COMPSCI 389
Introduction to Machine Learning

Value functions, Temporal Difference Learning, and Actor-Critics

Prof. Philip S. Thomas (pthomas@cs.umass.edu)

REINFORCE (Review)

Note: The actual REINFORCE algorithm sums up the changes to θ_i from the whole episode and then makes the changes. The pseudocode below changes each θ_i at time $t = 0$, and that change influences the derivative computed at subsequent times.

Algorithm 17.2: REINFORCE

for *each episode* **do**
  // Run one episode (play one game).
  for *each time t in the episode* **do**
    Agent observes state S_t;
    Agent selects action A_t according to the current policy, π_θ;
    Environment responds by transitioning from state S_t to state S_{t+1} and emitting reward R_t;
  end
  // Learn from the outcome of the episode.
  for *each time t in the episode* **do**
    $\forall i,\ \theta_i \leftarrow \theta_i + \alpha\gamma^t \left(\sum_{k=0}^{\infty} \gamma^k R_{t+k}\right) \frac{\partial \ln(\pi_\theta(S_t, A_t))}{\partial \theta_i}$;
  end
end

Two-Phases

- REINFORCE has two phases:
  - Run an episode, selecting actions and observing the outcome.
  - Update the policy.
- Waiting until the end of an episode to update the policy seems inefficient.

Algorithm 17.2: REINFORCE

---

for *each episode* **do**
  // Run one episode (play one game).
  for *each time t in the episode* **do**
    Agent observes state S_t;
    Agent selects action A_t according to the current policy, π_θ;
    Environment responds by transitioning from state S_t to state S_{t+1} and emitting reward R_t;
  end
  // Learn from the outcome of the episode.
  for *each time t in the episode* **do**
    $\forall i,\ \theta_i \leftarrow \theta_i + \alpha\gamma^t \left(\sum_{k=0}^{\infty} \gamma^k R_{t+k}\right) \frac{\partial \ln(\pi_\theta(S_t, A_t))}{\partial \theta_i}$;
  end
end

Idea: Consider expected future rewards

- Imagine that you play the lottery and learn that you have won.
- In the future, you will obtain the prize money, which may result in actual rewards (in the form of increased comfort and pleasure).
- However, you will likely be happy and celebrate, perhaps learning that you should play the lottery more, *before* you collect any prize money or see the actual impact that it has on your life!
- This isn’t due to any rewards – you have received no actual rewards yet.
  - This learning (change in behavior) is due to your expectations of future rewards.

Idea: Consider expected future rewards

- When you realize that you have won the lottery, you recognize that the expected outcome was better than you were previously expecting.
  - This is sufficient for you to learn!

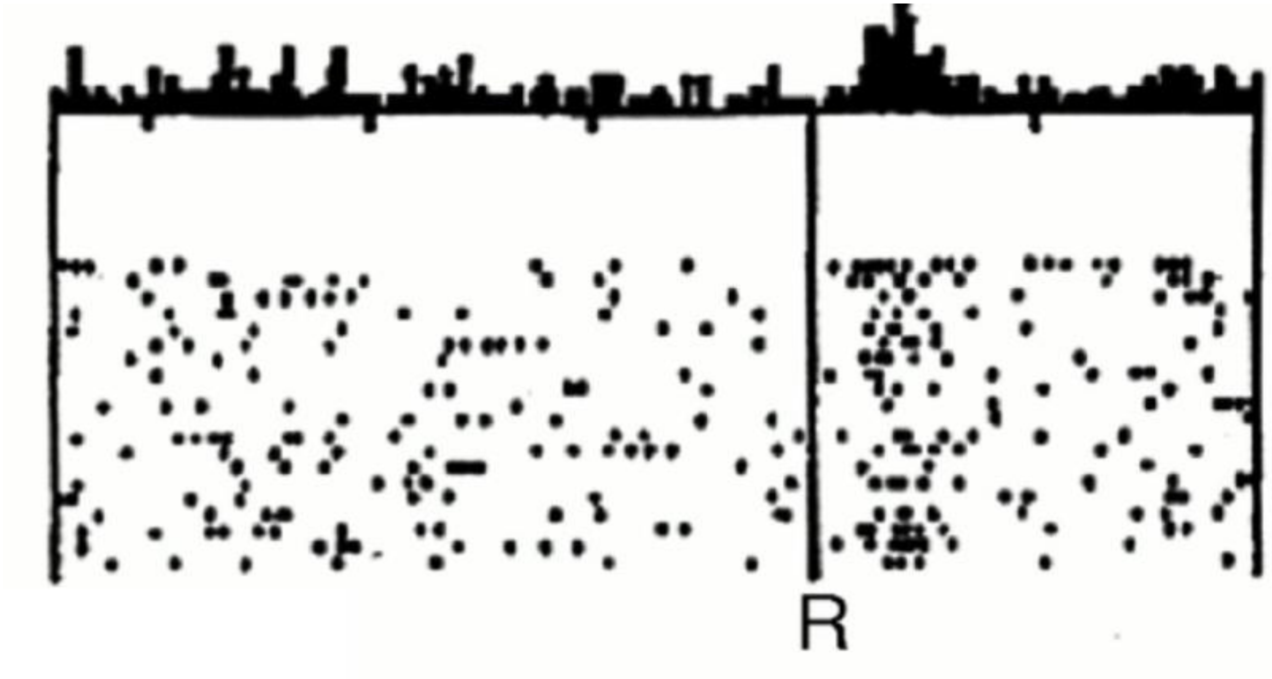
Idea: Consider expected future rewards

- Consider another example: a monkey

Firing of neurons that indicate “take recent actions more often.”

(Details later!)

Time

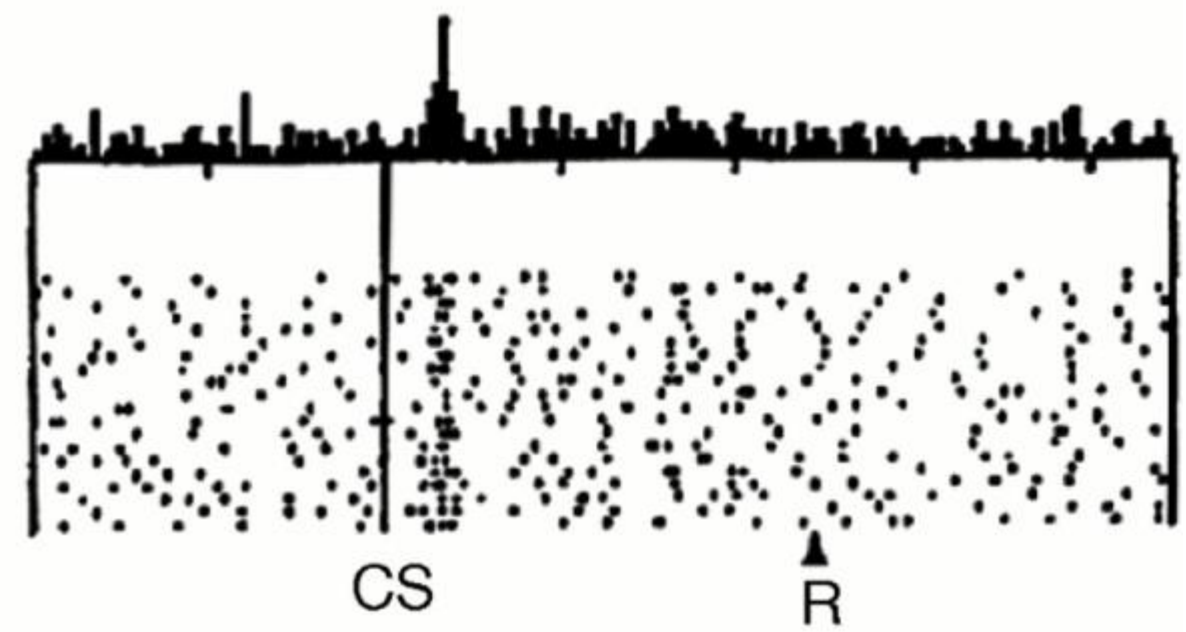


Apple Juice

Idea: Consider expected future rewards

- Consider another example: a monkey

Time



Firing of neurons that indicate "take recent actions more often."

(Details later!)

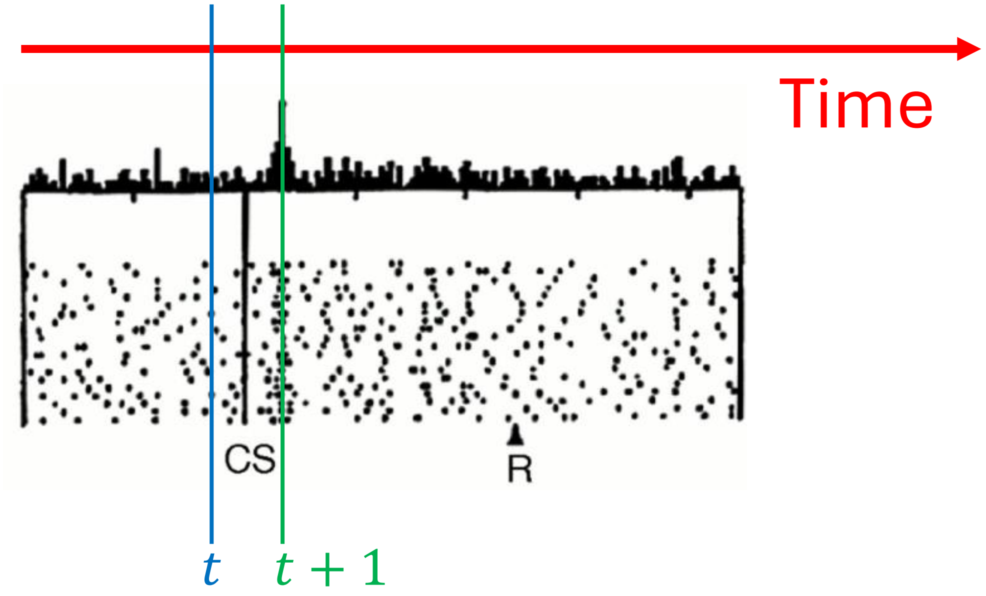
Seeing a light flash isn't a reward, but it tells the agent (monkey) that it is likely to receive a reward (apple juice) in the future.

Light

Apple Juice

Idea: Consider expected future rewards

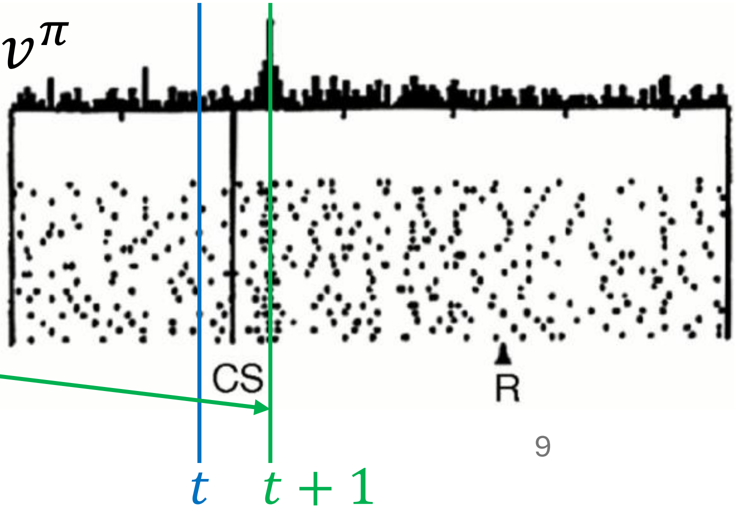
- **Before**: Make actions more likely when they result in an observed desirable outcome
- **New idea**: Make actions more likely when they cause the agent to believe that the outcome will be more desirable than expected.



Idea: Consider expected future rewards

- **Before**: Make actions more likely when they result in an observed desirable outcome
- **New idea**: Make actions more likely when they cause the agent to believe that the outcome will be more desirable than expected.
- To do this, the agent must have a notion of how much reward it expects to get in the future.
  - This is captured by a function called a **value function** v^π

To know that recent actions should be reinforced, we must know that seeing the light means that more reward is expected in the future.



Value Function v^π

- A function that depends on the policy π
  - **Input**: Any state s
  - **Output**: The expected discounted sum of rewards that the agent will receive in the future if it were in state s:

$$v^\pi(s) = \mathbf{E}\left[\sum_{k=0}^{\infty} \gamma^t R_{t+k} \middle| S_t = s; \pi\right].$$

- The right-hand side has t, but t doesn't show up on the left side!
  - Due to the **Markov property**, the value of $v^\pi(s)$ is the same for all t on the right-hand side.

Value Function v^{π}

- Consider a gridworld
  - Deterministic transitions
  - The agent always starts in the top-left.
  - The goal (a terminal state) is the bottom-right.
  - The reward is 0 at each time step, except when the agent **enters** the goal state, at which time it receives a reward of 100.
  - Let $\gamma = 0.5$
- Let π be an optimal policy. The value function is then→

				?

Value Function v^{π}

- Consider a gridworld
  - Deterministic transitions
  - The agent always starts in the top-left.
  - The goal (a terminal state) is the bottom-right.
  - The reward is 0 at each time step, except when the agent **enters** the goal state, at which time it receives a reward of 100.
  - Let $\gamma = 0.5$
- Let π be an optimal policy. The value function is then→

				?
				0

Value Function v^{π}

- Consider a gridworld
  - Deterministic transitions
  - The agent always starts in the top-left.
  - The goal (a terminal state) is the bottom-right.
  - The reward is 0 at each time step, except when the agent **enters** the goal state, at which time it receives a reward of 100.
  - Let $\gamma = 0.5$
- Let π be an optimal policy. The value function is then→

				?
				100
				0

Value Function v^{π}

- Consider a gridworld
  - Deterministic transitions
  - The agent always starts in the top-left.
  - The goal (a terminal state) is the bottom-right.
  - The reward is 0 at each time step, except when the agent **enters** the goal state, at which time it receives a reward of 100.
  - Let $\gamma = 0.5$
- Let π be an optimal policy. The value function is then→

				50
				100
				0

Value Function v^{π}

- Consider a gridworld
  - Deterministic transitions
  - The agent always starts in the top-left.
  - The goal (a terminal state) is the bottom-right.
  - The reward is 0 at each time step, except when the agent **enters** the goal state, at which time it receives a reward of 100.
  - Let $\gamma = 0.5$
- Let π be an optimal policy. The value function is then→

≈ 0.78	1.5625	3.125	6.25	12.5
1.5625	3.125	6.25	12.5	25
3.125	6.25	12.5	25	50
6.25	12.5	25	50	100
12.5	25	50	100	0

Value Function v^{π}

So, different policies result in different value functions!

- Consider a gridworld
  - Deterministic transitions
  - The agent always starts in the top-left.
  - The goal (a terminal state) is the bottom-right.
  - The reward is 0 at each time step, except when the **enters** the goal state, at which time it receives a reward of 100.
  - Let $\gamma = 0.5$
- Let π be the policy that always selects the down action. The value function is then→

0	0	0	0	12.5
0	0	0	0	25
0	0	0	0	50
0	0	0	0	100
0	0	0	0	0

Temporal Difference Error

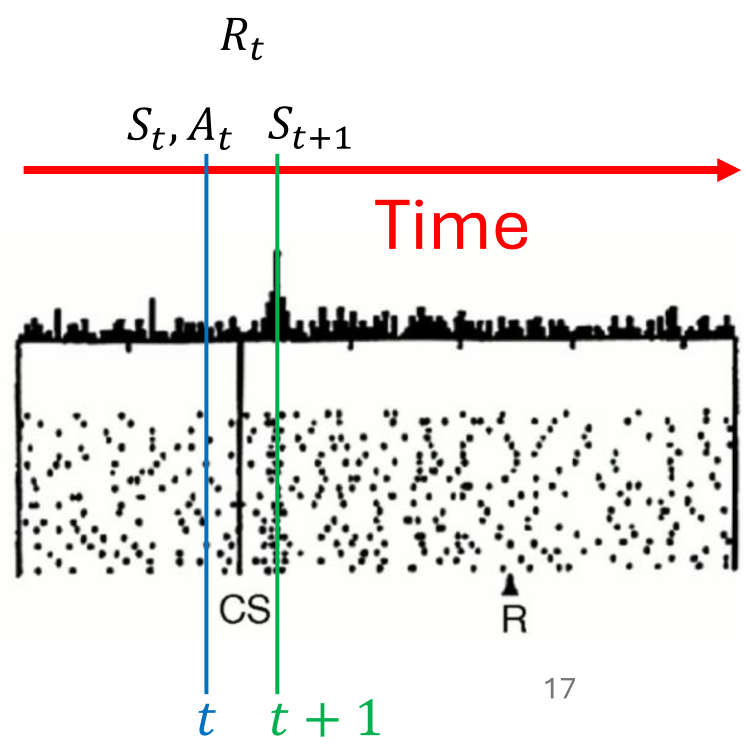
- Later we will discuss how the agent can learn (estimate) v^π from its experiences.
- First, let's explore how the agent can *use* v^π (or an estimate of v^π).
- Consider how an agent can update its policy when:
  - The agent observes S_t
  - The agent selects A_t
  - The environment transitions to S_{t+1} and emits reward R_t

$v^\pi(S_t)$: Expected total future reward

R_t: Reward received

$v^\pi(S_{t+1})$: Expected total future reward at next time

$$\underline{\qquad\qquad} \approx \underline{R_t} + \underline{\qquad\qquad}$$



Temporal Difference Error

- Later we will discuss how the agent can learn (estimate) v^π from its experiences.
- First, let's explore how the agent can *use* v^π (or an estimate of v^π).
- Consider how an agent can update its policy when:
  - The agent observes S_t
  - The agent selects A_t
  - The environment transitions to S_{t+1} and emits reward R_t

$v^\pi(S_t)$: Expected total future reward

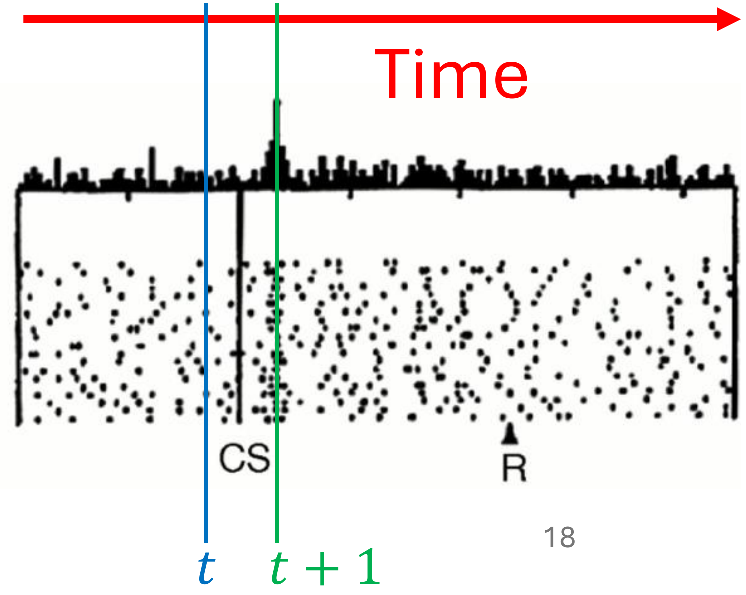
R_t: Reward received

$v^\pi(S_{t+1})$: Expected total future reward at next time

$$v^\pi(S_t) \approx R_t + v^\pi(S_{t+1})$$

R_t

S_t, A_t $\quad S_{t+1}$



Temporal Difference Error

- So, we expect that $v^{\pi}(S_t) \approx R_t + v^{\pi}(S_{t+1})$
  - Note: With reward discounting, $v^{\pi}(S_t) \approx R_t + \gamma v^{\pi}(S_{t+1})$
- **Question**: What does it mean if $R_t + \gamma v^{\pi}(S_{t+1}) > v^{\pi}(S_t)$?

This can happen when the reward is bigger than expected

This can happen when both

This can happen when the next state is better than expected

Temporal Difference Error

- So, we expect that $v^{\pi}(S_t) \approx R_t + v^{\pi}(S_{t+1})$
  - Note: With reward discounting, $v^{\pi}(S_t) \approx R_t + \gamma v^{\pi}(S_{t+1})$
- **Question**: What does it mean if $R_t + \gamma v^{\pi}(S_{t+1}) > v^{\pi}(S_t)$?
- **Answer**: The outcome of A_t was better than expected.
  - Make A_t *more* likely in state S_t
- **Question**: What does it mean if $v^{\pi}(S_t) > R_t + \gamma v^{\pi}(S_{t+1})$?
- **Answer**: The outcome of A_t was *worse* than expected.
  - Make A_t *less* likely in state S_t

Temporal Difference Error

- Let the **temporal difference error** or **TD error** δ_t be:

$$\delta_t = R_t + \gamma v^\pi(S_{t+1}) - v^\pi(S_t)$$

- When the TD error is positive, $R_t + \gamma v^\pi(S_{t+1}) > v^\pi(S_t)$
  - The agent should select A_t more often in S_t.
  - The magnitude of the TD error indicates how much better the outcome of A_t was, and can be used to scale how much more likely A_t is made.
- When the TD error is negative, $v^\pi(S_t) > R_t + \gamma v^\pi(S_{t+1})$
  - The agent should select A_t less often in S_t.
  - The magnitude [...]
- Idea: Replace the weight, $\sum_{t'=t}^{\infty} \gamma^{t'} R_{t'} = \gamma^t \sum_{k=0}^{\infty} \gamma^k R_{t+k}$ with δ_t.

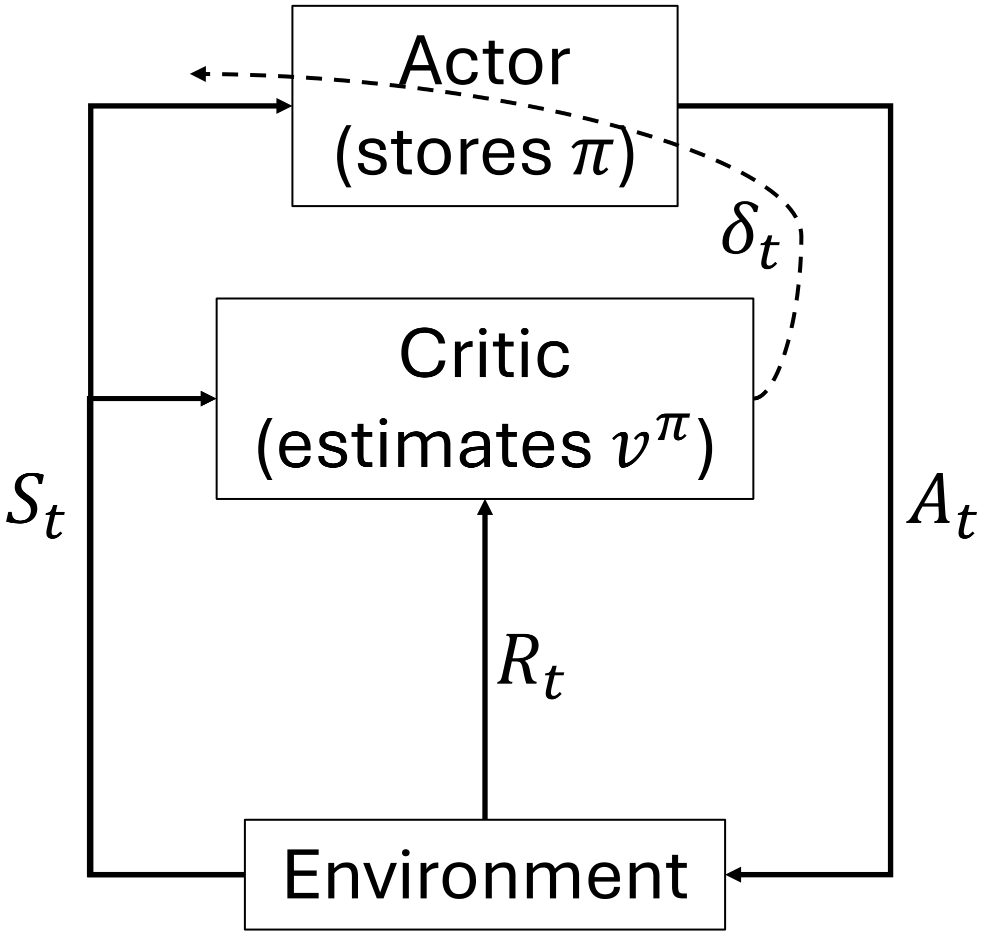
Algorithm 17.3: A simple RL algorithm inspired by MENACE, Version 5.0

for *each episode* **do**
    // Run one episode (play one game).
    for *each time t in the episode* **do**
        // Execute one time step of agent-environment interaction
        Agent observes state S_t;
        Agent selects action A_t according to the current policy, π_θ;
        Environment responds by transitioning from state S_t to state S_{t+1} and emitting reward R_t;
        // Learn from the outcome of this one time step
        $\delta_t \leftarrow R_t + \gamma v^\pi(S_{t+1}) - v^\pi(S_t)$;
        $\forall i,\ \theta_i \leftarrow \theta_i + \alpha \delta_t \frac{\partial \pi_\theta(S_t, A_t)}{\partial \theta_i}$;
    end
end

Note: We have replaced the discounted sum of rewards after A_t with the TD error. This allows us to perform policy updates *before* the episode ends!

Actor-Critic

$$\delta_t \leftarrow R_t + \gamma v^\pi(S_{t+1}) - v^\pi(S_t);$$
$$\forall i,\ \theta_i \leftarrow \theta_i + \alpha \delta_t \frac{\partial \pi_\theta(S_t, A_t)}{\partial \theta_i};$$



Actor-Critics

- Actor-critic algorithms are a class of algorithms, not one specific algorithm.
  - They have an **actor** (which stores the current policy) and a **critic** (which stores an estimate of a value function).
- Often actor-critic algorithms are **policy gradient** algorithms like REINFORCE: they change the policy following **estimates** of the gradient of

$$J(\theta) = \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t\right]$$

  - Usually these estimates are **biased** (not only do they have variance, but even with infinite data they point in the “wrong” direction!)
  - Still, these biased estimates can be quite effective!
    - State of the art algorithms like PPO and SAC are actor-critics / policy gradient methods.

Approximating the value function

- Let v_w be a parametric function approximator (like a model in supervised learning) with weights w.
- Our aim is to make $v_w(s) \approx v^\pi(s) = \mathbf{E}[\sum_{k=0}^{\infty} \gamma^k R_t \mid S_t = s]$.
- Recall: $\delta_t = R_t + \gamma v^\pi(S_{t+1}) - v^\pi(S_t)$
- We don't know v^π, so let's re-define the TD error to use the approximation:

$$\delta_t = R_t + \gamma v_w(S_{t+1}) - v_w(S_t).$$

Learning the value function

- Recall: $\delta_t = R_t + \gamma v_w(S_{t+1}) - v_w(S_t)$.
- **Question**: If $\delta_t > 0$, what does that say about $v_w(S_t)$?
- **Answer**: It should be increased!
- **Question**: If $\delta_t < 0$, what does that say about $v_w(S_t)$?
- **Answer**: It should be decreased!
- **Question**: How do we change w to increase $v_w(S_t)$?
- **Answer**: $w \leftarrow w + \alpha \frac{\partial v_w(S_t)}{\partial w}$
- **Update**: $w \leftarrow w + \alpha \delta_t \frac{\partial v_w(S_t)}{\partial w}$

Algorithm 18.2: An Actor-Critic Algorithm

for *each episode* **do**

    // Run one episode (play one game).

    for *each time t in the episode* **do**

        // Execute one time step of agent-environment interaction

        Agent observes state S_t;

        Agent selects action A_t according to the current policy, π_θ;

        Environment responds by transitioning from state S_t to state S_{t+1} and emitting reward R_t;

        // Learn from the outcome of this one time step

        $\delta_t \leftarrow R_t + \gamma v_w(S_{t+1}) - v_w(S_t)$;

        $\forall i,\ \theta_i \leftarrow \theta_i + \alpha \delta_t \frac{\partial \pi_\theta(S_t, A_t)}{\partial \theta_i}$ // Actor update

        $\forall j,\ w_j \leftarrow w_j + \beta \delta_t \frac{\partial v_w(S_t)}{\partial w_j}$ // Critic update

    end

end

Compute the TD error, δ_t.

Update the actor's policy. Optionally, use $\partial \ln(\pi_\theta(S_t, A_t))$.

Update the critic's value function estimate. Optionally, use a step size β that is different from that of the actor (α).

End