Reinforcement Learning with A* and a Deep Heuristic

Ariel Kesleman
Imagry
ariel@imagry.co

Sergey Ten
Imagry
sergey@imagry.co

Adham Ghazali
Imagry
adham@imagry.co

Majed Jubeh
Imagry
majed@imagry.co

(in prep.)

Vision-based autonomous driving

- ✗ HD mapping
- ✗ LIDAR
- ✗ Radar
- ✓ Cameras

A* is a path finding algorithm

- Finds the minimal cost path from point A to point B
- It uses domain knowledge, a so called Heuristic function **H** (**H** = an estimated cost to target)

1) Select starting node
2) List possible actions to take from current node
3) Push node and actions to a priority queue: **P** = cost + **H**
4) Pop bottom action, has minimum **P**
5) Take that action, it gets us to a new node
6) If reached target brake else goto 2

A* is special

If the Heuristic is consistent and admissible:

- Gives the absolute minimal path,
- While visiting the minimal number of nodes

Given the same domain knowledge, no other algorithm can do better!

Admissibility = never over-estimating cost to target
Consistency = estimated cost to move from x to y (by **H**) < real cost to move from x to y

Hart, P. E.; Nilsson, N. J.; Raphael, B. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". IEEE Transactions on Systems Science and Cybernetics SSC4

Shakey The Robot, 1966-1968

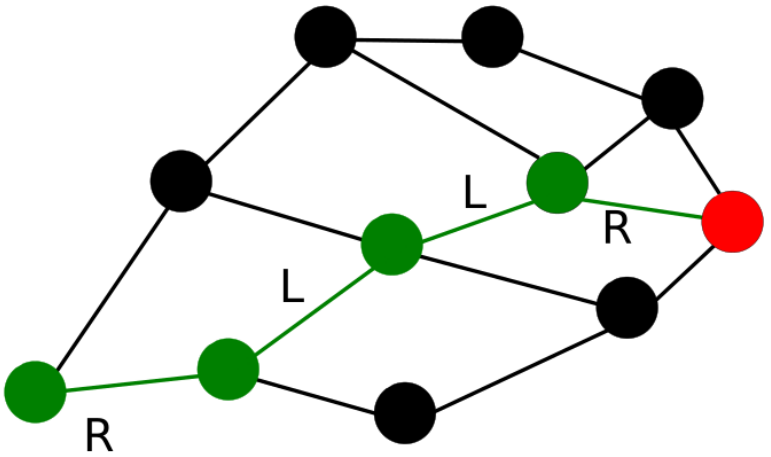
A1 → A2 → A*

because whatever version of the algorithm that comes after cannot be any better

An interpretation of path finding algorithms - 1

- Any algorithm that finds a path in a graph generates a sequence of actions

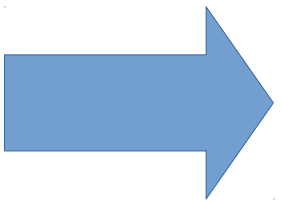
R → L → L → R → arrived!



- Lets define reward (**R**) as 1/cost
- Then, any algorithm that minimizes accumulated cost maximizes accumulated reward

Path finding algorithms

- Generate a sequence of actions
- Maximize reward



They should be good to solve MDPs ?

MDPs

- A mathematical framework to work with decisions
- Goal is to maximize reward
- Very useful:
  - Driving,
  - Atari games,
  - Basically any agent interacting with an environment.
- Solved by dynamic programming, Reinforcement Learning

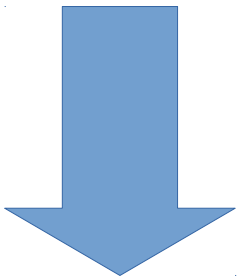
Can we use A* to solve MDPs?

A* has a problem!

There are may domains where a heuristic is not available

The New Idea

- DQN (an RL algorithm) is also a good solver of MDPs
- DQN learns expected future rewards… a bit like **H**



- Use a Deep-Q-Network as the A* heuristic → a learnable heuristic

- Coupling tree methods with deep networks was proven succesful, e.g. AlphaZero from DeepMind

Wasn’t this done before?

- No.
- Why not?
- Because A* has a problem. It has no good heuristic to solve many problems, like the game of Go.
- MCTS was invented as a mothod that could tackle Go in the 80’s.
- MCTS doesn’t need a heuristic
- MCTS continued to be used successfully where A* cannot, and continued to be used in modern methods like AlphaZero from DeepMind

Wasn’t the difference between A* and MCTS?

MCTS is a completely different beast:

- While it converges to the best actions, it doesn’t visit a minimal number of nodes, even if combined with a good heuristic.
- It cannot do so because it doesn’t track accumulated reward, only discounted future reward
- MCTS is stochastic, it needs stochasticity to perform well.

  There is nothing inherently stochastic about A*

  (altough we do add a DQN-like e-greedy exploration)

subroutine: Build a tree

- uses a Heuristic Network:
  - input: sensors (calculated from environment and state)
  - output: action values (Q, expected discounted future reward per action)
- Prepare initial node (nodes contain state and not-discounted accumulated reward)
- Calculate sensors (from node state and environment)
- Evaluate Heuristic
- Push to priority queue node and actions with **P** = accumulated reward + **Q(a)**
- Pop top from queue (max **P**)
- From corresponding node take corresponding action, get a new **node**, **reward** and **done**

subroutine: backpropagate Q

- Uses an existing tree (a list of nodes)
- For each action **A** define the number of visits **V** as its total number of sub-nodes
- Lets define `**a**` as the children actions of **A**
- Define the **value** of node **n** as the weighted average of its actions:

  value(n) = sum(**V**(**a**) * **Q**(**a**)) / sum(**V**(**a**))
- For all actions have **Q**(**A**) = reward + gamma * **value**(**n**)
- (gamma is the discount factor)
- For `done` node actions set **Q**(**A**) = 0

subroutine: Accumulate experiences

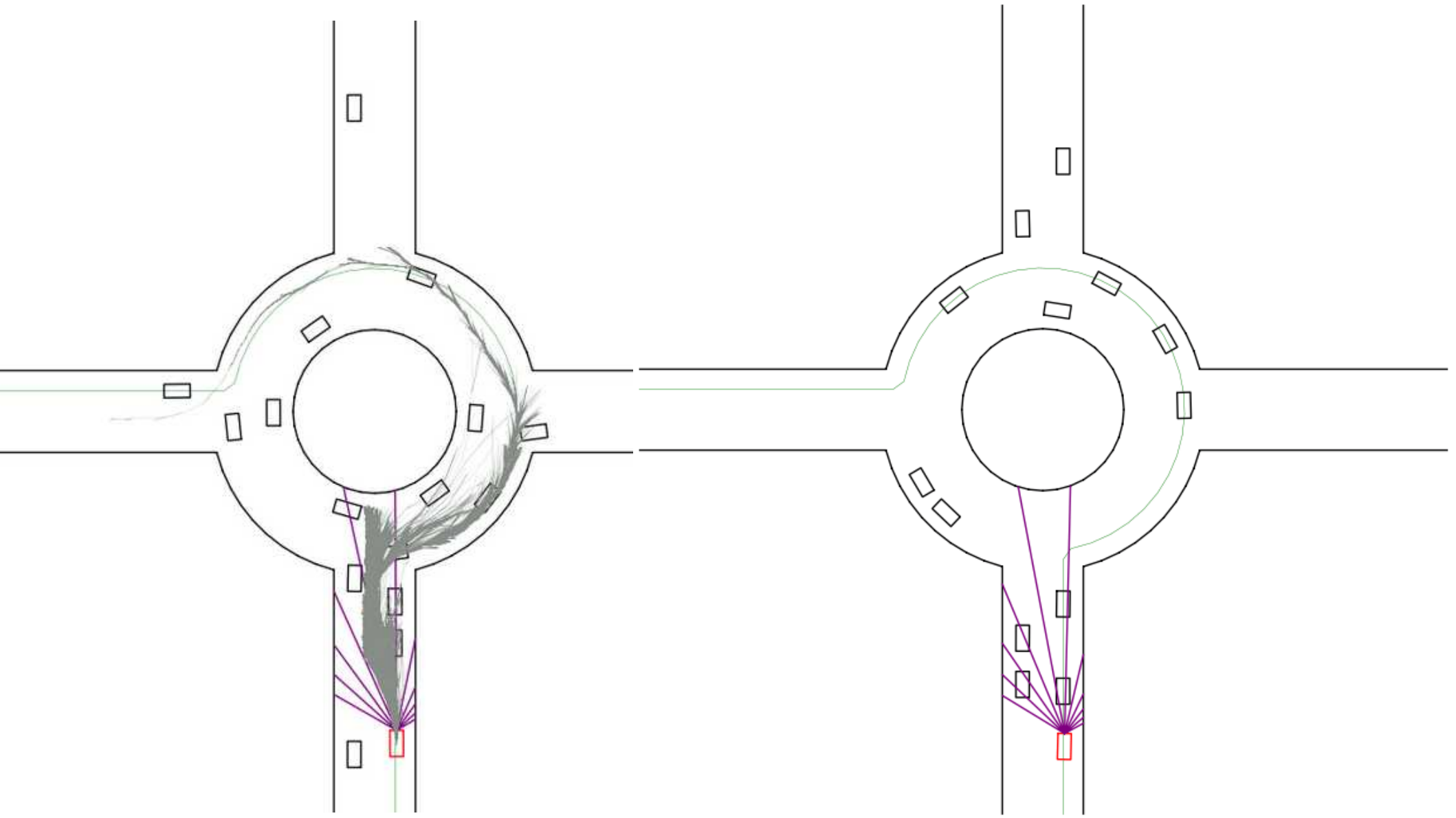
- Uses a backpropagated tree, uses a list of tuples (Q, sensors, priority)
- Calculate the median priority
- For all node in the tree:
  - Push Q (a vector with len = number of possible actions), sensors and the median priority
  - If length of list is too big trim it from the beginning

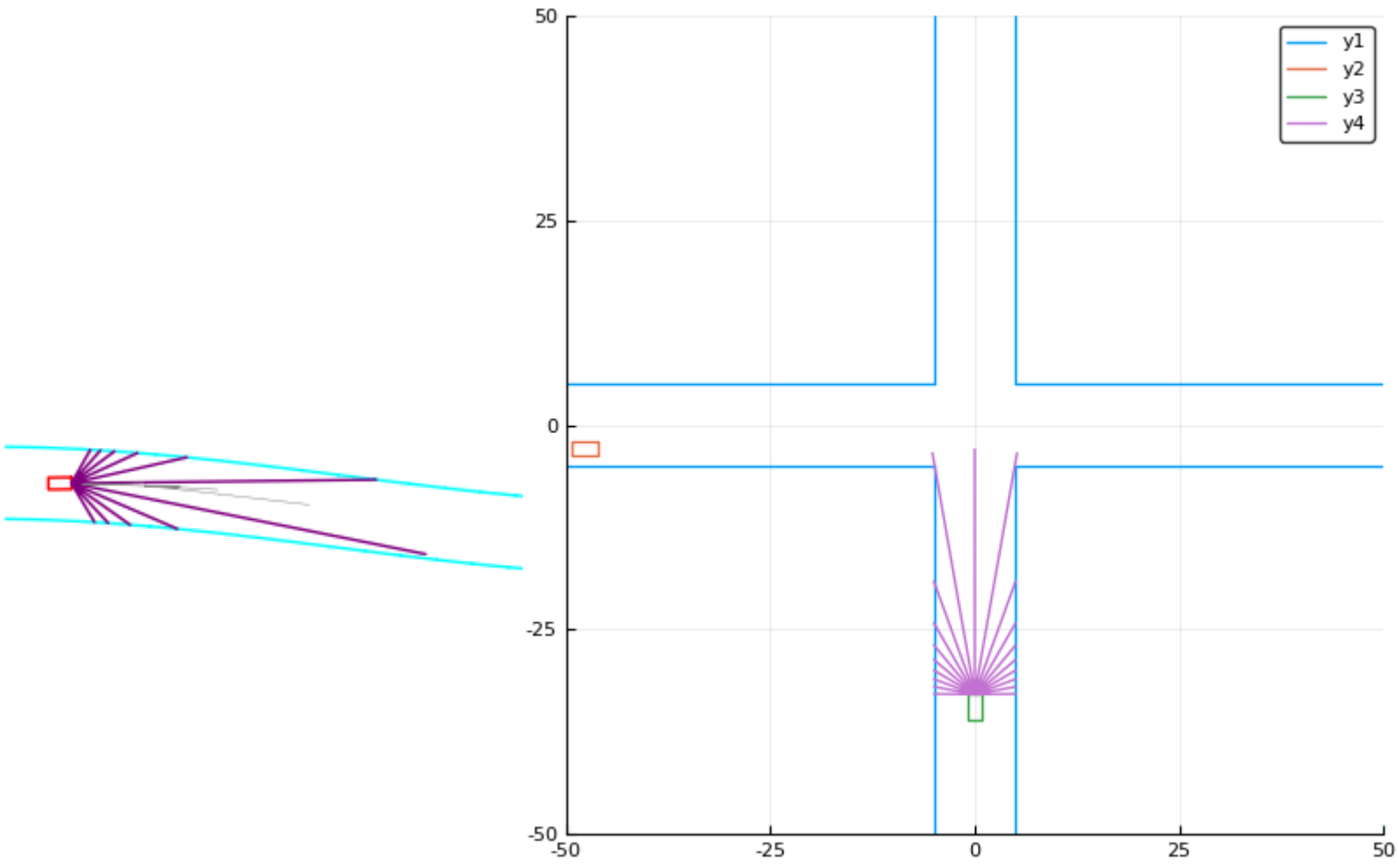
subroutine: Train iteration

- Uses an experience replay buffer
- Build a random batch X=sensors, Y=Qs from the replay buffer
- The probablity of choosing a sample is proportional to the priority
- Train on batch
- For all samples in batch set priority = loss ^ alpha.

The full algorithm

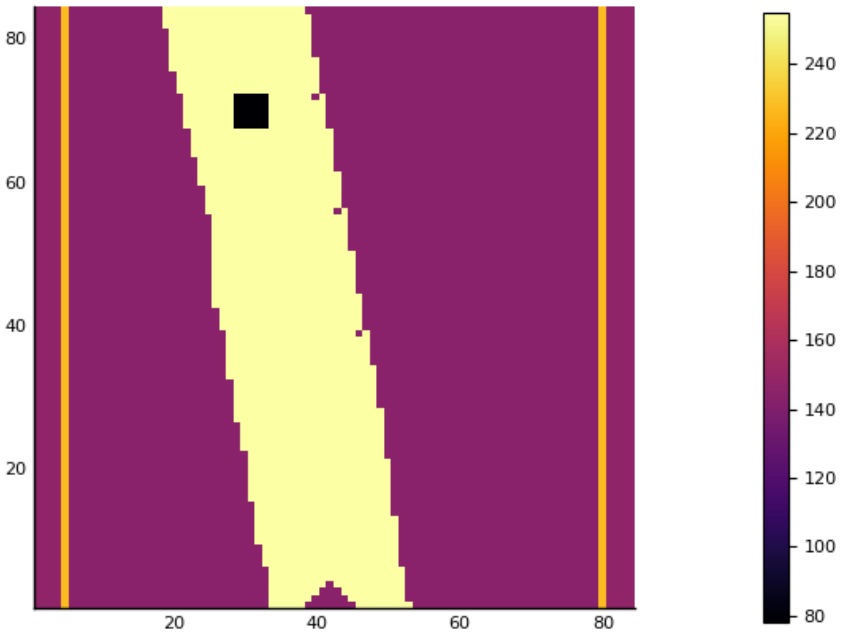
1) Initialize:
   - replay buffer
   - heuristic network
   - environment
   - state
2) Build a tree
3) Backpropagate the tree
4) Accumulate experiences
5) Train
6) Goto 1




y1
y2
y3
y4
50
25
0
-25
-50
-50
-25
0
25
50

Can it work on images?

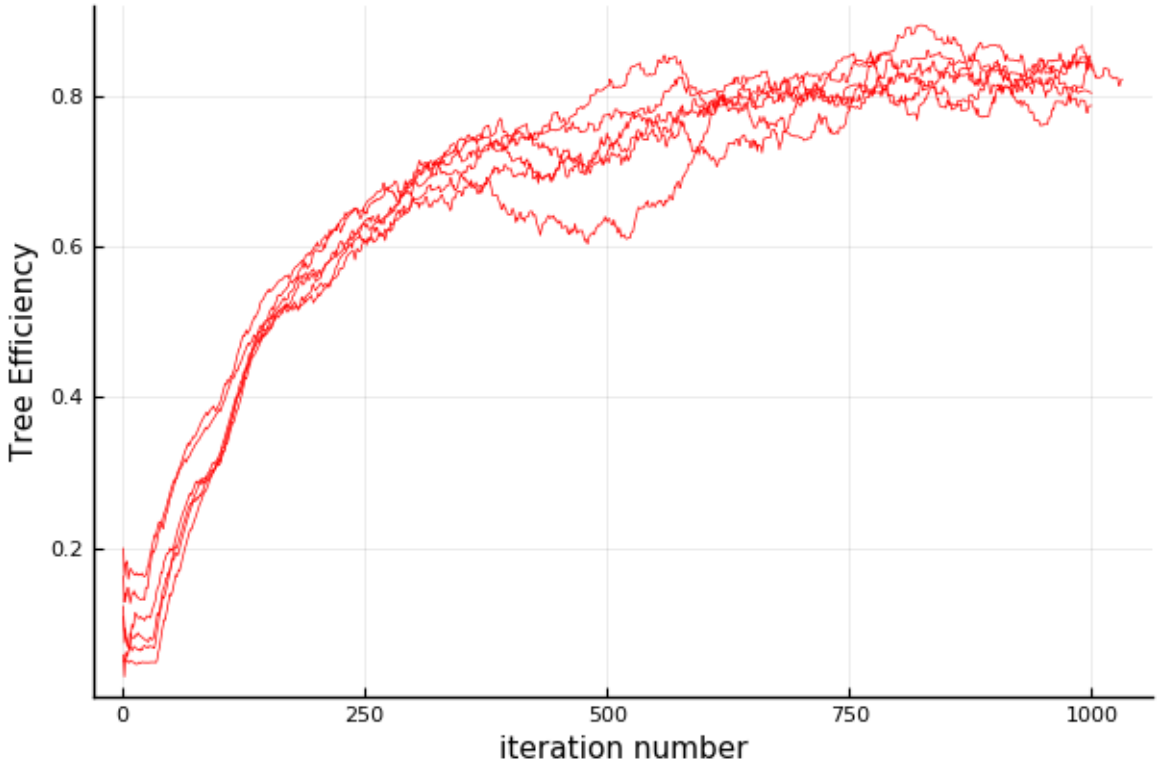
- Sure! We are releasing such an environment for you to play and reproduce the results presented here:



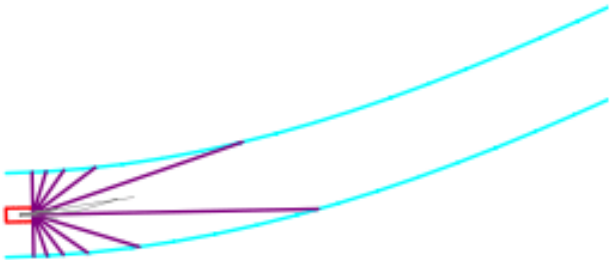
- https://github.com/imagry/aleph_star

Are you using the tree on runtime?

- Good question!
- Sometime yes, sometimes not
- Because we deal with A*, an efficient method guided by a learned heuristic, the tree can be very efficient!



Efficient tree at runtime

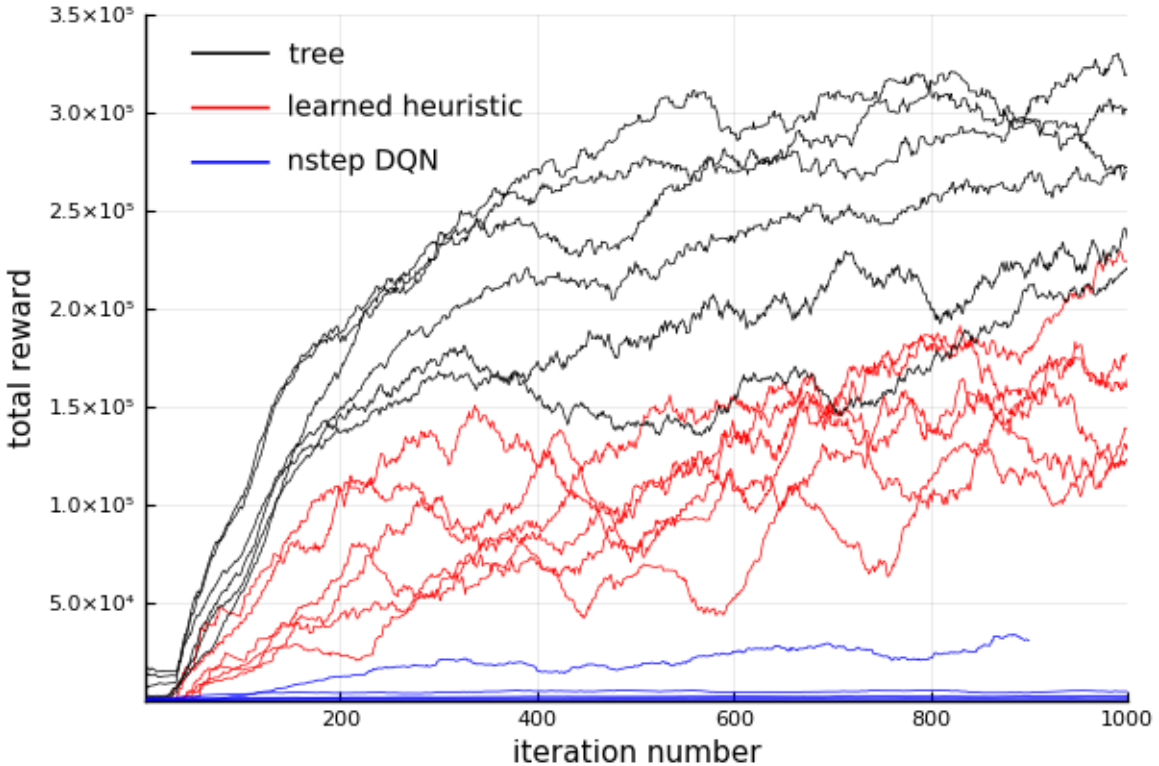


This works with a tree
also on pixel input

Is it any better than DQN?

Yes. But beware that doing an apples to apples comparison is not trivial. Anyway, I did my best and compared with n-step DQN which is hard to beat

Training takes ~ 2hrs on my laptop GTX-860M



Future Work

- A proper comaprison to MCTS:
  - Training time
  - Validation performance

  this is not trivial, AlphaZero is fit for reward at the end (win or loose) it has to be modified to fit a continuous reward, there may be several ways to do this.

  Also AlphaZero is expensive to train: the network predicts state value, backpropagation requires evaluating all leafs. It was trained in 3 days on 5K TPUs

- More complex environments

Summary

- Presented a novel RL algorithm based on A* and a learnable heuristic
- Demonstrated efficient training on images on a laptop
- Demonstrated efficient tree on runtime on image input

Thank you for listening!

- Questions?
-