# Reinforcement Learning for Critical Domains

Daan Bloembergen

joint work with

Richard Klima (UoL), Michael Kaisers (CWI),
Karl Tuyls (DeepMind)

# Motivation

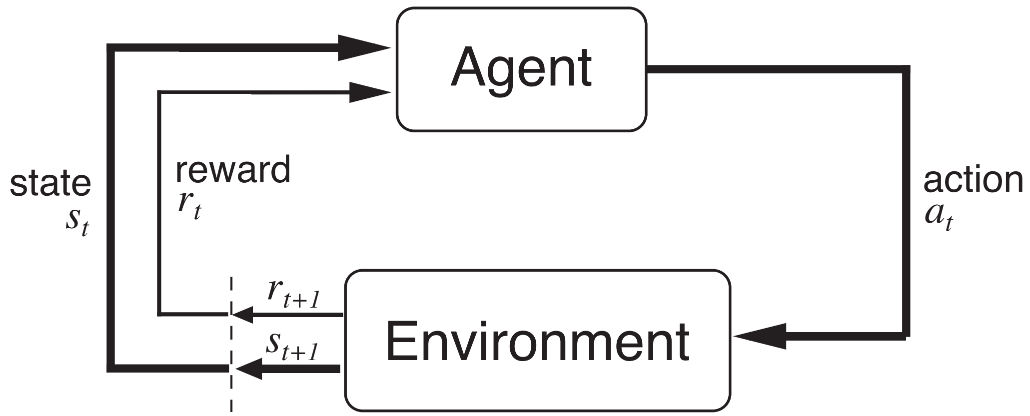
- Many large scale **critical systems** are highly sensitive to the local performance of individual components.
  - traffic and transport networks, security systems, power grids, …

- Local failure or attack could destabilise the whole system.

- Goal: explicitly encode robustness against significant rare events in the learning method.

# What is Reinforcement Learning?

- Goal-oriented: learning about, from, and while **interacting** with an external environment
- Learning what to do — **how to map situations to actions** — so as to maximize a numerical reward signal

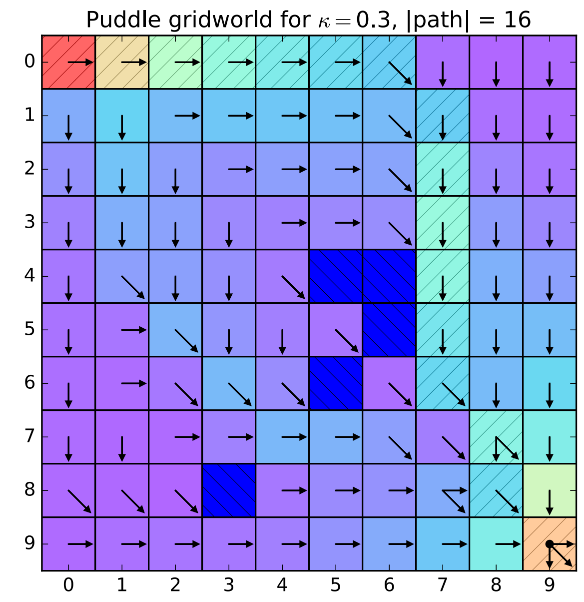


# The Agent Learns a Policy

**Policy** at step $t$, $\pi_t$:

- a mapping from states to action probabilities

- $\pi_t(s, a) =$ probability that $a_t = a$ when $s_t = s$




- **Reinforcement learning** methods specify how the agent changes its policy as a result of experience.
- Roughly, the agent's goal is to get as much reward as it can over the long run.

# Formal Model of Decision Problem

- A Markov Decision Process is defined by:
  - **state and action sets**
  - next-state **transition probabilities**
  - **reward expectations**

- The value of a state $s$ in an MDP under policy $\pi$ is

$$V^{\pi}(s) = E_{\pi}\left[\underbrace{\sum_{k=0}^{\infty} \gamma^{k} r_{t+k+1}}_{\text{return}} \middle| s_t = s\right]$$

**return**: discounted sum of rewards

CWI

# Bellman Equation (1950s)

State value function can be written as:

$$
\begin{aligned}
V^{\pi}(s) &= E_{\pi}\{R_t | s_t = s\} \\
&= E_{\pi}\{r_{t+1} + \gamma R_{t+1}\} \\
&= \sum_{a} \pi(s,a) \sum_{s'} \boldsymbol{P}^{a}_{ss'} \left[\boldsymbol{R}^{a}_{ss'} + \gamma V^{\pi}(s')\right]
\end{aligned}
$$

- The equation is **recursive**: $V(s)$ depends on $V(s')$.
- It sums over all possible **future returns**, weighted by their probability of occurring:
  - the **action probability** given by $\pi(s,a)$
  - the **state transition probability** given by $\boldsymbol{P}^{a}_{ss'}$

# Temporal Difference (TD) Learning

- **Temporal difference**:
  Look at the difference between the *current estimate* of the value of a state and the *sampled reward plus the discounted value of the next state*.

- Keep adjusting the value function aiming to reduce the **TD error** (until convergence).

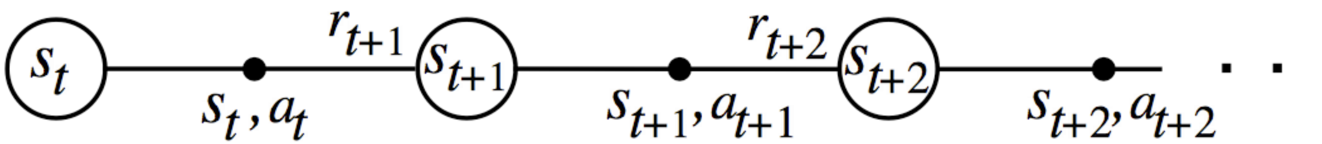
**TD error**: target – current estimate

$$V(s_t) \leftarrow V(s_t) + \alpha[\overbrace{\underbrace{r_{t+1} + \gamma V(s_{t+1})} - V(s_t)}]$$

**target**: an estimate of the return

# Learning an Action-Value Function

Estimate $Q^{\pi}$ for the current behaviour policy $\pi$ .




After every transition, update your estimate for Q as:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \underbrace{[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]}_{\text{TD error for Q}}$$

(if $s_{t+1}$ is terminal, then $Q(s_{t+1}, a_{t+1}) = 0$)

# Deriving the Policy

- For any (optimal) value function, we can easily derive a policy that yields it:

$$\pi(s) = \underset{a}{\operatorname{argmax}}\, Q(s, a)\,, \forall s \in S$$

- Typically, during learning we want to balance **exploration** (random actions) and **exploitation** (greedy actions)

- We distinguish a **target policy** (what we wish to learn) and a **behavior policy** (how we collect experience)

# On-Policy vs. Off-Policy

**Sarsa** (on-policy):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

**target policy**

**Q-learning** (off-policy):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right]$$

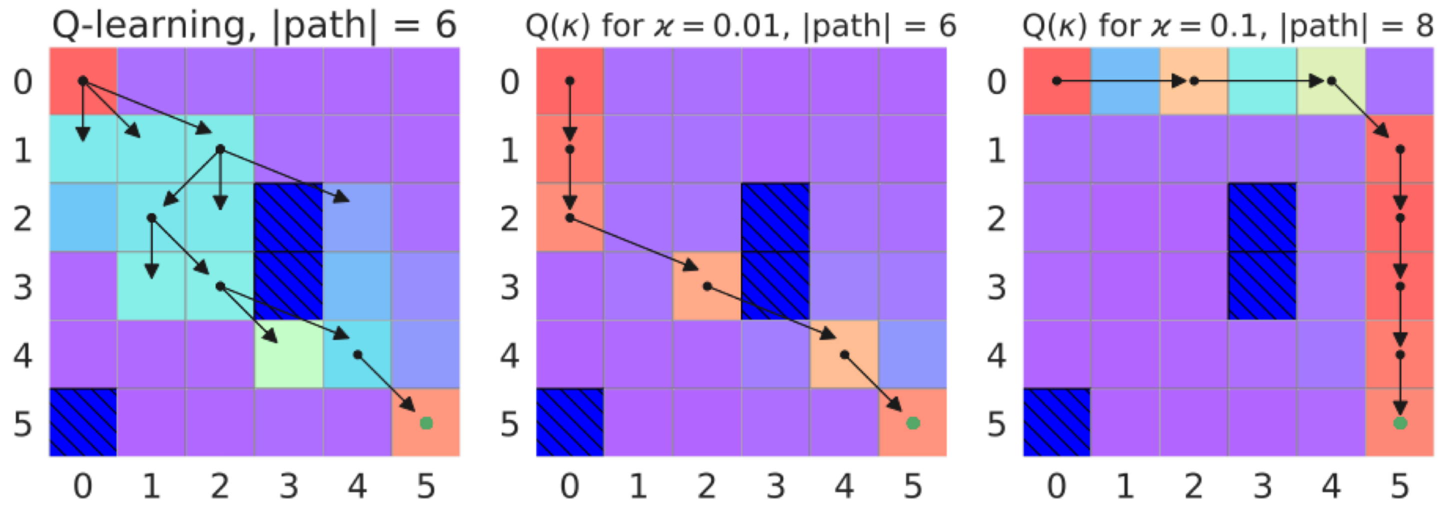
# Robust RL for Critical Domains

- Our idea in a nutshell: encode the expected probability (and model) of attacks or failures in the TD error $\delta_t$.

- E.g., malicious attack with probability $\varkappa$:

$$\delta_t = r_{t+1} + \gamma \underbrace{\left( (1-\varkappa) \max_a Q(s_{t+1}, a) + \varkappa \min_a Q(s_{t+1}, a) \right)}_{\text{modified estimate of next state value}} - Q(s_t, a_t)$$

# Results



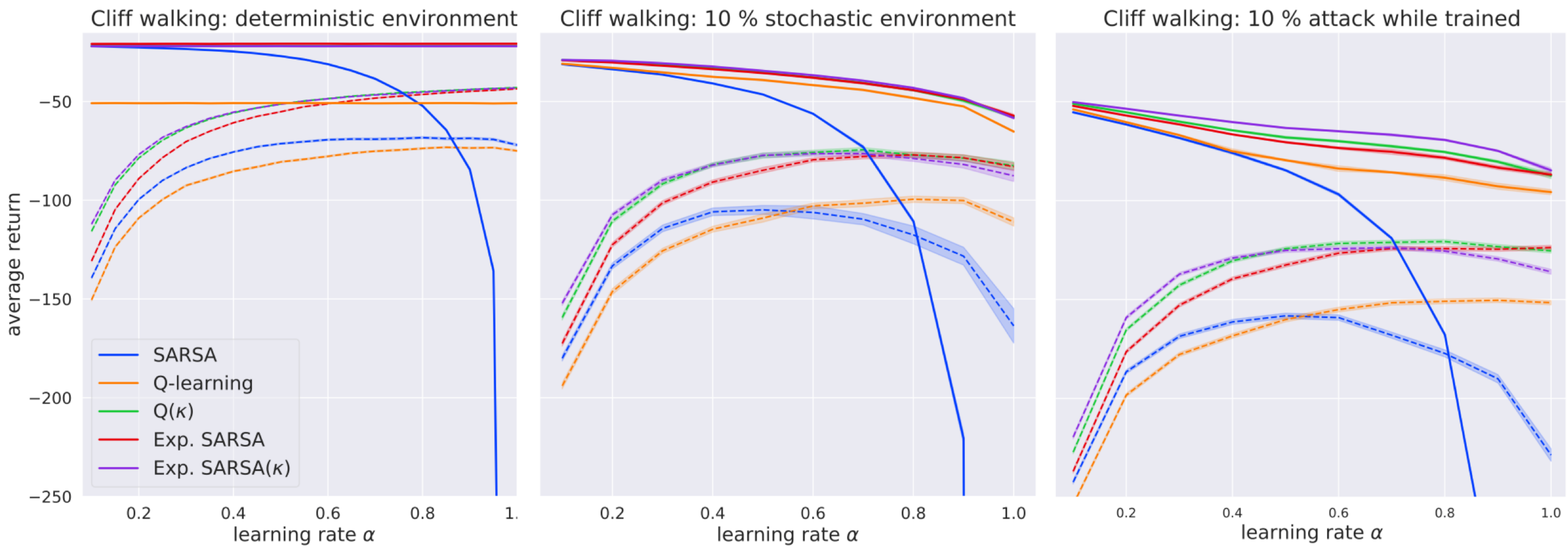

- For increasing attack probability $\varkappa$ we learn an increasingly safe policy

# $\kappa$-methods

We can easily build versions of standard TD methods based on this idea.

- Modified TD update rule: $$Q^\pi(s,a) \leftarrow Q^\pi(s,a) + \alpha\Big[\underbrace{r + \gamma V^\kappa(s')}_{\text{target}} - Q^\pi(s,a)\Big]$$
- **Q($\kappa$)-learning:** $$V^\kappa(s) = (1-\varkappa)\max_a Q(s,a) + \varkappa \min_a Q(s,a)$$
- **Expected Sarsa($\kappa$):** $$\begin{aligned} V^\kappa(s) &= (1-\varkappa)\mathbb{E}_{a\sim\pi}\big[Q(s,a)\big] + \varkappa \min_a Q(s,a) \\ &= (1-\varkappa)\sum_a \pi(a|s)Q(s,a) + \varkappa \min_a Q(s,a) \end{aligned}$$
- General $\kappa$-model: $$V^\kappa(s) = \sum_{\sigma\in C} p(\sigma|s)\sum_a \sigma(s,a)Q^\pi(s,a)$$

# Results

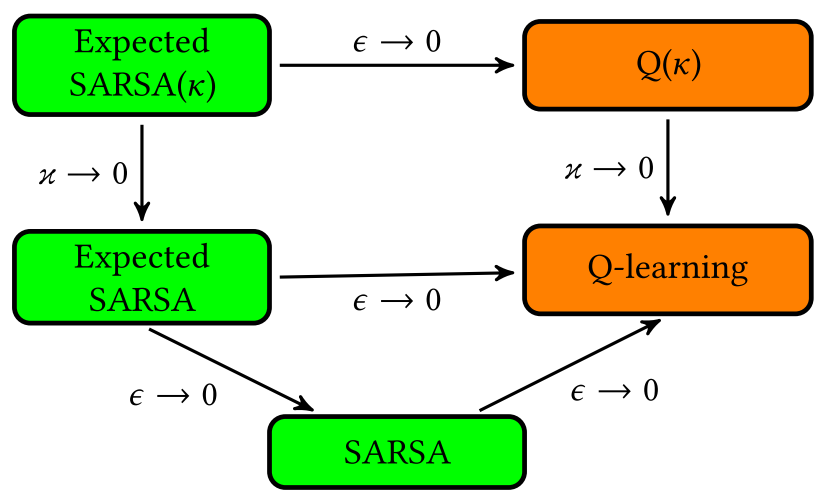



- Our new methods (**green** and **purple**) outperform the original TD methods on which they are based

# Theoretical Analysis

- We prove convergence of both Q($\kappa$) and Expected SARSA($\kappa$) to two different fixed points:
  - to the optimal value function $Q^\star$ of the original MDP in the limit where $\kappa \to 0$; and
  - to the optimal robust value function $Q^\star_\kappa$ of the MDP that is generalized w.r.t. $\kappa$ for constant parameter $\varkappa$.

- Note that ***optimality*** in this sense is purely induced by the relevant operator.



# Conclusion

- $\kappa$-versions of standard TD methods learn a **safer** policy **before experiencing the deviation**
  - Especially beneficial in the early learning stage.
  - Robust against model mis-specification.

- Proven **convergence** to *optimal* value function.

- Promising empirical results in single- and multi-agent settings.