Deep Reinforcement Learning Part II

Johanni Brea

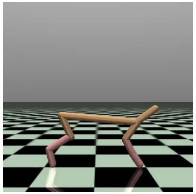
29 May 2020

Artificial Neural Networks CS-456

Deep Reinforcement Learning Applications

Video Games

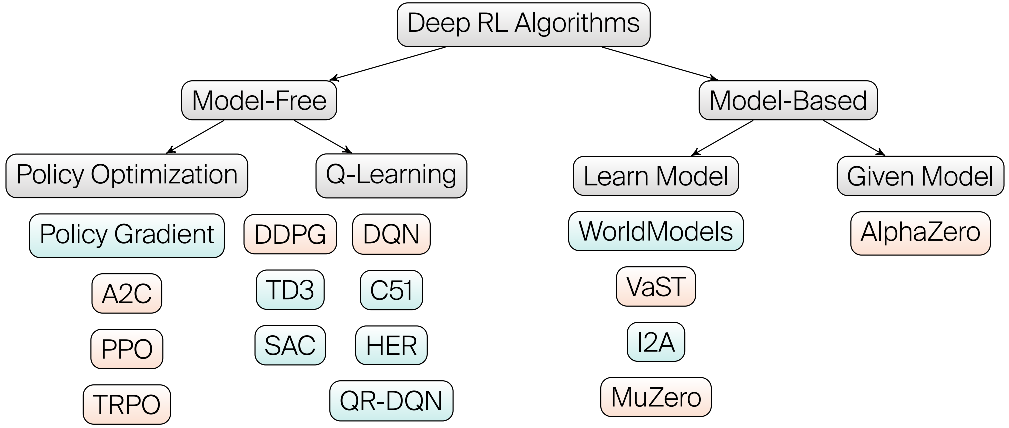
Simulated Robotics



Board Games

Chess
Shogi
Go

A Classification of Deep Reinforcement Learning Methods



inspired by `https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html`

Outline

1. **Mini-Batches in On- and Off-Policy Deep Reinforcement Learning.**
    1. Temporally Correlated Weight Updates Can Cause Instabilities.
    2. Deep Q-Network (DQN).
    3. Advantage Actor-Critic (A2C).
    4. Pros and Cons of On- and Off-Policy Deep RL.
2. **Deep Reinforcement Learning for Continuous Control.**
    1. Proximal Policy Optimization (PPO).
    2. Deep Deterministic Policy Gradient (DDPG).
    3. Comparison of Algorithms in Simulated Robotics.
3. **Model-Based Deep Reinforcement Learning.**
    1. Background Planning and Decision Time Planning.
    2. Model-Based Deep RL with Variational State Tabulation (VaST).
    3. Monte Carlo Tree Search (MCTS).
    4. AlphaZero.
    5. MuZero.

Mini-Batches in On- and Off-Policy Deep RL

Usually we train deep neural networks with independent and identically distributed (iid) mini-batches of training data.

In this section you will learn

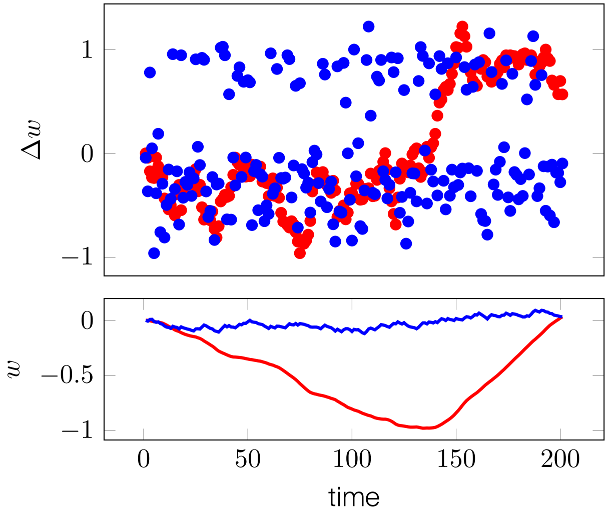
1. that we should not form mini-batches from sequentially acquired data in RL, but
2. use a **replay buffer** from which one can sample iid, or
3. run **multiple actors in parallel**.

Suggested reading: Mnih et al. 2015 and Mnih et al. 2016

Correlation of Subsequent Observations in RL

- Subsequent images are highly correlated.
- Images at the end of the episode may look quite differently from those at the beginning of the episode.
- In image classification we shuffle the training data to have approximately independent and identically distributed mini-batches.

Temporally Correlated Weight Updates Can Cause Instabilities



- $w_t = w_{t-1} + 0.02 \cdot \Delta w_t$.
- Temporally correlated weight updates can cause large weight changes $\Rightarrow$ potentially unstable learning.
- Reshuffling weight updates helps to prevent this $\Rightarrow$ reshuffling stabilizes learning.

Proposed Solutions for Deep RL

On-policy methods, like policy gradient or SARSA, attempt to improve the policy that is used to make decisions, whereas off-policy methods, like Q-Learning, improve a policy different from that used to generate the data.

Off-Policy Deep RL
e.g. DQN

- Put many (e.g. 1M) experiences (observation, action, reward) of a single agent into **replay buffer** (a first-in-first-out memory buffer).
- Randomly sample from the replay buffer to obtain mini-batches for training.

On-Policy Deep RL
e.g. A2C

- Run **multiple agents** (e.g. 16) and environment simulations with different random seeds in parallel (ideally, every agents sees a different observation at any moment in time)
- Obtain mini-batches from the observations, actions and rewards of the multiple actors.

EPFL

Deep Q-Network (DQN)

```
1:  Initialize neural network $Q_\theta$ and empty replay buffer $R$.
2:  Set target $\hat{Q} \leftarrow Q_\theta$, counter $t \leftarrow 0$, observe $s_0$.
3:  **repeat**
4:      Take action $a_t$ and observe reward $r_t$ and next state $s_{t+1}$
5:      Store $(s_t, a_t, r_t, s_{t+1})$ in $R$
6:      Sample random minibatch of transitions $(s_j, a_j, r_j, s_{j+1})$ from $R$
7:      Update $\theta$ with gradient of $\sum_j \left(r_j + \max_{a'} \hat{Q}(s_{j+1}, a') - Q_\theta(s_j, a_j)\right)^2$
8:      Increment $t$ and reset $\hat{Q} \leftarrow Q_\theta$ every $C$ steps.
9:  **until** some termination criterion is met.
10: **return** $Q_\theta$
```

EPFL

Advantage Actor-Critic (A2C)

```
1:  Initialize neural networks $\pi_\theta$ and $V_\phi$.
2:  Set counter $t \leftarrow 0$, observe $s_0$.
3:  **repeat**
4:      **for all** workers $k = 1, \dots, K$ **do**
5:          Take action $a_t^{(k)}$ and observe reward $r_t^{(k)}$ and next state $s_{t+1}^{(k)}$
6:          Compute $R_t^{(k)} = r_t^{(k)} + \gamma V_\phi(s_{t+1}^{(k)})$ and advantage $A_t^{(k)} = R_t^{(k)} - V_\phi(s_t^{(k)})$
7:      **end for**
8:      Update $\theta$ with gradient of $\sum_k A_t^{(k)} \log \pi_\theta(a_t^{(k)}; s_t^{(k)})$
9:      Update $\phi$ with gradient of $\sum_k \left(R_t^{(k)} - V_\phi(s_t^{(k)})\right)^2$.
10:     Increment $t$.
11: **until** some termination criterion is met.
12: **return** $\pi_\theta$ and $V_\phi$
```

Pros and Cons of On- and Off-Policy Deep RL

Off-Policy Deep RL

+ lower sample complexity
i.e. fewer interactions with the environment are needed, because experiences in the replay buffer can be used multiple times

- higher memory complexity
need to store many experiences in the replay buffer.

On-Policy Deep RL

- higher sample complexity
because old experiences cannot be used to update a policy that has already changed.

+ lower memory complexity
only the current observations, actions and rewards of the parallel agents are kept to update the policy.

Quiz

- If we use the SARSA loss $r_j + \hat{Q}(s_{j+1}, a_{j+1}) - Q_\theta(s_j, a_j)$ in the DQN algorithm, we just need to include also the next action a_{j+1} in the replay buffer and everything will work.
- We could use multiple actors instead of a replay memory with Q-Learning.
- In A2C, if all parallel workers K start together in the first step of the episode and every episode has the same length, we do not get the desired effect of iid minibatches.

Outline

Deep Reinforcement Learning for Continuous Control

- High-dimensional continuous action spaces (e.g. forces and torques). and observation spaces (e.g. positions, angles and velocities).
- Standard policy gradient could be applied, but it is difficult to find hyper-parameter settings such that learning is neither unstable nor very slow.
- Standard DQN cannot be applied, because it is designed for discrete actions.

In this section you will learn about

1. proximal policy optimization (PPO) methods that improve standard policy gradient methods and
2. an adaptation of DQN to continuous action spaces (DDPG).

Suggested reading: Kakade & Langford 2002, Schulman et al. 2015 & 2017, Lillicrap et al. 2016

How Big a Step Can We Make in Policy Gradient?

In simple Policy Gradient, the parameters θ of a neural network change according to

$$\theta' = \theta + \alpha \nabla J(\theta)$$

$$J(\theta) = E_{s_0 \sim p(s_0)}[V_\theta(s_0)] = E_{s_t, a_t \sim p_\theta, \pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t R^{a_t}_{s_t \to s_{t+1}}\right]$$

$$= \sum_{t=0}^{\infty} \sum_{s_t, s_{t+1}, a_t} \gamma^t R^{a_t}_{s_t \to s_{t+1}} P^{a_t}_{s_t \to s_{t+1}} \pi_\theta(a_t; s_t) p_\theta(s_t)$$

with $p_\theta(s_t) = \sum_{s_0, \dots, s_{t-1}, a_0, \dots, a_{t-1}} p(s_0) \prod_{\tau=0}^{t-1} P^{a_\tau}_{s_\tau \to s_{\tau+1}} \pi_\theta(a_\tau; s_\tau)$.

We actually want $J(\theta') - J(\theta)$ to be as large as possible.

How Big a Step Can We Make in Policy Gradient?

$$\begin{aligned}
J(\theta') - J(\theta) &= J(\theta') - E_{s_0 \sim p(s_0)}[V_\theta(s_0)] \\
&= J(\theta') - E_{s_t, a_t \sim p_{\theta'}, \pi_{\theta'}}[V_\theta(s_0)] \\
&= J(\theta') - E_{s_t, a_t \sim p_{\theta'}, \pi_{\theta'}}\left[\sum_{t=0}^{\infty} \gamma^t V_\theta(s_t) - \sum_{t=1}^{\infty} \gamma^t V_\theta(s_t)\right] \\
&= J(\theta') + E_{s_t, a_t \sim p_{\theta'}, \pi_{\theta'}}\left[\sum_{t=0}^{\infty} \gamma^t \big(\gamma V_\theta(s_{t+1}) - V_\theta(s_t)\big)\right] \\
&= E_{s_t, a_t \sim p_{\theta'}, \pi_{\theta'}}\left[\sum_{t=0}^{\infty} \gamma^t \big(R_{s_t \to s_{t+1}}^{a_t} + \gamma V_\theta(s_{t+1}) - V_\theta(s_t)\big)\right] \\
&= E_{s_t, a_t \sim p_{\theta'}, \pi_{\theta'}}\left[\sum_{t=0}^{\infty} \gamma^t A_\theta(s_t, a_t)\right] = \sum_{t=0}^{\infty} E_{s_t, a_t \sim p_{\theta'}, \pi_\theta}\left[\frac{\pi_{\theta'}(a_t; s_t)}{\pi_\theta(a_t; s_t)} \gamma^t A_\theta(s_t, a_t)\right]
\end{aligned}$$

http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-9.pdf

Proximal Policy Optimization: Idea

$$J(\theta') - J(\theta) = \sum_{t=0}^{\infty} E_{s_t, a_t \sim p_{\theta'}, \pi_\theta} \left[\underbrace{\frac{\pi_{\theta'}(a_t; s_t)}{\pi_\theta(a_t; s_t)}}_{= r_{\theta'}(s_t, a_t)} \gamma^t A_\theta(s_t, a_t) \right]$$

As long as $p_{\theta'}$ is close to p_θ such that

$$E_{s_t, a_t \sim p_{\theta'}, \pi_\theta} \left[r_{\theta'}(s_t, a_t) \gamma^t A_\theta(s_t, a_t) \right] \approx E_{s_t, a_t \sim p_\theta, \pi_\theta} \left[r_{\theta'}(s_t, a_t) \gamma^t A_\theta(s_t, a_t) \right]$$

we can take the samples $s_t, a_t \sim p_\theta, \pi_\theta$ obtained with the old policy and optimize the loss

$$\hat{L}(\theta') = \sum_{t=0}^{\infty} r_{\theta'}(s_t, a_t) \gamma^t A_\theta(s_t, a_t)$$

Proximal Policy Optimization: Losses

$$\hat{L}(\theta') = \sum_{t=0}^{\infty} r_{\theta'}(s_t, a_t)\gamma^t A_\theta(s_t, a_t)$$

Trust-Region Policy Optimization (TRPO)

Maximize $\hat{L}(\theta')$ subject to $\mathrm{KL}[\pi_\theta \| \pi_{\theta'}] \leq \delta$.

Clipped Surrogate Objectives (PPO-CLIP)

Maximize $\hat{L}^{\mathrm{CLIP}}(\theta') = \sum_{t=0}^{\infty} \min(r_{\theta'}\gamma^t A_\theta, \mathrm{clip}(r_{\theta'}, 1-\epsilon, 1+\epsilon)\gamma^t A_\theta)$.

Proximal Policy Optimization

```
1:  Initialize neural networks $\pi_\theta$ and $V_\phi$.
2:  Set counter $t \leftarrow 0$, observe $s_0$.
3:  **repeat**
4:      **for all** workers $k = 1, \dots, K$ **do**
5:          Take action $a_t^{(k)}$ and observe reward $r_t^{(k)}$ and next state $s_{t+1}^{(k)}$
6:          Compute $R_t^{(k)} = r_t^{(k)} + \gamma V_\phi(s_{t+1}^{(k)})$ and advantage $A_t^{(k)} = R_t^{(k)} - V_\phi(s_t^{(k)})$
7:      **end for**
8:      Optimize surrogate loss (TRPO or PPO-CLIP) wrt. $\theta'$ with $M$ epochs.
9:      Update $\phi$ with gradient of $\sum_k \left(R_t^{(k)} - V_\phi(s_t^{(k)})\right)^2$.
10:     Increment $t$.
11: **until** some termination criterion is met.
12: **return** $\pi_\theta$ and $V_\phi$
```

Summary & Quiz

One can improve the stability and sample efficiency of policy gradient methods by maximizing in an inner loop a surrogate objective function, like the one of TRPO or PPO-CLIP.

- With the update of policy gradient, $\theta' = \theta + \alpha \nabla J(\theta)$ and a fixed learning rate α, $J(\theta') - J(\theta)$ will allways be positive.
- In proximal policy optimization methods we want to keep the ratio $r_{\theta'}(s_t, a_t) = \frac{\pi_{\theta'}(a_t; s_t)}{\pi_\theta(a_t; s_t)}$ close to one, such that the state visitation probabilites $p_\theta(s_t)$ and $p_{\theta'}(s_t)$ are roughly the same.
- A2C uses each minibatch once to update the policy, whereas proximal policy methods use each minibatch multiple times, usually.

Deep Deterministic Policy Gradient (DDPG) [1]

In DQN for discrete actions the Q-values for N_a actions are given as the activity of N_a output neurons of a neural network with parameters θ and input given by the state s.

For continuous actions there are infinitely many values; obviously we do not want $N_a = \infty$.

Proposed solution: use a policy network $\pi_\psi(s)$ that maps deterministically states s to continuous actions a.

[1] The name is confusing: DDPG is more closely related to DQN than to PG!

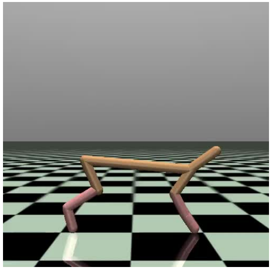
Lillicrap et al. 2016

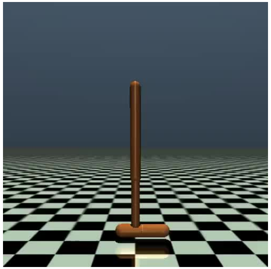
Deep Deterministic Policy Gradient (DDPG)

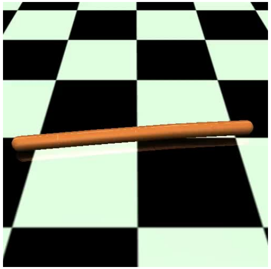
```
1:  Initialize neural networks $Q_\theta$, $\pi_\psi$ and empty replay buffer $R$.
2:  Set target $\hat{Q} \leftarrow Q_\theta$, $\hat{\pi} \leftarrow \pi_\psi$, counter $t \leftarrow 0$, observe $s_0$.
3:  **repeat**
4:      Take action $a_t = \pi_\psi(s_t) + \epsilon$ and observe reward $r_t$ and next state $s_{t+1}$
5:      Store $(s_t, a_t, r_t, s_{t+1})$ in $R$
6:      Sample random minibatch of transitions $(s_j, a_j, r_j, s_{j+1})$ from $R$
7:      Update $\theta$ with gradient of $\sum_j \left(r_j + \hat{Q}(s_{j+1}, \hat{\pi}(s_{j+1})) - Q_\theta(s_j, a_j)\right)^2$
8:      Update $\psi$ with gradient of $\sum_j Q_\theta(s_j, \pi_\psi(s_j))$.
9:      Increment $t$ and reset $\hat{Q} \leftarrow Q_\theta, \hat{\pi} \leftarrow \pi_\psi$ every $C$ steps.
10: **until** some termination criterion is met.
11: **return** $Q_\theta$
```

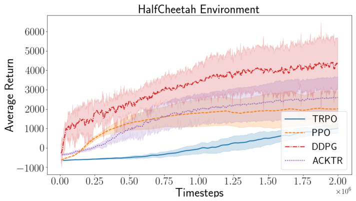
Lillicrap et al. 2016

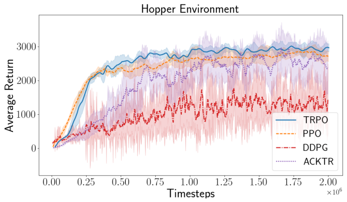
Comparison of Algorithms in Simulated Robotics

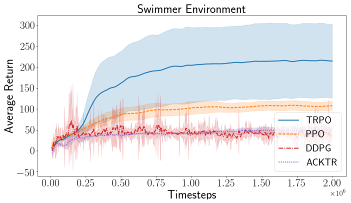












Henderson et al. 2017

Summary

- DQN can be adapted to domains with continuous actions by training an additional policy network π_ψ (DDPG).
- Which algorithm works best depends on the problem, usually.
- We did not discuss sufficient and efficient exploration, but it usually has a strong impact on the learning curve. A simple strategy for Policy Gradient methods is to add entropy regularization such that the policy does not become deterministic too quickly, but there are more advanced methods (see e.g. SAC).

Outline

Model-Based Deep Reinforcement Learning

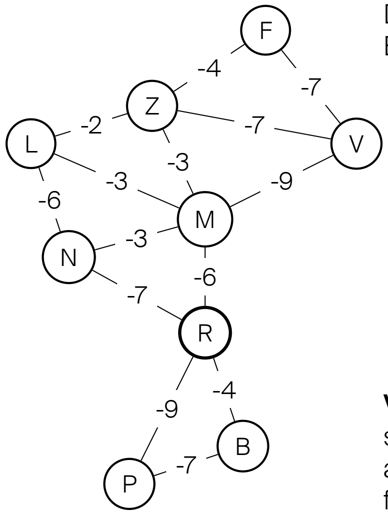
A model-based reinforcement learning method estimates (or knows explicitly) the transition dynamics (e.g. $P^a_{s\to s'}$) and reward structure (e.g. $R^a_{s\to s'}$).

In this section you will learn

1. how the model can be used for **planning**,
2. how we can use standard planning methods in deep RL with variational state tabulation (**VaST**),
3. why we may prefer **decision time planning** over **background planning**,
4. how **AlphaZero** learns to play Go with **Monte Carlo Tree Search**,
5. how **MuZero** reaches the same level as AlphaZero without explicitly knowing the rules of the game.

EPFL Suggested: Sutton & Barto Ch. 8, Corneil et al. 2018, Silver et al. 2016 & 2018, Schrittwieser et al. 2019

Background Planning



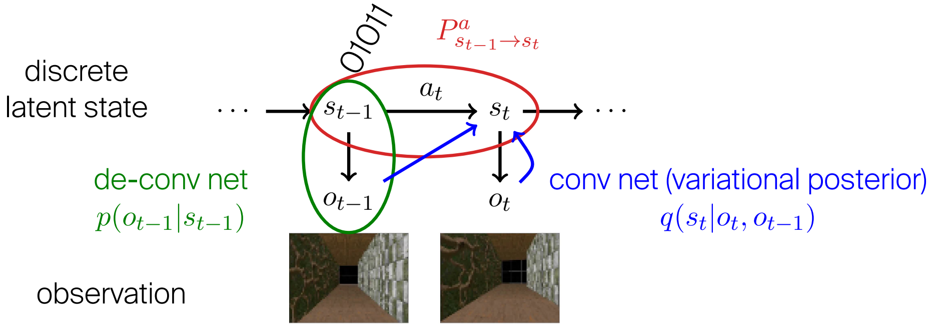
Deterministic dynamics ($P^{\text{south}}_{\text{M}\to\text{R}} = 1$, $R^{\text{south}}_{\text{M}\to\text{R}} = -6$)
Episode ends when the agent reaches R.

1. Start in R
2. for every state s that leads to R ($P^a_{s\to R} = 1$ for some a), compute its value $V(s) = \max_a R^a_{s\to\text{R}}$, e.g. $V(\text{M}) = -6$
3. for every state s that leads to an already visited states s' compute its value
$$V(s) = \max_a \sum_{s'} P^a_{s\to s'}\left(R^a_{s\to s'} + V(s')\right) \tag{1}$$
e.g. $Q(\text{L}, \text{south}) = -13$, $V(\text{L}) = -3 + (-6) = -9$.

Value Iteration Start with arbitrary values $V(s)$ for every state and iterate equation 1 until convergence. This works also in infinite-horizon settings, if we include a discount factor $\gamma < 1$.

Model-Based Deep RL with Variational State Tabulation (VaST)

Learn to encode images as discrete states (similar images o, same discrete state s). Whenever the agent learns something new about the environment ($P^a_{s\to s'}$ or $R^a_{s\to s'}$ changes), update offline the policy by running (a smart version of) value iteration.



EPFL

Corneil, Gerstner, Brea (2018), ICML

Summary: Background Planning and Variational State Tabulation

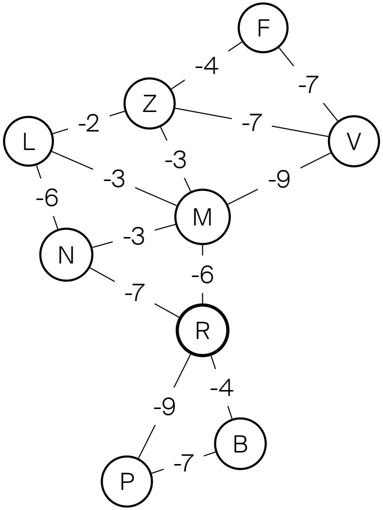
- In model-based reinforcement learning we can use background planning to update the Q- and V-values efficiently (e.g. by value iteration), if the number of states and actions is not too large.
- Learned abstractions may be useful to map high dimensional states (like images) to discrete states that can be used with standard planning methods.

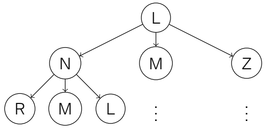
Value Iteration Does Not Scale!

- There are presumably less than 10^8 intersections of roads in Europe. Value iteration on current compute hardware is feasible for this problem.
- But there are more than 10^{170} possible positions for the game of Go. (The number of atoms on earth is roughly 10^{50}).

EPFL

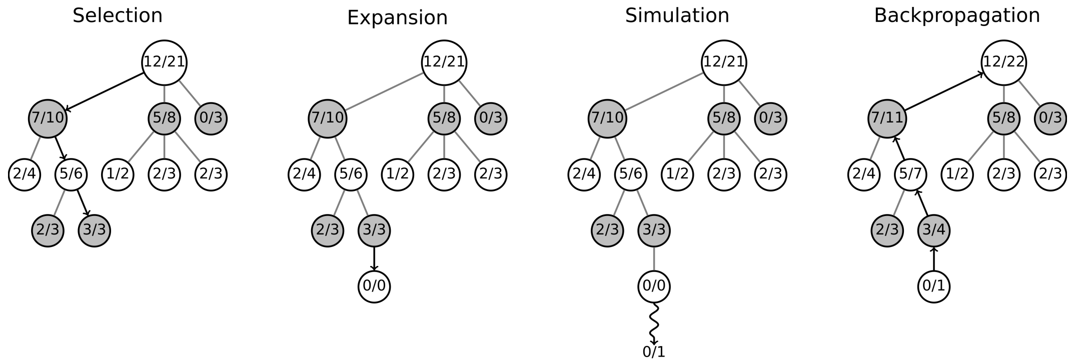
Decision Time Planning





1. Start where you are e.g. in L
2. Expand the decision tree, i.e. move to every state s that is a successor of L ($P^a_{\text{L}\to s}$).
3. Iteratively move to the successors of the successors up to a certain depth or until a terminal state is reached.
4. Propagate values backwards along the decision tree like in backward planning (but only along the decision tree).

Monte Carlo Tree Search (MCTS)



These four steps are iterated many times before choosing an actual action.

If you don't know MCTS: watch `https://youtu.be/UXW2yZndl7U`

AlphaZero

Each state-action pair (s, a) stores the visit counts $N(s, a)$, the total action value $W(s, a)$, the mean action value $Q(s, a)$ and the prior action probability $P(s, a)$.

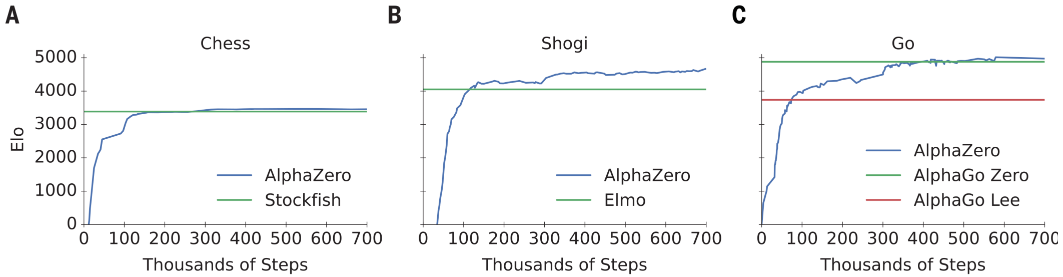
1. **Selection** $a_t = \arg\max_a Q(s_t, a) + P(s_t, a) C(s) \sqrt{N(s_t)}/(1 + N(s_t, a))$
   with exploration rate $C(s) = \log((1 + N(s) + c_{\text{base}})/c_{\text{base}}) + c_{\text{init}}$
2. **Expansion** of leaf node s_L: initialize $N(s_L, a) = W(s_L, a) = Q(s_L, a) = 0$ and $P(s_L, a) = p_a$ with $(\boldsymbol{p}, v) = f_\theta(s_L)$
3. No **simulation**.
4. **Backprop** $N(s_t, a_t) = N(s_t, a_t) + 1, W(s_t, a_t) = W(s_t, a_t) + v, Q(s_t, a_t) = \frac{W(s_t, a_t)}{N(s_t, a_t)}$

Actual action sampled from $\pi(a|s_0) = N(s_0, a)^{1/\tau} / \sum_b N(s_0, b)^{1/\tau}$.

Neural network loss: $l(\theta) = (z - v)^2 - \pi \log \boldsymbol{p} + c||\theta||^2$
where $z \in \{-1, 0, 1\}$ is the outcome of the game, and $(\boldsymbol{p}, v) = f_\theta(s_0)$

Silver et al. 2016, Silver et al. 2018

AlphaZero: Success Story



Silver et al. 2018

AlphaZero: Summary

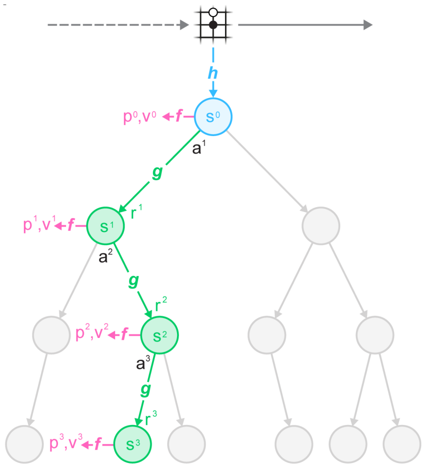
- In each position of the game (for both players), MCTS runs for some time.
- The sub-tree below the actual next state is retained as the initial MCTS tree.
- In contrast to traditional Go/Chess/Shogi software there is no hand-crafting of promising tree expansions or position evaluations: selection of MCTS is focused on promising actions (according to learned prior action probability $\boldsymbol{p}$) and the value of leaf nodes v is based on a learned position evaluation function.
- Prior action probabilities $\boldsymbol{p}$ are trained with the cross-entropy loss to match the actual policy π of MCTS and the values v are trained with the squared error loss to the actual outcome z of the game.
- AlphaZero does not need to learn the reward and transition model $R^a_{s \to s'}, P^a_{s \to s'}$; for MCTS it relies on the hard coded rules of the game. What if the model is unknown?

What If the Model Is Unknown?

In high dimensional settings, like in the game Go, it seems hopeless to learn explicitly $R^a_{s\to s'}$ and $P^a_{s\to s'}$.

Instead, MuZero demonstrates that it is possible to use MTCS with a learned latent representation of observations and a learned dynamics model.With this, MuZero reaches state-of-the-art performance not only on board games Chess, Shogi and Go but also in the domain of Atari video games.

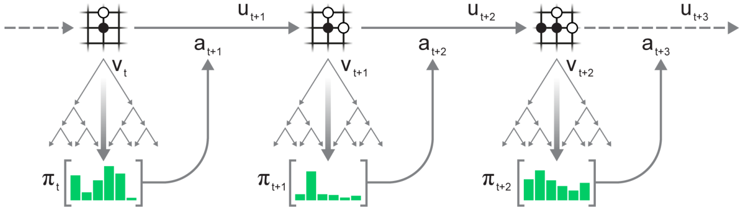
How MuZero uses its model to plan



The model consists of three connected components for representation, dynamics and prediction. The initial hidden state $s^{(0)}$ is obtained by passing the past observations (e.g. the Go board or Atari screen) into a representation function h. Given a previous hidden state $s^{(k-1)}$ and a candidate action $a^{(k)}$, the dynamics function g produces an immediate reward $r^{(k)}$ and a new hidden state $s^{(k)}$. The policy $p^{(k)}$ and value function $v^{(k)}$ are computed from the hidden state $s^{(k)}$ by a prediction function f.

Schrittwieser et al. 2019

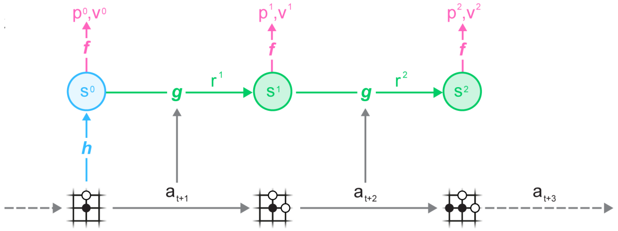
How MuZero acts in the environment



A Monte-Carlo Tree Search is performed at each timestep t, as described on the previous slide. An action a_{t+1} is sampled from the search policy π_t, which is proportional to the visit count for each action from the root node. The environment receives the action and generates a new observation o_{t+1} and reward u_{t+1}. At the end of the episode the trajectory data is stored into a replay buffer.

Schrittwieser et al. 2019

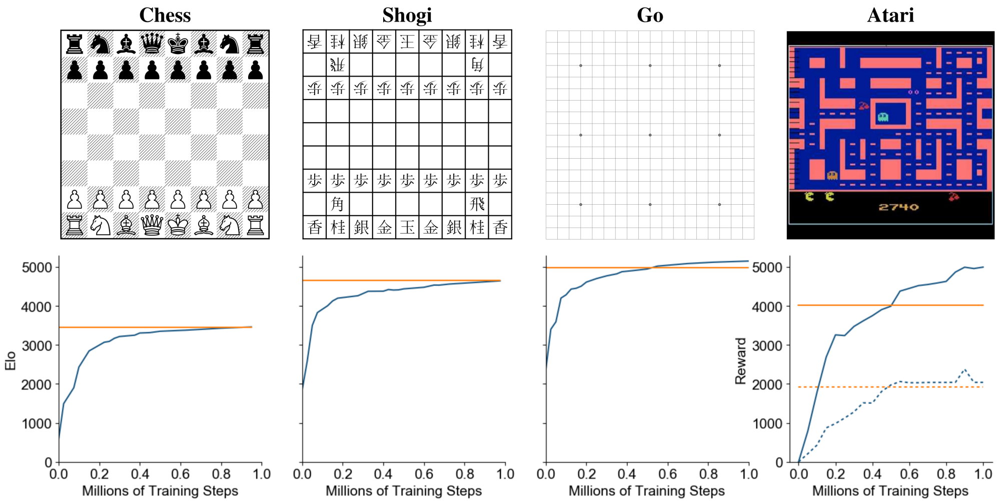
How MuZero trains its model



A trajectory is sampled from the replay buffer. For the initial step, the representation function h receives as input the past observations $o_1, \ldots, o_t$ from the selected trajectory.The model is subsequently unrolled recurrently for K steps. At each step k, the dynamics function g receives as input the hidden state $s^{(k-1)}$ from the previous step and the real action a_{t+k}. The parameters of the representation, dynamics and prediction functions are jointly trained, end-to-end by backpropagation-through-time, to predict three quantities: the policy $p^{(k)} \approx \pi_{t+k}$, value function $v^{(k)} \approx z_{t+k}$, and reward $r^{(t+k)} \approx u_{t+k}$, where z_{t+k} is a sample return: either the final reward (board games) or n-step return (Atari).

Schrittwieser et al. 2019

MuZero: Success Story



Schrittwieser et al. 2019

MuZero: Summary

- Instead of relying on a given reward and transition model, MuZero learns an encoding of observations to states h and a transition function from states to next states g.
- Like AlphaZero it relies on MCTS with a learned function f that estimates the action probability $\boldsymbol{p}$ and value v of a state.

Quiz

- With background planning, decision making is computationally very cheap, since one can use the Q-values to find the optimal action.
- In Monte Carlo Tree Search, decision making is computationally very cheap, since one can use the Q-values to find the optimal action.
- In AlphaZero's MCTS the actual actions are sampled from the prior action probability $P(s, a)$.
- In AlphaZero's MCTS the prior action probability is trained to be as close as possible to the actual policy of MCTS.
- The latent representation in AlphaZero is a binary word, e.g. 10110.

Conclusions

- The problem of correlated samples can be solved with a replay buffer for off-policy model-free reinforcement learning (DQN) and multiple parallel actors for on-policy model-free reinforcement learning (A2C).
- Both, on- and off-policy deep reinforcement learning methods can be adapted to continuous action spaces, by either making policy gradient methods more sample efficient and robust (PPO) or adding a deterministic policy network (DDPG).
- In model-based reinforcement learning we can use background planning (e.g. value iteration) in moderately sized state spaces and decision-time planning (e.g. MTCS) for large state spaces and rely on learned state abstractions to perform planning not on the level of raw observations but on compressed representations.

References

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W. (2017).
Hindsight Experience Replay.
arXiv e-prints, page arXiv:1707.01495.

Bellemare, M. G., Dabney, W., and Munos, R. (2017).
A Distributional Perspective on Reinforcement Learning.
arXiv e-prints, page arXiv:1707.06887.

Corneil, D., Gerstner, W., and Brea, J. (2018).
Efficient model-based deep reinforcement learning with variational state tabulation.
In Dy, J. and Krause, A., editors, Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research, pages 1057–1066, Stockholmsmässan, Stockholm Sweden. PMLR.

Dabney, W., Rowland, M., Bellemare, M. G., and Munos, R. (2017).
Distributional Reinforcement Learning with Quantile Regression.
arXiv e-prints, page arXiv:1710.10044.

Fujimoto, S., van Hoof, H., and Meger, D. (2018).
Addressing Function Approximation Error in Actor-Critic Methods.
arXiv e-prints, page arXiv:1802.09477.

Ha, D. and Schmidhuber, J. (2018).
World Models.
ArXiv e-prints.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018).
Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.
In Dy, J. and Krause, A., editors, Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research, pages 1861–1870, Stockholmsmässan, Stockholm Sweden. PMLR.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2017).
Deep Reinforcement Learning that Matters.
arXiv e-prints, page arXiv:1709.06560.

Kakade, S. M. (2002).
A natural policy gradient.
In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, Advances in Neural Information Processing Systems 14, pages 1531–1538. MIT Press.

Lillicrap, T. P., Cownden, D., Tweed, D. B., and Akerman, C. J. (2016).
Random synaptic feedback weights support error backpropagation for deep learning.
Nature Communications, 7:13276.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016).
Asynchronous methods for deep reinforcement learning.
In Balcan, M. F. and Weinberger, K. Q., editors, Proceedings of The 33rd International Conference on Machine Learning, volume 48 of Proceedings of Machine Learning Research, pages 1928–1937, New York, New York, USA. PMLR.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., and et al. (2015).
Human-level control through deep reinforcement learning.
Nature, 518(7540):529–533.

Racanière, S., Weber, T., Reichert, D., Buesing, L., Guez, A., Jimenez Rezende, D., Puigdomènech Badia, A., Vinyals, O., Heess, N., Li, Y., Pascanu, R., Battaglia, P., Hassabis, D., Silver, D., and Wierstra, D. (2017).
Imagination-augmented agents for deep reinforcement learning.
In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, Advances in Neural Information Processing Systems 30, pages 5690–5701. Curran Associates, Inc.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., and Silver, D. (2019).
Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model.
arXiv e-prints, page arXiv:1911.08265.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. (2015).
Trust Region Policy Optimization.
ArXiv e-prints.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017).
Proximal Policy Optimization Algorithms.
arXiv e-prints, page arXiv:1707.06347.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., and et al. (2016).
Mastering the game of go with deep neural networks and tree search.
Nature, 529(7587):484–489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., and et al. (2018).
A general reinforcement learning algorithm that masters chess, shogi, and go through self-play.
Science, 362(6419):1140–1144.