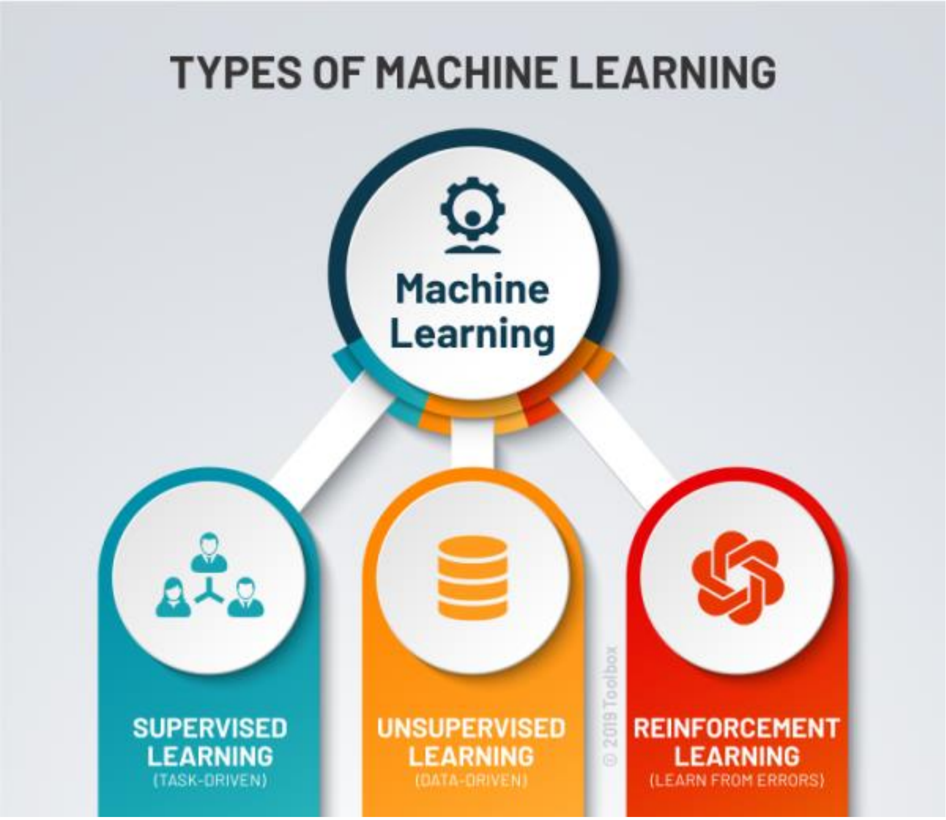
Reinforcement Learning

Jan Harms
Gran Sasso Science Institute

Reinforcement Learning

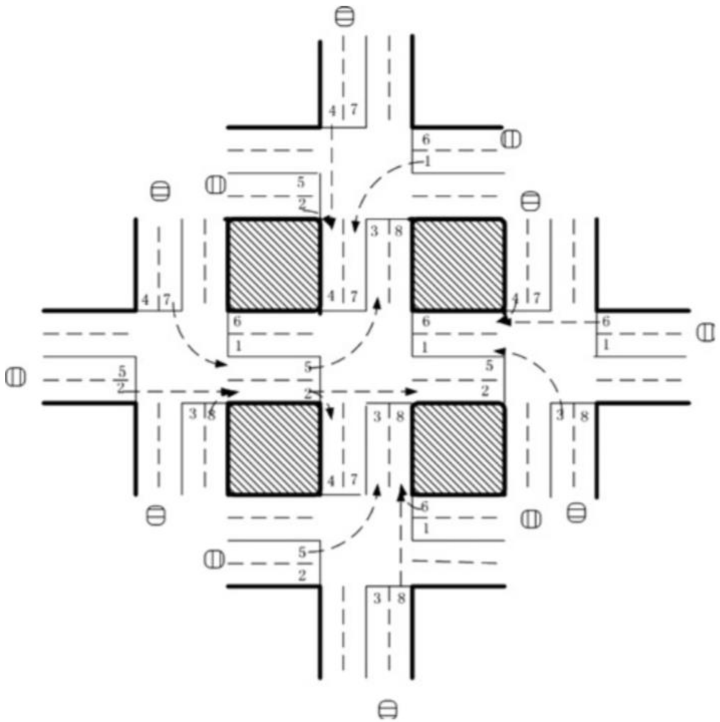


- Supervised learning produces a static map from features to labels
- Reinforcement learning guides the actions of an **agent in an environment**
- RL uses **goals** instead of targets
- Even in abstract environments, this scheme imposes the **notion of time** since the learning process is step wise.
- The interaction can stop when a goal is achieved, which concludes an **episode**


Reinforcement
Learning
An Introduction
second edition
Richard S. Sutton and Andrew G. Barto

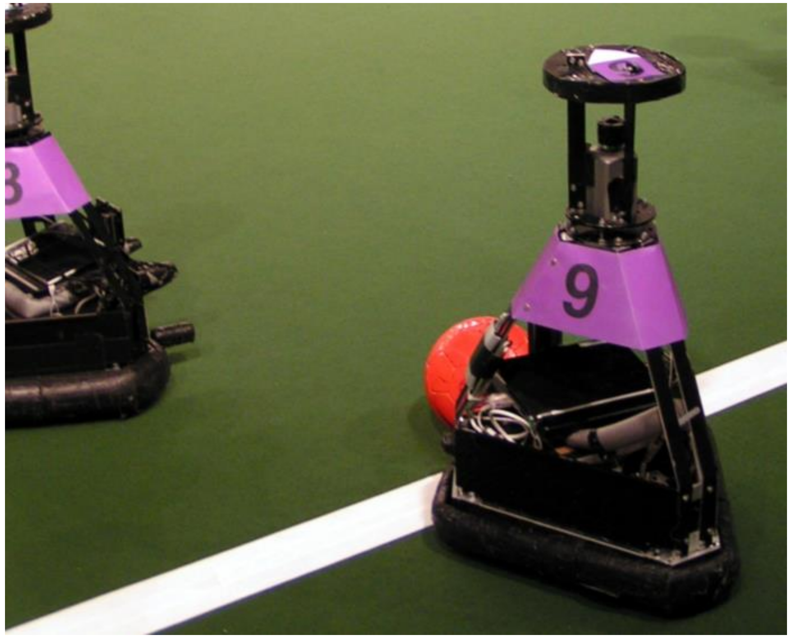
Examples

Traffic light control



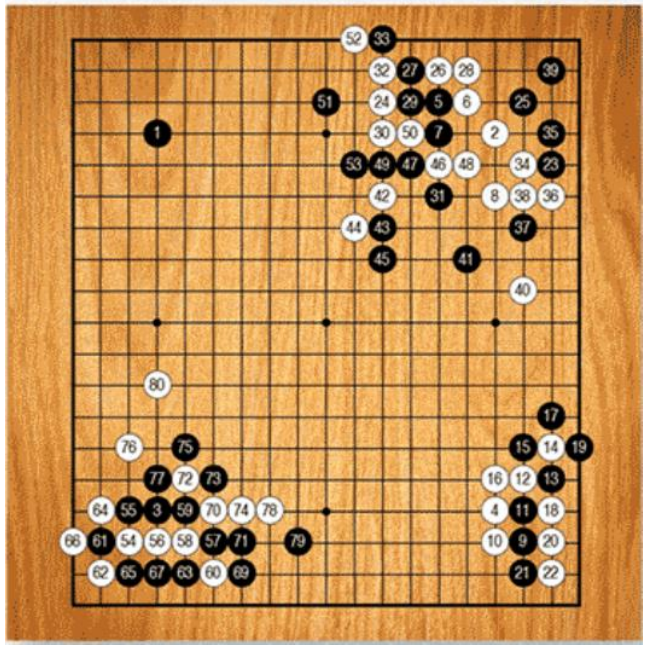
Arel et al (2010)

Robotics



Riedmiller et al (2009)

Games



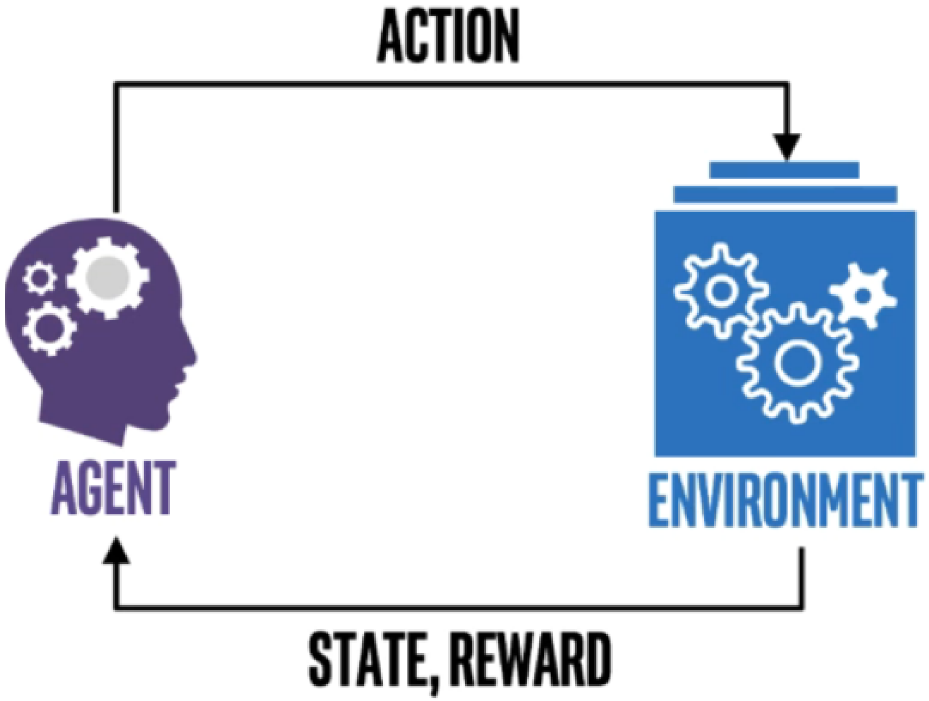
Google DeepMind (2017)

Markov Decision Process

Basic quantities: state, action, reward

Policy
(generally probabilistic)

$$\pi(a|s)$$



Transition probability
(does not depend on
states «before» s)

$$p(s', r|s, a)$$

Reinforcement

The goal is to maximize returns
(which is the sum over future rewards)

$$R(t) = \sum_{\tau=1}^{T-t} r(t+\tau) = r(t+1) + \ldots + r(T)$$

Rewards can be discounted to express growing uncertainty (also used in open horizon problems)

$$R(t) = \sum_{\tau=1}^{T-t} \gamma^{\tau-1} r(t+\tau) = r(t+1) + \gamma r(t+2) + \ldots + \gamma^{T-t-1} r(T)$$

Value Function

What is called (simply) value function is the expected future return when being in state s: $V(s) = \langle R|_s \rangle$

Bootstrapping

$$V(s) = \langle r(t+1) + \gamma V(s')|_s \rangle$$

$$V(s) = \sum_a \sum_{s'} \sum_r \pi(a|s) p(s', r|s, a)(r + \gamma V(s'))$$

Bellman equation

Step 1: **Inference**

Given a policy π, estimate the value function V(s)

Step 2: Control Problem

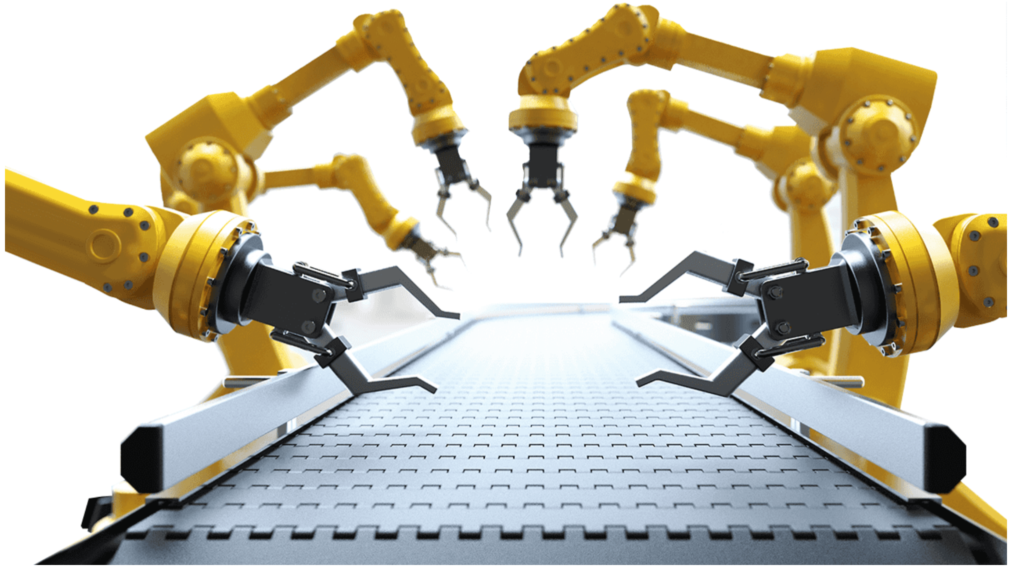
Find the policy π^* that maximizes V(s) for all states s.

Q-value

$$Q(s,a) = \sum_{s'} \sum_{r} p(s', r | s, a)(r + \gamma V(s'))$$

Optimal action with maximized Q-values

$$a^* = \underset{a}{\operatorname{argmax}}\, Q^*(s,a)$$



The optimal policy is the collection of optimal actions a* for all states.

Solving the Bellman Equation

Recursive relation to estimate the mean

$$\langle x\rangle_N = \frac{N-1}{N}\langle x\rangle_{N-1} + \frac{1}{N}x_N = \langle x\rangle_{N-1} + \frac{1}{N}(x_N - \langle x\rangle_{N-1})$$

Gradient descent

$$L = (R - V(s))^2$$

$$V(s) \leftarrow V(s) - \frac{1}{2}\eta\nabla_{V(s)}L$$

$$\nabla_{V(s)}L = -2(R - V(s))$$

$$V(s) \leftarrow V(s) + \eta(R - V(s))$$

Temporal difference

$$R(t) = r(t+1) + \gamma R(t+1)$$

$$R(t) = r + \gamma V(s')$$

Predicted return!

Q-Learning

Update relation:

$$y = r + \gamma \max_{a'} Q(s', a')$$

$$Q(s,a) \leftarrow Q(s,a) + \eta(y - Q(s,a))$$

Important observation

The target y is not a product of the current policy.

This is why Q-learning is also known as off-policy RL algorithm.

Possibilities and constraints

1. Q-learning can be extended to continuous state spaces, e.g., representing Q(s,a) by a neural network (DQN)
2. Since the maximization of Q-values requires an evaluation over all possible actions in each state s, the method cannot be applied to continuous action spaces

INFN
Istituto Nazionale di Fisica Nucleare

Reinforcement Learning Algorithms

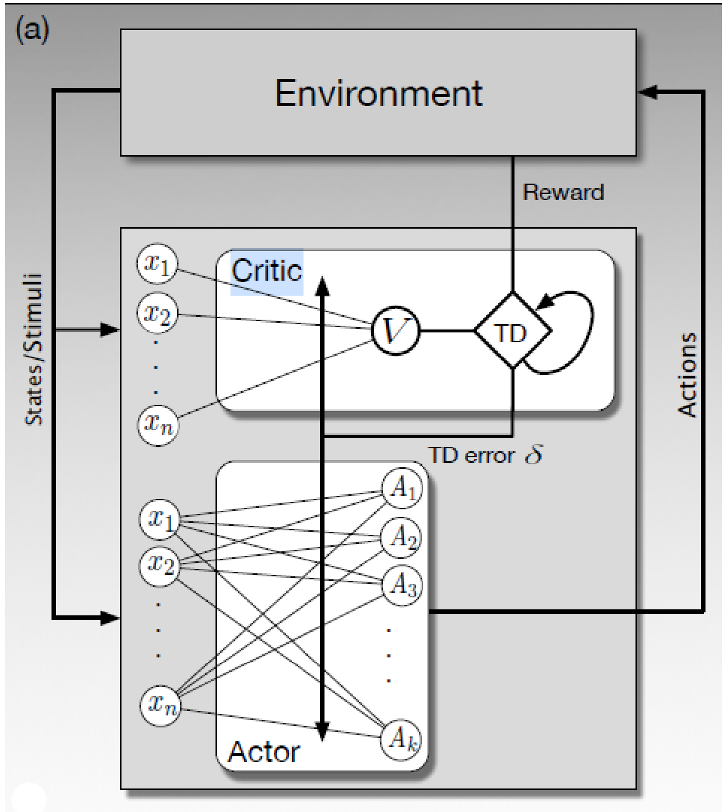
GSSI

Algorithm	Description	Model	Policy	Action Space	State Space	Operator
Monte Carlo	Every visit to Monte Carlo	Model-Free	Either	Discrete	Discrete	Sample-means
Q-learning	State–action–reward–state	Model-Free	Off-policy	Discrete	Discrete	Q-value
SARSA	State–action–reward–state–action	Model-Free	On-policy	Discrete	Discrete	Q-value
Q-learning - Lambda	State–action–reward–state with eligibility traces	Model-Free	Off-policy	Discrete	Discrete	Q-value
SARSA - Lambda	State–action–reward–state–action with eligibility traces	Model-Free	On-policy	Discrete	Discrete	Q-value
DQN	Deep Q Network	Model-Free	Off-policy	Discrete	Continuous	Q-value
DDPG	Deep Deterministic Policy Gradient	Model-Free	Off-policy	Continuous	Continuous	Q-value
A3C	Asynchronous Advantage Actor-Critic Algorithm	Model-Free	On-policy	Continuous	Continuous	Advantage
NAF	Q-Learning with Normalized Advantage Functions	Model-Free	Off-policy	Continuous	Continuous	Advantage
TRPO	Trust Region Policy Optimization	Model-Free	On-policy	Continuous	Continuous	Advantage
PPO	Proximal Policy Optimization	Model-Free	On-policy	Continuous	Continuous	Advantage
TD3	Twin Delayed Deep Deterministic Policy Gradient	Model-Free	Off-policy	Continuous	Continuous	Q-value
SAC	Soft Actor-Critic	Model-Free	Off-policy	Continuous	Continuous	Advantage

Wikipedia (2020)

Actor-Critic

- Actor implemented as policy-gradient method
- Critic a bootstrapping value learner



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 $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S, \boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

Sutton/Barto (2018)

A2C and A3C

Advantage

$$A(s_t, a_t) = Q_w(s_t, a_t) - V_v(s_t)$$

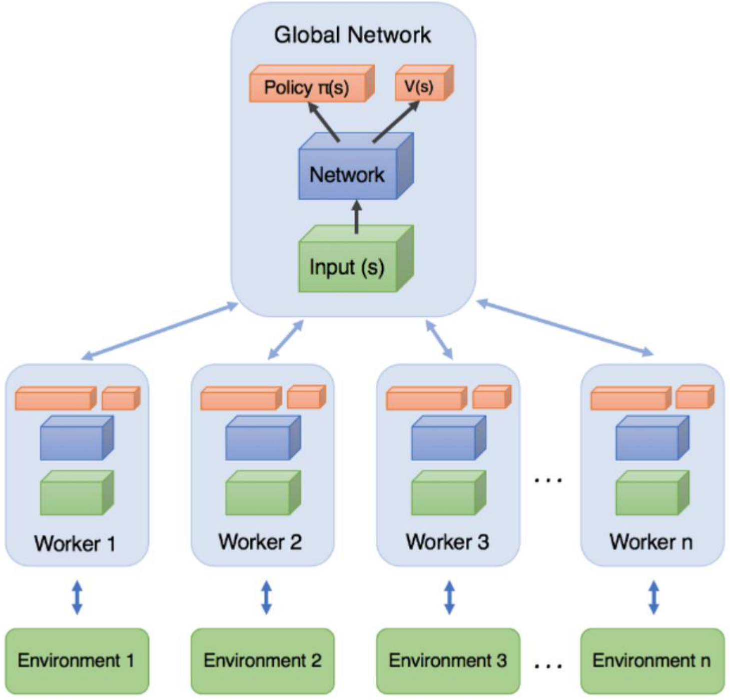
Loss as function of policy parameters

$$\nabla_\theta J(\theta) \sim \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)(r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t))$$
$$= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t)$$

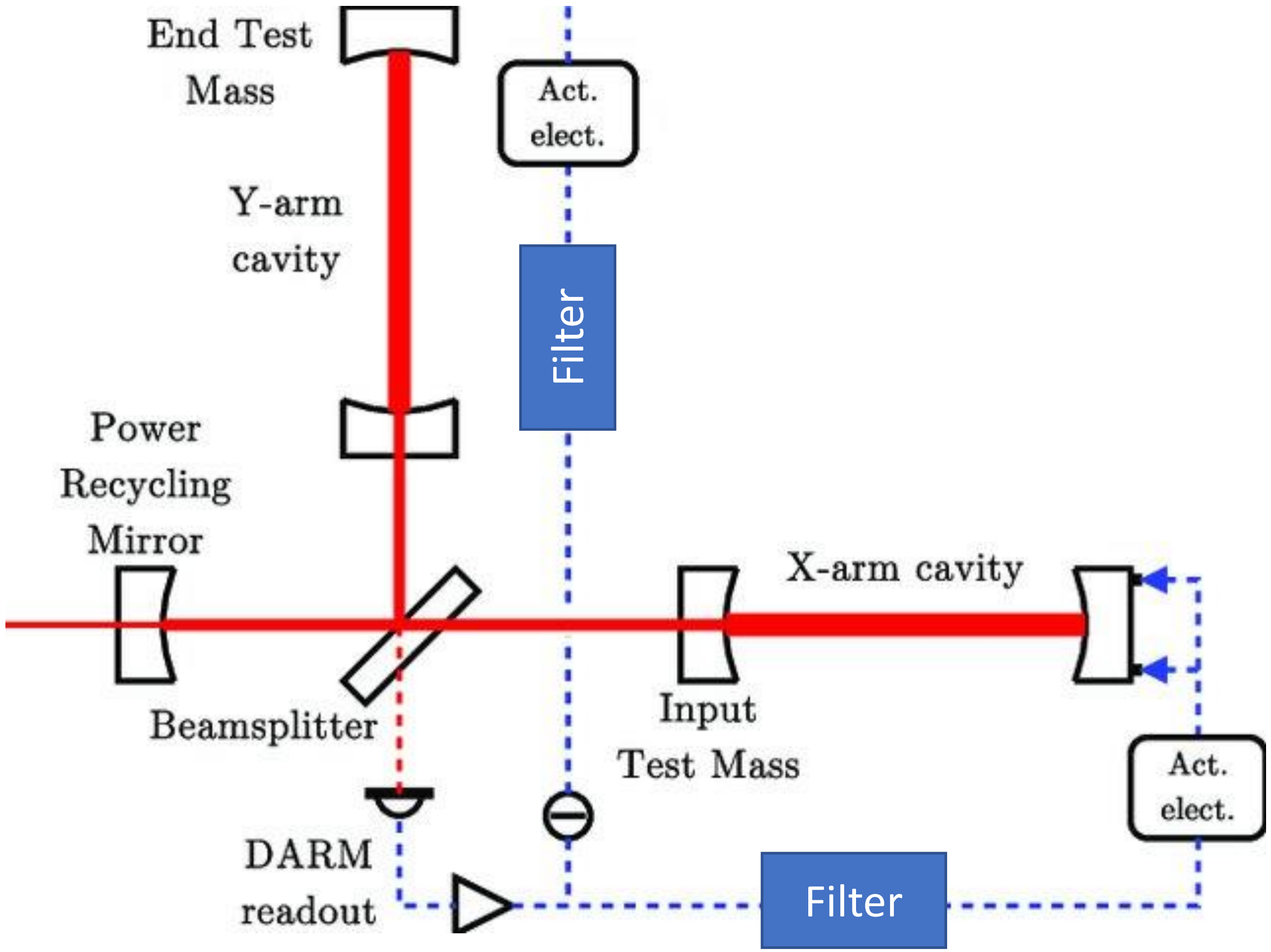
Here, you find the value function in standard policy-gradient methods.

A3C

Parallelization by using multiple workers to learn global value and policy functions



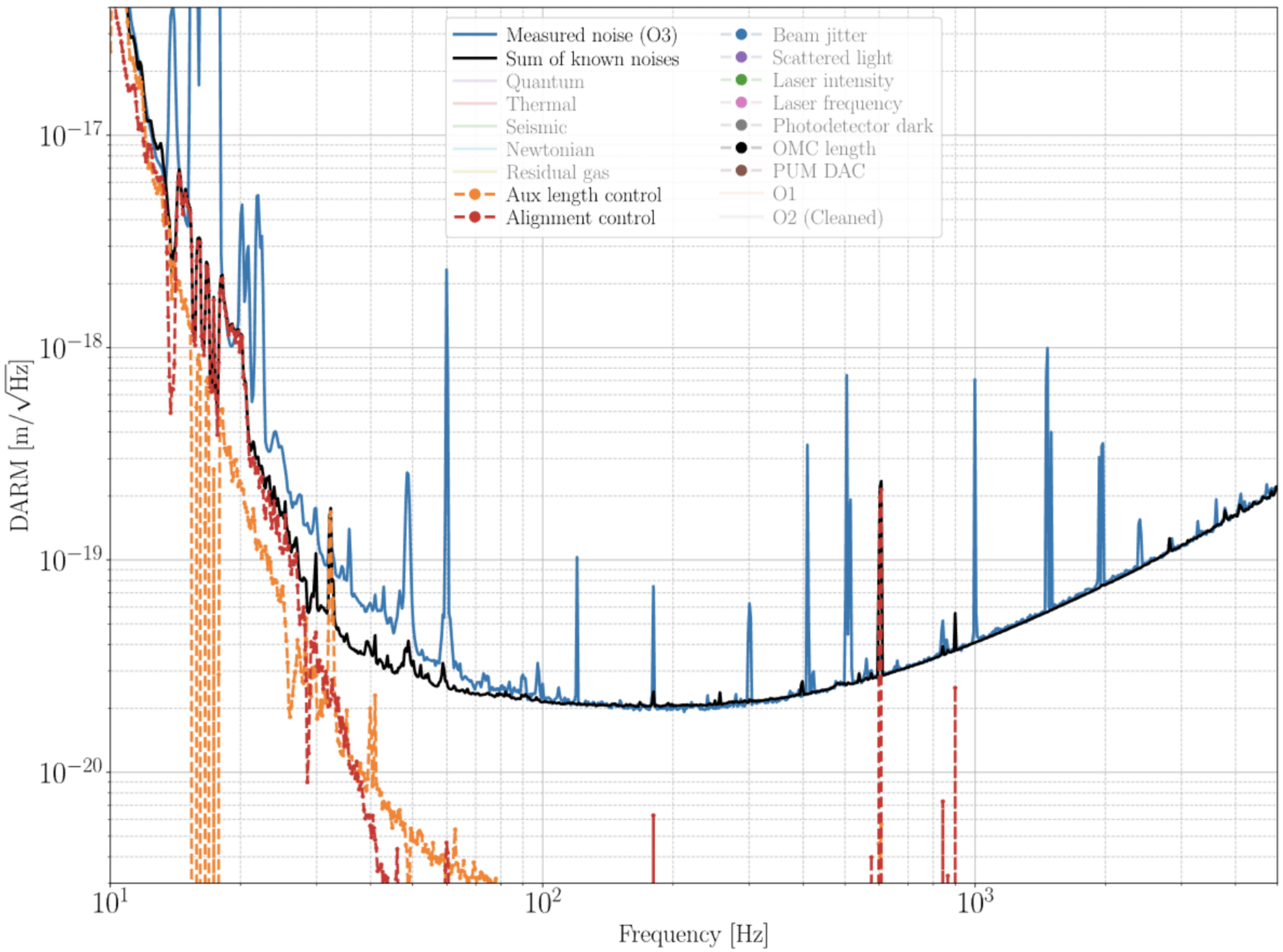
Control in LIGO/Virgo



Filter design

- Linearity, stability
- Goal can be to reduce rms of test-mass motion often dominated by <10Hz motion
- Important is not to introduce too much noise above 10Hz
- Filter design typically obtained by a half-quantitative, half-intuitive method
- A form of optimal linear filter can be calculated, but stability remains an issue

Example: LIGO Controls Noise



RL for LIGO/Virgo

What could be achieved?

- Further reduce rms of test-mass motion and/or controls noise seen in GW data
- Adaptability of control to changes in the detector (light power, temperature, mirror deformations, environmental noise)
- Lock an interferometer

Implementation?

- Start with simulations, make use of infrastructure for testing (prototypes, and table-top experiments)
- Design as parallel path to the linear filter, substitute linear filter using smooth controls hand-off, …
- Apply transfer learning to adapt between simulations and reality