}(M_1, M_3) \leq \Delta^{(t)}(M_2, M_3) \circ \Delta^{(t)}(M_1, M_2).$$

Standard Reference Machine

- How to define a "Standard Reference Machine" (SRM), which can serve as a anchor point for concrete complexity statements and universal induction?
- Idea: Define the SRM as the "center" of the smallest ball enclosing current real world computing machines.

Standard Reference Machine



Standard Reference Machine

Qualitative criteria:

1. The SRM should be in reasonable correspondence to current computing devices.

2. The SRM should have a clean definition enabling mathematical analysis.

Iterated Boolean Circuits

The predictor dynamic is defined by a boolean transition function $F_P : 2^{n_P} \to 2^{n_P}$. The environment dynamic is defined by a boolean transition function $F_E : 2^{n_E} \to 2^{n_E}$.

A full cycle is realized by the following steps:

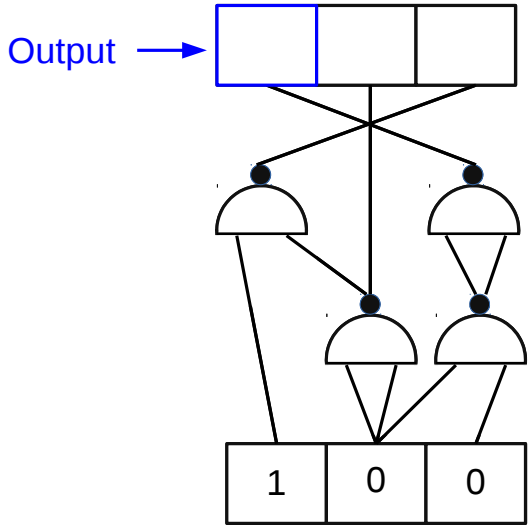
- the output bits of the current environment vector are written to the percept bits of the current predictor vector.
- application of F_P to the predictor vector results in a new predictor vector
- the newly computed predictor bits are written to an output tape
- application of F_E results in a new environment vector

Iterated Boolean Circuits: Prediction

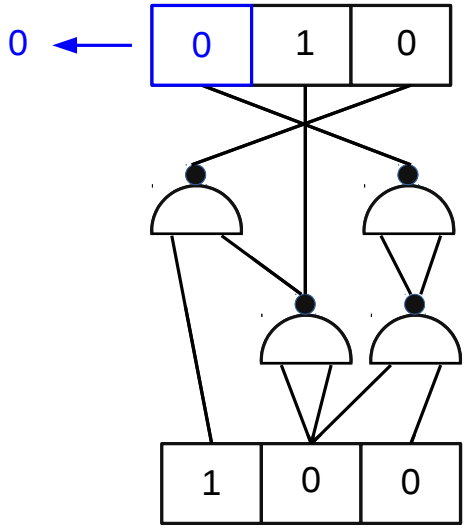
00010001000 ?

- Can be generated by an Iterated Boolean Circuit with 4 NAND-gates.
- Prediction for the 12th bit is a “1”.

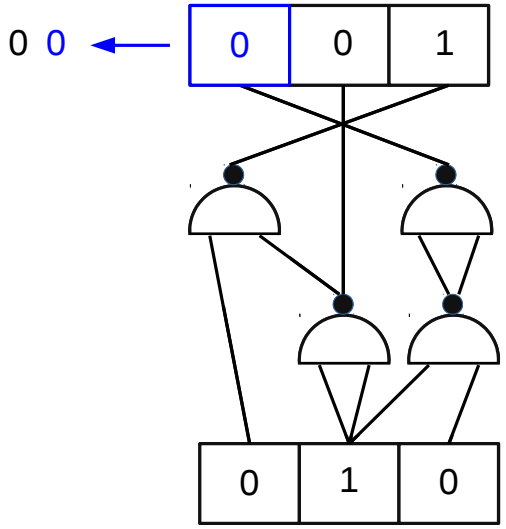
Iterated Boolean Circuits: Prediction



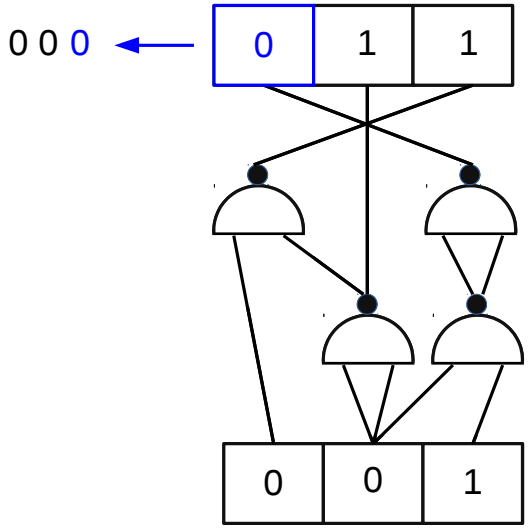
Iterated Boolean Circuits: Prediction



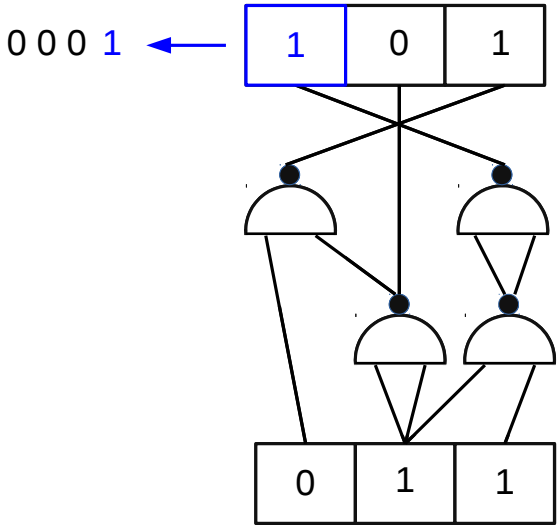
Iterated Boolean Circuits: Prediction



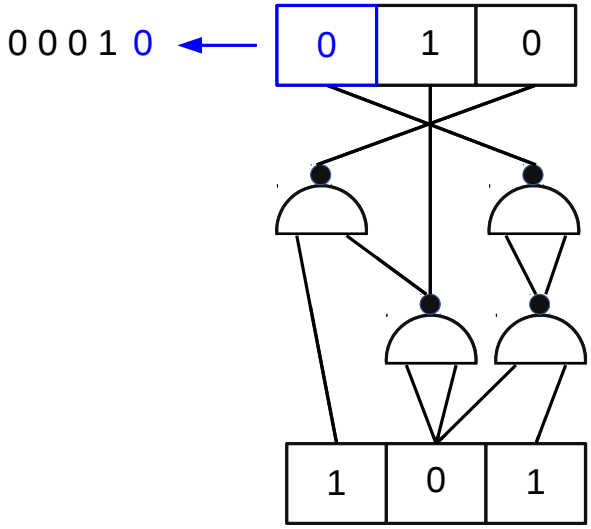
Iterated Boolean Circuits: Prediction



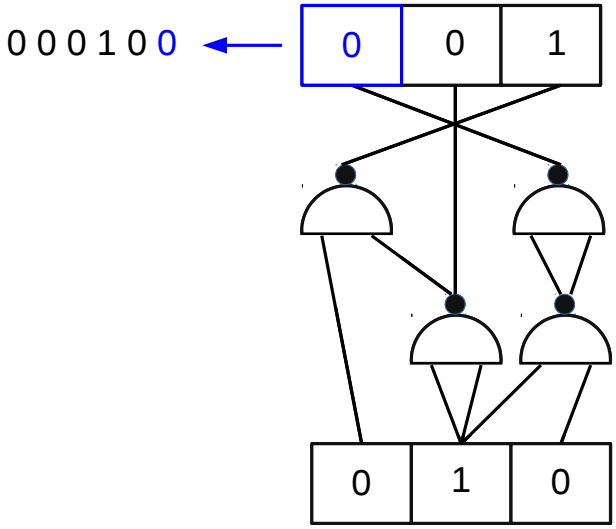
Iterated Boolean Circuits: Prediction



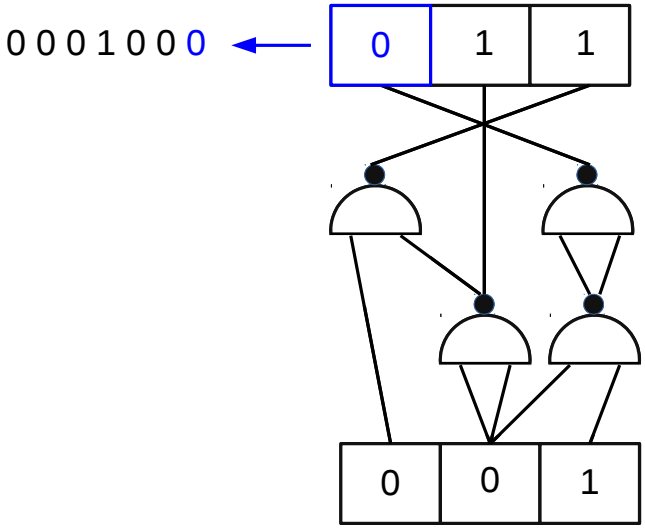
Iterated Boolean Circuits: Prediction



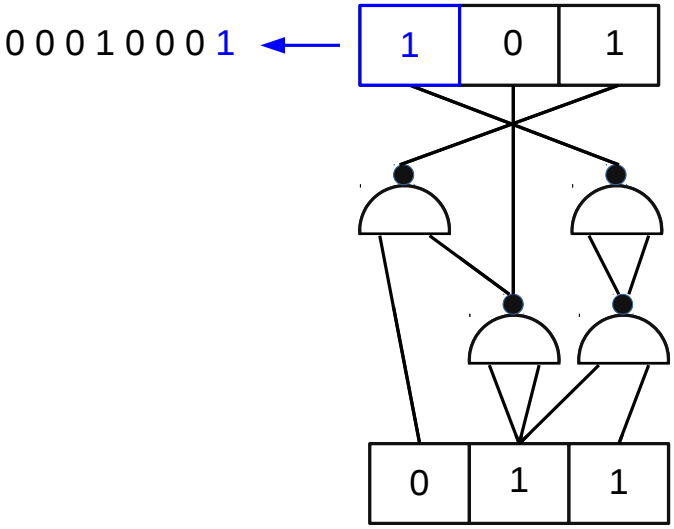
Iterated Boolean Circuits: Prediction



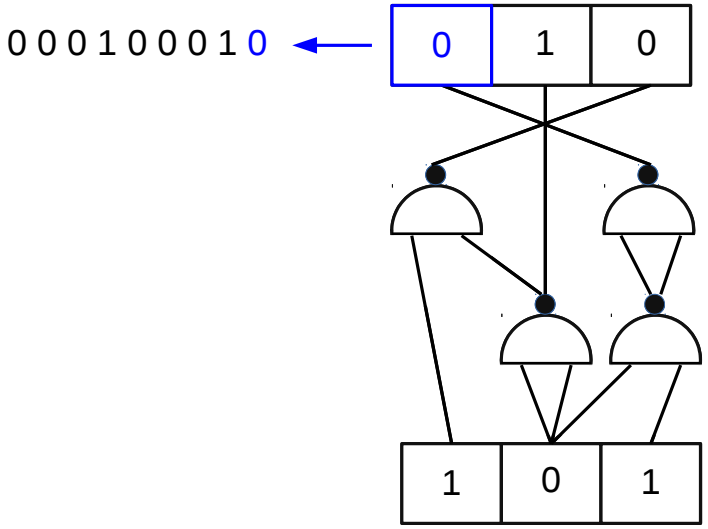
Iterated Boolean Circuits: Prediction



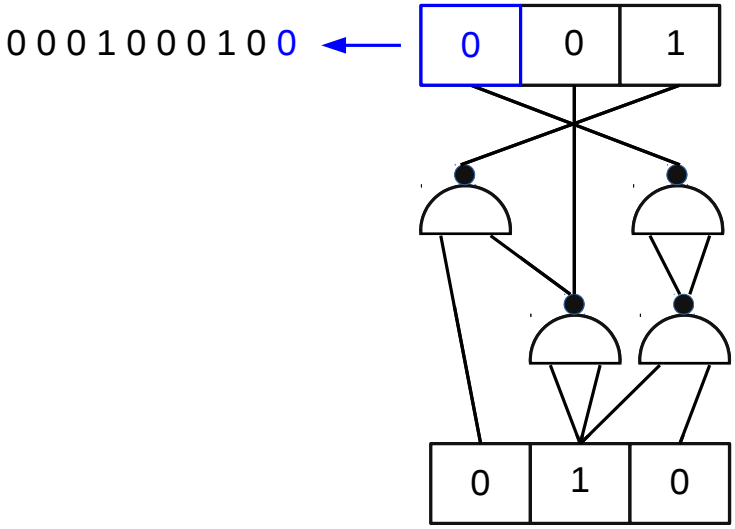
Iterated Boolean Circuits: Prediction



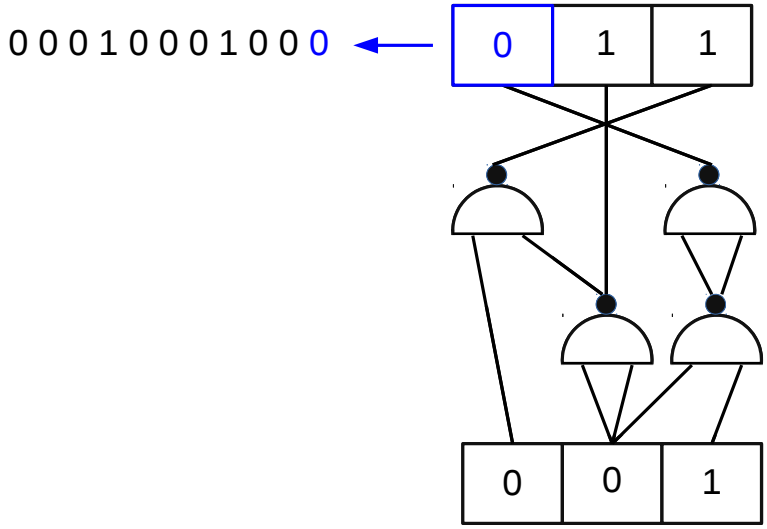
Iterated Boolean Circuits: Prediction



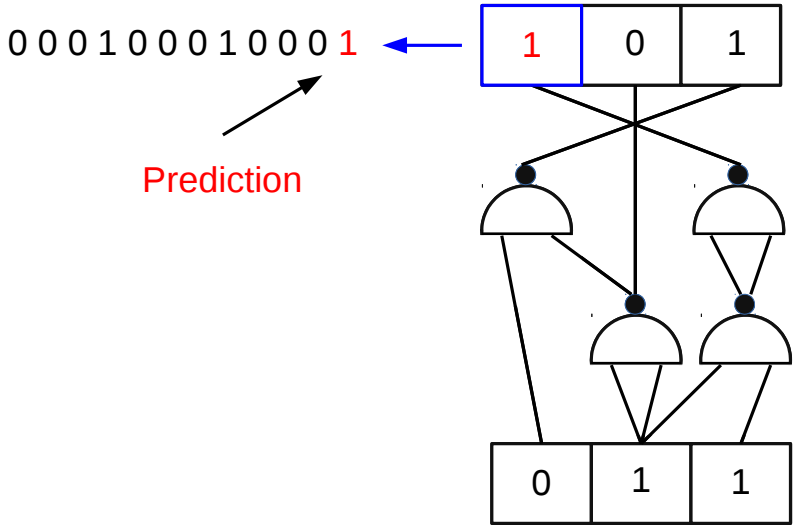
Iterated Boolean Circuits: Prediction



Iterated Boolean Circuits: Prediction



Iterated Boolean Circuits: Prediction



How to compute probabilistic predictions with IBCs

- We use a Boolean Circuit Description Language
- Identify "program" with "circuit description"

If $l(p)$ is the length of a circuit description in bits, then the prior probability is defined as follows:

$$\pi(p) \sim 2^{-l(p)}$$

How to search in circuit space?

- Number of possible circuits grows superexponentially fast with the number of gates
- Even for small numbers (like 10) an exhaustive search is impossible
- Idea: in order get non-trivial results in circuit space, combine evolutionary optimization, (deep) machine learning, and high performance computing.

Playing Go using Deep Learning: AlphaGo

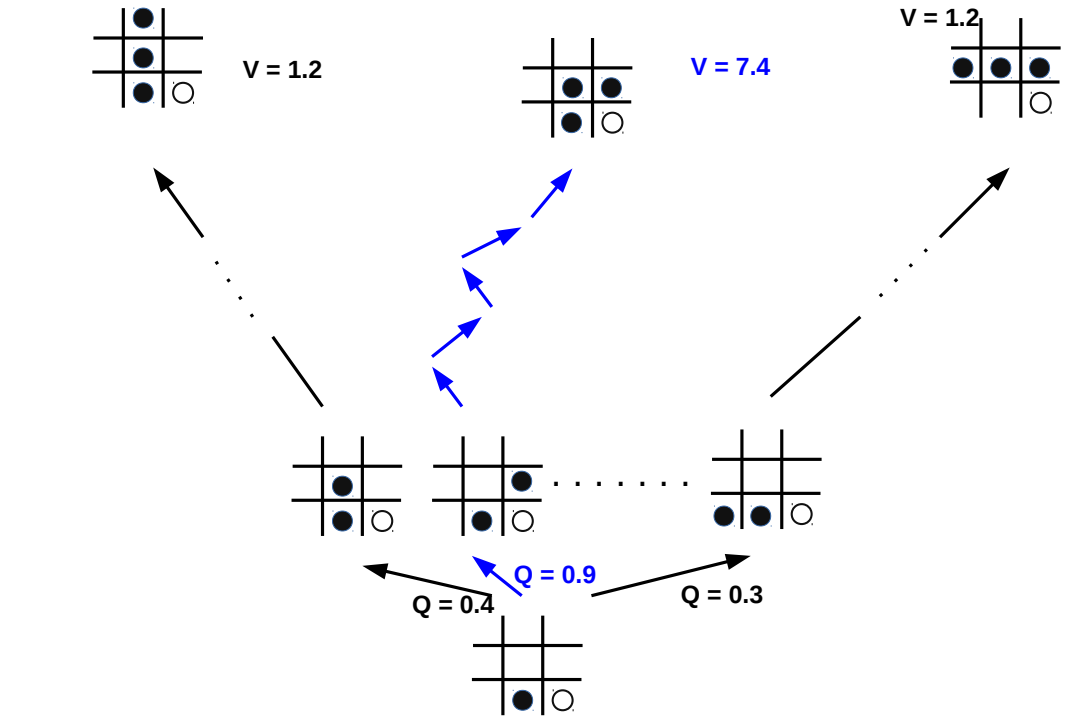
- In March 2016 a Go program developed by Google Deep Mind won against one of the best professional Go-players, Lee Sedol.
- Based on previous progress of Computer Go this was not expected within the next decade.
- Core idea: use deep learning to focus exploration of the game tree on the most promising parts.

Deep Learning $\sim$ Neural Networks with many hidden layers

Playing Go using Deep Learning: AlphaGo

- Value network: approximates the value of a position
- Policy network: Move selection
- These networks are trained in two phases: supervised learning using an expert games DB and self-play.
- VN and PN are realized as reinforcement learning, the Q-function is learned by a deep neural network.

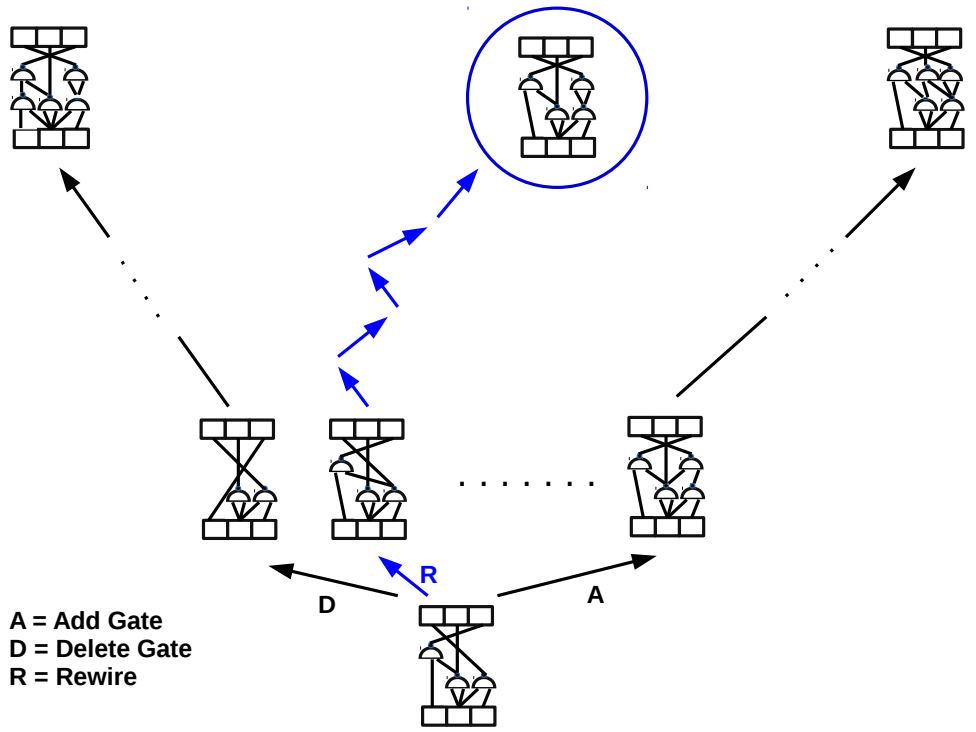
Deep-Learning driven Search in Go Game-Tree



How to search in circuit space

- By introducing operators on circuit space (like adding a gate, removing a gate, rewire a connection,...) we transform the induction problem into a reachability problem for graphs.
- Now we transfer methods used for AlphaGo to navigate the game tree to the graph search problem, i.e. we use deep learning to focus search on the promising parts of the graph.

How to search in circuit space?



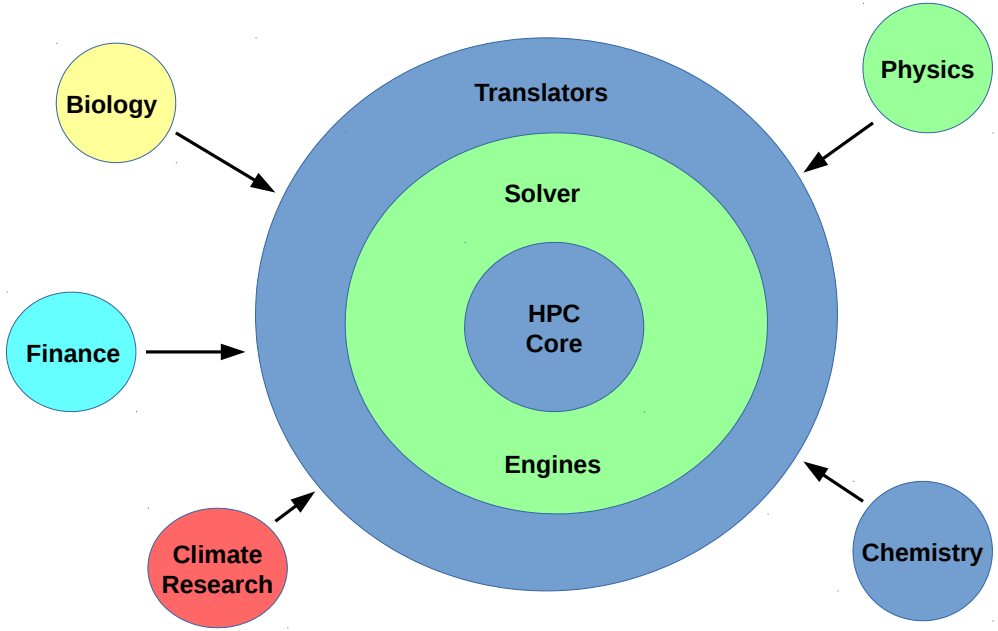
Smart Search: Identifying General Problem Classes

- Induction problem
- Strategies for (board)games
- Optimization Problems
- Inverse Problems
- Reachability

Three Realms

- Application Domain: any research field (having formalizable subproblems)
- Translator Domain: Formalizing an application problem into a solver description language
- Solver Domain: SAT-Solver, LP-Solver, Graph Solver,...

Three Realms



Outlook

- Developing deep-learning-driven solvers for general problem classes
- Learning in program space $\rightarrow$ learning in circuit space
- This may lead the way to practical applications of universal induction.