

NPFL099 Statistical Dialogue Systems

7. Dialogue Management (2)

Action Selection/Policy

Ondřej Dušek

<http://ufal.cz/npfl099>

10. 11. 2025



Charles University
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

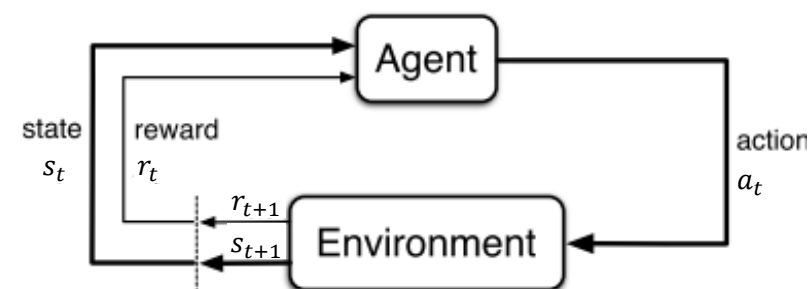
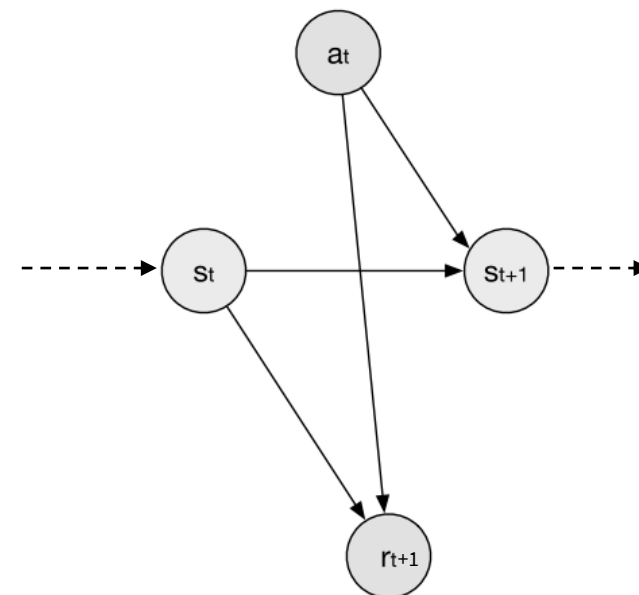


unless otherwise stated

Action selection: Recap

- Action selection: deciding what to do (or say) next
 - based on dialogue state (i.e. uses tracking output)
 - follows a **policy** towards an end goal
- FSM, frames, rule-based
- **trained policies**: typically with RL
 - explore more different paths than supervised
 - plan ahead – optimize for the whole dialogue, not just 1 turn
- RL: MDP formalism – agent in an environment, **state-action-reward**
 - POMDP = MDP with continuous states
 - trained with user simulator

(from Milica Gašić's slides)



(Sutton & Barto, 2018)

Reinforcement learning: Definition

- RL = finding a **policy that maximizes long-term reward**
 - unlike supervised learning, we don't know if an action is good
 - immediate reward might be low while long-term reward high

accumulated long-term reward
(from turn t onwards)

$$R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i+1}$$

alternative – **episodes**: only count to T when we encounter a terminal state
(e.g. 1 episode = 1 dialogue)

$\gamma \in [0,1]$ = **discount factor**
(immediate vs. future reward trade-off)

$\gamma = 1$: no discount, only usable if $i \leq T$
 $\gamma < 1$: R_t is finite (if r_t is finite)
 $\gamma = 0$: greedy approach (ignore future rewards)

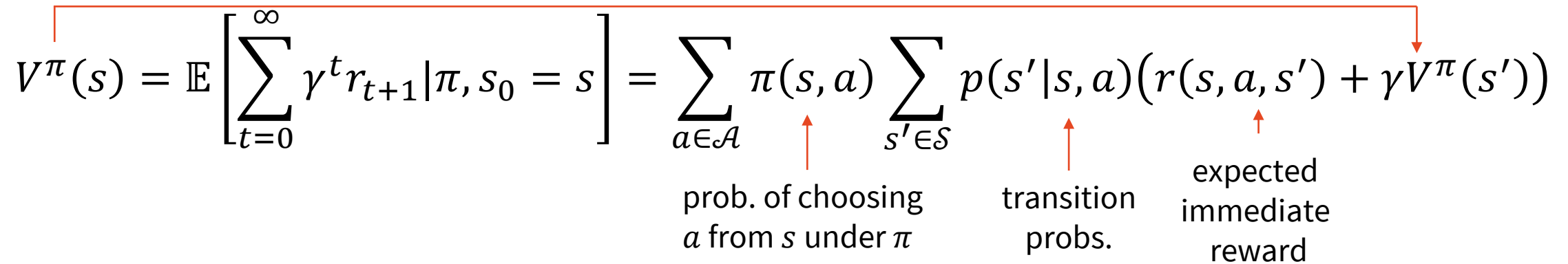
- state transition is stochastic \rightarrow maximize **expected return**

$$\mathbb{E}[R_t | \pi, s_0] \quad \leftarrow \text{expected } R_t \text{ if we start from state } s_0 \text{ and follow policy } \pi$$

State-value Function

- Using return, we define the **value of a state** s under policy π : $V^\pi(s)$
 - Expected return for starting in state s and following policy π
- Return is recursive: $R_t = r_{t+1} + \gamma \cdot R_{t+1}$
- This gives us a recursive equation (**Bellman Equation**):

$$V^\pi(s) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid \pi, s_0 = s \right] = \sum_{a \in \mathcal{A}} \pi(s, a) \sum_{s' \in \mathcal{S}} p(s' | s, a) (r(s, a, s') + \gamma V^\pi(s'))$$



prob. of choosing a from s under π transition probs. expected immediate reward


- $V^\pi(s)$ defines a **greedy policy**:

$$\pi(s, a) := \begin{cases} \frac{1}{\# \text{ of } a's} & \text{for } a = \arg \max_a \sum_{s' \in \mathcal{S}} p(s' | s, a) (r(s, a, s') + \gamma V^\pi(s')) \\ 0 & \text{otherwise} \end{cases}$$

actions that look best for the next step

Action-value (Q-)Function

- $Q^\pi(s, a)$ – return of taking action a in state s , under policy π
 - Same principle as value $V^\pi(s)$, just **considers the current action, too**
 - Has its own version of the Bellman equation

$$Q^\pi(s, a) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid \pi, s_0 = s, a_0 = a \right] = \sum_{s' \in \mathcal{S}} p(s' | s, a) \left(r(s, a, s') + \gamma \sum_{a' \in \mathcal{A}} Q^\pi(s', a') \pi(s', a') \right)$$


- $Q^\pi(s, a)$ also defines a greedy policy:

$$\pi(s, a) := \begin{cases} \frac{1}{\# \text{ of } a' \text{'s}} & \text{for } a = \arg \max_a Q^\pi(s, a) \\ 0 & \text{otherwise} \end{cases}$$

again, “actions that look best for the next step”

simpler: no need to enumerate s' ,
no need to know $p(s' | s, a)$ and $r(s, a, s')$

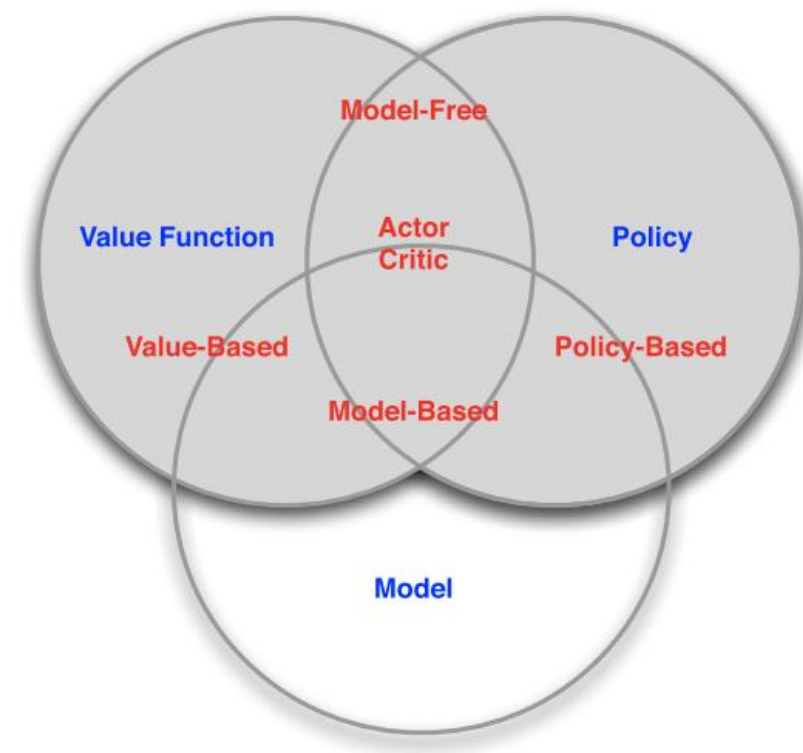
but Q function itself tends to be more complex than V

Optimal Policy in terms of V and Q

- **optimal policy** π^* – one that maximizes expected return $\mathbb{E}[R_t|\pi]$
 - $V^\pi(s)$ expresses $\mathbb{E}[R_t|\pi] \rightarrow$ use it to define π^*
- π^* is a policy such that $V^{\pi^*}(s) \geq V^{\pi'}(s) \quad \forall \pi', \forall s \in \mathcal{S}$
 - π^* always exists in an MDP (need not be unique)
 - π^* has the **optimal state-value function** $V^*(s) := \max_{\pi} V^{\pi}(s)$
 - π^* also has the **optimal action-value function** $Q^*(s, a) := \max_{\pi} Q^{\pi}(s, a)$
- greedy policies with $V^*(s)$ and $Q^*(s, a)$ are optimal
 - we can search for either π^* , $V^*(s)$ or $Q^*(s, a)$ and get the same result
 - each has their advantages and disadvantages

RL Agents Taxonomy

- Quantity to optimize:
 - value function – **critic**
 - either Q or V , typically Q in practice
 - policy – **actor**
 - both – **actor-critic**
- Environment model:
 - **model-based** (assume known $p(s'|s, a), r(s, a, s)$)
 - nice but typically not satisfied in practice
 - **model-free** (don't assume anything, sample)
 - this is the usual real-world case
 - this is where using Q instead of V comes handy



(from David Silver's slides)

Reinforcement Learning Approaches

- How to optimize:
 - **dynamic programming** – find the exact solution from Bellman equation
 - iterative algorithms, refining estimates
 - expensive, assumes known environment → not practical for real-world use
 - **Monte Carlo learning** – learn from experience
 - sample, then update based on experience
 - **Temporal difference learning** – like MC but look ahead (bootstrap)
 - sample, refine estimates as you go
- Sampling & updates:
 - **on-policy** – improve the policy while you're using it for decisions
 - can't use that with batch learning (decision policy is changing constantly)
 - **off-policy** – decide acc. to a different policy

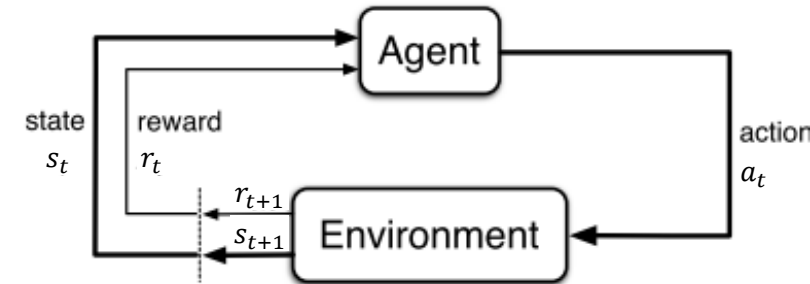
} both used
in practice



https://twitter.com/srush_nlp/status/1729981547644698816

Deep Reinforcement Learning

- Exactly the same as “plain” RL
 - agent & environment, actions & rewards
- **“deep” = part of the agent is handled by a NN**
 - value function (typically Q)
 - policy
- function approximation approach
 - Q values / policy are represented as a parameterized function $Q(s, a; \theta) / \pi(s; \theta)$
 - enumerating in a table would take up too much space, be too sparse
 - the parameters θ are optimized
- assuming huge state space
 - much fewer weights than possible states
 - update based on one state changes many states
- needs tricks to make it stable



(Sutton & Barto, 2018)

- temporal difference – update Q as you go

- off-policy – directly estimates best Q^*

- regardless of policy used for sampling

- choose learning rate α , initialize Q arbitrarily

- for each episode:

- choose initial s

- for each step:

- choose a from s according to **ϵ -greedy policy** based on Q

- take action a , observe observe reward r and state s'

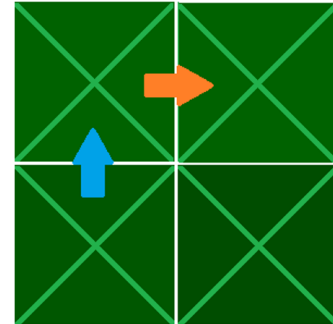
- $Q(s, a) \leftarrow (1 - \alpha) \cdot Q(s, a) + \alpha \left(r + \gamma \cdot \max_{a'} Q(s', a') \right)$

- $s \leftarrow s'$

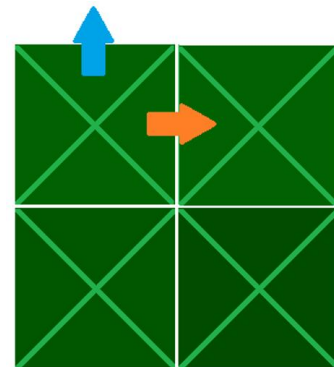
TD: moving estimates

update uses best a' , regardless of current policy:
 a' is not necessarily taken in the actual episode

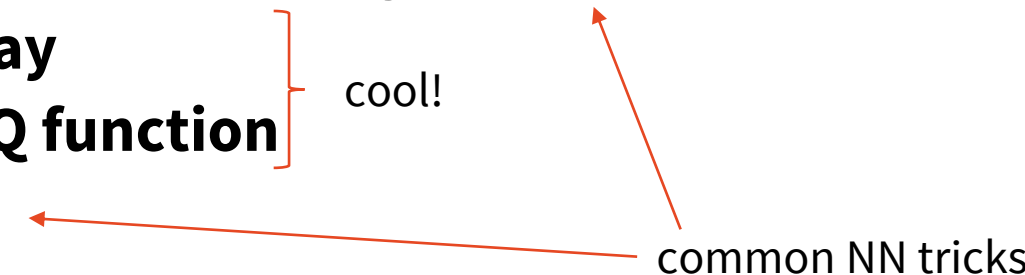
any policy that chooses all actions & states enough times will converge to $Q^*(s, a)$: we need to explore to converge



State: S
Action taken: North
Action with max Q value at S' : East



State: S'
Action taken: North (any action)

- Q-learning, where Q function is represented by a neural net
 - “Usual” Q-learning doesn’t converge well with NNs:
 - a) SGD is unstable
 - b) correlated samples (data is sequential)
 - c) TD updates aim at a moving target (using Q in computing updates to Q)
 - d) scale of rewards & Q values unknown → numeric instability
 - → DQN adds fixes:
 - a) minibatches (updates by averaged n samples, not just one)
 - b) experience replay**
 - c) freezing target Q function**
 - d) clipping rewards
- 
- Diagram illustrating common NN tricks:
- cool! (points to experience replay and freezing target Q function)
 - common NN tricks (points to experience replay, freezing target Q function, and clipping rewards)

DQN tricks ~ making it more like supervised learning

• Experience replay – break correlated samples

- run through some episodes (dialogues, games...) ← *“generate your own ‘supervised’ training data”*
- store all tuples (s, a, r', s') in a buffer
- for training, don't update based on most recent moves – use buffer
 - sample minibatches randomly from the buffer
- overwrite buffer as you go, clear buffer once in a while
- only possible for off-policy

$$\text{loss} := \mathbb{E}_{(s,a,r',s') \in \text{buf}} \left[\left(r' + \gamma \max_{a'} Q(s', a'; \bar{\theta}) - Q(s, a; \theta) \right)^2 \right]$$

• Target Q function freezing

- fix the version of Q function used in update targets
 - have a copy of your Q network that doesn't get updated every time
- once in a while, copy your current estimate over ← *“have a fixed target, like in supervised learning”*

DQN algorithm

- initialize θ randomly
- initialize replay memory D (e.g. play for a while using current $Q(\theta)$)
- repeat over all episodes:
 - set initial state s
 - for all timesteps $t = 1 \dots T$ in the episode:
 - select action a_t from ϵ -greedy policy based on $Q(\theta)$
 - take a_t , observe reward r_{t+1} and new state s_{t+1}
 - store $(s_t, a_t, r_{t+1}, s_{t+1})$ in D
 - sample a batch B of random (s, a, r', s') 's from D
 - update θ using loss $\mathbb{E}_{(s,a,r',s') \in B} \left[\left(r' + \gamma \max_{a'} Q(s', a'; \bar{\theta}) - Q(s, a; \theta) \right)^2 \right]$
- once every λ steps (rarely):
 - $\bar{\theta} \leftarrow \theta$

storing experience
(1 step of Q-learning exploration)

“replay”
a. k. a. training
(1 update)

update the frozen target function

DQN for Dialogue Systems

(Li et al., 2017)

<https://arxiv.org/abs/1703.01008>

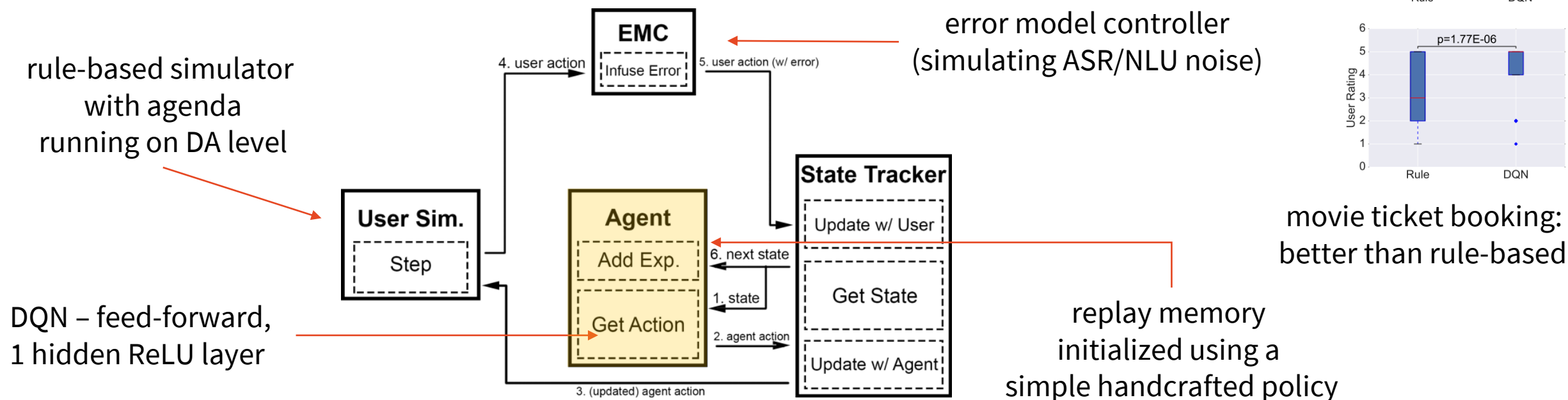
<https://github.com/MiuLab/TC-Bot>

Q | TD | model-free | off-policy

(Lipton et al., 2018)

<https://arxiv.org/abs/1608.05081>

- DQN can drive dialogue action selection/policy
- **warm start** needed to make the training actually work:
 - **pretrain** the network using supervised learning
 - **replay buffer spiking** – initialize using simple rule-based policy
 - so there are at least a few successful dialogues
 - the RL agent has something to catch on



Policy Gradients

- Instead of value functions, train a **network to represent the policy**
 - allows better action sampling – according to actual stochastic policy
 - no need for ϵ -greedy (which is partially random, suboptimal)
- To optimize, we need a **performance metric**: $J(\theta) = V^{\pi_\theta}(s_0)$
 - expected return in starting state when following π_θ
 - we want to directly optimize this using gradient ascent
- **Policy Gradient Theorem**:
 - expresses $\nabla J(\theta)$ in terms of $\nabla \pi(a|s, \theta)$

$$\nabla J(\theta) \propto \underbrace{\sum_s \mu(s)}_{\text{state probability}} \sum_a Q^\pi(s, a) \nabla \pi(a|s, \theta) = \mathbb{E}_\pi \left[\sum_a Q^\pi(s, a) \nabla \pi(a|s, \theta) \right]$$

$\mu(s)$ is state probability under π – this is the same as expected value \mathbb{E}_π

REINFORCE: Monte Carlo Policy Gradients

- direct search for policy parameters by stochastic gradient ascent

- looking to maximize performance $J(\boldsymbol{\theta}) = V^{\pi_{\boldsymbol{\theta}}}(s_0)$

- choose learning rate α , initialize $\boldsymbol{\theta}$ arbitrarily

- loop forever:

- generate an episode $s_0, a_0, r_1, \dots, s_{T-1}, a_{T-1}, r_T$, following $\pi(\cdot | \cdot, \boldsymbol{\theta})$

- for each $t = 0, 1 \dots T$: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t R_t \nabla \ln \pi(a_t | s_t, \boldsymbol{\theta})$

this will guarantee
the right state
distribution/frequency $\mu(s)$

returns $R_t = \sum_{i=t}^{T-1} \gamma^{i-t} r_{i+1}$

variant – **advantage** instead of returns:
discounting a **baseline**

$b(s)$ (predicted by any model)
 $A_t = R_t - b(s_t)$ instead of R_t
gives better performance

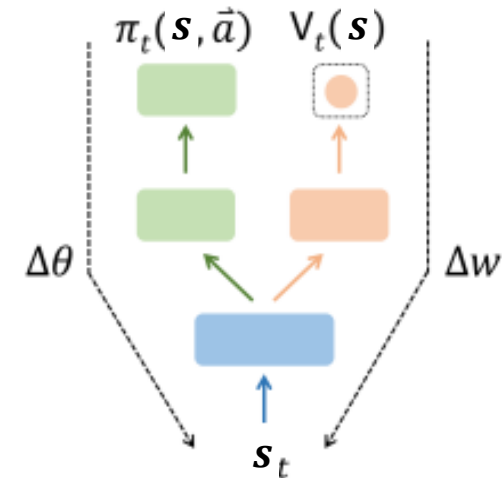
$V(s)$ is actually a good $b(s)$

this is stochastic $\nabla J(\boldsymbol{\theta})$:

- from policy gradient theorem
- using single action sample a_t
- expressing Q^{π} as R_t (under \mathbb{E}_{π})
- using $\nabla \ln x = \frac{\nabla x}{x}$

Policy Gradients (Advantage) Actor-Critic

- REINFORCE + V approximation + TD estimates – better convergence
 - differentiable policy $\pi(a|s, \theta)$
 - differentiable state-value function parameterization $\hat{V}(s, w)$
 - two learning rates α^θ, α^w
- loop forever:
 - set initial state s for the episode
 - for each step t of the episode:
 - sample action a from $\pi(\cdot | s, \theta)$, take a and observe reward r and new state s'
 - compute **advantage** $A \leftarrow r + \gamma \hat{V}(s', w) - \hat{V}(s, w)$
 - update $\theta \leftarrow \theta + \alpha^\theta \gamma^t A \nabla \ln \pi(a|s, \theta)$, $w \leftarrow w + \alpha^w \cdot A \nabla \hat{V}(s, w)$
 - $s \leftarrow s'$



actor (policy update)

critic (value function update)

same as REINFORCE, except:

- we use $\hat{V}(s, w)$ as baseline
- r is used instead of R_t (TD instead of MC)

TD: update
after each step,
moving estimates

ACER: Actor-Critic with Experience Replay

- off-policy actor-critic – using **experience replay** buffer
 - same approach as Q-learning
 - since ER buffer has past experience with out-of-date policies (using “old” $\tilde{\theta}$), it’s considered off-policy (behaviour policy $\pi_{\tilde{\theta}} \neq$ target policy π_{θ})
 - sampling behaviour from $\pi_{\tilde{\theta}}$ is biased w. r. t. π_{θ}
 - correcting the bias – **importance sampling**: multiply by importance weight $\rho_t = \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\tilde{\theta}}(a_t|s_t)}$
 - all updates are summed over batches & importance-sampled
 - new objective/performance metric: $\hat{\mathbb{E}}_t \left[\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\tilde{\theta}}(a_t|s_t)} \hat{A}_t \right]$
 - batch average over timesteps t
 - importance sampled
 - using advantage instead of returns

(Wang et al., 2017) <http://arxiv.org/abs/1611.01224>

(Su et al., 2017) <http://arxiv.org/abs/1707.00130>

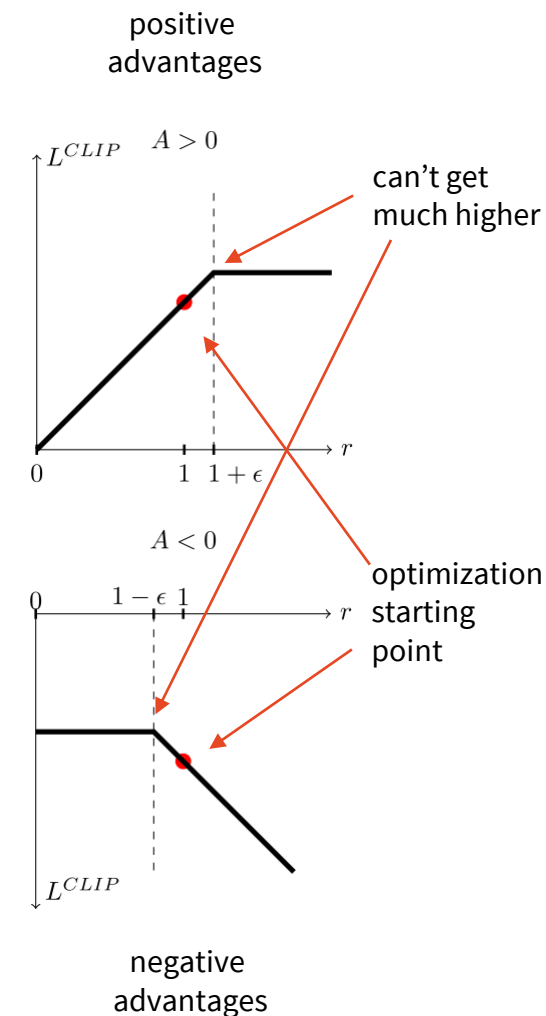
(Weisz et al., 2018) <http://arxiv.org/abs/1802.03753>

- ACER is prone to very large updates, unstable
 - to avoid going “off a cliff”, it needs very low LR, trains slowly
 - → change the objective to produce more stable updates

- Basically clipping the ACER objective

- define $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\tilde{\theta}}(a_t|s_t)}$ – ratio to old params
- starting from $\hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\tilde{\theta}}(a_t|s_t)} \hat{A}_t \right] = \hat{\mathbb{E}}_t [r_t(\theta) \hat{A}_t]$ (see ACER)
- using $\hat{\mathbb{E}}_t \left[\min \left(\underbrace{r_t(\theta) \hat{A}_t}_{\text{original}}, \underbrace{\text{clip}[r_t(\theta)]_{1-\epsilon}^{1+\epsilon} \hat{A}_t}_{\text{clipped to stay close to 1}} \right) \right]$

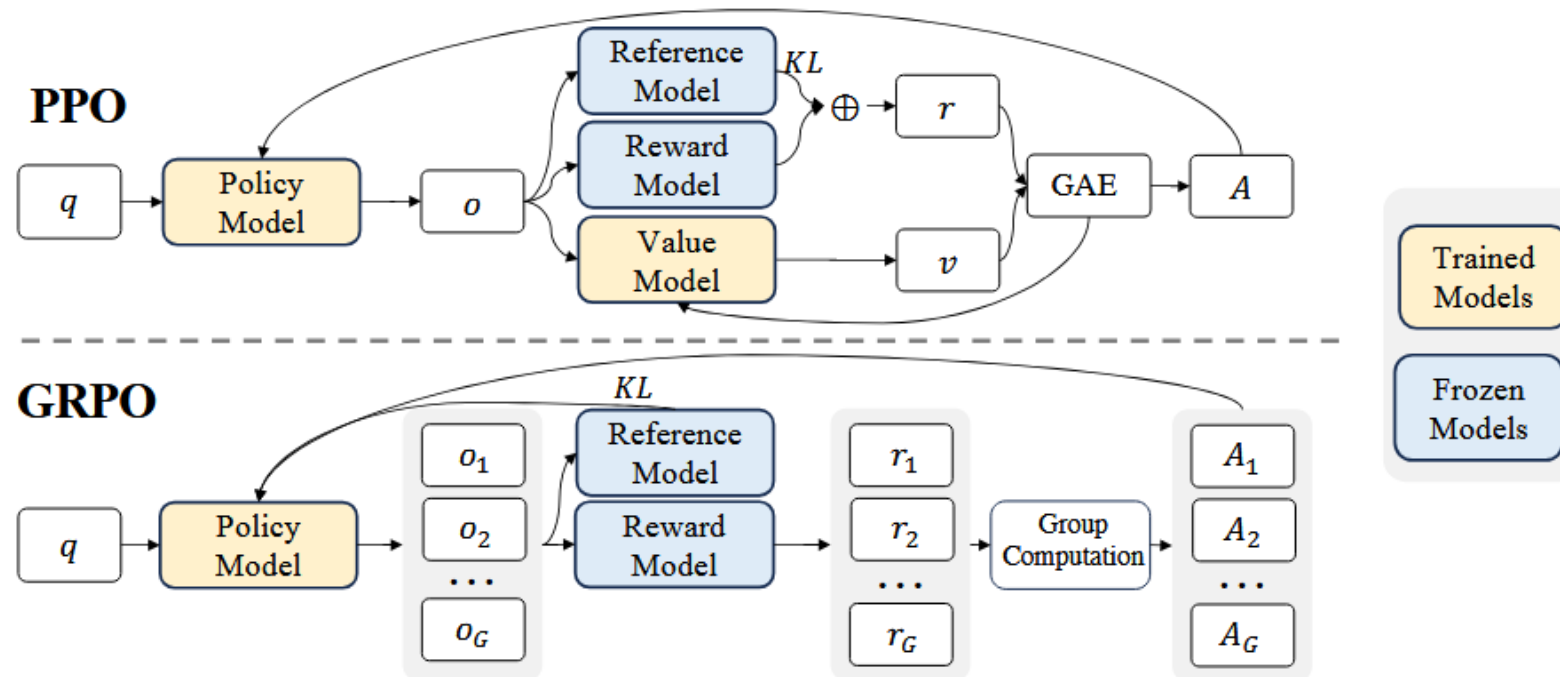
minimum – lower bound on the unclipped objective



Group Relative Policy Optimization

- Same as PPO, just drop the critic
 - we save memory by not having to model values
- Instead, compute multiple outputs per input & use their average as baseline
 - compute advantages based on that

(Shao et al., 2024)
<https://arxiv.org/pdf/2402.03300>



Rewards in RL

- Reward function is critical for successful learning
- Typical handcrafted setup in dialogues:
 - High positive for successful dialogue
 - high negative for unsuccessful dialogue
 - -1 per turn (promote efficiency)
- Handcrafting is not ideal
 - domain knowledge typically needed to detect dialogue success
 - need simulated or paid users,
can't learn from users without knowing their task
 - paid users often fail to follow pre-set goals
- Having users provide feedback is costly & inconsistent
 - real users don't have much incentive to be cooperative
- Learning/optimizing the rewards is desirable

Turn-level rewards

(Schmitt & Ultes, 2015; Ultes et al., 2017; Ultes, 2019; Ultes & Maier, 2021)

<https://doi.org/10.1016/j.specom.2015.06.003>

<https://doi.org/10.21437/Interspeech.2017-1032>

<https://aclweb.org/anthology/W19-5902/>

<https://aclanthology.org/2021.sigdial-1.42>

- **Interaction quality**

- hand-annotated turns for ~200 dialogues
- SVM/RNN on low-level domain-independent features (ASR confidence, # reprompts etc.)

- **Discriminator**

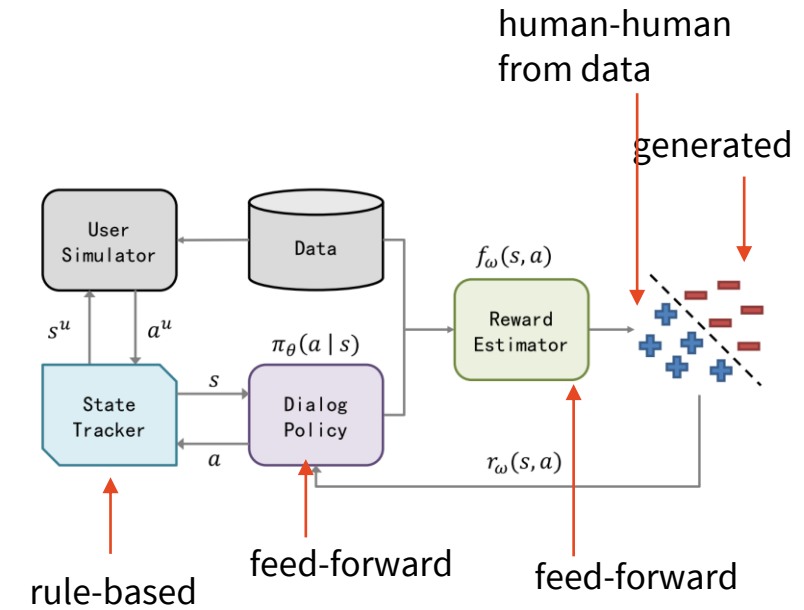
- policy vs. human-human (iterative, adversarial learning)
- reward for appearing human-like at each turn

- **Information gain**

- reward system asking \approx changes in belief state distributions (Jensen-Shannon divergence \geq threshold)
- combined with task success

(Geishauser et al., 2021) <http://arxiv.org/abs/2109.07129>

(Takanobu et al., 2019) <http://arxiv.org/abs/1908.10719>



Alternating supervised & RL

- we can do better than just supervised pretraining
- alternate regularly
 - start with supervised more frequently
 - alleviate sparse rewards, but don't completely avoid exploring
 - later do more RL
 - but don't forget what you learned by supervised learning
- options:
 - schedule supervised every N updates
 - same + increase N gradually
 - use supervised after RL does poorly (worse than baseline)
 - baseline = moving average over history + $\lambda \cdot$ std. error of the average
 - agent is less likely to be worse than baseline in later stages of learning



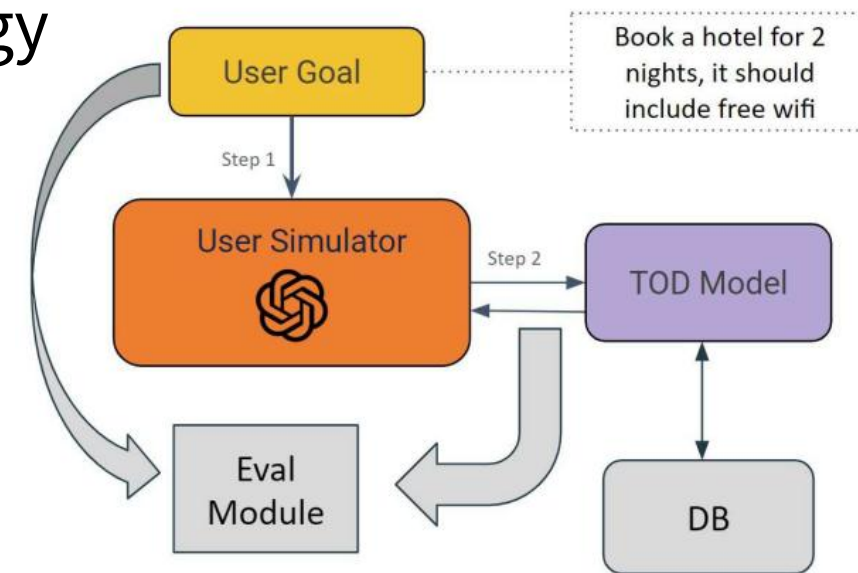
Mark Riedl
@mark_riedl

Everyone knows this, right? Right? Most RL agents are overfit and can be defeated by acting out-of-distribution. Everyone should know this.

https://twitter.com/mark_riedl/status/1682937331727192065

LLM-based simulators

- Closer to humans than traditional simulators, but cheaper
- Work best in text mode (for full dialogue system)
- Prompt off-the-shelf LLMs with tasks from ontology
 - direct prompting
 - chain of thought
 - explicit user state tracking
- Reward: can be computed by LLM too
 - feed LLM with whole dialogue
 - ask if goal was fulfilled
- Even closer to humans: finetune LLMs specifically
 - based human-LLM conversations, with generated intents
 - use non-instruction tuned LLMs for this
 - this makes the simulator more challenging to learn with

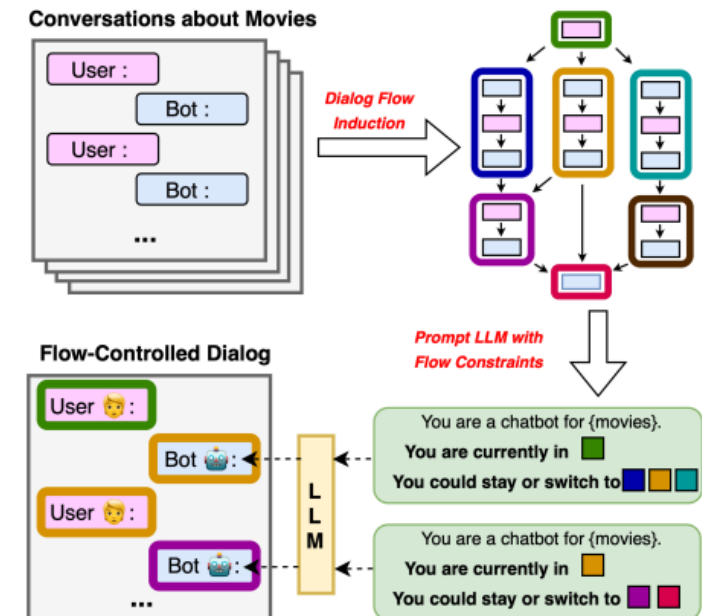
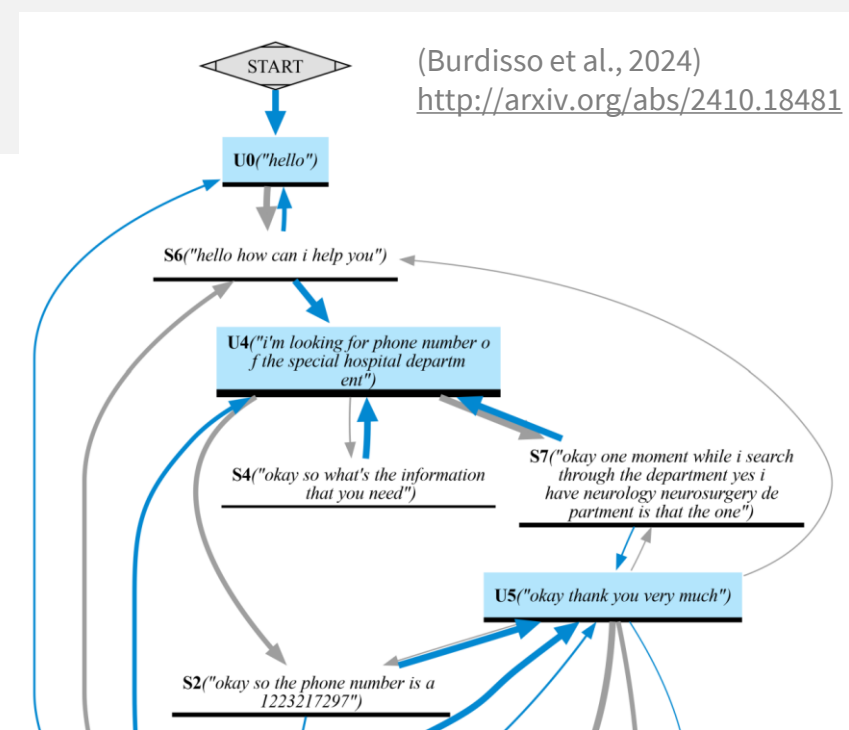


(Kazi et al., 2024) <http://arxiv.org/abs/2411.09972>

(Naous et al., 2025) <http://arxiv.org/abs/2510.06552>

Embeddings/LLM Dialogue Flow Induction

- No RL, creating **rule-based** flows automatically
- No need for annotation
 - good if you have e.g. call center recordings
- Analyze existing data, with dialogue embeddings
 - BERT finetuned on many dialogue datasets
 - cluster actions
 - create flow graph based on actions in data
- Prompt LLM to write dialogue flows
 - multi-step, with feedback & update
 - use real dialogues to augment the LLM-written flows
 - cluster actions & use dialogues with centroids as representatives



(Agrawal et al., 2024)
<https://aclanthology.org/2024.sigdial-1.6>

Summary

- **RL** for action selection / dialogue policy
 - MDP / agent in an environment, taking actions, getting rewards
 - dynamic programming, **Monte Carlo**, **Temporal Difference**
 - optimizing **value function** V/Q (**critic**), **policy** (**actor**), or both (**actor-critic**)
 - learning **on-policy** or **off-policy** (act by the policy you learn/not)
 - Combination with supervised makes sense
 - Rewards can be learned/estimated
- **DQN** – representing & optimizing Q function with a network
 - minibatches, target function freezing, experience replay
- **Policy gradients** – policy network & direct policy optimization
 - **REINFORCE** (MC policy gradients) + advantage
 - **Actor-critic** (REINFORCE + TD + V estimates) + extensions (ACER, PPO)
- LLMs can work as a simulator or to extract rule-based flows

Thanks

Contact us:

[https://ufaldsg.slack.com/
odusek@ufal.mff.cuni.cz](https://ufaldsg.slack.com/odusek@ufal.mff.cuni.cz)

Skype/Meet/Zoom/Troja (by agreement)

Next week: holiday
2 weeks from now: NLG
+ 4th assignment

Get these slides here:

<http://ufal.cz/npfl099>

References/Inspiration/Further:

- Sutton & Barto (2018): Reinforcement Learning: An Introduction (2nd ed.)
<http://incompleteideas.net/book/the-book.html>
- Nie et al. (2019): Neural approaches to conversational AI: <https://arxiv.org/abs/1809.08267>
- Filip Jurčiček's slides (Charles University): <https://ufal.mff.cuni.cz/~jurcicek/NPFL099-SDS-2014LS/>
- Milica Gašić's slides (Cambridge University): <http://mi.eng.cam.ac.uk/~mg436/teaching.html>
- Heidrich-Meisner et al. (2007): Reinforcement Learning in a Nutshell: <https://christian-igel.github.io/paper/RLiaN.pdf>
- Young et al. (2013): POMDP-Based Statistical Spoken Dialog Systems: A Review:
<http://cs.brown.edu/courses/csci2951-k/papers/young13.pdf>