Accelerator control with Advanced Algorithms and Machine Learning

V. Kain with material from the CERN ML Community Forum

Overview

★ Controlling CERN accelerators - the classical approach

★ Numerical optimisation

★ Reinforcement learning

Physics models in the control room

Beam dynamics equation of motion in static magnetic and RF fields linearised and solved
→ **closed form solutions used as models in the control room to control**

- ★ mean energy, energy spread, beam size, orbit,…collective motion of particles,…

→ **global parameters:** $B\rho$ or p, the tunes Q_x, Q_y, Q_s and tune spreads through additional global parameters e.g. chromaticity $Q'_{x,y}$

→ Use **high level physics parameters** to control accelerators instead of direct hardware parameters: i.e. normalised magnetic fields instead of currents.

- ★ e.g. dipole magnet's control parameter: bending angle change $\Delta x'$ or $\Delta y'$ instead of current in power supply.
- ★ needs hardware to physics parameter translation: e.g.. *transfer function* $Bl \rightarrow I$ for every magnetic circuit

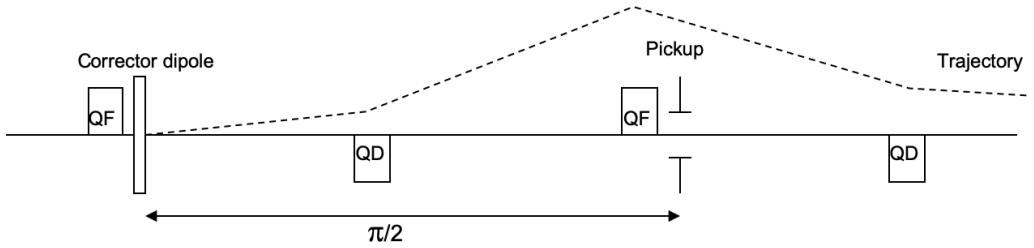
[1] D. Jacquet et al, LSA- The high level application software of the LHC and its performance during the first 3 years of operation , ICALEPS 2013

Physics models in the control room

Build parameter models, store transfer functions in controls DB, pre-calculate the settings of accelerate according to uploaded "optics", injection/extraction momentum,…

This allows: model-based one-shot correction of imperfections

★ E.g. trajectory correction



★ Calculate response R, with R^{-1} settings for correctors for given $(\Delta x_1, \Delta x_2, \ldots \Delta x_m)$

$$R \begin{pmatrix} \Delta x_1' \\ \Delta x_2' \\ \cdots \\ \Delta x_n' \end{pmatrix} = \begin{pmatrix} \Delta x_1 \\ \Delta x_2 \\ \cdots \\ \Delta x_m \end{pmatrix}$$

✳ R is linear for our machines (i.e. matrix) → SVD algorithm

Classical approach is not enough

Our goal for accelerator operation: maximum efficiency and maximum flexibility while achieving maximum performance

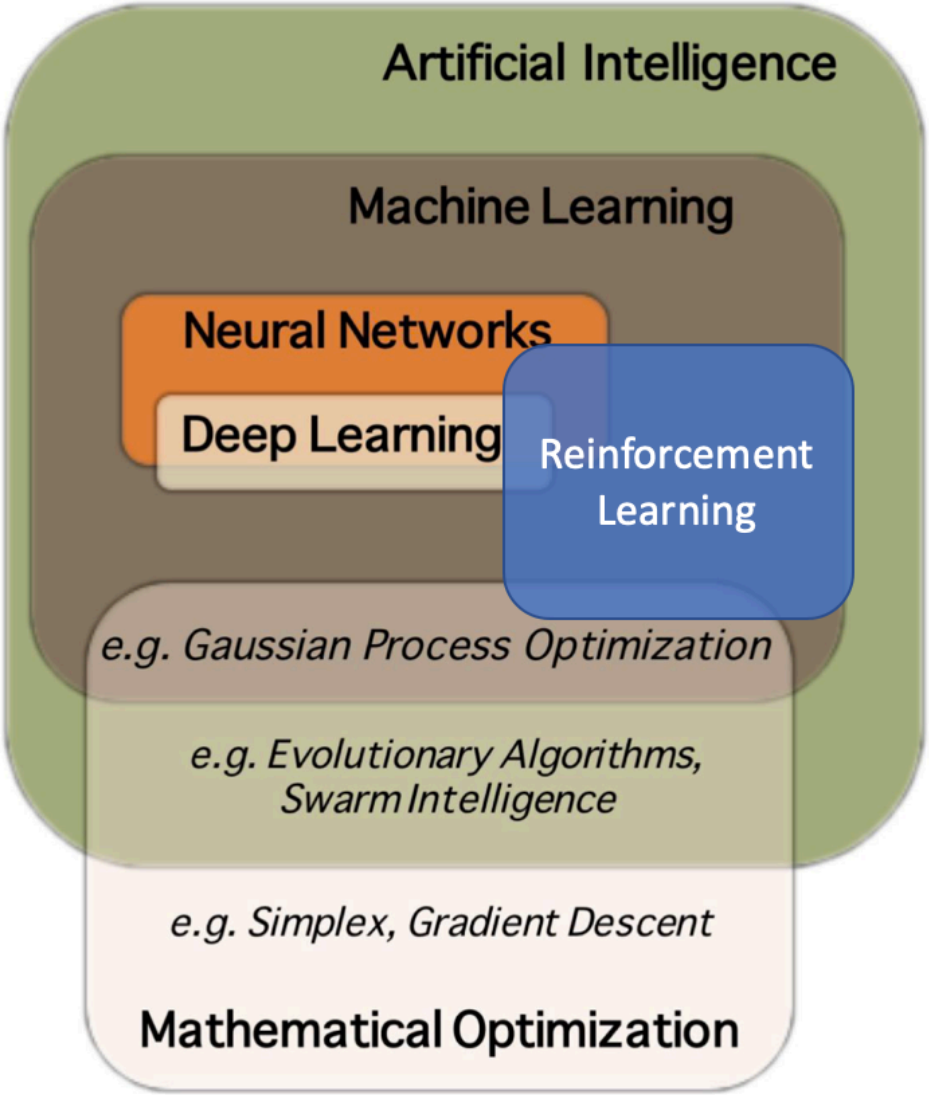
→ physics based deterministic operation of accelerators, no trial and error.

Not always possible:

- ★ need models, and models online available; models can be very complicated
- ★ there are drifts → modelling even more complicated
- ★ need sufficient beam instrumentation
- ★ need algorithms on top of models; models not always easily invertible

One way out → automated and sample-efficient optimisation algorithms

The landscape



Courtesy A. Edelen

Numerical Optimisation

Definitions and Basics

The ***generic optimisation problem***:

$$x^* = \text{argmin } f(x) \quad \text{subject to} \quad \begin{matrix} c_i(x) = 0 \\ c_i(x) \geq 0 \end{matrix}$$

where $f(x)$ is the (scalar) ***objective function*** to be minimised or maximised, x is the vector of ***unknowns*** or ***parameters*** and c_i are ***constraint functions***.

Convexity

Many algorithms work best if $f(x)$ is ***convex***:

local minimum = global minimum

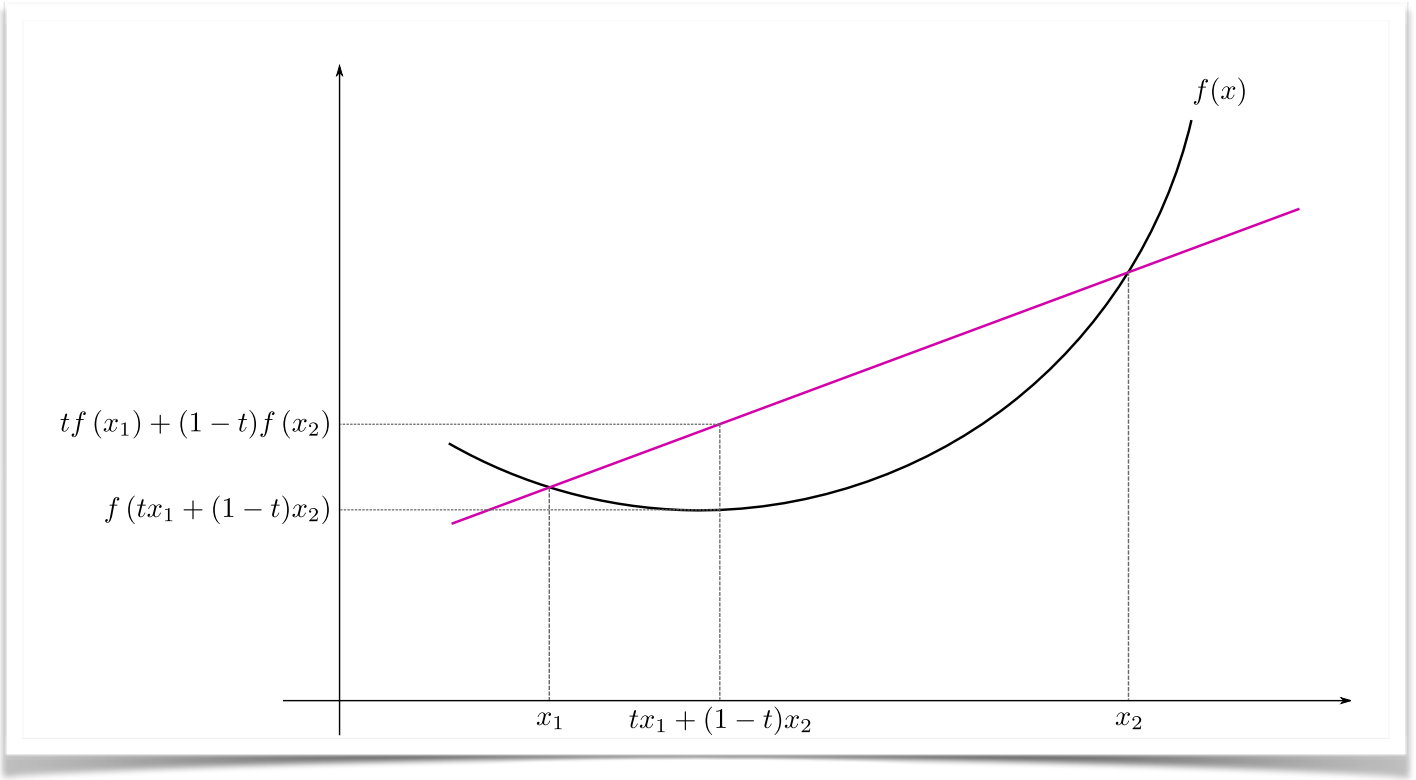
Mathematical definition for $x_1, x_2 \in X$, a convex subset:

for all $0 \leq t \leq 1$

$$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$$

Convexity

$f(x)$ is convex: if the line segment between any two points $f(x_1), f(x_2)$ is either equal or above $f(x)$ for $x = tx_1 + (1-t)x_2$



By Eli Osherovich - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=10764763

Algorithm types

multi-objective, constraints, bounds, model-based, derivative-free,…

Focus here on single-objective, derivative-free.

- ★ derivative-free: as our objective is a black-box function
  - ✳ E.g. sum of beam losses in the extraction region, injection efficiency,…
- ★ Algorithm types: conjugate direction methods, Nelder-Mead method, model-based methods, simulated annealing,…

Focus here on **model-based algorithms:**

Start with algorithms that build local deterministic models over **trust regions.**

Later: probabilistic global models with **Bayesian Optimisation**

Model-based, derivative free: trust region method

Some of the most effective algorithms for unconstrained optimisation: compute steps by minimising over a quadratic model of $f(x)$.

If derivatives of $f(x)$ not available, need to define model m_k as quadratic function that interpolates $f(x)$

$$m_k(x_k + p) = c + g^T p + \frac{1}{2} p^T G p$$

where the scalar c, the vector g and the symmetric matrix G are calculated with the **interpolation** conditions

$$m_k(y^l) = f(y^l) \quad \text{for } l = 1,2,\ldots,q$$

through a **linear system of equations**.

Number of interpolation points needed: $q = \frac{1}{2}(n+1)(n+2)$ for n number of degrees of freedom and the q interpolation points need to be non-singular.

Model-based, derivative free: trust region method

Compute step p by approximately solving the trust region subproblem

$$\min_p \; m_k(x_k + p) \qquad \text{subject to } \|p\|_2 \leq \Delta$$

where Δ is the **trust region radius**.

If $f(x_k + p)$ gives a sufficient reduction, next iterate: $x_{k+1} = x_k + p$ and Δ is updated according to

$$\rho = \frac{\text{actual reduction}}{\text{predicted reduction}} = \frac{f(x_k) - f(x_k^+)}{m_k(x_k) - m_k(x_k^+)}$$

If $\rho \geq \eta$ (some constant $\in (0,1)$), **increase** Δ, do the step, replace one element in Y by x_k^+.

Else: check whether Y needs to be improved (geometrical improvement) and start again or otherwise just shrink Δ and go to next iteration with $x_{k+1} = x_k$.

Model-based, derivative free: trust region method

Draw back:

★ Need $O(n^2)$ function evaluations before algorithm can start

Way out:

★ linear model: only need $n+1$ initial points, but model not as fast convergent as cannot model curvature.

★ → some algorithms start with linear model and then use quadratic when sufficient number points.

★ Powell's algorithm NEWUOA and BOBYQA use quadratic models with only $2n+1$, by calculating

$$min\|G_k - G_{k-1}\|_F \quad \text{subject to} \quad m_k(y^l) = f(y^l) \quad \text{for } l = 1,2,\ldots,q$$

One of our favourite algorithms: BOBYQA [1] (quadratic model and bounds) in the implementation Py-BOBYQA [2]

[2] M.J.D. Powell, The BOBYQA algorithm for bound constrained optimization without derivatives, 2009

[3] https://numericalalgorithmsgroup.github.io/pybobyqa/build/html/index.html

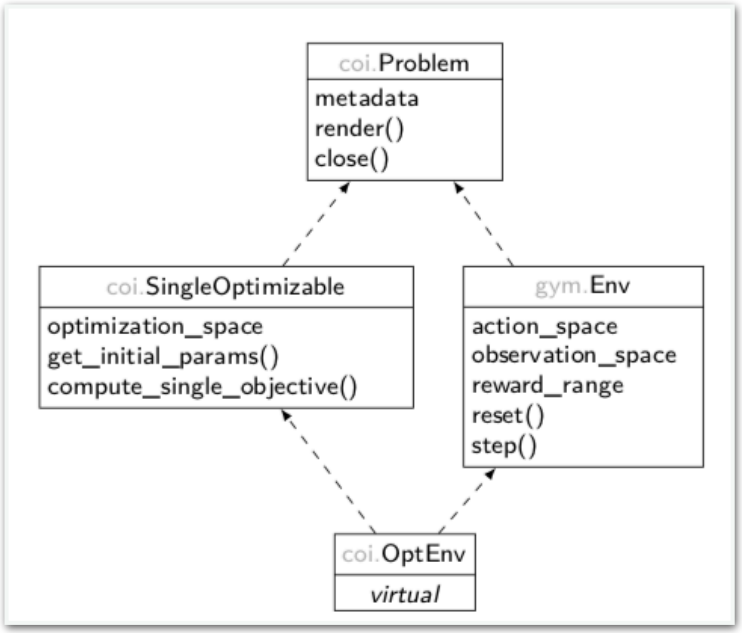
BOBYQA for the control room

From the py-BOBYQA package:

```
pybobyqa.solve(objfun, x0, args=(), bounds=None, npt=None,
               rhobeg=None, rhoend=1e-8, maxfun=None, nsamples=None,
               user_params=None, objfun_has_noise=False,
               seek_global_minimum=False,
               scaling_within_bounds=False,
               do_logging=True, print_progress=False)
```

CERN Generic Optimization Framework Frontend **GeOFF**

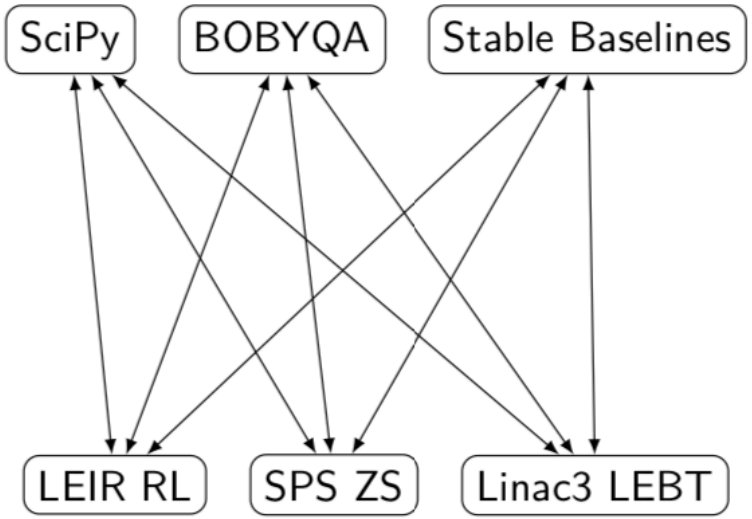
- ★ Common Optimization Interface (COI)
- ★ Based on OpenAI Gym for RL
- ★ extends Gym's metadata with CERN specifics



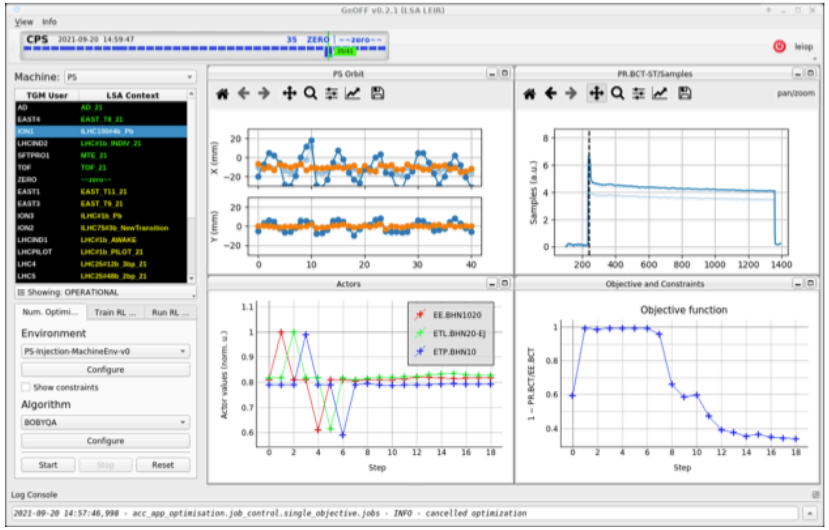
BOBYQA for the control room

Allows to combine different algorithms and different optimisation problems plug & play

packages with algos



different optimisation problems



Offer **GUI** to load optimisation problems solve them with different algorithms

→ deals with controls aspects, hide complexity of algos

→ algos and optimisation problems are configurable

→ offers basic default plotting

→ allows custom plotting

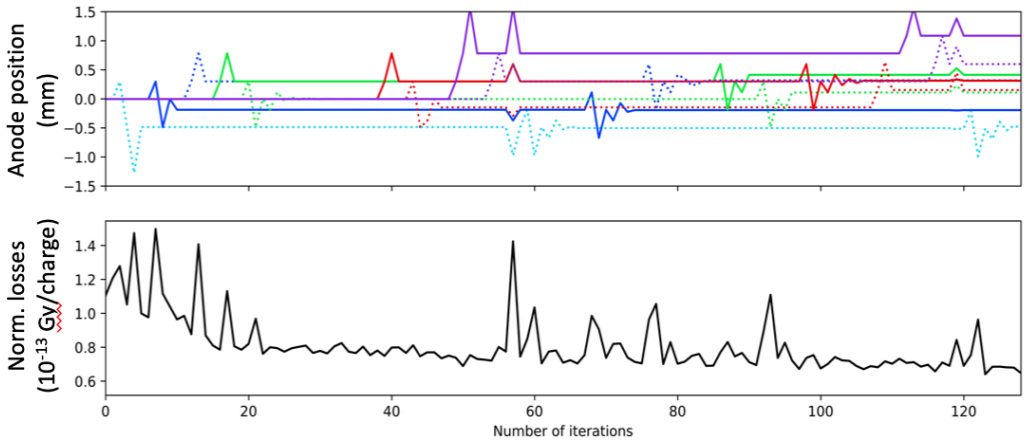
→ not only numerical optimisation, but also **reinforcement learning**

Example: Alignment of Electro-static Septum in the SPS

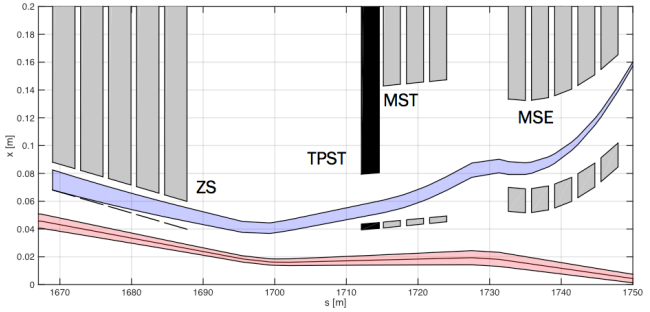
Objective: loss minimisation in extraction region, 9 degrees of freedom

November 2018: algorithm POWELL

- Before numerical optimisation for alignment: ~ 8 h

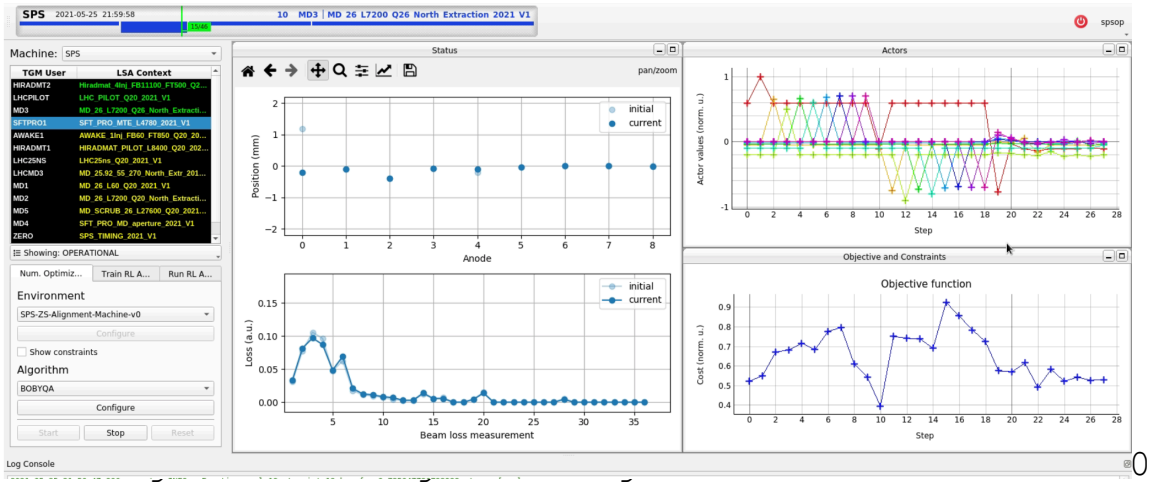


Circulating (red, ±3σ) and extracted (blue) horizontal beam envelopes and apertures in the LSS2 extraction region.



2018
~ 130 iterations.
~ 45 minutes

2021 BOBYQA and generic optimisation framework (based on OpenAi Gym)



2021
~ 30 iterations.

Bayesian Optimisation - learn global models

Machine-learning based, derivative-free, global optimisation method.

Basic idea:

- ★ Fit $f(x)$ with ***Gaussian Process*** probabilistic model, incorporate new data points using Bayesian statistics (→ posterior probability distribution)
- ★ Use model and uncertainty to define so-called ***Acquisition Function*** $u(x)$
- ★ Minimise/maximise Acquisition Function to define next x_{t+1}; observe $f(x_{t+1})$
- ★ do this for user-defined number of steps n

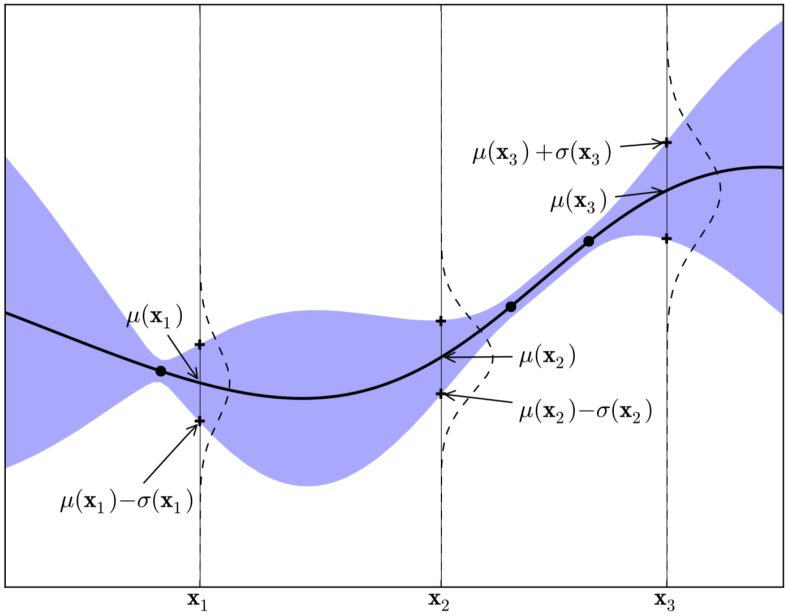
[4] C.E. Rasmussen, C.K.I. Williams, "Gaussian Processes for Machine Learning", the MIT press, 2006

Gaussian Processes (GP)

GP are extension of multivariate Gaussian to infinite dimension stochastic process.

GP is a distribution over functions completely defined by:

- ★ a mean function $\mu(x)$
- ★ a covariance function $k(x, x')$, covariance matrix $K_{ij} = k(x_i, x_j)$



Every Bayesian method needs a prior, an initial assumption for the model, with an initial $\mu(x)$ (e.g. zero) and $k(x, x')$.

Bayesian rule $P(M \mid E) \propto P(E \mid M)P(M)$ to calculate posterior, where M is the model (i.e. the prior) and E the evidence (i.e. data).

→ For inference: calculate the conditional probability $P(y_{t+1} \mid f(\mathbf{x}), x_{t+1}, \mathbf{x}) = N(\mu_t(x_{t+1}), \sigma_t^2(x_{t+1}))$ with

$$\mu_t(x_{t+1}) = \mathbf{k}^T\mathbf{K}^{-1}f(\mathbf{x}) \quad \text{and} \quad \sigma_t^2(x_{t+1}) = k(x_{t+1}, x_{t+1}) - \mathbf{k}^T\mathbf{K}^{-1}\mathbf{k}$$

Gaussian Process Regression

The covariance function (or kernel) defines how smooth, sparse,…the model will be.

Many different ones… The art is to use appropriate kernel for given problem.

Popular one: RBF (radial basis function) or squared exponential

$$\exp\left(-\frac{1}{2}\frac{|\mathbf{x_p}-\mathbf{x_q}|^2}{l^2}\right)$$

l…model parameter, length scale

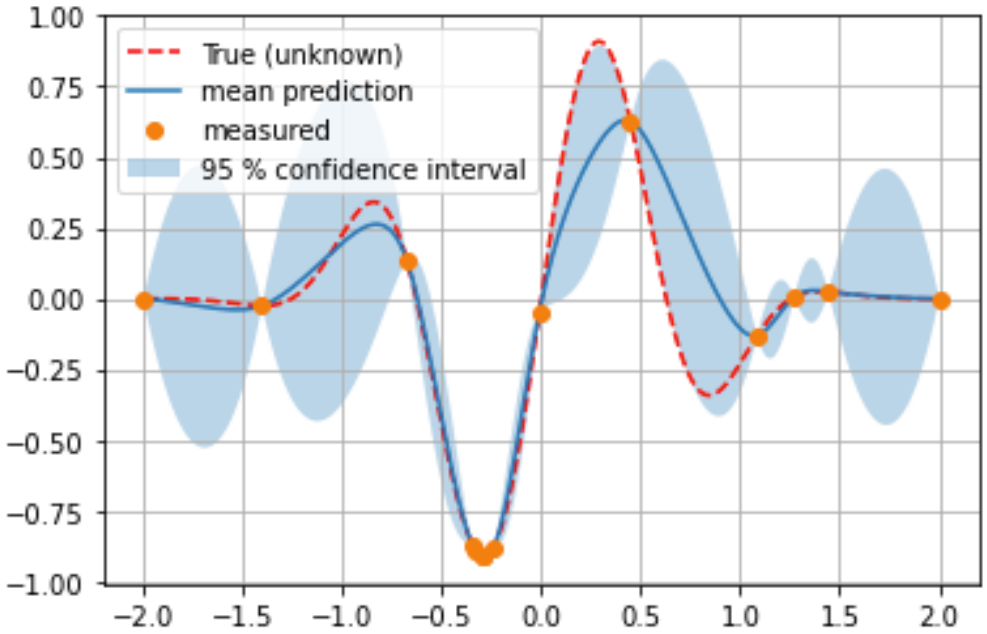
Model parameters are learned from data through the usual *max likelyhood* in regression

$$w^* = arg\max_w p(\mathbf{y}\,|\,\mathbf{x}, w)$$

GP Regression with sklearn

```
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, Matern
X_train=x_0
y_train = y_0
kernel = 1 * Matern(length_scale=1.0, length_scale_bounds=(1e-2, 1e2))
gaussian_process = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=9)
gaussian_process.fit(X_train, y_train)
kernel=gaussian_process.kernel_
```

```
y_pred,y_sigma = gaussian_process.predict(x,return_std=True)
```



Acquisition Function $u(x)$

Back to Bayesian Optimisation: The function $u(x)$ based on the posterior that is actually optimised to propose the next point x_{t+1}

Popular acquisition function: **Expected improvement** $EI(x)$ …x_t^+ best point so far.

$$-EI(x) = -\mathbb{E}[f(x) - f(x_t^+)]$$

$$EI(\mathbf{x}) = \begin{cases} (\mu(\mathbf{x}) - f(\mathbf{x}^+) - \xi)\Phi(Z) + \sigma(\mathbf{x})\phi(z) & \text{if } \sigma(\mathbf{x}) > 0 \\ 0 & \text{if } \sigma(\mathbf{x}) = 0 \end{cases} \text{ where}$$

$$Z = \begin{cases} (\frac{\mu(\mathbf{x}) - f(\mathbf{x}^+) - \xi}{\sigma(\mathbf{x})} & \text{if } \sigma(\mathbf{x}) > 0 \\ 0 & \text{if } \sigma(\mathbf{x}) = 0 \end{cases} \text{ and and } \Phi(Z) \text{ and } \phi(Z) \text{ are CDF and PDF of standard normal distribution.}$$

ξ is hyperparameter to guide exploration/exploitation. Good value of $\xi = 0.01$

Another acquisition function:

Lower confidence bound: $LCB(x) = \mu_{GP}(x) - \kappa\sigma_{GP}(x)$

κ is hyperparameter to guide exploration/exploitation. Default value $\kappa = 1.96$

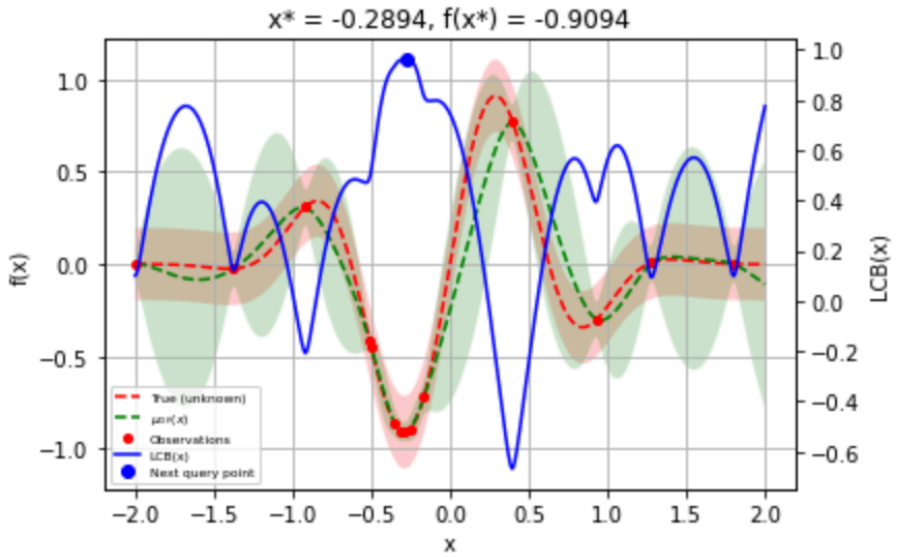
Basic Bayesian Optimization with skopt

In [37]:
```
from skopt import gp_minimize,Optimizer

res = gp_minimize(f,
                  [(-2.0,2.0)],
                  acq_func="LCB",
                  n_calls=15,
                  kappa=1.96,
                  n_random_starts=5,
                 )


from skopt.plots import plot_gaussian_process
# Plot f(x) + contours
print("total n_calls = ", len(res.models))
_ = plot_gaussian_process(res, objective=f_wo_noise,n_calls=10,
                          noise_level=noise_level,show_acq_func=True,show_next_point=True
                         )
```

```
total n_calls =  11
```



Reinforcement Learning

And Reinforcement Learning (RL)?

Numerical optimisation needs exploration phase at each deployment.

With RL (after training) exploration phase is reduced to a minimum → one iteration in the best case.

The reason:

- ★ it learns underlying **dynamics of the problem**
- ★ but needs additional input: ***state*** information
  - ✳ Given the ***state***, it applies the ***action*** to achieve maximum ***reward***

→ Controllers like with model-predictive control.

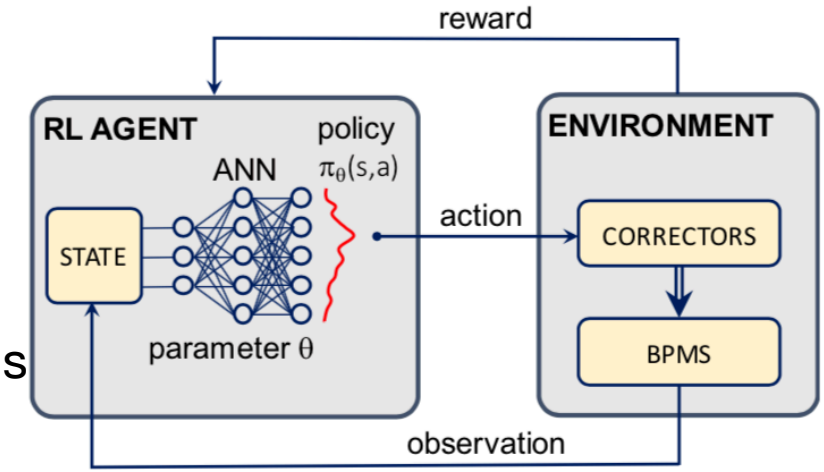
Basics of Reinforcement Learning

RL: learning how to **act** given a certain state to maximise cumulative reward.

Simple example: trajectory steering

State s :
- reading of BPMs

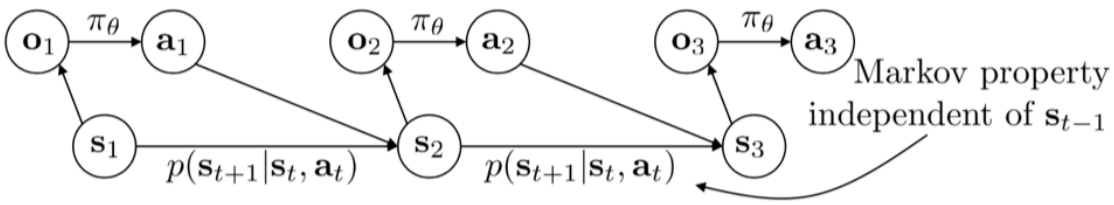
Action a :
- dipole corrector settings



Reward r could be:
- intensity on target
- RMS of trajectory $\times(-1)$
- losses $\times(-1)$

$\mathbf{s}_t$ – state
$\mathbf{o}_t$ – observation
$\mathbf{a}_t$ – action

$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ – policy
$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ – policy (fully observed)



Partly from course "Deep Reinforcement Learning", Sergey Levine

Basics of Reinforcement Learning

Goal of RL = find θ that maximises total reward

Episodic learning → maximise reward during episode along state trajectory $s_1 \ldots s_T$

$$\theta^* = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(s_t, a_t) \right]$$

Concepts to find optimum policy: Q and V (*value)* function

$$Q^\pi(s_t, a_t) = \sum_{t'=t}^{T} E_{\pi_\theta} \left[r(s_{t'}, a_{t'}) \mid s_t, a_t \right]$$: total reward from taking a_t in s_t

$$V^\pi(s_t) = \sum_{t'=t}^{T} E_{\pi_\theta} \left[r(s_{t'}, a_{t'}) \mid s_t \right]$$: total reward from s_t

$$V^\pi(s_t) = E_{a_t \sim \pi(a_t|s_t)} \left[Q^\pi(s_t, a_t) \right]$$

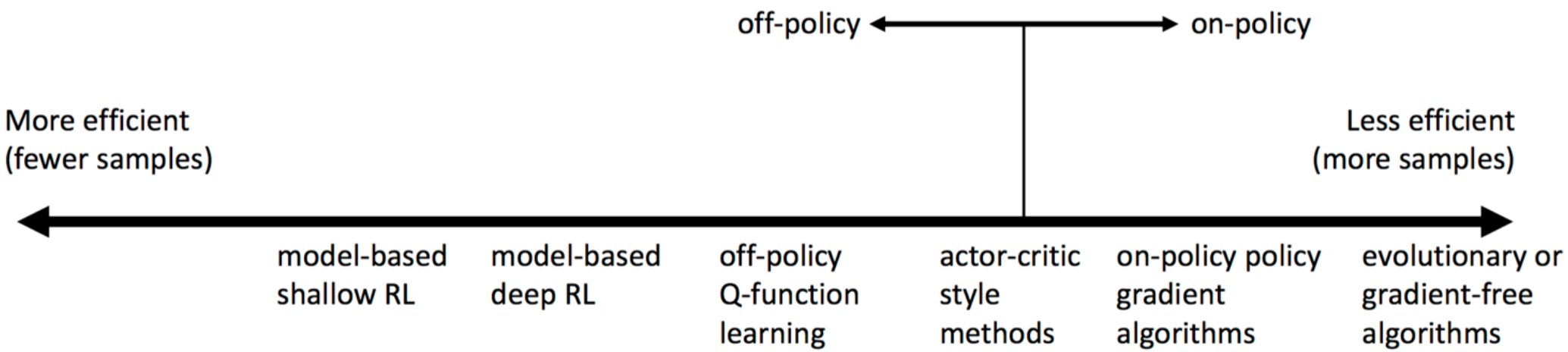
Algorithm Types

Goal: $\theta^* = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(s_t, a_t) \right]$

- ★ Policy gradients: directly differentiate the above objective
- ★ Value-based: estimate value function or Q-function of the optimal policy
  - ✳ (no explicit policy)
- ★ Actor-critic: estimate value function or Q-function of the current policy, use it to improve the policy
- ★ Model-based RL: estimate the transition model $p(s_{t+1} \mid s_t, a_t)$ and then
  - ✳ Use it for planning (no explicit policy)
  - ✳ Use it to improve a policy
  - ✳ …

Sample efficiency

How many interactions does RL algorithm need until it has learned the optimal policy/ Q-function/…?



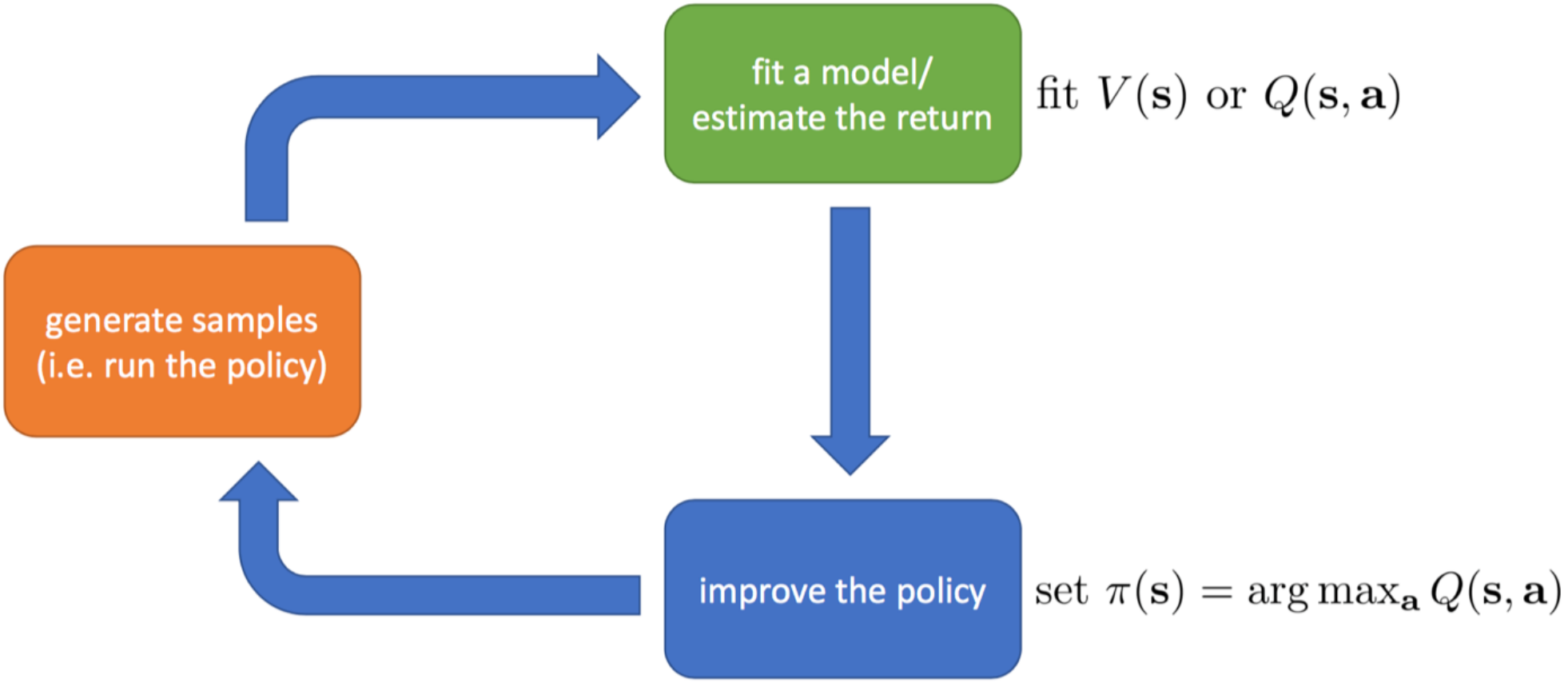
From course "Deep Reinforcement Learning", Sergey Levine

Machine time is expensive. Some algorithms are excluded on the machine (PPO,…)

→ because of algorithm simplicity we started with: Q-learning and Actor-critic methods

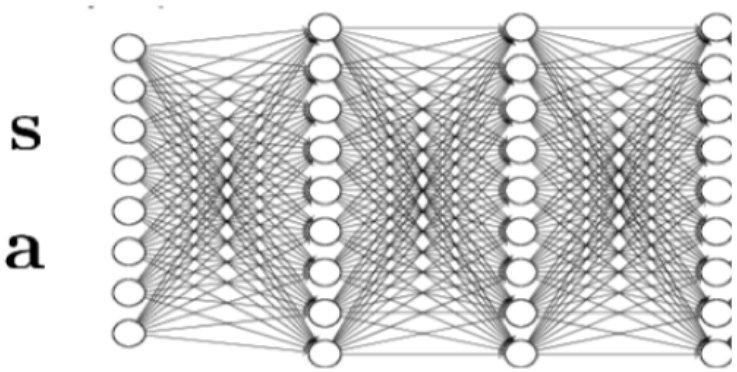
→ then moved to model-based RL: albeit only some methods studied so far

Basic Q-learning algorithm



Partly from course "Deep Reinforcement Learning", Sergey Levine

Basic Q-learning algorithm



$Q_\phi(\mathbf{s}, \mathbf{a})$
parameters ϕ

1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy

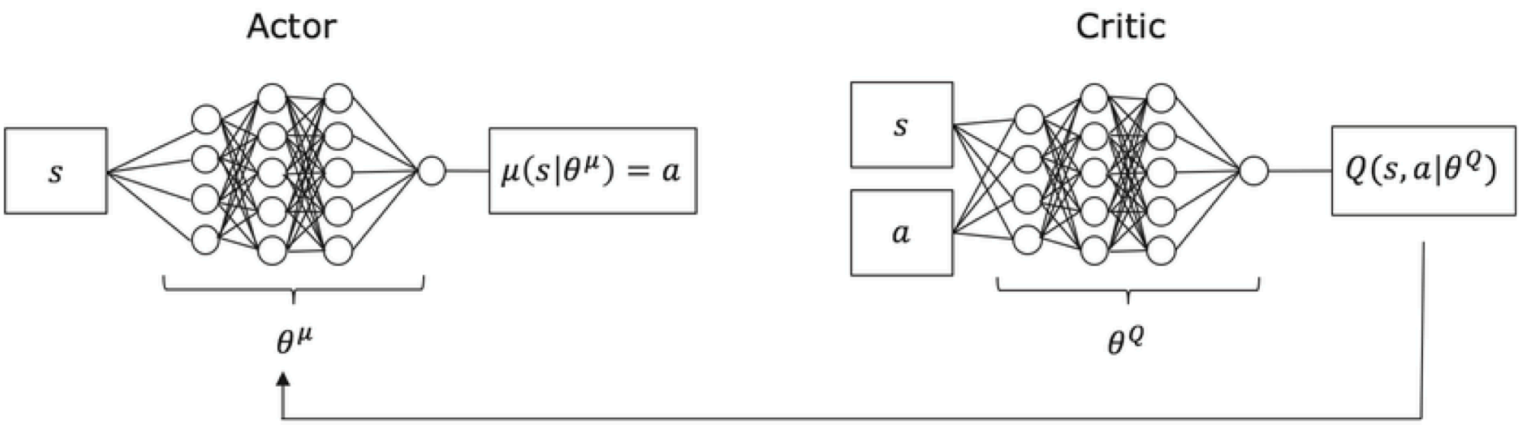
$K\times$

2. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \left\| Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i \right\|^2$

Issue for continuous actions a: $\max_a Q(s,a)$ in update rule and $\pi(s) = \arg\max_a Q(s,a)$; maximisation might be not be straight forward for non-trivial Q

Continuous actions - Actor-Critic

The Actor-Critic algorithm (simplest form DDPG)



Policy Gradient: $\nabla_{\theta^\mu}\mu = \mathbb{E}_\mu[\nabla_{\theta^\mu}Q(s,\mu(s|\theta^\mu)|\theta^Q)] = \mathbb{E}_\mu[\nabla_a Q(s,a|\theta^Q)\cdot\nabla_{\theta^\mu}\mu(s|\theta^\mu)]$

Main ingredients

- **Actor** *(= policy network)*: parameterized policy function, proposes action to given input state
- **Critic** *(= Q-net)*: like DQN, estimator for $Q(s,a)$, i.e. evaluates how good proposed action is to given state
- **Policy gradient: critic feeds back to actor** on (s, a) pair

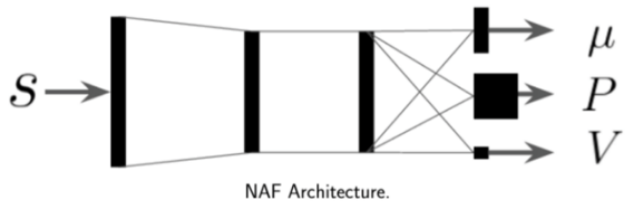
Q-learning with NAF

Various ways to overcome the Q maximisation issue with continuous action space.

If convex problem, can use a trick:

- ★ Q function is assumed to belong to function class that is easy to optimise. E.g. NAF (Normalised Advantage Function) algorithm

$$Q_\phi(\mathbf{s},\mathbf{a}) = -\frac{1}{2}(\mathbf{a}-\mu_\phi(\mathbf{s}))^T P_\phi(\mathbf{s})(\mathbf{a}-\mu_\phi(\mathbf{s})) + V_\phi(\mathbf{s})$$



NAF Architecture.

$$\arg\max_{\mathbf{a}} Q_\phi(\mathbf{s},\mathbf{a}) = \mu_\phi(\mathbf{s}) \qquad \max_{\mathbf{a}} Q_\phi(\mathbf{s},\mathbf{a}) = V_\phi(\mathbf{s})$$

Gu, Lillicrap, Sutskever, L., ICML 2016

Model-free RL test bed 2019

CERN accelerators in shutdown 2019 and most of 2020.

Except: AWAKE e^- line and commissioning run of H^- LINAC4

Initial test cases on AWAKE and later for LINAC4: **trajectory correction**

- ★ **ideal test case**
- ★ well defined state s
- ★ high dimensional action and state space
- ★ can compare with existing algorithms and can solve the problem analytically.

Goal: train controller that corrects as well as SVD → similar RMS and ideally within 1 iteration.

Implemented NAF with *Prioritised Experience Replay: arXiv:1511.05952*

Also used DDPG variant TD3 from package `stable-baselines` for **AWAKE optics matching**.

OpenAI Gym

CERN has python interface to accelerator control system: `pyjapc`

Key component for algorithm development and comparing algorithms: decision to implement all our problems as ***OpenAI Gym environments***

```
class Env(object):
    """The main OpenAI Gym class. It encapsulates an environment with
    arbitrary behind-the-scenes dynamics. An environment can be
    partially or fully observed.

    The main API methods that users of this class need to know are:

        step
        reset
        render
        close
        seed

    And set the following attributes:

        action_space: The Space object corresponding to valid actions
        observation_space: The Space object corresponding to valid observations
        reward_range: A tuple corresponding to the min and max possible rewards

    Note: a default reward range set to [-inf,+inf] already exists. Set it if you want a narrower range.

    The methods are accessed publicly as "step", "reset", etc...
    """
```

From OpenAI Gym GitHub: https://github.com/openai/gym/tree/master/gym

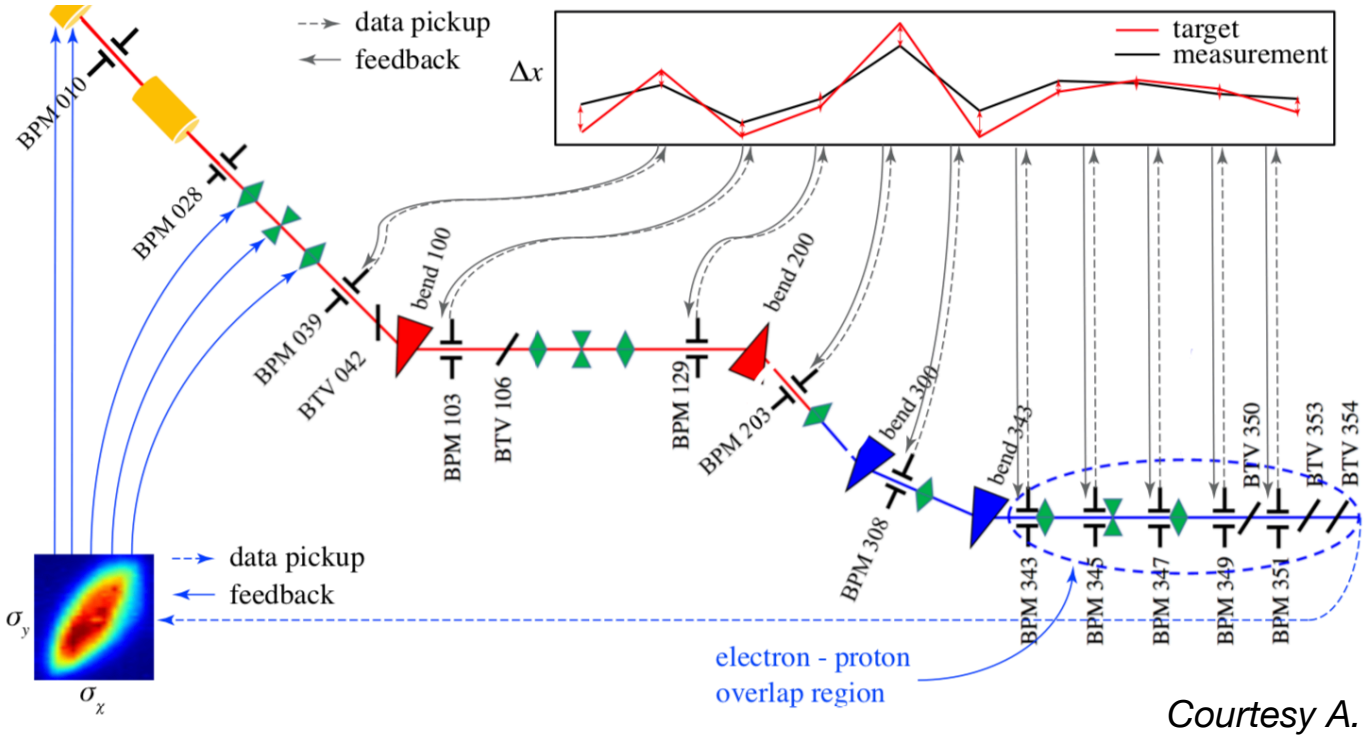
AWAKE

AWAKE: proton-driven plasma wakefield test facility.

e^- line: 20 MeV RF station, ~ 15 m transport to plasma cell

2 Gym environments:

- trajectory steering: 10 BPMs, 10 correctors
- auto-matching@ plasma entrance: BTV 354, 2 solenoids + 3 quadrupoles

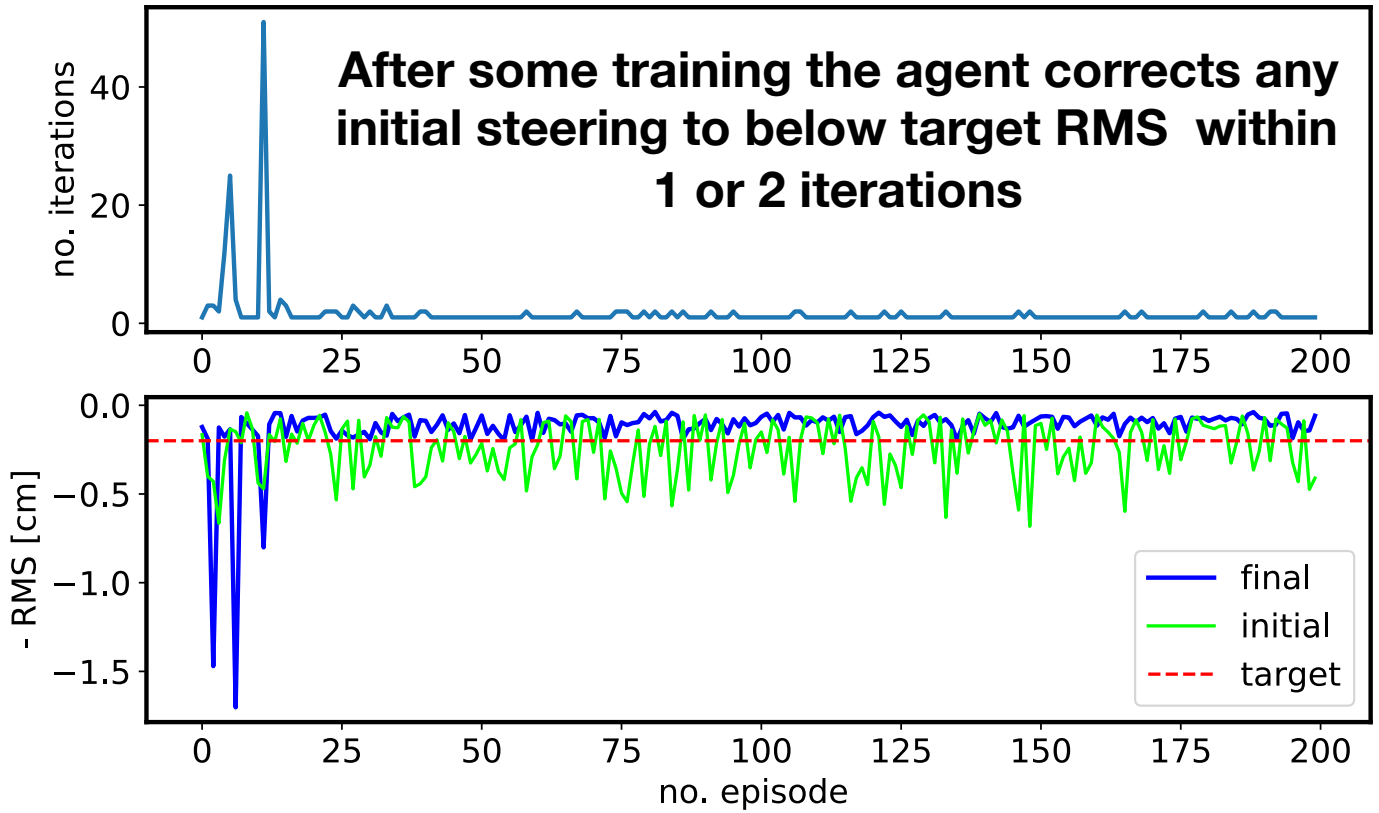


Courtesy A. Scheinker

Model-free online learning for AWAKE trajectory steering

Proof-of-principle: learn how to steer AWAKE e^- - line in H

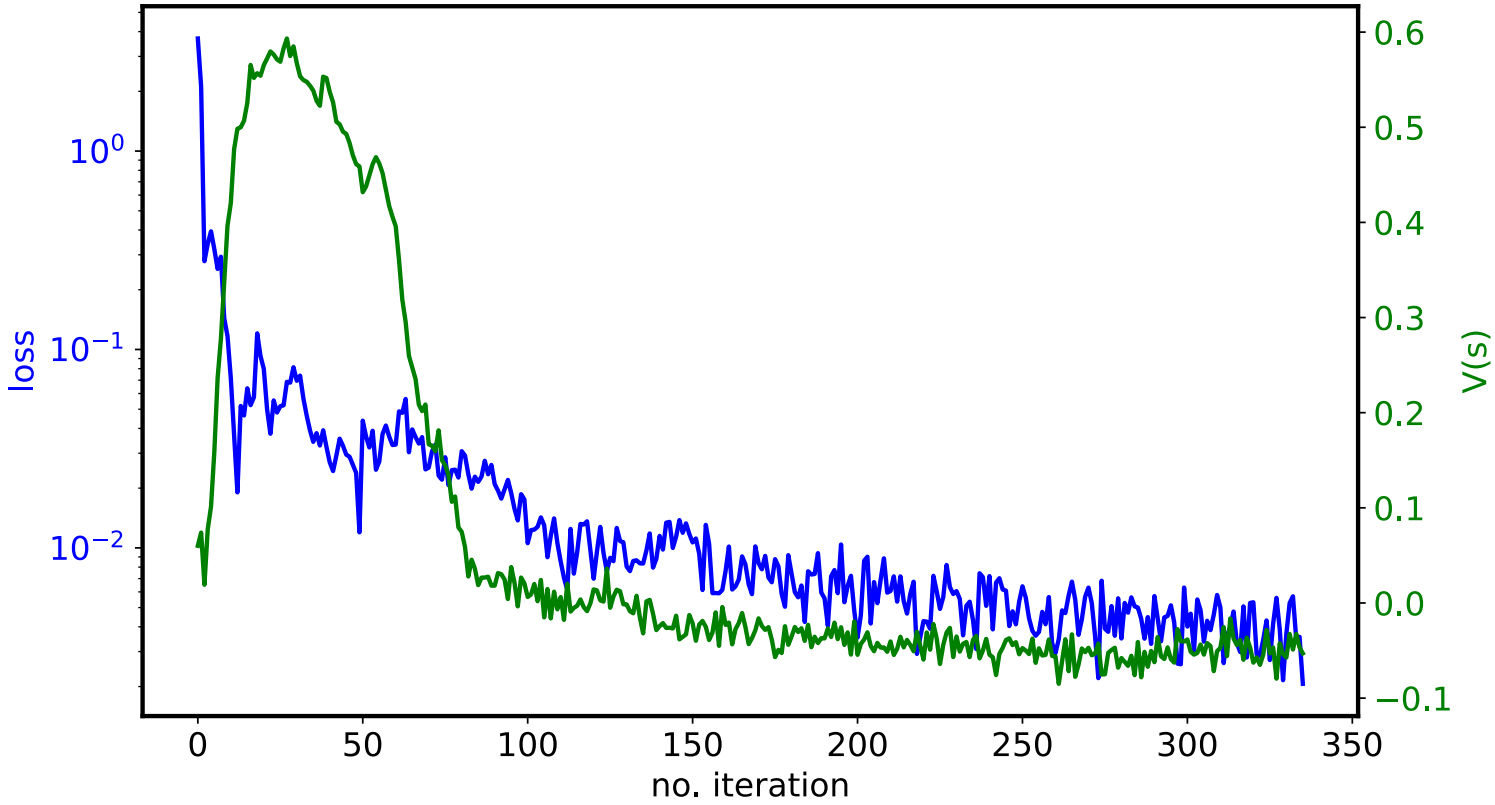
Q-learning with very sample-efficient NAF algorithm



Problem with 10 DOF

Training evolution

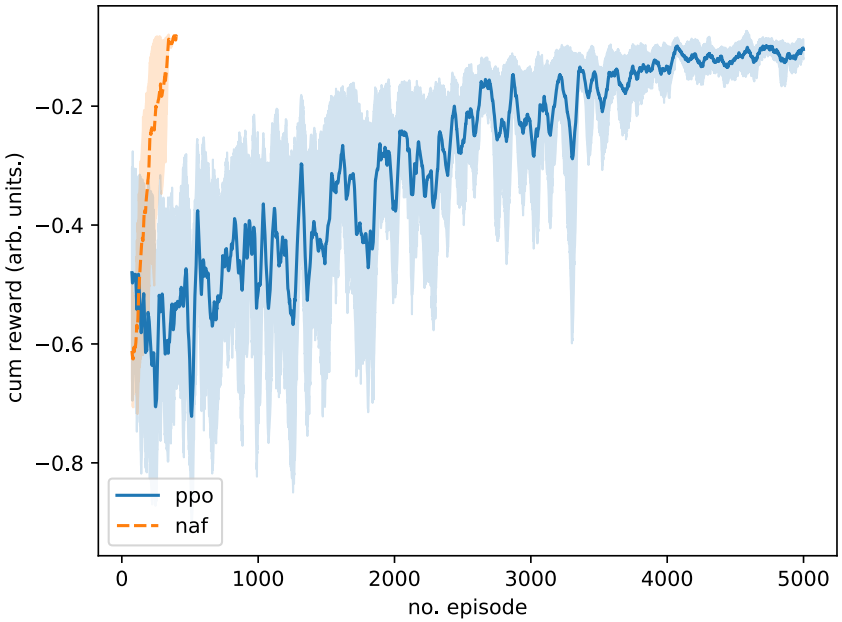
What does the training of the NAF networks look like?



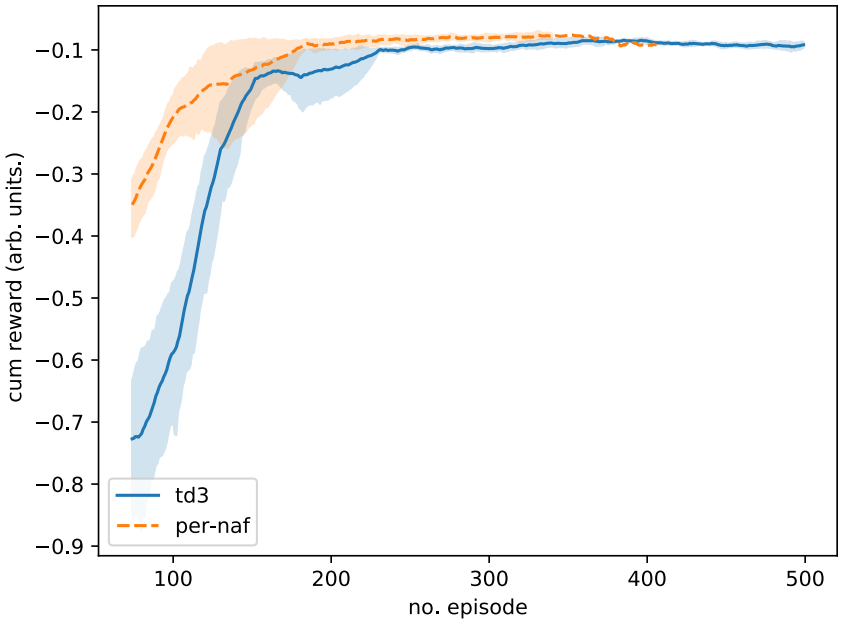
After ~ 90 iterations the agent starts correcting well $V(s)$ continues to improve for another ~100 iterations and converges at $V(s) \sim -0.05$.

Comparison with other algorithms

Policy-gradient algorithm PPO versus NAF for AWAKE steering problem in simulation:

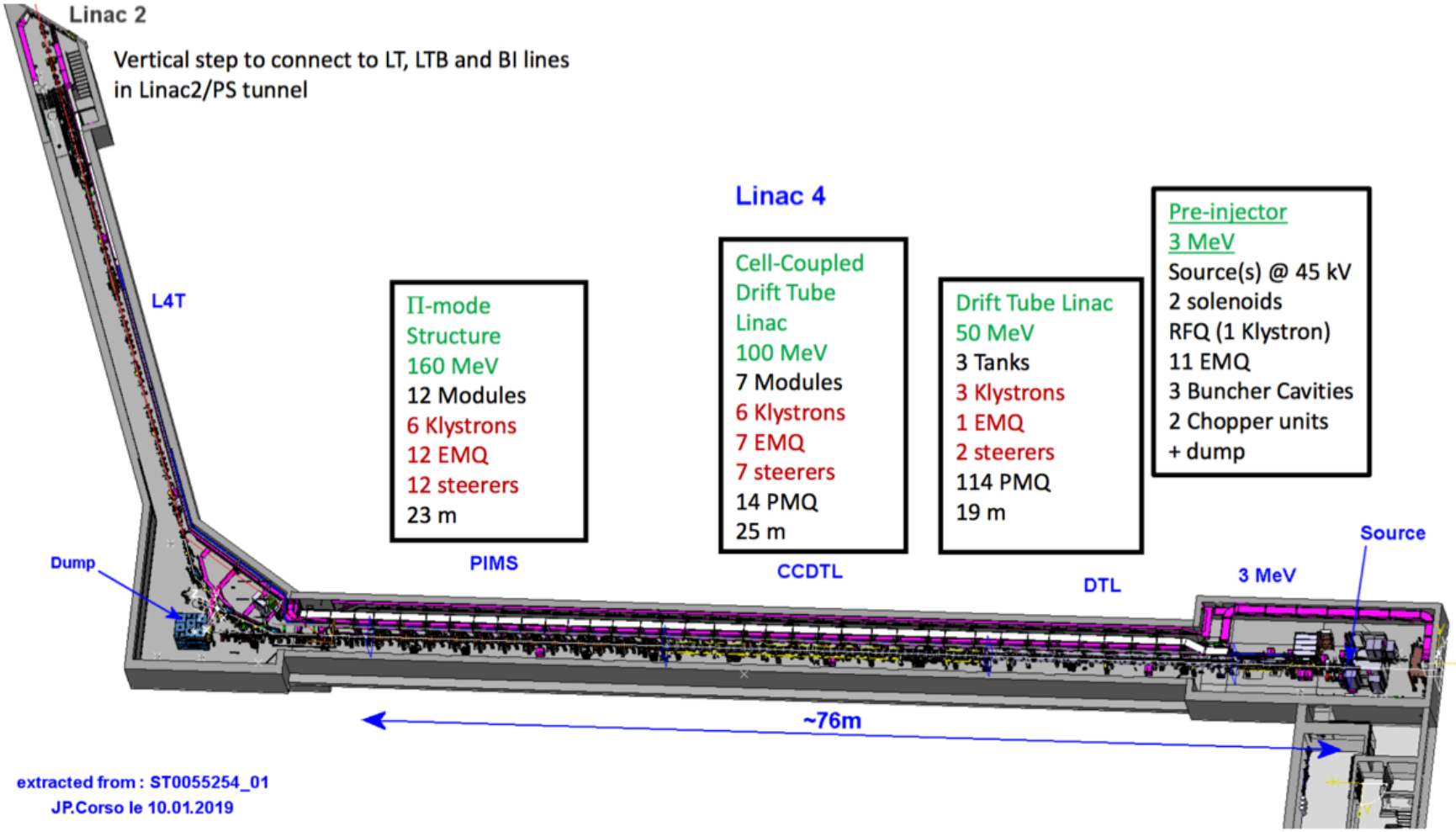


TD3 versus NAF for AWAKE steering problem in simulation: similar performance



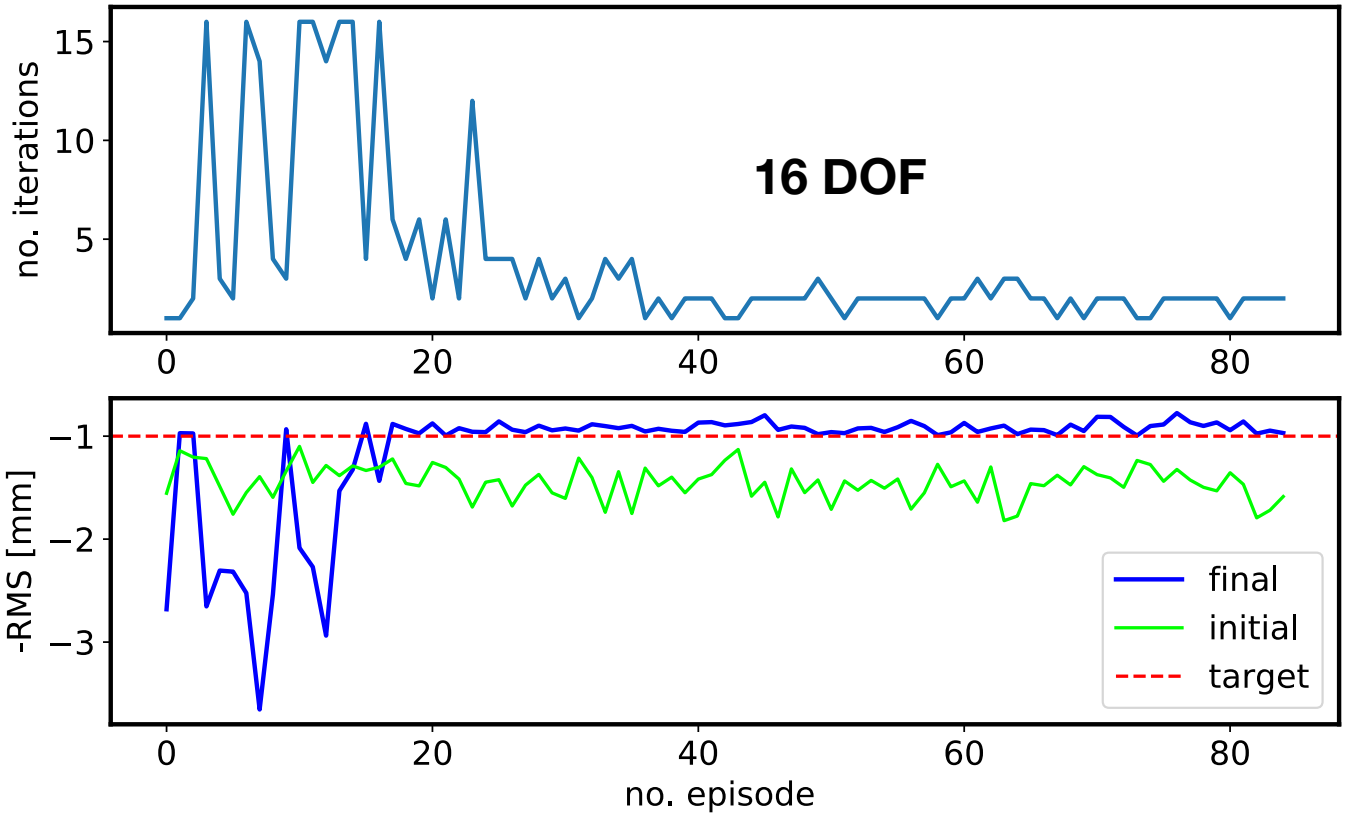
→ Q - **learning much more sample efficient than policy gradient algorithms**

Other example with NAF: agent for LINAC4 steering



Other example with NAF: agent for LINAC4 steering

Inexpensive way of learning any (also non-linear) response and solve control problem.



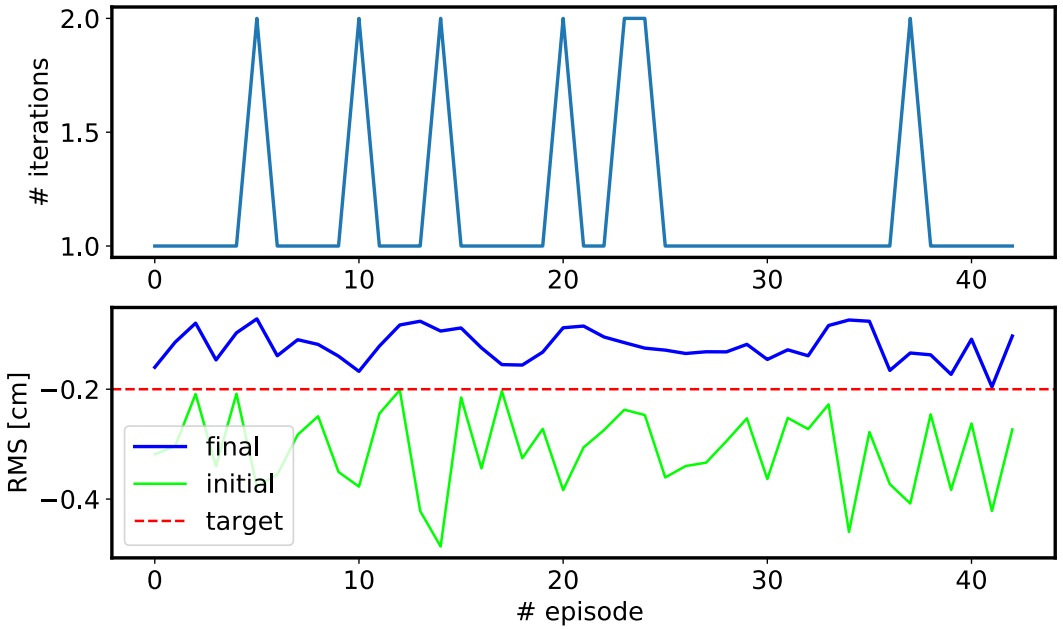
Train on simulation and apply on machine?

2 ways to circumvent the *sample efficiency* issue even further

→ Model-based RL: learn explicitly the model and train agent at the same time; see talk @ OWLE…add link

→ Train on simulation, apply on machine (transfer learning): typically relies on high level parameter control system and sufficiently good modelling

AWAKE training on simulation for trajectory steering; validation of trained agent on machine



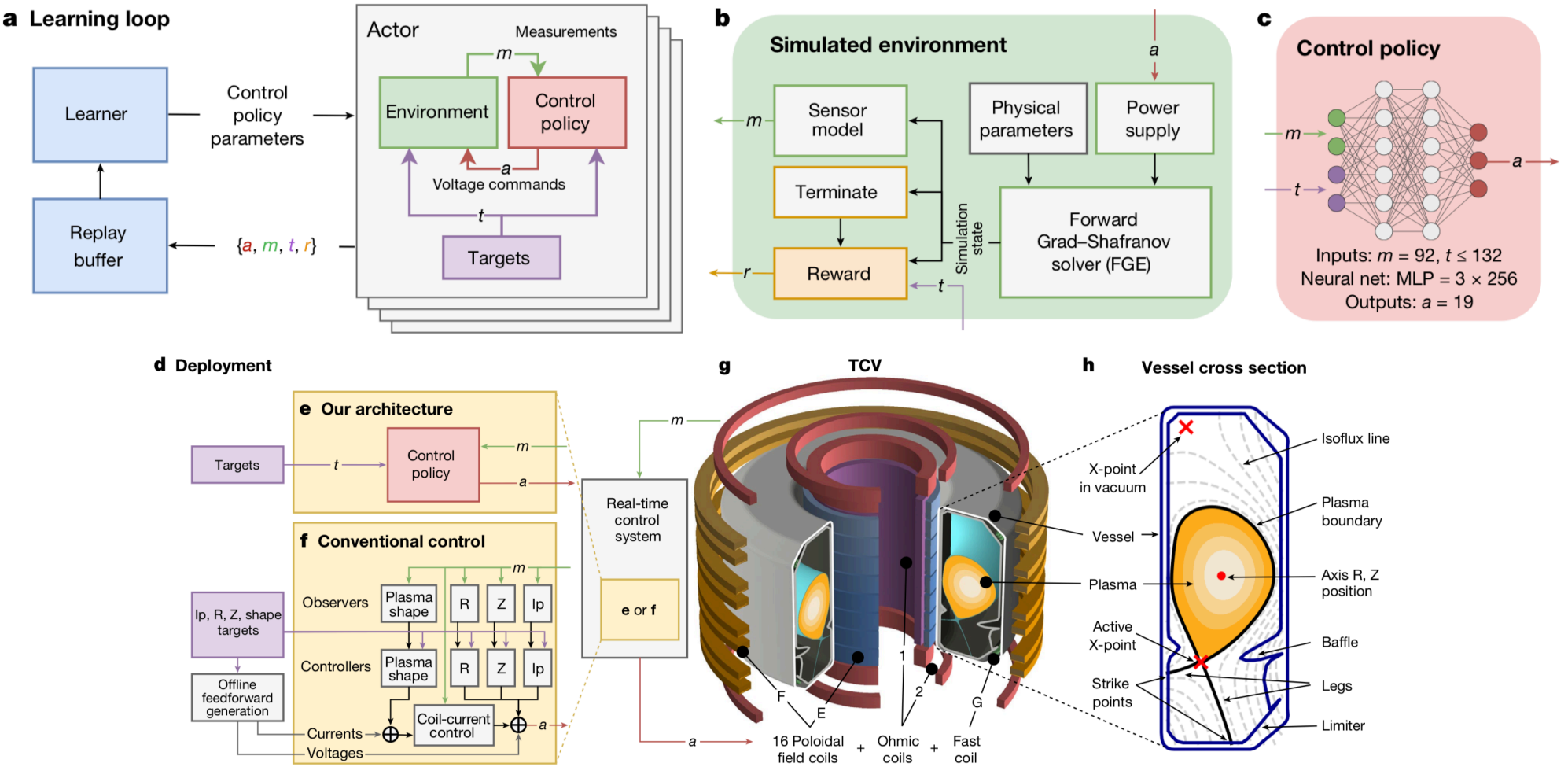
If simulation and machine not perfect match, could use "residual physics"

Deepmind and Tokamak control

Magnetic control of tokamak plasmas through deep reinforcement learning

https://doi.org/10.1038/s41586-021-04301-9

Time-varying, non-linear, multi-variate control problem solved with actor-critic agent

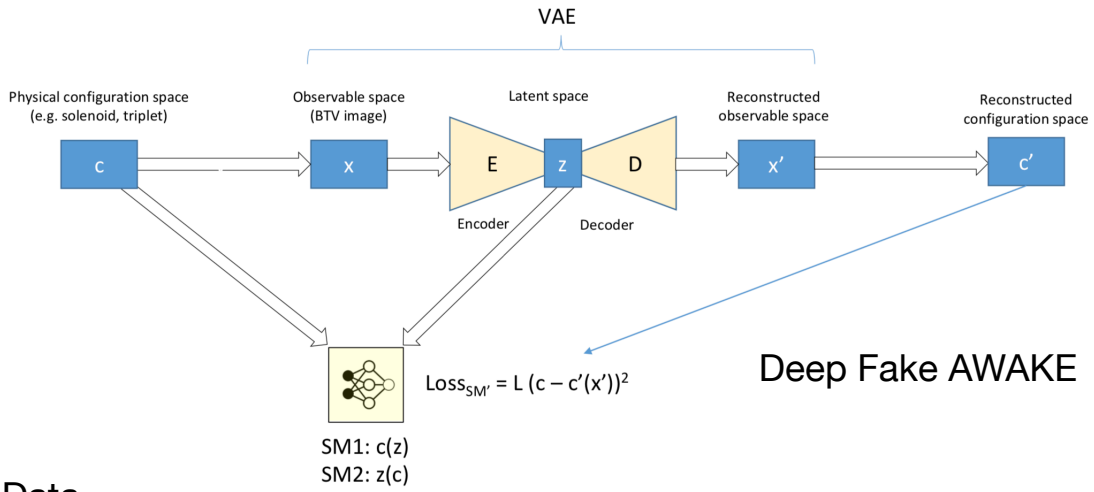


Example with TD3: Auto-matching at AWAKE

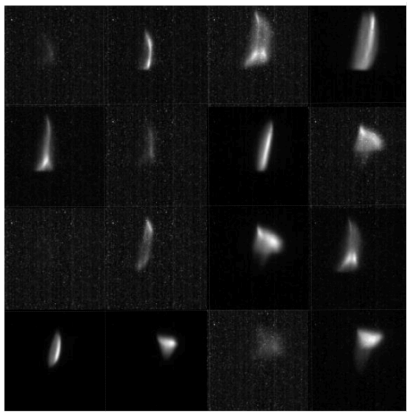
Problem: optimise spot size at plasma entrance.

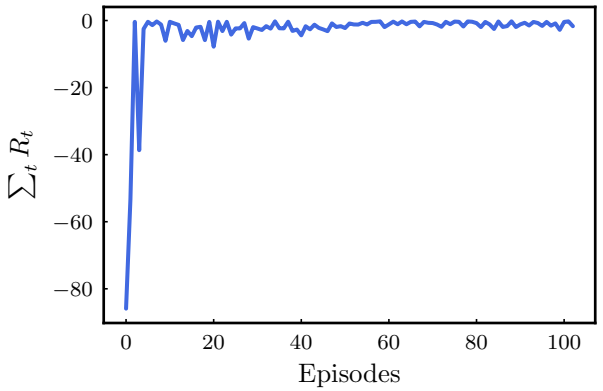
→ computer vision for state definition

→ automated how to establish reward threshold

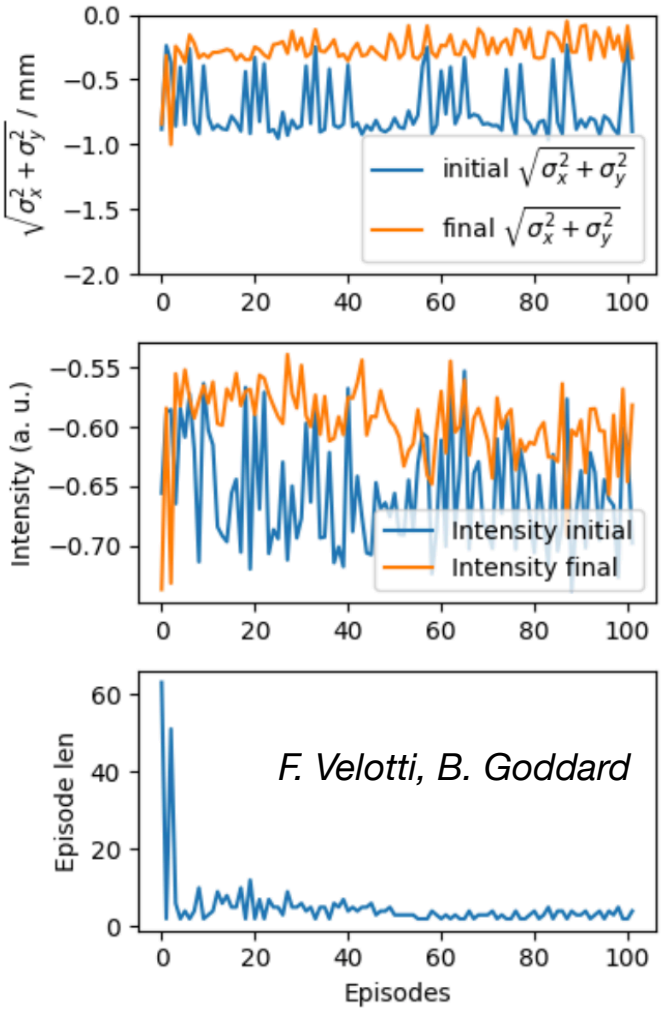


The Data



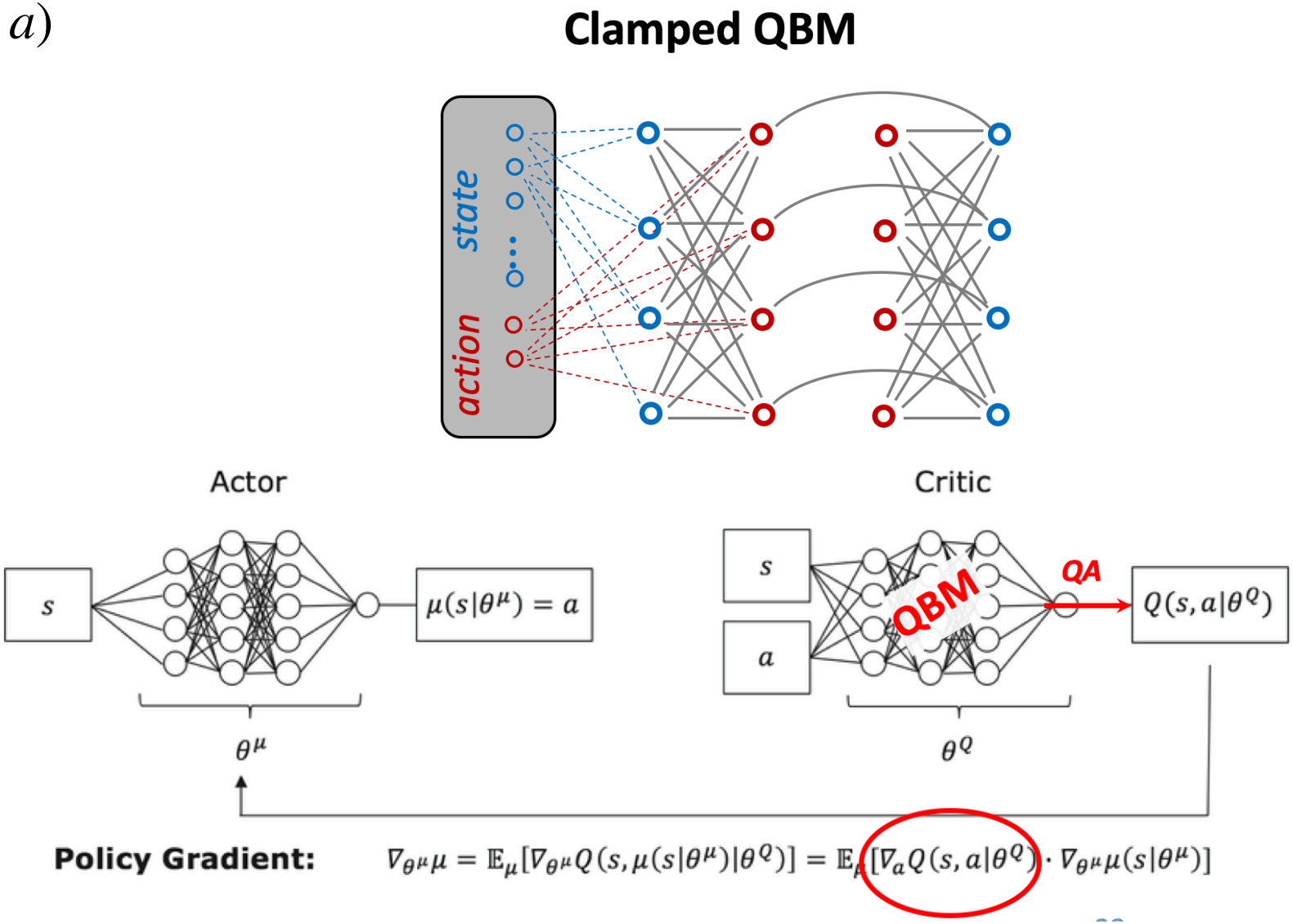


TD3 results:



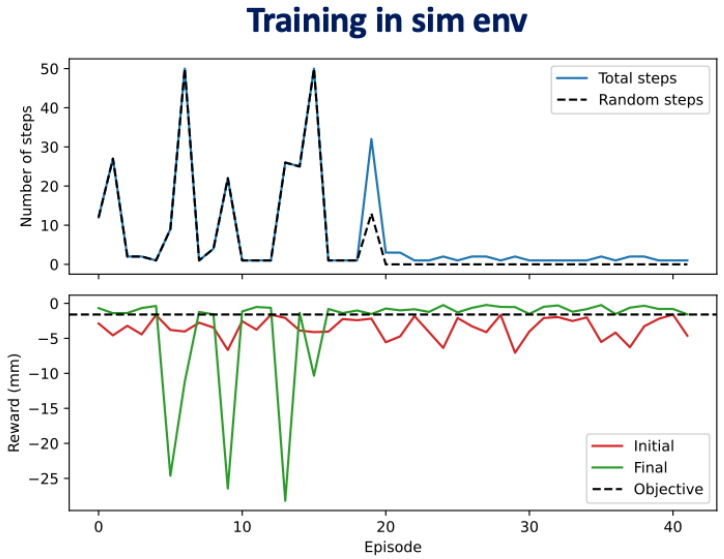
Quantum Actor-Critic

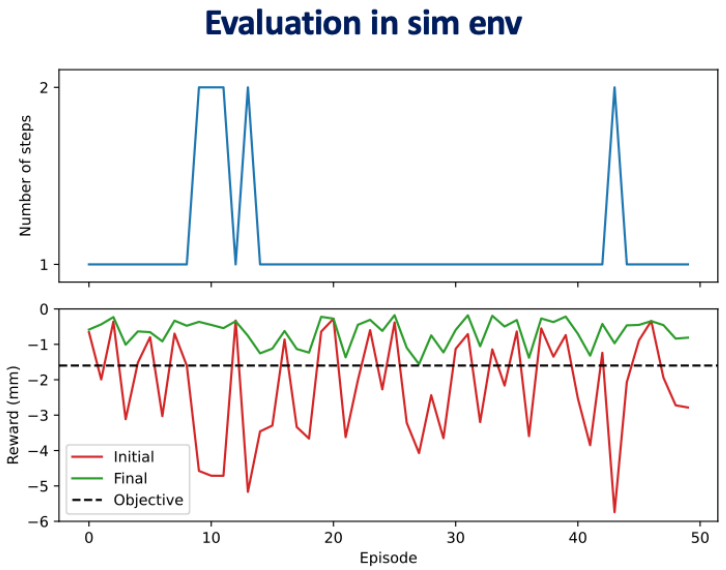
Use Free Energy of Clamped Quantum Boltzmann Machine (QBM) to model $Q(s,a)$



Quantum Actor-Critic @ AWAKE

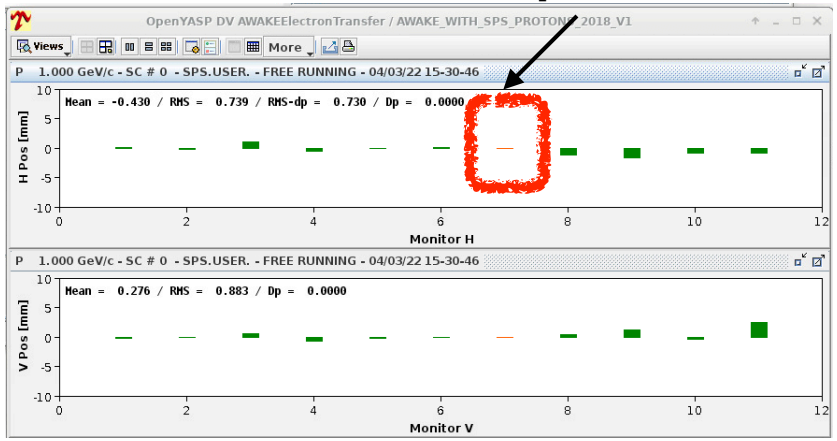
Training with Quantum Critic

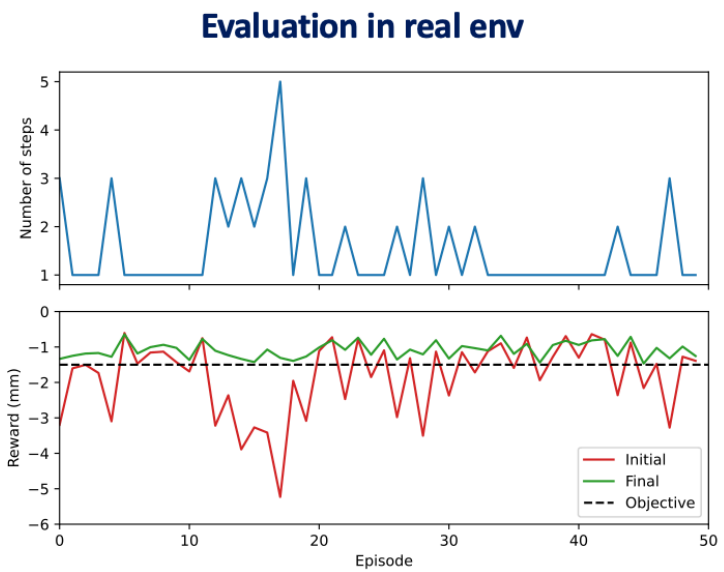




Successful evaluation no real e^- AWAKE line: only actor network

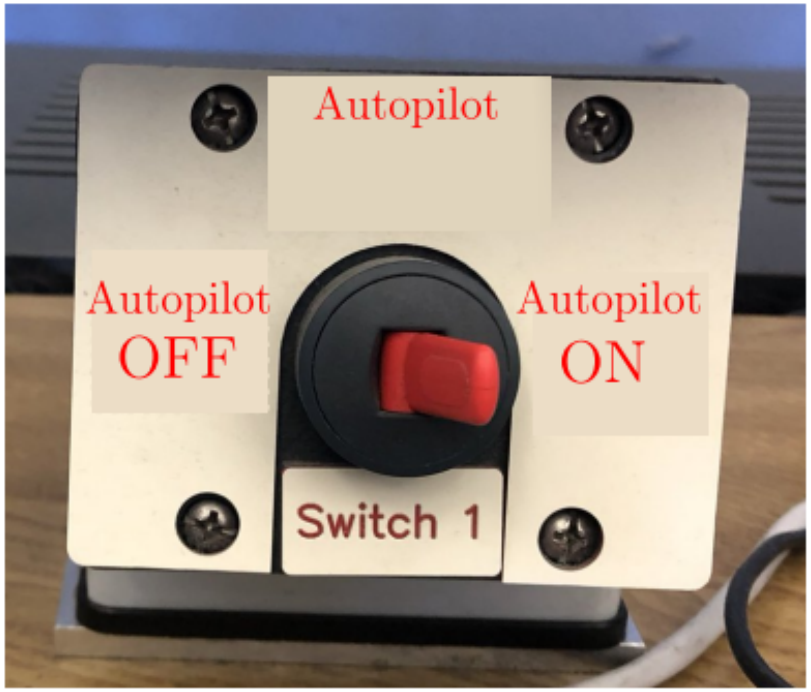
Broken beam position monitor





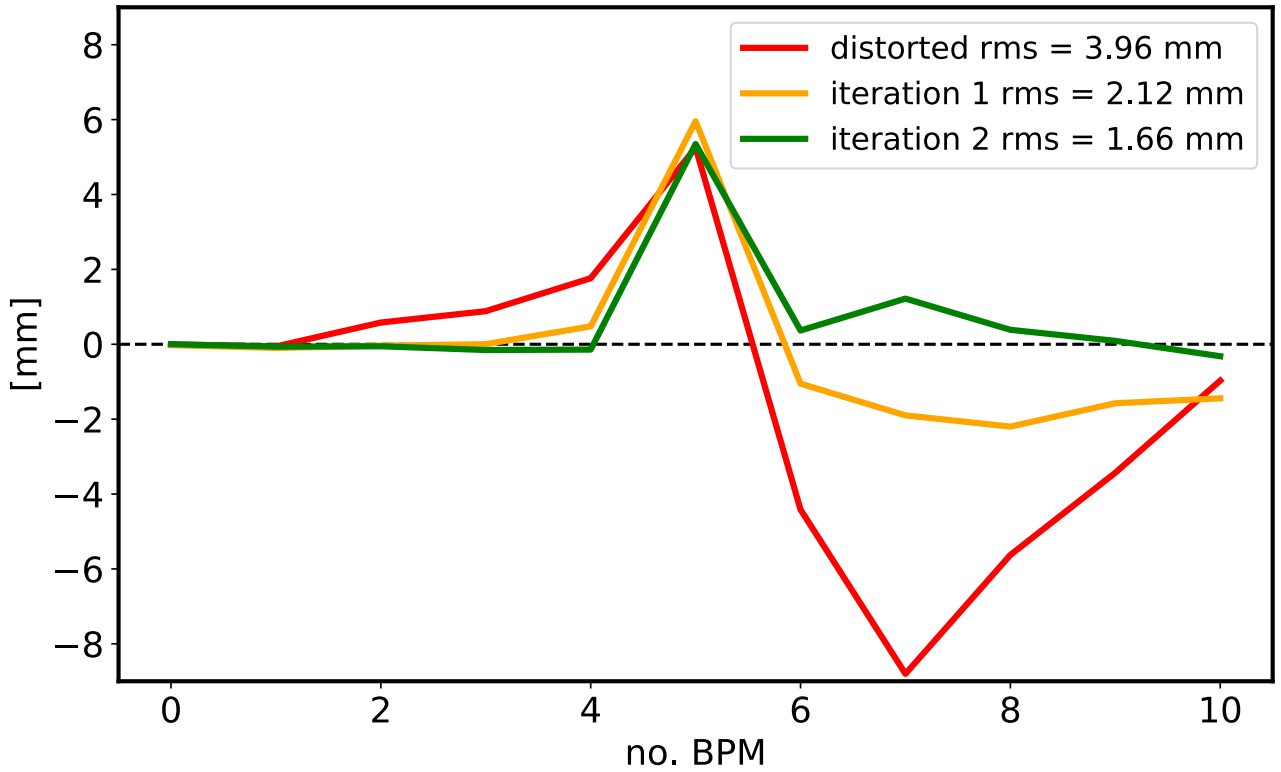
The end

CERN control room towards:



Extra

SVD on AWAKE


distorted rms = 3.96 mm
iteration 1 rms = 2.12 mm
iteration 2 rms = 1.66 mm
[mm]
no. BPM
8
6
4
2
0
−2
−4
−6
−8
0
2
4
6
8
10