NPFL114, Lecture 12

Sequence Prediction II, Reinforcement Learning II

Milan Straka

Structured Prediction

Structured Prediction

Conditional Random Fields (CRF)

Motivation

- Label-bias problem

Conditional Random Fields (CRF)

Motivation

- Label-bias problem
  - not a concrete definition
  - the inability of the model to assign individual weights to different sequence elements

Conditional Random Fields (CRF)

Motivation

- Label-bias problem
  - not a concrete definition
  - the inability of the model to assign individual weights to different sequence elements
- Global decoding

Conditional Random Fields (CRF)

Motivation

- Label-bias problem
  - not a concrete definition
  - the inability of the model to assign individual weights to different sequence elements
- Global decoding
  - *Česko Slovensko porazilo.*

Conditional Random Fields (CRF)

Proposed Solution

- Performm sentence-level softmax, and add weights for neighboring sequence outputs.

Conditional Random Fields (CRF)

Proposed Solution

- Performm sentence-level softmax, and add weights for neighboring sequence outputs.

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

Conditional Random Fields (CRF)

Proposed Solution

- Performm sentence-level softmax, and add weights for neighboring sequence outputs.

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

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

Conditional Random Fields (CRF)

Proposed Solution

- Performm sentence-level softmax, and add weights for neighboring sequence outputs.

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

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

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

Conditional Random Fields (CRF)

Computation

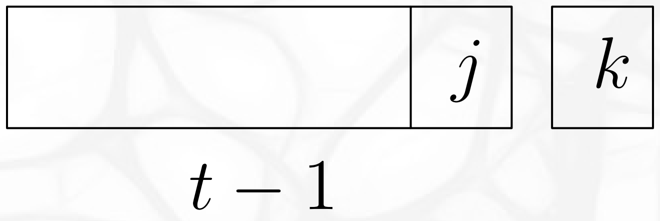
We can compute $p(\boldsymbol{y}|\boldsymbol{X})$ efficiently using dynamic programming. If we denote $\alpha_t(k)$ as probability of all sentences with t elements with the last y being k.

Conditional Random Fields (CRF)

Computation

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.

The core idea is the following:



$$\alpha_t(k) = f_{\boldsymbol{\theta}}(y_t = k|\boldsymbol{X}) + \operatorname{logadd}_{j\in Y)}(\alpha_{t-1}(j) + \boldsymbol{A}_{j,k}).$$

Conditional Random Fields (CRF)

Computation

Inputs: Network computing $f_{\boldsymbol\theta}(y_t = k|\boldsymbol X)$, an unnormalized probability of output sequence element probability being k in time t.
Inputs: Transition matrix $\boldsymbol A \in \mathbb{R}^{Y\times Y}$.
Inputs: Input sequence $\boldsymbol X$ of length N, gold labeling $\boldsymbol y \in Y^N$.
Outputs: Value of $\log p(\boldsymbol y|\boldsymbol X)$.
Complexity: $\mathcal{O}(N \cdot Y^2)$.

```
• For $k = 1, \ldots, Y$:
    ◦ $\alpha_0(k) \leftarrow 0$
• For $t = 1, \ldots, N$:
    ◦ For $k = 1, \ldots, Y$ :
        ▪ $\alpha_t(k) \leftarrow 0$
        ▪ For $j = 1, \ldots, Y$:
            ▪ $\alpha_t(k) \leftarrow \operatorname{logadd}(\alpha_t(k), \alpha_{t-1}(j) + \boldsymbol A_{j,k})$
        ▪ $\alpha_t(k) \leftarrow \alpha_t(k) + f_{\boldsymbol\theta}(y_t = k|\boldsymbol X)$
```

Conditional Random Fields (CRF)

Decoding

We can perform optimal decoding, by using the same algorithm, only replacing logadd with $\max$ and tracking where the maximum was attained.

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 labels 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 (denoted $\mathcal{B}$):

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

Connectionist Temporal Classification

We enlarge the set of output labels 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 (denoted $\mathcal{B}$):

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

We enlarge the set of output labels 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 (denoted $\mathcal{B}$):

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.

Denoting the probability of label l at time t as p_l^t, we define

$$\alpha^t(s) \stackrel{\text{def}}{=} \sum_{\text{labeling } \boldsymbol{\pi}:\mathcal{B}(\boldsymbol{\pi}_{1:t})=\boldsymbol{y}_{1:s}} \prod_{t'=1}^{t} p_{\boldsymbol{\pi}_{t'}}^{t'}.$$

CRF and CTC Comparison

In CRF, we normalize the whole sentences, therefore we need to compute unnormalized probabilities for all the (exponentially many) sentences. Decoding can be performed optimally.

CRF and CTC Comparison

In CRF, we normalize the whole sentences, therefore we need to compute unnormalized probabilities for all the (exponentially many) sentences. Decoding can be performed optimally.

In CTC, we normalize per each label. However, because we do not have explicit alignment, we compute probability of a labeling by summing probabilities of (generally exponentially many) extended labelings.

Connectionist Temporal Classification

Computation

When aligning an extended labeling to a regular one, we need to consider whether the extended labeling ends by a *blank* or not. We therefore define

$$\alpha_-^t(s) \overset{\text{def}}{=} \sum_{\text{labeling } \boldsymbol{\pi}:\mathcal{B}(\boldsymbol{\pi}_{1:t})=\boldsymbol{y}_{1:s},\pi_t=-} \prod_{t'=1}^{t} p_{\boldsymbol{\pi}_{t'}}^{t'}$$

$$\alpha_*^t(s) \overset{\text{def}}{=} \sum_{\text{labeling } \boldsymbol{\pi}:\mathcal{B}(\boldsymbol{\pi}_{1:t})=\boldsymbol{y}_{1:s},\pi_t\neq-} \prod_{t'=1}^{t} p_{\boldsymbol{\pi}_{t'}}^{t'}$$

and compute $\alpha^t(s)$ as $\alpha_-^t(s) + \alpha_*^t(s)$.

Connectionist Temporal Classification

Computation

We initialize αs as follows:

- $\alpha_-^1(0) \leftarrow p_-^1$
- $\alpha_*^1(1) \leftarrow p_{y_1}^1$

Connectionist Temporal Classification

Computation

We initialize αs as follows:

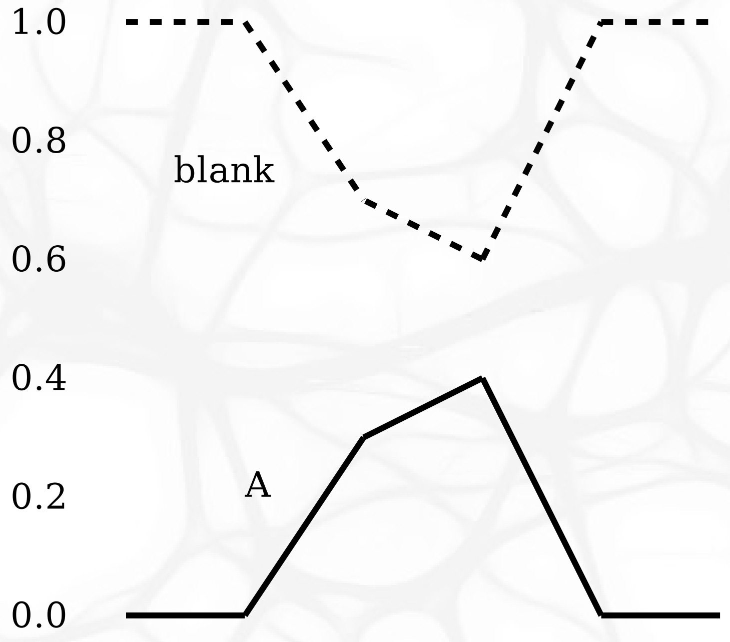
- $\alpha_-^1(0) \leftarrow p_-^1$
- $\alpha_*^1(1) \leftarrow p_{y_1}^1$

We then proceed recurrently according to:

- $\alpha_-^t(s) \leftarrow p_-^t\left(\alpha_-^{t-1}(s) + \alpha_*^{t-1}(s)\right)$
- $\alpha_*^t(s) \leftarrow \begin{cases} p_{y_t}^t\left(\alpha_*^{t-1}(s) + \alpha_*^{t-1}(s-1) + a_-^{t-1}(s-1)\right), \text{if } y_t \neq y_{t-1} \\ p_{y_t}^t\left(\alpha_*^{t-1}(s) + a_-^{t-1}(s-1)\right), \text{if } y_t = y_{t-1} \end{cases}$

CTC Decoding

Unlike CRF, we cannot perform the decoding optimally. The key observation is that while an optimal extended labeling can be extended into an optimal labeling of a larger length, the same does not apply to regular (non-extended) labeling. The problem is that regular labeling coresponds to many extended labelings, which are modified each in a different way during an extension of the regular labeling.



p(l=blank) = p(- -)
= 0.7*0.6
= 0.42

p(l=A) = p(AA)+p(A-)+p(-A)
= 0.3*0.4 + 0.3*0.6 + 0.7*0.4
= 0.58

CTC Decoding

Beam Search

To perform beam search, we keep k best regular labelings for each prefix of the extended labelings. For each regular labeling we keep both α_- and a_* and by *best* we mean such regular labelings with maximum $\alpha_- + \alpha_*$.

To compute best regular labelings for longer prefix of extended labelings, for each regular labeling in the beam we consider the following cases:

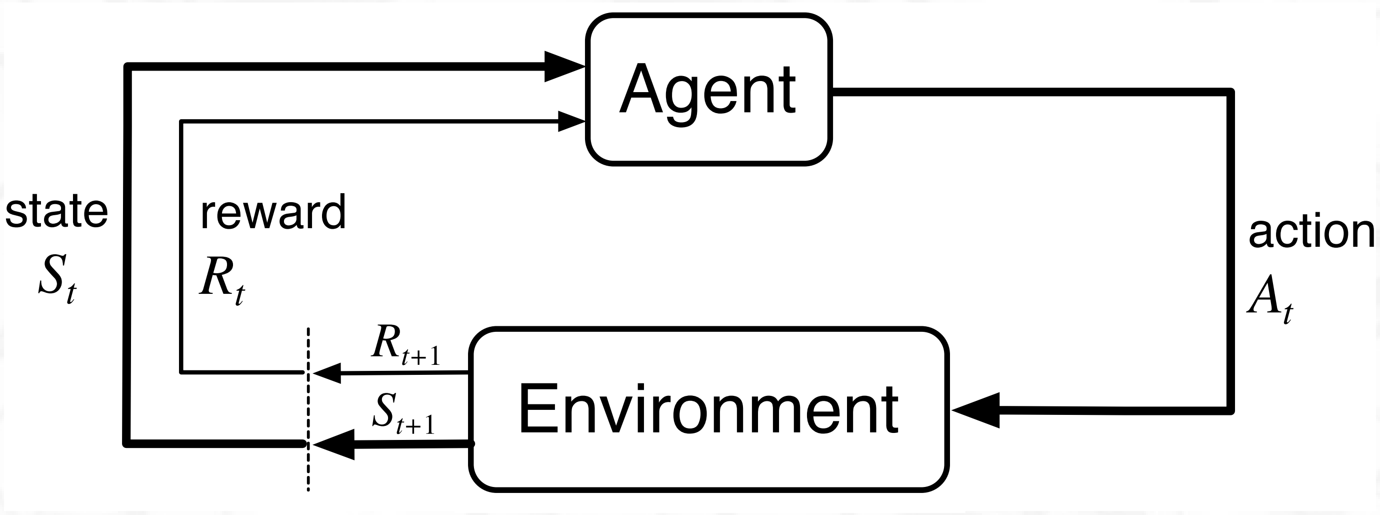
- adding a *blank* symbol, i.e., updating both α_- and α_*;
- adding any non-blank symbol, i.e., updating α_*.

Then, we merge the resulting candidates according to their regular labeling and keep only the k best.

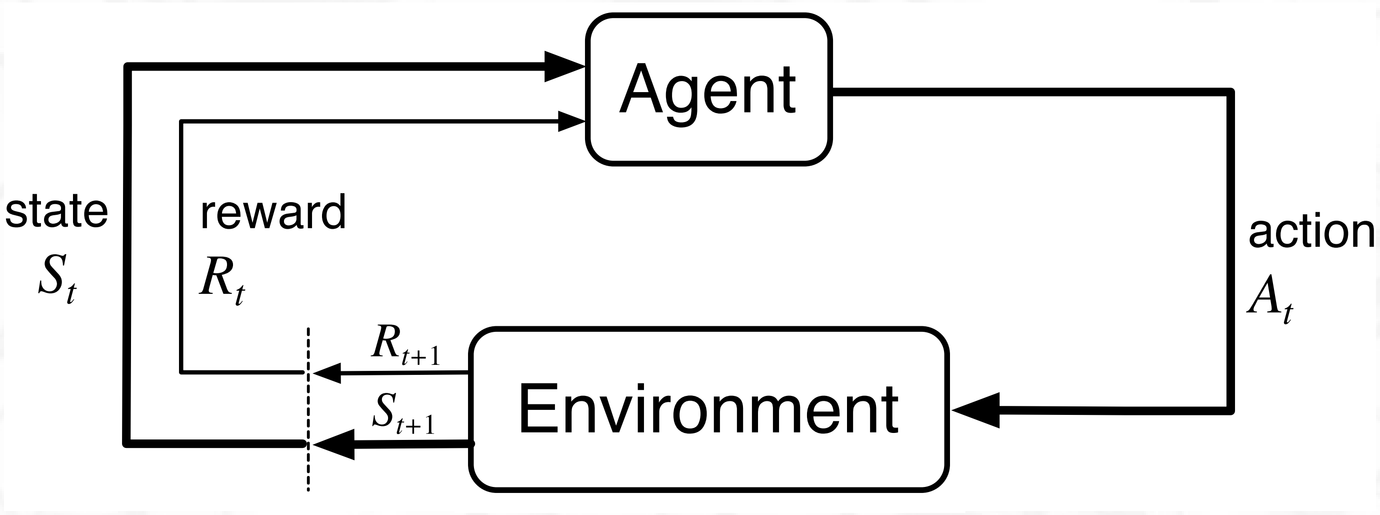
Reinforcement Learning

Reinforcement Learning

Reinforcement Learning



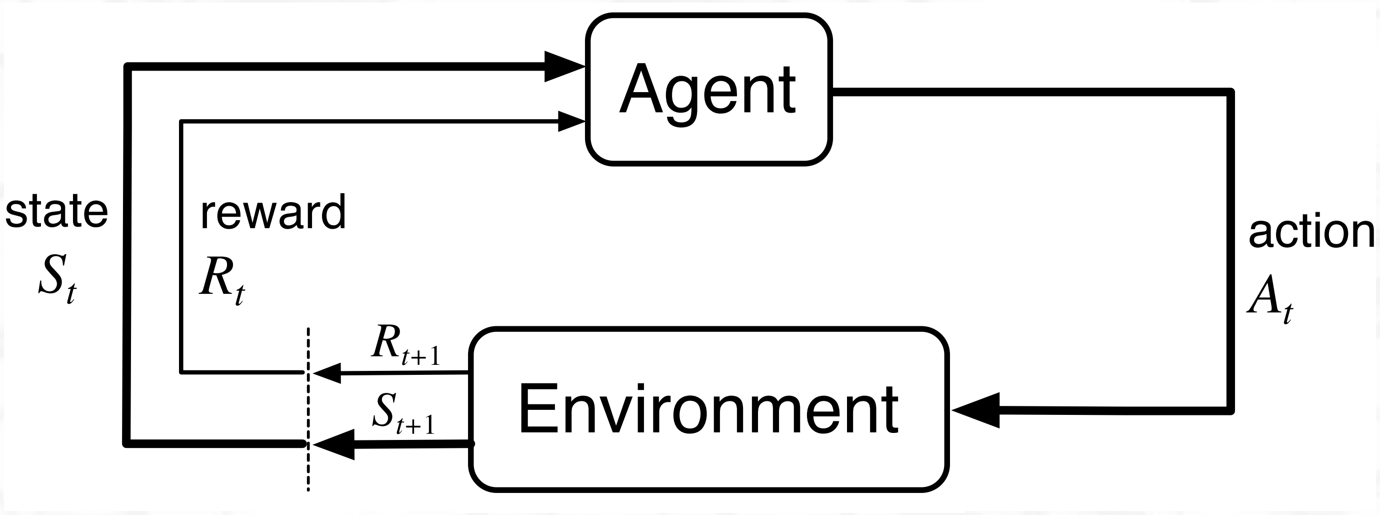
Reinforcement Learning



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 $G_t \overset{\text{def}}{=} \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}$.

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

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

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

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

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

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

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

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

It follows that

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

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 policies (both deterministic or stochastic), 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 $\pi \approx \pi_*$

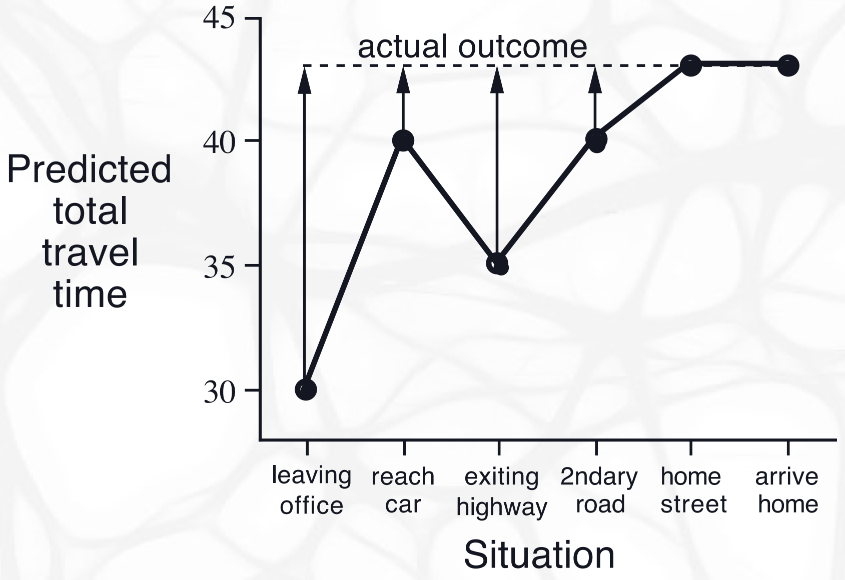
```
Algorithm parameter: small $\varepsilon > 0$
Initialize:
    $\pi \leftarrow$ an arbitrary $\varepsilon$-soft policy
    $Q(s, a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
    $Returns(s, a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):
    Generate an episode following $\pi$: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    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 $Returns(S_t, A_t)$
            $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
            $A^* \leftarrow \arg\max_a Q(S_t, a)$                    (with ties broken arbitrarily)
            Fo  all $a \in \mathcal{A}(S_t)$:
                $\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$
```

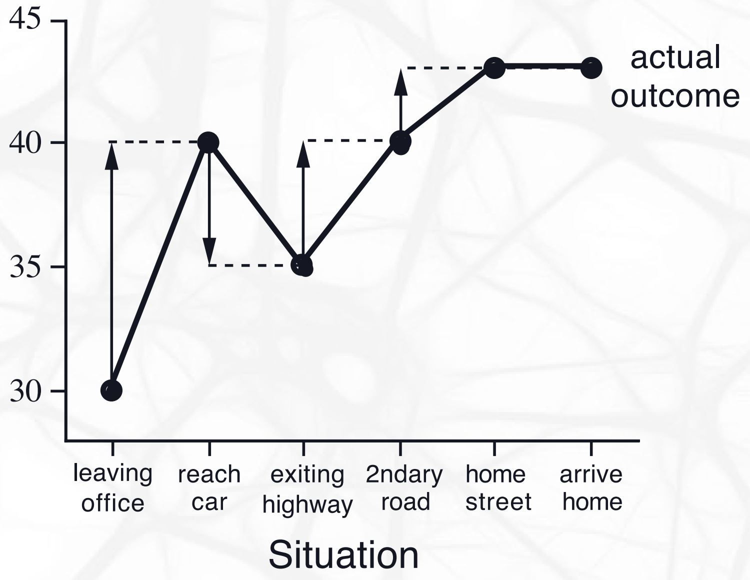
Temporal Difference Methods

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 $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Loop for each step of episode:
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-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

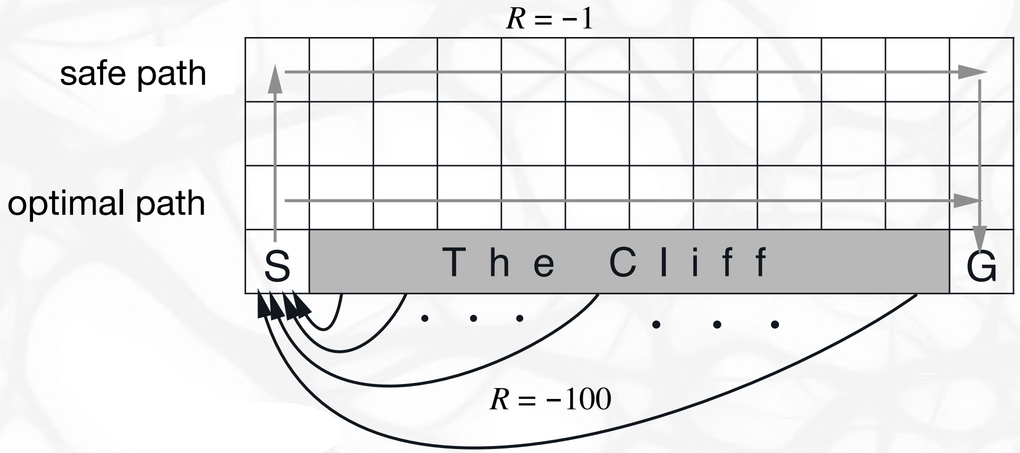
Q-learning is another TD control algorithm by (Watkins, 1989), defined by

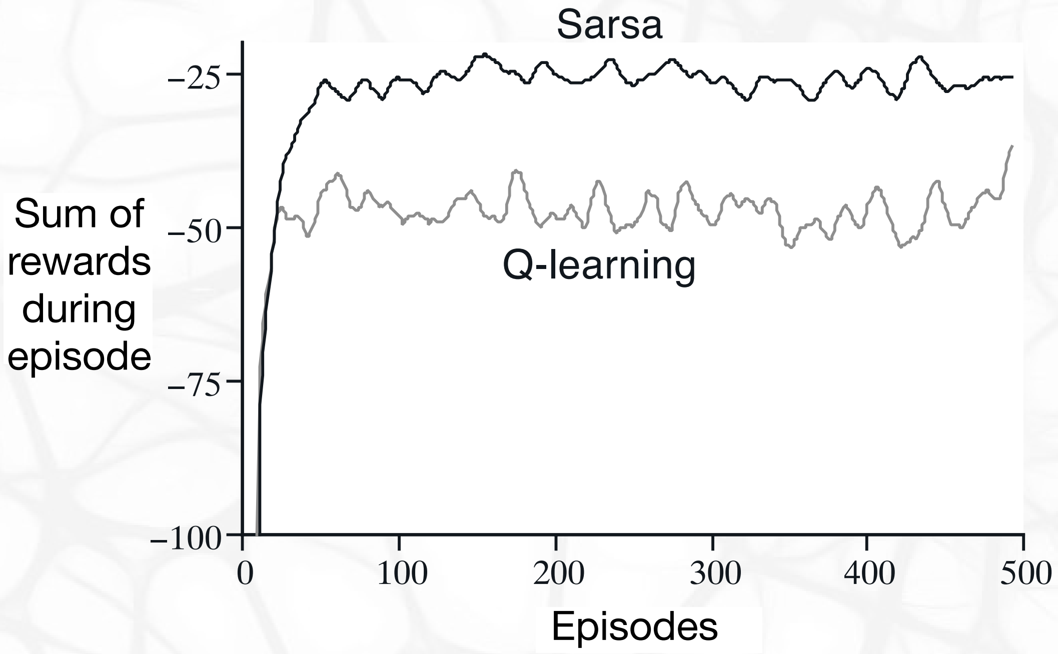
$$Q(S_t, A_T) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)].$$

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

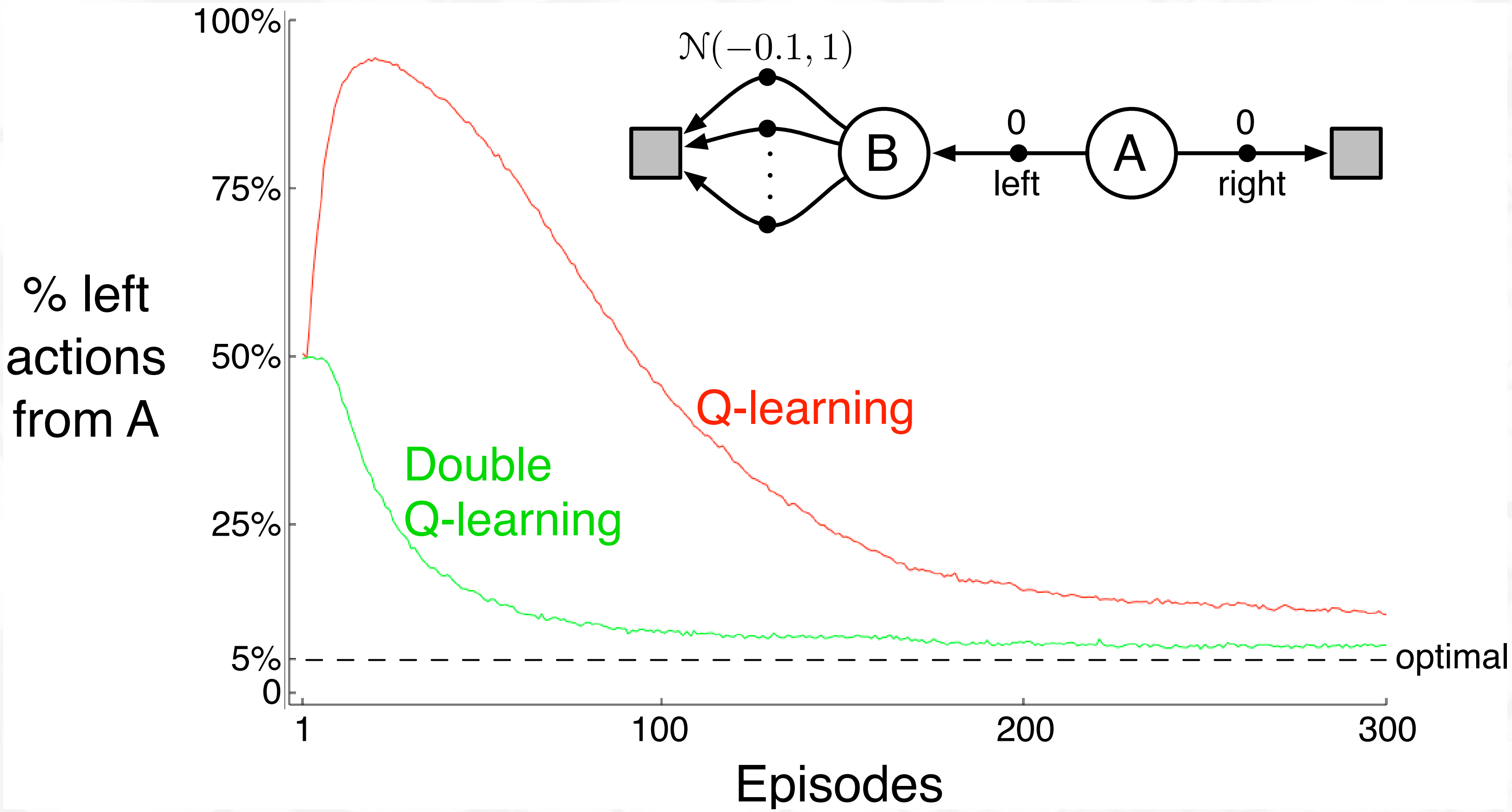
```
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-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 $\varepsilon$-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
```

Bridging Q-learning and Monte Carlo

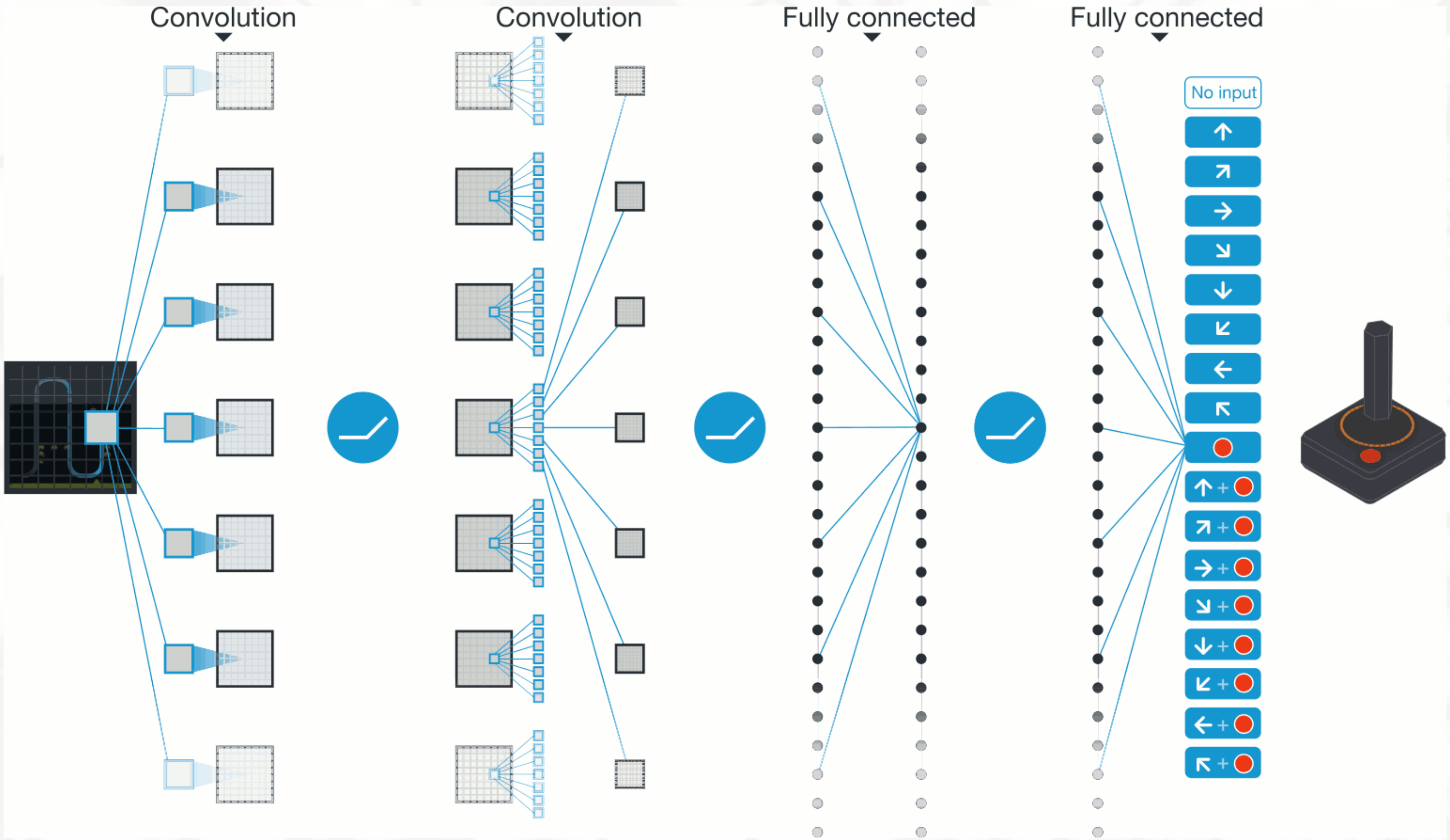
Monte Carlo uses whole episode returns, while Q-learning uses single-step rewards. We can connect these approaches by considering n-step returns:

$$\sum_{k=1}^{n} \gamma^{k-1} R_{t+k}.$$

We can then approximate full returns as

$$G_t \approx \sum_{k=1}^{n} (\gamma^{k-1} R_{t+k}) + v_\pi(S_{t+n}).$$

Deep Q Networks



Deep Q Networks

No proofs of convergence; the training can be extremely brittle. Several improvements to increase stability of the training:

Deep Q Networks

No proofs of convergence; the training can be extremely brittle. Several improvements to increase stability of the training:

- experience replay,

Deep Q Networks

No proofs of convergence; the training can be extremely brittle. Several improvements to increase stability of the training:

- experience replay,
- separate target network $\hat{Q}$,

Deep Q Networks

No proofs of convergence; the training can be extremely brittle. Several improvements to increase stability of the training:

- experience replay,
- separate target network $\hat{Q}$,
- clipping $R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)$ to $[-1, 1]$.

Policy Gradient Methods

The main idea of *policy gradient method* is to train the policy itself, instead of basing it on action-value function q.

However, for that we need to be able to compute a derivation of state-value function, i.e., $\nabla v_\pi(s)$.

Hopefully, a *policy gradient theorem* comes to the rescue.

Policy Gradient Methods

The main idea of *policy gradient method* is to train the policy itself, instead of basing it on action-value function q.

However, for that we need to be able to compute a derivation of state-value function, i.e., $\nabla v_\pi(s)$.

Hopefully, a *policy gradient theorem* comes to the rescue.

Policy Gradient Theorem

Let π be a given policy. We denote the on-policy distribution under π as $\mu(s)$. Then

$$\nabla v_\pi(s) \propto \sum_{s \in \mathcal{S}} \mu(s) \sum_{a \in \mathcal{A}} q_\pi(s, a) \nabla \pi(a|s; \boldsymbol{\theta}).$$

Policy Gradient Theorem

$$
\begin{aligned}
\nabla v_\pi(s) &= \nabla \Bigg[\sum_a \pi(a|s) q_\pi(s,a)\Bigg], \quad \text{for all } s \in \mathcal{S} && \text{(Exercise 3.16)} \\
&= \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \nabla q_\pi(s,a)\Big] && \text{(product rule of calculus)} \\
&= \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \nabla \sum_{s',r} p(s',r|s,a)\big(r + v_\pi(s')\big)\Big] \\
& && \text{(Exercise 3.17 and Equation 3.2)} \\
&= \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \sum_{s'} p(s'|s,a) \nabla v_\pi(s')\Big] && \text{(Eq. 3.4)} \\
&= \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \sum_{s'} p(s'|s,a) && \text{(unrolling)} \\
&\qquad \sum_{a'} \big[\nabla \pi(a'|s') q_\pi(s',a') + \pi(a'|s') \sum_{s''} p(s''|s',a') \nabla v_\pi(s'')\big]\Big] \\
&= \sum_{x \in \mathcal{S}} \sum_{k=0}^{\infty} \Pr(s \rightarrow x, k, \pi) \sum_a \nabla \pi(a|x) q_\pi(x,a),
\end{aligned}
$$

Finally, we obtain the required form by dividing the result by an average length of an episode.

REINFORCE Algorithm

The REINFORCE algorithm (Williams, 1992) uses directly the policy gradient theorem, approximating the expectation by a single sample.

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

```
Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol\theta)$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol\theta \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol\theta)$
    Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
        $G \leftarrow \sum_{k=t+1}^{T} R_k$                                        $(G_t)$
        $\boldsymbol\theta \leftarrow \boldsymbol\theta + \alpha G \nabla \ln \pi(A_t|S_t,\boldsymbol\theta)$
```

REINFORCE with baseline Algorithm

The gradient estimation used in REINFORCE has high variance.

However, we can decrease the variance by considering a *baseline* $b(s)$, which as an arbitrary function not depending on action a, using the following generalization of policy gradient theorem:

$$\nabla v_\pi(s) \propto \sum_{s\in\mathcal{S}} \mu(s) \sum_{a\in\mathcal{A}} \big(q_\pi(s,a) - \mathbf{b(s)}\big) \nabla \pi(a|s;\boldsymbol\theta).$$

The introduction of the baseline is possible, because

$$\sum_{a\in\mathcal{A}} b(s) \nabla\pi(a|s;\boldsymbol\theta) = b(s)\nabla\sum_{a\in\mathcal{A}}\pi(a|s;\boldsymbol\theta) = b(s)\nabla 1 = 0.$$

REINFORCE with baseline Algorithm

REINFORCE with Baseline (episodic), for estimating $\pi_{\boldsymbol\theta} \approx \pi_*$

```
Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol\theta)$
Input: a differentiable state-value function parameterization $\hat v(s,\mathbf{w})$
Algorithm parameters: step sizes $\alpha^{\boldsymbol\theta} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol\theta \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol\theta)$
    Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
        $G \leftarrow \sum_{k=t+1}^{T} R_k$                                        $(G_t)$
        $\delta \leftarrow G - \hat v(S_t,\mathbf{w})$
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat v(S_t,\mathbf{w})$
        $\boldsymbol\theta \leftarrow \boldsymbol\theta + \alpha^{\boldsymbol\theta} \delta \nabla \ln \pi(A_t|S_t, \boldsymbol\theta)$
```

Actor Critic

A combination of Q-learning and REINFORCE is also possible and called Actor Critic algorithm.

One-step Actor–Critic (episodic), for estimating $\pi_{\boldsymbol\theta} \approx \pi_*$

```
Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol\theta)$
Input: a differentiable state-value function parameterization $\hat v(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol\theta} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol\theta \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
    Initialize $S$ (first state of episode)

    Loop while $S$ is not terminal (for each time step):
        $A \sim \pi(\cdot|S,\boldsymbol\theta)$
        Take action $A$, observe $S', R$
        $\delta \leftarrow R + \gamma \hat v(S',\mathbf{w}) - \hat v(S,\mathbf{w})$        (if $S'$ is terminal, then $\hat v(S',\mathbf{w}) \doteq 0$)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \ \delta \nabla \hat v(S,\mathbf{w})$
        $\boldsymbol\theta \leftarrow \boldsymbol\theta + \alpha^{\boldsymbol\theta} \ \delta \nabla \ln \pi(A|S,\boldsymbol\theta)$
        $S \leftarrow S'$
```