Aalto University
School of Electrical
Engineering

ELEC-E8125 Reinforcement Learning
Model-based RL

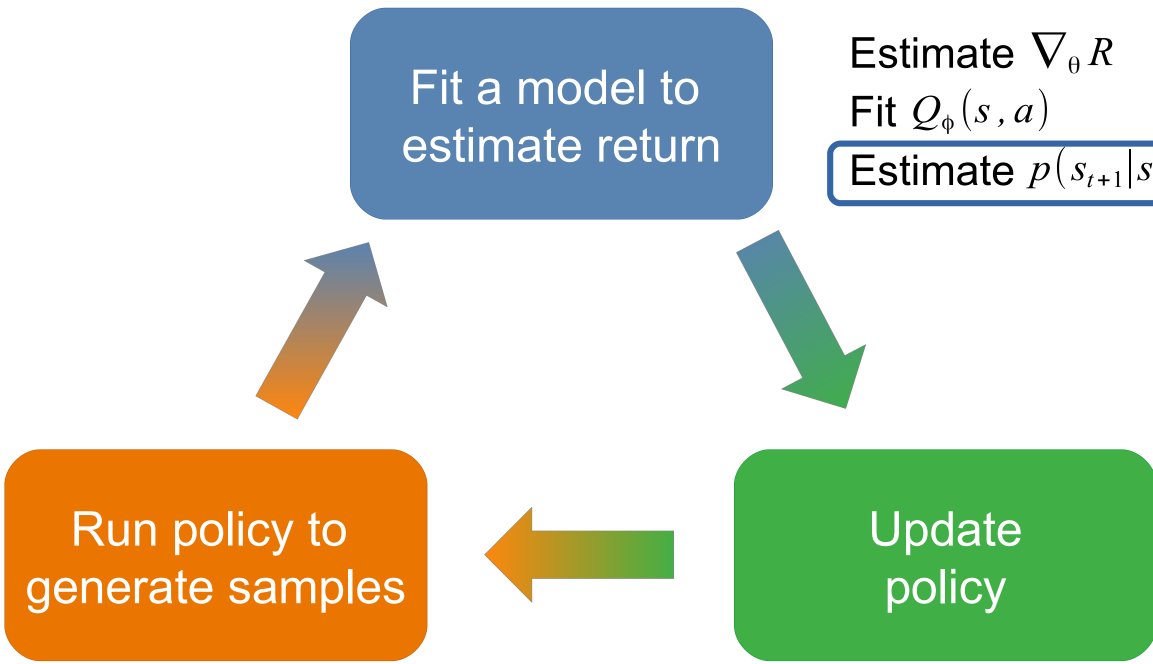
Ville Kyrki

27.10.2020

Learning goals

- Understand basic approaches for model-based reinforcement learning.

Anatomy of reinforcement learning
Model-based



Estimate $\nabla_\theta R$	(policy gradient)
Fit $Q_\phi(s,a)$	(Q-learning, actor-critic)
$\boxed{\text{Estimate } p(s_{t+1}\mid s_t, a_t)}$	(model-based)

$\theta \leftarrow \theta + \alpha \nabla_\theta R(\theta)$	(policy gradient)
$arg\,max_u Q_\phi(s,a)$	(Q-learning)
$\boxed{\text{Optimize } \pi_\theta(a\mid s)}$	(model-based)

Adopted from Sergey Levin.

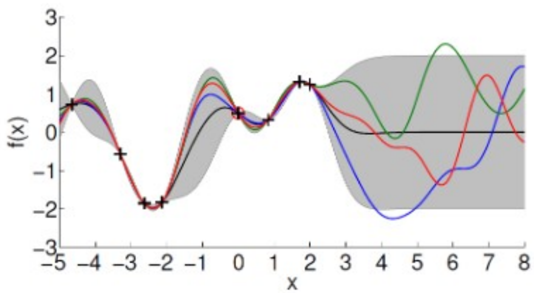
Motivation (partial recap)

- Reinforcement learning has limited sample efficiency.
- Learned policies are task(reward-function)-specific, learned policies cannot be directly reused.
- Learned dynamics model is reusable and can be used to reason about potential futures.
- Sometimes we know the model, e.g. in games!

Model definition and types

- Dynamics model $\boldsymbol{s}_{t+1} = f(\boldsymbol{s}_t, \boldsymbol{a}_t)$ or $f(\boldsymbol{s}_{t+1} | \boldsymbol{s}_t, \boldsymbol{a}_t)$
- Reward model $r_{t+1} = r(\boldsymbol{s}_t, \boldsymbol{a}_t)$ or $r(r_{t+1} | \boldsymbol{s}_t, \boldsymbol{a}_t)$
- Models are usually learned.
  – Parametric regression (e.g. neural net) common.
- May be also known (e.g. games, simulators)
  – Even physics based models need to be often calibrated.
- Also other possibilities (active research area)
  – Latent variable models, graph neural networks, non-parametric regression models such as Gaussian processes, ...

Which model to use?



Gaussian process (GP)

- Data-efficient
- Slow with big datasets
- May be too smooth for non-smooth dynamics

Neural networks

- Expressive
- Unpredictable with sparse data (overfit)

Linear models

- May be used locally
- Do not overfit

Domain specific parametric models (e.g. physics parameters) can also be used.
→ Traditional control engineering approach of model identification + control.

Spectrum of model-based RL

how to act in
current situation
(choose action)

Time of planning

learn to act in
any situation
(learn policy)

On-line
- Act on current state
- Act without learning
- Better in unfamiliar situations

Off-line
- Fast online computation
- Predictable within familiar situations

Spectrum of model-based RL

Time of planning

On-line

Discrete actions
- MCTS
- ...

Continuous actions
- iLQR
- ...

Off-line

Simulate environment
- DYNA
- ...

Assist learning
- Policy backprop
- ...

Spectrum of model-based RL

Time of planning

On-line

Discrete actions
- MCTS
- ...

Continuous actions
- iLQR
- ...

Off-line

Simulate environment
- DYNA
- ...

Assist learning
- Policy backprop
- ...

We kind of saw this already last week.

Continuous on-line planning: iLQR + learned model

Input: base policy π_0

Run base policy to collect data $D \leftarrow \{(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')_i\}$

Repeat

Fit dynamics model $f(\boldsymbol{s}, \boldsymbol{a})$ to minimize $\sum_i \|f(\boldsymbol{s}_i, \boldsymbol{a}_i) - \boldsymbol{s}_i'\|^2$

Use model to plan (e.g. iLQR) actions

Execute first planned action, observe resulting state $\boldsymbol{s}'$

Update dataset $D \leftarrow D \cup \{(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')\}$

Continuous on-line planning: iLQR + learned model

Input: base policy π_0

Run base policy to collect data $D \leftarrow \{(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')_i\}$

Repeat

Fit dynamics model $f(\boldsymbol{s}, \boldsymbol{a})$ to minimize $\sum_i \|f(\boldsymbol{s}_i, \boldsymbol{a}_i) - \boldsymbol{s}_i'\|^2$

Use model to plan (e.g. iLQR) actions

Execute first planned action, observe resulting state $\boldsymbol{s}'$

Update dataset $D \leftarrow D \cup \{(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')\}$

- Sample efficient.
- Computationally expensive for two reasons.
  - Dynamics fitting costly → model may be fitted only periodically (every n steps).
  - Planning costly for long horizons.
- Robust to moderate model errors.
- Choice of regression model is an important design parameter.

This is model-predictive control (MPC) with learned dynamics.
MPC horizon length is limited, can we do something?

Continuous on-line planning: iLQR + learned model

Input: base policy π_0

Run base policy to collect data $D \leftarrow \{(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')_i\}$

Repeat

Fit dynamics model $f(\boldsymbol{s}, \boldsymbol{a})$ to minimize $\sum_i \|f(\boldsymbol{s}_i, \boldsymbol{a}_i) - \boldsymbol{s}_i'\|^2$

Use model to plan (e.g. iLQR) actions

Execute first planned action, observe resulting state $\boldsymbol{s}'$

Update dataset $D \leftarrow D \cup \{(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')\}$

- Sample efficient.
- Computationally expensive for two reasons.
  - Dynamics fitting costly → model may be fitted only periodically (every n steps).
  - **Planning costly for long horizons.**
- Robust to moderate model errors.
- Choice of regression model is an important design parameter.

This is model-predictive control (MPC) with learned dynamics. MPC horizon length is limited, can we do something?

Spectrum of model-based RL

Time of planning

On-line

Discrete actions
- MCTS
- ...

Continuous actions
- iLQR
- ...

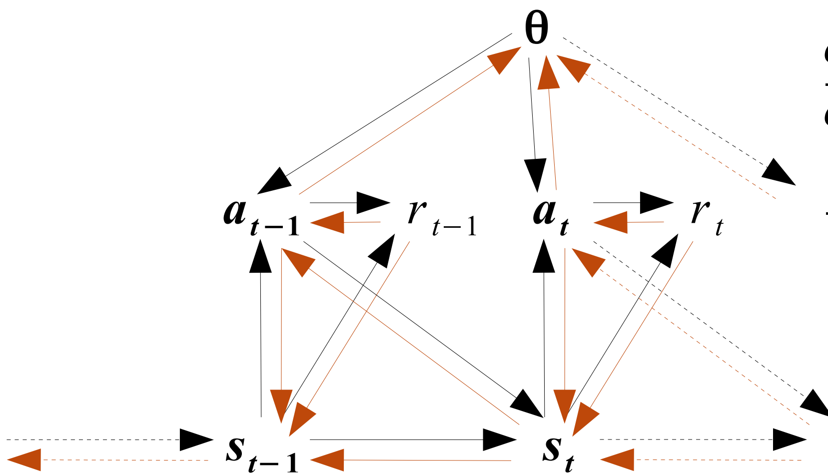
Off-line

Simulate environment
- DYNA
- ...

Assist learning
- Policy backprop
- ...

Backprop ~ chain rule of partial derivatives

Combining parametric policy with learned dynamics by backpropagation



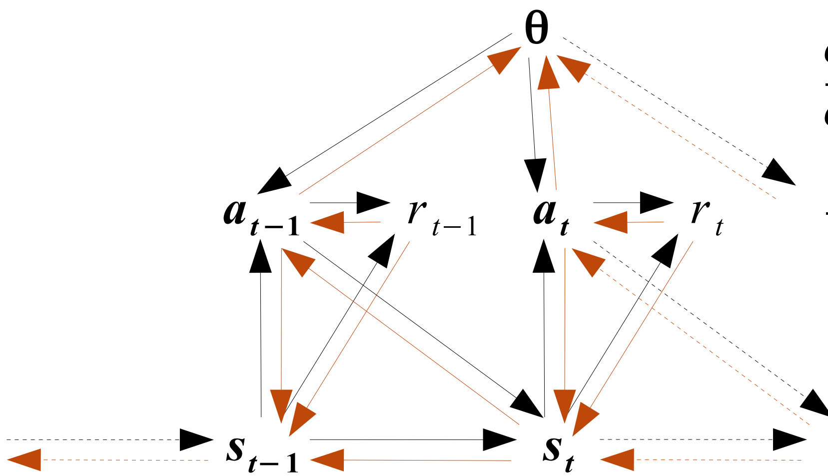
$$\frac{\partial r_t}{\partial \theta} = \frac{\partial r_t}{\partial \boldsymbol{a}_t}\frac{\partial \boldsymbol{a}_t}{\partial \theta} + \frac{\partial r_t}{\partial \boldsymbol{s}_t}\frac{\partial \boldsymbol{s}_t}{\partial \theta}$$

$$\frac{\partial \boldsymbol{s}_t}{\partial \theta} = \frac{\partial \boldsymbol{s}_t}{\partial \boldsymbol{s}_{t-1}}\frac{\partial \boldsymbol{s}_{t-1}}{\partial \theta} + \frac{\partial \boldsymbol{s}_t}{\partial \boldsymbol{a}_{t-1}}\frac{\partial \boldsymbol{a}_{t-1}}{\partial \theta}$$

policy	reward	dynamics
$\nabla_\theta \pi(s_t, a_t)$	$\nabla_a r(s_t, a_t)$	$\nabla_s f(s_{t-1}, a_{t-1})$
	$\nabla_s r(s_t, a_t)$	$\nabla_a f(s_{t-1}, a_{t-1})$

Backprop ~ chain rule of partial derivatives

Combining parametric policy with learned dynamics by backpropagation



$$\frac{\partial r_t}{\partial \theta} = \frac{\partial r_t}{\partial \boldsymbol{a}_t}\frac{\partial \boldsymbol{a}_t}{\partial \theta} + \frac{\partial r_t}{\partial \boldsymbol{s}_t}\frac{\partial \boldsymbol{s}_t}{\partial \theta}$$

$$\frac{\partial \boldsymbol{s}_t}{\partial \theta} = \frac{\partial \boldsymbol{s}_t}{\partial \boldsymbol{s}_{t-1}}\frac{\partial \boldsymbol{s}_{t-1}}{\partial \theta} + \frac{\partial \boldsymbol{s}_t}{\partial \boldsymbol{a}_{t-1}}\frac{\partial \boldsymbol{a}_{t-1}}{\partial \theta}$$

Run base policy to collect data $D \leftarrow \{(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')_i\}$

Repeat

- Fit dynamics model $f_\phi(\boldsymbol{s}, \boldsymbol{a})$ to minimize $\sum_i \|f_\phi(\boldsymbol{s}_i, \boldsymbol{a}_i) - \boldsymbol{s}_i'\|^2$
- Calculate policy gradient update by backpropagating through dynamics
- Execute updated policy (1 or more steps), collect data
- Update dataset $D \leftarrow D \cup \{(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')\}$

Tools handle this automatically by automatic differentiation.

Continuous on-line planning: iLQR + learned model

Input: base policy π_0

Run base policy to collect data $D \leftarrow \{(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')_i\}$

Repeat

Fit dynamics model $f(\boldsymbol{s}, \boldsymbol{a})$ to minimize $\sum_i \|f(\boldsymbol{s}_i, \boldsymbol{a}_i) - \boldsymbol{s}_i'\|^2$

Use model to plan (e.g. iLQR) actions

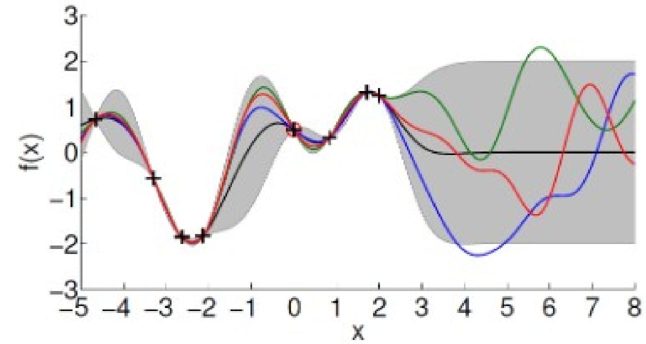
Execute first planned action, observe resulting state $\boldsymbol{s}'$

Update dataset $D \leftarrow D \cup \{(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')\}$

- Sample efficient.
- Computationally expensive for two reasons.
  - Dynamics fitting costly → model may be fitted only periodically (every n steps).
  - Planning costly for long horizons.
- Robust to moderate model errors.
- **Choice of regression model is an important design parameter.**

Example
PILCO (Deisenroth&Rasmussen, 2011)

- Dynamics learning: Use Gaussian process models to include model uncertainty. Known quadratic reward.
- Simulation: Simulate trajectory with learned model, including uncertainty.
- Policy: Radial basis function.
- Policy update: Calculate analytically policy gradient using learned dynamics and optimize with quasi-Newton optimizer (BFGS).
- GP → Very sample efficient. Cannot handle large dataset.



Aalto University
School of Electrical
Engineering

Reward function can also be learned using GP, e.g. BlackDROPS (2017).

- Idea: Learn also regression function for rewards.
- BlackDROPS (2017) uses a Gaussian process to model reward function as well as dynamics.
- Uses CMA-ES (gradient free optimizer) for planning.

Spectrum of model-based RL

Time of planning

On-line

Discrete actions

- MCTS
- ...

Continuous actions

- iLQR
- ...

Off-line

Simulate environment

- DYNA
- ...

Assist learning

- Policy backprop
- ...

Simulate environment to generate additional data: DYNA

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) $\boxed{Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]}$

(e) $\boxed{Model(S,A) \leftarrow R, S' \text{ (assuming deterministic environment)}}$

(f) Loop repeat n times:

$S \leftarrow$ random previously observed state

$A \leftarrow$ random action previously taken in S

$\boxed{R, S' \leftarrow Model(S,A)}$

$\boxed{Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]}$

Update using experience

Learn dynamics model

Generate data by simulating dynamics

Update using simulated experience

Spectrum of model-based RL

Time of planning

- **On-line**
  - **Discrete actions**
    - MCTS
    - ...
  - **Continuous actions**
    - iLQR
    - ...
- **Off-line**
  - **Simulate environment**
    - DYNA
    - ...
  - **Assist learning**
    - Policy backprop
    - ...

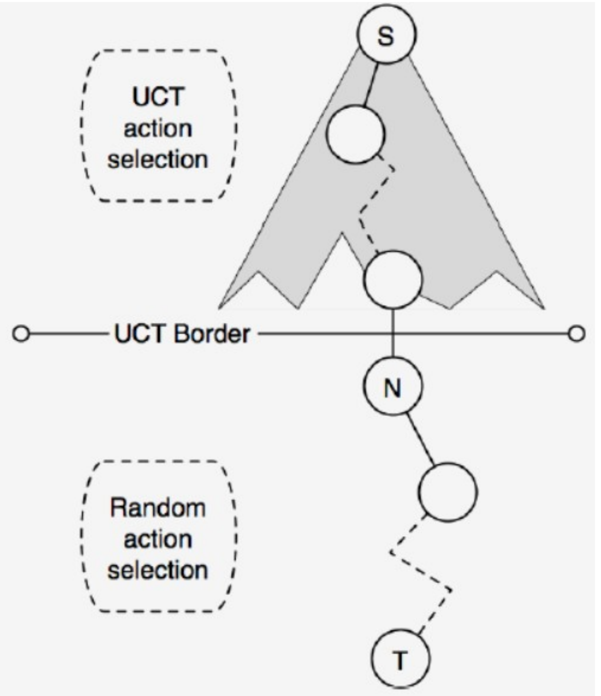
Monte Carlo tree search

- Search method for optimal decision making.
- State-of-the-art for playing games (e.g. Alpha Go).

- Iteratively builds a search tree.
- Phases:
  - Selection: Choose a promising node to expand.
  - Expansion: Add a new node.
  - Simulation: Simulate value for new node.
  - Backup: Back-up value to root (update values for parents).

MCTS operation

- From start node *S* choose actions to walk down tree until reaching a leaf node.
- Choose an action and create a child node *N* for that action.
- Perform a **random** roll-out (take random actions) until end of episode (or for a fixed horizon).
- Record returns as value for *N* and back up value to root.



Node selection in MCTS

- Node selection in search has to balance exploration and exploitation (note difference to RL, here x&x is made only using simulation).
- Idea: Explore when uncertain of outcome.
- Upper confidence bound 1 (UCB1) on trees (UCT).
  - A bound for value of a node (Kocsis&Szepesvari, 2006).

$$Q^{+}(s,a)=Q(s,a)+c\sqrt{\frac{\log N(s)}{N(s,a)}}$$

Positive exploration constant

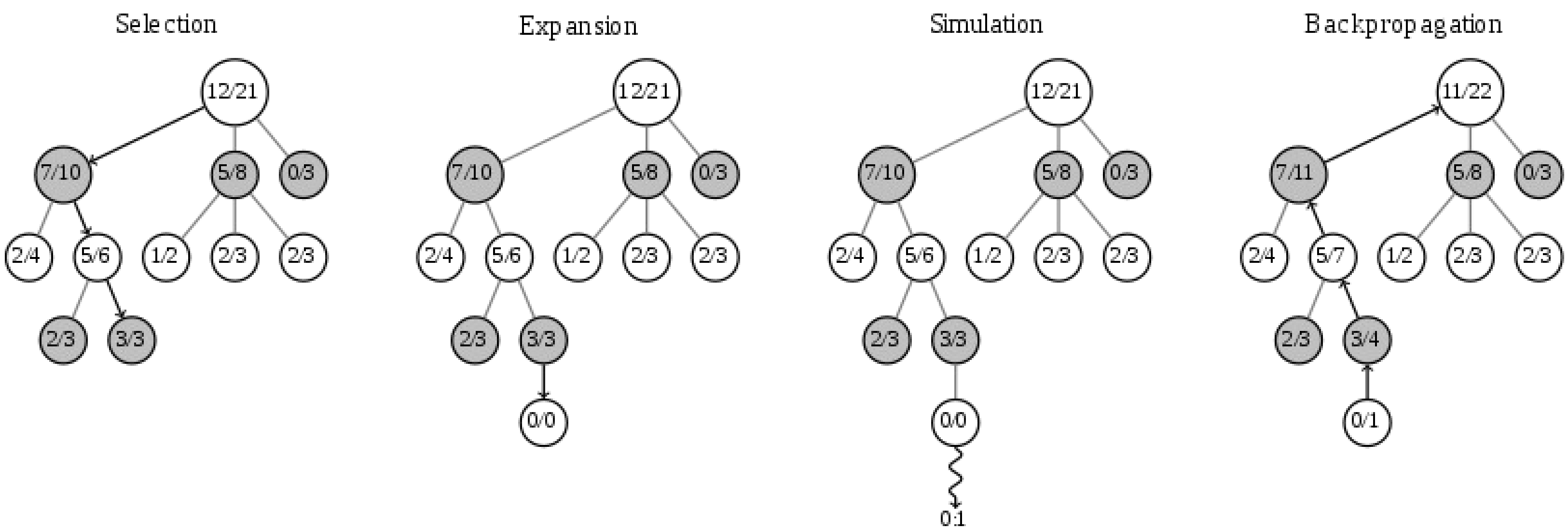
Visitation count

MCTS simulation phase

- Perform one or several roll-outs from leaf node using random action selection.
- Stop at terminal state or until a discount horizon is reached.
- Estimate value of state as mean return of the *N* simulations: $V(s) = \frac{1}{N} \sum_i G_i$

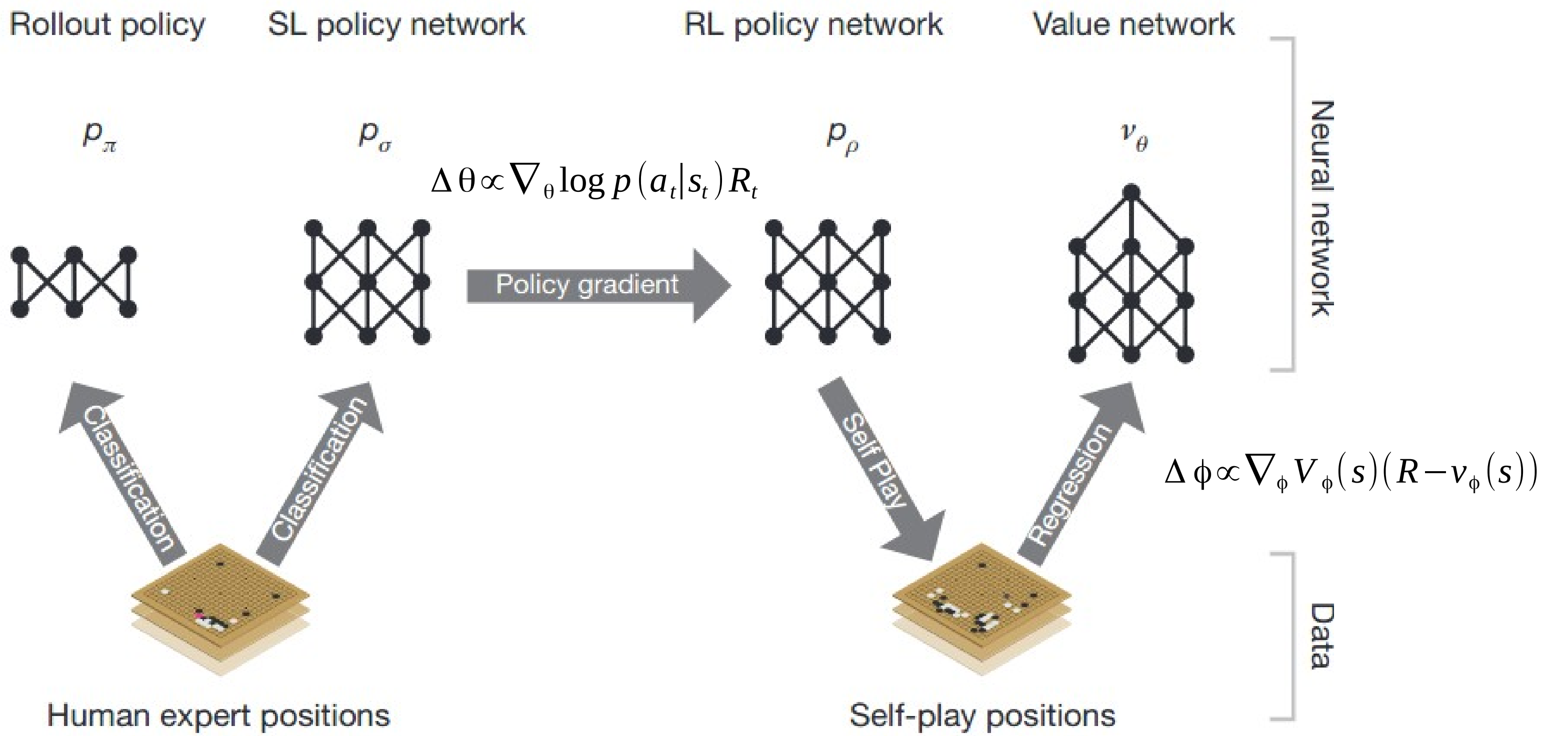
MCTS: Example in game playing

- Value number of won games.



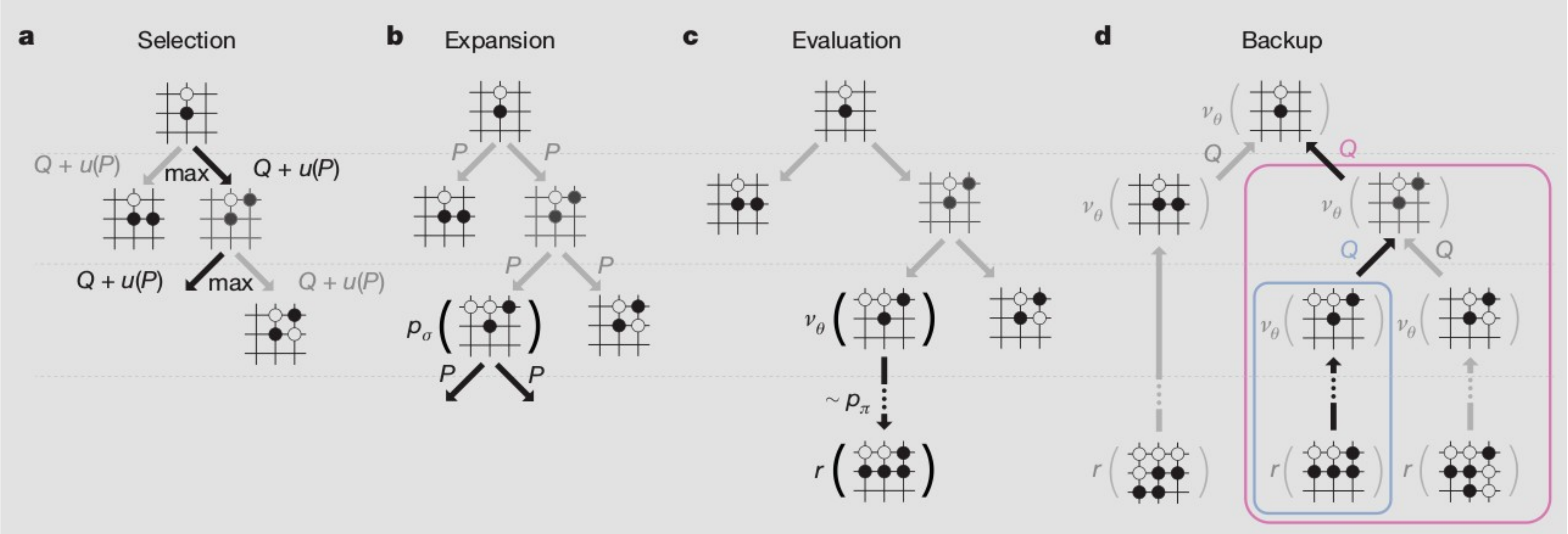
Example: Alpha Go (2016)

- Policy learned initially to imitate human players.
- Updated through policy gradient and self-play.



Example: Alpha Go (2016)

- Action chosen by MCTS.
- Action evaluation uses estimated value and a roll-out.



Spectrum of model-based RL

Time of planning

- **On-line**
  - **Discrete actions**
    - MCTS
    - ...
  - **Continuous actions**
    - iLQR
    - ...
- **Off-line**
  - **Simulate environment**
    - DYNA
    - ...
  - **Assist learning**
    - Policy backprop
    - ...

The ideas can also be combined!

Summary

- Model-based RL requires typically less data than value-based or policy gradient approaches.
- Learned dynamics can be transferred across tasks.
- Potentially suboptimal: models do not optimize for task performance and policy optimization may be prone to local minima.
- Sometimes models are harder to learn than policy.
- Often require explicit choices (e.g. time horizon).

Next: Partial observability and POMDPs

- Next week: Guest lecture!

  Afterwards:
- What changes if we cannot observe state directly?

- Reading: Tony Cassandra's on-line tutorial (see MyCourses for details)