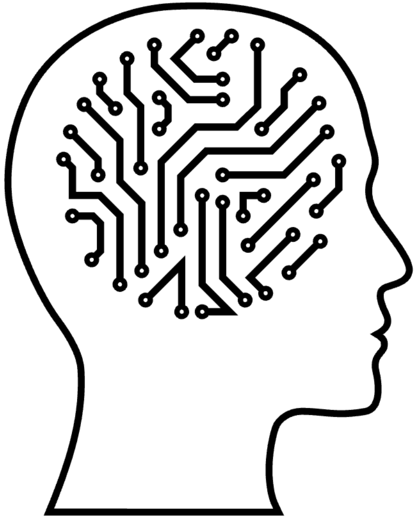


Reinforcement learning
and its application in particle accelerators

Andrea Santamaría García

12/09/2022 - Conceptual Advances in Deep Learning

Machine learning in the search for new fundamental physics

Georgia Karagiorgi, Gregor Kasieczka, Scott Kravitz, Benjamin Nachman & David Shih

Nature Reviews Physics **4**, 399–412 (2022) | Cite this article

Abstract

Compelling experimental evidence suggests the existence of new physics be
established and tested standard model of particle physics. Various current ar
experiments are searching for signatures of new physics. Despite the variety

Machine Learning Pins Down Cosmological Parameters

August 19, 2022 • *Physics* 15, s111

Cosmological constraints can be improved by applying machine learning to a combination of data from two leading probes of the large-scale structure of the Universe.

Pervasive machine learning in physics

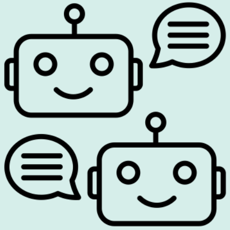
Nature Reviews Physics **4**, 353 (2022) | Cite this article

No longer restricted to data analysis, machine learning is now increasingly being used in theory, experiment and simulation – a sign that data-intensive science is starting to encompass all traditional aspects of research.

ARTIFICIAL INTELLIGENCE (AI)

Computers mimic human behaviour

- First chatbots
- Robotics
- Expert systems
- Natural language processing
- Fuzzy logic
- Explainable AI



Narrow AI

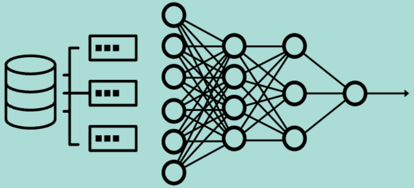
MACHINE LEARNING (ML)

Computers learn without being explicitly programmed to do so and improve with experience

Collection of **data-driven** methods / algorithms

Focused on **prediction** / **optimization / control** based on properties learned from data

Tries to **generalize** to unseen scenarios



Data + Algorithm

DEEP LEARNING (DL)

Multi-layered neural networks perform certain tasks with high accuracy

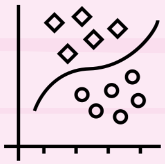
- Speech/handwriting recognition
- Language translation
- Recommendation engines
- Computer vision

MACHINE LEARNING

SUPERVISED LEARNING

Classification, prediction, forecasting

computer learns by example



- Spam detection
- Weather forecasting
- Housing prices prediction
- Stock market prediction

UNSUPERVISED LEARNING

Segmentation of data

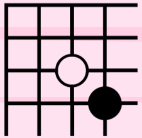
computer learns without prior information about the data

- Medical diagnosis
- Fraud (anomaly) detection
- Market segmentation
- Pattern recognition

REINFORCEMENT LEARNING

Real-time decisions

computer learns through trial and error



- Self-driving cars
- Make financial trades
- Gaming (AlphaGo)
- Robotics manipulation

Machine Learning

Learning style	Task	Popular algorithms
Supervised Learning We know the input & output (labeled data)	**Classification** discrete variables **Regression** continuous variables	▪ Neural networks (e.g. stochastic gradient descent, backpropagation) ▪ Support Vector Machine ▪ K-nearest neighbor ▪ Decision Tree algorithms (e.g. Classification and Regression Tree) ▪ Random Forest (ensemble) ▪ Uni or multivariate, linear or logistic
Unsupervised Learning We only know the input (unlabeled data)	**Clustering**	▪ K-means ▪ K-medians ▪ Expectation Maximization (EM) ▪ Hierarchical clustering
	Association	▪ Apriori algorithm ▪ Eclat algorithm
Reinforcement Learning	**Control** **Prediction**	▪ Value based (Q-learning) ▪ Policy based ▪ Actor critic ▪ Policy gradient or actor-critic ▪ Model-free or model based

Deep Learning Networks
- Convolutional Neural Networks
- Recurrent Neural Networks
- Long Short-Term Memory Networks
- Autoencoders
- Deep Boltzmann Machine
- Deep Belief Networks

Bayesian Algorithms
- Naive Bayes
- Gaussian Naive Bayes
- Bayesian Network
- Bayesian Belief Network
- Bayesian optimization

Regularization, dimensionality reduction, ensemble, evolutionary algorithms, computer vision, recommender systems, …

this slide is not exhaustive

Reinforcement learning

more than machine learning

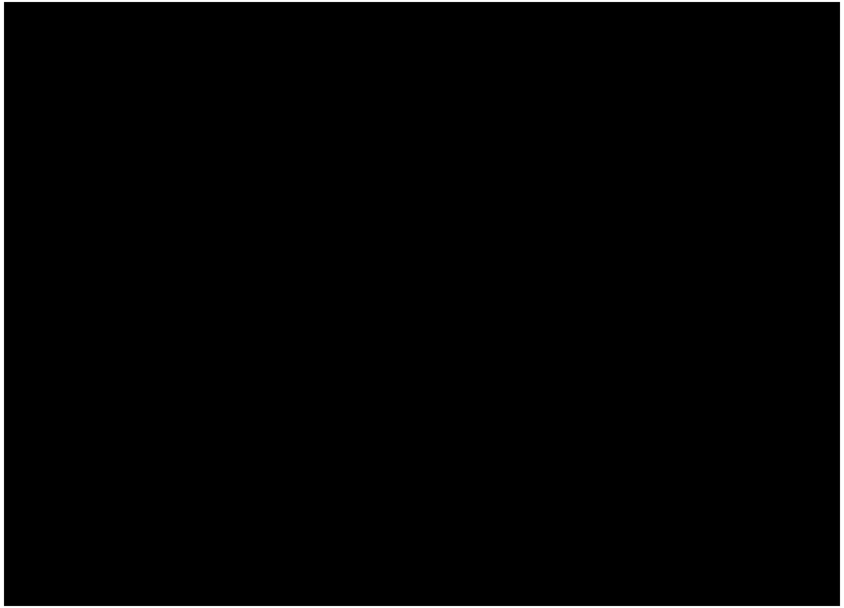
BEHAVIOR LEARNING

Psychology (classical conditioning)
Neuroscience (reward system)
Economics (game theory)
Mathematics (operations research)
Engineering (optimal control, planning)

Reinforcement learning

understanding how the human brain learns makes decisions

https://arxiv.org/abs/1707.02286



https://www.deepmind.com/publications/playing-atari-with-deep-reinforcement-learning

The RL problem

Reward hypothesis

all goals can be described by the maximization of expected cumulative sum of a received scalar signal

"Reward is enough"

Reward

scalar feedback signal $\mathcal{R}_t$ that indicates how well the agent is doing at step t

Goal

maximization of cumulative reward through selected actions

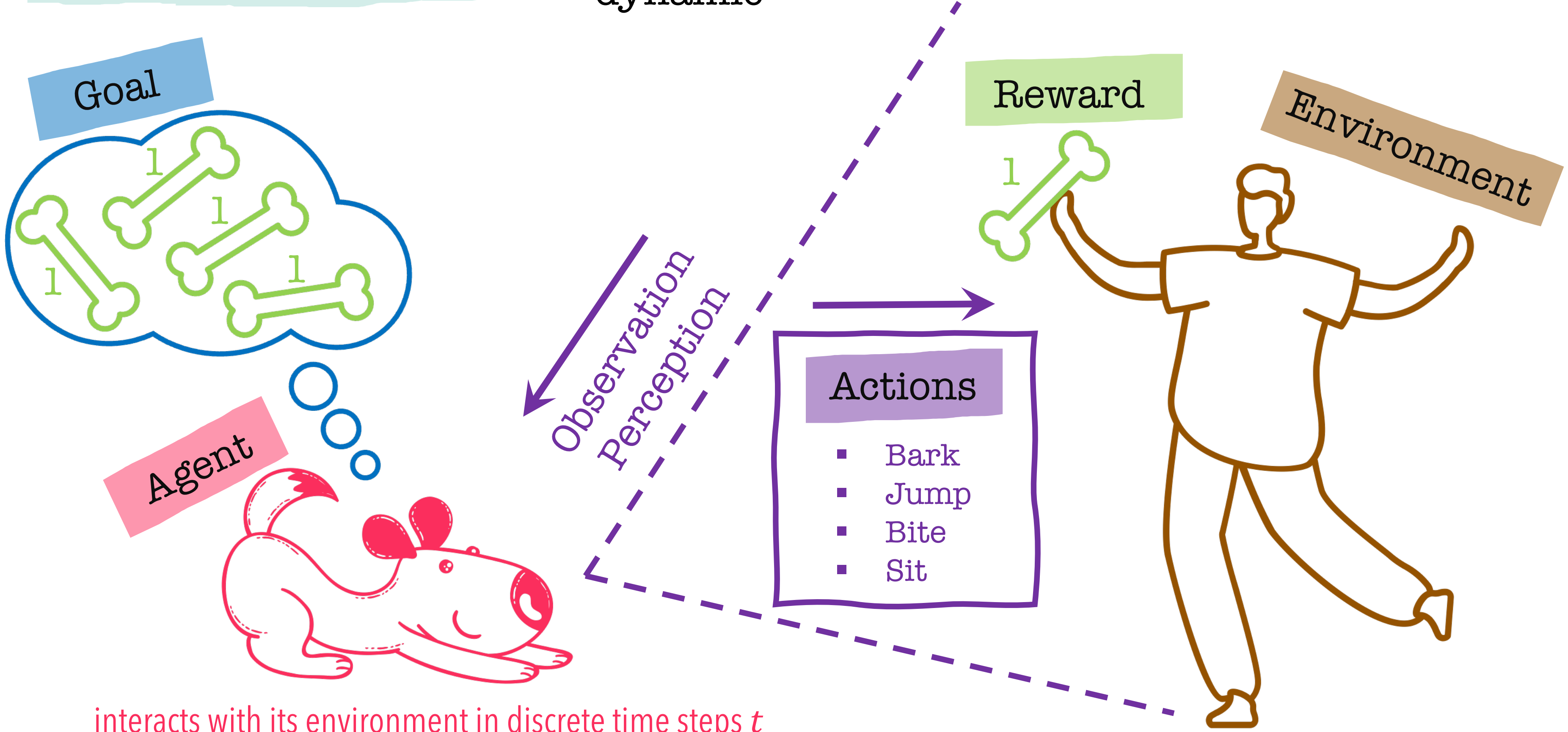
Agent

executes action
→ receives observation
→ receives scalar reward

an agent must learn through trial-and-error interactions with a dynamic environment

The RL problem

interactive
dynamic



interacts with its environment in discrete time steps t

Icons from the noun project

How to cumulate reward?

Agent

Model

agent's representation of the environment

Which behaviors perform well in this environment?

Policy agent's behaviour function (how the agent picks its actions)

Estimate the utility of taking actions in particular states of the environment (evaluation of the policy)

Value function how good each state and/or action are

- **Prediction**: evaluate the future given a policy
- **Control**: optimize the future (find the best policy)

Challenges in RL

Trade-off between exploitation and exploration

- Actions may have long-term consequences
- Reward might be delayed (does not happen immediately)

↳ should the agent sacrifice immediate reward to gain more long term reward?

The agent needs to:

✓ **Exploit** what it has already experienced in order to obtain reward now

✓ **Explore** the environment to select better actions in the future by sacrificing known reward now

…and both cannot be pursued exclusively without failing at the task

The agent

Must:

- Be able to **sense the state** of its environment to some extent
- Be able to **take actions** that affect that state
- **Have a goal** or goals relating to the state of the environment

Sensation

"Free-will"

Motivation

Markov Decision Processes

Include this 3 elements without trivializing any of them

Markov Decision Process (MDP)

Mathematical framework for modelling sequential decision making

A Markov Decision Process is a 5-tuple: $(\mathcal{S}, \mathcal{A}, \mathcal{P}^{a}_{ss'}, \mathcal{R}^{a}_{s}, \gamma)$ $\mathcal{S}$ = finite set of states

State information used to determine what happens next

A state transition can be:

- **Deterministic** $s_{t+1} = f(\mathcal{H}_t)$
- **Stochastic** $s_{t+1} \sim \mathbb{P}(s_{t+1} | \tau_t)$

Environment state ($\mathcal{S}^e$): environment's internal representation, usually not visible to the agent

Agent state ($\mathcal{S}^a$): agent's internal representation, used by the RL algorithm to pick the next action

Observation ($\mathcal{O}$):partial description of a state, which may omit information

Trajectory sequence of states and actions until time t

$$\tau = (s_0, a_0, s_1, a_1, s_2, a_2, \ldots)$$

Markov Decision Process (MDP)

Mathematical framework for modelling sequential decision making

A Markov Decision Process is a 5-tuple: $(\mathcal{S}, \mathcal{A}, \mathcal{P}_{ss'}^{a}, \mathcal{R}_{s}^{a}, \gamma)$ $\mathcal{S}$ = finite set of states

State information used to determine what happens next

A state transition can be:

- **Deterministic** $s_{t+1} = f(\mathcal{H}_t)$
- **Stochastic** $s_{t+1} \sim \mathbb{P}(s_{t+1} | \tau_t)$

Trajectory sequence of states and actions until time t

$$\tau = (s_0, a_0, s_1, a_1, s_2, a_2, \dots)$$

Markov state / property

A state is Markov if and only if:

$$\mathbb{P}[s_{t+1} | s_t] = \mathbb{P}\left[s_{t+1} | s_{1,\dots,t}\right]$$

- The state is a sufficient statistic of the future
- The future is independent of the past, given the present
- Once the state is known, the history may be discarded

state transitions of an MDP satisfy the Markov property

Fully observable environments $\mathcal{O}_t = \mathcal{S}_t^a = \mathcal{S}_t^e$

- Agent directly observes environment state
- Necessary condition to formalize an RL problem with an MDP

Partially observable environments $\mathcal{S}_t^a \neq \mathcal{S}_t^e$

Agent constructs its own state representation:

- Complete trajectory: $\mathcal{S}_t^a = \tau_t$
- Beliefs of environment state: $\mathcal{S}_t^a = (\mathbb{P}[\mathcal{S}_t^e = s_1], \dots, \mathbb{P}[\mathcal{S}_t^e = s_n])$
- Recurrent neural networks: $\mathcal{S}_t^a = \sigma(w_0 \mathcal{O}_t + w_s \mathcal{S}_{t-1}^a)$

→ Partially observable MDP

Markov Decision Process (MDP)

Mathematical framework for modelling sequential decision making

A Markov Decision Process is a 5-tuple: $(\mathcal{S}, \mathcal{A}, \mathcal{P}^a_{ss'}, \mathcal{R}^a_s, \gamma)$

State transition model / probability

Predicts the next state (dynamics of the environment)

$$\mathcal{P}^a_{ss'} = \mathbb{P}[\mathcal{S}_{t+1} = s' | \mathcal{S}_t = s, \mathcal{A} = a]$$

Probability of ending in state s' after taking action a while being in state s

$$\mathcal{P} = \begin{pmatrix} \boxed{\mathcal{P}_{11} & \cdots & \mathcal{P}_{1n}} \\ \vdots & \ddots & \vdots \\ \mathcal{P}_{n1} & \cdots & \mathcal{P}_{nn} \end{pmatrix} \Sigma{=}1$$

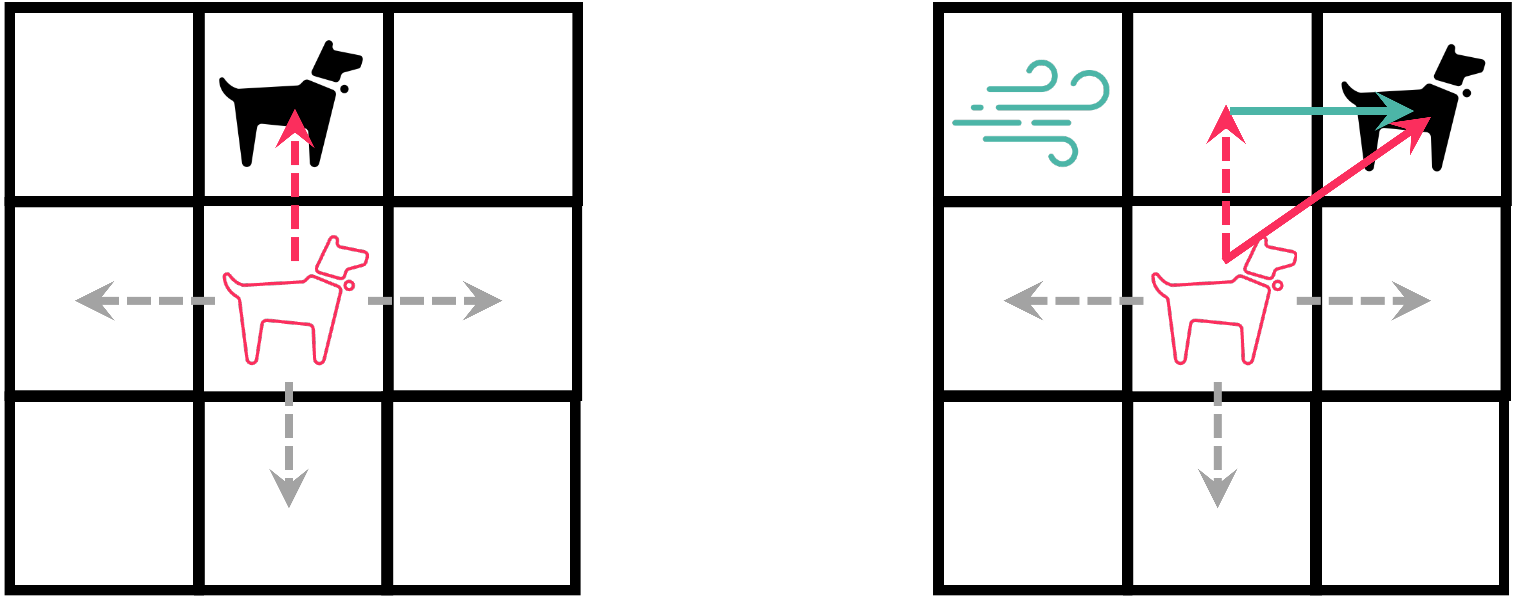
If probabilities change overtime = **non-stationary Markov process**

Transition probabilities from all states and successor states

Non-deterministic environment

Taking the same action in the same state on two different occasions may result in different next states

$t = t_0$ $\qquad$ $t = t_0 + \tau$



Markov Decision Process (MDP)

Mathematical framework for modelling sequential decision making

A Markov Decision Process is a 5-tuple: $(\mathcal{S}, \mathcal{A}, \mathcal{P}^a_{ss'}, \mathcal{R}^a_s, \gamma)$

Return Total discounted reward from time step t

$$\mathcal{G}_t = \mathcal{R}_{t+1} + \gamma \mathcal{R}_{t+2} + \cdots = \sum_{t=0}^{\infty} \gamma^t \mathcal{R}_{t+1}$$

"infinite-horizon discounted return"

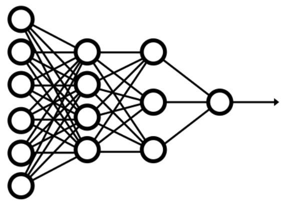
The goal is to maximize the return

- The discount factor $\gamma \in [0, 1)$ avoids infinite returns (sum converges)
- It values immediate reward over delayed reward (human-like)
- It deals with uncertainty about the future (no perfect model of env.)

Policy

Map from state
to action

- Policy π completely defines how the agent will behave
- It's a distribution over actions given a certain state



Deterministic: $a = \pi(s)$

Stochastic: $\pi(a|s) = \mathbb{P}[\mathcal{A}_t = a | \mathcal{S}_t = s]$

Probability of taking a specific
action by being in a specific state

Categorical (discrete action spaces)
Gaussian (continuous action spaces)

Given an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and a policy π:

$$\mathcal{P}^{\pi}_{s,s'} = \sum_{a \epsilon \mathcal{A}} \pi(a|s)\, \mathcal{P}^{a}_{s,s'} \qquad \mathcal{R}^{\pi}_{s} = \sum_{a \epsilon \mathcal{A}} \pi(a|s)\, \mathcal{R}^{a}_{s}$$

Value function

Estimation of expected future reward

A way to compare policies

- Used to choose between states depending on how much reward we expect to get
- Depends on the agent's behavior (policy)

State-value function

Expected return starting from state s and following policy π (evaluates the policy)

$$\mathcal{V}_{\pi}(s) = \mathbb{E}_{\pi}[\mathcal{G}_t \mid \mathcal{S}_t = s]$$

"on policy"

Action-value function

Expected return starting from state s , taking action a , and following policy π

$$\mathcal{Q}_{\pi}(s, a) = \mathbb{E}_{\pi}[\mathcal{G}_t \mid \mathcal{S}_t = s, \mathcal{A}_t = a]$$

"Q function"

Bellman optimality equation

The state-value function can be decomposed into:

- **immediate reward** $\mathcal{R}_{t+1}$
- **discounted value of next state** $\gamma\, v(\mathcal{S}_{t+1})$

$$\mathcal{V}(s) = \mathbb{E}[\mathcal{G}_t \mid \mathcal{S}_t = s]$$

$$= \mathbb{E}[\mathcal{R}_{t+1} + \gamma\, \mathcal{R}_{t+2} + \gamma^2\, \mathcal{R}_{t+3} \ldots \mid \mathcal{S}_t = s]$$

$$= \mathbb{E}[\mathcal{R}_{t+1} + \gamma\, (\mathcal{R}_{t+2} + \gamma\, \mathcal{R}_{t+3} \ldots) \mid \mathcal{S}_t = s]$$

$$= \mathbb{E}[\mathcal{R}_{t+1} + \gamma\, \mathcal{G}_{t+1} \mid \mathcal{S}_t = s]$$

$\mathbb{E}(\mathrm{f}) = \mathbb{E}(\mathbb{E}(\mathrm{f}))$

$$= \mathbb{E}[\mathcal{R}_{t+1} + \gamma\, \mathcal{V}(\mathcal{S}_{t+1}) \mid \mathcal{S}_t = s]$$

Reward you expect to get from being in your current state

Expected value of wherever state you land next

$$\boxed{\mathcal{V}(s) = \mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{s,s'}\, \mathcal{V}(s')}$$

Bellman expectation equation

Considering the policy π we get:

$$\boxed{\mathcal{V}(s) = \sum_{a \in \mathcal{A}} \pi\,(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{s,s'}^a \mathcal{V}(s') \right)}$$

Direct solution only for small MRPs

- System of $\mathcal{S}$ simultaneous linear equations with $\mathcal{S}$ unknowns

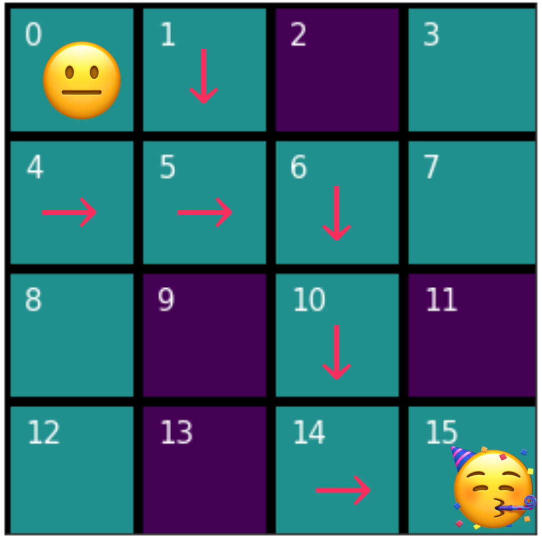
Other ways of solving it:

- Iteratively (dynamic programming)
- Sampling (Monte-Carlo evaluation)
- Approximation (temporal-difference learning)

Example: gridworld

The agent needs to get from state **0** to state **15** to get out of the maze

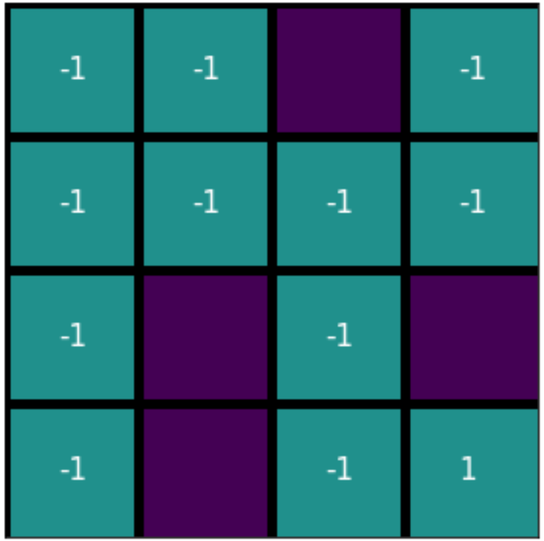
States



Actions $\mathcal{A} = (\uparrow, \downarrow, \leftarrow, \rightarrow)$

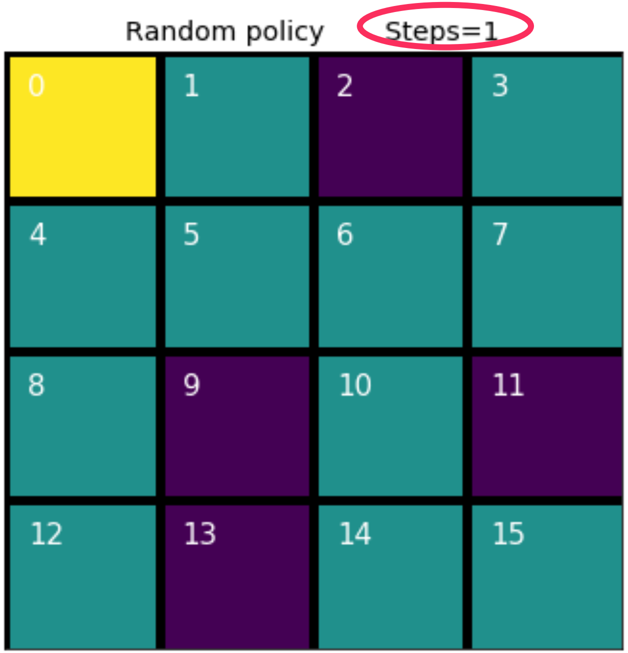
Deterministic env: $\mathcal{P}^{a}_{s,s'} = 1$

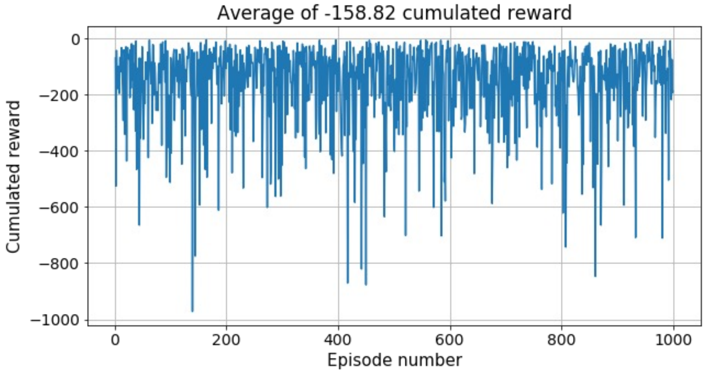
Rewards no discount γ



Example: gridworld

$$\textbf{Policy}\quad \pi(a|s) = \mathbb{P}[\mathcal{A}_t = a|\mathcal{S}_t = s] \longrightarrow \pi(a|s) = \overbrace{\mathbb{P}[\uparrow,\downarrow,\leftarrow,\rightarrow \mid \mathcal{S}_t] = 0.25}^{\text{random policy}}$$





MC prediction

Example: gridworld

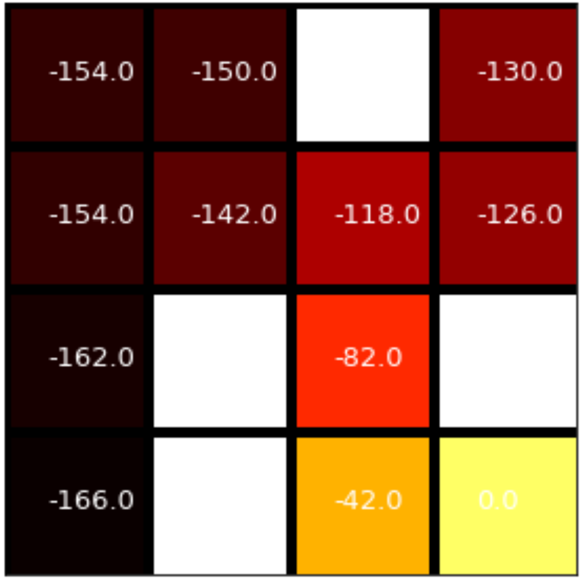
Value function

$$\mathcal{V}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{s,s'}^a \mathcal{V}(s') \right)$$

Solving simultaneously linear set of equations:
- environment's dynamics are completely known

```
0.5*v0 - 0.25*v1 - 0.25*v4 + 1.0 = 0
-0.25*v0 + 0.5*v1 - 0.25*v5 + 1.0 = 0
0.25*v3 - 0.25*v7 + 1.0 = 0
-0.25*v0 + 0.75*v4 - 0.25*v5 - 0.25*v8 + 1.0 = 0
-0.25*v1 - 0.25*v4 + 0.75*v5 - 0.25*v6 + 1.0 = 0
-0.25*v10 - 0.25*v5 + 0.75*v6 - 0.25*v7 + 1.0 = 0
-0.25*v3 - 0.25*v6 + 0.5*v7 + 1.0 = 0
-0.25*v12 - 0.25*v4 + 0.5*v8 + 1.0 = 0
0.5*v10 - 0.25*v14 - 0.25*v6 + 1.0 = 0
0.25*v12 - 0.25*v8 + 1.0 = 0
-0.25*v10 + 0.5*v14 + 0.5 = 0

11 variables,  11 equations
```



$\pi \rightarrow \mathcal{V}_\pi$ = policy evaluation

how much value this policy has?

Example: gridworld

Value function

$$\mathcal{V}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{s,s'}^a \mathcal{V}(s') \right)$$

Solving iteratively:

➢ Bellman equation becomes an update rule

$$\mathcal{V}_{k+1}(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{s,s'}^a \mathcal{V}_k(s') \right)$$

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated

$V \leftarrow \vec{0}, V' \leftarrow \vec{0}$

Loop:

$\quad \Delta \leftarrow 0$

$\quad$ Loop for each $s \in \mathcal{S}$:

$$V'(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')]$$

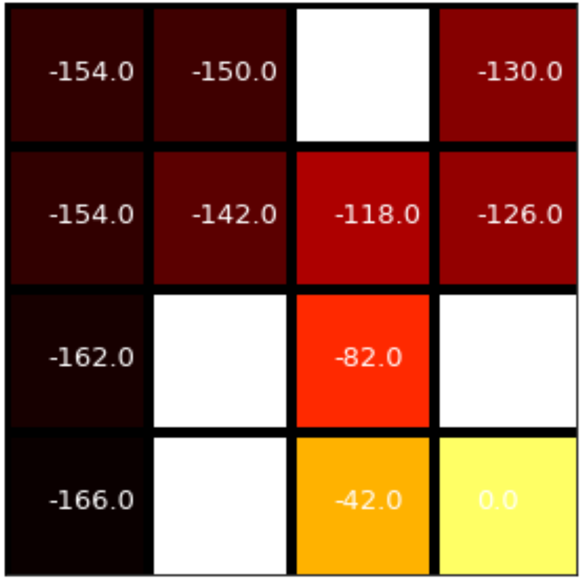
$$\Delta \leftarrow \max(\Delta, |V'(s) - V(s)|)$$

$\quad V \leftarrow V'$

until $\Delta < \theta$ (a small positive number)

Output $V \approx v_\pi$

Coursera



$\pi \rightarrow \mathcal{V}_\pi$ = **policy evaluation**

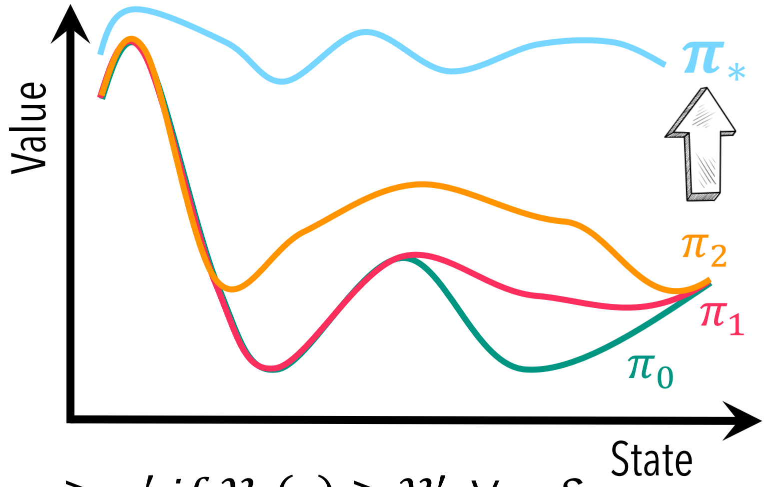
how much value this policy has?

Dynamic programming algorithms

turn the Bellman eq. into update rules

- ✅ **Prediction**: what's the value for a specific policy?
- **Control**: which policy gives as much reward as possible?
  → the policy with more value!

$\pi_* \geq \pi_2 \geq \pi_1 \geq \pi_0$



$\pi \geq \pi' \; if \; \mathcal{V}_\pi(s) \geq \mathcal{V}'_\pi, \forall \; s \epsilon \mathcal{S}$

For any MDP:

- There exists an <u>optimal policy $\boldsymbol{\pi}_*$</u> that is better or equal to all other policies $\pi_* \geq \pi \; \forall \pi$
- All optimal policies achieve the optimal value function $\mathcal{V}_{\pi_*} = \mathcal{V}_*(s)$ and $Q_{\pi_*} = Q_*(s,a)$

So...do I have to calculate the value of every policy and compare them?

$|\mathcal{A}|^{|\mathcal{S}|}$ deterministic policies in an MDP

$4^{11} \approx 4$ million policies for simple gridworld example

Bellman optimality equations

$$\mathcal{V}_{\pi*}(s) = \mathbb{E}_{\pi*}[\mathcal{G}_t \mid \mathcal{S}_t = s] = \max_{\pi} \mathcal{V}_{\pi}(s) \qquad \forall\, s \epsilon \mathcal{S}$$

$$\mathcal{Q}_{\pi*}(s) = \max_{\pi} \mathcal{Q}_{\pi}(s) \qquad \forall\, s \epsilon \mathcal{S}, a \epsilon \mathcal{A}$$

Optimal value functions

By replacing the optimal policy on the Bellman equations we get:

π_* assigns probability 1 to the action that receives the highest value

$$\boxed{\begin{aligned} \boldsymbol{\mathcal{V}}_*(\boldsymbol{s}) &= \max_a \left(\mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{s,s'} \mathcal{V}_*(s') \right) \\ \boldsymbol{\mathcal{Q}}_*(\boldsymbol{s}, \boldsymbol{a}) &= \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{s,s'}^a \max_{a'} \mathcal{Q}_*(s', a') \end{aligned}}$$

maximum value over every next possible state

- Nonlinear (max), no closed-form solution
- Dynamic programming solutions only applicable if the dynamics of the system $\mathcal{P}$ are known

Determining an optimal policy

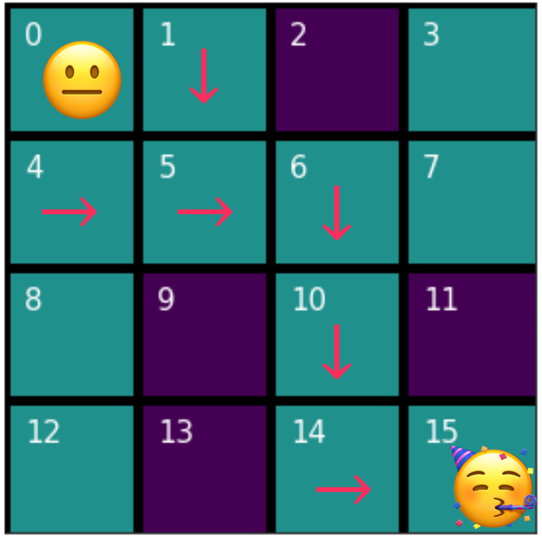
$$\boldsymbol{\mathcal{V}}_*(\boldsymbol{s}) = \max_a \left(\mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{s,s'} \mathcal{V}_*(s') \right)$$

maximum over all actions

For any state we look at each available action and take the one that maximizes the argument

$$\boldsymbol{\pi}_*(\boldsymbol{s}) = \operatorname*{argmax}_a \left(\mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{s,s'} \mathcal{V}_*(s') \right)$$

particular action that
achieves that maximum
(greedy action)

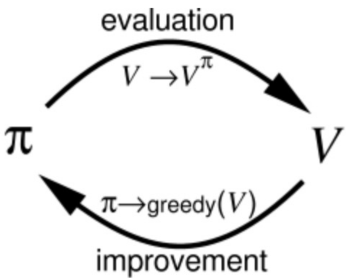


$$\boldsymbol{\pi}_*(\boldsymbol{s}) = \operatorname*{argmax}_a Q_*$$

Policy improvement & iteration

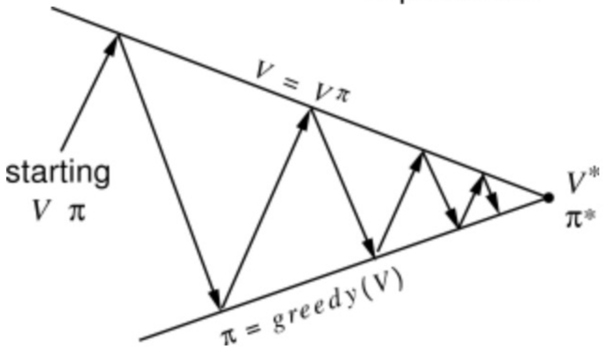
Let's consider a value function $\mathcal{V}_\pi$ that is non-optimal, and we select an action that is greedy with respect to it:

$$\pi_1 \xrightarrow{\text{evaluation}} \mathcal{V}_{\pi 1} \xrightarrow{\text{improvement}} \pi_2 \rightarrow \cdots \rightarrow \pi_*$$



$$\boldsymbol{\pi}'(\boldsymbol{s}) = \underset{a}{\operatorname{argmax}} \left(\mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{s,s'} \mathcal{V}_\pi(s') \right)$$

- If the action has a higher value, the policy is better
- $\mathcal{V}_*$ is the unique solution to the Bellman optimality eq.
- If this greedy operation does not change $\mathcal{V}$, then it converged to the optimal policy because it satisfies the Bellman optimality eq.



Images from http://incompleteideas.net/book/ebook/node46.html

Dynamic programming algorithms

turn the Bellman eq.
into update rules

Problem	Bellman equation	Algorithm	Sample-based version
Prediction	Expectation equation	Iterative policy evaluation	Temporal difference
Control	Expectation equation + greedy policy	Policy iteration	Sarsa
Control	Optimality equation	Value iteration	Q-learning

when we don't know $\mathcal{P}$

Off-policy learning

On-policy: improve and evaluate the policy being used to select actions

Off-policy: improve and evaluate a different policy from the one used to select actions

- ➢ Learn a target policy π (optimal policy) while…
- ➢ …selecting actions from behavior policy b (exploratory policy)

Provides another strategy for continuous exploration (experiences a larger # of states)

Temporal difference learning

Learning method specialized for multi-step **prediction learning**

- TD learning is learning a prediction from another, later learned prediction
  - learning a guess from a guess (you don't know the true $\mathcal{V}$)

$$\mathcal{V}(s) \leftarrow \mathcal{V}(s) + \alpha[\mathcal{R} + \gamma\mathcal{V}(s') - \mathcal{V}(s)]$$

- Difference between both predictions = temporal difference
- No $\mathcal{P}$ model needed (unlike in dynamic programming)

- Allows you to estimate the value function before the episode is finished
- Making long-term predictions is exponentially complex
  - Memory scales with the #steps of the prediction
- TD model = standard model of reward systems in the brain

Q-learning

Off-policy TD control

$$\mathcal{Q}(s,a) \leftarrow \mathcal{Q}(s,a) + \alpha[\mathcal{R} + \gamma \max \mathcal{Q}(s',a) - \mathcal{Q}(s,a)]$$

Converges to the optimal value function as long as the agent continues to explore sampling the state-action space

Overview of RL methods

Tabular solution methods
- ➢ Iterative (dynamic programming)
- ➢ Sample-based (Monte-Carlo evaluation)
- ➢ Temporal-difference learning

- Used to solve finite MDPs
- Value functions are stored as arrays (tables)
- Methods can often find exact solutions

In real-life situations, we cannot store the values of each possible state in an array, especially in continuous problems

- ➢ Autonomous driving: array per possible image the camera sees?

Approximate solution methods
- ➢ Value-based
- ➢ Policy-based
- ➢ Policy gradient
- ➢ Actor-critic

- Approximate value by function parametrized by a weight vector
  --> **neural networks (learning!)**
- Applicable to partially observable problems

Approximate solution methods

Value-based
contains a value function, policy is implicit

- Sample efficient
- Computationally fast
- Unstable (bias, don't know true $\mathcal{V}$)

DQN, NAF

Policy-based
does not store the value function, only the policy

Policy gradient
optimizes parametrized policies with gradient descent

- Convergence guarantees
- Sensitive to stepsize choice
- Poor sample efficiency
- Large variance

PPO, TD3, DDPG

Actor-critic
stores both the policy and value function

ACER, A2C/A3C, SAC

	Description	Policy	Action space	State space	Operator
DQN	Deep Q Network	Off-policy	Discrete	Continuous	Q-value
DDPG	Deep Deterministic Policy Gradient	Off-policy	Continuous	Continuous	Q-value
A3C	Asynchronous Advantage Actor-Critic Algorithm	On-policy	Continuous	Continuous	Advantage
TRPO	Trust Region Policy Optimization	On-policy	Continuous	Continuous	Advantage
PPO	Proximal Policy Optimization	On-policy	Continuous	Continuous	Advantage
TD3	Twin Delayed Deep Deterministic Policy Gradient	Off-policy	Continuous	Continuous	Q-value
SAC	Soft Actor Critic	Off-policy	Continuous	Continuous	Advantage

Model-free

The agent simply relies on some trial-and-error experience for action selection

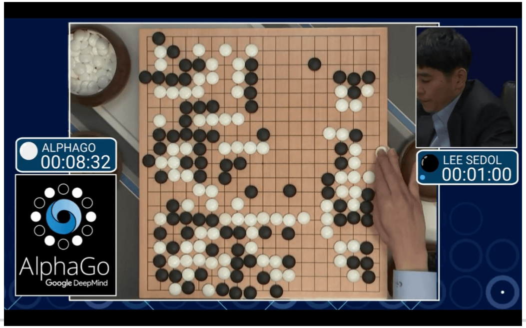
- The environment is initially unknown
- The agent interacts with the environment
- The agent improves its policy

all algorithms from previous slide

Model-based

Predictive model: "what will happen if I take this action?"

- The environment is known
- The agent performs internal computations with its model without external interaction
- The agent improves its policy

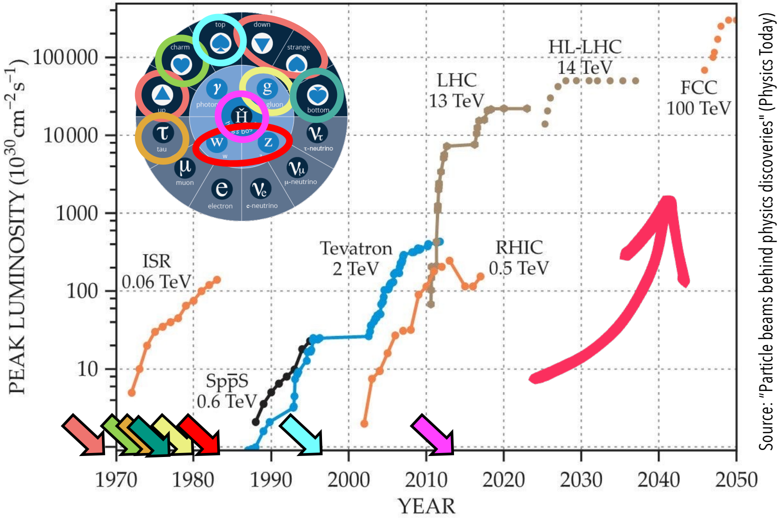


Particle accelerators …

…make fundamental discoveries in particle physics

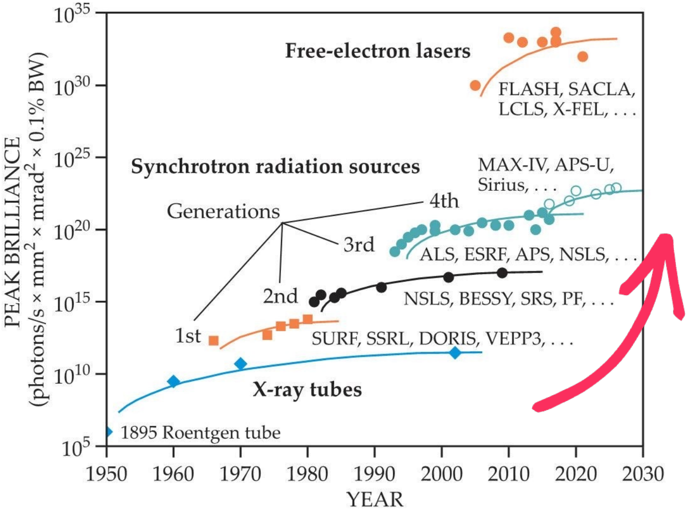
…are major tools for basic and applied research, industry & medicine worldwide

Ability to generate new particles via high-energy collisions



Source: "Particle beams behind physics discoveries" (Physics Today)

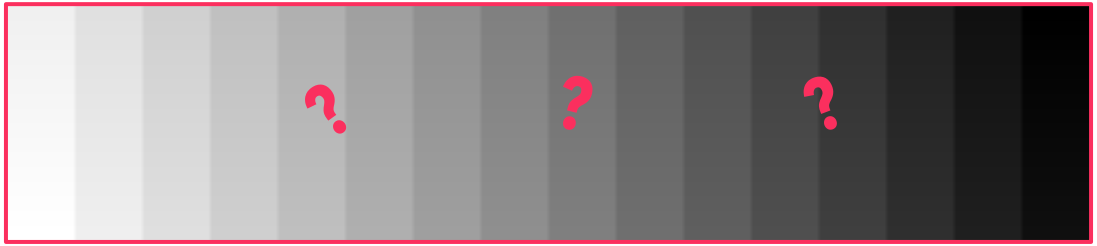
Ability to probe atomic structures



Technological innovation is needed to keep up with the challenging goals!

When to apply machine learning?

Classical control theory



Machine Learning

there are some clear cases

Optimization and control tasks in accelerators

Both perform equally

Cost of implementation and maintenance should then be considered

Future accelerators trends and challenges

and this is not considering user's needs!

Denser beams for higher luminosity & brilliance	Larger circular colliders for higher energies	Compact plasma accelerators with higher gradients
• complex beam dynamics & instabilities • complex design & operation	• orders of magnitude more signals • machine protection limits	• very tight tolerances • very high-quality beams required

What can machine learning do for us?

Very fast predictions by evaluating an already trained model

Classification task

Detect outliers and anomalies in accelerator data

- Fault detection
- Predictive maintenance
- Data cleaning

Optimization task

Achieve desired beam properties or states by tuning machine parameters

- Bayesian algorithms
- Optimizers

Prediction task

Predict the beam properties based on current accelerator parameters

- Surrogate models
- Virtual diagnostics

Control task

Control the state of the beam in real time in a dynamically changing environment

- Reinforcement learning

Check out the references we provide here!

https://github.com/ansantam/2022-MT-ARD-ST3-ML-workshop/blob/main/references/references.pdf

Motivation

Coherent Synchrotron Radiation (CSR)

Interesting for users

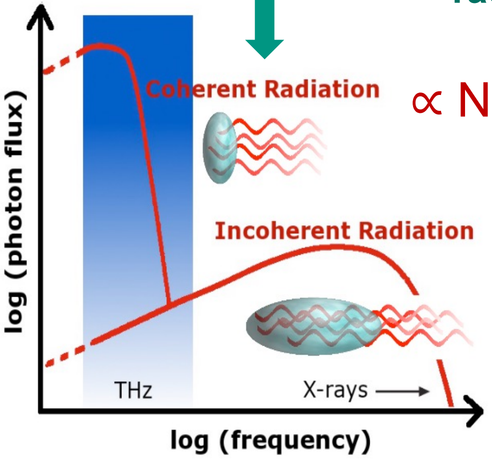
Achieved in short bunch operation mode (low-α optics)

Short bunch

Increased radiation power

Microstructures appear

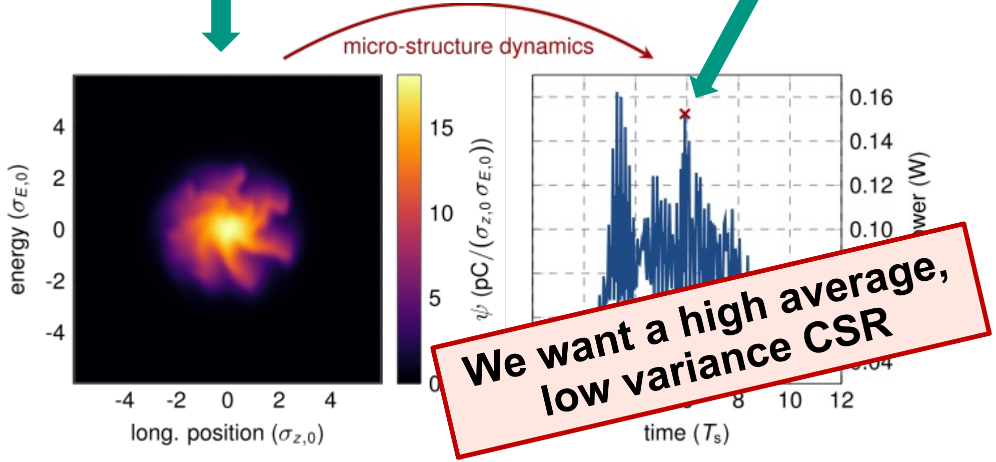
CSR power fluctuation



$\propto N^2$

$\propto N$

Image courtesy of A.-S. Müller



We want a high average, low variance CSR

Simulation code: Parallelized VFP solver ***Inovesa***

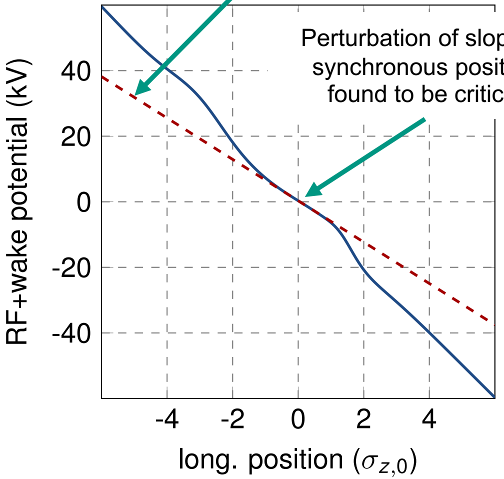
[T. Boltz et al, MOPGW017, IPAC'19](#)

Influencing the micro-bunching instability

CSR self interaction

Continuously changing

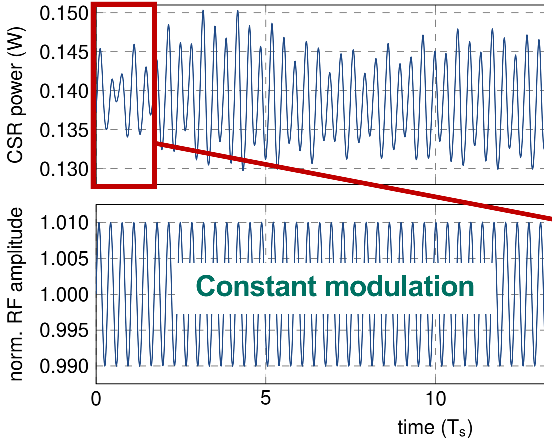
$$V_{eff}(q) = V_{RF}(q) + V_{CSR}(q)$$



Compensate the effect of the CSR perturbation by **modulating the RF voltage (amplitude)**

$$V_{RF} = \hat{V}(t)\sin(2\pi f_{RF} t)$$

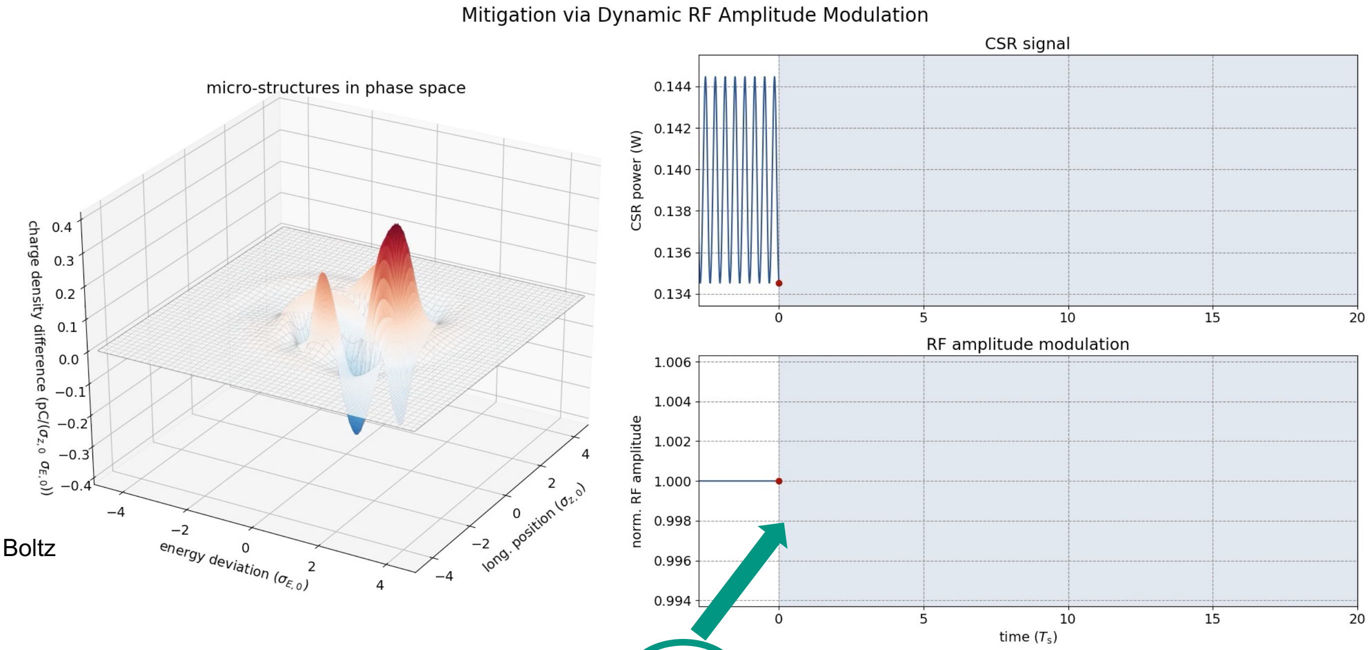
$$\hat{V}(t) = \hat{V}_0 + A_{mod}\ \sin(2\pi f_{mod} + \varphi_{mod})$$



Initial damping, but quickly out of sync…we need **dynamic control**!

Images courtesy of T. Boltz

RF voltage modulation with manual control



Courtesy of T. Boltz

$$V_{RF} = \hat{V}(t)\sin(2\pi f_{RF} t)$$

$$\hat{V}(t) = \hat{V}_0 + A_{mod}\ \sin(2\pi f_{mod} + \varphi_{mod})$$

High average, low variance CSR. So far so good…now to Reinforcement Learning!

Applying reinforcement learning

Action

$$\hat{V}(t) = \hat{V}_0 + A_{mod}\ sin(2\pi f_{mod} + \varphi_{mod})$$

Reward

R = $\mu_{CSR} - w\ \sigma_{CSR}$ where w is a weight

Could we improve the reward definition?

Observable (state definition)

Charge distribution

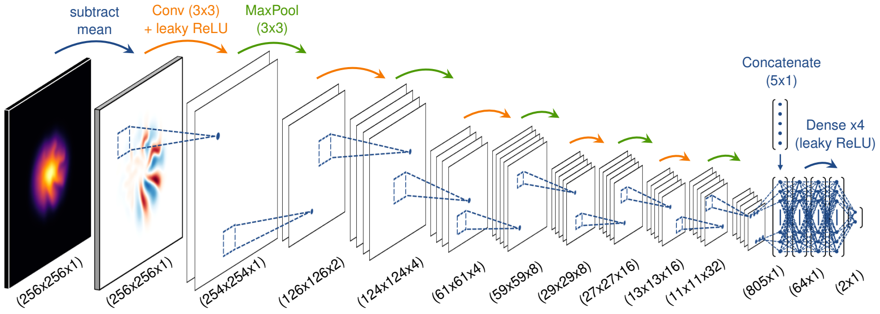
Input: (256x256) matrix + (5x1) feature vector

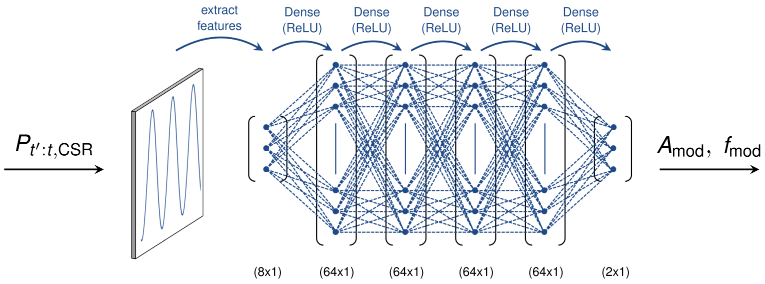
Observable (state definition)

CSR signal

Input: (8x1) feature vector

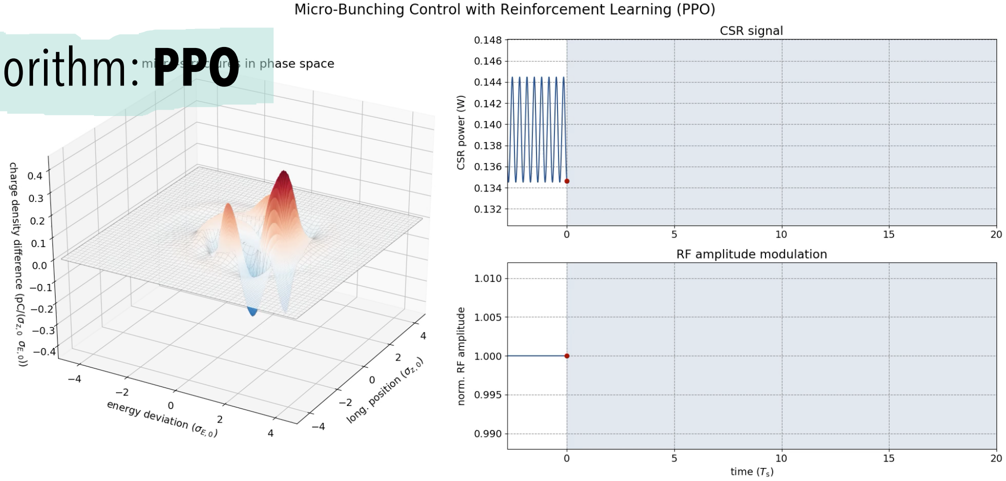
Easier to measure & process



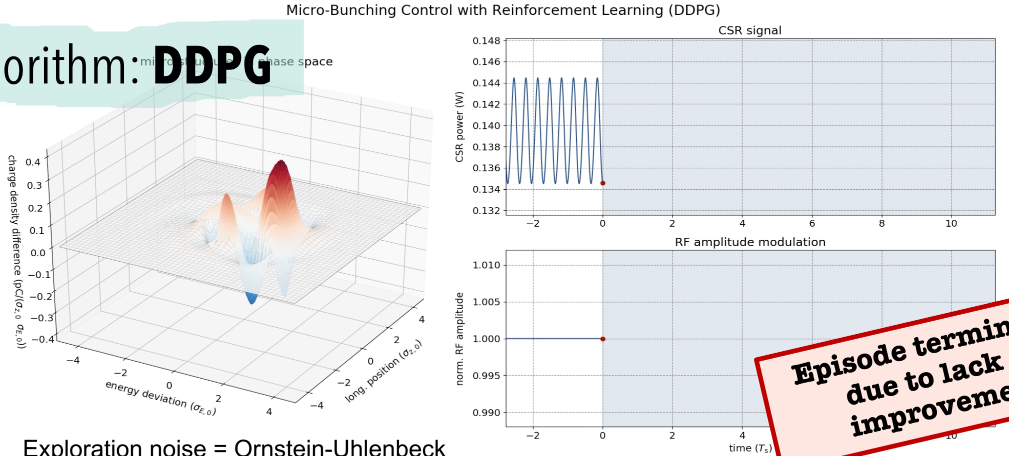


Images courtesy of T. Boltz

Algorithm: **PPO**



Algorithm: **DDPG**



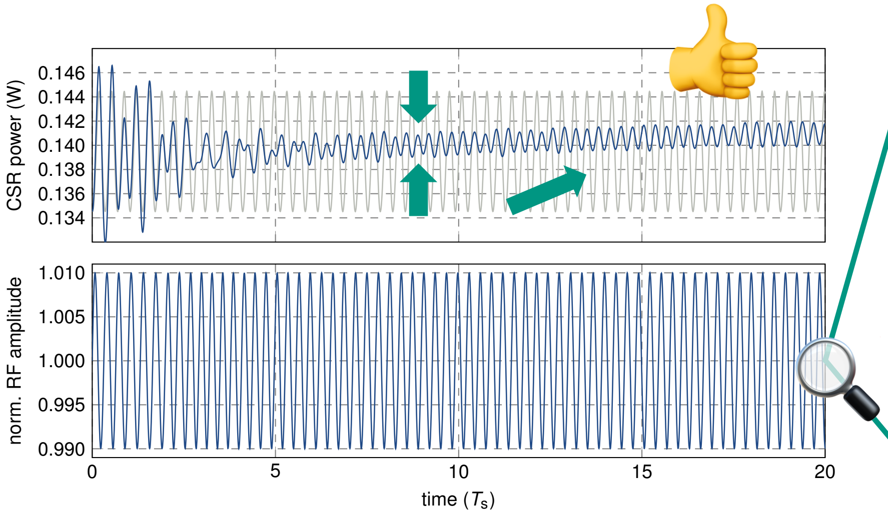
Episode terminated due to lack of improvement

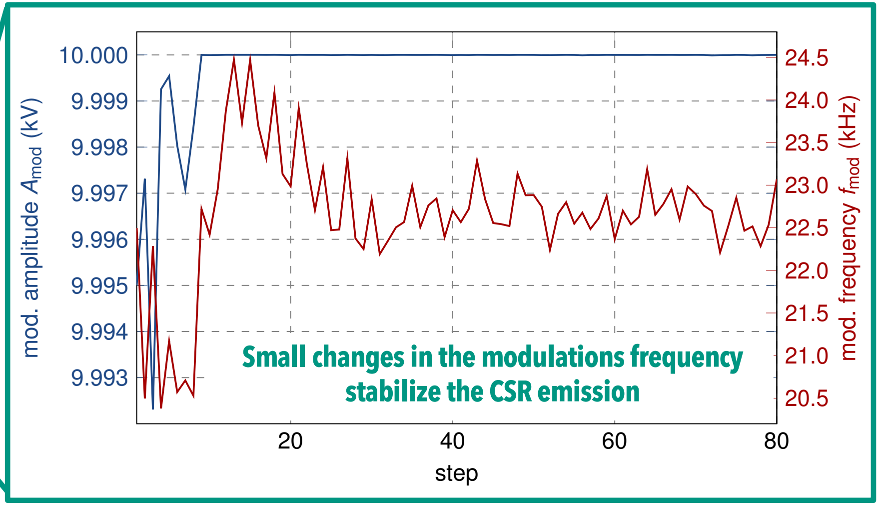
Exploration noise = Ornstein-Uhlenbeck

Tests in simulation with CSR signal as state definition

Courtesy of T. Boltz

Evolution of the actions with time (PPO)





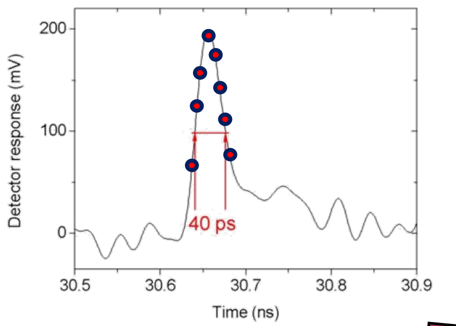
Other strategies are possible, such as varying the amplitude more

Images courtesy of T. Boltz

1 step = 0.25 synchrotron periods (chosen small enough for the agent to be able to react to the changing micro-structure dynamics)

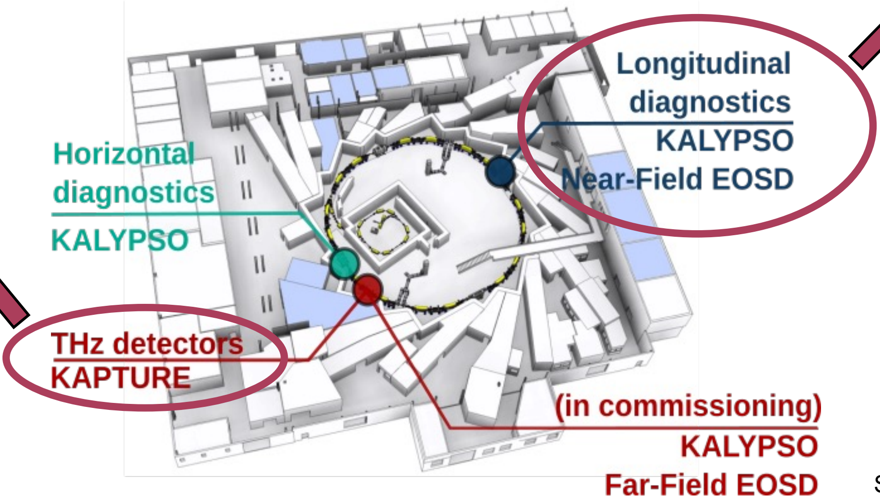
Real-time, high-repetition data acquisition

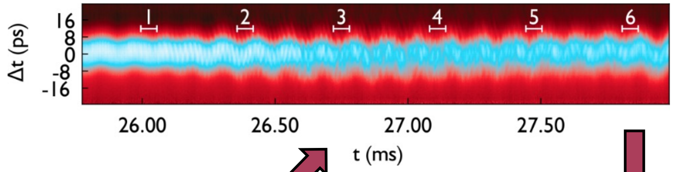
State-of-the-art detectors

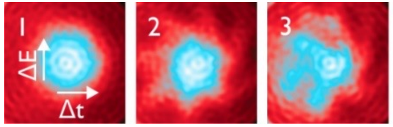


Meghana M Patil, 12th International Particle Accelerator Conference (IPAC'21)

KARA







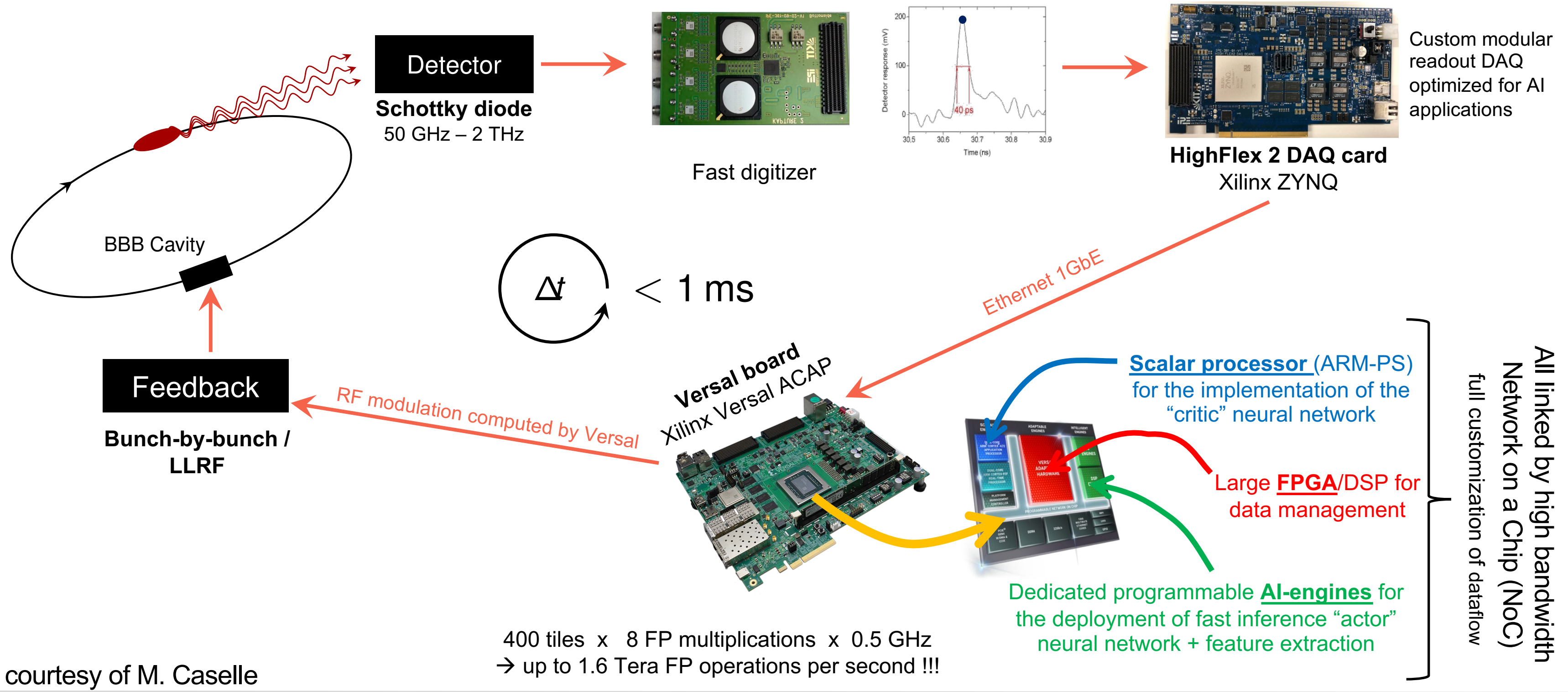
Phase space density reconstruction

Stefan Funkner; DOI: 10.1103/PhysRevAccelBeams.22.022801

In practice: we need hardware!

Fast feedback for real-time optimization

"Accelerated deep reinforcement learning for fast feedback of beam dynamics at KARA," W. Wang et al, IEEE TNS, vol. 68, 2021 (doi: 10.1109/TNS.2021.3084515)



Images courtesy of M. Caselle

Thank you for your attention!

What questions do you have for me?

- Sutton & Barto book
- https://arxiv.org/pdf/cs/9605103.pdf
- Reinforcement learning lectures by David Silver
- https://spinningup.openai.com/en/latest/
- Coursera RL specialization

Let's connect! andrea.santamaria@kit.edu / @ansantam

All icons from this talk from TheNounProject

Reinforcement learning

python

Frameworks

OpenAI Baselines

Dopamine

rllib

Chainer RL

DeepMind TRFL

Tensorforce

Stable baselines

TF-Agents

Environments

OpenAI Gym

Google DeepMind

DeepMind Control Suite

MuJoCo

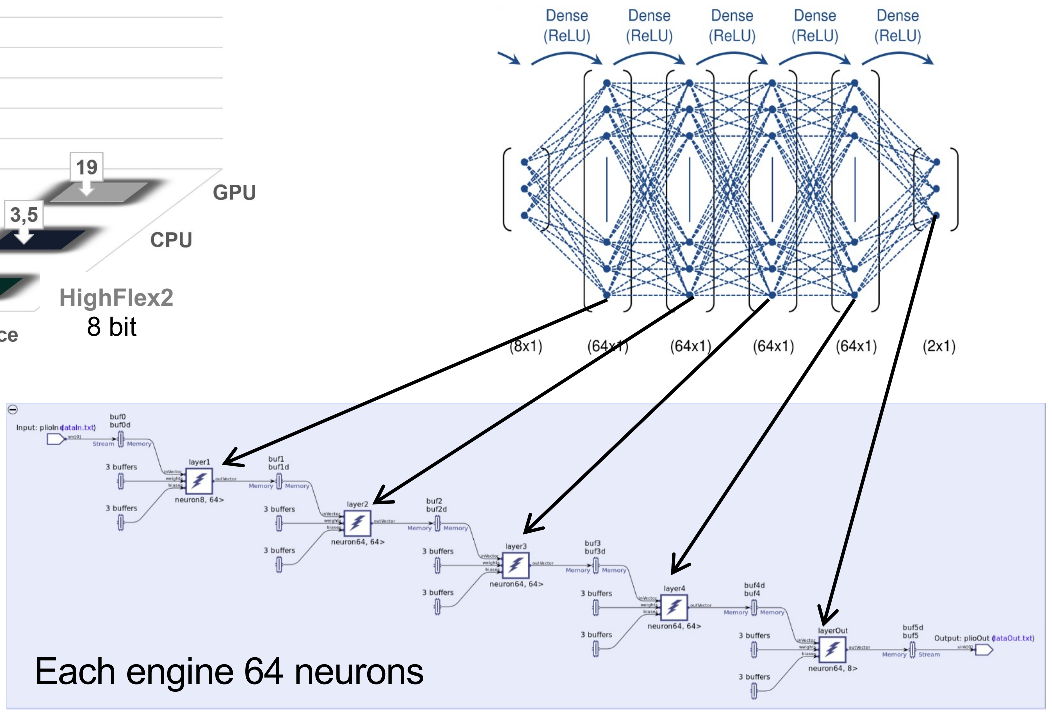
Advanced physics simulation

https://neptune.ai/blog/the-best-tools-for-reinforcement-learning-in-python

How fast can neural networks run?

Latency measured with Versal: 4.5 μs

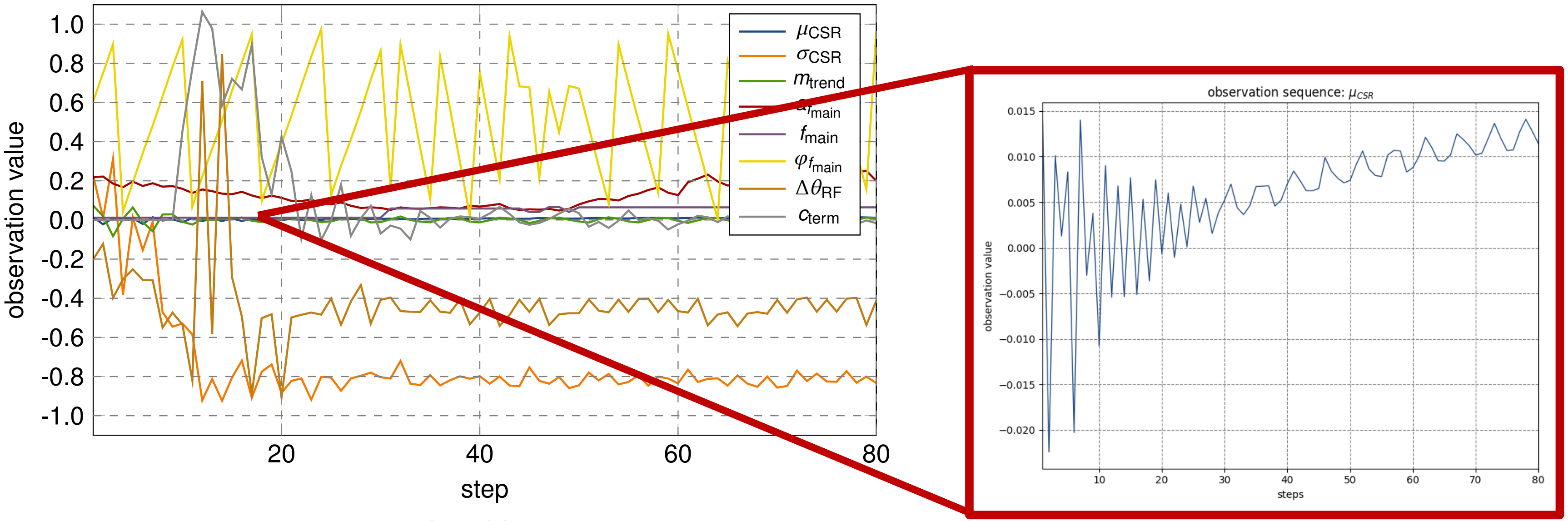
- 32 bit floating point computation
- 4 times better than previous Ultrascale+ family FPGA (on 8 bit integers)



Images courtesy of E. Bründermann, M. Caselle, L. Scomparin

W. Wang, M. Caselle, et al IEEE TNS, https://doi.org/10.1109/TNS.2021.3084515 (2021)

Observation vector based on the CSR signal



- μ_{CSR} is the normalized mean of the CSR power signal in the last time period.
- σ_{CSR} is the normalized standard deviation of the CSR power signal in the last time period.
- m_{trend} is a slow trend of the CSR power signal
- $a_{f_{main}}$ is the amplitude of the main frequency in the Fourier transformed CSR signal.
- f_{main} is the main frequency in the Fourier transformed CSR signal.
- $\varphi_{f_{main}}$ is the phase of the main frequency in the Fourier transformed CSR signal.
- $\Delta\theta_{RF}$ is the relative phase between the CSR signal and the applied RF signal (amplitude modulation).
- c_{term} models the termination condition (difference between the last reward and the one 10 steps prior).