8.
Deep Q-Learning Variants

Fixed Q-value Targets

- In the basic deep Q-learning algorithm, the model is used both to make predictions and to set its own targets.
  - This feedback loop can destabilize the network, causing it to diverge, oscillate, or freeze.
- DeepMind researchers used two DQNs instead of one:
  - *online model*: learns at each step and is used to move the agent around.
  - *target model*: used only to define the targets and is a clone of online model.
- In the training loop, we periodically copy the online model's weights to the target model (e.g., every 50 episodes).
- Since the target model updates less frequently than the online model, Q-value targets remain more stable, damping the feedback loop and reducing its negative effects.

Double DQN

- If all actions are equally good, the target model's Q-values should be identical, but due to approximation errors cause some to be slightly higher by chance.

- The target model always selects the largest Q-value, which will be slightly greater than the mean Q-value, resulting in overestimation:

$$Q_{\text{target}}(s,a) \underset{\alpha}{\leftarrow} r + \gamma . \max_{a\prime} Q_{\text{target}}(s', a')$$

- To address this, researchers proposed using the online model to select the best actions for the next states, and the target model to estimate the Q-values of those actions:

$$Q_{\text{target}}(s,a) \underset{\alpha}{\leftarrow} r + \gamma . Q_{\text{target}}(s', \arg\max_{a'} Q_{\text{online}}(s', a'))$$

Prioritized Experience Replay

- *Importance sampling* (IS) or *prioritized experience replay* (PER): instead of sampling experiences *uniformly* from the replay buffer, sample important experiences more frequently.
- Experiences are considered "important" if they are likely to lead to fast learning progress.
  - How can we estimate this?
- One reasonable approach: use the TD error $\delta = r + \gamma . V(s') - V(s)$.
- A large TD error indicates that a transition (s, a, s') is very surprising, and thus probably worth learning from.
- The probability of sampling an experience is proportional to $|\delta|^{\zeta}$, where ζ controls the greediness of importance sampling.

Dueling DQN

- The Q-value of a state-action pair (s, a) can be expressed as $Q(s, a) = V(s) + A(s, a)$, where $V(s)$ is the value of state s and $A(s, a)$ is the *advantage* of action a over others in state s.
- The state value equals the Q-value of the best action a^* for that state, so $V(s) = Q(s, a^*)$, implying $A(s, a^*) = 0$.
- In dueling DQN, the model separately estimates $V(s)$ and $A(s, a)$.
  - The best action's advantage is zero, so the model subtracts the maximum predicted advantage from all advantages:

$$Q(s, a) = V(s) + \left(A(s, a) - \max_{a'} A(s, a')\right)$$

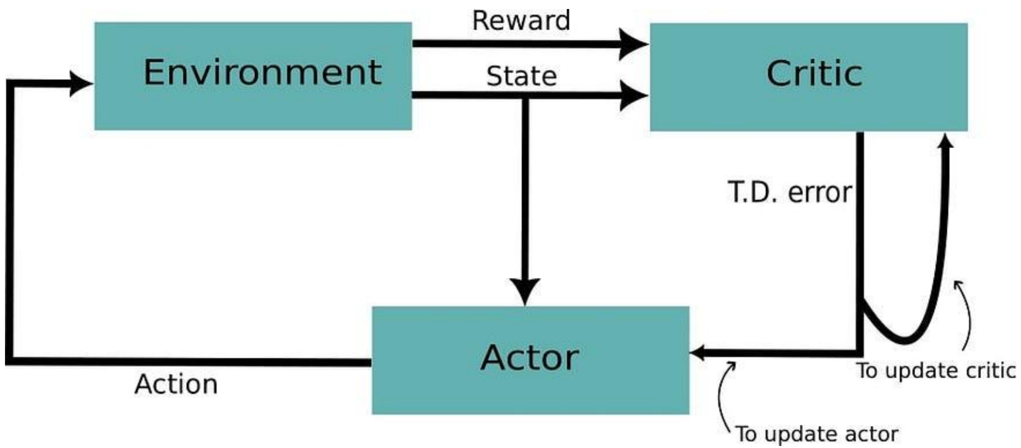
  - To achieve more stable training use average advantage aggregation:

$$Q(s, a) = V(s) + \left(A(s, a) - \frac{1}{|\mathcal{A}|}\sum_{a' \in \mathcal{A}} A(s, a')\right)$$

9.
Actor-Critic Algorithms

Actor-critic algorithms

- Actor-critic algorithms combine the strengths of policy gradient and value-based methods by using two neural networks:
  - The actor network directly learns a policy: a mapping from states to actions (or action probabilities).
  - The critic network estimates the value function (e.g., $V(s)$ or $Q(s,a)$), and is trained using temporal-difference learning.



A3C and A2C

- *Asynchronous advantage actor-critic* (A3C) is a RL algorithm where multiple agents learn in parallel, exploring different and independent copies of the same environment.
  - At regular but asynchronous intervals, each agent pushes weight updates to a master network, then pulls the latest weights from that network.
  - The critic estimates the value of each state, and the advantage of an action is computed by subtracting this value from the observed return.
  - A policy gradient update is then applied using the advantage by the actor.
- *Advantage actor-critic (A2C)* is a synchronous variant of the A3C algorithm.
  - Model updates are synchronous, so gradient updates are performed over larger batches, allowing the model to better utilize the power of the GPU.

Soft Actor-Critic

- Soft Actor-Critic (SAC) is an off-policy algorithm that optimizes a trade-off between maximizing expected rewards and maximizing the entropy of the policy.
- It encourages the agent to act as unpredictably (or randomly) as possible while still achieving high rewards.
  - This promotes better exploration of the environment, which can speed up training and helps prevent the policy from prematurely converging to suboptimal actions, especially when value estimates are imperfect.
- Due to this balance, SAC has demonstrated remarkable sample efficiency, often learning much faster than all the previous RL algorithms.

Proximal Policy Optimization

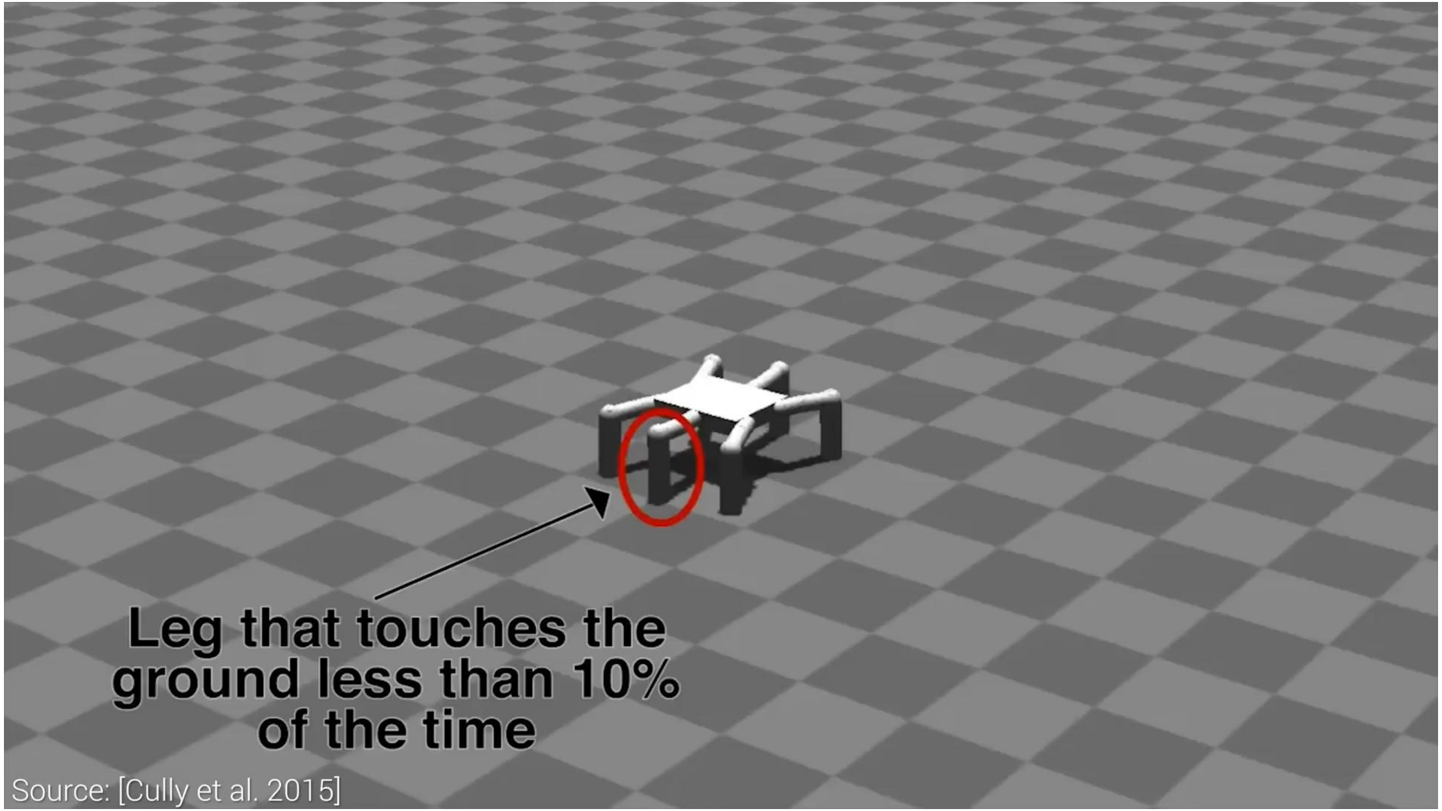
- Proximal Policy Optimization (PPO), developed by OpenAI, is based on the A2C but uses a clipped surrogate loss to prevent overly large policy updates that cause instability.
- PPO is a simpler and more scalable variant of the *trust region policy optimization* (TRPO), that retains strong performance.
- Notably, OpenAI Five, which used PPO, defeated the world champions in the multiplayer game Dota 2 in 2019.
- Many variants of PPO have emerged to improve its efficiency.
  - DeepSeek R1 utilizes *Group Relative Policy Optimization* (GRPO), an RL algorithm built upon PPO, which eliminates the need for a separate value function model, reducing memory usage and computational overhead by approximately 50%.

10.
Reinforcement Learning Challenges

RL Challenges

- **Training Instability**
  - High sensitivity to initial conditions and hyperparameters.
  - Learning targets shift over time (non-stationarity), leading to divergence.
- **Sample Inefficiency**
  - Requires vast amounts of data and interactions to learn.
  - Not practical in real-world settings without simulators.
- **Scalability**
  - Struggles with large or continuous state/action spaces.
  - Training deep RL models requires extensive resources and tuning.
- **Reward Hacking**
  - Agents exploit poorly designed reward signals.
  - E.g. racing agent loops endlessly around checkpoints to maximize points.

Reward Hacking Example



RL Challenges

- **Exploration vs. Exploitation**
  - Balancing trying new strategies vs. leveraging what’s known is hard.
  - Poor exploration often leads to premature convergence.
- **Multi-Agent Environments**
  - Other learning agents cause environment dynamics to shift constantly.
  - Adds complexity in coordination, communication, and competition.
- **Reality Gap (Sim-to-Real Transfer)**
  - Policies trained in simulation often fail when deployed in the real world.
  - Due to unmodeled dynamics, noise, or hardware constraints.
- **Sparse and Delayed Rewards**
  - Hard to attribute delayed rewards to specific actions.
  - Learning becomes inefficient or fails altogether.

Curiosity-based Exploration

- A useful approach to tackle the challenge of sparse rewards in RL is curiosity-based learning.
  - Why not ignore external rewards and instead make the agent intrinsically curious about exploring its environment?
- The agent continuously tries to predict the outcome of its actions, and it seeks situations where the outcome differs from its predictions.
  - If the outcome is predictable (boring), it moves on to explore elsewhere.
  - If the outcome is unpredictable but the agent notices that it has no control over it, the agent loses interest and stops focusing there.
- Using only this curiosity-driven signal, researchers have successfully trained agents to play many video games.
  - Even though the agent gets no penalty for losing, the game resets after losing, which is boring so it learns to avoid it and to survive longer.