



**POLITECNICO**  
MILANO 1863

## Policy Gradient Algorithms for the Asset Allocation Problem

Pierpaolo Necchi  
[pierpaolo.necchi@gmail.com](mailto:pierpaolo.necchi@gmail.com)

December 18, 2016

# Animal Spirits



# The Sound of Silence



1. Basics of Reinforcement Learning
2. Policy Gradient Algorithms
3. Asset Allocation with Transaction Costs
4. Conclusions

1. Basics of Reinforcement Learning
2. Policy Gradient Algorithms
3. Asset Allocation with Transaction Costs
4. Conclusions

# The Reinforcement Learning Framework

# The Reinforcement Learning Framework

Environment



$\mathcal{P}, \mathcal{R}$

# The Reinforcement Learning Framework

Agent



$\pi$

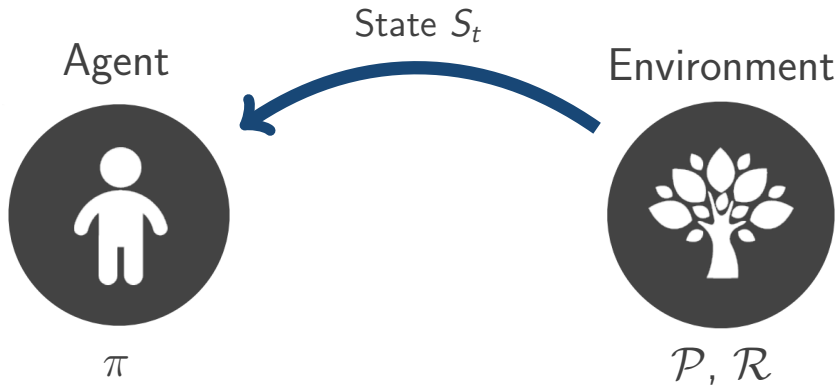
Environment



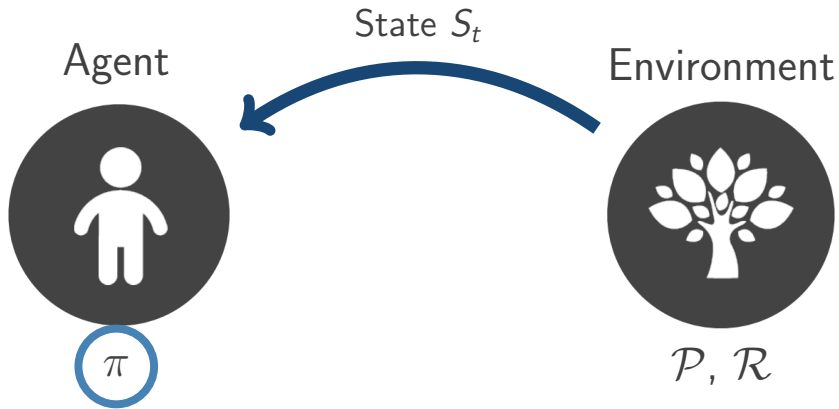
$\mathcal{P}, \mathcal{R}$



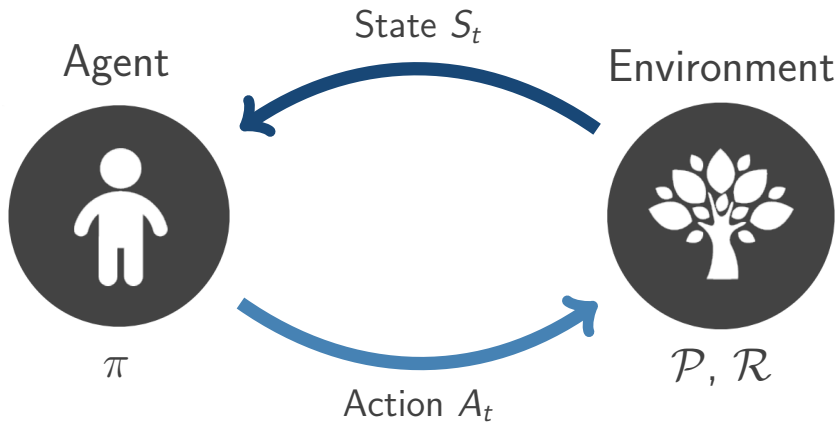
# The Reinforcement Learning Framework



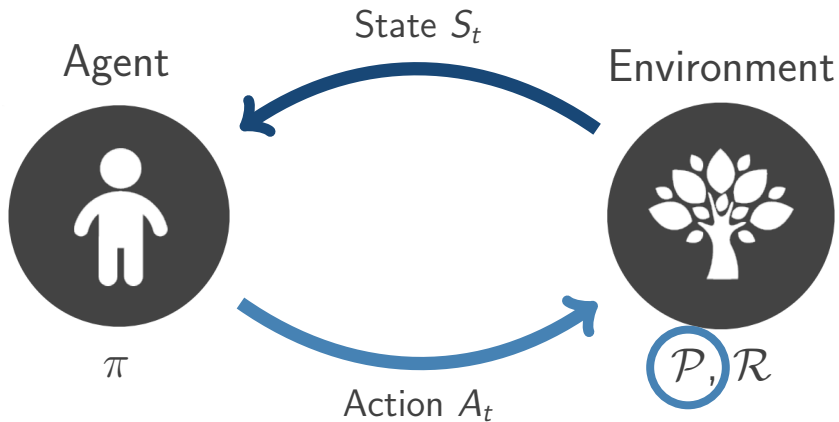
# The Reinforcement Learning Framework



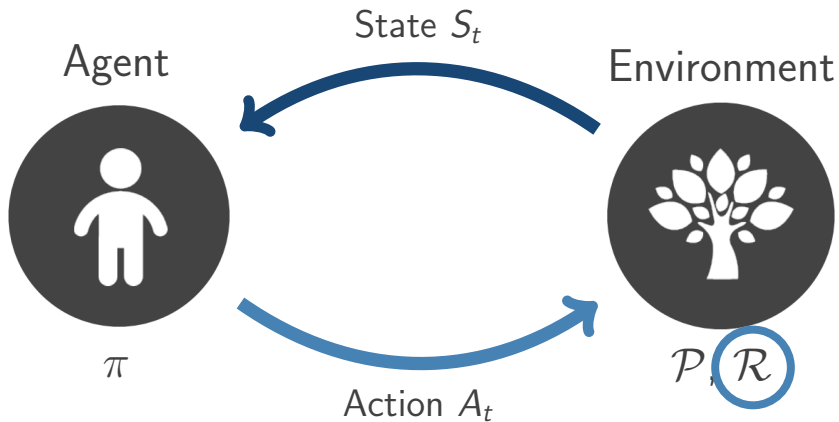
# The Reinforcement Learning Framework



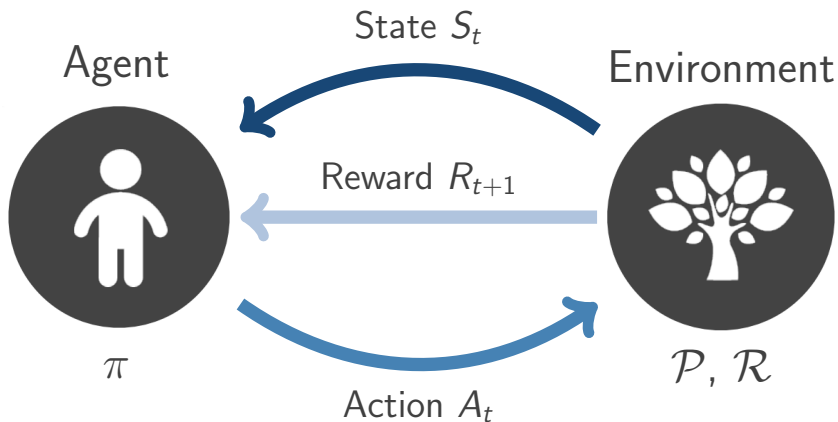
# The Reinforcement Learning Framework



# The Reinforcement Learning Framework



# The Reinforcement Learning Framework



## State-value function

$$V_{\pi}(s) = \mathbb{E}_{\pi} \left[ \sum_{t=0}^{\infty} \gamma^t R_{t+1} \middle| S_0 = s \right]$$

# Mathematical Formulation

## State-value function

$$V_{\pi}(s) = \mathbb{E}_{\pi} \left[ \sum_{t=0}^{\infty} \gamma^t R_{t+1} \middle| S_0 = s \right]$$

## Action-value function

$$Q_{\pi}(s, a) = \mathbb{E}_{\pi} \left[ \sum_{t=0}^{\infty} \gamma^t R_{t+1} \middle| S_0 = s, A_0 = a \right]$$



# Mathematical Formulation

## State-value function

$$V_{\pi}(s) = \mathbb{E}_{\pi} \left[ \sum_{t=0}^{\infty} \gamma^t R_{t+1} \middle| S_0 = s \right]$$

## Action-value function

$$Q_{\pi}(s, a) = \mathbb{E}_{\pi} \left[ \sum_{t=0}^{\infty} \gamma^t R_{t+1} \middle| S_0 = s, A_0 = a \right]$$

## Optimal value functions

$$V_*(s) = \max_{\pi} V_{\pi}(s) \quad Q_*(s, a) = \max_{\pi} Q_{\pi}(s, a)$$

# Mathematical Formulation

## State-value function

$$V_{\pi}(s) = \mathbb{E}_{\pi} \left[ \sum_{t=0}^{\infty} \gamma^t R_{t+1} \middle| S_0 = s \right]$$

## Action-value function

$$Q_{\pi}(s, a) = \mathbb{E}_{\pi} \left[ \sum_{t=0}^{\infty} \gamma^t R_{t+1} \middle| S_0 = s, A_0 = a \right]$$

## Optimal value functions

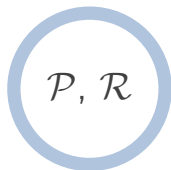
$$V_*(s) = \max_{\pi} V_{\pi}(s) \quad Q_*(s, a) = \max_{\pi} Q_{\pi}(s, a)$$

## Optimal policy

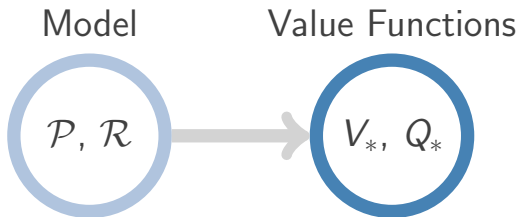
$$\pi_* \text{ s.t. } V_{\pi_*}(s) = V_*(s), \forall s \in \mathbb{S}$$

# Taxonomy of RL Algorithms

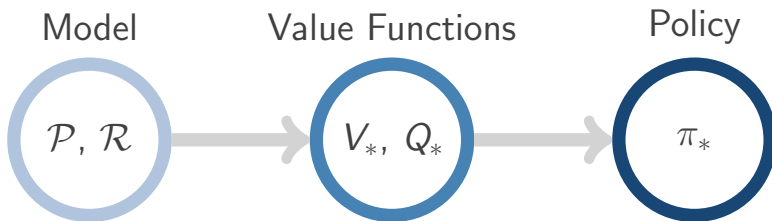
Model



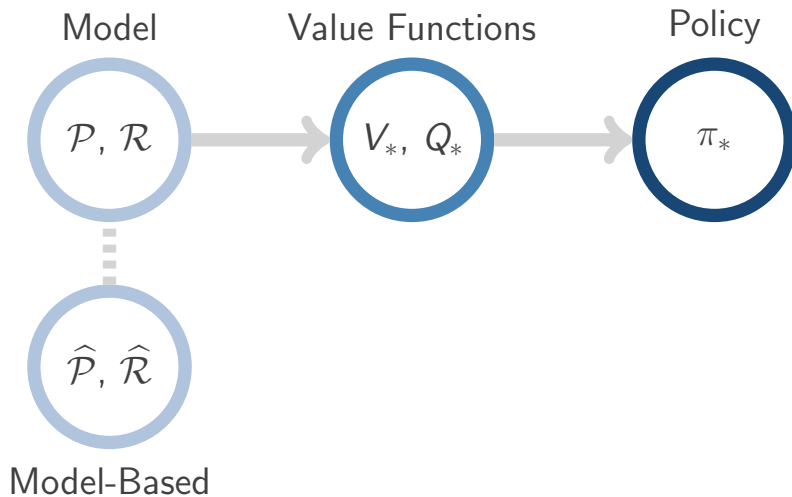
# Taxonomy of RL Algorithms



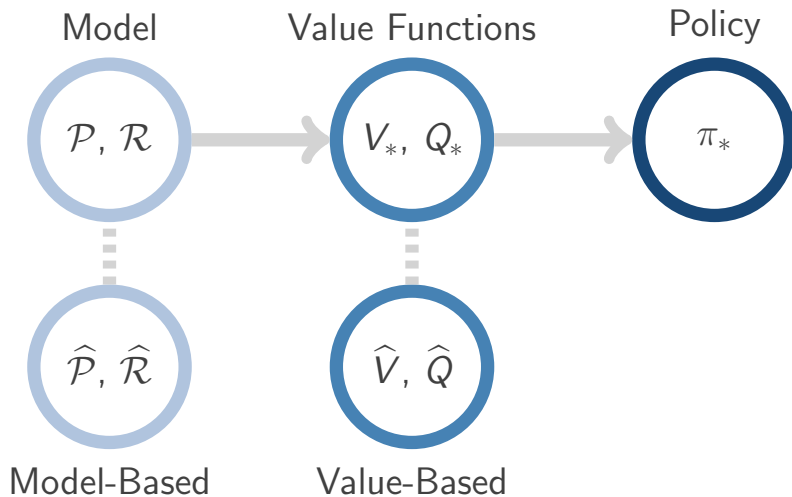
# Taxonomy of RL Algorithms



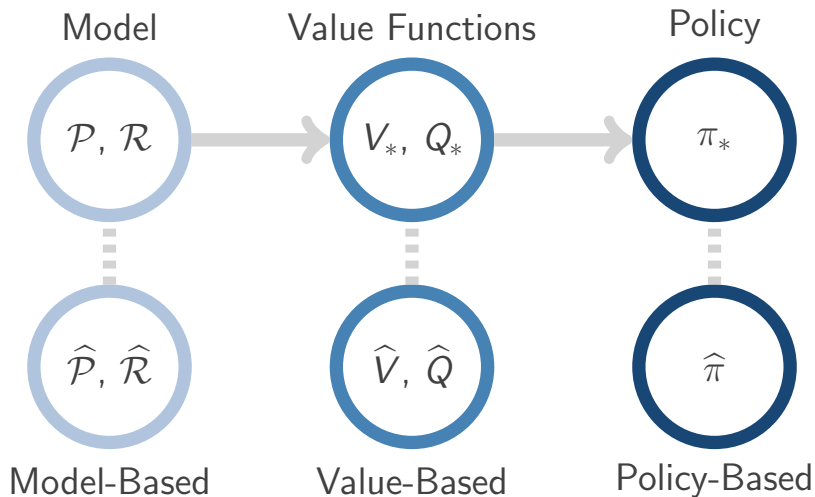
# Taxonomy of RL Algorithms



# Taxonomy of RL Algorithms



# Taxonomy of RL Algorithms





# Plan

1. Basics of Reinforcement Learning
2. Policy Gradient Algorithms
3. Asset Allocation with Transaction Costs
4. Conclusions

## Key Idea

1. The optimal policy  $\pi_*$  is approximated with a parametric policy  $\pi_{\theta^*}$

## Key Idea

1. The optimal policy  $\pi_*$  is approximated with a parametric policy  $\pi_{\theta^*}$
2. The parameters  $\theta^*$  of the policy solve the optimization problem

$$\max_{\theta \in \Theta} J(\theta) = V_{\pi_{\theta}}(s)$$

# Policy Gradient Algorithms

## Key Idea

1. The optimal policy  $\pi_*$  is approximated with a parametric policy  $\pi_{\theta^*}$
2. The parameters  $\theta^*$  of the policy solve the optimization problem

$$\max_{\theta \in \Theta} J(\theta) = V_{\pi_\theta}(s)$$

3. Which is solved numerically via gradient descent

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

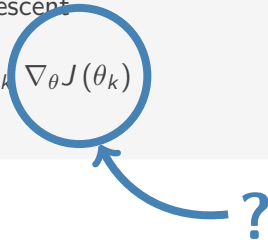
# Policy Gradient Algorithms

## Key Idea

1. The optimal policy  $\pi_*$  is approximated with a parametric policy  $\pi_{\theta^*}$
2. The parameters  $\theta^*$  of the policy solve the optimization problem

$$\max_{\theta \in \Theta} J(\theta) = V_{\pi_\theta}(s)$$

3. Which is solved numerically via gradient descent

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


# The Keystone of Policy Gradient Algorithms

## Policy Gradient Theorem

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

# The Keystone of Policy Gradient Algorithms

## Policy Gradient Theorem

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

For an episodic environment, the policy gradient can be approximated via Monte-Carlo

$$\nabla_{\theta} J(\theta_k) \approx \frac{1}{M} \sum_{m=0}^M \sum_{u=0}^{T^{(m)}-1} \nabla_{\theta} \log \pi_{\theta_k} \left( s_u^{(m)}, a_u^{(m)} \right) \sum_{v \geq u}^{T^{(m)}-1} \gamma^{v-u} r_{v+1}^{(m)}$$

# The Keystone of Policy Gradient Algorithms

## Policy Gradient Theorem

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

For an episodic environment, the policy gradient can be approximated via Monte-Carlo

$$\nabla_{\theta} J(\theta_k) \approx \frac{1}{M} \sum_{m=0}^M \sum_{u=0}^{T^{(m)}-1} \nabla_{\theta} \log \pi_{\theta_k} \left( s_u^{(m)}, a_u^{(m)} \right) \sum_{v \geq u}^{T^{(m)}-1} \gamma^{v-u} r_{v+1}^{(m)}$$

However, this estimate is characterized by a large variance. Possible improvements:

1. Optimal baseline
2. Actor-critic methods
3. Natural gradient



# Policy Gradient with Parameter-Based Exploration (PGPE)

## Key Idea

1. Actions are selected using a deterministic parametric controller  $F_\theta$

# Policy Gradient with Parameter-Based Exploration (PGPE)

## Key Idea

1. Actions are selected using a deterministic parametric controller  $F_\theta$
2. The controller parameters are drawn from a probability distribution  $p_\xi$

# Policy Gradient with Parameter-Based Exploration (PGPE)

## Key Idea

1. Actions are selected using a deterministic parametric controller  $F_\theta$
2. The controller parameters are drawn from a probability distribution  $p_\xi$
3. The search for an optimum is performed in the space of the hyperparameters  $\xi$

# Policy Gradient with Parameter-Based Exploration (PGPE)

## Key Idea

1. Actions are selected using a deterministic parametric controller  $F_\theta$
2. The controller parameters are drawn from a probability distribution  $p_\xi$
3. The search for an optimum is performed in the space of the hyperparameters  $\xi$

More formally, the update scheme becomes

$$\xi_{k+1} = \xi_k + \alpha_k \nabla_\xi J(\xi_k)$$

where the policy gradient is given by

## Parameter-Based Policy Gradient Theorem

$$\nabla_\xi J(\xi) = \mathbb{E}_{\substack{S \sim d^\xi \\ \theta \sim p_\xi}} [\nabla_\xi \log p_\xi(\theta) Q_\xi(S, F_\theta(S))]$$

1. Basics of Reinforcement Learning
2. Policy Gradient Algorithms
3. Asset Allocation with Transaction Costs
4. Conclusions

# Problem Formulation

## Investor's 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.

# Problem Formulation

## Investor's 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.

**Rewards:** portfolio log-return with transaction costs

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

# Problem Formulation

## Investor's 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.

**Rewards:** portfolio log-return with transaction costs

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



# Problem Formulation

## Investor's 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.

**Rewards:** portfolio log-return with transaction costs

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

# Problem Formulation

## Investor's 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.

**Rewards:** portfolio log-return with transaction costs

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

# Problem Formulation

## Investor's 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.

**Rewards:** portfolio log-return with transaction costs

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

# Problem Formulation

## Investor's 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.

**Rewards:** portfolio log-return with transaction costs

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

**Actions:** Portfolio weights

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

# Problem Formulation

## Investor's 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.

**Rewards:** portfolio log-return with transaction costs

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

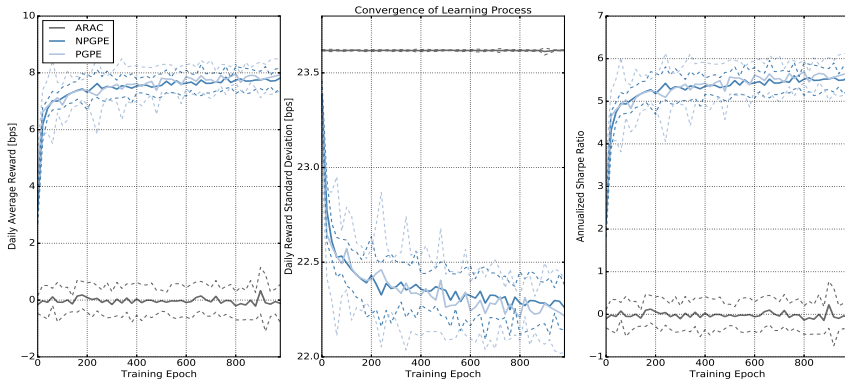
**Actions:** Portfolio weights

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

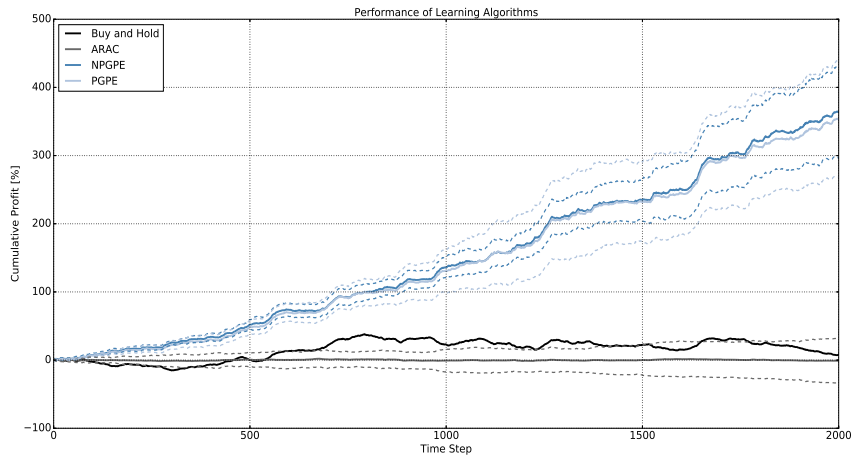
**States:** assets past returns and current allocation

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

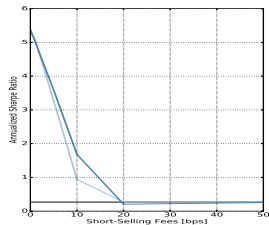
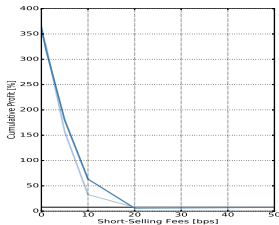
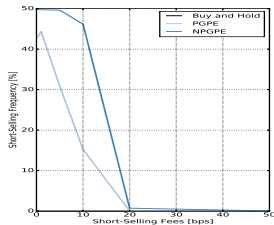
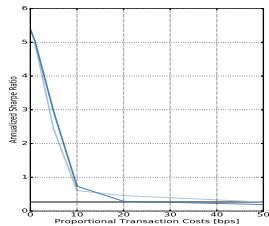
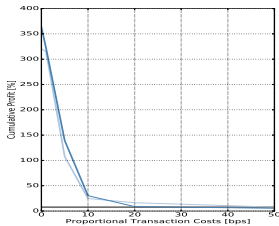
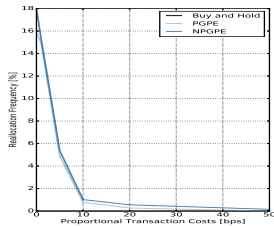
# Synthetic Asset: Convergence



# Synthetic Asset: Backtest Performance



# Synthetic Asset: Impact of Transaction Costs



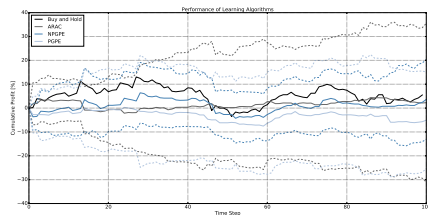


# Not So Fast

## Insuccess on Historical Data

Successfully applying these RL algorithms to historical data is much more challenging

1. Fail to converge
2. The strategies learned are not profitable

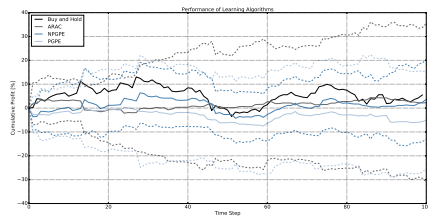


# Not So Fast

## Insuccess on Historical Data

Successfully applying these RL algorithms to historical data is much more challenging

1. Fail to converge
2. The strategies learned are not profitable



## Possible Explanations

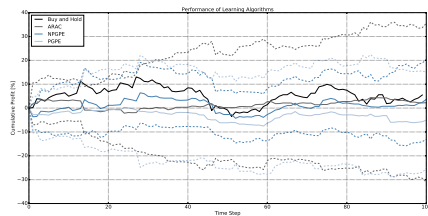
1. **Low signal-to-noise ratio:** extremely difficult to find tradable patterns in markets

# Not So Fast

## Insuccess on Historical Data

Successfully applying these RL algorithms to historical data is much more challenging

1. Fail to converge
2. The strategies learned are not profitable



## Possible Explanations

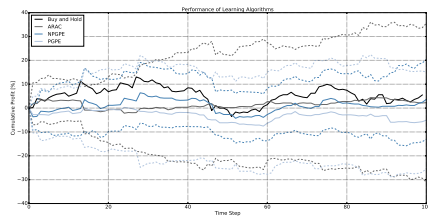
1. **Low signal-to-noise ratio:** extremely difficult to find tradable patterns in markets
2. **Quality of data:** unlikely to find patterns in daily prices of liquid stocks

# Not So Fast

## Insuccess on Historical Data

Successfully applying these RL algorithms to historical data is much more challenging

1. Fail to converge
2. The strategies learned are not profitable



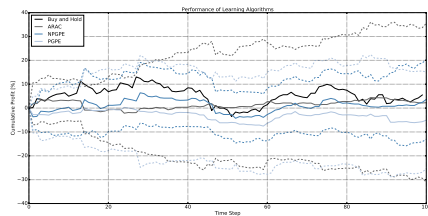
## Possible Explanations

1. **Low signal-to-noise ratio:** extremely difficult to find tradable patterns in markets
2. **Quality of data:** unlikely to find patterns in daily prices of liquid stocks
3. **Weak features:** parametric policy must be powerful enough to capture the signal

## Insuccess on Historical Data

Successfully applying these RL algorithms to historical data is much more challenging

1. Fail to converge
2. The strategies learned are not profitable



## Possible Explanations

1. **Low signal-to-noise ratio:** extremely difficult to find tradable patterns in markets
2. **Quality of data:** unlikely to find patterns in daily prices of liquid stocks
3. **Weak features:** parametric policy must be powerful enough to capture the signal
4. **Non-stationarity of financial time-series:** a signal needs to be persistent

# Plan

1. Basics of Reinforcement Learning
2. Policy Gradient Algorithms
3. Asset Allocation with Transaction Costs
4. Conclusions

## What Has Been Done

1. In-depth bibliographical study of state-of-the-art policy gradient algorithms
2. Innovative contributions to the policy gradient literature
3. Applied these techniques to find a profitable long-short trading strategy

# Conclusions

## What Has Been Done

1. In-depth bibliographical study of state-of-the-art policy gradient algorithms
2. Innovative contributions to the policy gradient literature
3. Applied these techniques to find a profitable long-short trading strategy





## Research Directions

1. Improve the algorithms performance on historical data
2. Develop more complex features for the trading strategy
3. Combine policy gradient algorithms with state-of-the-art deep learning techniques
4. RL framework is versatile and can be applied to other financial decision problems



Thank you for your attention!

# References

-  Deng, Y., Bao, F., Kong, Y., Ren, Z., and Dai, Q. (2016).  
Deep direct reinforcement learning for financial signal representation and trading.  
*IEEE Transactions on Neural Networks and Learning Systems*.
-  Peters, J. and Schaal, S. (2008).  
Reinforcement learning of motor skills with policy gradients.  
*Neural networks*, 21(4):682–697.
-  Sehnke, F. (2012).  
*Parameter exploring policy gradients and their implications*.  
PhD thesis, Technische Universität München.
-  Sutton, R. S. and Barto, A. G. (1998).  
*Introduction to reinforcement learning*, volume 135.  
MIT Press Cambridge.

# 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  $\langle \mathbb{S}, \mathbb{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

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

# Policy Gradient Theorem: Statement and Proof

## Policy Gradient Theorem

Let  $\pi_\theta$  be a differentiable policy. For the gradient of the average reward is

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

where  $d^\theta$  is the stationary distribution of the Markov chain induced by  $\pi_\theta$ .

### Proof

$$\nabla_\theta V_\theta(s) = \nabla_\theta \int_{\mathbb{A}} \pi_\theta(s, a) Q_\theta(s, a) da = \int_{\mathbb{A}} [\nabla_\theta \pi_\theta(s, a) Q_\theta(s, a) + \pi_\theta(s, a) \nabla_\theta Q_\theta(s, a)] da$$

$$\nabla_\theta Q_\theta(s, a) = \nabla_\theta \left[ \mathcal{R}(s, a) - \rho_\theta + \int_{\mathbb{S}} \mathcal{P}(s, a, s') V_\theta(s') ds' \right] = -\nabla_\theta \rho_\theta + \int_{\mathbb{S}} \mathcal{P}(s, a, s') \nabla_\theta V_\theta(s') ds'$$

$$\nabla_\theta V_\theta(s) = \int_{\mathbb{A}} \nabla_\theta \pi_\theta(s, a) Q_\theta(s, a) da - \nabla_\theta \rho_\theta + \int_{\mathbb{A}} \pi_\theta(s, a) \int_{\mathbb{S}} \mathcal{P}(s, a, s') \nabla_\theta V_\theta(s') ds'$$

$$\int_{\mathbb{S}} d^\theta(s) \int_{\mathbb{A}} \pi(s, a) \int_{\mathbb{S}} \mathcal{P}(s, a, s') \nabla_\theta V(s') ds' dads = \int_{\mathbb{S}} d^\theta(s) \nabla_\theta V_\theta(s) ds$$

$$\nabla_\theta \rho_\theta = \int_{\mathbb{S}} d^\theta(s) \int_{\mathbb{A}} \nabla_\theta \pi_\theta(s, a) Q_\theta(s, a) dads = \int_{\mathbb{S}} d^\theta(s) \int_{\mathbb{A}} \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a) Q_\theta(s, a) dads$$

# Monte-Carlo Policy Gradient: Pseudocode

**Input:** Stochastic policy  $\pi_\theta$ , Initial parameters  $\theta_0$ , learning rate  $\{\alpha_k\}$

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

1: **repeat**

2:     Sample  $M$  trajectories  $h^{(m)} = \{(s_t^{(m)}, a_t^{(m)}, r_{t+1}^{(m)})\}_{t=0}^{T^{(m)}}$  under policy  $\pi_{\theta_k}$

3:     Approximate policy gradient

$$\nabla_\theta J(\theta_k) \approx \frac{1}{M} \sum_{m=0}^M \sum_{u=0}^{T^{(m)}-1} \nabla_\theta \log \pi_{\theta_k} \left( s_u^{(m)}, a_u^{(m)} \right) \sum_{v \geq u}^{T^{(m)}-1} \gamma^{v-u} r_{v+1}^{(m)}$$

4:     Update parameters using gradient ascent  $\theta_{k+1} = \theta_k + \alpha_k \nabla_\theta J(\theta_k)$

5:      $k \leftarrow k + 1$

6: **until** converged

# Episodic PGPE Algorithm: Pseudocode

**Input:** Controller  $F_\theta$ , hyper-distribution  $p_\xi$ , initial guess  $\xi_0$ , learning rate  $\{\alpha_k\}$

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

- 1: **repeat**
- 2:     **for**  $m = 1, \dots, M$  **do**
- 3:         Sample controller parameters  $\theta^{(m)} \sim p_{\xi_k}$
- 4:         Sample trajectory  $h^{(m)} = \{(s_t^{(m)}, a_t^{(m)}, r_{t+1}^{(m)})\}_{t=0}^{T^{(m)}}$  under policy  $F_{\theta^{(m)}}$
- 5:     **end for**
- 6:     Approximate policy gradient

$$\nabla_\xi J(\xi_k) \approx \frac{1}{M} \sum_{m=1}^M \nabla_\xi \log p_\xi(\theta^{(m)}) \left[ G(h^{(m)}) - b \right]$$

- 7:     Update hyperparameters using gradient ascent  $\xi_{k+1} = \xi_k + \alpha_k \nabla_\xi J(\xi_k)$
- 8:      $k \leftarrow k + 1$
- 9: **until** converged

# Natural PGPE Algorithm: Pseudocode

**Input:** Controller  $F_\theta$ , hyper-distribution  $p_\xi$ , initial guess  $\xi_0$ , learning rate  $\{\alpha_k\}$

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

- 1: **repeat**
- 2:   Observe current state  $s_k$
- 3:   Draw  $\zeta_k \sim \mathcal{N}(0, I_n)$
- 4:   Compute controller parameters  $\theta_k = \mu_k + \Gamma^T \zeta_k$
- 5:   Perform action  $a_k = F_{\theta_k}(s_k)$  and receive reward  $r_{k+1}$
- 6:   Update average reward estimate  $\hat{\rho}_{k+1} = \hat{\rho}_k + \alpha_k(r_{k+1} - \hat{\rho}_k)$
- 7:   Compute natural policy gradients

$$\tilde{\nabla}_\mu \log p_{\xi_k}(\theta_k) = \theta_k - \mu_k \quad \tilde{\nabla}_\Gamma \log p_{\xi_k}(\theta_k) = \left( \text{triu}(\zeta_k \zeta_k^T) - \frac{1}{2} \text{diag}(\zeta_k \zeta_k^T) - \frac{1}{2} I \right) \Gamma$$

- 8:   Update eligibility trace  $e_k = \lambda e_{k-1} + \nabla_\xi \log p_{\xi_k}(\theta_k)$
- 9:   Update hyper-parameters  $\xi_{k+1} = \xi_k + \alpha_k(r_{k+1} - \hat{\rho}_k)e_k$
- 10:    $k \leftarrow k + 1$
- 11: **until** converged

## Experiment on Synthetic Asset

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  $\{\beta_t\}$

$$z_t = z_{t-1} + \beta_{t-1} + \kappa \epsilon_t$$

$$\beta_t = \alpha \beta_{t-1} + \nu_t$$

The policy used for the PGPE and the NPGPE algorithms is

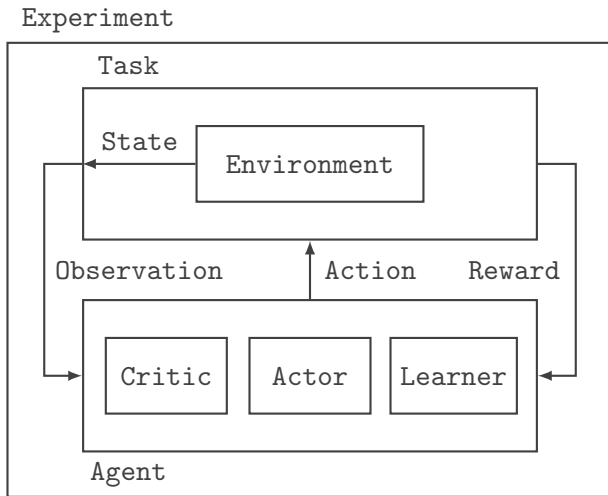
$$F_\theta(s) = \text{sign}(\theta \cdot s)$$

where

$$\theta \sim \mathcal{N}(\mu, \text{diag}(\sigma))$$



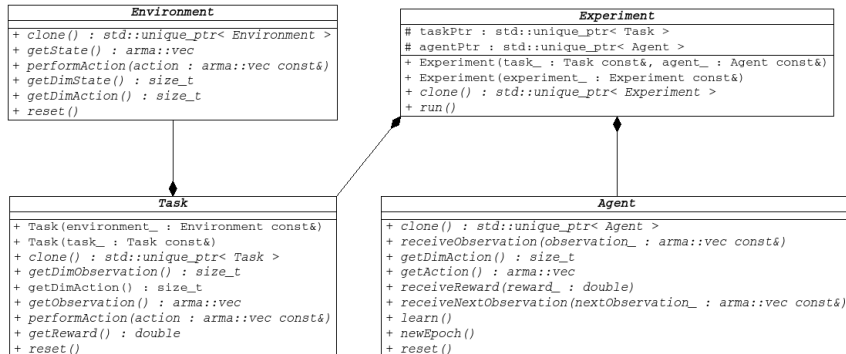
# PyBrain's Architecture for a RL Problem



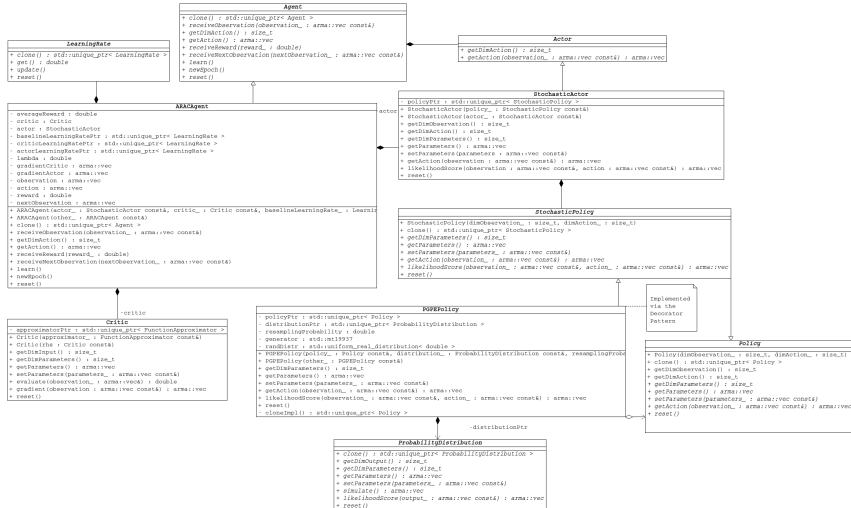
# Agent-Environment Interaction in C++

## 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

