NPFL114, Lecture 10

Sequence Prediction, Reinforcement Learning

Milan Straka

Structured Prediction

Structured Prediction

Consider generating a sequence of $y_1, \ldots, y_N \in Y^N$ given input $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$.

Structured Prediction

Consider generating a sequence of $y_1, \ldots, y_N \in Y^N$ given input $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$. Predicting each sequence element models the distribution $P(y_i | \boldsymbol{X})$.

Structured Prediction

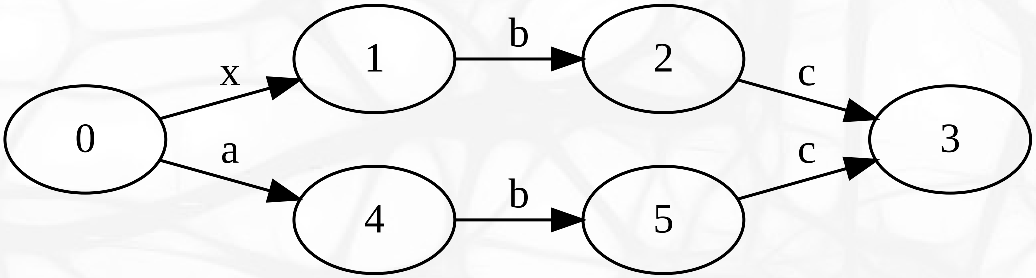
Consider generating a sequence of $y_1, \ldots, y_N \in Y^N$ given input $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$. Predicting each sequence element models the distribution $P(y_i|\boldsymbol{X})$. There are two problems with the above approach:

1. There may be dependencies among the y_i themselves.

Structured Prediction

Consider generating a sequence of $y_1, \ldots, y_N \in Y^N$ given input $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$. Predicting each sequence element models the distribution $P(y_i | \boldsymbol{X})$. There are two problems with the above approach:

1. There may be dependencies among the y_i themselves.
2. Even if we consider dependencies, if we compute $\operatorname{softmax}$ for each y_i individually, we will hit the *label bias problem*.



Seq2seq Beam Search Decoding

So far we described only greedy decoding in a sequence to sequence decoder.

However, such decoding might not find the most probable output sequence.

We might consider a *beam search*, where we iteratively compute some fixed number (a *beam size*) of best output sequences.

Conditional Random Fields

Let $G = (V, E)$ be a graph such that Y is indexed by vertices of G. Then $(\boldsymbol{X}, \boldsymbol{y})$ is a conditional markov field, if the random variables $\boldsymbol{y}$ conditioned on $\boldsymbol{X}$ obey the Markov property with respect to the graph, i.e.,

$$P(y_i | \boldsymbol{X}, y_j, i \neq j) = P(y_i | \boldsymbol{X}, y_j, (i, j) \in E).$$

Conditional Random Fields

Let $G = (V, E)$ be a graph such that Y is indexed by vertices of G. Then $(\boldsymbol{X}, \boldsymbol{y})$ is a conditional markov field, if the random variables $\boldsymbol{y}$ conditioned on $\boldsymbol{X}$ obey the Markov property with respect to the graph, i.e.,

$$P(y_i | \boldsymbol{X}, y_j, i \neq j) = P(y_i | \boldsymbol{X}, y_j, (i, j) \in E).$$

Usually we assume that dependencies of $\boldsymbol{y}$, conditioned on $\boldsymbol{X}$, form a chain.

CRF Output Layer

For a sequence of $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$ and $(y_1, \ldots, y_N)$, we define a score as

$$s(\boldsymbol{X}, \boldsymbol{y}; \boldsymbol{\theta}, \boldsymbol{A}) = \sum_{i=1}^{N} \big(\boldsymbol{A}_{y_{i-1}, y_i} + f_{\boldsymbol{\theta}}(y_i | \boldsymbol{X})\big).$$

CRF Output Layer

For a sequence of $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$ and $(y_1, \ldots, y_N)$, we define a score as

$$s(\boldsymbol{X}, \boldsymbol{y}; \boldsymbol{\theta}, \boldsymbol{A}) = \sum_{i=1}^{N} \left(\boldsymbol{A}_{y_{i-1}, y_i} + f_{\boldsymbol{\theta}}(y_i | \boldsymbol{X})\right).$$

We then define

$$p(\boldsymbol{y}|\boldsymbol{X}) = \operatorname{softmax}_{\boldsymbol{z} \in Y^N} \big(s(\boldsymbol{X}, \boldsymbol{z})\big)_{\boldsymbol{z}},$$

so that

$$\log p(\boldsymbol{y}|\boldsymbol{X}) = s(\boldsymbol{X}, \boldsymbol{y}) - \operatorname{logadd}_{\boldsymbol{z} \in Y^N} (s(\boldsymbol{X}, \boldsymbol{z})).$$

CRF Output Layer

We can compute $p(\boldsymbol{y}|\boldsymbol{X})$ efficiently using dynamic programming. If we denote $\alpha_t(k)$ as probability of all sentences with t elements with the last y being k.

We can then show that

$$\begin{aligned}\alpha_t(k) &= \operatorname{logadd}_{\boldsymbol{z}}(s(\boldsymbol{X}, \boldsymbol{z})) \\ &= f_{\boldsymbol{\theta}}(y_t = k|\boldsymbol{X}_{1:t}) + \operatorname{logadd}_i(\alpha_{t-1}(i) + \boldsymbol{A}_{i,k}).\end{aligned}$$

For efficient implementation, we use the fact that

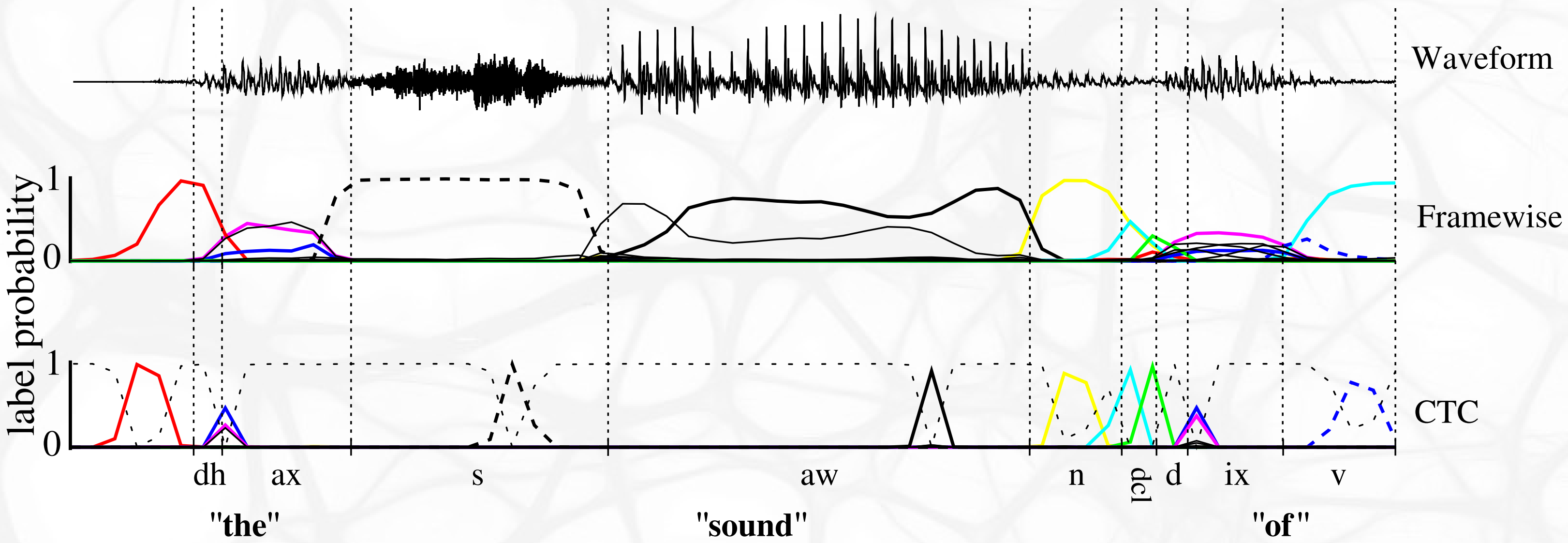
$$\ln(a+b) = \ln a + \ln(1 + e^{\ln b - \ln a}).$$

Connectionist Temporal Classification

Let us again consider generating a sequence of $y_1, \ldots, y_M$ given input $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, but this time $M \leq N$ and there is no explicit alignment of $\boldsymbol{x}$ and y in the gold data.

Connectionist Temporal Classification

Let us again consider generating a sequence of $y_1, \ldots, y_M$ given input $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, but this time $M \leq N$ and there is no explicit alignment of $\boldsymbol{x}$ and y in the gold data.



Connectionist Temporal Classification

We enlarge the set of output labes by a – (*blank*) and perform a classification for every input element to produce an *extended labeling*. We then post-process it by the following rules:

1. We remove neighboring symbols.
2. We remove the –.

Connectionist Temporal Classification

We enlarge the set of output labes by a – (*blank*) and perform a classification for every input element to produce an *extended labeling*. We then post-process it by the following rules:

1. We remove neighboring symbols.
2. We remove the –.

Because the explicit alignment of inputs and labels is not known, we consider *all possible* alignments.

Connectionist Temporal Classification

Let us denote the probability of label l at time t as p_l^t.

We now consider a modified label sequence by inserting a – to the beginning, to the end and between every pair of labels. Using this modified labeling we define

$$\alpha_t(s) \stackrel{\text{def}}{=} \sum_{\text{labeling } \boldsymbol{\pi} \text{ with labels of } \boldsymbol{\pi}_{1:t} = \boldsymbol{y}_{1:s}} \prod_{t'=1}^{t} p_{\boldsymbol{\pi}_{t'}}^{t'}.$$

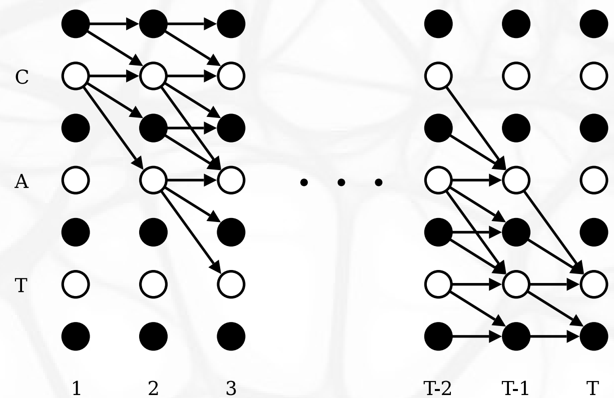
Connectionist Temporal Classification

Let us denote the probability of label l at time t as p_l^t.

We now consider a modified label sequence by inserting a – to the beginning, to the end and between every pair of labels. Using this modified labeling we define

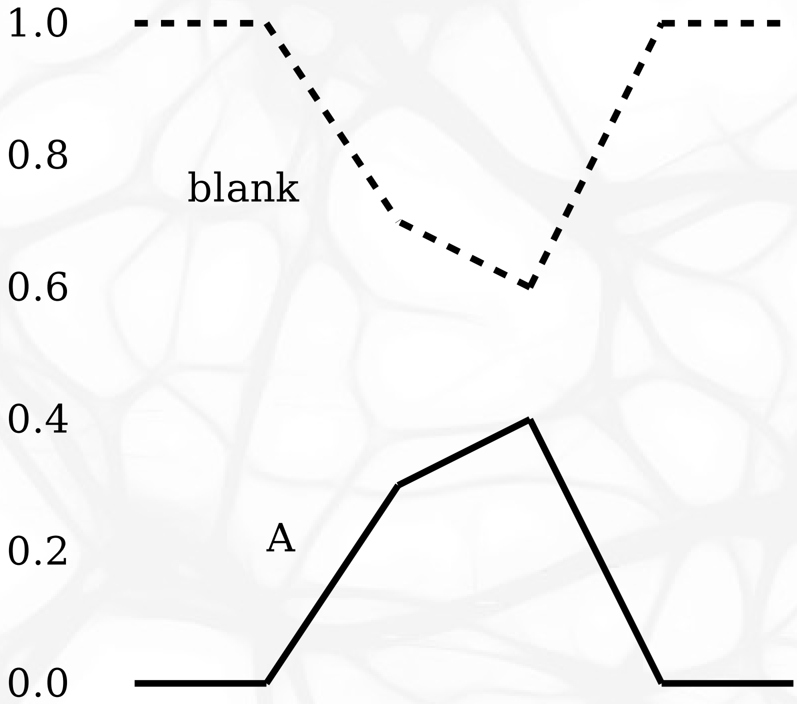
$$\alpha_t(s) \stackrel{\text{def}}{=} \sum_{\text{labeling } \boldsymbol\pi \text{ with labels of } \boldsymbol\pi_{1:t} = \boldsymbol y_{1:s}} \prod_{t'=1}^{t} p_{\boldsymbol\pi_{t'}}^{t'}.$$

Similarly to the CRF, we can use dynamic programming to compute α in polynomial time.



CTC Decoding

Unlike CRF, we cannot easily perform the decoding in an optimal way. We therefore either use greedy decoding, or a beam search.

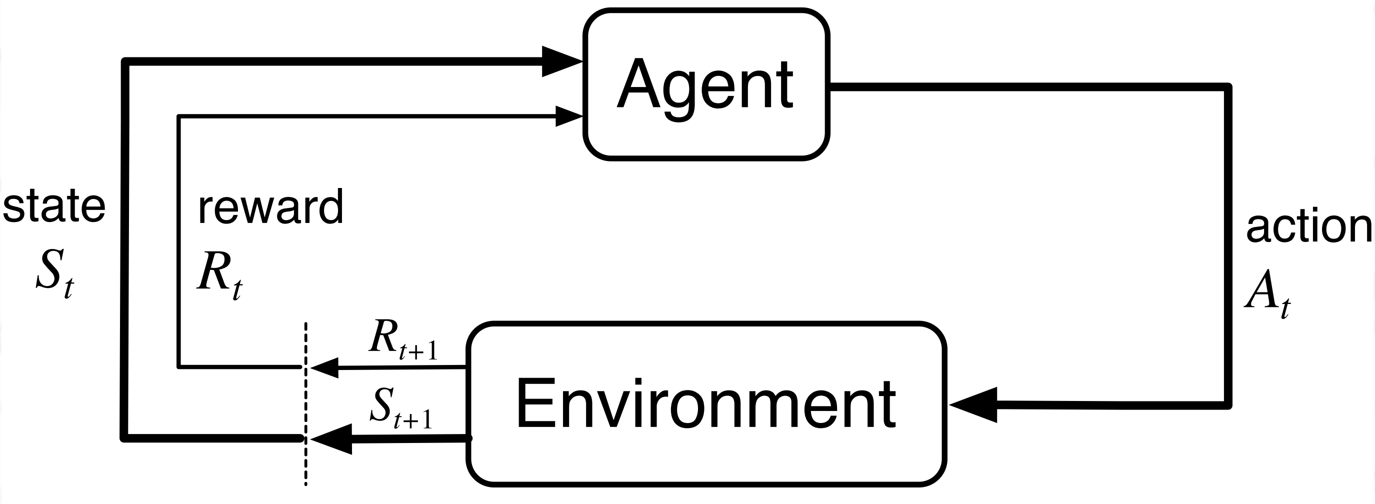


$$
\begin{aligned}
p(l=\text{blank}) &= p(\text{- -}) \\
&= 0.7*0.6 \\
&= 0.42
\end{aligned}
$$

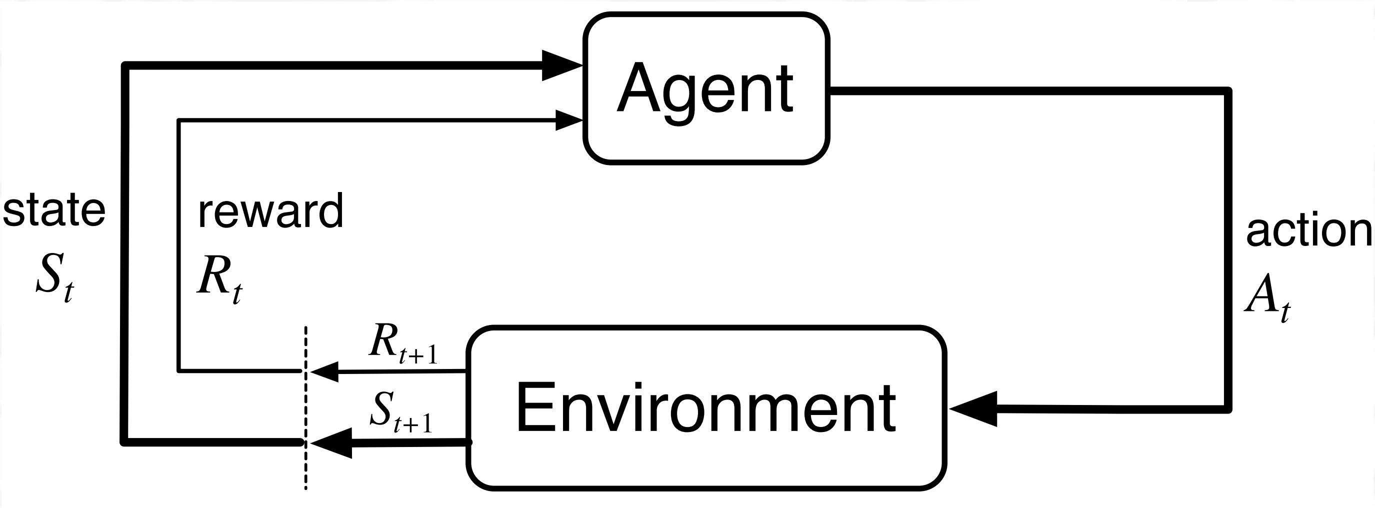
$$
\begin{aligned}
p(l=A) &= p(AA)+p(A\text{-})+p(\text{-}A) \\
&= 0.3*0.4 + 0.3*0.6 + 0.7*0.4 \\
&= 0.58
\end{aligned}
$$

Reinforcement Learning

Reinforcement Learning



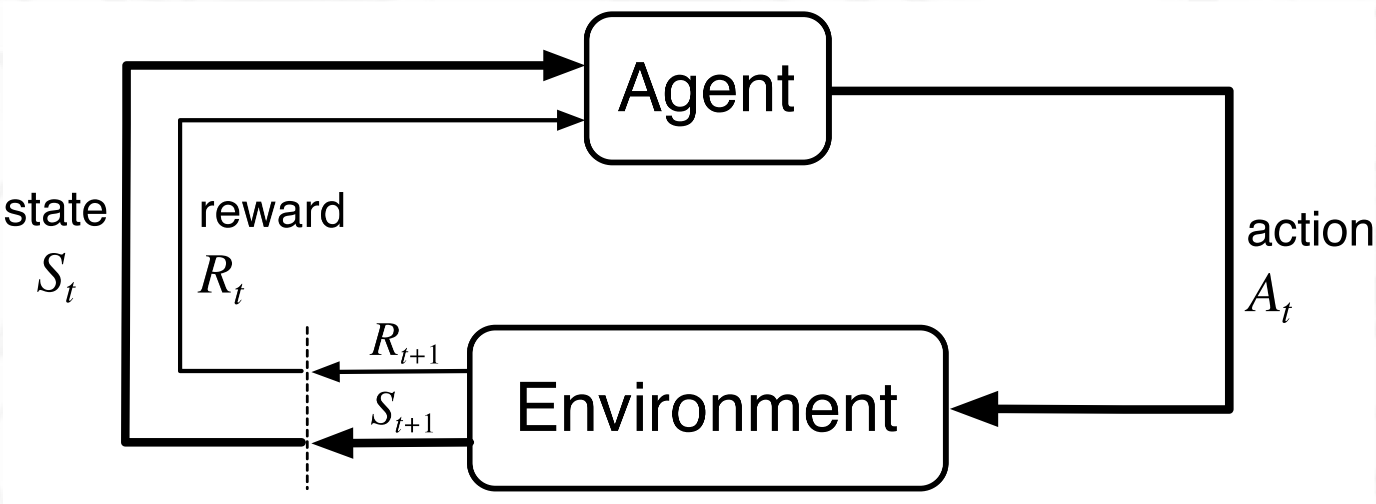
Reinforcement Learning



A *Markov decision process* is a quadruple $(\mathcal{S}, \mathcal{A}, P, \gamma)$, where:

- $\mathcal{S}$ is a set of states,
- $\mathcal{A}$ is a set of actions,
- $P(S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a)$ is a probability that action $a \in \mathcal{A}$ will lead from state $s \in \mathcal{S}$ to $s' \in \mathcal{S}$, producing a *reward* $r \in \mathbb{R}$,
- $\gamma \in [0, 1]$ is a *discount factor*.

Reinforcement Learning

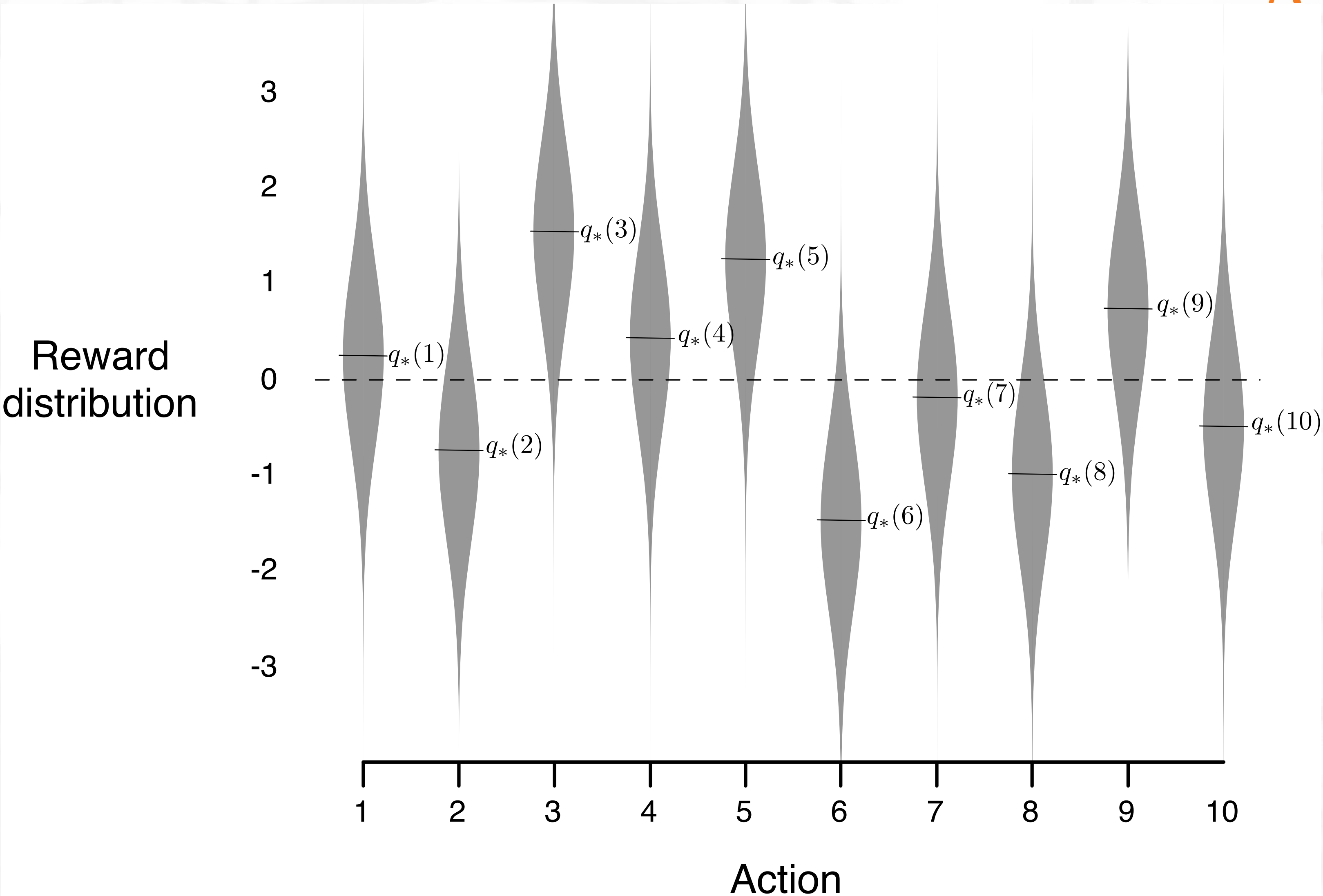


A *Markov decision process* is a quadruple $(\mathcal{S}, \mathcal{A}, P, \gamma)$, where:

- $\mathcal{S}$ is a set of states,
- $\mathcal{A}$ is a set of actions,
- $P(S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a)$ is a probability that action $a \in \mathcal{A}$ will lead from state $s \in \mathcal{S}$ to $s' \in \mathcal{S}$, producing a *reward* $r \in \mathbb{R}$,
- $\gamma \in [0, 1]$ is a *discount factor*.

Let a *return* G_t be $G_t \stackrel{\text{def}}{=} \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}$.

K-armed Bandits



K-armed Bandits

Let $q_*(a)$ be the real *value* of an action a:

$$q_*(a) = \mathbb{E}[R_{t+1} | A_t = a].$$

K-armed Bandits

Let $q_*(a)$ be the real *value* of an action a:

$$q_*(a) = \mathbb{E}[R_{t+1} | A_t = a].$$

Denoting $Q_t(a)$ our estimated value of action a at time t, we would like $Q_t(a)$ to converge to $q_*(a)$.

K-armed Bandits

Let $q_*(a)$ be the real *value* of an action a:

$$q_*(a) = \mathbb{E}[R_{t+1} | A_t = a].$$

Denoting $Q_t(a)$ our estimated value of action a at time t, we would like $Q_t(a)$ to converge to $q_*(a)$.

A natural way to estimate $Q_t(a)$ is

$$Q_t(a) \stackrel{\text{def}}{=} \frac{\text{sum of rewards when action } a \text{ is taken}}{\text{number of times action } a \text{ was taken}}.$$

K–armed Bandits

Following the definition of $Q_t(a)$, we could choose a *greedy action* A_t as

$$A_t(a) \stackrel{\text{def}}{=} \arg\max_a Q_t(a).$$

K–armed Bandits

Following the definition of $Q_t(a)$, we could choose a *greedy action* A_t as

$$A_t(a) \stackrel{\text{def}}{=} \arg\max_a Q_t(a).$$

Exploitation versus Exploration

Choosing a greedy action is *exploitation* of current estimates. We however also need to *explore* the space of actions to improve our estimates.

K-armed Bandits

Following the definition of $Q_t(a)$, we could choose a *greedy action* A_t as

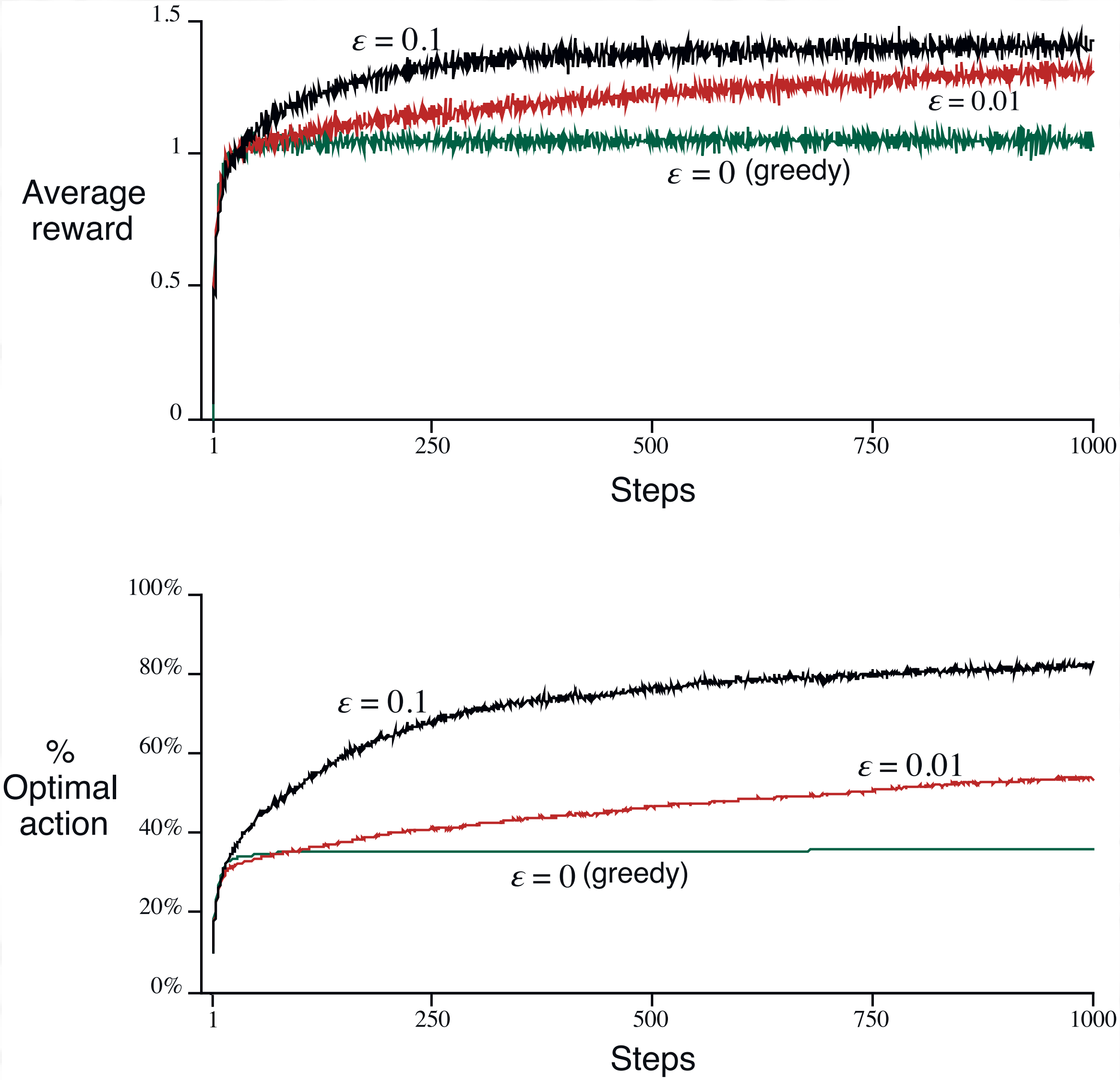
$$A_t(a) \stackrel{\text{def}}{=} \arg\max_a Q_t(a).$$

Exploitation versus Exploration

Choosing a greedy action is *exploitation* of current estimates. We however also need to *explore* the space of actions to improve our estimates.

An ε-*greedy* method follows the greedy action with probability $1-\varepsilon$, and chooses a uniformly random action with probability ε.

K-armed Bandits



K–armed Bandits

Incremental Implementation

Let Q_n be an estimate using n rewards $R_1, \ldots, R_n$.

$$
\begin{aligned}
Q_n &= \frac{1}{n} \sum_{i=1}^{n} R_i \\
&= \frac{1}{n}\Big(R_n + \frac{n-1}{n-1} \sum_{i=1}^{n-1} R_i\Big) \\
&= \frac{1}{n}\big(R_n + (n-1) Q_{n-1}\big) \\
&= \frac{1}{n}\big(R_n + nQ_{n-1} - Q_{n-1}\big) \\
&= Q_{n-1} + \frac{1}{n}\big(R_n - Q_{n-1}\big)
\end{aligned}
$$

Non-stationary Problems

Analogously to the solution obtained for a stationary problem, we consider

$$Q_{n+1} = Q_n + \alpha(R_{n+1} - Q_n).$$

Policies

A *policy* π computes a distribution of actions in a given state, i.e., $\pi(a|s)$ corresponds to a probability of performing an action a in state s.

Policies

A *policy* π computes a distribution of actions in a given state, i.e., $\pi(a|s)$ corresponds to a probability of performing an action a in state s.

Value Function

To evaluate a quality of policy, we define *value function* $v_\pi(s)$, or more explicitly *state-value function*, as

$$v_\pi(s) \stackrel{\text{def}}{=} \mathbb{E}_\pi[G_t | S_t = s].$$

Policies

A *policy* π computes a distribution of actions in a given state, i.e., $\pi(a|s)$ corresponds to a probability of performing an action a in state s.

Value Function

To evaluate a quality of policy, we define *value function* $v_\pi(s)$, or more explicitly *state-value function*, as

$$v_\pi(s) \stackrel{\text{def}}{=} \mathbb{E}_\pi\left[G_t | S_t = s\right].$$

An *action-value function* for policy π is defined analogously as

$$q_\pi(s, a) \stackrel{\text{def}}{=} \mathbb{E}_\pi\left[G_t | S_t = s, A_t = a\right].$$

Policies

A *policy* π computes a distribution of actions in a given state, i.e., $\pi(a|s)$ corresponds to a probability of performing an action a in state s.

Value Function

To evaluate a quality of policy, we define *value function* $v_\pi(s)$, or more explicitly *state-value function*, as

$$v_\pi(s) \stackrel{\text{def}}{=} \mathbb{E}_\pi[G_t|S_t = s].$$

An *action-value function* for policy π is defined analogously as

$$q_\pi(s, a) \stackrel{\text{def}}{=} \mathbb{E}_\pi[G_t|S_t = s, A_t = a].$$

It follows that

$$q_\pi(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = a].$$

Optimal Policy

As value functions define an partial ordering of policies ($\pi' \geq \pi$ if and only if for all states s, $v_{\pi'}(s) \geq v_\pi(s)$), it can be proven that there always exists an *optimal policy* π_*, which is better or equal to all other policies.

Intuitively, $\pi_*(s) = \arg\max_a q_*(s, a)$.

Optimal Policy

As value functions define an partial ordering of policies ($\pi' \geq \pi$ if and only if for all states s, $v_{\pi'}(s) \geq v_\pi(s)$), it can be proven that there always exists an *optimal policy* π_*, which is better or equal to all other policies.

Intuitively, $\pi_*(s) = \arg\max_a q_*(s, a)$.

Policy Improvement Theorem

Let π and π' be any pair of deterministic policies, such that $q_\pi(s, \pi'(s)) \geq v_\pi(s)$. Then $\pi' \geq \pi$, i.e., for all states s, $v_{\pi'}(s) \geq v_\pi(s)$.

Monte Carlo Control

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-soft policy

$Q(s, a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s, a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \operatorname{average}(Returns(S_t, A_t))$

$A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)

Fo all $a \in \mathcal{A}(S_t)$:

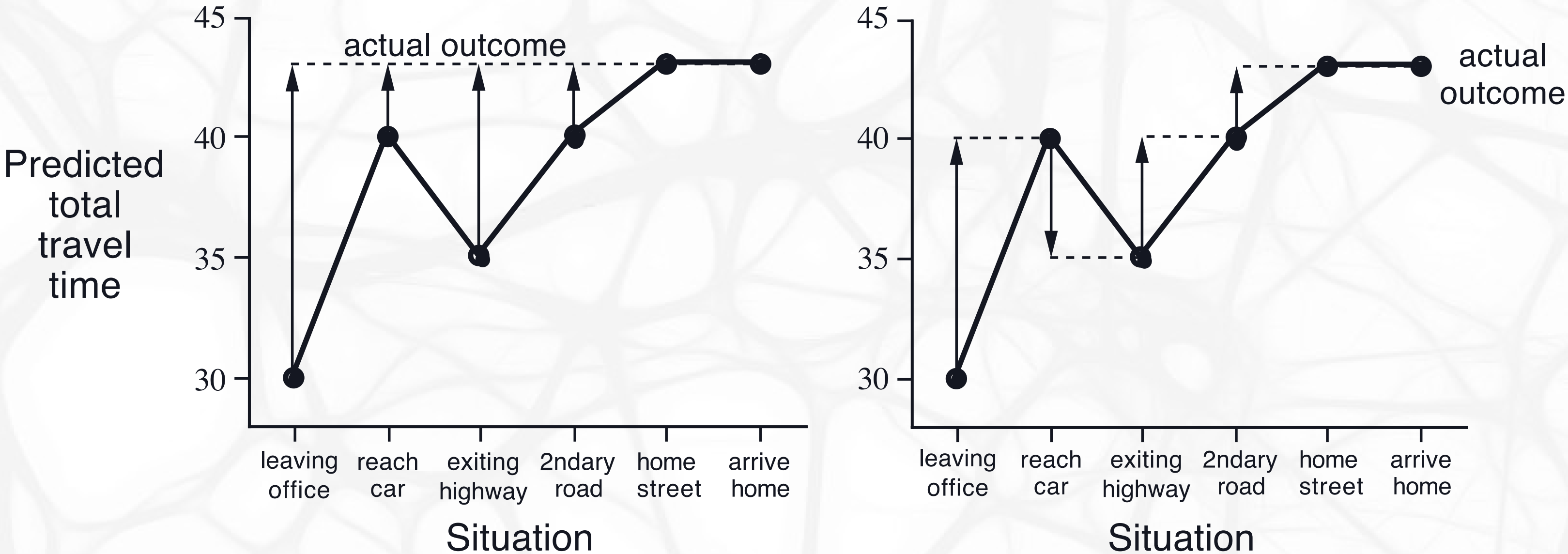
$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

Temporal Difference Methods

State	*Elapsed Time (minutes)*	*Predicted Time to Go*	*Predicted Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

Temporal Difference Methods

State	*Elapsed Time (minutes)*	*Predicted Time to Go*	*Predicted Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43



Temporal Difference Methods

A straightforward modification of Monte Carlo algorithm with constant-step update and temporal difference is given by

$$Q(S_t, A_T) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

and is called *Sarsa* $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$.

Temporal Difference Methods

A straightforward modification of Monte Carlo algorithm with constant-step update and temporal difference is given by

$$Q(S_t, A_T) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

and is called *Sarsa* $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$.

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Loop for each step of episode:
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until S is terminal

Q-learning

Q-learning is another TD control algorithm by (Watkins, 1989), defined by

$$Q(S_t, A_T) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)].$$

Q-learning

Q-learning is another TD control algorithm by (Watkins, 1989), defined by

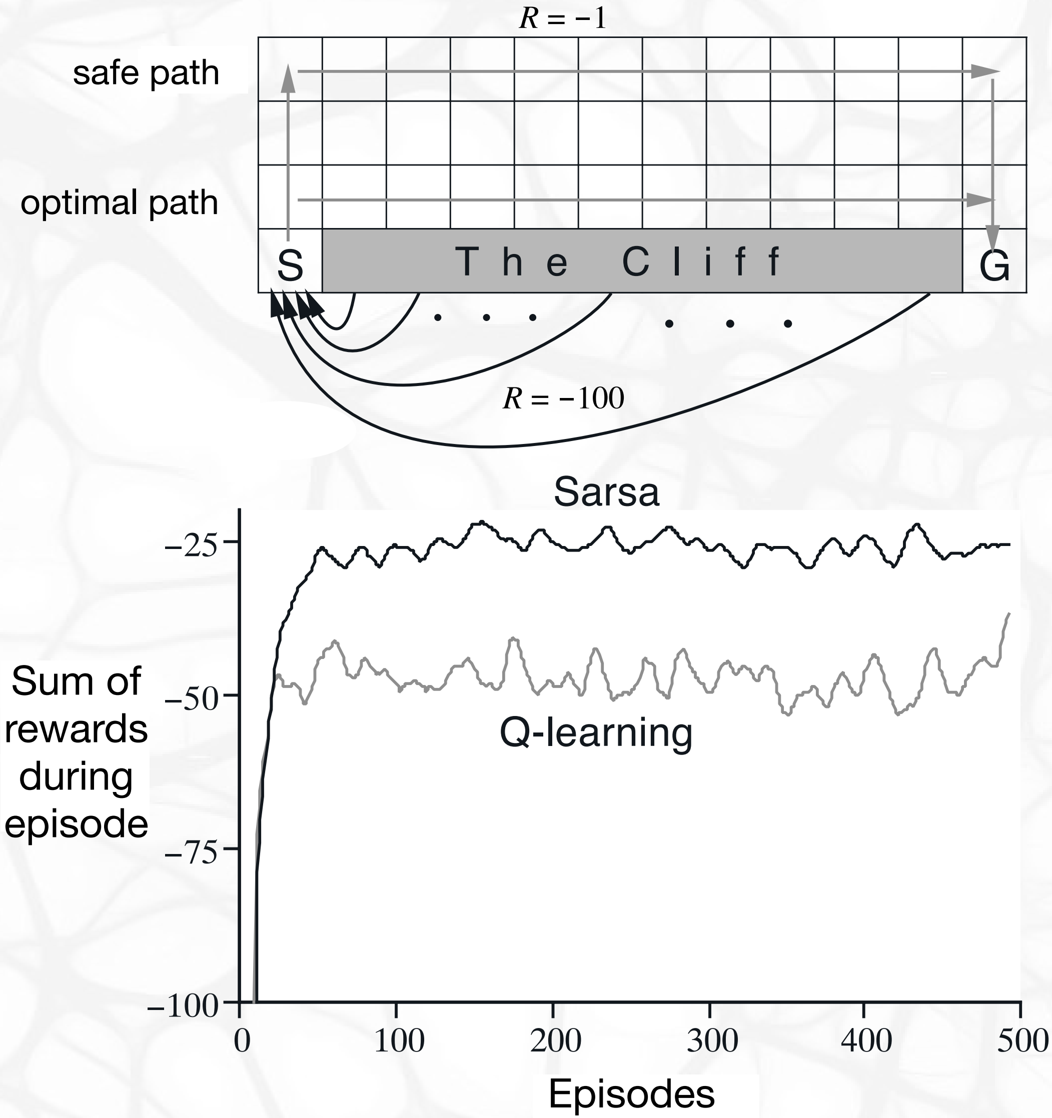
$$Q(S_t, A_T) \leftarrow Q(S_t, A_t) + \alpha\big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\big].$$

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

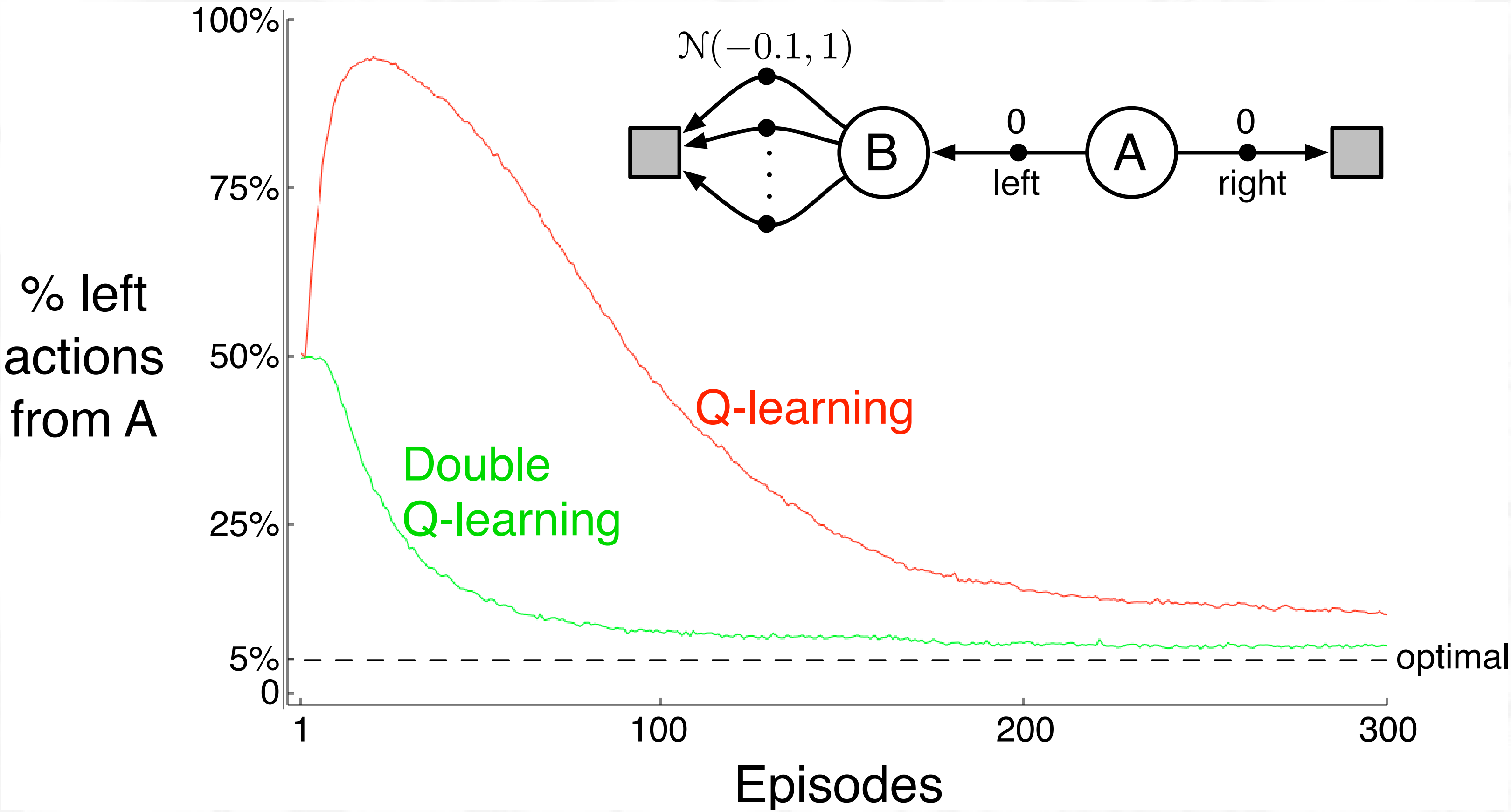
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$
    until S is terminal

Sarsa vs Q-learning



Double Q-learning



Double Q-learning

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(\textit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using the policy ε-greedy in $Q_1 + Q_2$
    Take action A, observe R, S'
    With 0.5 probabilility:

$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S', a)\big) - Q_1(S, A)\Big)$$

    else:

$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S', a)\big) - Q_2(S, A)\Big)$$

    $S \leftarrow S'$
  until S is terminal