10/25
Danish M

MARLTS (Multi agent reinforcement learning trading system)

I)

- Initially:
  - In train_multi.py → "policy_cl": (None, obs_space, act_space, {}),
    - Instantiates <u>two</u> neural networks to become the "brains" of the agent.
      - Two because using PPO → Actor-Critic method
  - There are

Soldiers → • Actor → learns the policy $\pi_\theta(a|s)$

- takes a state (observation), s, and decides on action, a, to take. In this project it takes observation vectors and outputs desired portfolio <u>weights</u>, θ. Initially, θ (weights) are initialized randomly → New, no skills, etc.

Strategist → • Critic, learns value function $V_\phi(s)$

- Takes a State, s, and estimates total expected <u>future reward</u> from that point onwards. provides a strategic assessment of how "good" current situation is, but initially, θ are randomized, signifying no strategic sense.

- The other policies do not get trained, and do not learn, they're basically NPC's. The RL agent is the hero.

Then:

- Deploying our "Soldiers" and "Strategists" to "Corporals":
- Config = config.env_runners(num_env_runners = 1, ...)
  - Sets up multiple, identical "battlefields"
  - Instead of learning from one "battle" at a time, the agent can fight in multiple simultaneously and pool together the intelligence gained from them

II

- Lets go through a single episode. these its a sequence of interactions lasting for an episode_length = 252 timesteps

- Timestep t = 0

1. Environment picks random start date from historical data ! RL agents portfolio is initialized with cash = 100_000, holdings = [0, 0, 0, 0] → env.reset()

2. Environment generates first observation vector s_0, This vector contains historical log returns, all-zero holdings and normalized cash value.

3. s_0 is sent to policy-PI's "Actor" (Solidity) network. Right now, this network's weights, θ, are random, so it outputs basically a random action, a_0, which is a vector of portfolio weights

4. Environment also gets actions from the heuristic agents based on the same market data (ex rolling mean, maximizing, etc. agents)

5. Market reaction. The environment aggregates, or combines all these actions into net_notational, calculates the market impact, determines the exec_prices and updates everyone's portfolios.

6. Environment calculates reward r_0, and the next observation s_1. Stored as tuple (s_0, a_0, r_0, s_1) → "Experience"

↓

• Timestep t = "some generic step in the episode", agent has some cash and holdings from previous random actions.

1. perception: agent receives current observation vector s_t.

2. s_t is now fed to both Actor & Critic networks

↳ • Actor → outputs prob. distr. over actions.

The function "compute_single_action" then "samples" an action, a_t, from this distribution. Agent still exploring, but decisions now slightly biased by small updates received from previous learning steps

↳ • Critic → outputs a single number $V(s_t)$, which is the current, yet still poor, estimate of the total future reward it expects to get from the current state

3. Environment collects PL agent's actions, a_t, as well as deterministic actions from mom, mom, etc. agents

↳ impact = np.clip(1.0 + (self.impact_coeff * net_notational / (liquidity + 1e-9)

exec_prices = prev_prices * impact

↳ execution

4. Step function makes the trade at the impacted prices and calculates reward r_t. Reward is MARL-aware, taking feedback both from profit & performance relative to the group.

- This goes on for all steps $t = 2 \dots T$. Results in a "complete repost": $[(s_0, a_0, r_0, V(s_0)), (s_1, a_1, r_1, V(s_1)), \dots]$

III

PPO Learning Phase:

- Learning begins once parallel environments have collected a batch of trajectories.

1. Calculating advantage

- For a given state s_t, was the action a_t that was taken better or worse than the average action we could've taken?

↓

- Advantage function: $A(s, a)$

- Estimating advantage using "Temporal Difference" Error

↓

- $\hat{A}_t = r_t + \gamma V(s_{t+1}) - V(s_t)$

(Why? - reduces variance !"one offs")

- $r_t + \gamma V(s_{t+1})$ is the improved estimate after seeing the real immediate reward and cities assessment of the next State s_{t+1}. γ is a factor that "discounts" predicted rewards, and values immediate ones higher

- $V(s_t)$ was critics prediction before action

- $\hat{A}_t > 0$ → Outcome better than expected, a_t was good
- $\hat{A}_t < 0$ → Outcome worse than expected, a_t was bad

2. Training Actor

- Goal: ↑ probability of actions that had a positive advantage and ↓ probability of those with negative advantage

- Conventionally, a single "bad batch" of skewed experiences could ruin a policy update.

- PPO solution → Clipping:

$$L^{CLIP}(\theta) = \hat{E}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right]$$

? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?

- $\hat{E}_t$: for all the experiences in the batch
- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$: Probability ratio, measures how much more likely our new, updated policy ($\pi\theta$) is to take the same action, a_t, compared to old policy, ($\pi_{\theta_{old}}$) that gathered the data.
- $r_t(\theta)\hat{A}_t$: Standard objective. If advantage $\hat{A}_t$ is positive, we want to push $r_t(\theta)$ up ↑
- $clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t$: Secondary objective, where probability ratio is "clipped" or confined to a narrow window around 1, (like 0.8 or 1.2, where $\epsilon = 0.2$).
- min( ) : take the minimum of the two objectives

Why clip? : Assuming we get a really high $\hat{A}_t$, the normal objective would tell us to increase probability of action by a lot. But the clipped objective puts a limit to that, preventing the agent from ...

... getting too "cocky". → Ensures stable, reliable learning.

- This results in a "pessimistic" policy update

- To maximize → use Gradient Ascent to nudge Actor's weights θ in the right direction, Policy clips the update so each "new" model doesn't differ too much from the last

- Through gradient ascent, actor hopes to adjust weights in $r_t(\theta)$, and more fully: $L^{CLIP}(\theta)$

- $\theta_{k+1} = \theta_k + \alpha \nabla_\theta L^{CLIP}(\theta_k)$

  - θ_{k+1}: new weights
  - α: learning rate
  - $\nabla_\theta L^{CLIP}(\theta_k)$: gradient ∇ of objective function
  - θ_k: current weights

3. What about the critic?

- Needs to get better at predicting value function
  - prediction was $V_\phi(s_t)$
  - "true" value (target) is $V_t^{targ} = r_t + \gamma V(s_{t+1})$
  - Loss is simply mean squared error: $(V_\phi(s_t) - V_t^{targ})^2$

- Use Gradient descent to minimize loss, $(L^{VF}(\phi))$
  - Critics predic. become more accurate
  - Better critic leads to better advantage estimate

↓
Actor learns more effectively

- For Critic Network Update:
  - Need:
    1. States (s_t)
    2. Value Targets (V_{target_t} or $\hat{R}_t$)

$$\rightarrow V_{target_t} = \hat{A}_t + V_{\phi_{old}}(s_t)$$

- $$L^{VF}(\phi) = \frac{1}{N}\sum_{t=1}^{N}\left(V_\phi(s_t) - V_{target_t}\right)^2$$

↳ $$\phi_{k+1} = \phi_k - \beta \nabla_\phi L^{VF}(\phi_k)$$

(β: Critic learning rate; $\nabla_\phi L^{VF}(\phi_k)$: loss gradient)

* After num_sgd_iter passes over batch, Actor & Critic networks now slightly smarter → Updated policy sent to the env → Cycle Repeats, and the initially random agent slowly gets "smarter" compared to others.

PPO Learning → By Hand

- Setup:
  - env: MultiAssetMarketEnv: Continuous action in $[0,1]^n$ → normalized to weights → trades execute simultaneously → reward is shaped
  - agents: "mom_0", "mean_0", "mm_0", "rl_trader_0"
  - PPO trained: "policy_fl" only in "train_multi.py"
  - action: RLlib (library) PPO uses a diagonal Gaussian policy, squashes box bounds → env renormalizes to sum to 1. Going to ignore tanh Jacobian term for hand calc simplicity.

- reward shaping in code:

```
base = (cash + holdings.new_prices) - (cash + holdings.prev_prices)

mean_val = mean(all_vals), std_val = std(all_vals) + 1e-9

#cooperation →  coop_bonus = coop_weight * (1.0 / (1.0 + std_val))

#competition →  comp_bonus = comp_weight * ((mean_val - all_vals[a]) / (std_val + 1e-9))

shaped_reward[a] = base + coop_bonus - comp_bonus
                           ↓ (coop_weight = 0.05)   ↓ (comp_weight = 0.1)
```

Our pseudo-market: 2 assets, one step, no comission, no impact

- prices at t: $p_t = [100, 50]$
- prices at $t+1$: $p_{t+1} = [101, 49]$
- each agent starts w cash = 100,000; holdings = [0, 0]
- (example) agents weights at t:
  - mom_0: [0.5, 0.5]
  - mean_0: [0.3, 0.7]
  - mm_0: [0.05, 0.95]
  - rl_trader_0: [0.6, 0.4]

A. converting weights to trades

- target_dollar (in each asset) = weights * total_portfolio_value at time t

- For PPO (rl_trader_0):
  - total V_t (value at t) = 100,000
  - target dollars = [60,000, 40,000]
  - Current $ in assets (prior to trading) = [0, 0]
  - dollar_diff = [60k, 40k]
  - qty to buy = (dollar_diff)/(price_t) = $\left[\frac{60k}{100}, \frac{40k}{50}\right]$
    = [600, 800]

• Could then do this for all other agents

agent	weights	target $	qty at t (p=[100,50])
mom_0	[0.5, 0.5]	[50k, 50k]	[500, 1000]
mean_0	[0.3, 0.7]	[30k, 70k]	[300, 1400]
mm_0	[0.05, 0.95]	[5k, 95k]	[50, 1900]
rl_trader_0	[0.6, 0.4]	[60k, 40k]	[600, 800]

• In code/practice, would include a small price impact based on impact_coeff and liquidity_scale and then exec_price = prev_prices * impact. For hard calc → impact = 1.0

• Post trade, holdings & cash → mo holdings = [600, 800]

cash = [100,000 - 60,000 - 40000] = 0

B

• Environment advances to t+1 and computes base profit and loss (P&L) record

• Code:

prev_val = cash + holdings * p_t

new_val = cash + holdings * p_{t+1}

base = new_val - prev_val

PPO:

- prev_val = 0.4 [600, 800] · [100, 50] = 60,000 + 40,000 = 100,000
- new_val = 0.4 [600, 800] · [101, 49] = 60,600 + 39,200 = 99,800
- base = 99,800 − 100,000 = −200 → PPO agent lost $200
  - Had 40% in asset that fell

Other agents:

mom_0: . prev = 100k, new = (500·101) + (1000·40) = 99,500, base = −500

mean_0: new = 98,900 | base = −1100

mm_0: new = 98,150 | base = −1850

Raw P&L Rankings: PPO > mom > mean > mm

C

- Record Shaping
  - Stats using new values V_{t+1}:
    - new_vals:
      - mom: 99,800
      - mean: 98,900
      - mm: 98,150
      - PPO: 99,800

$\sigma = \sqrt{\frac{\Sigma(x-\mu)^2}{n}}$

- mean_vals = (99500 + 98900 + 98150 + 99800)/4
  = 99,087.5
- deviations:
  [412.5, -187.5, -937.5, 712.5] → $(x-\mu)$
- Squares:
  [170,156, 35,156, 878,906, 507,656]
- mean square =
  (1,591,875)/4 = 397,968.75

- std = $\sqrt{397968.75}$ = 631.0 (σ)

- Bonus given by PPO:
  - coop_bonus = 0.05 · $\left(\frac{1}{1+std}\right)$ = 0.05 · $\frac{1}{632}$ = 0.000079
  - Comp_bonus = 0.1 $\left(\frac{mean\_val - ppo\_val}{std}\right)$
    = 0.1 $\left(\frac{-712.5}{631}\right)$ ≈ -0.113

- Final Reward:
  r_t(ppo) = base + coop_bonus - comp_bonus
  ≈ -200 + 0.000079 · (-0.113)
  ≈ -199.89

- (Subtracting comp because comp calculated as negative, so to cancel out)

D

- Returns & Advantages
  - Discount $\gamma = 0.99$
  - GAE parameter $\lambda = 0.95$
  - $\hat{A}_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ ↙ Advantage Function

Model Outputs for Critic:

- $V(s_t) = \hat{V}_t = +100.0$ (Remember, critic outputs reward expectation, so this indicates small positive)
- $V(s_{t+1}) = \hat{V}_{t+1} = +98.0$

TD-error δ_t → Temporal Difference (discrepancy between current reward exAcc. and next step)

- $\delta_t = r_t + \gamma \hat{V}_{t+1} - \hat{V}_t = -199.887 + 99.02 - 100 = -202.867$

GAE advantage A_t: For one step $A_t \approx \delta_t$, (longer rollouts: $\gamma \delta_{t+1} + \ldots$)

So in: $A_t \approx -202.867$

Actual Return ↷ Monte-Carlo Return R_t: $R_t = r_t + \gamma \hat{V}_{t+1} = -199.887 + 97.02 = -102.867$

- Critics loss will later try to fit $\hat{V}_t \rightarrow R_t$ (expected returns → Actual returns) (Through NN Backprop)

E: Log Probs, Ratios and PPO Objective

- **Calculating Updates**
  - Assuming 2-dimension action (Pre-squashing), independent Gaussians.
  - At s_t, the old policy - which generated the trajectory - had:
    - means: $\mu_{old} = [0.55, 0.45]$
    - Std's $\sigma_{old} = [0.10, 0.10]$
  - Sample Action to produce weights initially: $a_t = [0.62, 0.41]$
  - Current policy (after one gradient step):
    means $\mu = [0.58, 0.43]$
    Stds $\sigma = [0.10, 0.10]$ (same for ex)
  - Using log probability of a diagonal gaussian

$$? \rightarrow \underbrace{r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}}_{\text{Probability ratio}} \rightarrow \log r_t(\theta) = \log \pi_\theta(a_t|s_t) - \log \pi_{\theta_{old}}(a_t|s_t)$$

- $\log \pi(a|s) = \sum_i \left[-\frac{1}{2}\left(\frac{a_i - \mu_i}{\sigma_i}\right)^2 - \log\left(\sqrt{2\pi}\,\sigma_i\right) \right]$

- Per dim ($a_t = [0.62, 0.41]$ $\mu_{old} = [0.55, 0.45]$)
  - old:

dim1 • $(0.62 - 0.55)/0.1 = 0.7 \rightarrow -\frac{1}{2}(0.7)^2 = -0.245$

dim2 • $(0.41 - 0.45)/0.1 = -0.4 \rightarrow -\frac{1}{2}(0.4)^2 = -0.08$

Same (constants) • $-\log(\sqrt{2\pi} \cdot 0.1) \approx -\log(0.25006) \approx +1.383$

• Sum: $(0.245 + 1.383) + (0.08 + 1.383)$

$\hookrightarrow = 2.441$

- Current:
  - dim1: $(0.62 - 0.58)/0.1 = 0.4 \xrightarrow{-\frac{1}{2}(0.4)^2} -0.08$
  - dim2: $(0.41 - 0.43)/0.1 = -0.2 \xrightarrow{"} -0.02$
  - Same constants
  - sum: $(0.08 + 1.383) + (0.02 + 1.383) = 2.666$

↳ $\log \pi_{old}(a_t|s_t) = 2.441 \neq \log \pi(a_t|s_t) = 2.666$

$e^{\wedge}(\log r_t(\theta)) \rightarrow r_t(\theta) = e^{\wedge}(\log \pi - \log \pi_{old}) = \exp(2.666 - 2.441)$

$= \exp(0.225) \approx 1.252$

↳ $\sigma_t(\theta) = 1.252$

Clipping PPO Objective:

$$L_t^{CLIP} = \min(r_t A_t, \text{clip}(r_t, 1-\epsilon, 1+\epsilon) \cdot A_t), \epsilon = 0.2$$

- $A_t = -202.867$
- $r_t A_t = 1.252(-202.867) = -254.39$
- clip r_t to $[\underbrace{0.8, 1.2}_{-0.2, +0.2}] \rightarrow r_t^{clipped} = 1.2$
- clipped term = $1.2 * (-202.867) = -243.44$

pessimistic → • Taking min:

$r_t A_t$ ↓ , clip ↓

$$L_t^{CLIP} = \min(-254.39, -243.44)$$

$$L_t^{CLIP} = -254.39$$

- Because $A_t < 0$, increasing r_t beyond the clip hurts the surrogate more → clipping protects against overly large policy changes.

- In a batch, w/ multiple results, PPO sums L_t over timesteps and takes gradient ascent. we're only doing 1 step.

F

- Critic Loss

↳ $L_t^{VF} = (R_t - \hat{V}_t)^2 = (-102.867 - 100)^2 = (-202.867)^2 \approx 41,156$

- entropy → encourages exploration → more risk bonus
  ↳ more reward?

↳ $L_t^{ENT} = \beta \cdot \mathcal{H}[\pi(\cdot|s_t)]$ with small weight β (0.0-0.1)

- For gaussian (like our distribution:
  - $\mathcal{H} = \sum_i \frac{1}{2} \log(2\pi e \sigma_i^2)$
  - $\sigma_i = 0.1 \rightarrow \mathcal{H}$ is a constant per dimension

- Total PPO objective:

$$J(\theta, \phi) = \mathbb{E}\left[L^{CLIP}(\theta) c_v L^{VF}(\phi) c_e \mathcal{H}(\theta)\right]$$

↳ (c_v and c_e are coefficients)

↳ puts everything together.

Optimizer updates:

- actor params θ to increase $L^{CLIP} + c_e H$
- critic params ϕ to decrease loss L^{VF}

→ • Advantage is −, PPO decreases (log) prob. of action taken

• critic lowers $V(s_t)$ because $-102.9 < +100$, (reality < expectation), over time $V(s_t)$ becomes more accurate)

• PPO beat mean, so "comp" made reward less negative

• GAE can blend many δ_t so policy learns from patterns rather than days