Internship Defense

David Taralla

University of Liège

Thursday 19 December 2013
MCTS algorithm discovery

- Much research in AI games uses MCTS
- Problem known in advance: Customize MCTS in a problem-driven way
- Why not automatize this task?
  ⇒ Monte Carlo search algorithm discovery, for finite-horizon fully-observable deterministic sequential decision-making problems

For example:
- Sudoku puzzles
- Pyramid card game
- ...
Grammar & algorithm space

- Generate a rich space of MCTS algorithms thanks to search components
  - simulate
  - repeat
  - step
  - ...

- Space cardinality grows combinatorially with length and # of search comp.

- Multi-armed bandit approach to get a collection of well-performing algorithms
Multi-armed bandit model

Bandit in this context

- Machine with multiple arms
- Pulling an arm has a budget cost and gives some reward
- Finite budget
Multi-armed bandit model

Model description

Here,

- **Arm** = algorithm execution
- **Reward** = this algorithm execution reward
- **We want the best arm to be the algorithm with the best mean reward**
  i.e. the algorithm performing the best on average
Multi-armed bandit model

Model flaws

- **Discrete**
  One cannot pull half an arm!

- **Big cardinality**
  Existing methods not really adapted to big cardinality with finite budget

- **They used UCB policy with $100 \times \text{#AlgoSpace}$ steps**
  Length up to 5 → #AlgoSpace = 3155: this method is not easily scalable
Multi-armed bandit model
An alternative approach

Design an alternative to standard UCB arm space exploration

- This is the best arm identification problem

- Get info. about pulled arms so far, select next arm accordingly
  ⇒ Perform some kind of information transfer from a (set of) arm(s) to another
  ⇒ This internship was about this problem
Basic idea

- Maximize the “distance” between the pulled arms and the next pull
  Get maximal information $\rightarrow$ Reduce required samples amount!

- Many challenges in this “simple” idea
Best arm identification algorithm

1. Create sampling plan
2. Add resulting data to memory
3. Get a regressor using RLS on data gathered so far
4. Get best arm $a^*$ using predictions
5. Are we confident enough for $a^*$?
   - If No, Prune arm space and get lower & upper confidence bounds
   - If Yes, Return $a^*$
From the idea to the theoretical implementation

Create sampling plan

- **G-optimal** experiment design
  - Concerned with the **variance** of predictions
  - Get allocation vector $\gamma$ s.t. information is, in some way, maximized
    (Erratum — Report says we maximize $J(\gamma)$. That is incorrect, we **minimize** $J(\gamma)$).

- **Simple rounding procedure**
  - “Translate” $\gamma$ into a sequence of arms to pull
From the idea to the theoretical implementation

Get a regressor using RLS on data gathered so far

Predictions?
- Regressor $\theta$
- Features $\Phi$
- $r_a = \langle \phi_a, \theta \rangle = \langle \phi_a, \hat{\theta} \rangle + \eta$
From the idea to the theoretical implementation

Get a regressor using RLS on data gathered so far

- Predictions?
  - Regressor $\theta$
  - Features $\Phi$
  - $r_a = \langle \phi_a, \theta \rangle = \langle \phi_a, \hat{\theta} \rangle + \eta$

- Features of an algorithm
Get a regressor using RLS on data gathered so far

Predictions?
- Regressor $\theta$
- Features $\Phi$
- $r_a = \langle \phi_a, \theta \rangle = \langle \phi_a, \hat{\theta} \rangle + \eta$

Features of an algorithm
- ???
From the idea to the theoretical implementation

Get a regressor using RLS on data gathered so far

Predictions?
- Regressor \( \theta \)
- Features \( \Phi \)
- \( r_a = \langle \phi_a, \theta \rangle = \langle \phi_a, \hat{\theta} \rangle + \eta \)

Features of an algorithm
- ???
- In fact, we just need features to compute \( \hat{r}_a = \langle \phi_a, \hat{\theta} \rangle \)
From the idea to the theoretical implementation

Get a regressor using RLS on data gathered so far

▶ Predictions?
- Regressor $\theta$
- Features $\Phi$
- $r_a = \langle \phi_a, \theta \rangle = \langle \phi_a, \hat{\theta} \rangle + \eta$

▶ Features of an algorithm
- ???
- In fact, we just need features to compute $\hat{r}_a = \langle \phi_a, \hat{\theta} \rangle$
- Features dual: kernels

$$n \text{ arms} \ (\ldots) \Rightarrow \exists \hat{\alpha} \in \mathbb{R}^{n \times 1} :$$

$$\langle \phi_a, \hat{\theta} \rangle = \left\langle \phi_a, \sum_{t=1}^{n} \hat{\alpha}_t \phi_a \right\rangle = \sum_{t=1}^{n} \hat{\alpha}_t \left\langle \phi_a, \phi_{a_t} \right\rangle$$

$$K(a,a_t)$$
From the idea to the theoretical implementation

Get a regressor using RLS on data gathered so far

— Kernels —

The kernel “mimics” the inner product of two feature vectors
From the idea to the theoretical implementation

Get a regressor using RLS on data gathered so far

--- Kernels ---

The kernel “mimics” the inner product of two feature vectors

Estimating $\theta$ → Estimating $\alpha$

Based on features

Get $\hat{\theta}$ → Get $\hat{r}_a$

Based on kernel

Get $\hat{\alpha}$ → Get $\hat{r}_a$
From the idea to the theoretical implementation

Get a regressor using RLS on data gathered so far
— Kernels —

The kernel “mimics” the inner product of two feature vectors

Estimating $\theta$  
Based on features  
Get $\hat{\theta} \rightarrow$ Get $\hat{r}_a$

Estimating $\alpha$  
Based on kernel  
Get $\hat{\alpha} \rightarrow$ Get $\hat{r}_a$
From the idea to the theoretical implementation

Get a regressor using RLS on data gathered so far

— Regularization parameter $\lambda$ —

- **Auto tuning of $\lambda$ given dataset**

  Minimize $e(\lambda) = \frac{1}{n} \sum_{i=1}^{n} (f_{D-i, \lambda(a_i)} - r_i)^2$

- **Naïve approach:**
  1. Get $\hat{\alpha}$ — $O(n^3)$ (1 matrix inversion)
  2. Do it for $n$ different datasets — $O(n)$

    $\Rightarrow$ If $M$ evaluations of $e(\lambda)$, total complexity of $O(Mn^4)$!

- **Kernelized generalized cross-validation**

  $\Rightarrow$ If $M$ evaluations of $e(\lambda)$, achievable total complexity of $O(n^3 + Mn^2)$
From the idea to the theoretical implementation

Get a regressor using RLS on data gathered so far

— Regularization parameter $\lambda$ —

Example

Mean error when predicting the mean reward of an algorithm
From the idea to the theoretical implementation

- Theorem developed by Abbasi-Yadkori et al. (2011)
- Extension to the kernel case by Abbasi-Yadkori (2012)
- Given some assumptions on the model, allows to compute the (symmetrical) bounds
From the idea to the theoretical implementation

- Discard all arms whose *upper bound* is smaller than the *lower bound* on $a^*$
- Illustration [on the board]
Conclusion

Wrap up: Sudoku $16 \times 16$

Maybe a little wrap-up example?

Data

- Problem: $16 \times 16$ Sudoku, $\frac{1}{3}$ prefilled grid
- About $3200$ algorithms
- $2$ rounds with sampling plans consisting of sequences of $n_1$ and $n_2$ algorithms
Conclusion

**Wrap up: Sudoku 16 × 16**

- Create sampling plan
- Add resulting data to memory
- Get a regressor using RLS on data gathered so far
- Get best arm $a^*$ using predictions
- Are we confident enough for $a^*$?
- Prune arm space
- Get lower & upper confidence bounds

- Return $a^*$
Conclusion

This internship in a nutshell

► 1 month of preparation
  - Implement MCTS algorithms generation & execution
  - C++ was used
  - 1 week to implement, more than 3 weeks to debug

► 2 months in RLAI lab
  - Create a dataset thanks to Westgrid network
  - Design, implement and check correctness of each parts of this new approach
  - Sadly not enough time to do significant comparisons

► Half a month to complete and re-read report
Conclusion

Thank you for your attention

Special thanks to my mentors for making this internship possible.