Fraunhofer IIS

Fraunhofer-Institut für Integrierte Schaltungen IIS

Reinforcement Learning

—

Lecture 9: Model-based RL (Discrete Actions)

Christopher Mutschler

Introduction to Model-Based Reinforcement Learning

Outline

- Motivation: why model-based RL?
- What is a model? What are its inputs? What is a good model?
- How can we use a model?
  - Background Planning
    - Environment data augmentation / simulation
    - Sample-efficient policy learning
  - Online Planning
    - Discrete Actions
    - Continuous Actions
  - Auxiliary tasks
- Real-world application

Introduction to Model-Based Reinforcement Learning

Outline

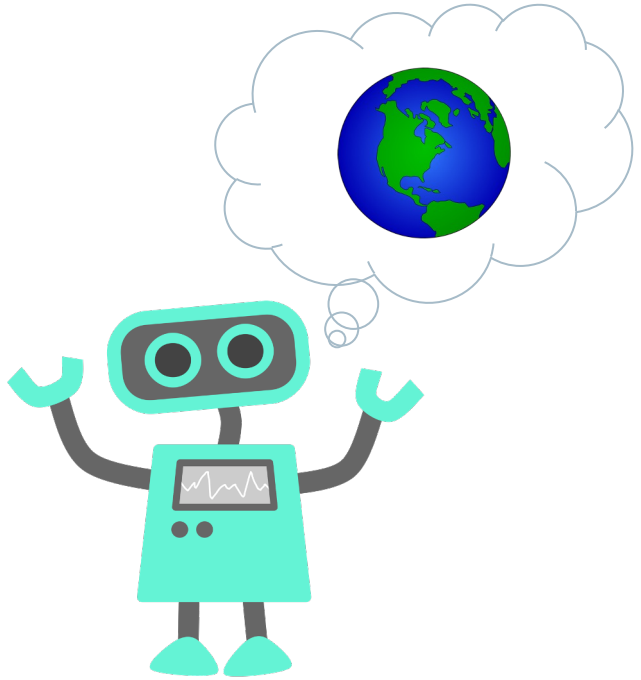
- **Motivation: why model-based RL?**
- What is a model? What are its inputs? What is a good model?
- How can we use a model?
  - Background Planning
    - Environment data augmentation / simulation
    - Sample-efficient policy learning
  - Online Planning
    - Discrete Actions
    - Continuous Actions
  - Auxiliary tasks
- Real-world application

Introduction to Model-Based Reinforcement Learning

Why Model-based RL?

"If the organism carries a **'small-scale model' of external reality** and of its own possible actions within its head, it is able to try out various alternatives, conclude which is the best of them, react to future situations before they arise, utilise the knowledge of past events in dealing with the present and future, and in every way to react in a much **fuller, safer, and more competent** manner to the emergencies which face it."

—Craik, 1943, The Nature of Explanation

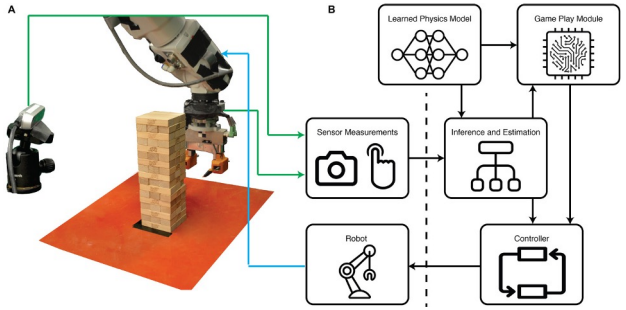


Slide adapted from https://sites.google.com/view/mbrl-tutorial

Introduction to Model-Based Reinforcement Learning

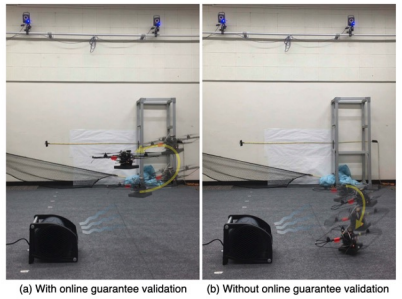
Why Model-based RL?

Robotic control



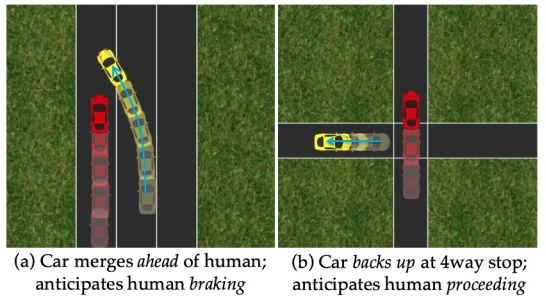
Fazeli et al. (2019). See, feel, act: Hierarchical learning for complex manipulation skills with multisensory fusion. Science Robotics, 4(26).

Safety



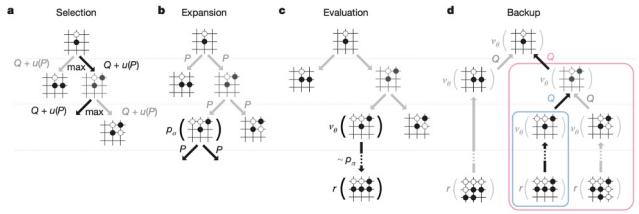
Fisac et al. (2019). A General Safety Framework for Learning-Based Control in Uncertain Robotic Systems. IEEE Transactions on Automatic Control.

HMI: H-AI-I



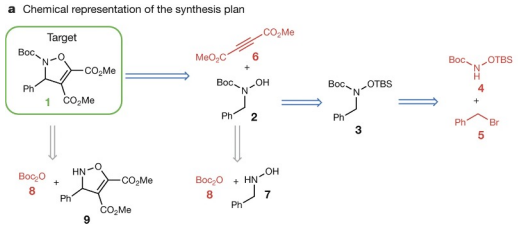
Sadigh et al. (2016). Planning for autonomous cars that leverage effects on human actions. RSS 2016.

Games



Silver et al. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), 484.

Science



Segler, Preuss, & Waller (2018). Planning chemical syntheses with deep neural networks and symbolic AI. Nature, 555(7698).

Operations Research

[1]

Salas & Powell (2013). Benchmarking a scalable approximate dynamic programming algorithm for stochastic control of multidimensional energy storage problems.

[1] *Warren Powell's 2017 ECSO tutorial, "A Unified Framework for Optimization under Uncertainty"*

Fraunhofer IIS

Introduction to Model-Based Reinforcement Learning

Model free vs. Model-based Reinforcement Learning

	Model-free	Model-based
Asymptotic Rewards	+	“depends”
Computation at deployment	+	- / +
Data efficiency	-	+
Speed to adapt to changing rewards	-	+
Speed to adapt to changing dynamics	-	+
Exploration	-	+

Introduction to Model-Based Reinforcement Learning

Outline

- Motivation: why model-based RL?
- **What is a model? What are its inputs? What is a good model?**
- How can we use a model?
  - Background Planning
    - Environment data augmentation / simulation
    - Sample-efficient policy learning
  - Online Planning
    - Discrete Actions
    - Continuous Actions
  - Auxiliary Tasks
- Real-world application

Introduction to Model-Based Reinforcement Learning

What is a model?

- Quick recap: sequential decision making
- Our goal:

$$\arg\max_{\pi} \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t),$$

subject to: $a_t = \pi(s_t)$, and

$$s_{t+1} = T(s_t, a_t)$$

- We collect data $\mathcal{D} = \{s_t, a_t, r_{t+1}, s_{t+1}\}_{t=0}^{T}$
- **Model-free RL:** learn policy directly from data

  $\mathcal{D} \rightarrow \pi$, e.g., with Q-Learning, policy gradients, …
- **Model-based RL:** learn model, then use it to learn or improve policy:

  $\mathcal{D} \rightarrow f \rightarrow \pi$

Introduction to Model-Based Reinforcement Learning

What is a model?

Definition: *A model is a representation that **explicitly** encodes knowledge about the structure of the environment and task.*

But what knowledge to explicitly encode?

- Transition/dynamics model: $s_{t+1} = f_s(s_t, a_t)$
- Reward model: $r_{t+1} = f_r(s_t, a_t)$

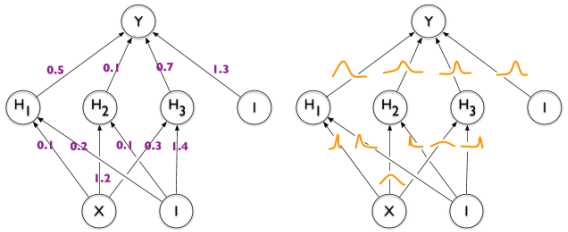
** typically what MBRL is doing*

- Inverse dynamics model: $a_t = f_s^{-1}(s_t, s_{t+1})$
- Distance model: $d_{ij} = f_d(s_i, s_j)$
- Future return model: $G_t = Q(s_t, a_t)$ or $G_t = V(s_t)$

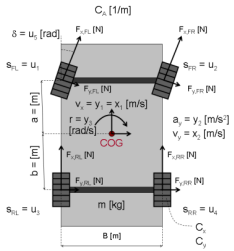
What is a model?

Parametric vs. non-parametric representations

- Parametric

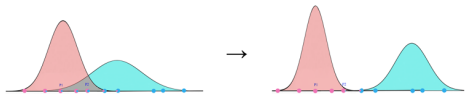


Neural Networks

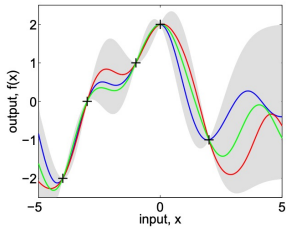


Physics-based

- Non-parametric



Gaussian Mixture Models



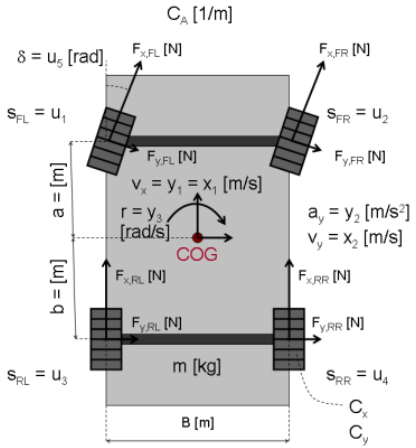
Gaussian Processes

Decision Trees or Random Forests

What is a model?

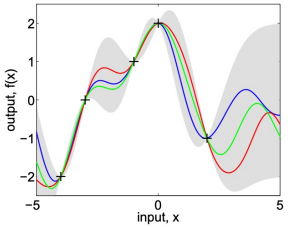
Model inputs: States

- Dynamical system states (classical control theory)



The Bicycle model is given by,

$$\dot{X} = v_x cos(\psi) - v_y sin(\psi) \tag{2.31a}$$
$$\dot{Y} = v_x sin(\psi) + v_y cos(\psi) \tag{2.31b}$$
$$\dot{\psi} = \frac{v_x}{l_f + l_r} tan(\delta) \tag{2.31c}$$
$$m\dot{v}_x = F_x + mv_y\dot{\psi} - 2F_{cf} sin(\delta) - F_a - F_r \tag{2.31d}$$
$$m\dot{v}_y = -mv_x\dot{\psi} + 2(F_{cf} cos(\delta) + F_{cr}) \tag{2.31e}$$
$$I\ddot{\psi} = 2(l_f F_{cf} cos(\delta) - l_r F_{cr}) \tag{2.31f}$$



https://de.mathworks.com/help/ident/ug/modeling-a-vehicle-dynamics-system.html

M. I. Palmqvist et al.: Model predictive control for autonomous driving of a truck. KTH Royal Institute of Technology School of Electrical Engineering. 2016.

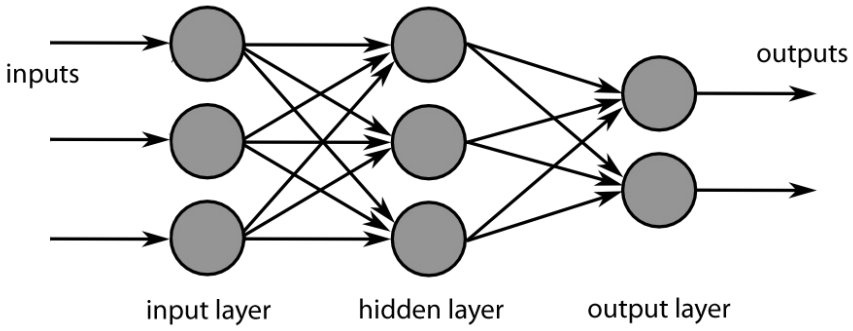
Fraunhofer IIS

What is a model?

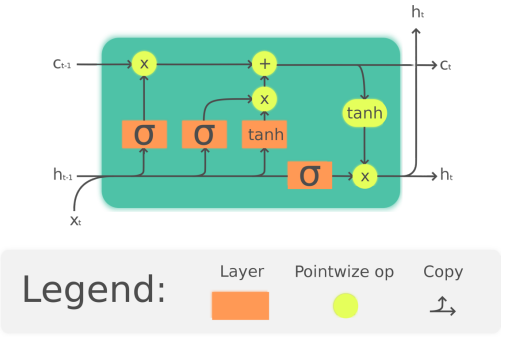
Model inputs: States

Pre-processed sensor data, e.g.

- objects/bounding boxes extracted from images



https://de.wikipedia.org/wiki/Deep_Learning



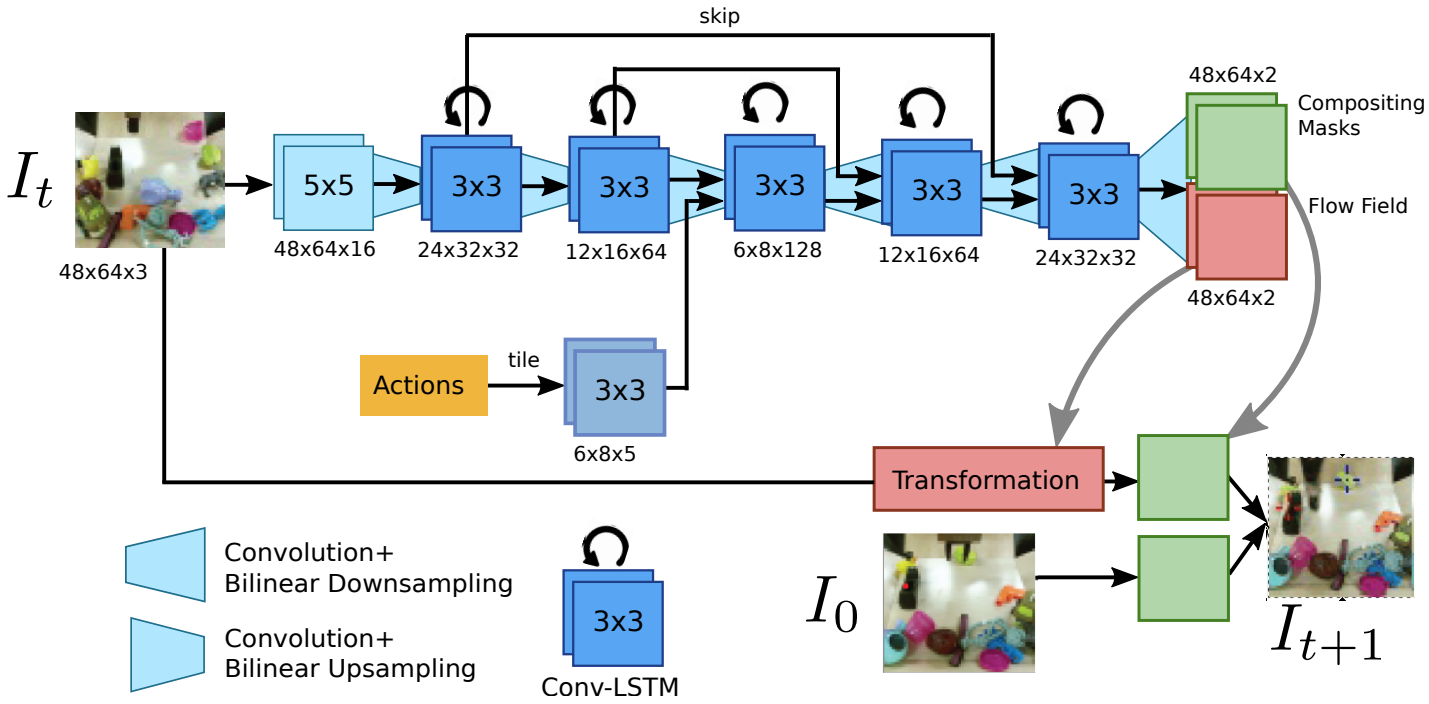
https://en.wikipedia.org/wiki/Long_short-term_memory

What is a model?

Model inputs: Observations

Raw sensor data (e.g., images or LIDAR scans)

- *Note: not that common in real-world applications*



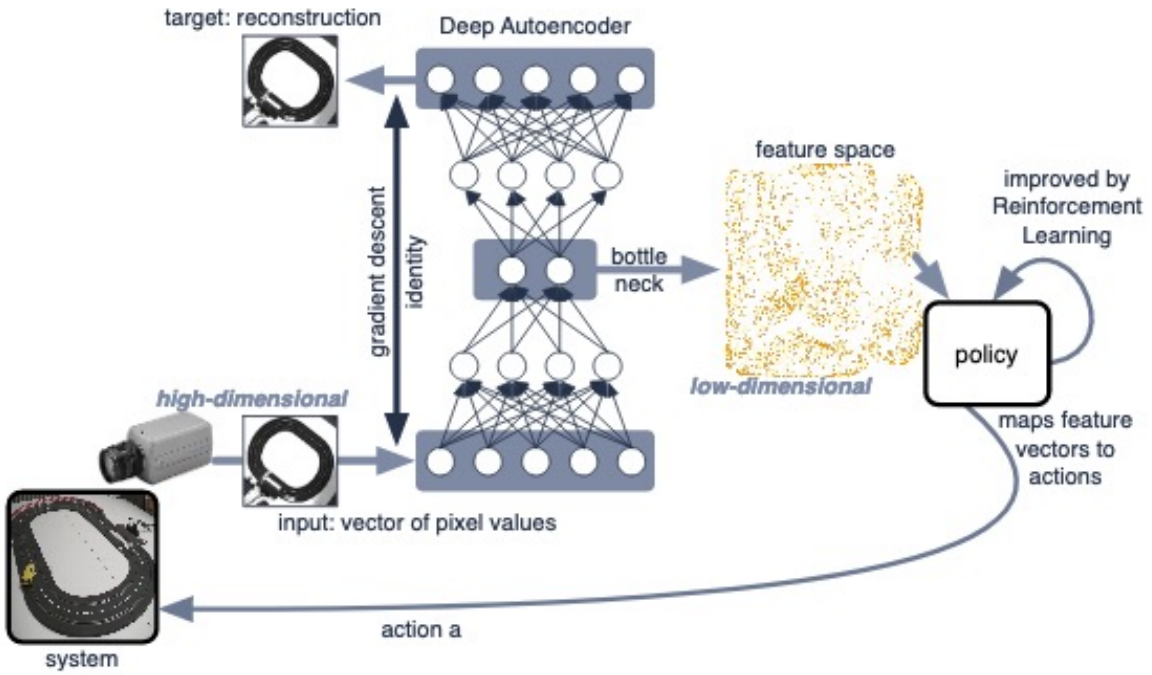
Ebert, Finn, et al. (2018); Finn & Levine (2017); Finn, Goodfellow, & Levine (2016)
https://bair.berkeley.edu/blog/2018/11/30/visual-rl/

What is a model?

Model inputs: Latent States

Extract useful features from the raw observations and use them as inputs to the model

- *Note: current practice*



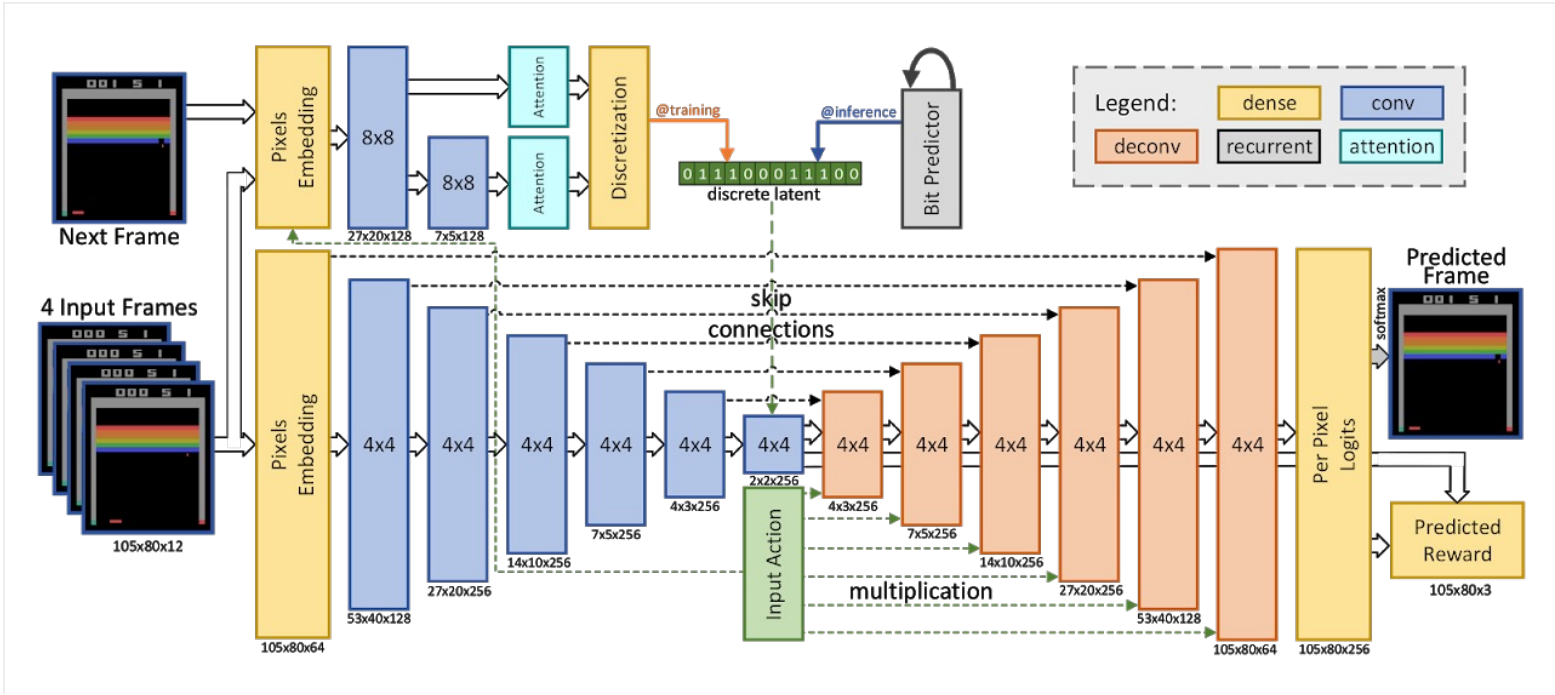
Lange et al.: Batch reinforcement learning. In Reinforcement learning (pp. 45-73). Springer, Berlin, Heidelberg. 2012.

What is a model?

Model inputs: Latent States

Extract useful features from the raw observations and use them as inputs to the model

- *Note: current practice*

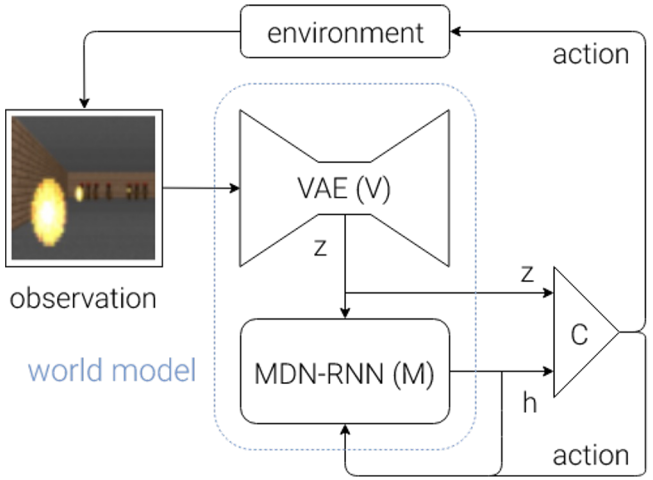


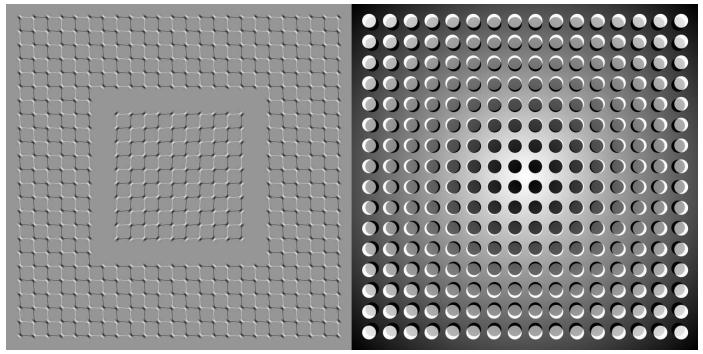
Kaiser et al.: Model-based reinforcement learning for atari. arXiv preprint arXiv:1903.00374. 2019.

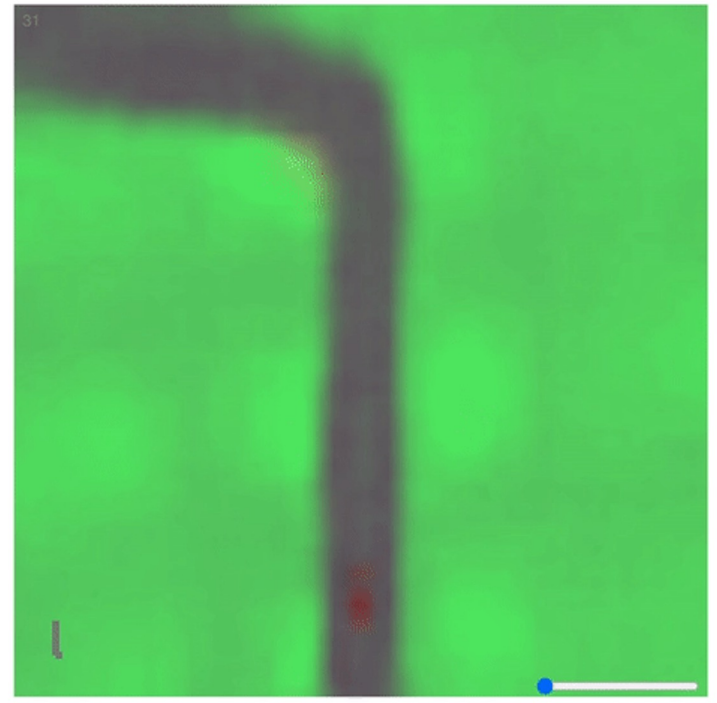
What is a model?

Model inputs: Latent States

- Extract useful features from the raw observations and use them as inputs to the model
- *Note: current practice*







Ha & Schmidhuber: World Models. NeurIPS 2018.

See also: https://worldmodels.github.io

What is a model?

Which model should we use?

Model desiderata

- **Learning sample efficiency**
  - We would like to interact with the real system as little as possible (e.g., due to time and safety constraints, hardware wear and tear, etc.)
- **Multi-step accuracy**
  - We require our model to be able to accurately predict several time-steps in the future (remember one-step rewards vs discounted return)
- **Required engineering / domain knowledge**
  - How easy it is to design a "simulator" for the real system using basic physics? How accurate such model would be?
  - Does it make sense from economical perspective?
- **Prediction speed**
  - Can we deploy in real-time systems (e.g., drones)?

What is a model?

Which model should we use?

Model desiderata

Name	Features	Speed of learning	Speed of predictions	Domain knowledge	Long-term accuracy
Dynamical system	States	Fast	Fast	High	High
MLP	States	Med	Fast	Low	Med
Observation	Observations	Slow	Slow	Low	Low
State-space models	Latent States	Slow	Fast	Low	Med

https://sites.google.com/view/mbrl-tutorial

Outline

Background Planning

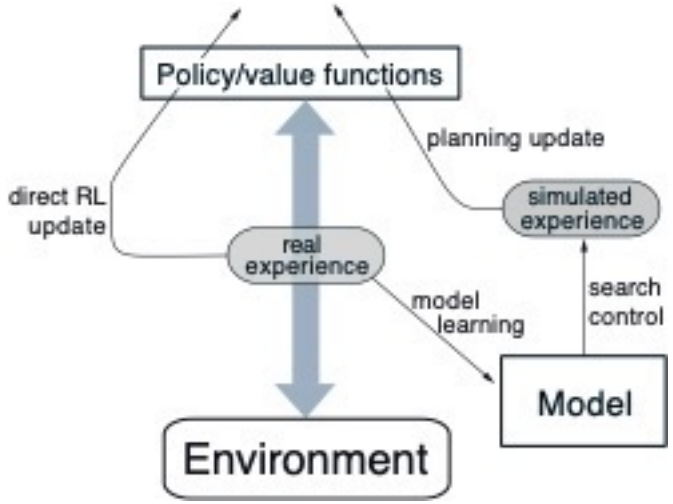
- Motivation: why model-based RL?
- What is a model? What are its inputs? What is a good model?
- **How can we use a model?**
  - **Background Planning**
    - **Environment data augmentation / simulation**
    - Sample-efficient policy learning
  - Online Planning
    - Discrete Actions
    - Continuous Actions
  - Auxiliary tasks
- Real-world application

Background Planning

Environment Data Augmentation

Dyna Architecture: Dyna-Q

- Use collected data to learn a transition and reward model
- Train a traditional RL algorithm (e.g., Q-Learning) using both environment data (real experience) and data generated from the learned model (simulated experience)



Tabular Dyna-Q

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Loop forever:
(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow \varepsilon$-greedy(S,Q)
(c) Take action A; observe resultant reward, R, and state, S'
(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
(f) Loop repeat n times:
$S \leftarrow$ random previously observed state
$A \leftarrow$ random action previously taken in S
$R, S' \leftarrow Model(S,A)$
$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

Background Planning

Environment Data Augmentation

Model-Based Policy Optimization

- Use collected data to learn $p_\theta(s', r | s, a)$, i.e., a predictive model of the environment (transition model)
- Apply traditional policy gradient methods on synthetic model rollouts
- Take action in real environment

Algorithm 2 Model-Based Policy Optimization with Deep Reinforcement Learning

Initialize policy π_ϕ, predictive model p_θ, environment dataset $\mathcal{D}_{\text{env}}$, model dataset $\mathcal{D}_{\text{model}}$
for N epochs **do**
  Train model p_θ on $\mathcal{D}_{\text{env}}$ via maximum likelihood
  for E steps **do**
    Take action in environment according to π_ϕ; add to $\mathcal{D}_{\text{env}}$
    for M model rollouts **do**
      Sample s_t uniformly from $\mathcal{D}_{\text{env}}$
      Perform k-step model rollout starting from s_t using policy π_ϕ; add to $\mathcal{D}_{\text{model}}$
    for G gradient updates **do**
      Update policy parameters on model data: $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi, \mathcal{D}_{\text{model}})$

Janner et al.: When to Trust Your Model: Model-Based Policy Optimization. NeurIPS 2019.

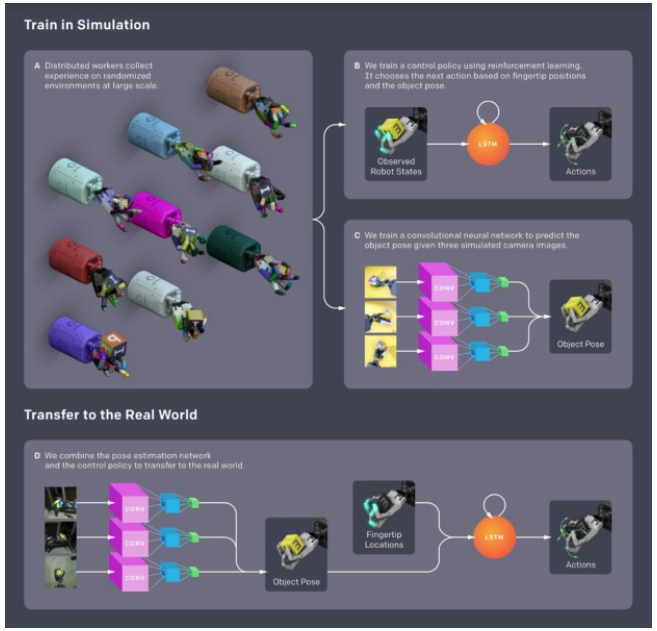
Background Planning

Environment Data Augmentation

Domain Randomization & Sim2Real

- If we have an available simulator (model), we can train an RL agent there
- But the simulation will always be different compared to the real system
- Solution: learn a good policy on a *"distribution of similar environments"*, differing in some physical parameters (e.g., masses or image textures)
- This way, the real system will be "another variation" for the policy

- *Note: seems super-simple but works remarkably in practice!*



Andrychowicz, OpenAI: Marcin, et al. "Learning dexterous in-hand manipulation." The International Journal of Robotics Research 39.1 (2020): 3-20.

Fraunhofer IIS

Outline

Background Planning

- Motivation: why model-based RL?
- What is a model? What are its inputs? What is a good model?
- **How can we use a model?**
  - **Background Planning**
    - Environment data augmentation / simulation
    - **Sample-efficient policy learning**
  - Online Planning
    - Discrete Actions
    - Continuous Actions
  - Auxiliary tasks
- Real-world application

Background Planning

Sample-efficient Policy Learning

Idea: train model and policy jointly end-to-end

- In other words: do what successfully worked in other domains (such as computer vision, speech recognition, etc.)

- Goal: maximize reward of parametric policy:

$$J(\theta) = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t), \qquad \text{with } a_t = \pi_\theta(s_t) \text{ and } s_{t+1} = T(s_t, a_t)$$

- Just apply gradient ascent on policy gradient $\nabla_\theta J$.

- But how to calculate $\nabla_\theta J$?
- Remember REINFORCE
  - High-variance
  - Requires stochastic policy

Background Planning

Sample-efficient Policy Learning

We can do more!

- Smooth models offer derivatives:

$$s_{t+1} = f_s(s_t, a_t) \qquad r_t = f_r(s_t, a_t)$$

$$\nabla_{s_t}(s_{t+1}), \nabla_{a_t}(s_{t+1}), \nabla_{s_t}(r_t), \nabla_{a_t}(r_t), \ldots$$

- How do small changes in action affect the next state?
- How do small changes in states affect the rewards?
- …

→ Allows end-to-end differentiation via backpropagation!

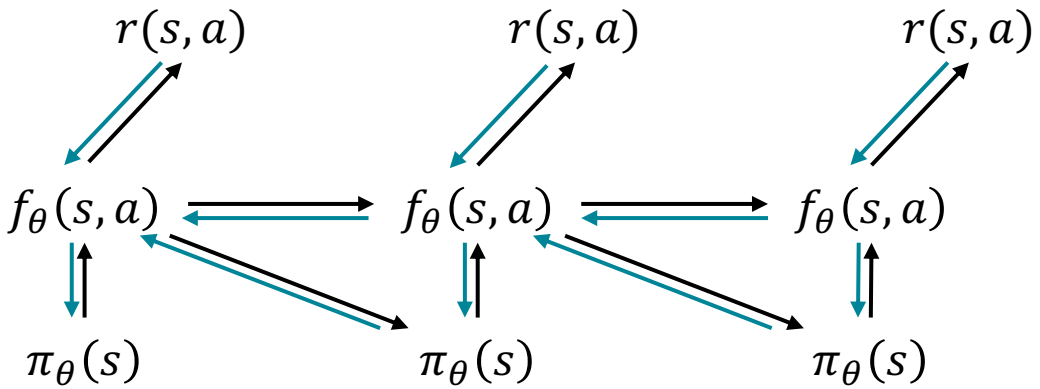
Background Planning

Policy Backprop

Back-propagate through the model to optimize the policy

Backprop:

$$\max_{\theta} \sum_{t} \gamma^t R(s_t, a_t)$$



Simple Algorithm:

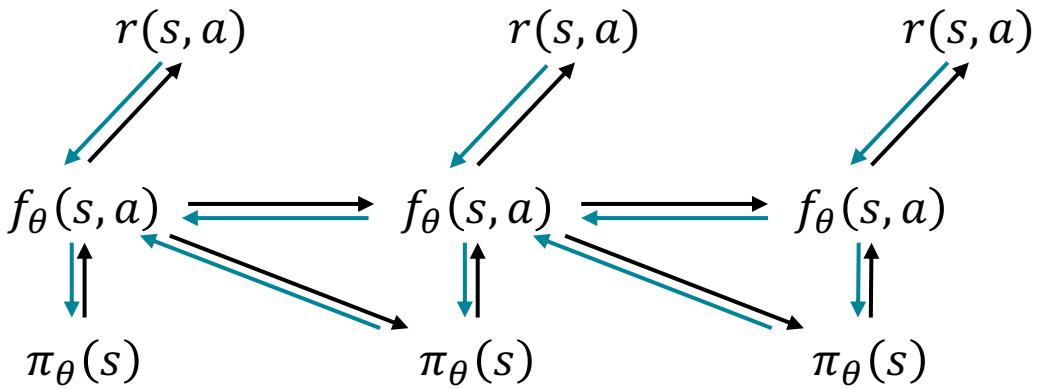
1. Run a base policy $\pi_0(a_t|s_t)$ (e.g., a random policy) to collect data samples $\mathcal{D}\{(s,a,s')_i\}$
2. Learn a dynamics model $f_\theta(s,a)$ by minimizing $\sum_i \|f_\theta(s_i,a_i) - s_i'\|^2$
3. Backpropagate through $f_\theta(s,a)$ into policy to optimize $\pi_\theta(a_t|s_t)$

Remember: Distribution Mismatch!

Background Planning

Policy Backprop

Back-propagate through the model to optimize the policy



Backprop:

$$\max_\theta \sum_t \gamma^t R(s_t, a_t)$$

Better Algorithm:

1. Run a base policy $\pi_0(a_t|s_t)$ (e.g., a random policy) to collect data samples $\mathcal{D}\{(s,a,s')_i\}$
2. Learn a dynamics model $f_\theta(s,a)$ by minimizing $\sum_i \|f_\theta(s_i,a_i) - s_i'\|^2$
3. Backpropagate through $f_\theta(s,a)$ into policy to optimize $\pi_\theta(a_t|s_t)$
4. Run $\pi_\theta(a_t|s_t)$
5. Append visited tuples (s,a,s') to $\mathcal{D}$

Background Planning

Policy Backprop

Back-propagate through the model to optimize the policy

1. Approximate transitions and rewards with differentiable models
2. Calculate policy gradient via back-prop-through-time (BPTT)

- **Pros:**
  - Long-term credit assignment
  - Differentiable transitions and rewards models → sample efficiency
  - Principles behind BPTT well understood
  - Deterministic & no variance involved

- **Cons:**
  - Similar problems to training long RNNs with BPTT → poor conditioning
    - Vanishing and exploding gradients
    - Unlike LSTM, we cannot just "choose" simple dynamics as dynamics are chosen by nature.
  - Prone to local minima

Outline

Online Planning

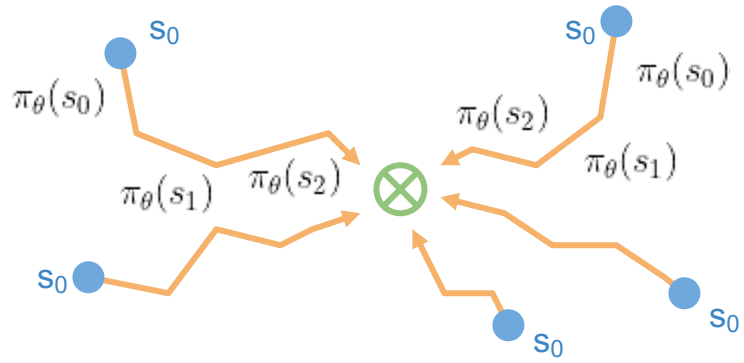
- Motivation: why model-based RL?
- What is a model? What are its inputs? What is a good model?
- **How can we use a model?**
  - Background Planning
    - Environment data augmentation / simulation
    - Sample efficient policy learning
  - **Online Planning**
    - Discrete Actions
    - Continuous Actions
  - Auxiliary tasks
- Real-world application

Online Planning

Background planning vs. Decision-Time planning

Background Planning

"Learn how to act for **any** situation"



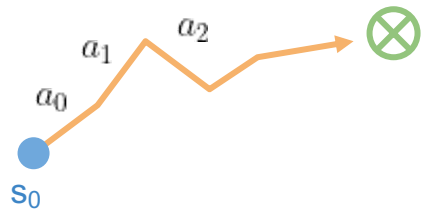
Optimization variables: shared parameters θ
(parameters of policy, or value, or …)

e.g.: $J(\theta) = \mathbb{E}_{s_0}[\sum_{t=0}^{H} \gamma^t r_t],\ a_t = \pi_\theta(s_t)$

Trained in expectation over all states

Decision-Time Planning

"Find best sequence of actions for my **current** situation"



Optimization variables: $a_0, a_1, \ldots, a_H$
(sequence of actions and/or states)

e.g.: $J(a_0, \ldots, a_H) = \sum_{t=0}^{H} \gamma^t r_t$

Optimizes only on a particular action sequence

https://sites.google.com/view/mbrl-tutorial
Sutton and Barto: Reinforcement Learning: An Introduction.

Online Planning

Background planning vs. Decision-Time planning

	Background Planning	Decision-Time Planning
Act on most recent state of the world	-	+
Act without any learning	-	+
Competent in unfamiliar situations	-	+
Independent of observation space	-	+
Partial observability	+	+ / -
Fast computation at deployment	+	-
Predictability and coherence	+	-
Same for discrete and continuous actions	+	-

Background

$$J(\theta) = \mathbb{E}_{s_0}[\sum_{t=0}^{H} \gamma^t r_t], \quad a_t = \pi_\theta(s_t)$$

joint angles? pixels? graphs?

Decision-Time

$$J(a_0, ..., a_H) = \sum_{t=0}^{H} \gamma^t r_t$$

https://sites.google.com/view/mbrl-tutorial

Outline

Online Planning with Discrete Actions

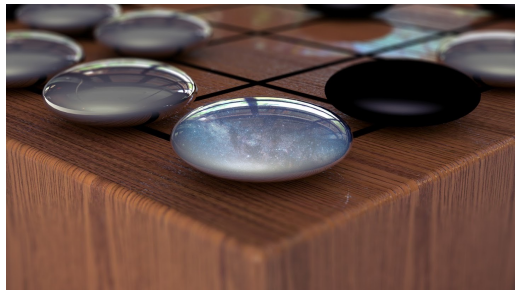
- Motivation: why model-based RL?
- What is a model? What are its inputs? What is a good model?
- **How can we use a model?**
  - Background Planning
    - Environment data augmentation / simulation
    - Sample efficient policy learning
  - **Online Planning**
    - **Discrete Actions**
    - Continuous Actions
  - Auxiliary tasks
- Real-world application

Online Planning with Discrete Actions

Discrete Actions

Environments with discrete actions

- Here we usually have problems with discrete actions and states (e.g., games, supply chains, production optimization, …)
- Their dimensionality is so large that it is impossible to enumerate the solutions

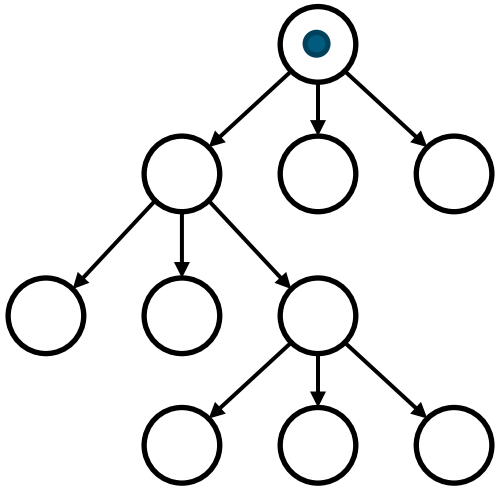


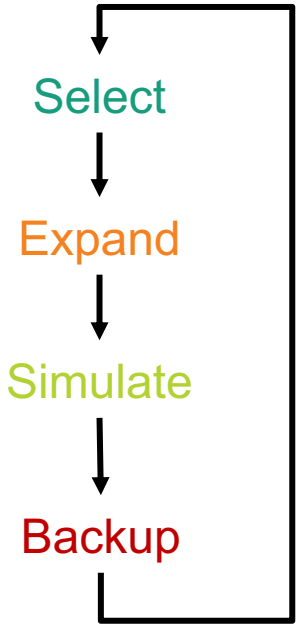
https://deepmind.com/research/case-studies/alphago-the-story-so-far

Online Planning with Discrete Actions

Monte Carlo Tree Search

- In the heart of modern online planning methods lies **Monte Carlo Tree Search (MTSC)**
- This is the work-horse behind Alpha Go and all its derivative work





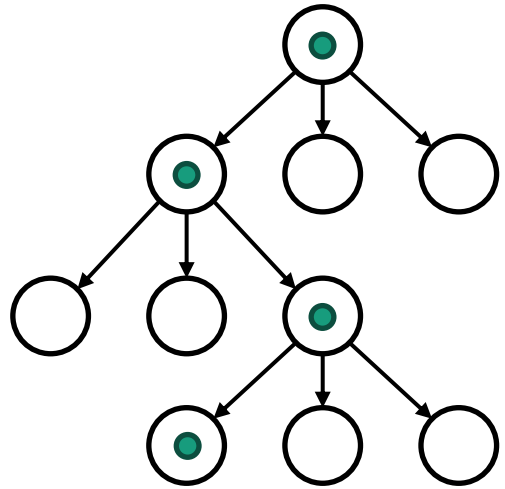
Online Planning with Discrete Actions

Discrete Actions: Monte Carlo Tree Search

1. Selection
   - How do we do selection in an MDP?
     → We follow a policy π_s
   - In nodes we have seen before, select action according to UCT[1] (UCB/Bandits):

$$a \sim V_i + C\sqrt{\frac{\ln(N)}{n_i}}$$

     - V_i is value estimate at node i
     - C is tunable exploration parameter
     - N is number of visits of parent node
     - n_i is number of visits at node i
   - We do this until we reach a leaf node (where we don't know what to do next)

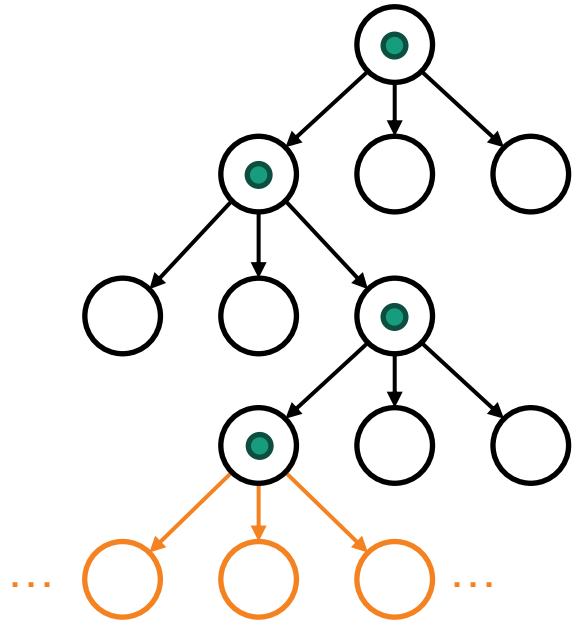


[1] *Upper Confidence bounds applied to Trees*

Online Planning with Discrete Actions

Discrete Actions: Monte Carlo Tree Search

1. Selection
2. Expansion
   - We know the (transition) model so we know where actions lets us end up in (or we do a bunch of one-step simulations to know successor states)

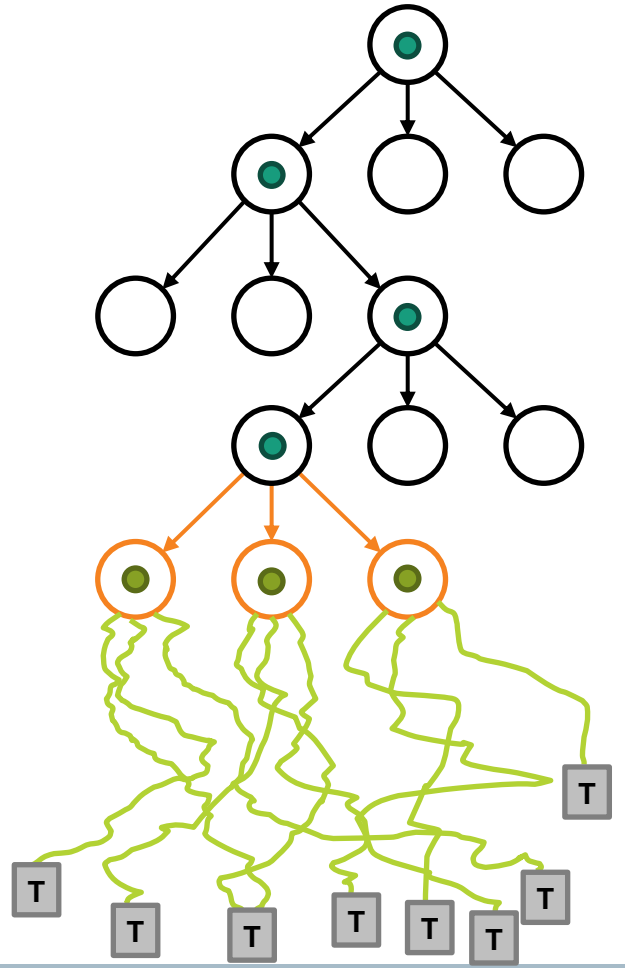


Online Planning with Discrete Actions

Discrete Actions: Monte Carlo Tree Search

1. Selection
2. Expansion
3. Simulation
   - Randomly **choose** a new child node
   - Play **randomly** (i.e., with a random rollout policy π_r) until game finishes

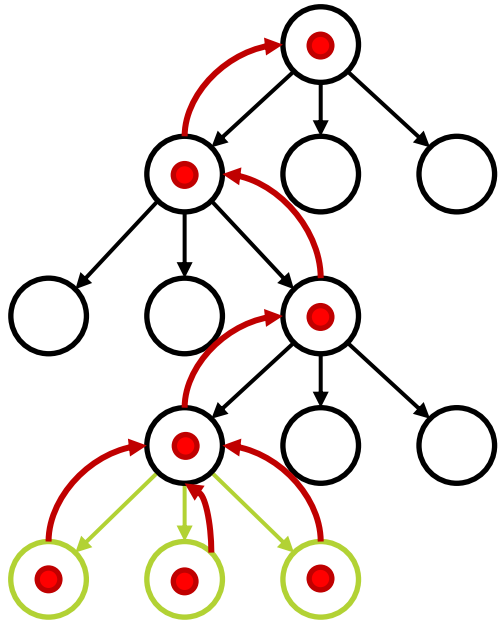
Doing all these Monte-Carlo rollouts gives us $\hat{Q}(s, a)$



Online Planning with Discrete Actions

Discrete Actions: Monte Carlo Tree Search

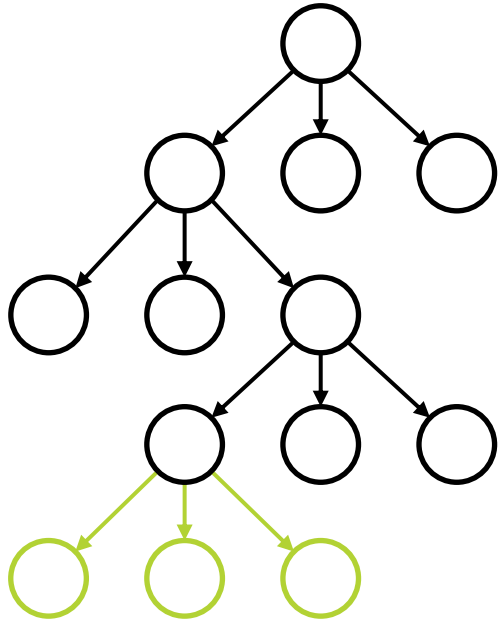
1. Selection
2. Expansion
3. Simulation
4. Backup
   - We now have estimates of $\hat{Q}(s, a)$ of the leaf nodes
   - Backup Q-values and number of visits all the way up to the root of the tree



Online Planning with Discrete Actions

Discrete Actions: Monte Carlo Tree Search

1. Selection
2. Expansion
3. Simulation
4. Backup
   - We now have estimates of $\hat{Q}(s,a)$ of the leaf nodes
   - Backup Q-values and number of visits all the way up to the root of the tree
   - In fact, we now know a lot more about the tree!



We so far used two policies:

- A selection policy π_s (that uses Q-values where we already have knowledge about)
- A random simulation policy π_r (for leaves onwards where we don't know anything)
- We can update our selection policy π_s now!
- π_s is getting deeper throughout the iterations of the algorithm and more accurate with backing up the information of leave nodes over time

Online Planning with Discrete Actions

Discrete Actions: Monte Carlo Tree Search

- Monte Carlo Tree Search is a planning algorithm, and we either
  - need a transition model, or we
  - need a way to do the simulation
- **But should we stop?**
  - In principle, we could run MCTS forever
  - There is a bunch of ways how to integrate planning and executing:
    1. Run MCTS until convergence starting from current real-world state
    2. take best action in real world
    3. go back to 1.
  - We could specify a maximal computation time per MCTS run
  - or: we solve the whole MDP and then run the tree greedily in real-world…

Online Planning with Discrete Actions

Discrete Actions: Monte Carlo Tree Search

- **One more thing about the rollout policy π_r:**
  - We approximate Q-values under the assumption that we are greedy w.r.t. π_s in knowledgeable states and that we behave randomly in simulation (→ we underestimate Q!)
  - Encode goal/success conditions and fail-by-termination (i.e., constraints)
    Example from Pac-Man: eating dots vs. avoiding ghosts

- **One more thing about UCT and the value range of Q:**
  - General: Selection with UCT: $a = \text{argmax}_i \; V_i + C\sqrt{\frac{\ln(N)}{n_i}}$
  - In many cases: $a = argmax_i \frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}}$

„winning“ ratio

In two-player zero sum games the value functions are assumed to be bounded within the $[0, 1]$ interval. This choice allows us to combine value estimates with probabilities using the pUCT rule (Eqn 2). However, since in many environments the value is unbounded, it is necessary to adjust the pUCT rule. A simple solution would be to use the maximum score that can be observed in the environment to either re-scale the value or set the pUCT constants appropriately [33]. However, both solutions are game specific and require adding prior knowledge to the *MuZero* algorithm. To avoid this, *MuZero* computes normalized Q value estimates $\overline{Q} \in [0, 1]$ by using the minimum-maximum values observed in the search tree up to that point. When a node is reached during the selection stage, the algorithm computes the normalized $\overline{Q}$ values of its edges to be used in the pUCT rule using the equation below:

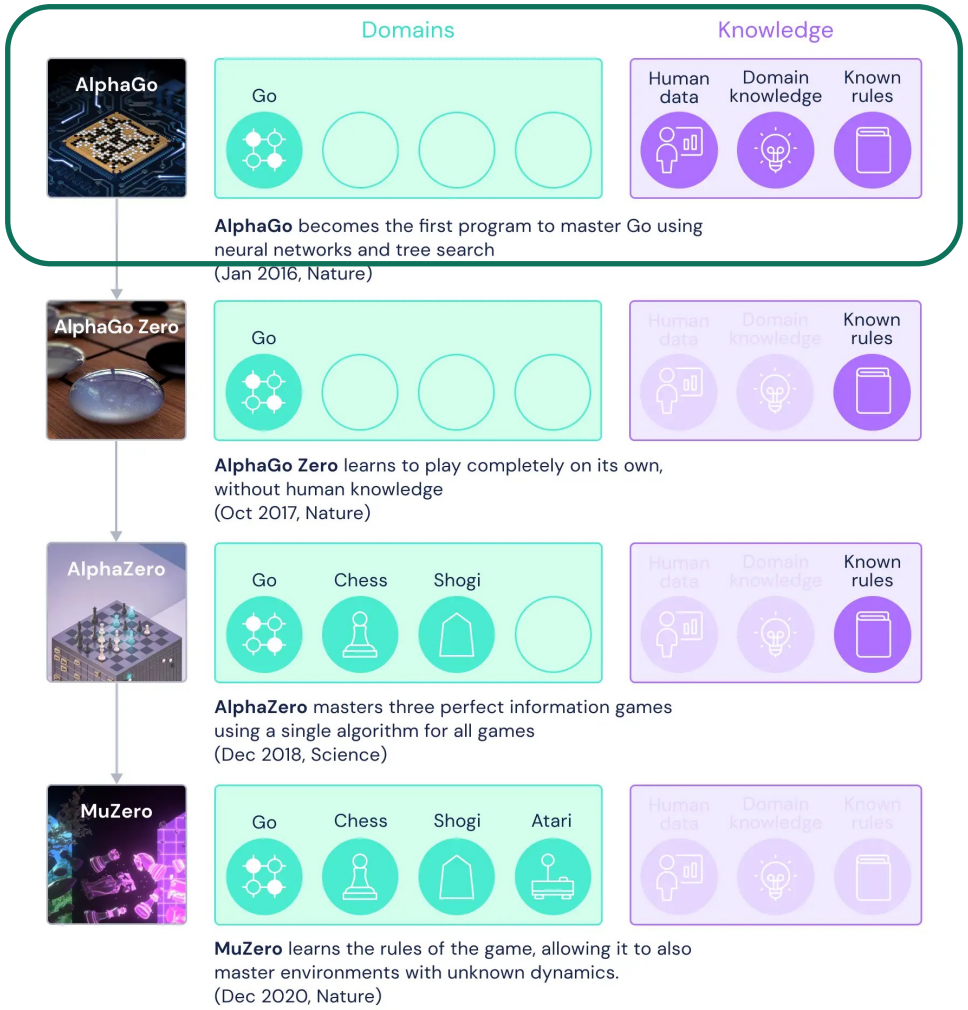
$$\overline{Q}(s^{k-1}, a^k) = \frac{Q(s^{k-1}, a^k) - \min_{s,a \in Tree} Q(s, a)}{\max_{s,a \in Tree} Q(s, a) - \min_{s,a \in Tree} Q(s, a)} \tag{5}$$

[3] In board games the discount is assumed to be 1 and there are no intermediate rewards.

Schrittwieser, Julian, et al. "Mastering atari, go, chess and shogi by planning with a learned model." Nature 588.7839 (2020): 604-609.

Online Planning with Discrete Actions

MCTS in practice: AlphaGo & Derivatives

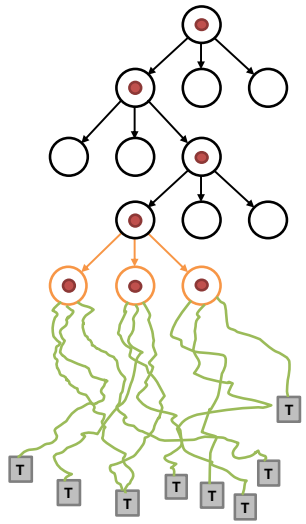


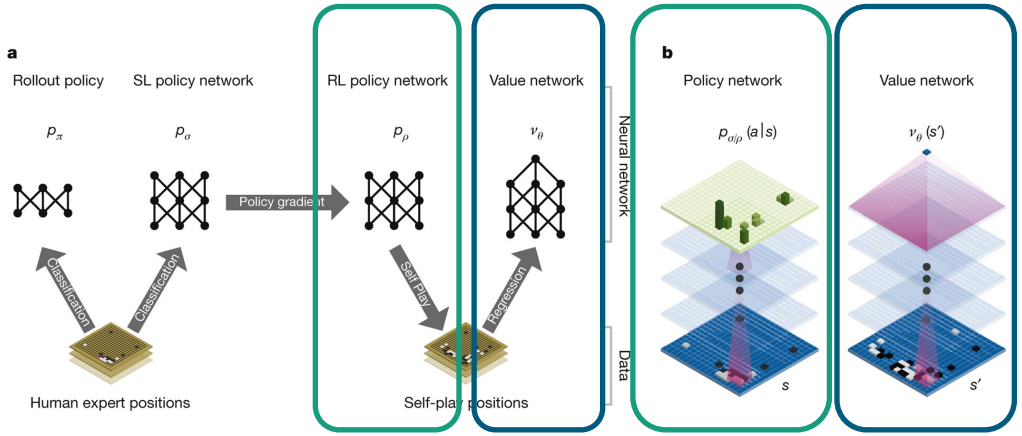
https://deepmind.com/blog/article/muzero-mastering-go-chess-shogi-and-atari-without-rules

Online Planning with Discrete Actions

MCTS in practice: AlphaGo

- Large breadth and depth ($b \approx 250, d \approx 150$)
- Main idea: enhance MCTS with a learning component
- RL Policy network: selects actions
- RL Value network: predicts win/lose for each position



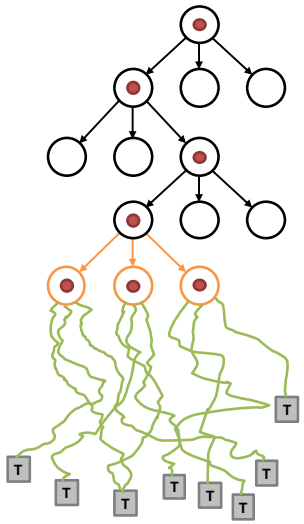


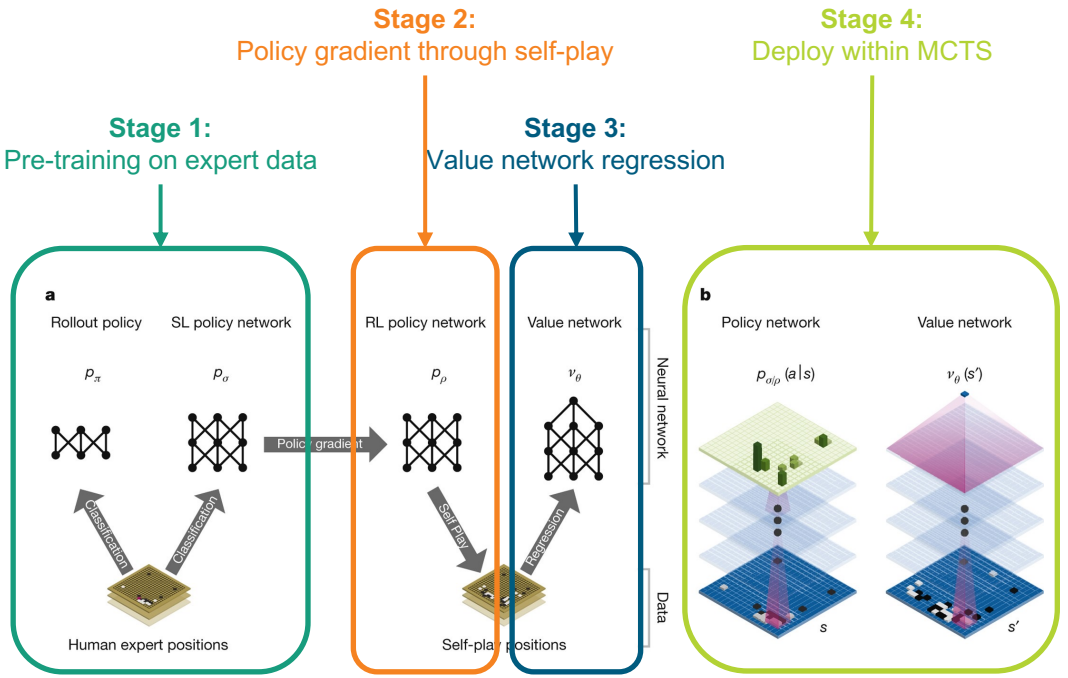
Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489.
https://www.davidsilver.uk/wp-content/uploads/2020/03/unformatted_final_mastering_go.pdf

Online Planning with Discrete Actions

MCTS in practice: AlphaGo

- Large breadth and depth ($b \approx 250, d \approx 150$)
- Main idea: enhance MCTS with a learning component





Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489.
https://www.davidsilver.uk/wp-content/uploads/2020/03/unformatted_final_mastering_go.pdf

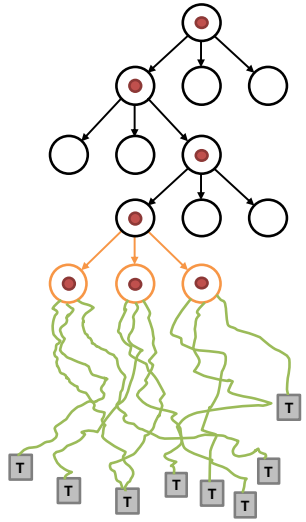
Online Planning with Discrete Actions

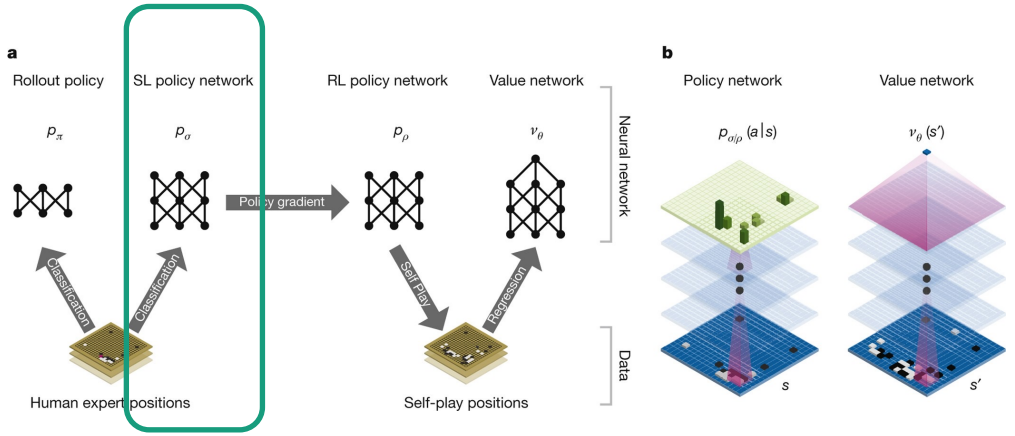
MCTS in practice: AlphaGo

- Supervised Learning Policy network p_σ
  - Goal: predict good actions in states: maximize likelihood by SGD:

$$\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial\sigma}$$

  - Trained with state-action pairs from human experts (30M positions from KGS Go Server)
  - 13-layer neural networks with conv-layers and ReLUs
  - Accuracy of 57.0% (SotA was 44% at that time)
  - 4 weeks training time on 50 GPUs using Google Cloud[1]





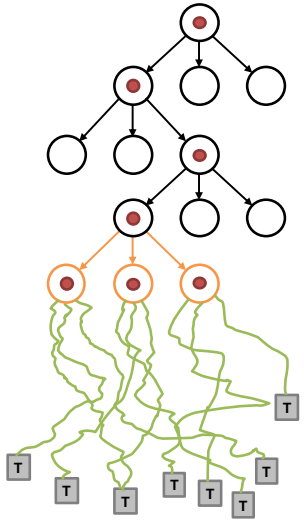
Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489.
https://www.davidsilver.uk/wp-content/uploads/2020/03/unformatted_final_mastering_go.pdf

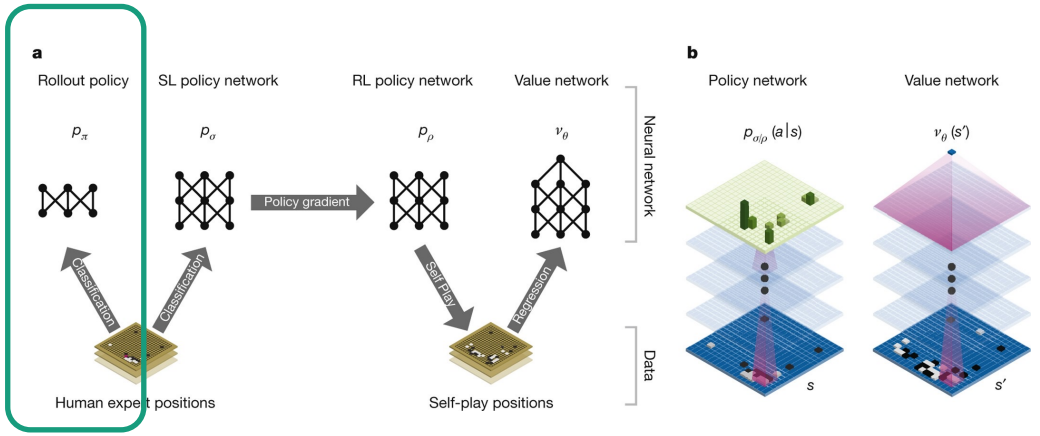
[1] *https://www.davidsilver.uk/wp-content/uploads/2020/03/AlphaGo-tutorial-slides_compressed.pdf*

Online Planning with Discrete Actions

MCTS in practice: AlphaGo

- Rollout policy network p_π
  - same initialization and training but faster and less accurate
  - network only uses linear softmax of small (hand-crafted) input features
  - Accuracy of 24.2%
  - Forward pass only takes $2\mu s$ (instead of 3ms for p_σ)





Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489.
https://www.davidsilver.uk/wp-content/uploads/2020/03/unformatted_final_mastering_go.pdf

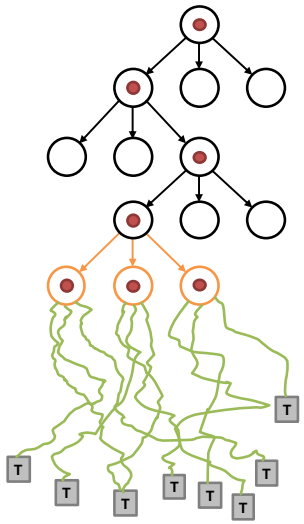
Online Planning with Discrete Actions

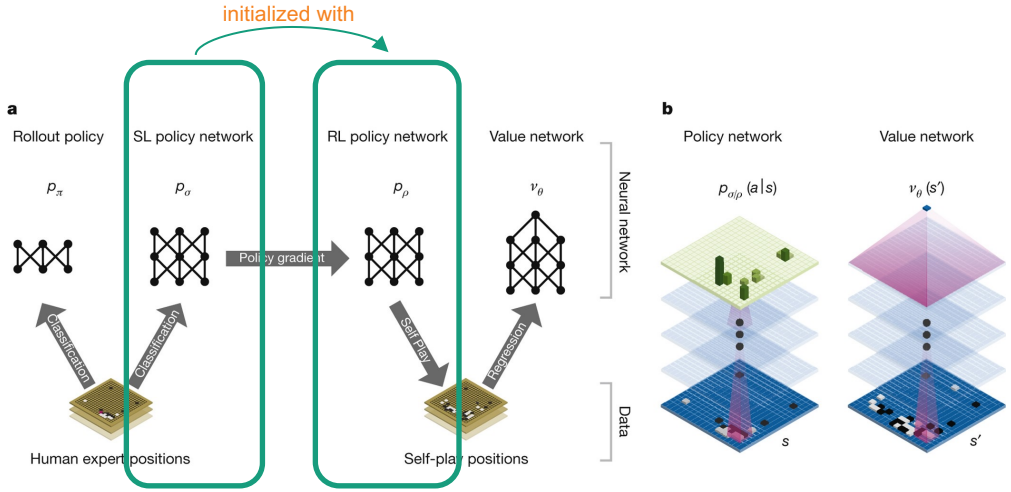
MCTS in practice: AlphaGo

- Reinforcement Learning of policy network p_ρ: self-play with random ancestor of p_ρ
  - Goal: maximize wins z_t by policy gradient RL:

$$\Delta\rho \propto \frac{\partial \log p_\rho(a_t|s_t)}{\partial \rho} z_t$$

  - Reward is 0 everywhere and $z = \pm 1$ at the end depending on outcome
  - This already yields a very strong player: wins 80% against p_σ, 85% against Piachi[1]
  - 1 week training time on 50 GPUs using Google Cloud[1]





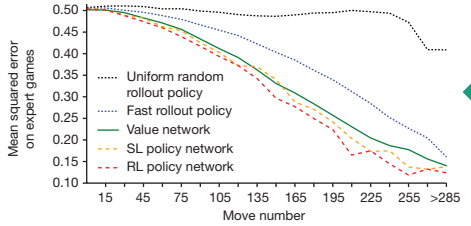
Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489.
https://www.davidsilver.uk/wp-content/uploads/2020/03/unformatted_final_mastering_go.pdf

[1] *as of 2016 the strongest open-source Go program a sophisticated Monte Carlo search program, ranked at 2 amateur dan on KGS*

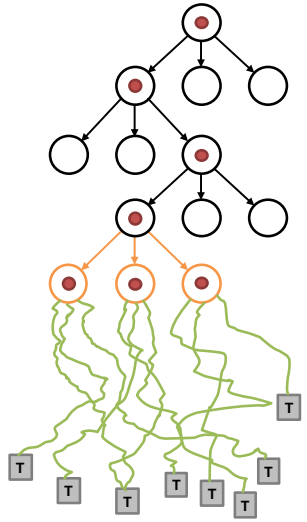
[1] *https://www.davidsilver.uk/wp-content/uploads/2020/03/AlphaGo-tutorial-slides_compressed.pdf*

Online Planning with Discrete Actions

MCTS in practice: AlphaGo



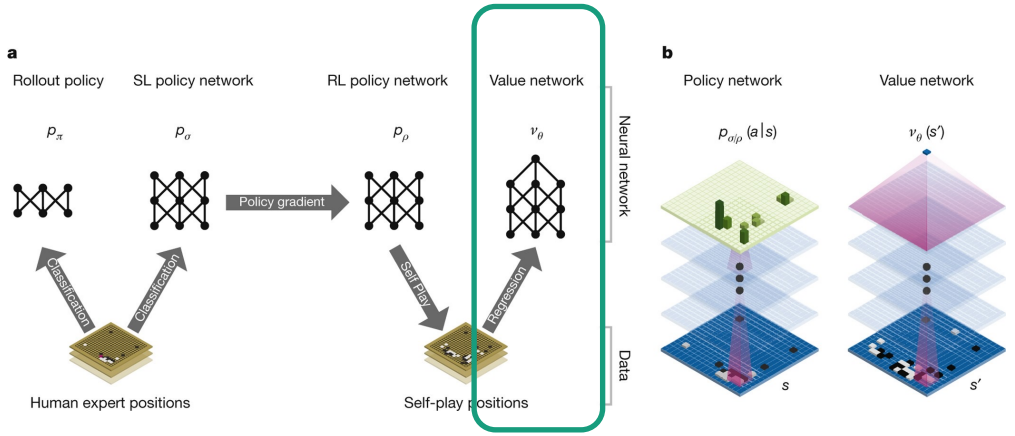
Accuracy of the position evaluation of value function vs. MC rollouts



- Reinforcement Learning of value network v_θ: predict outcome from position under p_ρ
  - Goal: minimize MSE on state-outcome pairs (s, z) and using SGD:

$$\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial\theta}(z - v_\theta(s))$$

  - 30M games of self-play as input
  - Neural network architecture like that of p_ρ but outputs single value
  - 1 week training time on 50 GPUs using Google Cloud



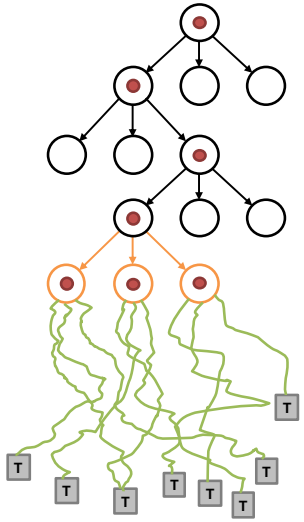
Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489.
https://www.davidsilver.uk/wp-content/uploads/2020/03/unformatted_final_mastering_go.pdf

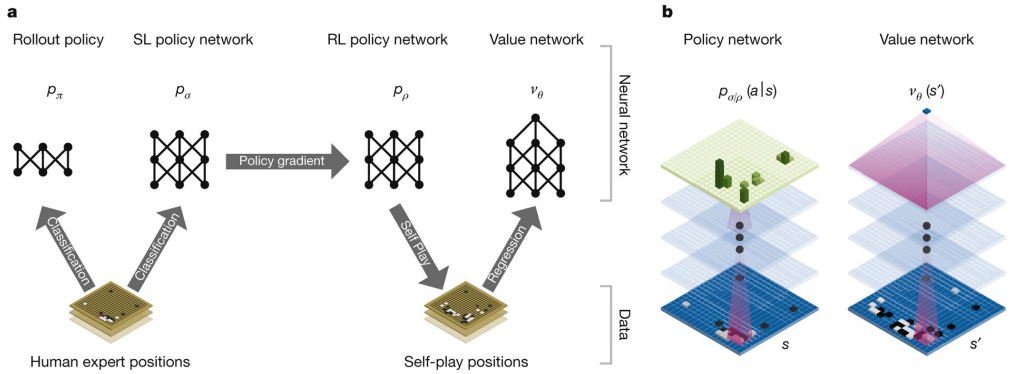
Online Planning with Discrete Actions

MCTS in practice: AlphaGo

Deployment:

- Policy: Online MCTS!!!





Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489.
https://www.davidsilver.uk/wp-content/uploads/2020/03/unformatted_final_mastering_go.pdf

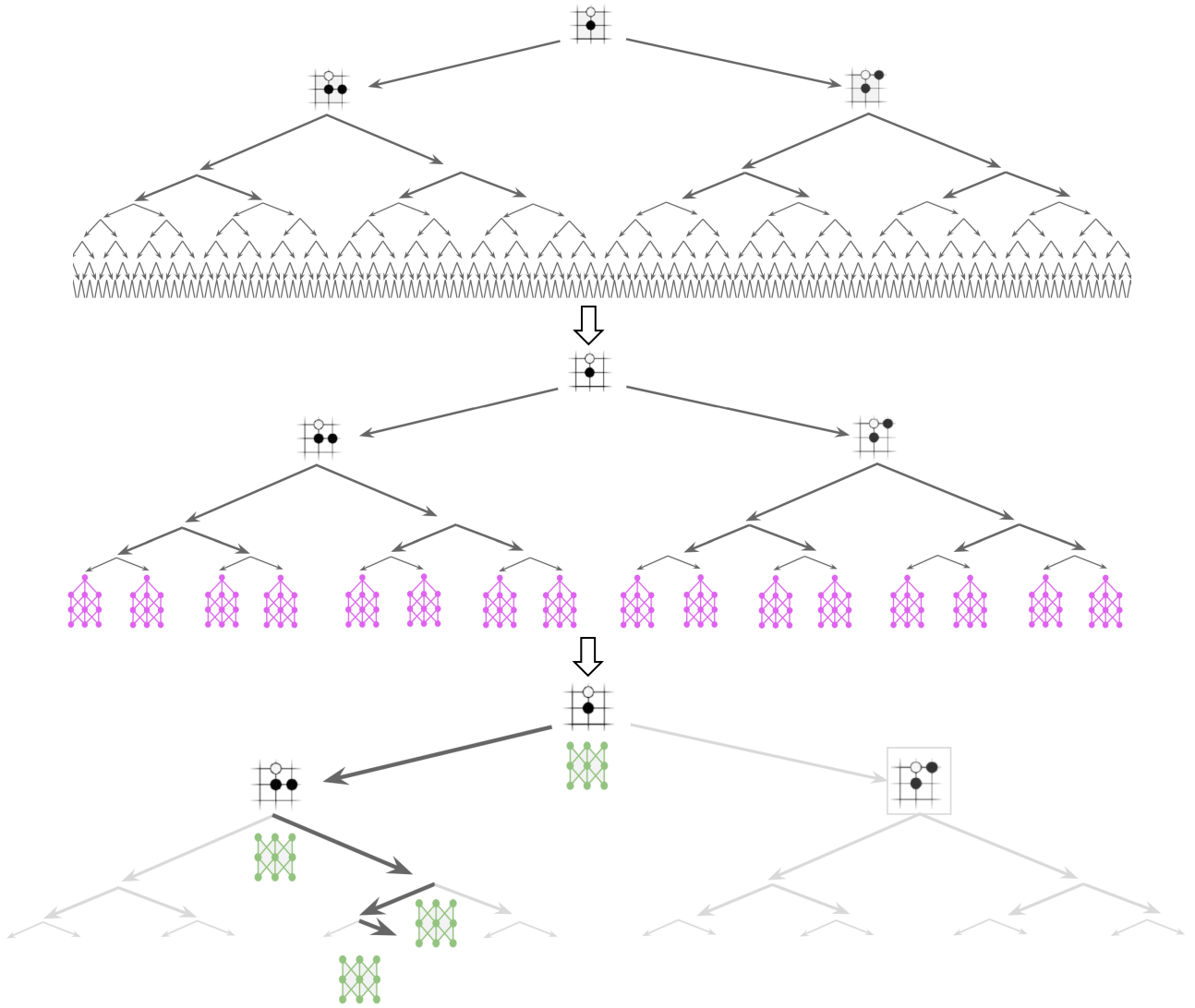
Online Planning with Discrete Actions

MCTS in practice: AlphaGo

Deployment:

- Policy: Online MCTS!!!
- Run for a pre-defined amount of time

- Search Complexity



Exhaustive search

Reducing depth with value network

Reducing breadth with policy network

Online Planning with Discrete Actions

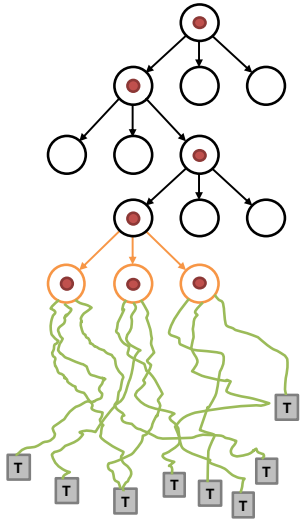
MCTS in practice: AlphaGo

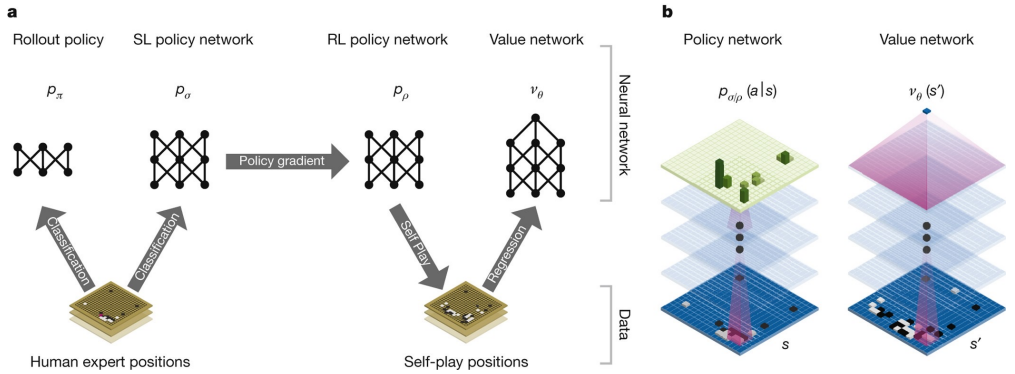
[1] UCB = upper confidence bounds, a theoretical framework to tackle exploration (see lecture on exploration strategies). We will not further go into details here. Important here is only the fact that we not simply take the argmax-action but we add an exploration bonus on top that we calculate through a state-visitation counter.

1. Selection

- In nodes we have visited before, we select which action to take based on the RL policy network action probabilities and the Q-approximation in the tree nodes, in an UCB-like process[1]
- Predictor + UCB applied to trees (PUCT) with policy network as predictor/prior:

$$\mathrm{a_t} = \mathrm{argmax_a} Q(s_t, a) + c_{puct} \cdot p_\rho\,(s)|_a \cdot \frac{\sqrt{\sum_b N(s,b)}}{1+N(s,\mathrm{a})} \qquad c_{puct} = 5$$





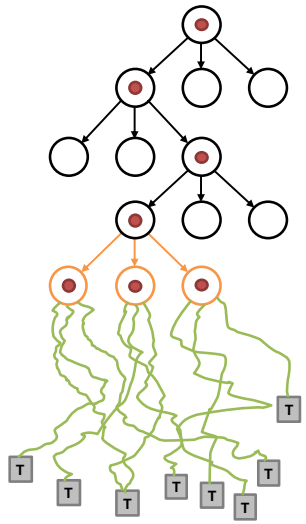
Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489.
https://www.davidsilver.uk/wp-content/uploads/2020/03/unformatted_final_mastering_go.pdf

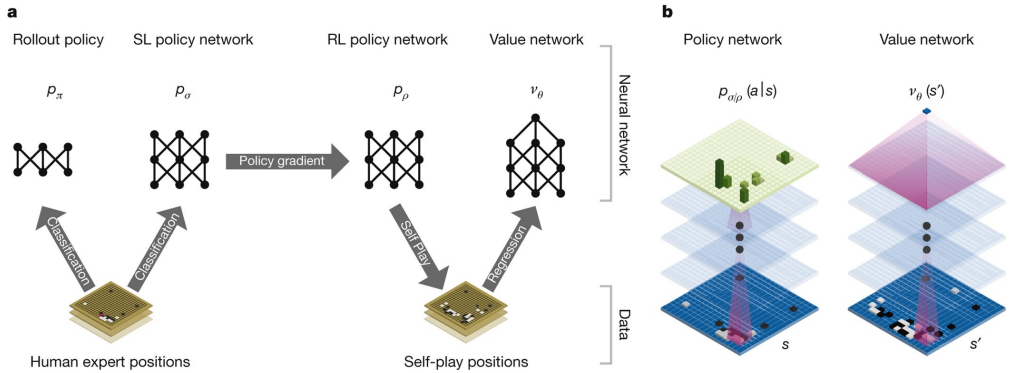
Online Planning with Discrete Actions

MCTS in practice: AlphaGo

1. Selection
2. Expansion
   - When we reach a leaf node, the SL Policy network is used to assign a prior action probability. The tree is expanded according to the most probable action of the RL Policy network

It is worth noting that the SL policy network p_σ performed better in AlphaGo than the stronger RL policy network p_ρ, presumably because humans select a diverse beam of promising moves, whereas RL optimizes for the single best move. However, the value function



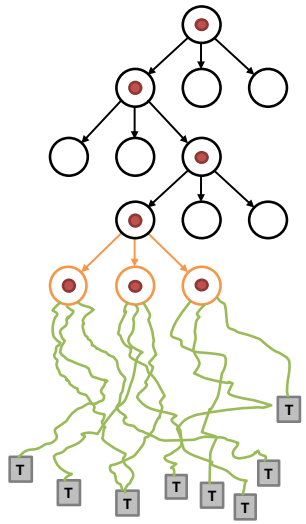


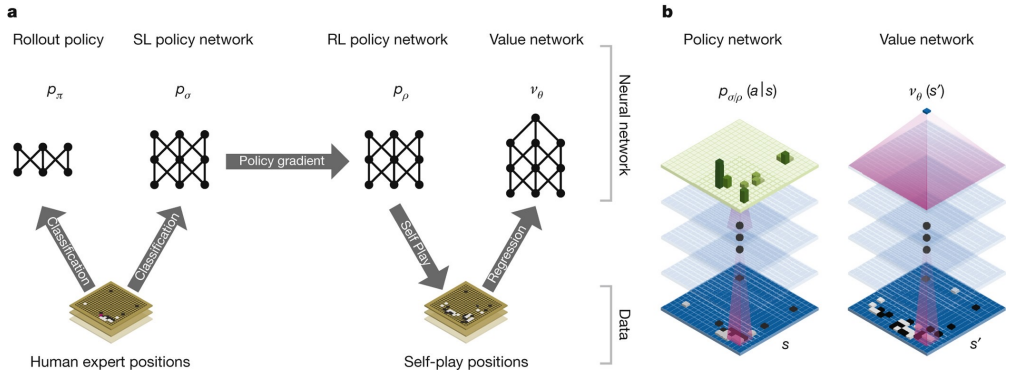
Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489.
https://www.davidsilver.uk/wp-content/uploads/2020/03/unformatted_final_mastering_go.pdf

Online Planning with Discrete Actions

MCTS in practice: AlphaGo

1. Selection
2. Expansion
3. Simulation
   - After the expansion, multiple simulations in parallel are performed
   - The "fast" rollout policy is used to play the game until it finishes.





Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489.
https://www.davidsilver.uk/wp-content/uploads/2020/03/unformatted_final_mastering_go.pdf

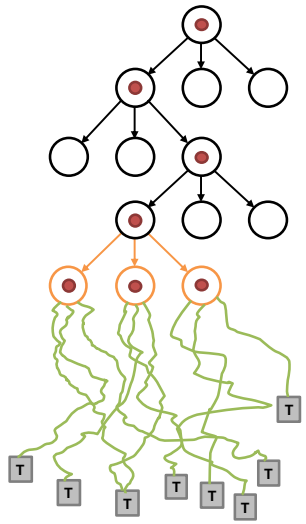
Online Planning with Discrete Actions

MCTS in practice: AlphaGo

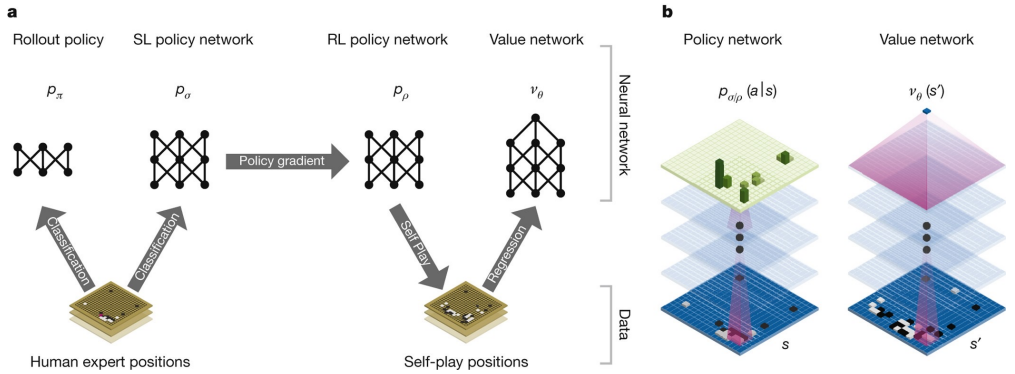
4. Backup:
   - The value of the previous leaf node that was expanded is updated combining the RL value network and the final average reward (z_l) from the rollouts:

$$V(s_L) = (1 - \lambda)v_\theta(s_L) + \lambda z_l$$

$\lambda = 0.5$



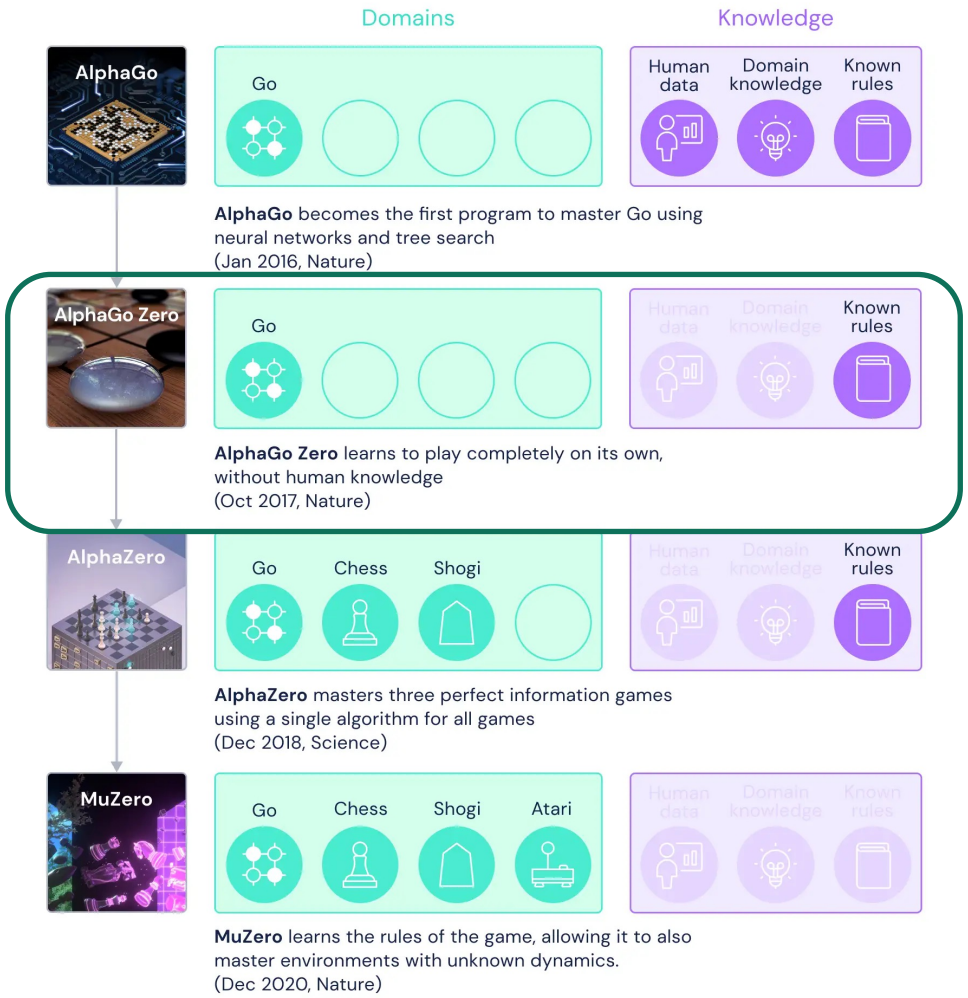
   - The Q values and visitations of the tree nodes are updated as in the classical MCTS algorithm.



Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489.
https://www.davidsilver.uk/wp-content/uploads/2020/03/unformatted_final_mastering_go.pdf

Online Planning with Discrete Actions

MCTS in practice: AlphaGo & Derivatives



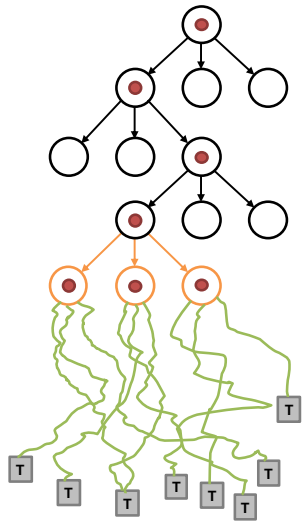
https://deepmind.com/blog/article/muzero-mastering-go-chess-shogi-and-atari-without-rules

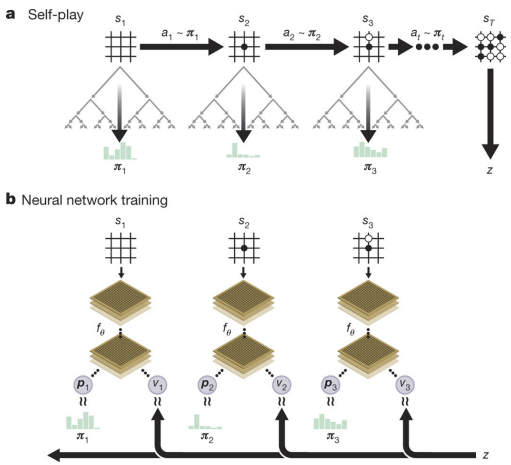
Online Planning with Discrete Actions

MCTS in practice: AlphaGo Zero

AlphaGo Zero

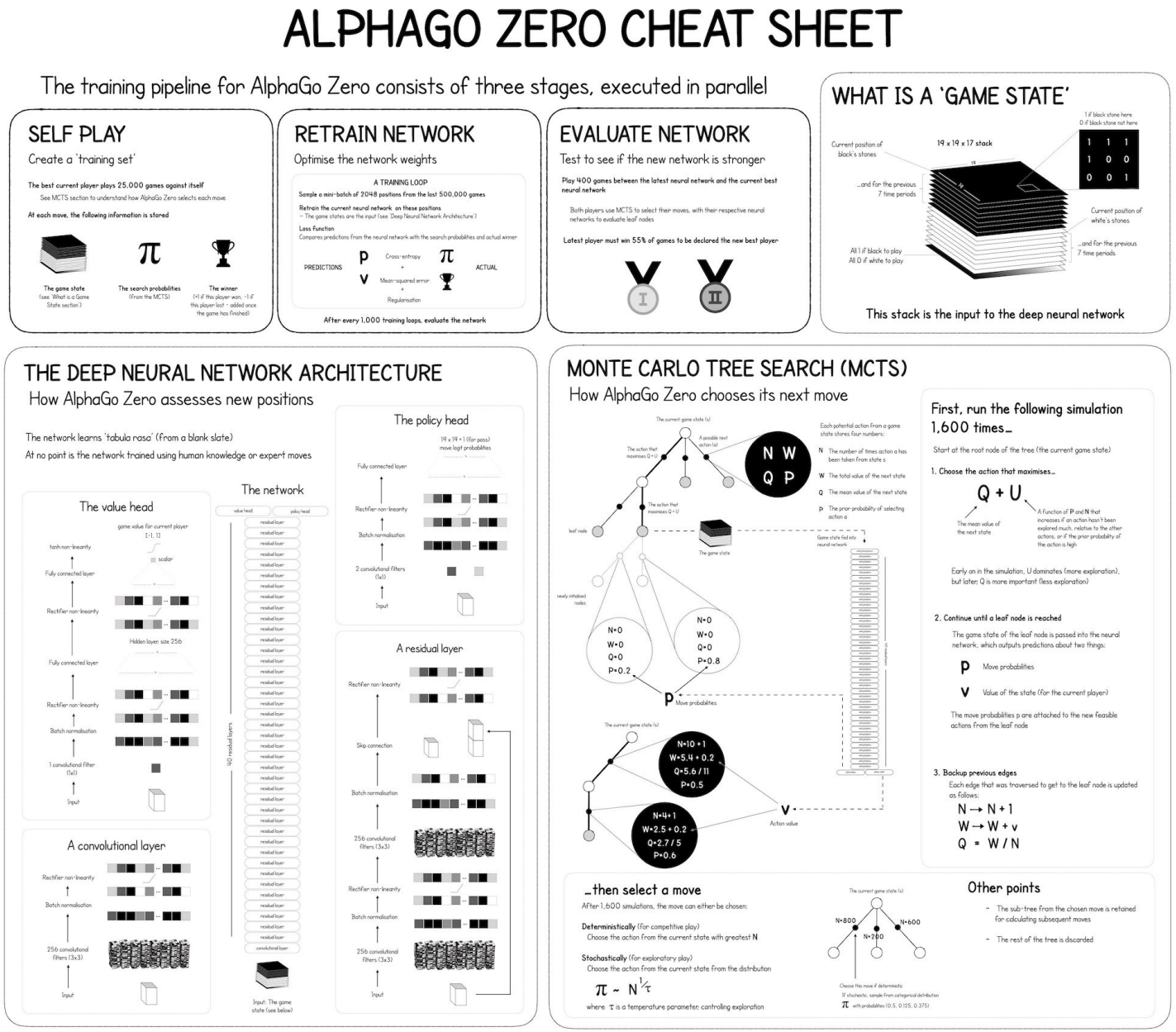
- Main differences to AlphaGo:
  1. No supervised initialization from human games
  2. Only black & white stones (no hand-crafted features)
  3. Joint RL Policy and Value network
  4. No simulation rollouts during deployment – rely only on networks
- Self-play against strongest version of the Policy/Value network
- Use MCTS also in training:
  - Train the RL Policy head of the network to mimic the powerful MCTS policy during training
  - Train the RL Value head of the network to predict the outcome of the game (as in Alpha Go)
  - **Think of it as trying to distill the power of MCTS search during training into the Policy/Value network**





Silver, David, et al. "Mastering the game of go without human knowledge." nature 550.7676 (2017): 354-359.

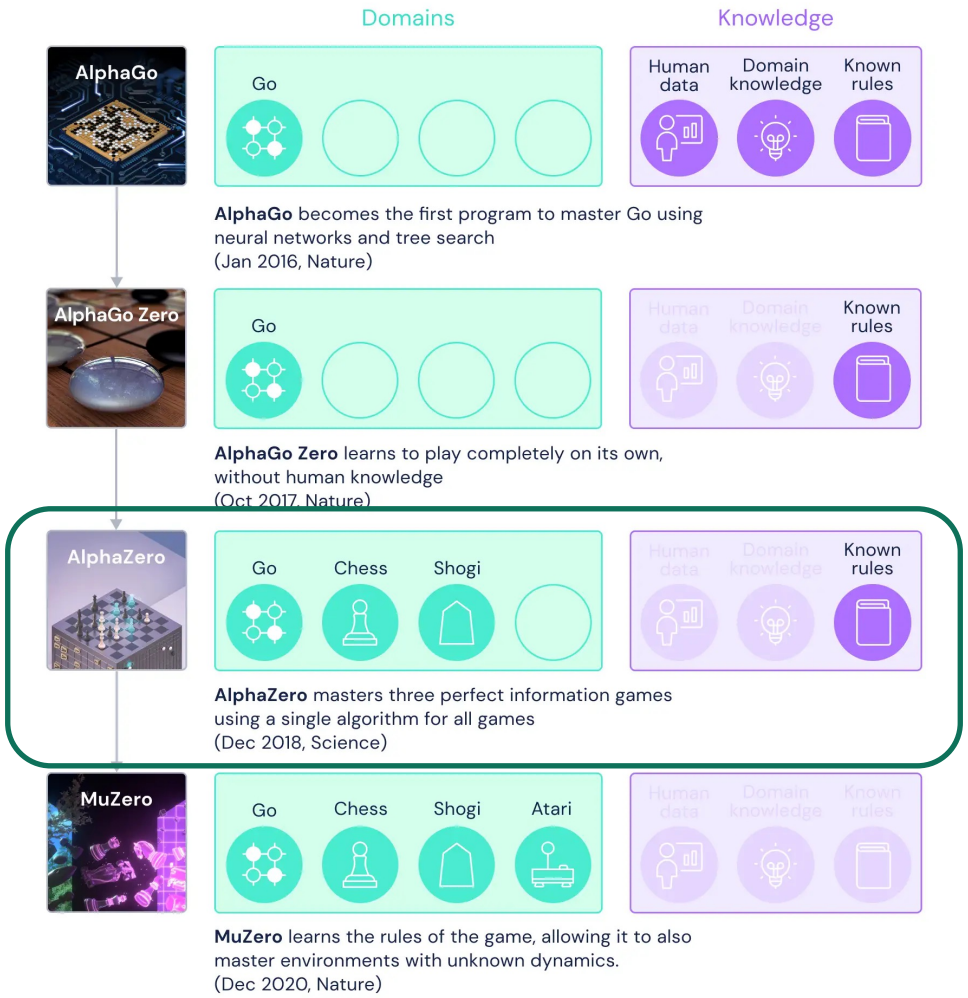
Online Planning with Discrete Actions



https://medium.com/applied-data-science/alphago-zero-explained-in-one-diagram-365f5abf67e0

Online Planning with Discrete Actions

MCTS in practice: AlphaGo & Derivatives

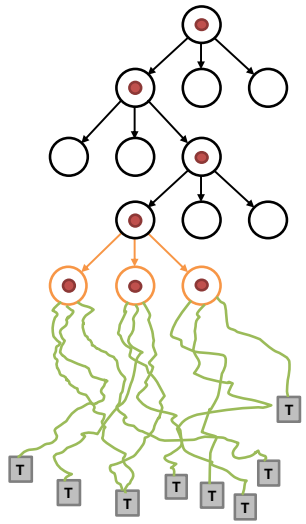


https://deepmind.com/blog/article/muzero-mastering-go-chess-shogi-and-atari-without-rules

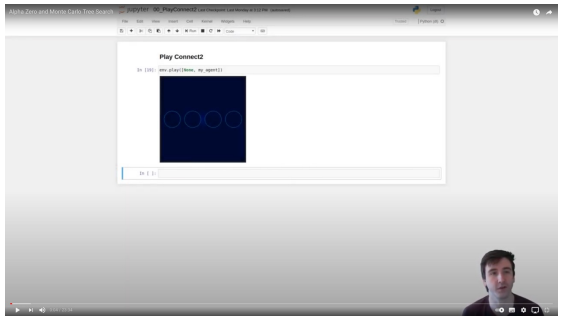
Online Planning with Discrete Actions

MCTS in practice: AlphaZero

- *The "smallest" technical jump between versions*
- Main differences to AlphaGo Zero:
  - Expand AlphaGo Zero ideas to other games
  - RL Policy and Value networks in AlphaGo & AlphaGo Zero exploited Go rules
    - Augmented data since Go board is invariant to rotation and translation
    - Go games can end with win or lose (no draw like chess). Training of AlphaGo (Zero) exploited this to estimate the probability of winning
  - Self-play is now against the current version of the networks and not against the strongest player so far.



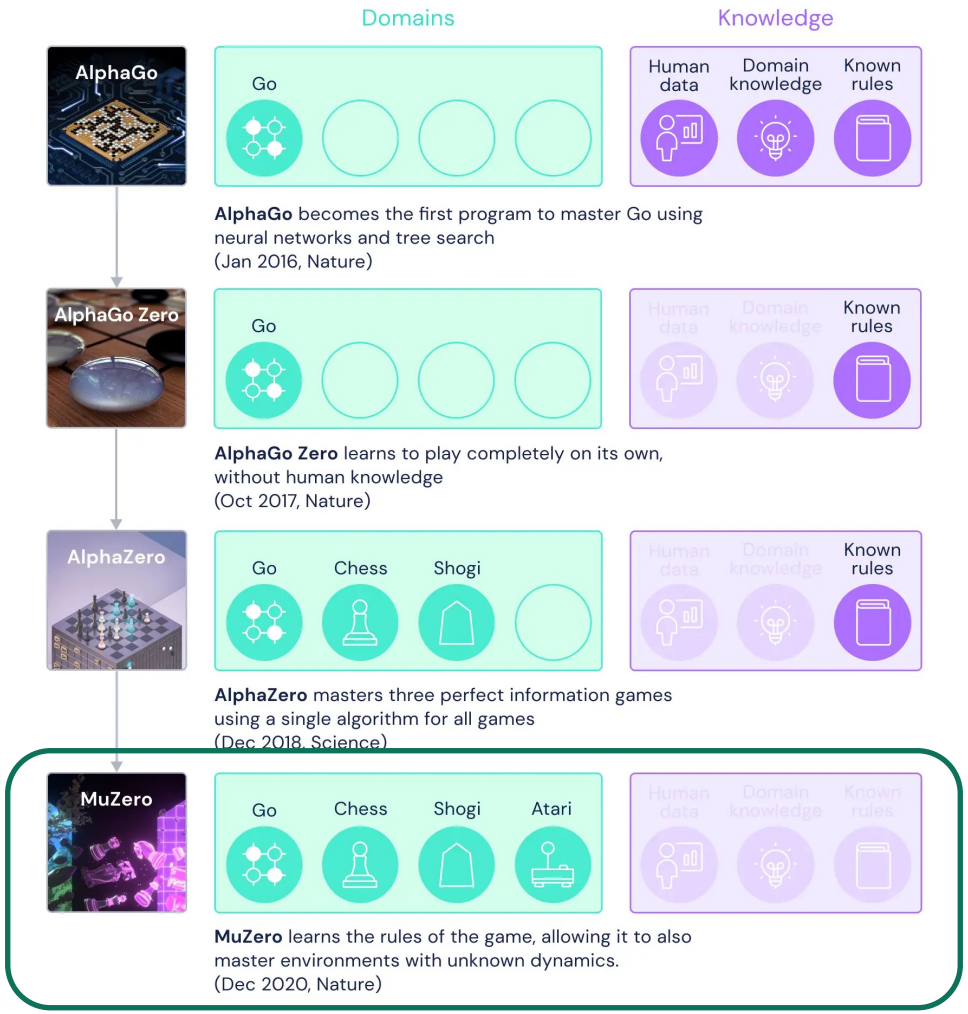
- See also "Alpha Zero and Monte Carlo Tree Search": https://www.youtube.com/watch?v=62nq4Zsn8vc



Silver, David, et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play." Science 362.6419 (2018): 1140-1144.

Online Planning with Discrete Actions

MCTS in practice: AlphaGo & Derivatives

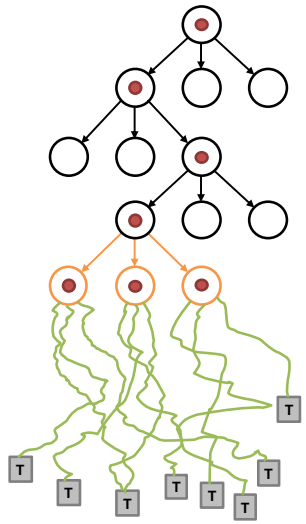


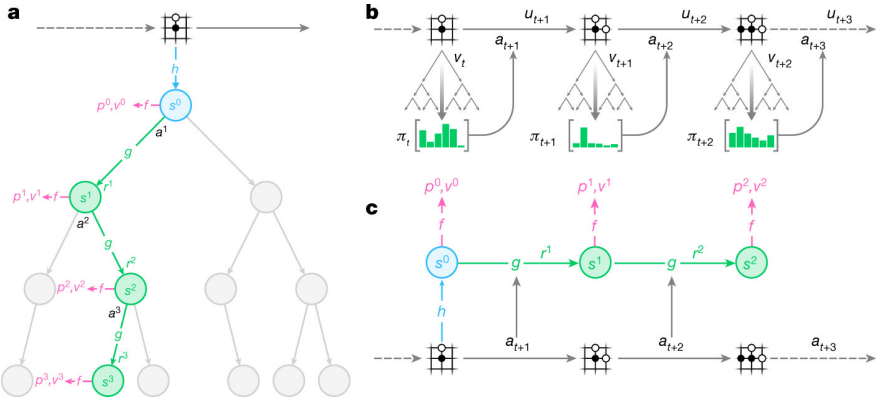
https://deepmind.com/blog/article/muzero-mastering-go-chess-shogi-and-atari-without-rules

Online Planning with Discrete Actions

MCTS in practice: MuZero

- Does not assume zero-sum games (win/lose) but general RL structure (rewards)
- Policy is still MCTS, but uses **learned model** instead of game rules
- Learns three components:
  - **h**: function that transforms observations to latent states
  - **g**: model that predicts next latent state and reward, based on current latent state and action
  - **f**: function that predicts current policy (from MCTS), and Value of latent state
- All components are learned end-to-end using real game data stored in a replay buffer.





Schrittwieser, Julian, et al. "Mastering atari, go, chess and shogi by planning with a learned model." Nature 588.7839 (2020): 604-609.