Learning to Search with MCTSnets

Minghan Li
Ignavier Ng

Motivation of MCTSnet

- MCTS is non-differentiable, which is difficult to optimize

- Keep algorithmic skeleton of MCTS, identify subcomponents, parametrize and optimize them
    - The functions of components are given by how they are reused across the model

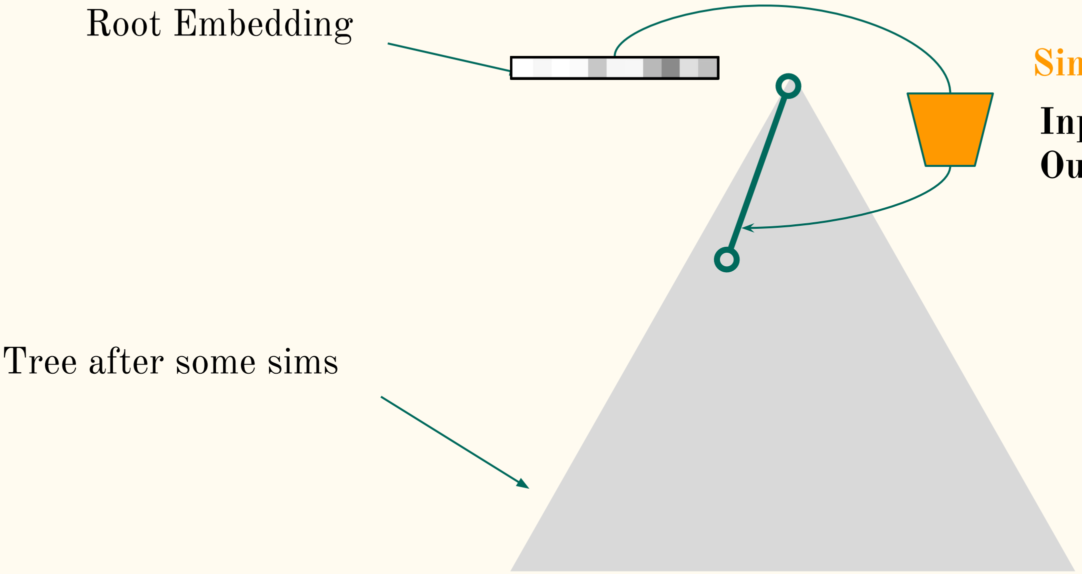
- Train end-to-end to optimize chosen loss function
    - Hope to get better results with fewer simulations than MCTS

Difference Between MCTS and MCTSnet

	MCTS	MCTSnet
Statistics	Q estimation	state embedding h
Simulation policy	UCT formula	policy network π
Leaf value estimation	Rollout/Value network	embedding network ε
Backup phase	Monte-Carlo return	back-up network β
Action selection	Most visited node	readout network ϱ

MCTSnet parametrizes each of the subcomponent using neural networks

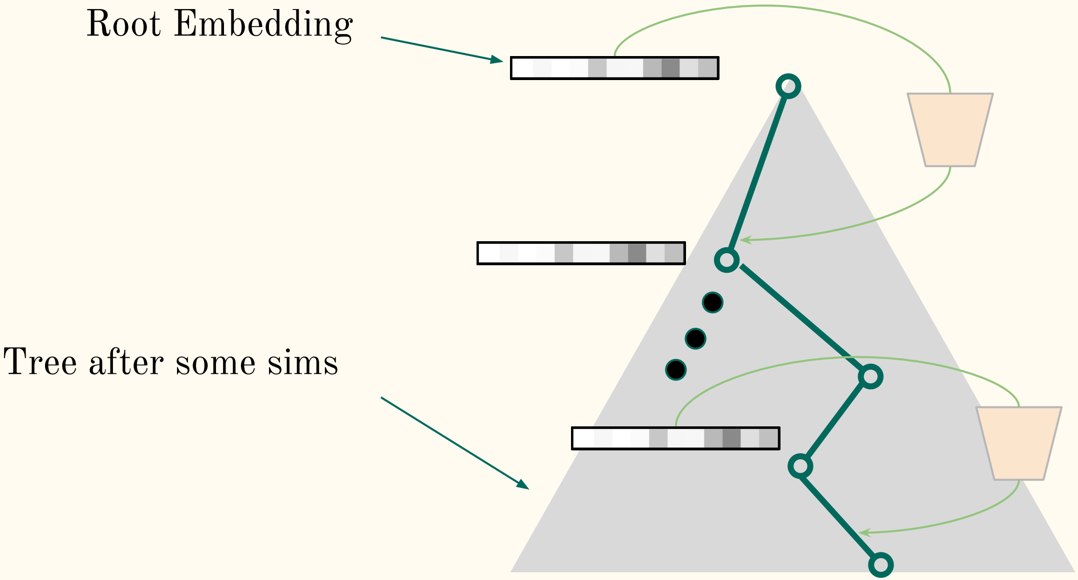
MCTSNet: A Single Simulation (Tree-Policy Phase)



Simulation Policy Network

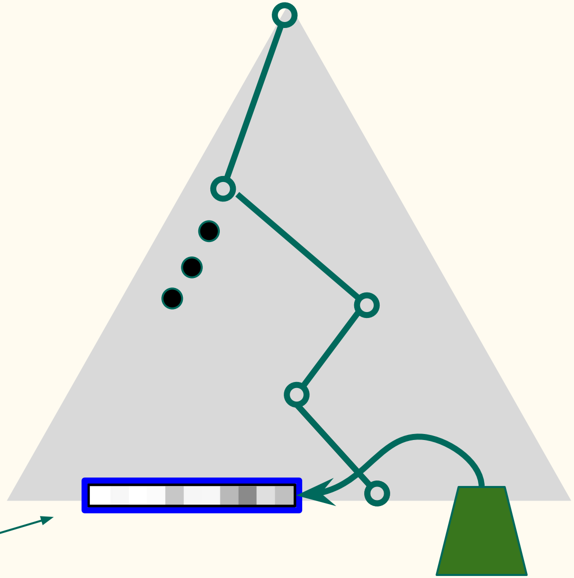
Input : State embedding h
Output : Sampled action

MCTSNet: A Single Simulation (Tree-Policy Phase)



Using true model for each transition

MCTSNet: A Single Simulation (Tree-Policy Phase)

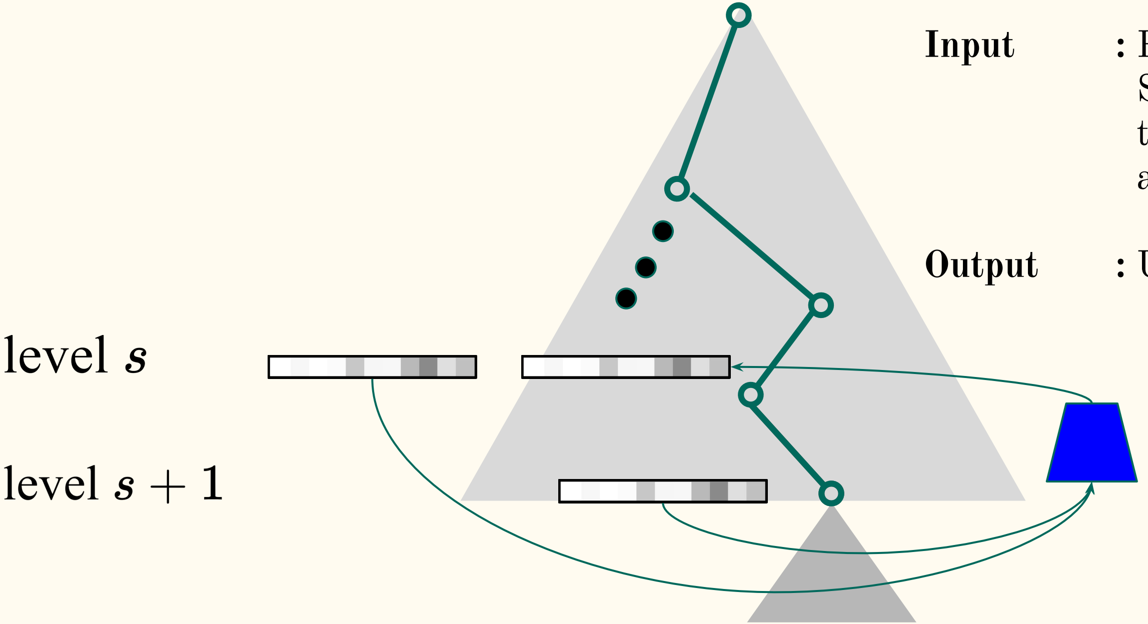


Leaf node
(hasn't been explored yet)

Embedding Network

Input : Game frames (raw state)
Output : State Embedding h

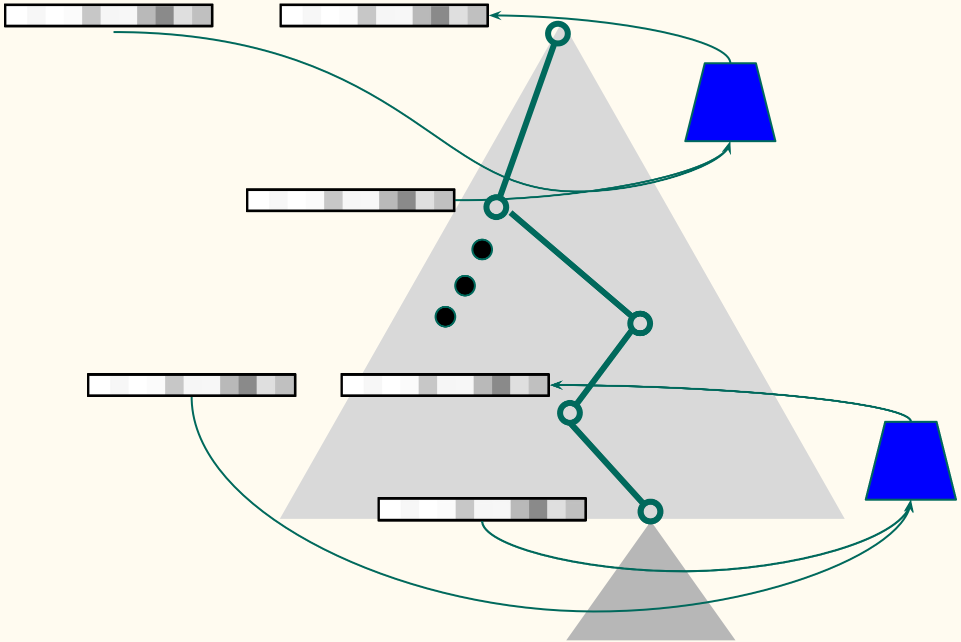
MCTSNet: A Single Simulation (Backup Phase)



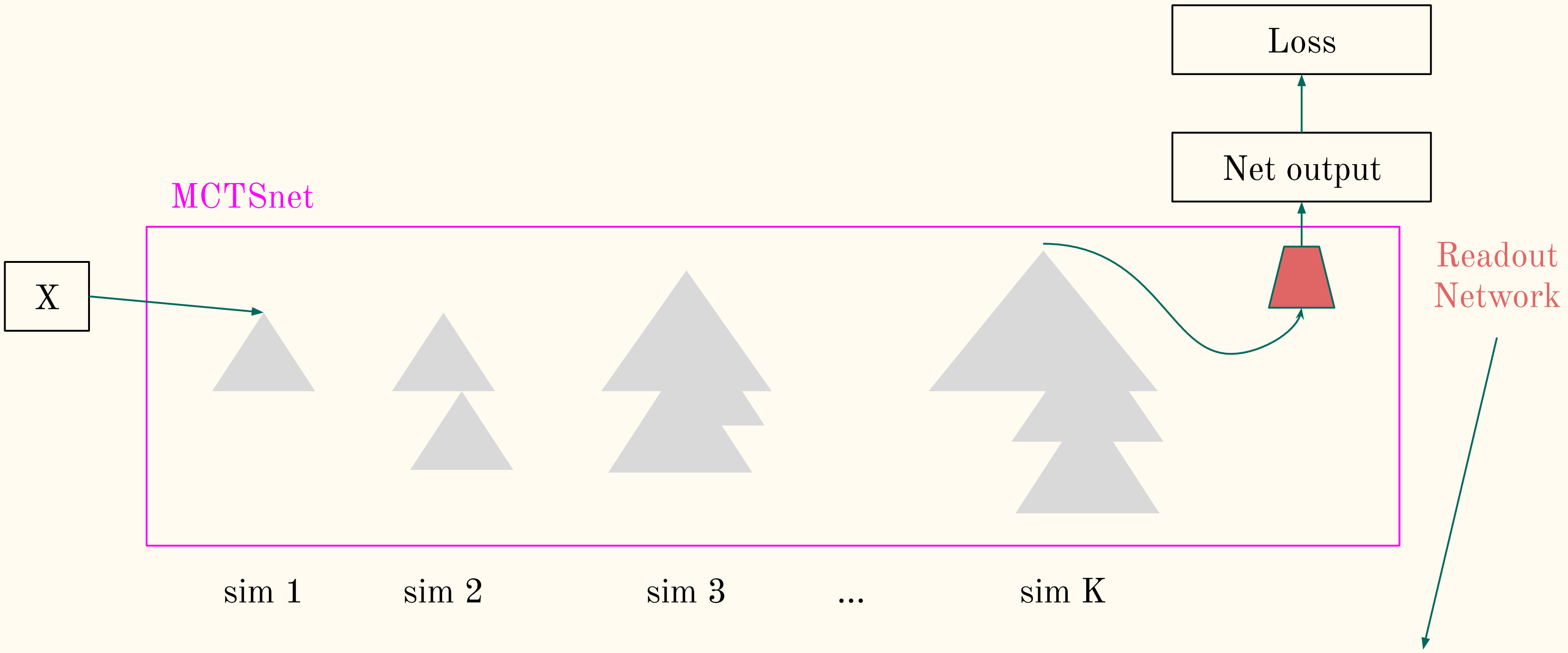
Input : Previous state embedding h_s^t
State embedding of child-node h_{s+1}^{t+1}
true reward r
action a

Output : Updated state embedding h_s^{t+1}

MCTSNet: A Single Simulation (Backup Phase)



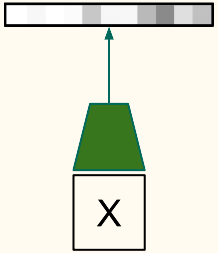
Multiple Simulations/Search



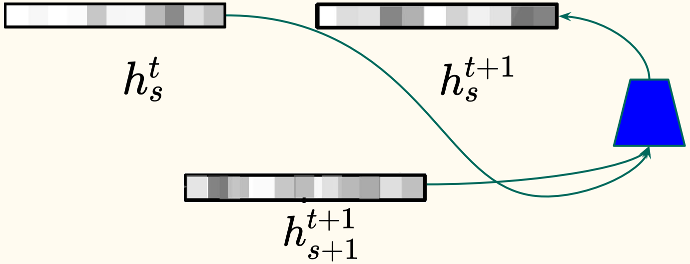
Input : state embedding of root
Output : Action distribution

Recap of MCTSnet Modules

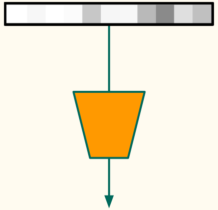
h_s^t stands for embedding h at level s of the tree in the t th simulation



Embedding network



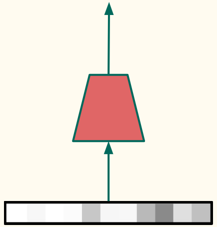
Backup network



action probability

Simulation policy network

action probability



Readout network

Difference Between MCTS and MCTSnet

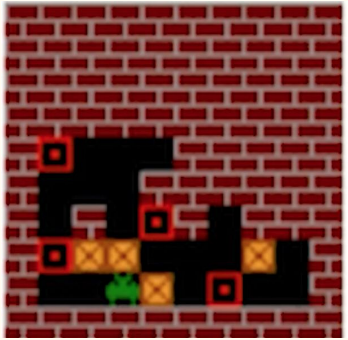
	MCTS	MCTSnet
Statistics	Q estimation	state embedding h
Simulation policy	UCT formula	policy network π
Leaf value estimation	Rollout/Value network	embedding network ε
Backup phase	Monte-Carlo return	back-up network β
Action selection	Most visited node	readout network ϱ

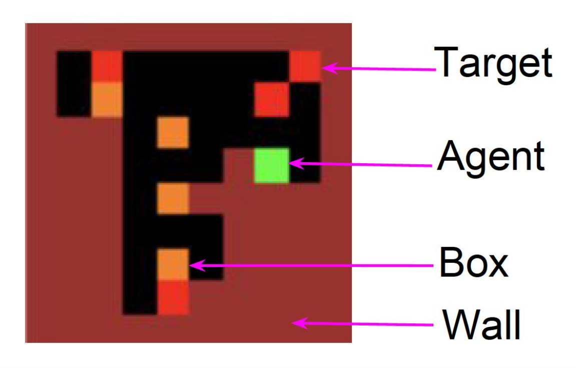
Problem Setting

Goal : Push the box onto the red targets but not pull (non-ergodic)

Input : x - game frames

Target : a* - “oracle” action (obtained from running a large scale MCTS)





Loss for a single step (M simulations)

Cross-entropy loss between the readout network's output and ground truth action:

Raw game frames

$$l(x) = E_z\left[-\log p_\theta(a^*|x, z)\right]$$

A set of all actions taken in the simulation

Gradient of the loss splits into **differentiable** and **non-differentiable** parts.

$$\nabla_\theta l(x) = -E_z\left[\underbrace{\nabla_\theta \log p_\theta(a^*|x, z)}_{\text{Standard backprop}} + \underbrace{\left(\sum_i \nabla_\theta \log \pi_s(a_i|H_i)\right)\underbrace{\boxed{\log p_\theta(a^*|x, z)}}_{\text{pseudo-reward}}}_{\text{REINFORCE}}\right]$$

Credit Assignment Technique

The REINFORCE term of equation (9), $-\nabla_\theta \log \pi(\mathbf{z}|s;\theta_s) \log p_\theta(a^*|s,\mathbf{z}) \triangleq A$, can be rewritten:

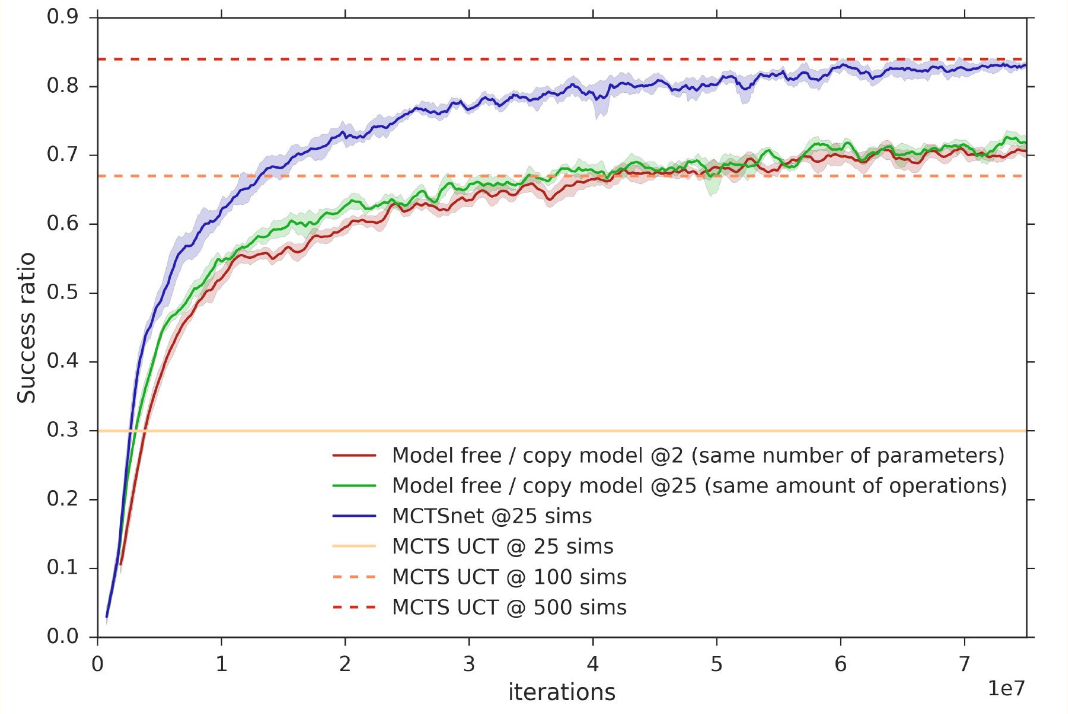
$$A = \sum_m \nabla_\theta \log \pi(z_m|s;\theta_s) R_1. \tag{10}$$

Since stochastic variables in z_m can only affect future rewards $r'_m, m' \geq m$, it follows from a classical policy gradient argument that (10) is, in expectation, also equal to:

$$A = \sum_m \nabla_\theta \log \pi(z_m|s, z_{<m};\theta_s) R_m \tag{11}$$

$$= -\sum_m \nabla_\theta \log \pi(z_m|s, z_{<m};\theta_s)(\ell_M - \ell_{m-1}). \tag{12}$$

Results: Contribution of Tree Search



Model Free vs Model Based

	Model Based	Model Free
Transition Function	T(s,a) = s’	T(s,a) = s
Reward Function	R(s,a) = r	R(s,a) = 0

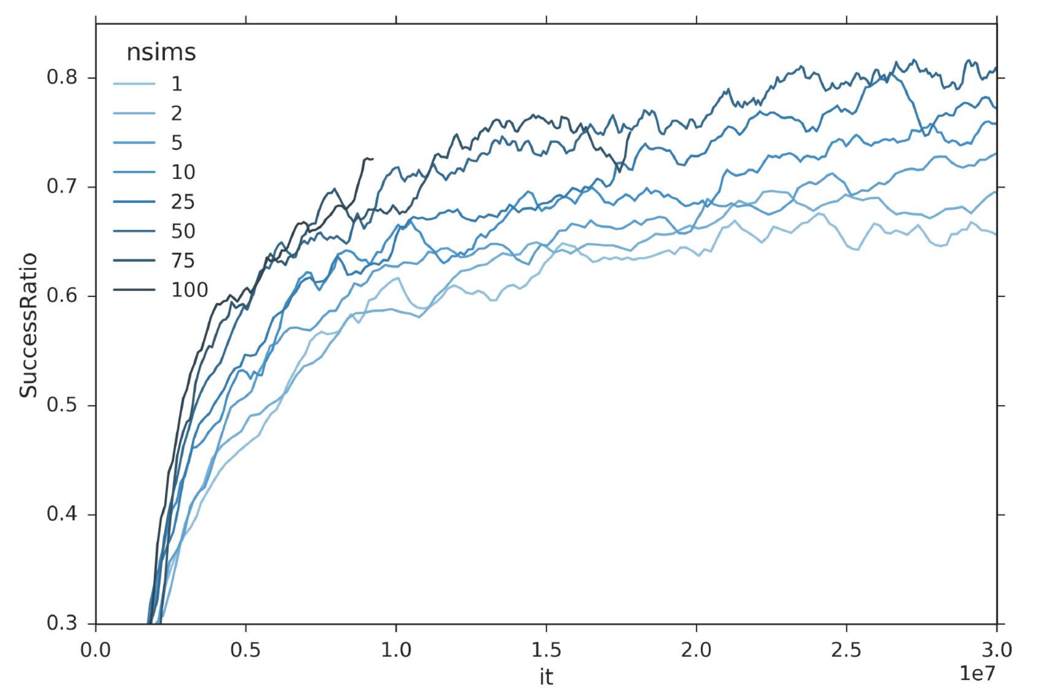
Model Free vs Model Based

- **Aim:** To test whether tree-search contributes to the final results (e.g., more accurate actions in the true environment), not just the neural network's credits.

- **Copy Model:** In the planning (simulation) loop, the network sees exactly the same state after taking each action and transition, which is in order to test whether solely using the statistics of the current state can give accurate actions

- **Conclusion:** Tree search and Neural Nets help each other

Results: Scalability



- Increasing # of simulations helps in terms of success ratio
- Same number of simulations is applied in both training and testing

Conclusion

- Learning to search, trained on a specific problem, improves performance compared to classical search techniques

- Planning-like behavior: performance increases with amount of time

- Credit assignment technique helps train anytime algorithm

Critical Questions

Paper:

- Fair comparison between MCTSNets trained with different number of simulations?
- Ablative analysis is absent(how each component contributes to the final result)?
- Scalability on more complex problems?
- Comparison with other classical DRL algorithms?
- Comparison on computational cost?
- Reproduction?

Method:

- Why using the results of MCTS as labels?
- If MCTS already gives the optimal results, then why bother to train a bunch of neural nets?
- Can a MCTSNet trained on one problem be transferred to other tasks (overfitting)?

Critical Questions

VI. RESULTS

We were not able to reproduce the results of the original article with Sokoban. Training time is huge and takes 20 seconds for 100 iterations on a GTX 1070. Training to 5.0×10^6 iterations would take 11 days, which was not possible. Furthermore our oracle was noisy and we doubt that the MCTSnet would have converged with such ground truth.

For the MouseGame we pushed the learning up to 15 hours, and the loss is given in Figure 8. The loss is indeed going down but the variance is increasing. Even after 15 hours of training, the MCTSnet is not able to beat a random agent, which raises some concerns (a standard DQN would give good results in 6 minutes of training, while a MCTS takes 4 seconds to build a very good solution).

It is unclear if the algorithm just needs a lot more training time to perform well, if it requires double precision floating point computation or if they are just a lot of room for improvement in our code.

https://github.com/faameunier/MCTSnet/blob/master/RL_Manuscript.pdf

Related Works: Learning to Search

- The learning-to-search framework (Chang et al., 2015) learns an evaluation function that is effective in the context of beam search

- The TD (leaf) algorithm (Baxter et al., 1998; Schaeffer et al., 2001) applies reinforcement learning to find an evaluation function that combines with minimax search to produce an accurate root evaluation

- In all cases, the evaluation function is scalar valued

Related Works: Meta Reasoning

- Kocsis et al. (2005) applies black-box optimization to learn the meta-parameters controlling an alpha-beta search
  - They do not learn fine-grained control over the search decision

- Pascanu et al. (2017) investigates learning-to-plan using neural networks
  - Their system uses an unstructured memory which makes complex branching very unlikely

Related Works: Search with Neural Nets

- The I2A architecture (Weber et al., 2017) aggregates the results of several simulations (from fixed policy) into its neural network computation
  - MCTSNets introduce a tree-structured memory and tree-expansion strategy

- Similar to I2A, the predictron architecture (Silver et al., 2017b) aggregates over multiple simulations
  - Simulations are rolled out in an implicit transition model
  - MCTSNets make concrete steps in the explicit (simulated) environment

Acknowledgement & Links

- https://github.com/keras-rl/keras-rl/issues/216
- https://github.com/faameunier/MCTSnet
- https://github.com/Chicoryn/dream-go/issues/32
- https://vimeo.com/312294797
- https://github.com/faameunier/MCTSnet/blob/master/RL_Manuscript.pdf

References

Baxter, J., Tridgell, A., and Weaver, L. Knightcap: A chess program that learns by combining td with gametree search. In *Proceedings of the 15th International Conference on Machine Learning*, 1998.

Chang, K.-W., Krishnamurthy, A., Agarwal, A., Daume, H., and Langford, J. Learning to search better than your teacher. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 2058–2066, 2015.

Kocsis, L., Szepesv´ari, C., and Winands, M. H. RSPSA: enhanced parameter optimization in games. In *Advances in Computer Games*, pp. 39–56. Springer, 2005.

Pascanu, R., Li, Y., Vinyals, O., Heess, N., Buesing, L., Racaniere, S., Reichert, D., Weber, T., Wierstra, D., and Battaglia, P. Learning model-based planning from scratch. *arXiv preprint arXiv:1707.06170*, 2017.

References

Schaeffer, J., Hlynka, M., and Jussila, V. Temporal difference learning applied to a high-performance game-playing program. In *Proceedings of the 17th international joint conference on Artificial intelligence-Volume 1*, pp. 529–534. Morgan Kaufmann Publishers Inc., 2001.

Silver, D., van Hasselt, H., Hessel, M., Schaul, T., Guez, A., Harley, T., Dulac-Arnold, G., Reichert, D., Rabinowitz, N., Barreto, A., et al. The predictron: End-to-end learning and planning. In *ICML*, 2017b.

Weber, T., Racani`ere, S., Reichert, D. P., Buesing, L., Guez, A., Rezende, D. J., Badia, A. P., Vinyals, O., Heess, N., Li, Y., et al. Imagination-augmented agents for deep reinforcement learning. *arXiv preprint arXiv:1707.06203*, 2017.

Q&A

Appendix

Dynamic Computation Graph