Machine Learning and Financial Applications

Lecture 4
Deep Reinforcement Learning in Portfolio Optimization

Liu Peng

liupeng@smu.edu.sg

Video tutorial

https://youtu.be/Yd3H_h-7C58?si=_gHkcvTs-TePQdUB

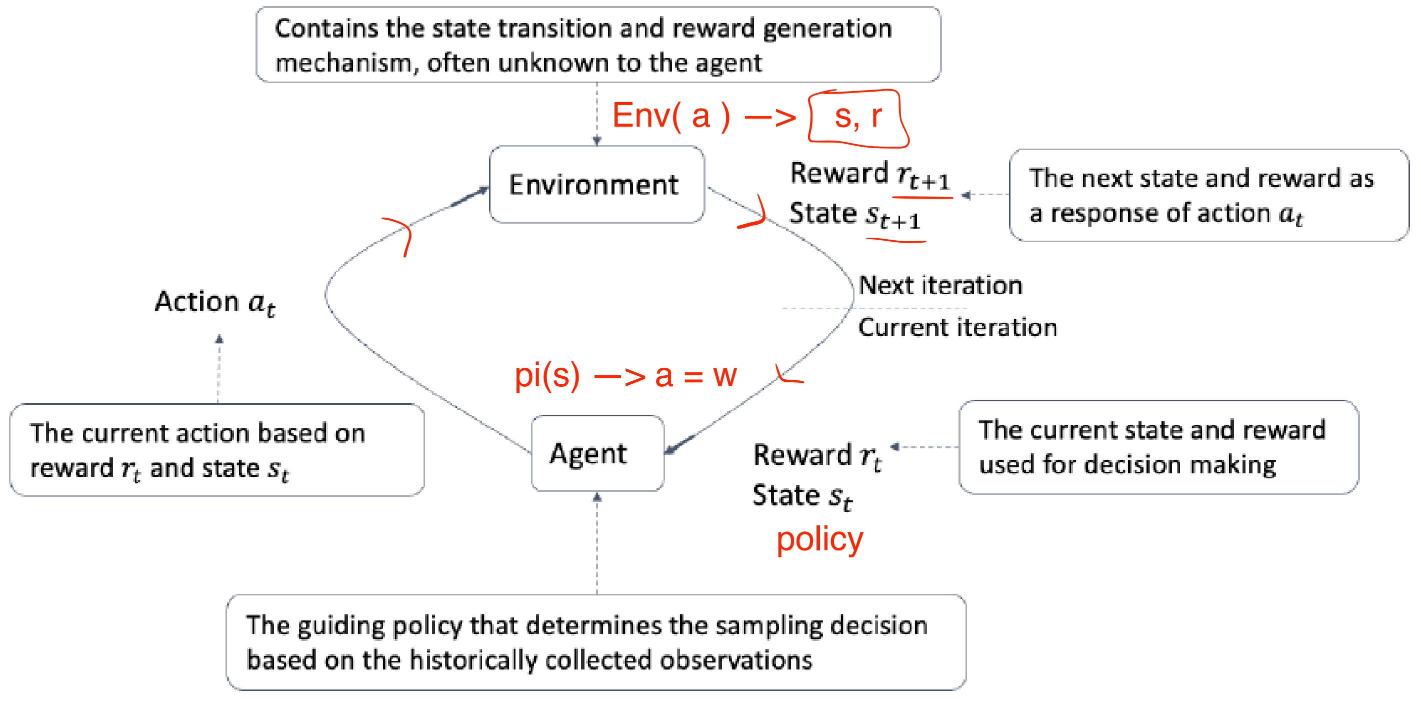
Learning outcomes

Get to know RL fundamentals

Able to create RL environment for portfolio optimization

Able to train and evaluate RL agent that performs portfolio optimization

RL Learning Framework



Contains the state transition and reward generation mechanism, often unknown to the agent
Env(a) —> s, r
Environment
Reward r_{t+1}
State s_{t+1}
The next state and reward as a response of action a_t
Action a_t
Next iteration
Current iteration
pi(s) —> a = w
The current action based on reward r_t and state s_t
Agent
Reward r_t
State s_t
The current state and reward used for decision making
policy
The guiding policy that determines the sampling decision based on the historically collected observations

Figure 1.1: Iterative interaction between the agent and the environment.

Using Reinforcement Learning for Portfolio Optimization

Agent–Environment Setup

- State: market observations (prices, indicators, macro factors)
- Action: portfolio weight vector allocation

Reward Design

- Risk-adjusted return

Key Architectures

- Value-based: Deep Q-Networks (DQN)
- Policy-based: PPO, A2C/A3C
- Actor-Critic: DDPG, SAC

Practical Considerations

- Transaction costs & market impact
- Position-sizing constraints (e.g. leverage, turnover limits)

Stability & Efficiency

- Experience replay, target networks
- Off-policy vs. on-policy trade-offs

Risk Management Integration

- Constraint handling via penalty in reward or Lagrangian methods
- Incorporating CVaR/drawdown objectives

Review of Modern Portfolio Theory

LR is good at PO, bad at forecasting

$$\begin{aligned} \underset{\mathbf{w}}{\text{minimize}} \quad & \underline{\mathbf{w}^T \Sigma \mathbf{w}} \\ \text{subject to} \quad & \left\{ \begin{aligned} & \mathbf{w}^T \mu = \underline{\mu_0}, \\ & \mathbf{w}^T 1 = 1, (\#eq:meanvar) \end{aligned} \right. \end{aligned}$$

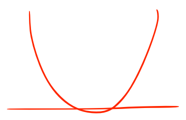
ptf risk

ptf target return

$$\Longleftrightarrow$$

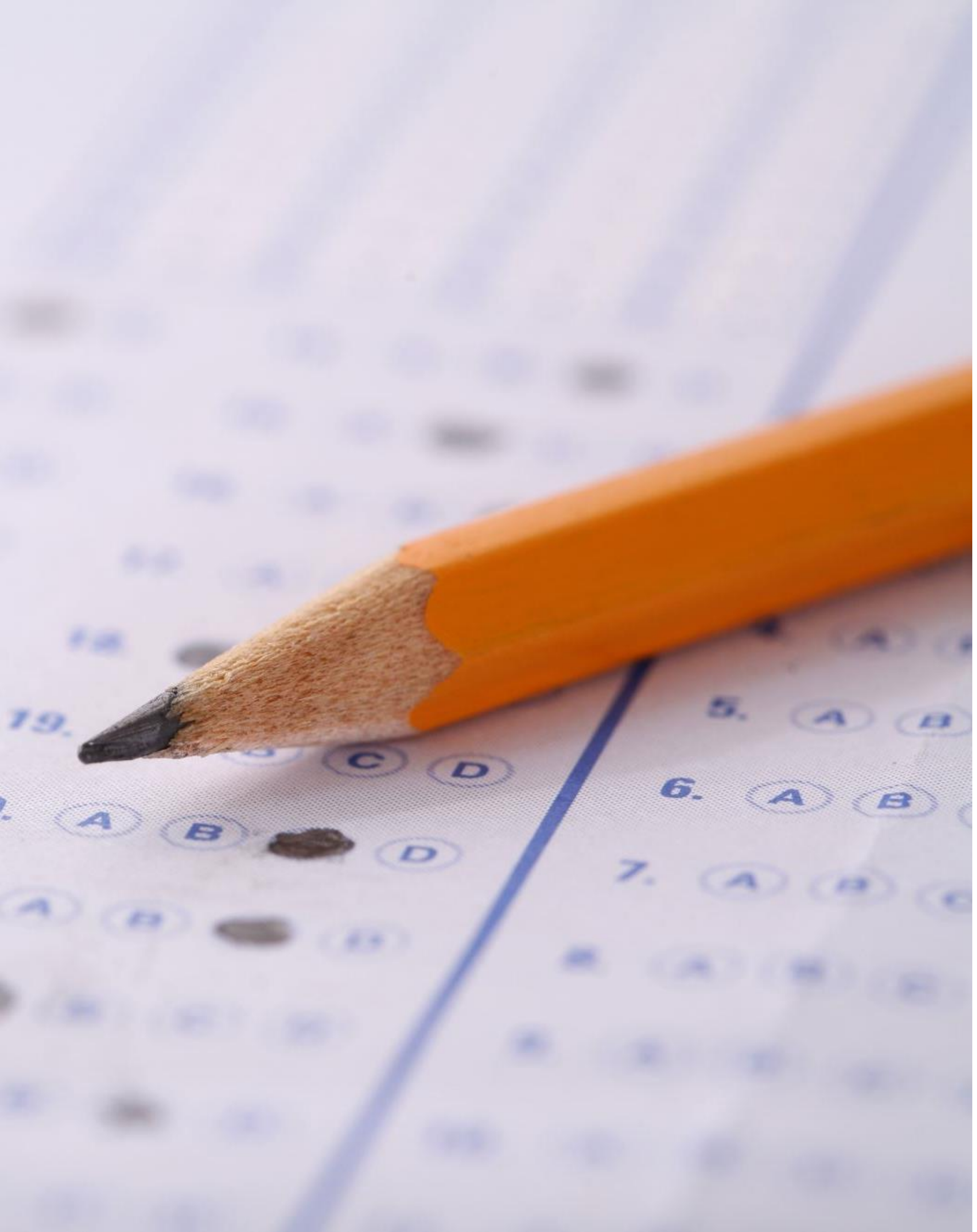
$$\begin{aligned} \underset{\mathbf{w}}{\text{maximize}} \quad & \mathbf{w}^T \hat{\mu} \quad \uparrow \\ \text{subject to} \quad & \mathbf{w}^T \Sigma \mathbf{w} \le \underline{\sigma_p^2}, \\ & \mathbf{w}^T 1 = 1, \end{aligned}$$

$$\downarrow \mathcal{L}(\mathbf{w}, \lambda_1, \lambda_2) = \mathbf{w}^T \Sigma \mathbf{w} - \underline{\lambda_1 (\mathbf{w}^T \mu - \mu_0)} - \underline{\lambda_2 (\mathbf{w}^T 1 - 1)}. \quad \star$$



$$\underline{\frac{\partial \mathcal{L}}{\partial \mathbf{w}}} = 2 \Sigma \mathbf{w} - \lambda_1 \mu - \lambda_2 1 = \underline{0}.$$

$$\underline{\mathbf{w}^*} = \left(\frac{C\mu_0 - B}{AC - B^2} \right) \Sigma^{-1} \mu + \left(\frac{A - B\mu_0}{AC - B^2} \right) \Sigma^{-1} \mathbf{1}$$



In-class quiz

- Q1-4

Group Discussion

- Propose a few user cases for RL based on your past project/work experiences

Why Deep RL for PO?

Model-Free End-to-End Learning

- Learns trading policies directly from raw market data, bypassing explicit return/risk prediction steps.

Continuous Adaptation

- Updates strategies online as new data arrive, reducing the need for periodic retraining on historical samples.

Capturing Non-Linear Dependencies

- Leverages deep networks to model complex asset relationships for more informed allocation decisions.

Scalable to High Dimensions

- Handles large universes of assets and intricate market structures, making it suitable for real-world portfolio tasks.

Built-in tradeoff between exploration and exploitation

Markov Decision Process

- A mathematical framework that provides a formal description of an environment for RL.

policy pi(s) —> a

$$(\overset{\checkmark}{S}, \overset{\checkmark}{A}, \overset{\checkmark}{P}, R, \gamma)$$

$$s_t = \begin{bmatrix} P_t \\ I_t \\ \boldsymbol{w}_{t-1} \end{bmatrix}$$

$$a_t = \boldsymbol{w}_t = \begin{bmatrix} w_{1,t} \\ w_{2,t} \\ \vdots \\ w_{N,t} \end{bmatrix}, \quad \text{subject to} \quad \sum_{i=1}^{N} w_{i,t} = 1.$$

immediate

ptf return

$$r(s_t, a_t) = \log\left(\sum_{i=1}^{N} w_{i,t} \frac{P_{i,t}}{P_{i,t-1}}\right).$$

transition prob (dynamics)

$$p(s_{t+1}|s_t, a_t) = \mathbb{P}(S_{t+1} = s_{t+1}|S_t = s_t, A_t = a_t),$$

gamma1 + r gamma2 + r^2 gamma3

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{T-1} \gamma^t r(s_t, \pi(s_t))|s_0\right],$$

RL Environment Setup for PO

- **State Space** ($\mathcal{S}$): Encodes market information at time t:

$$s_t = \begin{bmatrix} P_t \\ I_t \\ \mathbf{w}_{t-1} \end{bmatrix}, \quad \text{where } P_t : \text{price data}, I_t : \text{market indicators}, \mathbf{w}_{t-1} : \text{previous weights.}$$

- **Action Space** ($\mathcal{A}$): Portfolio allocation weights at time t:

$$a_t = \mathbf{w}_t = \begin{bmatrix} w_{1,t} \\ w_{2,t} \\ \vdots \\ w_{N,t} \end{bmatrix}, \quad \text{subject to} \quad \sum_{i=1}^{N} w_{i,t} = 1.$$

- **Reward Function** ($r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$): Immediate payoff (log portfolio return):

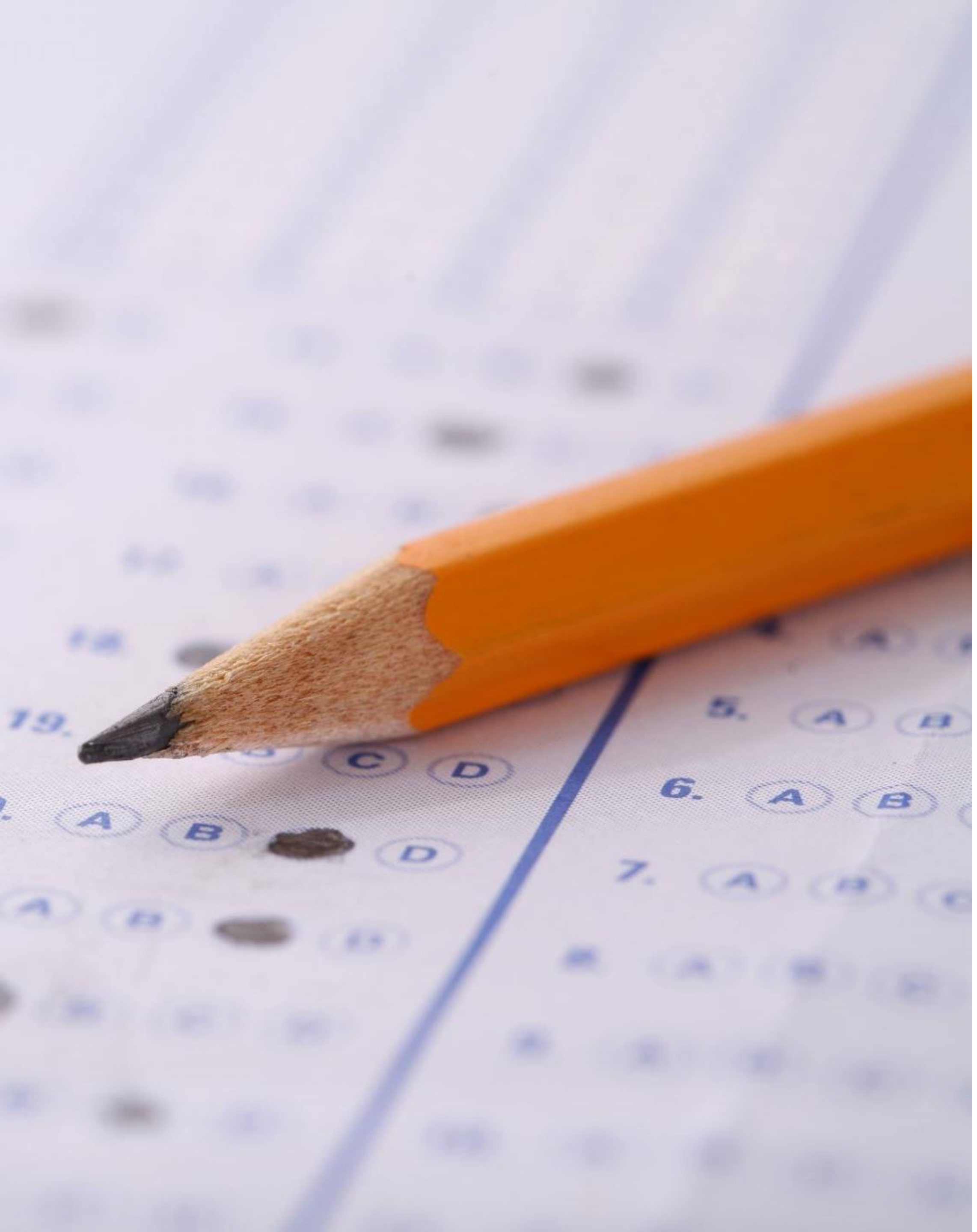
$$r(s_t, a_t) = \log \left(\sum_{i=1}^{N} w_{i,t} \frac{P_{i,t}}{P_{i,t-1}} \right).$$

- **Transition Dynamics** ($p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$): Probability of transitioning to next state, implicitly learned:

$$p(s_{t+1}|s_t, a_t) = \mathbb{P}(S_{t+1} = s_{t+1} \mid S_t = s_t, A_t = a_t).$$

- **Learning Objective**: Find optimal policy $\pi^* : \mathcal{S} \to \mathcal{A}$ maximizing cumulative discounted rewards:

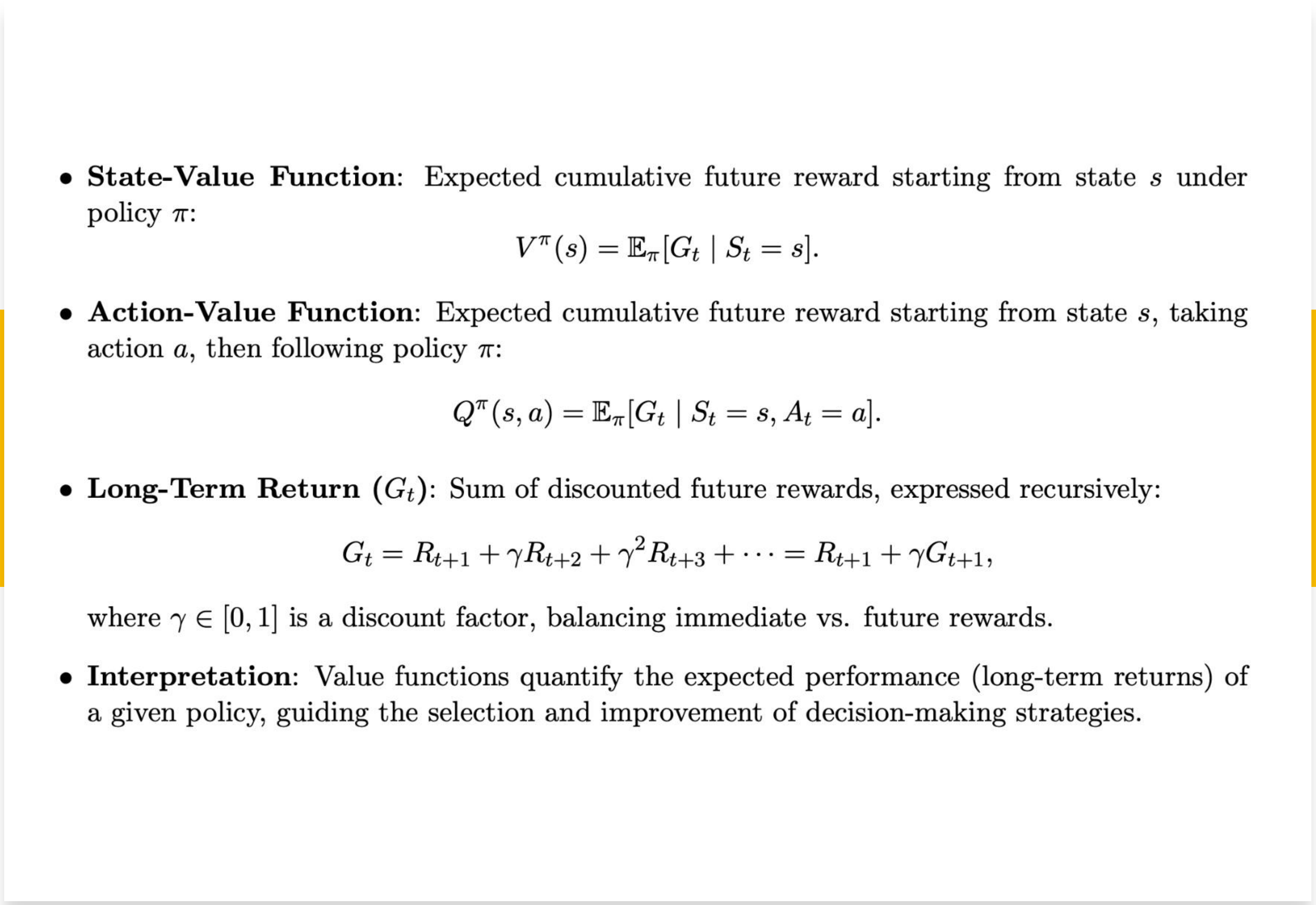
$$\pi^* = \arg\max_{\pi} \mathbb{E} \left[\sum_{t=0}^{T-1} \gamma^t r(s_t, \pi(s_t)) \right], \quad \gamma \in [0,1].$$



In-class quiz

- Q5-8

Value Functions



- **State-Value Function**: Expected cumulative future reward starting from state s under policy π:
$$V^\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s].$$
- **Action-Value Function**: Expected cumulative future reward starting from state s, taking action a, then following policy π:
$$Q^\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a].$$
- **Long-Term Return (G_t)**: Sum of discounted future rewards, expressed recursively:
$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = R_{t+1} + \gamma G_{t+1},$$
where $\gamma \in [0,1]$ is a discount factor, balancing immediate vs. future rewards.
- **Interpretation**: Value functions quantify the expected performance (long-term returns) of a given policy, guiding the selection and improvement of decision-making strategies.

Bellman Equations for State and Action-Value Functions

- **Bellman Equation (Intuition)**: Defines a recursive relationship between the value of a state (or state-action pair) and its successor states (or state-action pairs), based on the principle of optimality.
- **State-Value Bellman Equation**: Expresses value of a state as the immediate reward plus discounted future rewards:

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) \left[r + \gamma V^\pi(s')\right].$$

- **Action-Value Bellman Equation**: Expresses value of state-action pair as the immediate reward plus discounted future rewards:

$$Q^\pi(s,a) = \sum_{s',r} p(s',r|s,a) \left[r + \gamma \sum_{a'} \pi(a'|s') Q^\pi(s',a')\right].$$

- **Significance in RL**: These recursive equations form the foundation for iterative algorithms (value iteration, policy iteration, Q-learning) and are fundamental in deep RL for training value-function approximators.

Optimal Value Functions

The optimal state-value function $v^*(s)$ is the maximum value function over all policies:

$$v^*(s) = \max_{\pi} v_\pi(s). \tag{11}$$

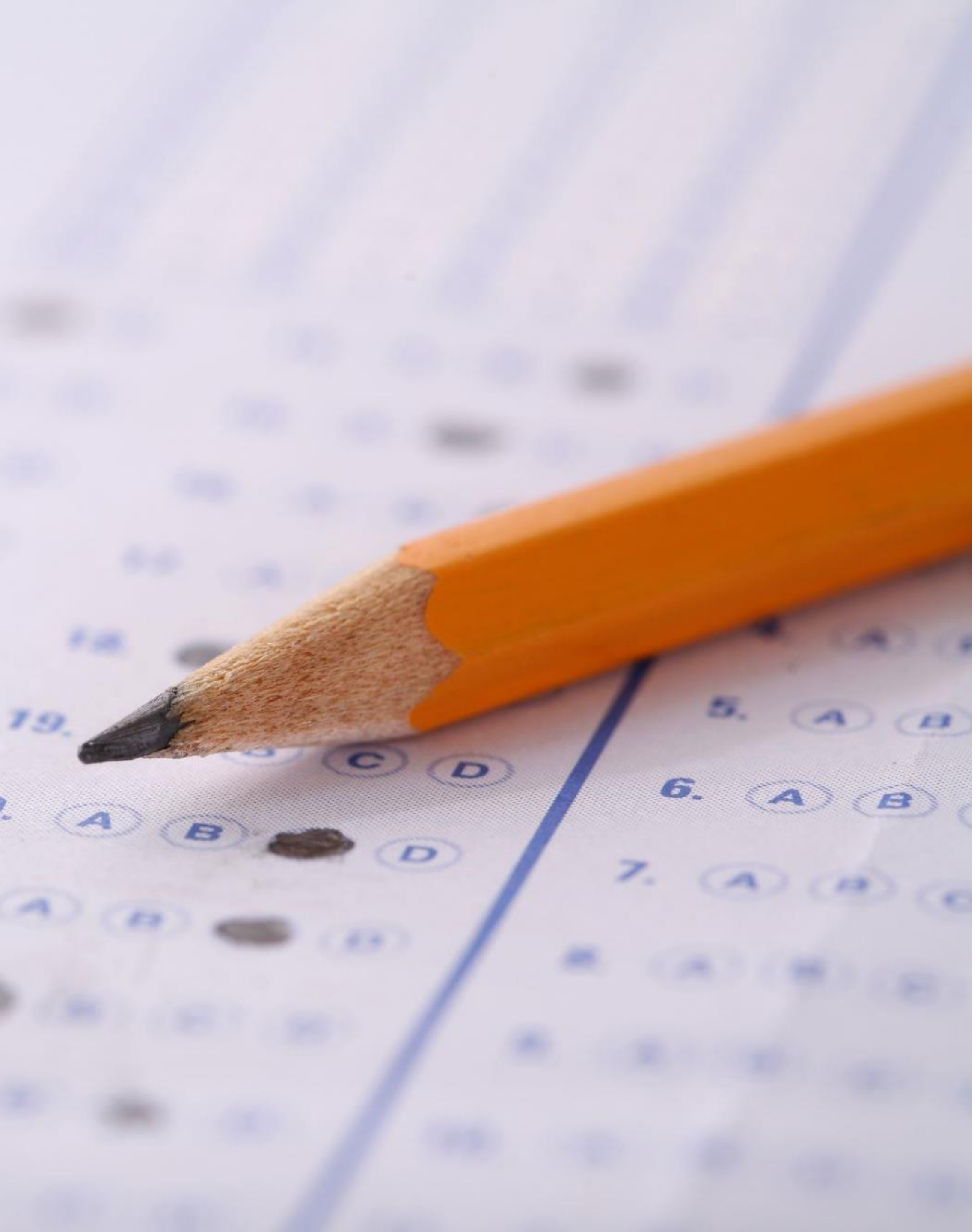
The optimal action-value function $q^*(s,a)$ is the maximum action-value function over all policies:

$$q^*(s,a) = \max_{\pi} q_\pi(s,a). \tag{12}$$

The Bellman optimality equations are:

$$v^*(s) = \max_{a \in A} \mathbb{E}[\overset{\text{immediate reward}}{\underset{\text{signal}}{R_{t+1}}} + \overset{\text{future}}{\gamma v^*(S_{t+1})} | S_t = s, A_t = a], \tag{13}$$

$$q^*(s,a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} q^*(S_{t+1}, a') | S_t = s, A_t = a]. \tag{14}$$



In-class quiz

- Q9-12

Q-Learning

Algorithm Type: Off-policy, model-free, value-based RL for learning $Q^*(s,a)$ without knowing dynamics.

Objective: Estimate

$$Q^*(s,a) = \max_\pi \mathbb{E}\big[G_t \mid S_t = s, A_t = a\big], \quad \pi^*(s) = \arg\max_a Q^*(s,a).$$

Core Update Rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big[\underline{r} + \overset{\text{long-term}}{\underline{\gamma \max_{a'} Q(s',a')}} - Q(s,a)\big],$$

where α is the learning rate and γ the discount factor.

Exploration–Exploitation: Use ε-greedy:

- With probability ε, choose a random action.
- Otherwise, choose $a = \arg\max_a Q(s,a)$.

Convergence Guarantees: With decaying $\{\alpha_t\}$ (s.t. $\sum_t \alpha_t = \infty$, $\sum_t \alpha_t^2 < \infty$) and sufficient exploration, $Q(s,a) \to Q^*(s,a)$.

Practical Considerations:

- *Tabular vs. Function Approximation:* Tabular for small discrete spaces; use neural nets when large/continuous.
- *Stability Tricks:* Experience replay and target networks (in Deep Q-Learning) to decorrelate updates.

Deep Q Network (DQN)

Core Idea:

Uses deep neural networks to approximate the Q-value function:

$$Q(s, a; \theta) \approx Q^*(s, a)$$

Algorithmic Components:

- **Experience Replay:**

  Stores past experiences (s,a,r,s') to stabilize training.

- **Target Network:**

  Periodically updated copy of Q-network to reduce correlations and improve stability.

- **Epsilon-Greedy Policy:**

  Balances exploration (random action) and exploitation (best-known action).

Loss Function (Bellman Error):

Minimizes difference between current and target Q-values:

$$L(\theta) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right]$$

Applications in Portfolio Optimization:

Learns optimal trading policies directly from market states without explicit forecasting of returns or risks.

Implementing DQN

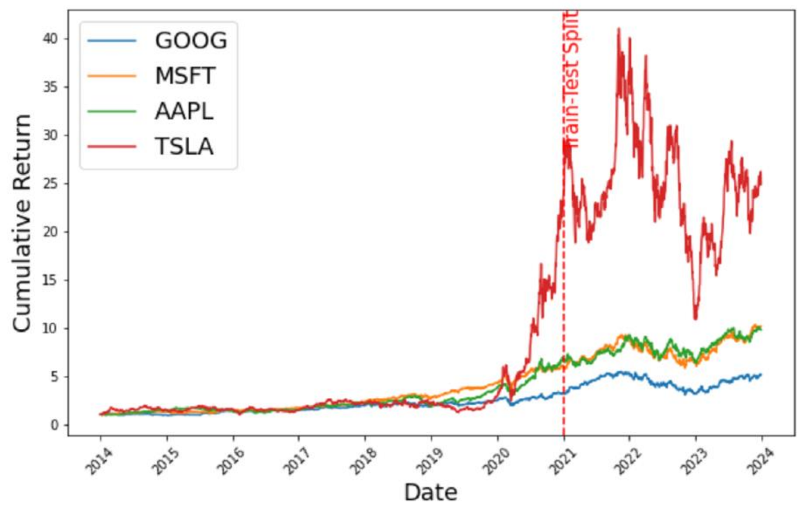


Exhibit 1: Daily cumulative returns of GOOG, MSFT, AAPL, and TSLA.

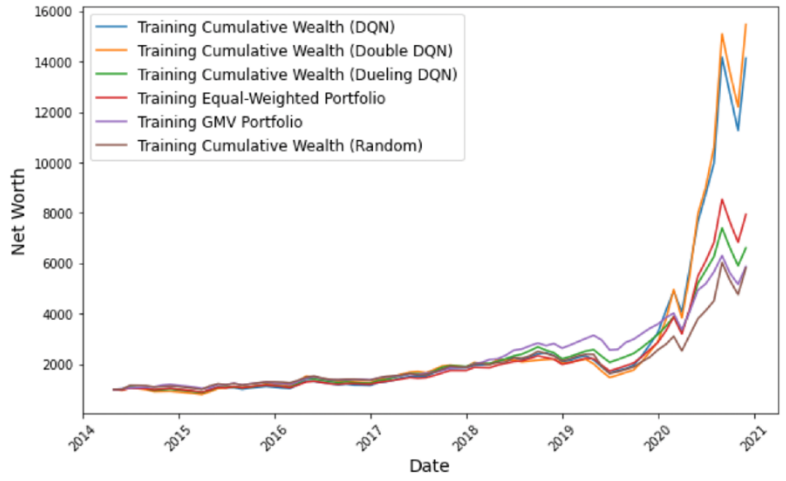


Exhibit 2: Cumulative wealth over time during the training period for all portfolio strategies. DQN and Double DQN performed best, followed by Dueling DQN and equal weighted portfolio. GMVP and random policy performed worst.

Deep Deterministic Policy Gradient (optional)

Actor-Critic Architecture:

- **Actor**: Represents the deterministic policy $\pi_\theta(s)$ that maps states s directly to continuous actions a.
- **Critic**: Estimates the action-value function $Q_\phi(s, a)$, evaluating the expected return of action a in state s.

Deterministic Policy Gradient Theorem:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s\sim\mathcal{D}}\left[\nabla_\theta \pi_\theta(s) \nabla_a Q^\pi(s,a)\Big|_{a=\pi_\theta(s)}\right]$$

See accompanying notes for derivation

- Guides the Actor to adjust policy parameters θ in directions that maximize the expected return.

Continuous Action Spaces:

- Enables precise control over actions without discretization, addressing scalability and precision issues inherent in traditional Q-learning.

Enhancements for Efficient Learning in DDPG (optional)

Experience Replay Buffer ($\mathcal{D}$):

$$\mathcal{D} = \{(s_t, a_t, r_t, s_{t+1})\}$$

- Stores past experiences to break temporal correlations and improve sample efficiency through random minibatch sampling.

Off-Policy Learning:

- Enables learning from historical data irrespective of the current policy, facilitating effective exploration and exploitation.

Actor-Critic Collaboration:

- **Critic** provides accurate Q-value estimates to guide the **Actor**.
- **Actor** continuously improves the policy to select actions that maximize the Critic's estimated returns, ensuring coherent and stable learning dynamics.

Actor-Critic Architecture (optional)

Critic Update via Bellman Equation:

$$L(\phi) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(Q_\phi(s,a) - \left(r + \gamma Q'_{\phi'}(s', \pi'_{\theta'}(s'))\right)\right)^2\right]$$

- Minimizes the Mean Squared Bellman Error (MSBE) to accurately estimate Q-values.

Actor Update Using Policy Gradient:

$$\theta \leftarrow \theta + \alpha_\theta \nabla_\theta J(\pi_\theta)$$

- Applies gradient ascent to optimize the policy based on Critic feedback.

Target Networks for Stability:

$$\phi' \leftarrow \tau\phi + (1-\tau)\phi'$$

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$$

- Soft updates with factor $\tau \ll 1$ ensure gradual parameter changes, mitigating oscillations and divergence during training.

DDPG Algorithm (optional)

1. **Initialize** Actor, Critic, Target Networks, and Replay Buffer.
2. **For each episode**:
   - Initialize state s_0.
   - **For each timestep**:
     - Select action $a_t = \pi_\theta(s_t) + \mathcal{N}_t$.
     - Execute a_t, observe r_t and s_{t+1}.
     - Store (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$.
     - Sample minibatch from $\mathcal{D}$.
     - Update Critic and Actor networks.
     - Soft update Target Networks.

Unique advantages

- **Handles Continuous Action Spaces**: Directly outputs precise actions without discretization.
- **Sample Efficiency**: Utilizes Experience Replay to make better use of collected data.
- **Stable Learning**: Target Networks and soft updates prevent oscillations and divergence.
- **Off-Policy Learning**: Learns from past experiences regardless of the current policy, enhancing exploration.

```
irror_mod = modifier_ob.
 mirror object to mirror
irror_mod.mirror_object
peration == "MIRROR_X":
irror_mod.use_x = True
irror_mod.use_y = False
irror_mod.use_z = False
 operation == "MIRROR_Y":
irror_mod.use_x = False
irror_mod.use_y = True
irror_mod.use_z = False
 operation == "MIRROR_Z":
irror_mod.use_x = False
irror_mod.use_y = False
irror_mod.use_z = True

election at the end -add
_ob.select= 1
er_ob.select=1
ntext.scene.objects.active
("Selected" + str(modifier
irror_ob.select = 0
 bpy.context.selected_ob
ata.objects[one.name].se

int("please select exactly

-- OPERATOR CLASSES ----

types.Operator):
 X mirror to the selected
ject.mirror_mirror_x"
rror X"

context):
ext.active_object is not
```

Coding session

Group Homework

- Open exercise
- Experiment with other types of RL algorithms, such as policy networks and actor-critic networks
- Consider other constraints and reward designs

Due one day before class starts next week

Homework

Watch/review video tutorials and class recording

Post learning reflections and questions if any

Complete group homework