POLITECNICO
MILANO 1863

Reinforcement Learning for Automated Trading

Pierpaolo Necchi
pierpaolo.necchi@gmail.com

September 21, 2016

Plan

Markov Decision Processes

Reinforcement Learning

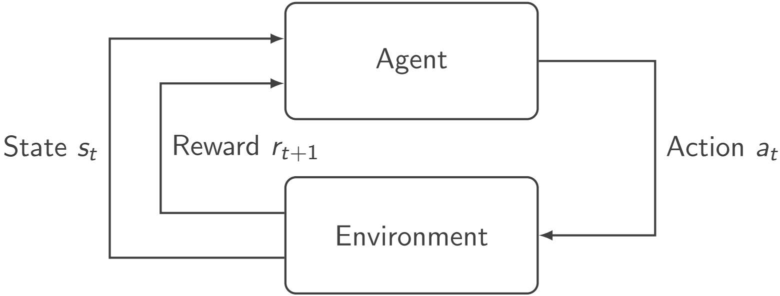
General class of algorithms that allow an agent to learn how to behave in a stochastic and possibly unknown environment by trial-and-error.

Markov Decision Process (MDP)

stochastic dynamical system specified by $< \mathbb{S}, \mathbb{A}, \mathcal{P}, \mathcal{R}, \gamma >$

1. $(\mathbb{S}, \mathcal{S})$ is a measurable state space
2. $(\mathbb{A}, \mathcal{A})$ is a measurable action space
3. $\mathcal{P} : \mathbb{S} \times \mathbb{A} \times \mathcal{S} \to \mathbb{R}$ is a Markov transition kernel
4. $\mathcal{R} : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ is a reward function
5. $0 < \gamma < 1$ is the discount factor.

Interaction Between Agent and Environment



Policy and Value Function

Policy

A policy is a function $\pi : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ such that, $\forall s \in \mathbb{S}$, $C \mapsto \pi(s, C)$ is a probability distribution over $(\mathbb{A}, \mathcal{A})$

Return

$$G_t = \sum_{t=0}^{\infty} \gamma^t R_{t+k+1}$$

Value Function

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t | S_t = s \right]$$

Agent's goal

Select a policy π^* that maximizes his expected return in all possible states. This policy is called *optimal*.

Policy Gradient Methods

Key idea

π_* is approximated using a parametrized policy $\pi_{\theta^*}(s, a)$, where

$$\theta^* = \arg\max_{\theta \in \Theta} J(\theta) = V_{\pi_\theta}(s_0)$$

Using gradient descent, we have the following update scheme

$$\theta_{k+1} = \theta_k + \alpha_k \nabla_\theta J(\theta_k)$$

Policy Gradient Theorem

Let π_θ be a differentiable policy. The policy gradient for the average reward formulation is given by

$$\nabla_\theta J(\theta) = \mathbb{E}_{\substack{S \sim d^\theta \\ A \sim \pi_\theta}} \left[\nabla_\theta \log \pi_\theta(S, A) Q_\theta(S, A) \right]$$

d^θ is the stationary distribution of the Markov chain induced by π_θ.

Monte-Carlo Policy Gradient Method

Algorithm 1 GPOMDP

Input:

- Initial policy parameters $\theta_0 = (\theta_0^1, \ldots, \theta_0^{D_\theta})^T$
- Learning rate $\{\alpha_k\}$
- Number of trajectories M

Output: Approximation of the optimal policy $\pi_{\theta^*} \approx \pi_*$

1: Initialize $k = 0$
2: **repeat**
3: Sample M trajectories $h^{(m)} = \{(s_t^{(m)}, a_t^{(m)}, r_{t+1}^{(m)}\}_{t=0}^{T^{(m)}}$ of the MDP under policy π_{θ_k}
4: Compute the optimal baseline

$$\widehat{b}_k^n = \frac{\sum_{m=1}^{M} \left[\sum_{i=0}^{T^{(m)}} \partial_{\theta_k} \log \pi_\theta \left(s_i^{(m)}, a_i^{(m)}\right)\right]^2 \sum_{j=0}^{T^{(m)}} \gamma^j r_{j+1}^{(m)}}{\sum_{m=1}^{M} \left[\sum_{i=0}^{T^{(m)}} \partial_{\theta_k} \log \pi_\theta \left(s_i^{(m)}, a_i^{(m)}\right)\right]^2}$$

5: Approximate policy gradient

$$\frac{\partial}{\partial \theta^n} J_{\text{start}}(\theta_k) \approx \widehat{g}_k^n = \frac{1}{M} \sum_{m=1}^{M} \sum_{i=0}^{T^{(m)}} \frac{\partial}{\partial \theta^n} \log \pi_{\theta_k} \left(s_i^{(m)}, a_i^{(m)}\right) \left(\sum_{j=i}^{T^{(m)}} \gamma^j r_{j+1}^{(m)} - \widehat{b}_k^n\right)$$

6: Update actor parameters $\theta_{k+1} = \theta_k + \alpha_k \widehat{g}_k$.
7: $k \leftarrow k + 1$
8: **until** converged

Asset Allocation with Transaction Costs

Goal

How to dynamically invest the available capital in a portfolio of different assets in order to maximize the expected total return or another relevant performance measure.

Reward function: portfolio log-return with transaction costs

$$R_{t+1} = \log\left\{1 + \sum_{i=0}^{I}\left[a_t^i X_{t+1}^i - \delta_i\left|a_t^i - \tilde{a}_t^i\right| - \delta_s (a_t^i)^-\right] - \delta_f \mathbf{1}_{a_t \neq \tilde{a}_{t-1}}\right\}$$

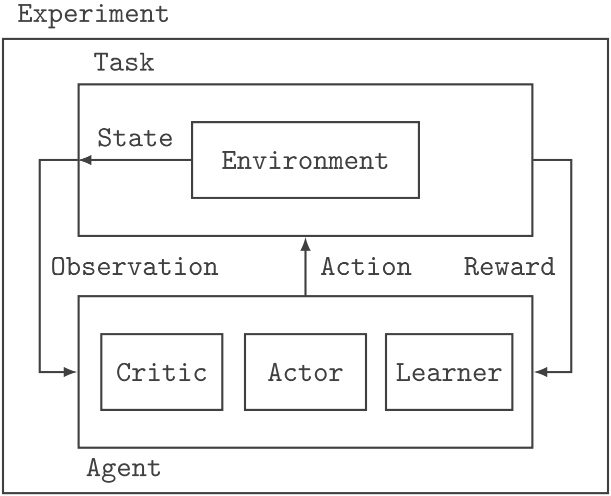
Actions: Portfolio weights

$$\{a_t^i\}_{i=0}^{I} \quad \text{s.t.} \quad \sum_{i=0}^{I} a_t^i = 1 \qquad \forall t \in \{0, 1, 2, \ldots\}$$

State: assets past returns and current allocation

$$S_t = \{X, X_t, X_{t-1}, \ldots, X_{t-P}, \tilde{a}_t\}$$

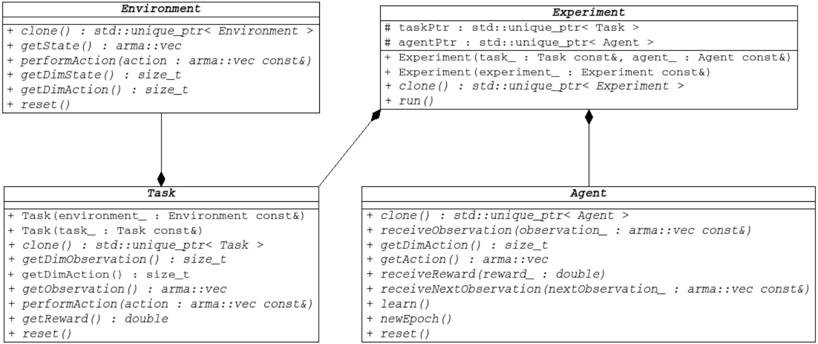
PyBrain's Architecture for a RL Problem



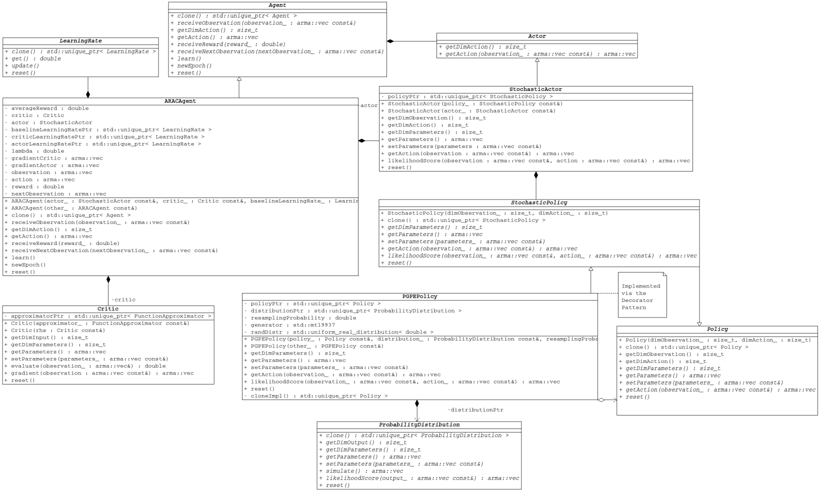
Agent-Environment Interaction in C++

Adapting PyBrain's Architecture

1. Defined standard interfaces via pure abstract classes
2. Achieved modularity via polymorphic composition



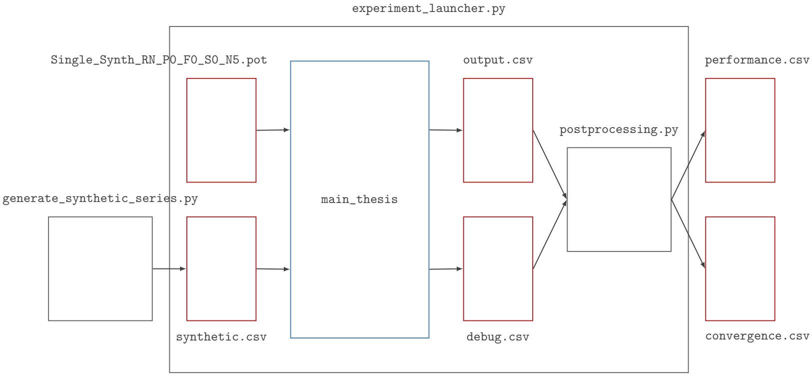
Agent's Architecture in C++



Execution Pipeline

experiment_launcher.py

1. Program execution is handled by a Python script
2. Responsible for analyzing the output of the C++ engine



Synthetic Asset

Goal

Evaluate different RL algorithms in a controlled environment, i.e. on a synthetic asset with profitably tradable features

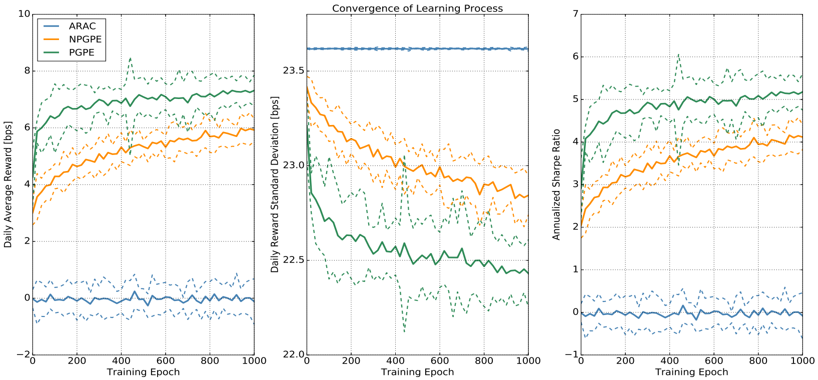
The synthetic asset price is given by

$$Z_t = \exp\left(\frac{z_t}{\max_t z_t - \min_t z_t}\right)$$

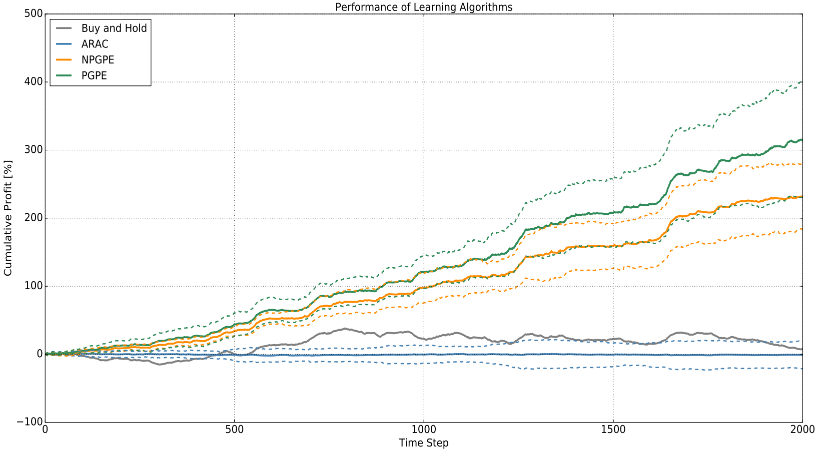
where $\{z_t\}$ is a random walk with autoregressive trend $\{\beta_t\}$

$$\begin{aligned} z_t &= z_{t-1} + \beta_{t-1} + \kappa\epsilon_t \\ \beta_t &= \alpha\beta_{t-1} + \nu_t \end{aligned}$$

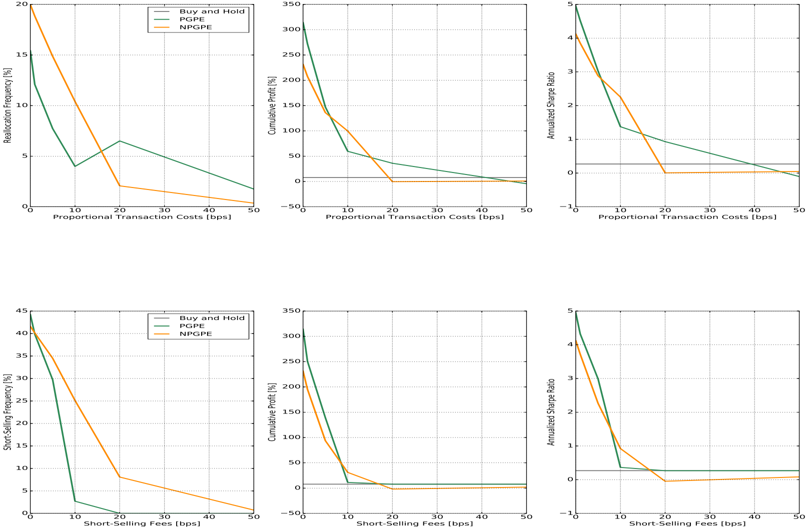
Convergence of RL algorithms



Backtest Performance of the Trading Strategies Learned



Impact of Transaction Costs



Conclusions and Future Developments

Conclusion

1. Applied state-of-the-art RL algos to find a profitable long-short trading strategy
2. RL strategies outperform the simple B&H for a synthetic asset
3. RL strategies are able to adapt to transaction costs
4. RL seems suitable to deal with many financial decision problems

Future Developments

1. Improve the algos performance on historical data
2. Developing more complex features for the trading strategy
3. Apply RL techniques to other financial problems

Thank you for your attention!

References

Joshi, M. S. (2008).
C++ design patterns and derivatives pricing, volume 2.
Cambridge University Press.

Moody, J., Wu, L., Liao, Y., and Saffell, M. (1998).
Performance functions and reinforcement learning for trading systems and portfolios.
Journal of Forecasting, 17:441–470.

Peters, J. and Schaal, S. (2008).
Reinforcement learning of motor skills with policy gradients.
Neural networks, 21(4):682–697.

Sutton, R. S. and Barto, A. G. (1998).
Introduction to reinforcement learning, volume 135.
MIT Press Cambridge.