DEEP REINFORCEMENT LEARNING

Ahmad A. Ismail Ghazala
Assistant Lecturer,
Faculty of computers and Artificial Intelligence,
Cairo University

Agenda

- What is Reinforcement Learning (RL)?
- Markov Decision Processes (MDP)
- RL Classifications
- Model Based
- Model Free
- Deep Reinforcement Learning (DRL)
- Decision Transformer (DT)

What is reinforcement learning (RL)?

- RL is a Semi-supervised machine-learning approach
- Concerned with learning control laws and policies to interact with a complex environment from experience
- The RL agent goes through many trial-and-error steps to maximize the reward it gets from the environment

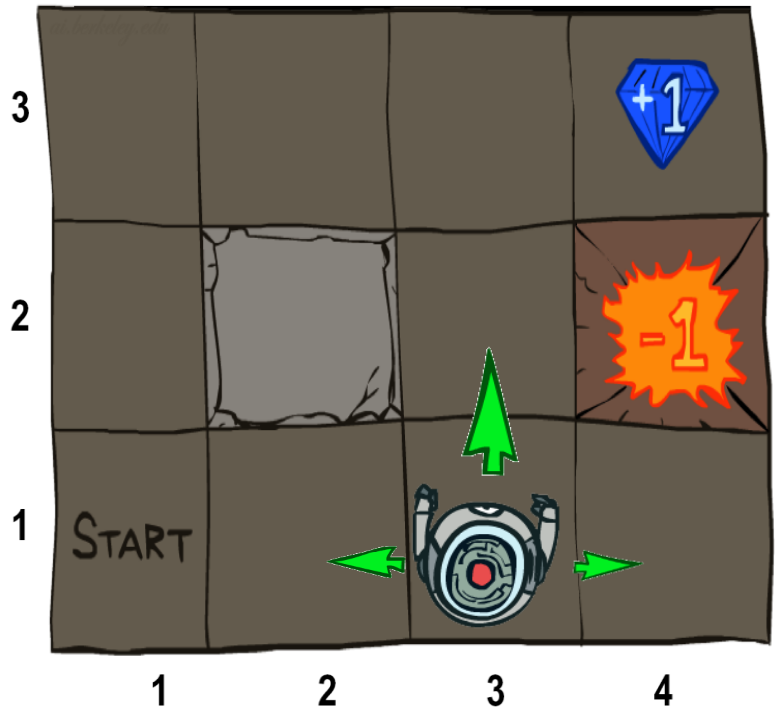


Fig. 1 Interaction of agent and environment.

Reinforcement Learning (RL)

- RL often begins with an unstructured *exploration*, where trial and error are used to learn the rules
- Followed by *exploitation*, where a strategy is chosen and optimized within the learned rules [1]
- RL is a Markov decision process (MDP)

Markov Decision Processes (MDP)

- An MDP is defined by:
  - A set of states $\mathcal{S}$
  - A set of actions $\mathcal{A}$
  - State-transition probabilities called the model dynamics $P(s'|s,a)$
  - A reward function $R(s,a,s')$

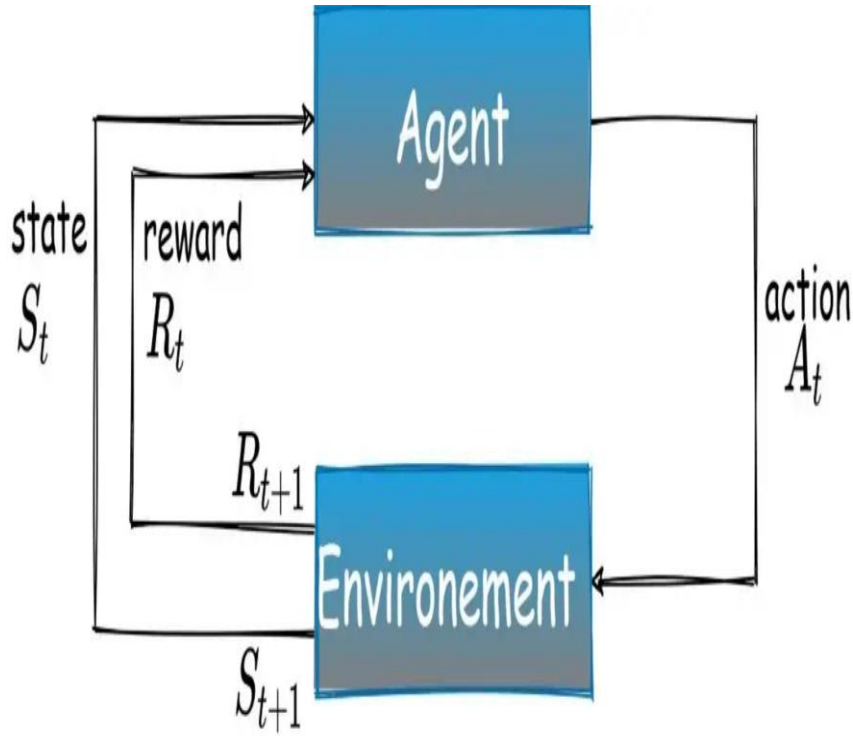


Fig. 2 Reinforcement Learning.

Policy

- Policy: is the strategy by which an agent takes an action in a state

$$\pi(s,a) = P(A=a|S=s)$$

- RL is often formulated as an optimization problem to learn the policy π (s, a)
- The policy may be a look-up table that is defined on the discrete state and action
- However, for most problems, representing and learning this policy becomes prohibitively expensive, and π must instead be represented as an approximate function that is parameterized by a lower-dimensional vector θ:

$$\pi(s,a) \approx \pi(s,a,\theta)$$

Value Function

- The value function is the expected return starting from state s, and then following policy π

$$V_\pi(s) = E[Gt|St = s]$$

$$V_\pi(s) = E(\sum_k \gamma^k r_k|S = s)$$

$$V(s) = max_\theta E(\sum_k \gamma^k r_k|S = s)$$

- The value at a state s may be written *recursively* as

$$V(s) = max_\theta E(r_0 + \sum_k \gamma^k r_k|S1 = s_{k+1})$$

$$V(s) = max_\theta E\big(r_0 + \gamma V(s_{k+1})\big)$$

- This expression, known as *Bellman's equation*

Quality function

- The action-value function $Q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π

$$Q_\pi(s, a) = E_\pi[Gt|St = s, At = a]$$

- Both policy iteration and value iteration above rely on the quality function Q(s, a), which describes the joint desirability of a given state/action pair

RL Classifications

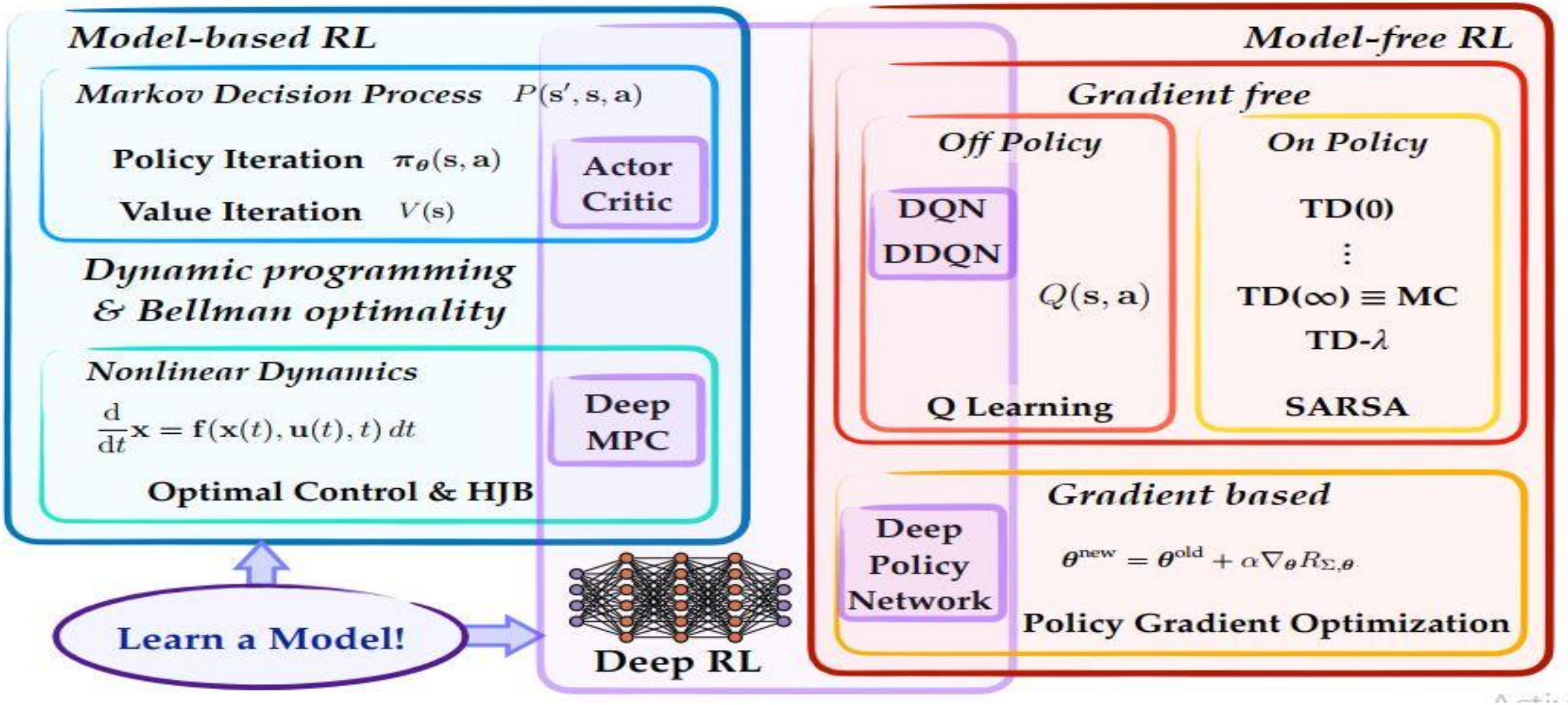


Fig. 3: RL Classifications [1].

RL Techniques

1. *Model-based RL:* if the complete model is known, the optimal solution can be found using Dynamic Programming.

2. *Model Free RL:* The agent learns a strategy to interact with the environment without any knowledge of the model and does not try to learn about the model's environment.

Model-based RL

- Learn either the optimal policy or value function through iterations
- It is using the *Bellman equation* which can be done using *dynamic programming*
- Dynamic programming reformulates the large optimization problem as *a recursive optimization* in terms of *smaller sub-problems* so that only a local decision need to be optimized.
- Dynamic programming is closely related to divide-and-conquer techniques
- There are two main approaches to dynamic programming, referred to as *top-down* and *bottom-up*

Planning by Dynamic Programming

- Dynamic programming assumes full knowledge of the MDP
- It is used for planning in an MDP
- For prediction:
  - Input: MDP S,A,P,R,γ and policy π
  - Output: value function $v\pi$
- Or for control:
  - Input: MDP S,A,P,R,γ
  - Output: optimal value function $v*$ and: optimal policy $\pi*$

Policy Iteration

- Policy iteration [2] is a two-step optimization procedure to simultaneously find an optimal value function vπ and the corresponding optimal policy π.
- Policy evaluation Estimate vπ
  e.g., Iterative policy evaluation
- Policy improvement Generate π0 ≥ π
  e.g., Greedy policy improvement

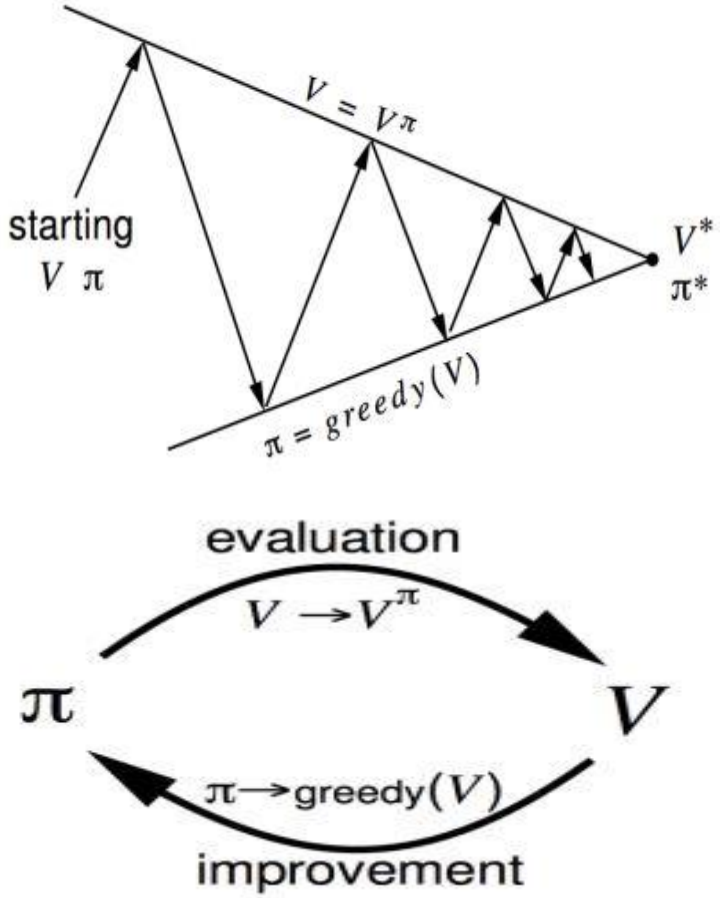


Fig. 4: Policy Iteration [2].

Value Iteration

- Similar to policy iteration, except that at every iteration only the value function is updated
- The optimal policy is extracted from this value function at the end.

$$V(\mathbf{s}) = \max_{\mathbf{a}} \sum_{\mathbf{s}'} P(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}) \left(R(\mathbf{s}', \mathbf{s}, \mathbf{a}) + \gamma V(\mathbf{s}') \right)$$

$$\boldsymbol{\pi}(\mathbf{s}, \mathbf{a}) = \operatorname{argmax}_{\mathbf{a}} \sum_{\mathbf{s}'} P(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}) \left(R(\mathbf{s}', \mathbf{s}, \mathbf{a}) + \gamma V(\mathbf{s}') \right)$$

Quality function

- Both policy iteration and value iteration rely on the quality function Q(s, a), which is defined as

$$\begin{aligned} Q(\mathbf{s}, \mathbf{a}) &= \mathbb{E}\left(R(\mathbf{s}', \mathbf{s}, \mathbf{a}) + \gamma V(\mathbf{s}')\right) \\ &= \sum_{\mathbf{s}'} P\left(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}\right)\left(R(\mathbf{s}', \mathbf{s}, \mathbf{a}) + \gamma V(\mathbf{s}')\right). \end{aligned}$$

- the optimal policy π(s, a) and the optimal value function V (s, a) contain redundant information, as one can be determined from the other via the quality function Q(s, a)

$$\boldsymbol{\pi}(\mathbf{s}, \mathbf{a}) = \underset{\mathbf{a}}{\operatorname{argmax}}\, Q(\mathbf{s}, \mathbf{a})$$

$$V(\mathbf{s}) = \max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a}).$$

Credit Assignment Problem (CAP)

- CAP discussed in Steps Toward Artificial Intelligence by Marvin Minsky (1961)
- *It is the problem of determining the actions that lead to a certain outcome.*
- An action that leads to a higher final cumulative reward should have more "credit" than an action that leads to a lower final reward.
- Most RL agents attempt to solve the CAP.
  - E.g., a Q -learning agent attempts to learn an optimal value function by determining the actions leading to the highest value in each state.

Model Free RL

- The model is not available
- Learn effective decision and control policies to interact with the environment.
- *Model-free prediction*
  Estimate the value function of an *unknown* MDP
- *Model-free control*
  Optimize the value function of an *unknown* MDP

Model Free prediction

1. *Monte Carlo learning*
   - The simplest approach to learning from experience
   - MC approaches require that the RL task is episodic, meaning that the task has a defined start and terminates after a finite number of actions
   - Calculate total cumulative reward at the end of the episode

2. *Temporal difference (TD) learning*
   - learns continuously by bootstrapping based on current estimates of the value function V or quality function Q
   - TD learning is typically more sample efficient than Monte Carlo learning

TD(0): 1-step look ahead

- The estimate of the one-step-ahead future reward is used to update the current value function.

$$V^{\text{new}}(\mathbf{s}_k) = V^{\text{old}}(\mathbf{s}_k) + \alpha \overbrace{\left(\underbrace{\mathbf{r}_k + \gamma V^{\text{old}}(\mathbf{s}_{k+1})}_{\text{TD target estimates } R_\Sigma} - V^{\text{old}}(\mathbf{s}_k) \right)}^{\text{TD error}}$$

- TD target is the estimate for the future reward, analogous to R in Monte Carlo learning
- the TD error is the difference between this target and the previous estimate of the value function, and it is used to update the value function

TD(n): n-step look ahead

- TD(n) is based on a multi-step look-ahead into the future.
- For example, TD(1) uses a TD target based on two steps into the future
- TD(n) uses a TD target based on n + 1 steps into the future

$$R_{\Sigma}^{(n)} = \mathbf{r}_k + \gamma \mathbf{r}_{k+1} + \gamma^2 \mathbf{r}_{k+2} + \cdots + \gamma^n \mathbf{r}_{k+n} + \gamma^{n+1} V(\mathbf{s}_{k+n+1})$$
$$= \sum_{j=0}^{n} \gamma^j \mathbf{r}_{k+j} + \gamma^{n+1} V(\mathbf{s}_{k+n+1}).$$

TD-λ: Weighted look ahead

- TD- $\boldsymbol{\lambda}$ creates a TD target that is a weighted average of the various TD(n) targets
- TD learning provides one of the strongest connections between reinforcement learning and learning in biological systems.
- These neural circuits are believed to estimate the future reward, and feedback is based on the difference between the expected reward and the actual reward, which is closely related to the TD error.

$$R_\Sigma^\lambda = (1-\lambda)\sum_{k=1}^{\infty}\lambda^{n-1}R_\Sigma^{(n)}$$

and the update equation is

$$V^{\text{new}}(\mathbf{s}_k) = V^{\text{old}}(\mathbf{s}_k) + \alpha\left(R_\Sigma^\lambda - V^{\text{old}}(\mathbf{s}_k)\right).$$

Model Free Control

- *On-policy learning*
  - "Learn on the job"
  - Learn about policy π from experience sampled from π
  - SARSA
- *Off-policy learning*
  - "Look over someone's shoulder"
  - The agent learns by being fed a dataset of states, actions, and rewards.
  - Learn about policy π from experience sampled from μ
  - Q-Learning

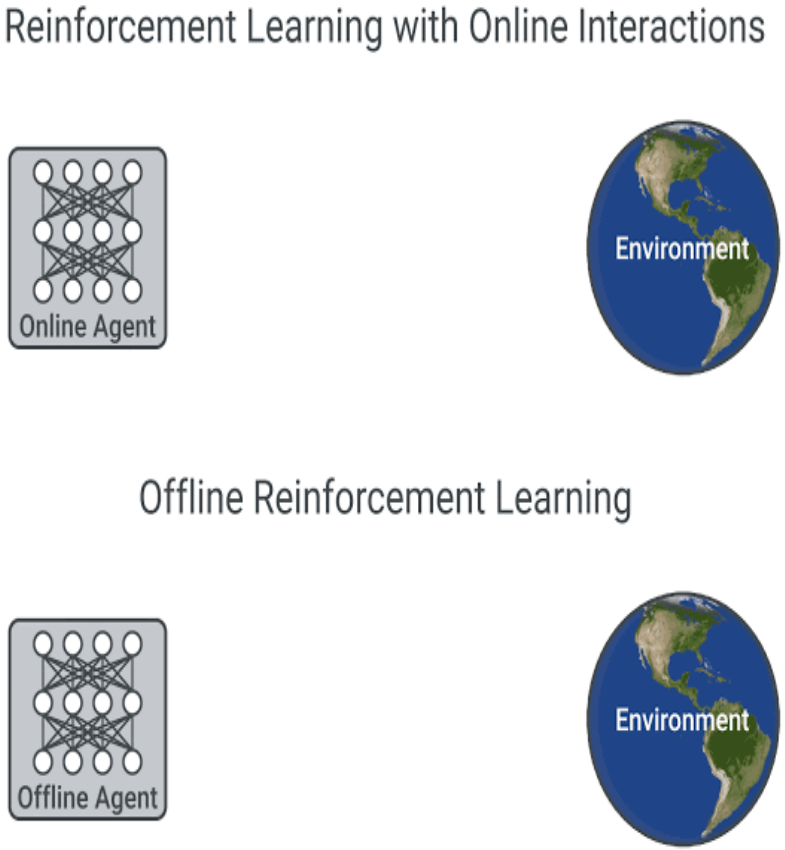


Fig. 5: on-policy vs off-policy

SARSA: State–action–reward–state–action learning

- SARSA is a popular TD algorithm that is used to learn the Q function on-policy.
- The Q update equation in SARSA(0) is nearly identical to the V update equation in TD(0)

$$Q^{\text{new}}(\mathbf{s}_k, \mathbf{a}_k) = Q^{\text{old}}(\mathbf{s}_k, \mathbf{a}_k) + \alpha \left(\mathbf{r}_k + \gamma Q^{\text{old}}(\mathbf{s}_{k+1}, \mathbf{a}_{k+1}) - Q^{\text{old}}(\mathbf{s}_k, \mathbf{a}_k) \right) .$$

Q-Learning

- One of the most central approaches in model-free RL.
- Notice that the only difference between Q-learning and SARSA(0) is that SARSA(0) uses Q(sk+1, ak+1) for the TD target, while Q-learning uses max_a Q(sk+1, a) for the TD target.
- Thus, SARSA(0) is considered on-policy because it uses the action ak+1 based on the actual policy: ak+1 = π(sk+1).
- n contrast, Q-learning is off-policy because it uses the optimal a for the update based on the current estimate for Q

$$Q^{\text{new}}(\mathbf{s}_k, \mathbf{a}_k) = Q^{\text{old}}(\mathbf{s}_k, \mathbf{a}_k) + \alpha \left(\mathbf{r}_k + \gamma \max_{\mathbf{a}} Q(\mathbf{s}_{k+1}, \mathbf{a}) - Q^{\text{old}}(\mathbf{s}_k, \mathbf{a}_k) \right).$$

Deep Reinforcement Learning (DRL)

- DRL is a combination of deep learning and RL
- Classic RL suffers from a representation problem, as many functions are defined over a very high dimensional state and action space.
- In 2015, Google DeepMind introduced deep Q-network (DQN)[3] delivering results exceeding humans in playing Atari games.

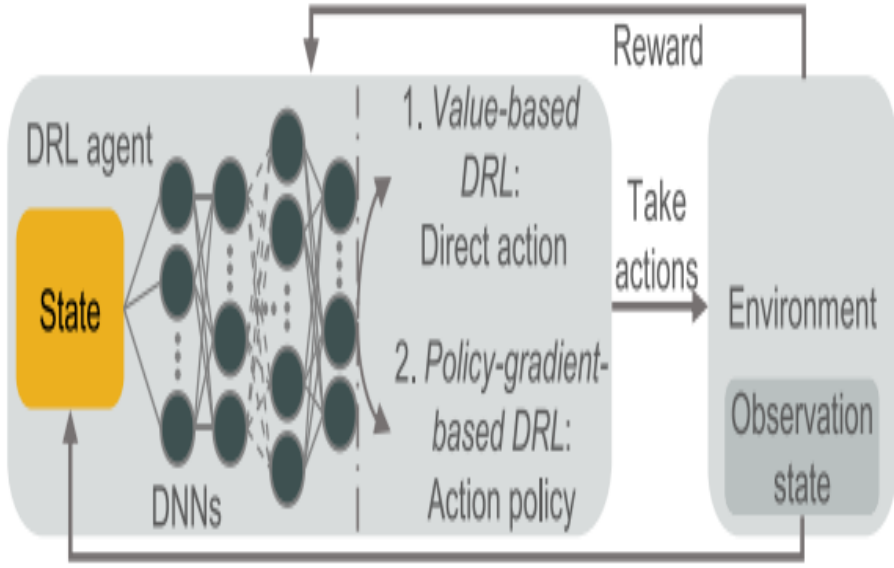


Fig. 5: Value-based and policy-gradient-based DRL approaches [1].

DRL Classifications

1. **Value-Based**
   - The estimation of the expected long-term reward of each state
   - Ex. Deep Q-learning, SRSA, Double deep Q-learning, etc.

2. **Policy-Based**
   - The goal is to produce the best policy
   - Ex. Policy gradient

3. **Actor-Critic**
   - Hybrid method combine both value-function and parameterized policies
   - Ex. Actor-critic, A2C, A3C, DPG, DDPG.

VALUE-BASED

Deep Q Learning (DQN)

- DQN[3] Uses deep neural networks to represent the quality function Q
- It is possible to approximate the Q function through some parameterization θ, where θ represents the weights of a deep neural network.

$$Q(\mathbf{s}, \mathbf{a}) \approx Q(\mathbf{s}, \mathbf{a}, \boldsymbol{\theta}),$$

$$\mathcal{L} = \mathbb{E}\left[\left(\mathbf{r}_k + \gamma \max_{\mathbf{a}} Q(\mathbf{s}_{k+1}, \mathbf{a}_{k+1}, \boldsymbol{\theta}) - Q(\mathbf{s}_k, \mathbf{a}_k, \boldsymbol{\theta})\right)^2\right]$$

- The training loss function is directly related to the standard Q-learning

Double Deep Q Learning (DDQN)

- A double DQN [4], uses different networks are used to represent the target and prediction Q functions, which reduces bias due to inaccuracies early in training
- Experience replay is a critical component of training a DQN
- It stores short segments of past experiences used in batches for the stochastic gradient descent during training.
- Moreover, it is possible to weigh past experiences by the magnitude of the TD error. This process is known as prioritized experience replay

Dueling deep Q networks (DDQNs)

- DDQNs [5] are used to improve training when actions have a marginal effect on the quality function.
  a DDQN splits the quality function into the sum of a value function and an *advantage* function *A*(s, a), which quantifies the additional benefit of a particular action over the value of being in that state
- The value and advantage networks have separate networks that are combined to estimate the *Q* function

$$Q(\mathbf{s}, \mathbf{a}, \boldsymbol{\theta}) = V(\mathbf{s}, \boldsymbol{\theta}_1) + A(\mathbf{s}, \mathbf{a}, \boldsymbol{\theta}_2).$$

POLICY-BASED

Policy Gradient Optimization

- Policy gradients are one of the most common and powerful techniques to optimize a policy that is parameterized
- When the policy π is parameterized by θ, it is possible to use gradient optimization on the parameters to improve the policy much faster than through traditional iteration.
- The parameterization may be a multi-layer neural network, in which case this would be a deep policy network
- There are several approaches to approximating this gradient, including through finite differences, the REINFORCE algorithm, and natural policy gradients

$$\nabla_{\boldsymbol{\theta}} R_{\Sigma,\boldsymbol{\theta}} = \mathbb{E}\left(Q(\mathbf{s},\mathbf{a})\nabla_{\boldsymbol{\theta}} \log\left(\boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{s},\mathbf{a})\right)\right).$$

$$\boldsymbol{\theta}^{\text{new}} = \boldsymbol{\theta}^{\text{old}} + \alpha \nabla_{\boldsymbol{\theta}} R_{\Sigma,\boldsymbol{\theta}},$$

ACTOR-CRITIC

Actor-critic network

- Actor-critic methods in reinforcement learning simultaneously learn a policy function and a value function, with the goal of taking the best of both value-based and policy-based learning
- The basic idea is to have an actor, which is policy-based, and a critic, which is value-based
- A simple actor-critic approach would update the policy parameters θ

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha \left(\mathbf{r}_k + \gamma V(\mathbf{s}_{k+1}) - V(\mathbf{s}_k) \right)$$

- Advantage actor-critic (A2C) network, the actor is a deep policy network, and the critic is a DDQNs. In this case, the update is given by

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha \nabla_{\boldsymbol{\theta}} \left((\log \boldsymbol{\pi}(\mathbf{s}_k, \mathbf{a}_k, \boldsymbol{\theta})) \, Q(\mathbf{s}_k, \mathbf{a}_k, \boldsymbol{\theta}_2) \right)$$

Decision Transformer (DT)

- *Don’t Optimize for Return, Just Ask for Optimality*
- The DT transforms the reinforcement learning problem into a sequence modeling problem [6].
- Traditional DRL agents are trained to optimize decisions to achieve the optimal return.
- *A Decision Transformer* is a sequence model that predicts future actions by considering past interactions, and a desired return to be achieved in future interactions.

Decision Transformer

- The core of the decision transformer model is based on the *GPT (Generative Pre-trained Transformer)* architecture
- The input tokens are embedded either with a linear layer if the state is a vector or a CNN encoder if it's framed.
- The decision transformer model is trained using the *offline reinforcement learning approach.*

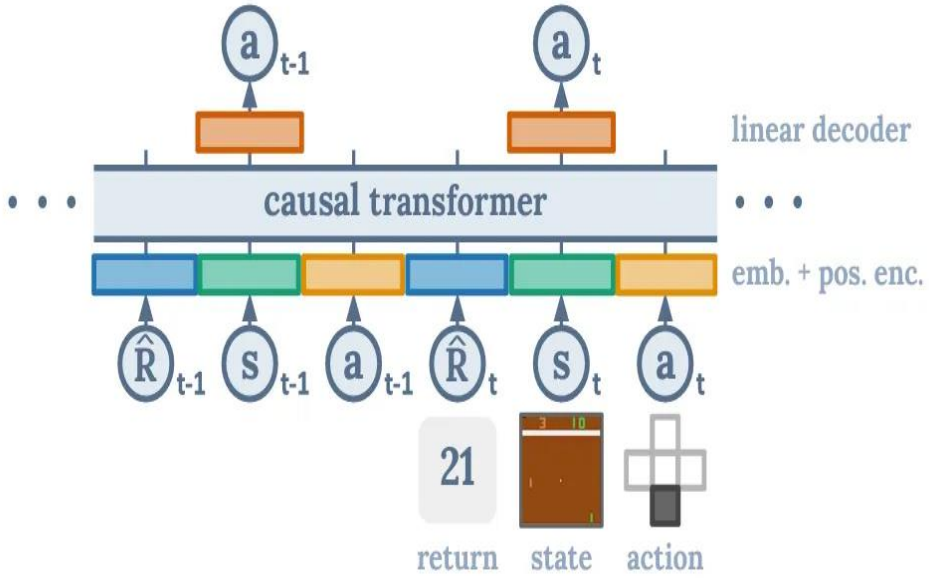
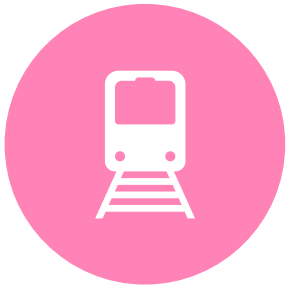


Figure 1: Decision Transformer architecture[1]. States, actions, and returns are fed into modality-specific linear embeddings and a positional episodic timestep encoding is added. Tokens are fed into a GPT architecture which predicts actions autoregressively using a causal self-attention mask.

A Multi-Game Decision Transformer (MGDT) [7]

Explore how to *build a generalist agent* to play many video games simultaneously.



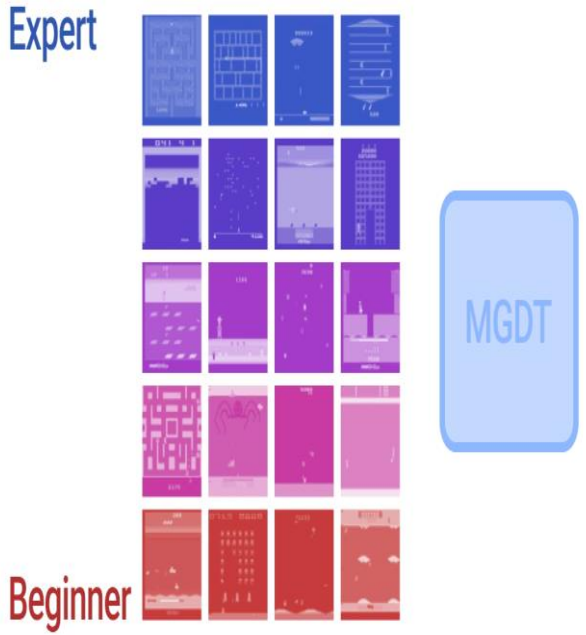
Model trains an agent that can play 41 Atari games at close-to-human performance and that can also be quickly adapted to new games via fine-tuning.

Use of *Decision Transformers* to train an RL agent.

This approach improves upon the few existing alternatives to learning multi-game agents, such as temporal difference (TD) learning or behavioral cloning (BC).

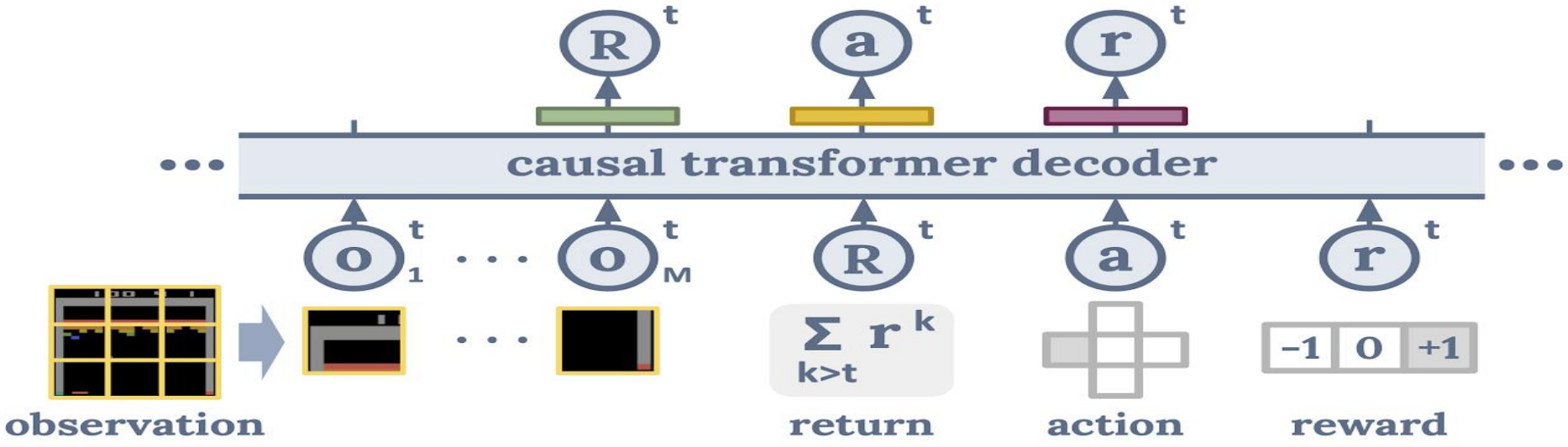
A Multi-Game Decision Transformer (MGDT)

- They model a *distribution of return magnitudes* based on past interactions with the environment during training.
- At inference time, they simply add an optimality bias that increases the probability of generating actions that are associated with higher returns.
- They also modified the Decision Transformer architecture to consider *image patches* instead of a global image representation.
- *Patches* allow the model to focus on local dynamics, which helps model game-specific information in further detail.



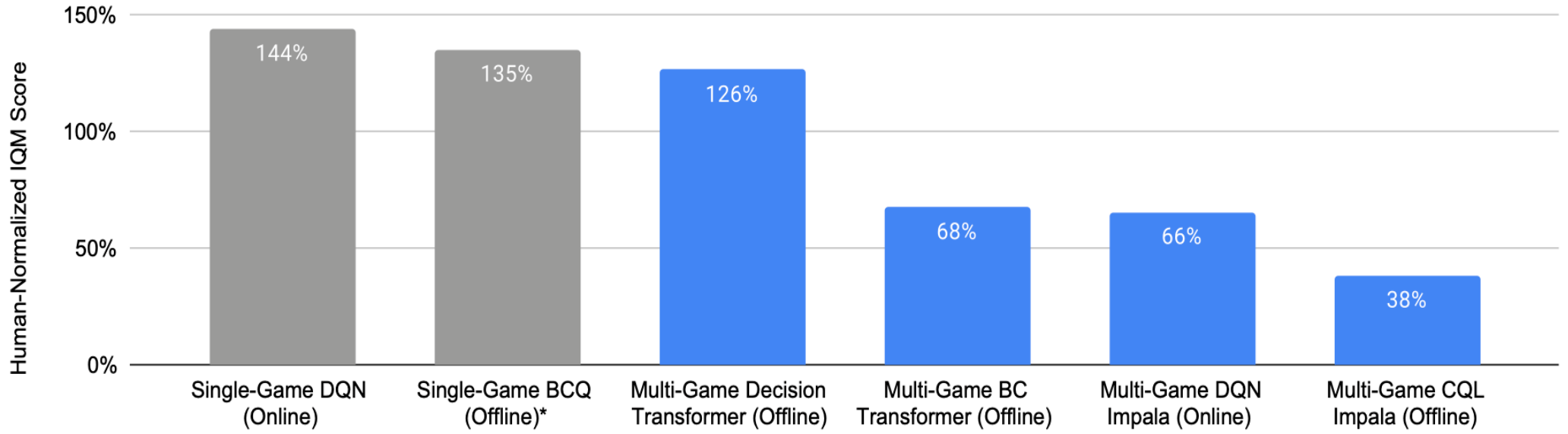
MGDT Model

- Each observation image is divided into a set of M patches of pixels which are denoted O.
- Return R, action a, and reward r follow these image patches in each input causal sequence.
- A Decision Transformer is trained to predict the *next input* to establish causality.
- MGDTs can be considered *pre-trained models* capable of being fine-tuned rapidly on small new gameplay data.



MGDT Results

- We train one Decision Transformer agent on a large (~1B) and broad set of gameplay experiences from 41 Atari games.
- In our experiments, the MGDT agent clearly outperforms existing reinforcement learning and behavioral cloning methods



Reference

[1] Brunton, Steven L., and J. Nathan Kutz. ”Data-driven science and engineering: Machine learning, dynamical systems, and control”. Ch11, Cambridge University Press, 2022.

[2] RL Course by David Silver here

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. Nature, 518(7540):529, 2015.

[4] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double Q-learning. In Proceedings of the AAAI conference on artificial intelligence, volume 30, 2016.

[5] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[6] Chen, Lili, et al. "Decision transformer: Reinforcement learning via sequence modeling." Advances in neural information processing systems 34 (2021): 15084-15097.

[7] Lee, Kuang-Huei, et al. "Multi-game decision transformers." arXiv preprint arXiv:2205.15241 (2022).

Thank You