Functional Prototype Demonstration 2

Team Epsilon-Greedy Quants
Michael Lee, Nikat Patel, Jose Antonio Alatorre Sanchez

What we did in this milestone?

We implemented the Policy Gradient algorithms REINFORCE with baseline and Actor-Critic. We also refactored our model code to work in PyTorch.

Presentation Overview

- REINFORCE with Baseline Summary
- Actor-Critic Summary
- Model Performance Overview
- Discussion of Problems Encountered
- Code Documentation/Organization
- Next Steps

REINFORCE Summary

Policy Gradient Method

- Estimates Policy directly, not from Action-Value function
- Continuous action space

REINFORCE

- Performance under Policy-Gradient Theorem: $\nabla J(\theta) \propto \sum_s \mu(s) \sum_a (q_\pi(s, a)) \nabla \pi(a|s, \theta)$
- Relies on estimated return by Monte-Carlo method
- Uses episode samples to update policy parameter θ
- **High variance results in slow learning**

REINFORCE with Baseline

- Compares the action-value to an arbitrary baseline b(s)
  - Performance under Policy-Gradient Theorem:
    $\nabla J(\theta) \propto \sum_s \mu(s) \sum_a (q_\pi(s,a) - b(s)) \nabla \pi(a|s,\theta)$
  - Can be any function or random variable as long as it does not vary with action **a**
  - Commonly used baseline: state value function $\hat{v}(S_t, w)$
  - Policy parameter θ is updated using baseline:
    $\theta_{t+1} = \theta_t + \alpha(G_t - b(S_t)) \frac{\nabla \pi(A_t|S_t,\theta_t)}{\pi(A_t|S_t,\theta_t)}$
- Baseline functions don't change expected value of update but **can reduce the variance** (speed up learning)

REINFORCE with Baseline Algorithm Steps

From Sutton and Barto, Chapter 13.4

Steps:

- Initialize the policy parameter $\boldsymbol{\theta}$ and state-value weights **w** at random.
- Loop forever (for each episode):
    - Generate one episode using policy π_θ : $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$.
    - Loop for each step of the episode t=0,1,...,T-1:
        - Estimate the return $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
        - **Calculate the delta between $\boldsymbol{G}$ and baseline function:** $\boldsymbol{\delta \leftarrow G - \hat{v}(S_t, w)}$
        - **Update the state-value weights:** $\boldsymbol{w \leftarrow w + \alpha^w \delta \nabla \hat{v}(S_t, w)}$
        - Update policy parameters: $\theta \leftarrow \theta + \alpha^\theta \gamma_t \delta \nabla ln\pi(A_t | S_t, \theta)$
            - α^θ - stepsize
            - γ - discount factor
            - $\nabla ln\pi(A_t | S_t, \theta)$ - eligibility vector: gradient of the probability of taking action A_t given a state S_t and policy π_θ

Actor-Critic Methods

In REINFORCE with baseline, the learned state-value function estimates the value of the only the first state of each state transition. This estimate sets a baseline for the subsequent return, but is made prior to the transition's action and thus cannot be used to assess that action. In actor-critic methods, on the other hand, the state-value function is applied also to the second state of the transition. The estimated value of the second state, when discounted and added to the reward, constitutes the one-step return, $G_{t:t+1}$ which is a useful estimate of the actual return and thus is a way of assessing the action.

When the state-value function is used to assess actions in this way it is called a critic, and the overall policy-gradient method is termed an actor-critic method. Note that the bias in the gradient estimate is not due to bootstrapping as such; the actor would be biased even if the critic was learned by a Monte Carlo method.

One-step Actor-critic

One-step actor-critic methods replace the full return of with the one-step return (and use a learned state-value function as the baseline)as follows

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(G_{t:t+1} - \hat{v}(S_t, \boldsymbol{w})\frac{\nabla\pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(R_{t+1}\gamma\hat{v}(S_{t+1}, \boldsymbol{w}) - \hat{v}(S_t, \boldsymbol{w})\frac{\nabla\pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})}$$

The main appeal of one-step methods is that they are fully online and incremental, yet avoid the complexities of eligibility traces. They are a special case of the eligibility trace methods, but easier to understand

One step pseudo code:

One-step Actor–Critic (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
    Initialize S (first state of episode)
    $I \leftarrow 1$
    Loop while S is not terminal (for each time step):
        $A \sim \pi(\cdot|S, \boldsymbol{\theta})$
        Take action A, observe S', R
        $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S, \boldsymbol{\theta})$
        $I \leftarrow \gamma I$
        $S \leftarrow S'$

Eligibility Traces Actor- Critic

The generalizations to the forward view of-step methods and then to a λ-return algorithm are straightforward. The one-step return in (1) is merely replaced by $G_{t:t+1}$ or Gt^{λ} respectively. The backward view of the λ-return algorithm is also straightforward, using separate eligibility traces for the actor and critic. Pseudocode for the complete algorithm is given in the box below

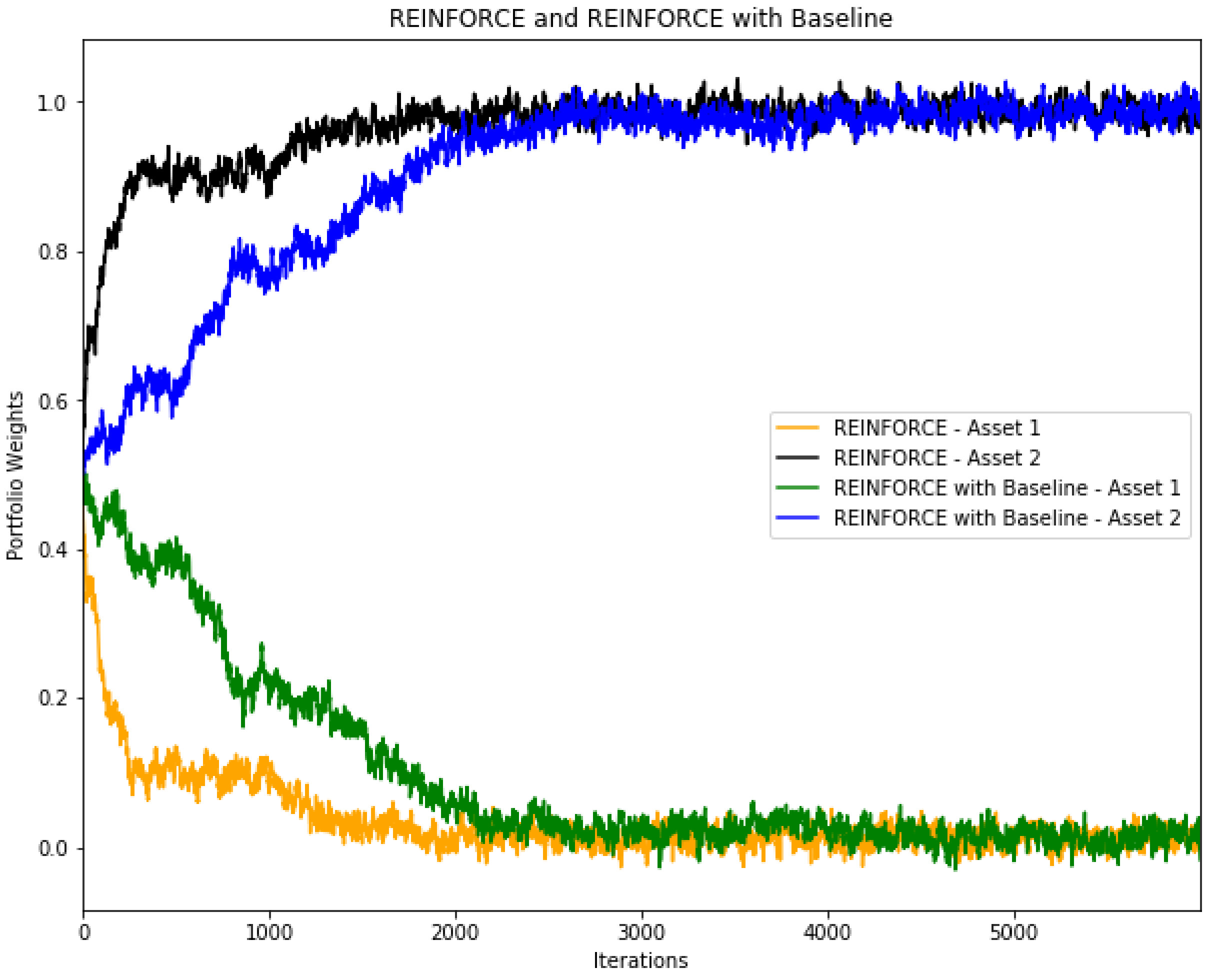
Actor–Critic with Eligibility Traces (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: trace-decay rates $\lambda^{\boldsymbol{\theta}} \in [0,1]$, $\lambda^{\mathbf{w}} \in [0,1]$; step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
    Initialize S (first state of episode)
    $\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \mathbf{0}$ (d'-component eligibility trace vector)
    $\mathbf{z}^{\mathbf{w}} \leftarrow \mathbf{0}$ (d-component eligibility trace vector)
    $I \leftarrow 1$
    Loop while S is not terminal (for each time step):
        $A \sim \pi(\cdot|S, \boldsymbol{\theta})$
        Take action A, observe S', R
        $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
        $\mathbf{z}^{\mathbf{w}} \leftarrow \gamma \lambda^{\mathbf{w}} \mathbf{z}^{\mathbf{w}} + \nabla \hat{v}(S,\mathbf{w})$
        $\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \gamma \lambda^{\boldsymbol{\theta}} \mathbf{z}^{\boldsymbol{\theta}} + I \nabla \ln \pi(A|S, \boldsymbol{\theta})$
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \mathbf{z}^{\mathbf{w}}$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \delta \mathbf{z}^{\boldsymbol{\theta}}$
        $I \leftarrow \gamma I$
        $S \leftarrow S'$

REINFORCE Results

REINFORCE - { Orange, Black }

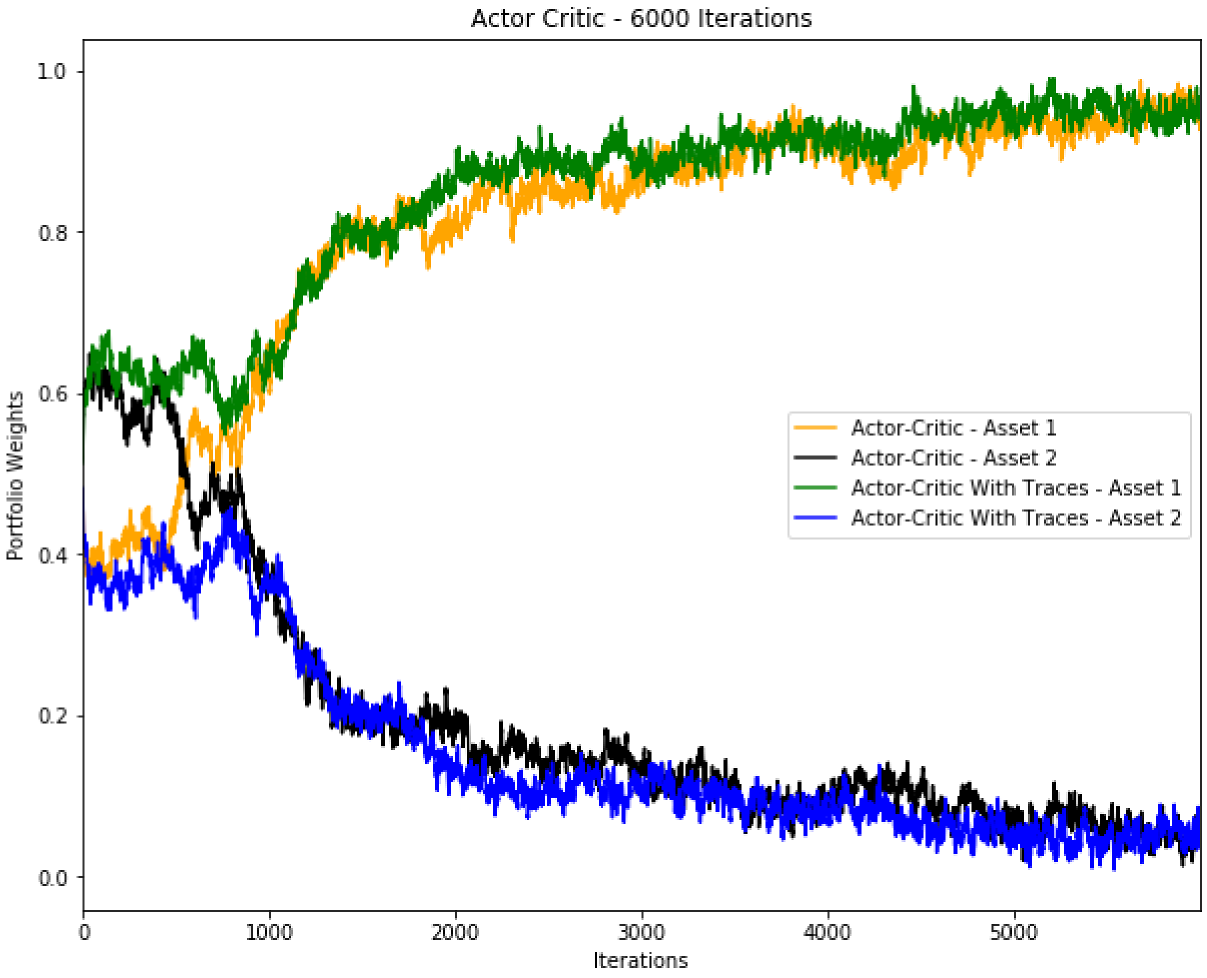
REINFORCE with Baseline - { Green, Blue }



Actor Critic Results

Actor Critic Without Eligibility Traces - { Orange, Black }

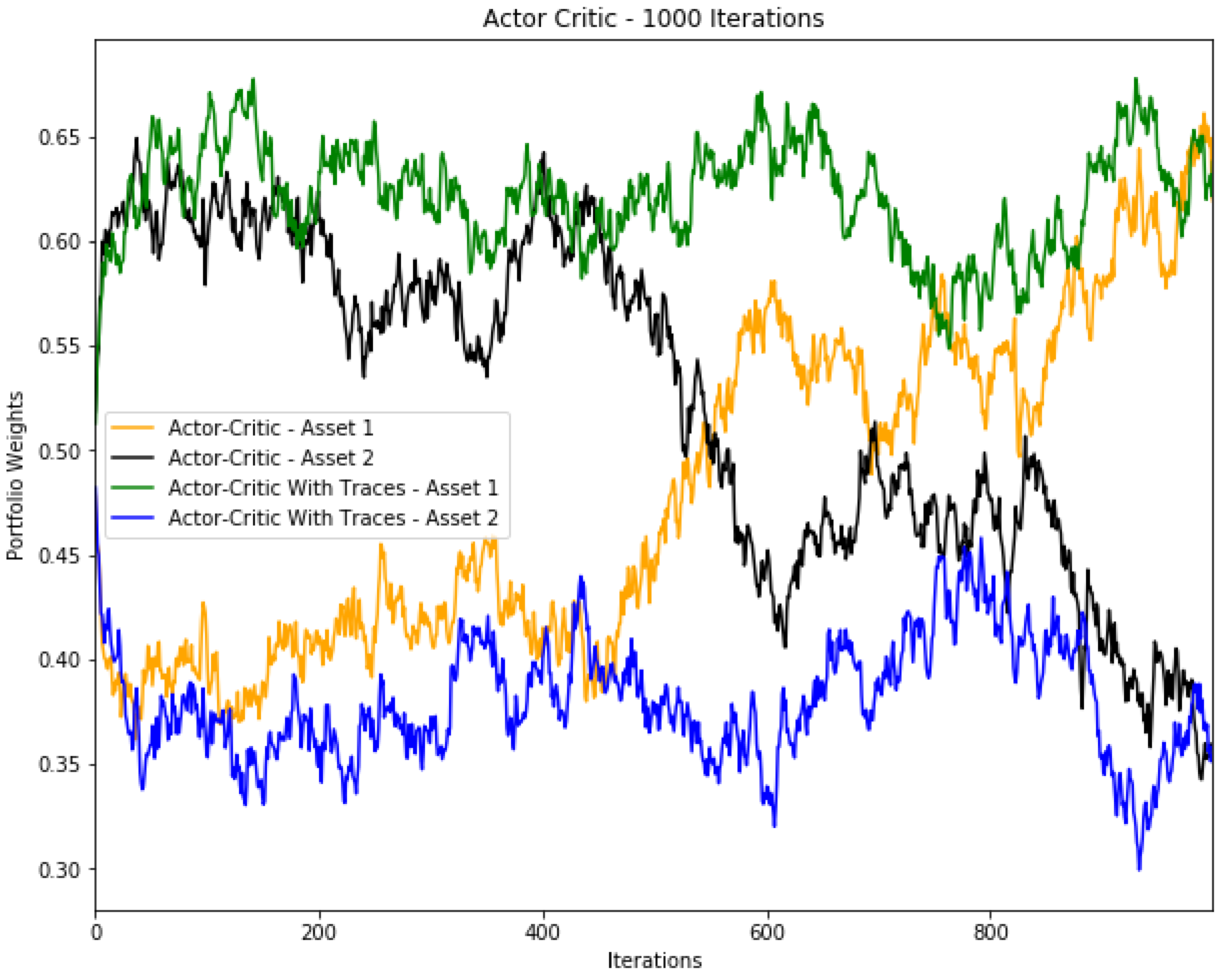
Actor Critic With Eligibility Traces - { Green, Blue }



Actor Critic Results

Actor Critic Without Eligibility Traces - { Orange, Black }

Actor Critic With Eligibility Traces - { Green, Blue }



Problems Encountered with TensorFlow

- Issue: TensorFlow REINFORCE model would not converge
  - Used tensorflow.GradientTape() for automatic differentiation
  - Experiemented with various keras optimizers

- Solution: Refactored Environment to support PyTorch
  - PyTorch REINFORCE model converged much faster than our standard REINFORCE model

Version Control Repository

- We have created a private GitHub repository that contains all our data, documentations, code, and notebooks.
- We use version control to develop, update, and collaborate our work.

```
├── README.md
├── data
├── data_env
│   ├── ief.parquet
│   └── spy.parquet
├── lib
│   ├── Benchmarks.py
│   ├── DataHandling.py
│   ├── Environment.py
│   ├── Environment_refactored.py
│   ├── __init__.py
├── notebooks
│   ├── Benchmark_EDA.ipynb
│   ├── ENVIRONMENT.ipynb
│   ├── ACTOR_CRITIC.ipynb
│   ├── REINFORCE_BASELINE.ipynb
│   ├── REINFORCE.ipynb
│   ├── data_handler_test.ipynb
│   ├── environment_gymai.ipynb
│   └── sharpe_sample.ipynb
├── static
```

Next Steps

- Experiment with Various Reward Functions to observe differences
    - Sortino Ratio to Control Drawdowns. Increase weight of negative rewards if drawdown reaches a certain threshold.
- Begin Testing our Models using Real-World DataSets.
- Begin Capstone Documentation.