

Deephack.RL

5vision solution

25 February 2017

Team



Maksim Kretov



Mikhail Pavlov



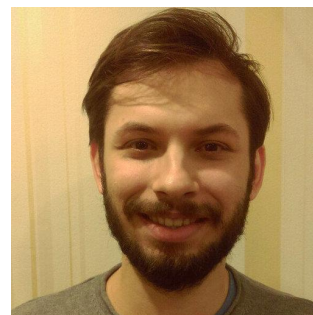
Alexey Seleznev



Alexander Fritsler

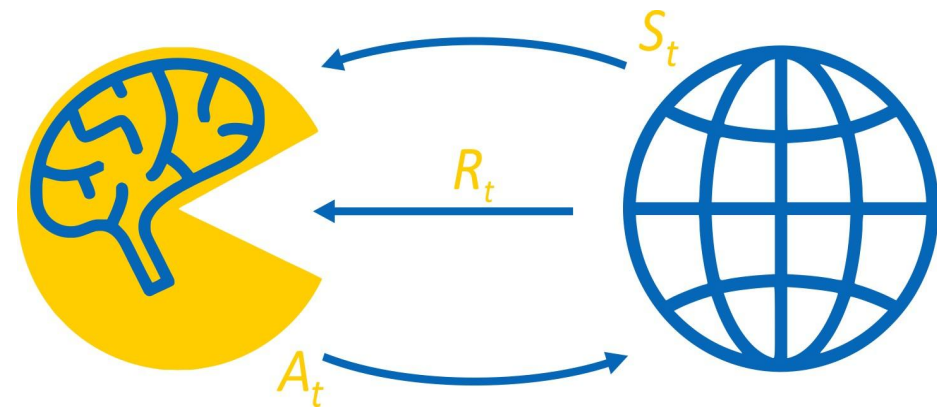


Vlad Zhukov



Artem Sorokin

About competition



Challenge:

write agent to play ATARI games

Some important points from rules:

- training algorithm should be same for all games.
- there could be meta-agent i.e. agent that selects best among other agents.
- the model of environment is available during training (we can save env state and go back to this state at any time) but not during testing.

Team plan for hackathon

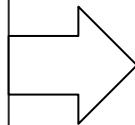
Base algorithms:

New algorithms:

- planning + supervised learning (SL)
- policy gradient q-learning
- trust region policy optimization
- strategic attentive writer

Backup algorithm:

- a3c algorithm (+ expl. bonus)



Bandit algorithm:

choose best algorithm for each game

Team work:

- each team member is working independently of the others
- everyone can try anything he wants
- everyone dives into his topic
- some very fresh algorithms have been tried (at least tried to be implemented)

Planning + supervised learning

Also known as **imitation learning**

Deep Learning for Real-Time Atari Game Play Using Offline Monte-Carlo Tree Search Planning

Xiaoxiao Guo
Computer Science and Eng.
University of Michigan
guoxiao@umich.edu

Satinder Singh
Computer Science and Eng.
University of Michigan
baveja@umich.edu

Honglak Lee
Computer Science and Eng.
University of Michigan
honglak@umich.edu

Richard Lewis
Department of Psychology
University of Michigan
rickl@umich.edu

Xiaoshi Wang
Computer Science and Eng.
University of Michigan
xiaoshiw@umich.edu

Algorithm:

- 1) Run planning algorithm on episodes to get dataset (screen, best_action)
- 2) Train CNN policy as classification task with cross entropy loss with

Some facts about article:

- Article was presented at NIPS 2014
- Classification as well as regression task (for q-val)
- Deterministic version of ALE
- Upper Confidence Bound 1 applied to trees (UCT) as planning
- DAGGER algorithm for data collection

Why UCT as planning?

- UCT + SL worked before for Atari games
- Best algorithm in original ALE paper (very high scores compared to other methods)
- Widely used in games

Game	Full Tree	UCT	Best Learner	Best Baseline
Asterix	2136	290700	987	650
Pacman	1709	22336	1691	506
Centipede	125123	110422	8804	16527
Times Best	4	45	3	3

Marc G. Bellemare et. al. The Arcade Learning Environment: An Evaluation Platform for General Agents

UCT for deterministic environment

UCT is monte carlo tree search (MCTS) with special form of exploration bonus

generic MCTS sketch



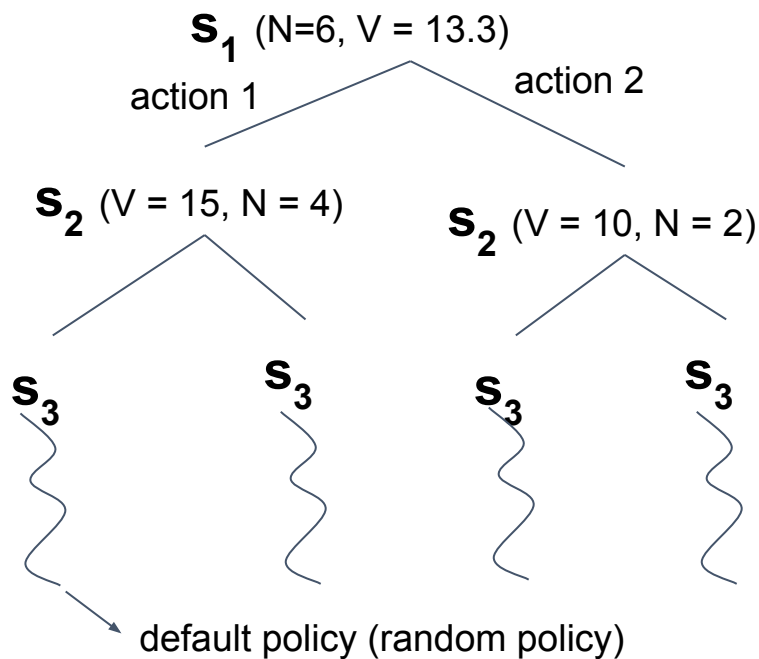
1. find a leaf s_l using $\text{TreePolicy}(s_1)$
2. evaluate the leaf using $\text{DefaultPolicy}(s_l)$
3. update all values in tree between s_1 and s_l

take best action from s_1

UCT $\text{TreePolicy}(s_t)$

if s_t not fully expanded, choose new a_t
else choose child with best $\text{Score}(s_{t+1})$

$$\text{Score}(S_{t+1}) = V(S_t) + C \sqrt{\frac{\ln N(s_{t-1})}{N(s_t)}}$$

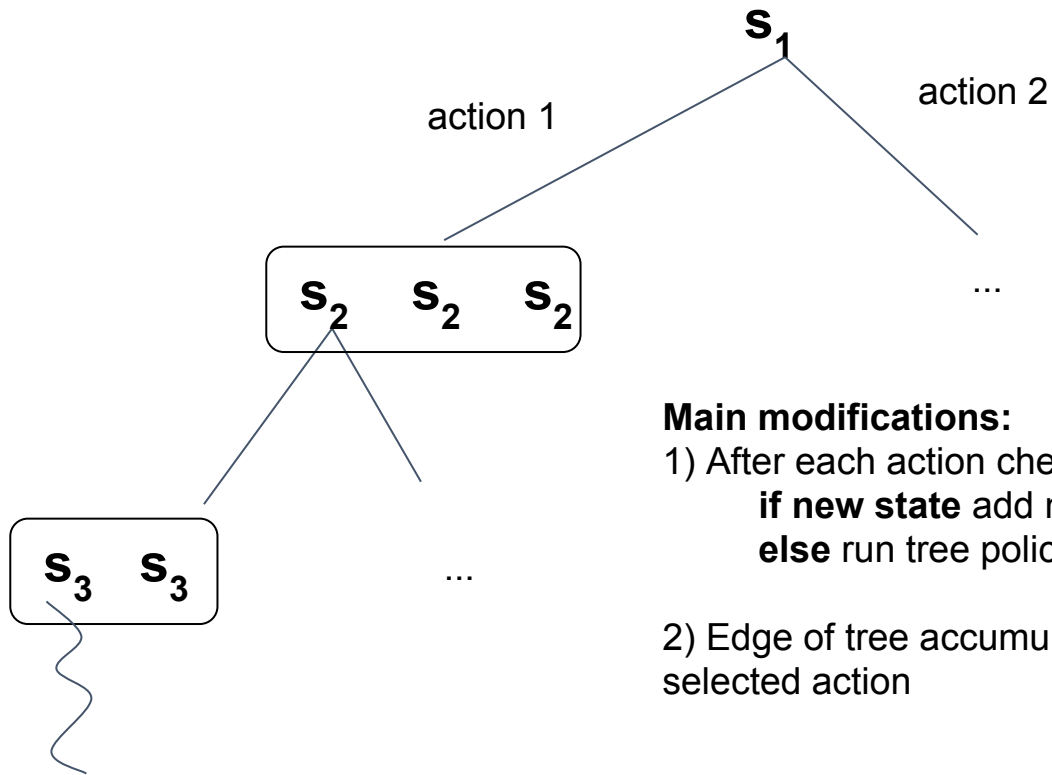


the same scheme works for stochastic environment. s_t in this case can represent several game states (abstract tree node)

UCT for stochastic environment. 5vision bicycle

2 source of stochastic behavior:

- 1) each action is repeated 2-5 times
- 2) 25% chance to ignore current action and repeat previous



Main modifications:

- 1) After each action check if we already have been in resulted state:
if new state add new node to tree and run default policy
else run tree policy
- 2) Edge of tree accumulates visits and values of all child nodes for selected action

UCT problems

Problem 1

Universal value of C in score formula
(because of various reward range fo different games)

$$Score(S_{t+1}) = V(S_t) + C \sqrt{\frac{\ln N(s_{t-1})}{N(s_t)}}$$

Soltion:

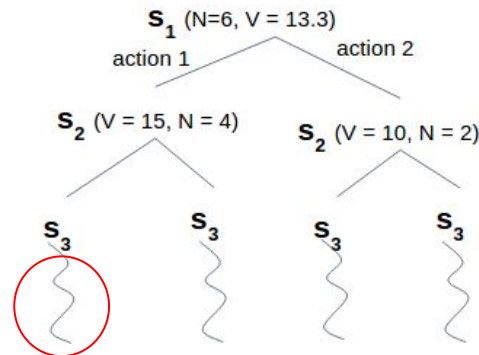
C = sqrt(2) and normalization of $V(S_t)$ for all actions in range 0-1 worked well

Problem 2

Length of rollout in default policy.
(do not want to make rollout till the end because time consuming)

Soltion:

Analyze reward distribution through time with random policy:
if most reward in the end do rollouts till the end of episode
else do 100 steps rollouts

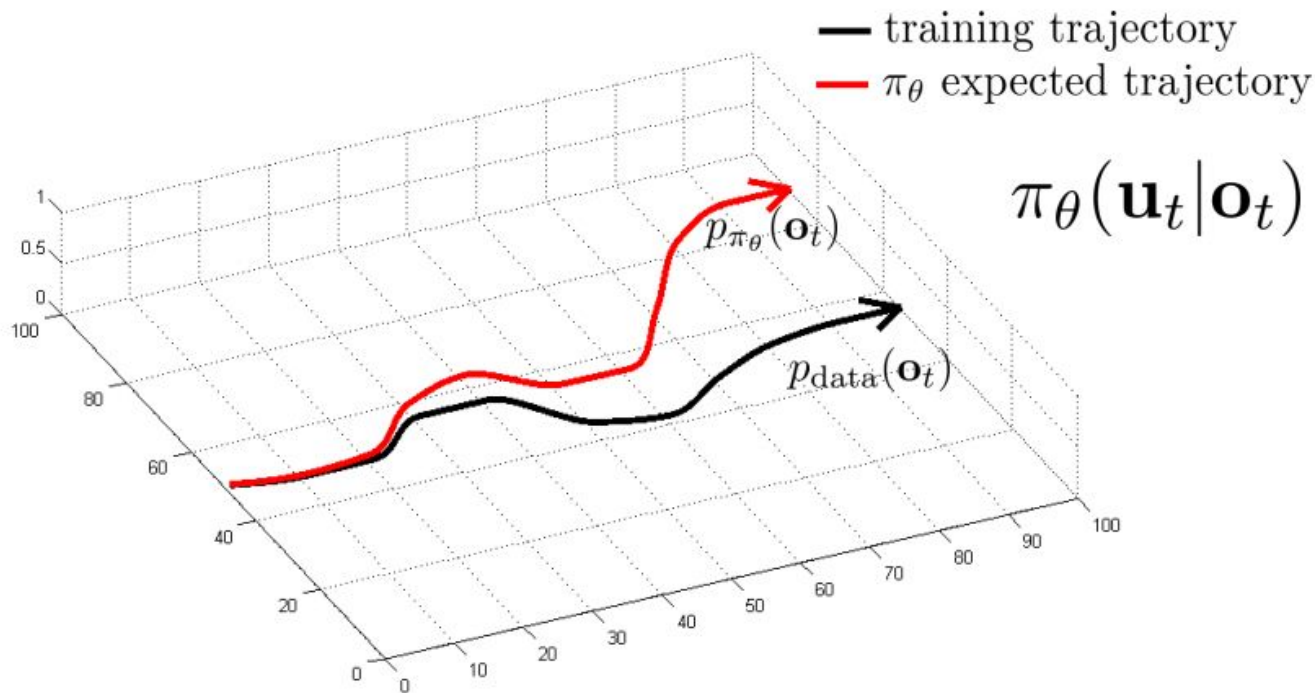


Supervised learning

- Usual CNN architecture (better results with openAI universe-starter-agent architecture)
- Training with cross-entropy loss (other losses were tested also)
- During testing better worked argmax not sampling
- Better results with under trained model (accuracy not good metric in that case)
- Better results with image augmentation
- Better results with making prediction on both original and flipped image then averaging

Supervised learning problems

UCT trajectory differs from SL trajectory / training states distribution differs from test states distribution




Supervised learning problems

Problem:

UCT trajectory differs from SL trajectory / training states distribution differs from test states distribution

Right solution:

use DAGGER algorithm (replace human to UCT below)

- 
1. train $\pi_{\theta}(\mathbf{u}_t|\mathbf{o}_t)$ from human data $\mathcal{D} = \{\mathbf{o}_1, \mathbf{u}_1, \dots, \mathbf{o}_N, \mathbf{u}_N\}$
 2. run $\pi_{\theta}(\mathbf{u}_t|\mathbf{o}_t)$ to get dataset $\mathcal{D}_{\pi} = \{\mathbf{o}_1, \dots, \mathbf{o}_M\}$
 3. Ask human to label \mathcal{D}_{π} with actions \mathbf{u}_t
 4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_{\pi}$

from deep RL course at berkley lecture 2

OUR solution:

Get MOAR data:

- flip frames and action vertically (increase sample size by 2)
- run a lot of parallel workers to get more trajectories (they will be different because environment is stochastic)
- image augmentations

dataset sizes: skiing 100k; pacman 700k; centipede 200k

Other ideas for UCT+SL

- train RNN (include prev act in state) + CNN (implemented, but model.load failed in keras)
- DAGGER (implemented, but not used)
- use trained policy as default/rollout policy (implemented, but not used - too slow)
- train value function to make shorter rollouts (not implemented)

Results

Game	Num actions	UCT score av/best	SL score av/best	A3C score av/best
Skiing	3	-5000/-3500	-4500/-3500	-9013/-9013
Pacman	8	7000/20000	2700/6700	1800/4000
Centipede	18	>500000/>500000	8000/12000	17130/17130

Final solution:

for skiing and pacman: uct + sl model

for centipede: a3c

Final result:

second place in playoff

second place in games average nomination

Code

UCT + SL:

https://github.com/5vision/uct_atari (all logic in file uct.py)

A3C (+exp bonus):

https://github.com/5vision/a3c_theano

policy gradient q-learning (pgq):

<https://github.com/Fritz449/Asynchronous-RL-agent>

UCT policy video

Pacman:

<https://drive.google.com/open?id=0B5TZszBC9SNkYU1NVIFjdEY2MUE>

Centipede (manually terminated):

<https://drive.google.com/file/d/0B5TZszBC9SNkSnQyYnVfRTVlbms/view?usp=sharing>

Thanks for attention