Machine Learning for Physics

Unsupervised Learning and Reinforcement Learning

Lisbon School on Machine Learning for Physics 2025, LIP Lisboa, Portugal

Pietro Vischia
pietro.vischia@cern.ch
@pietrovischia

If you are reading this as a web page: have fun! If you are reading this as a PDF: please visit

https://www.hep.uniovi.es/vischia/persistent/2025-03-12_LisbonMLSchoolPhysics_Unsupervised.html

to get the version with working animations

Learn in different ways (today: unsupervised and reinforcement I.)

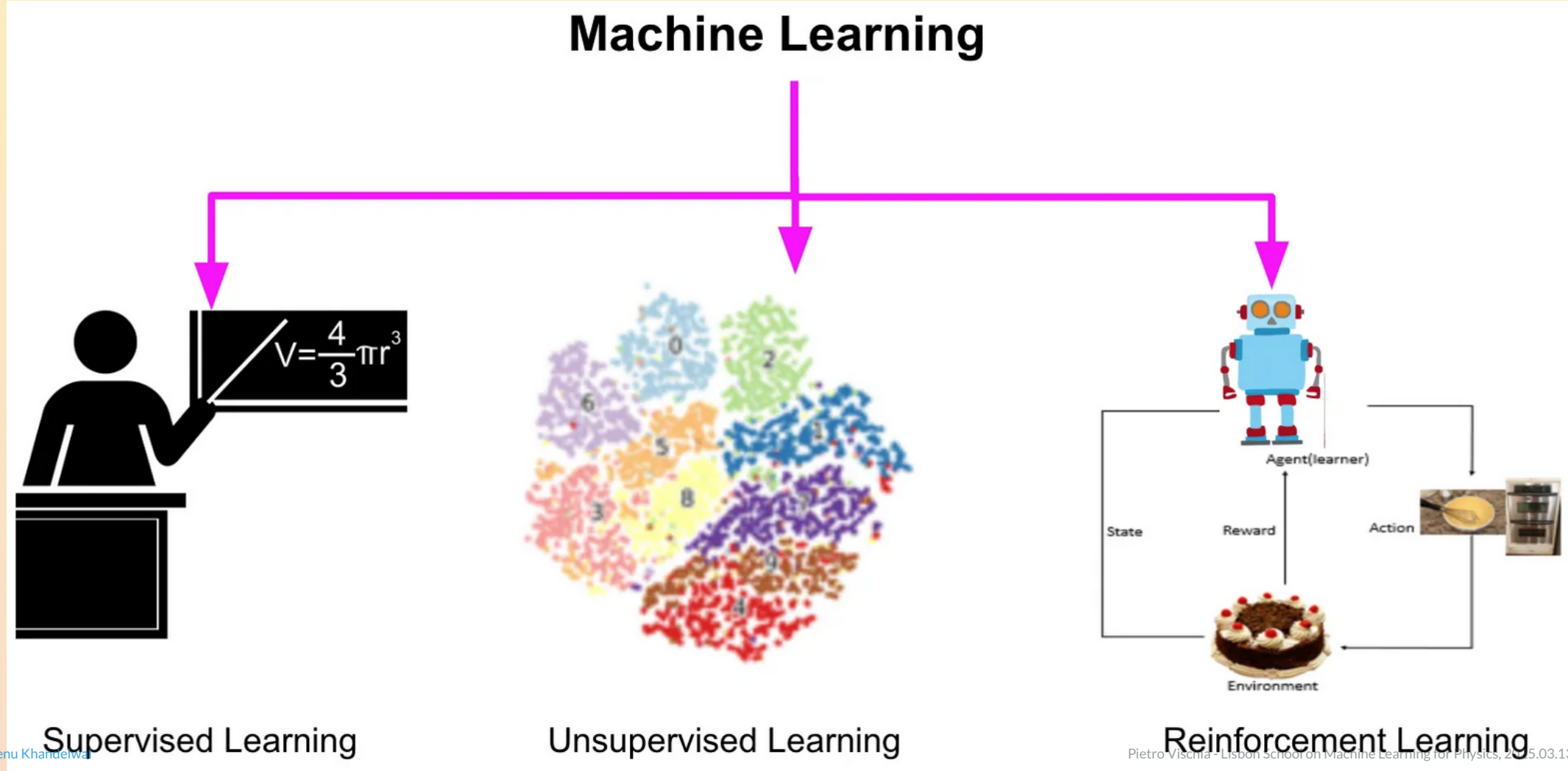


Image by Renu Khandelwal

Unsupervised learning

Unlabelled data

- We assume we only have input data X without labeled responses y.
  - Expensive, impractical, or impossible to obtain
- The goal is to discover hidden structures or patterns in the data
- Applications are numerous
  - Dimensionality reduction
  - Segmentation and/or clustering
  - Data compression
  - Anomaly detection
  - ...

Main Types of Unsupervised Learning

- Clustering
  - Group similar instances together.
  - Examples: K-Means, Hierarchical Clustering, Gaussian Mixture Models.
- Dimensionality Reduction
  - Compress data, reducing the number of features.
  - Examples: PCA, t-SNE, UMAP.
- Density Estimation
  - Estimate the distribution of data (e.g., Gaussian Mixture Models).
- Anomaly/Novelty Detection
  - Identify unusual patterns that do not conform to expected behavior.
  - Examples: Isolation Forest, One-class SVM.
- Feature Learning
  - Automatically learn representations (e.g., Autoencoders).

Clustering

- Regroup data points so that points in the same cluster are "closer" (more similar) to each other than to those in other clusters.
  - K-Means
  - Hierarchical Clustering
  - Gaussian Mixture Models
  - DBSCAN

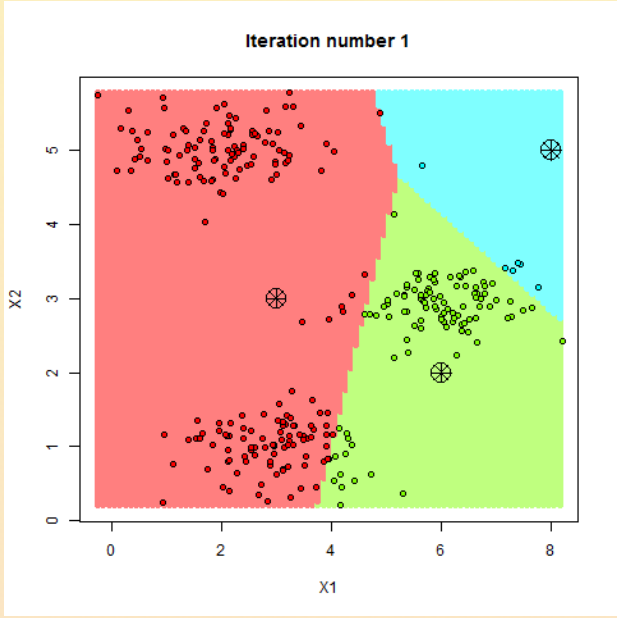
Clustering: K-Means

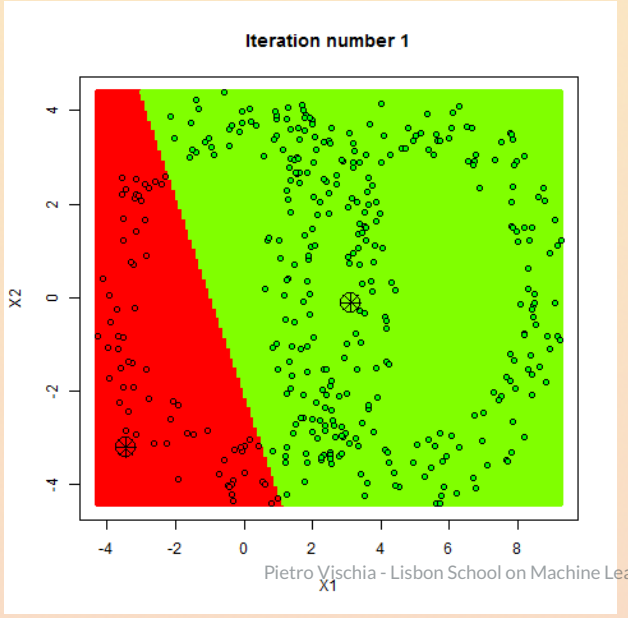
1. Choose the number of clusters, k
2. Initialize the k cluster centroids $\{\mu_i\}_{i=1}^{k}$
3. Assign each data point to the nearest centroid (typically Euclidean norm)
4. Recalculate centroids as the mean of assigned points
5. Repeat steps 3 and 4 until convergence

Objective function:

$$\min_{C_1,\ldots,C_k} \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

- Pros: Simple, fast, widely used
- Cons: Assumes spherical clusters, k choice is arbitrary, sensitive to outliers

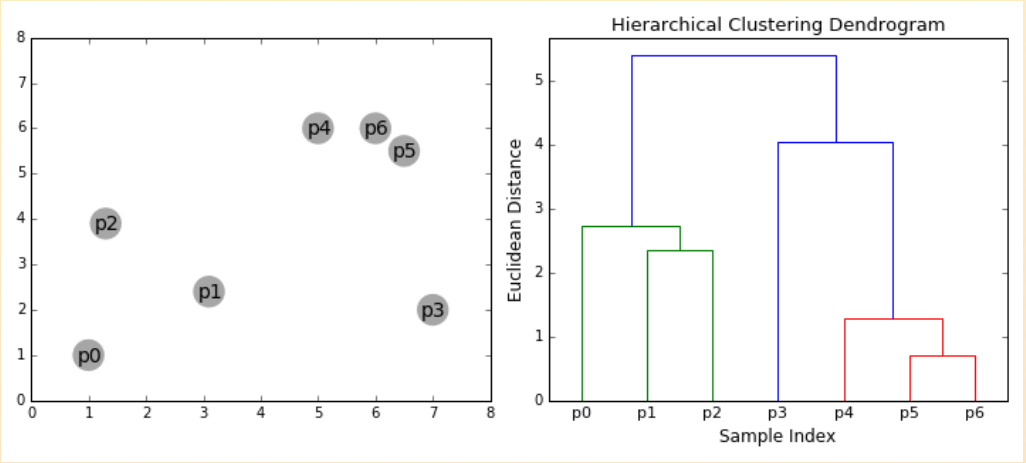




Animations by Sandipan

Hierarchical Clustering

- Bottom-up: the agglomerative approach
  1. Start with each data point in its own cluster
  2. Iteratively merge the two "closest" clusters until one cluster remains
  3. Linkage criterion: measure of dissimilarity between sets of observations as a function of the pairwise distance between data points



- Top-down: the divisive approach
  1. Start with all data in one cluster
  2. Recursively split clusters (e.g. choose object with max average dissimilarity, then attach to it all objects that are more similar to it than to the remainder objects)
- Dendrogram:
  - A tree-like structure showing how clusters are merged or split.
  - You can choose the cut of the dendrogram to get a desired number of clusters.

Animation by dashee87

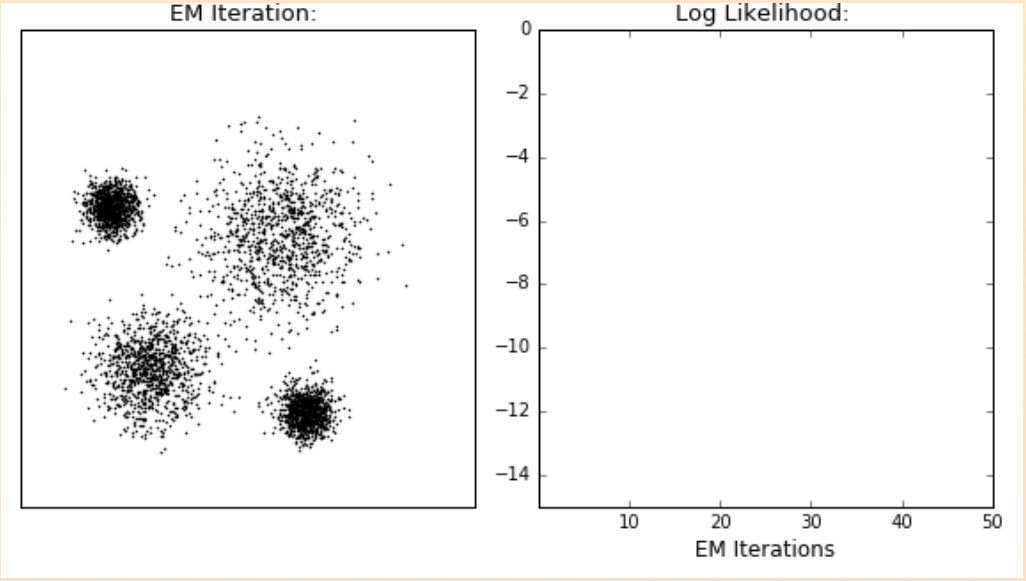
Clustering with Gaussian Mixture Models

- Probabilistic clustering approach modeling data as a mixture of Gaussians.
- Each cluster is modeled by a Gaussian distribution:

$$p(x) = \sum_{i=1}^{k} \pi_i \mathcal{N}(x \mid \mu_i, \Sigma_i)$$

where π_i are mixing coefficients.

- Expectation-Maximization (EM) algorithm iteratively refines:
  1. E-step: Estimate posterior probabilities of each point belonging to each cluster.
  2. M-step: Update parameters π_i, μ_i, Σ_i to maximize likelihood.

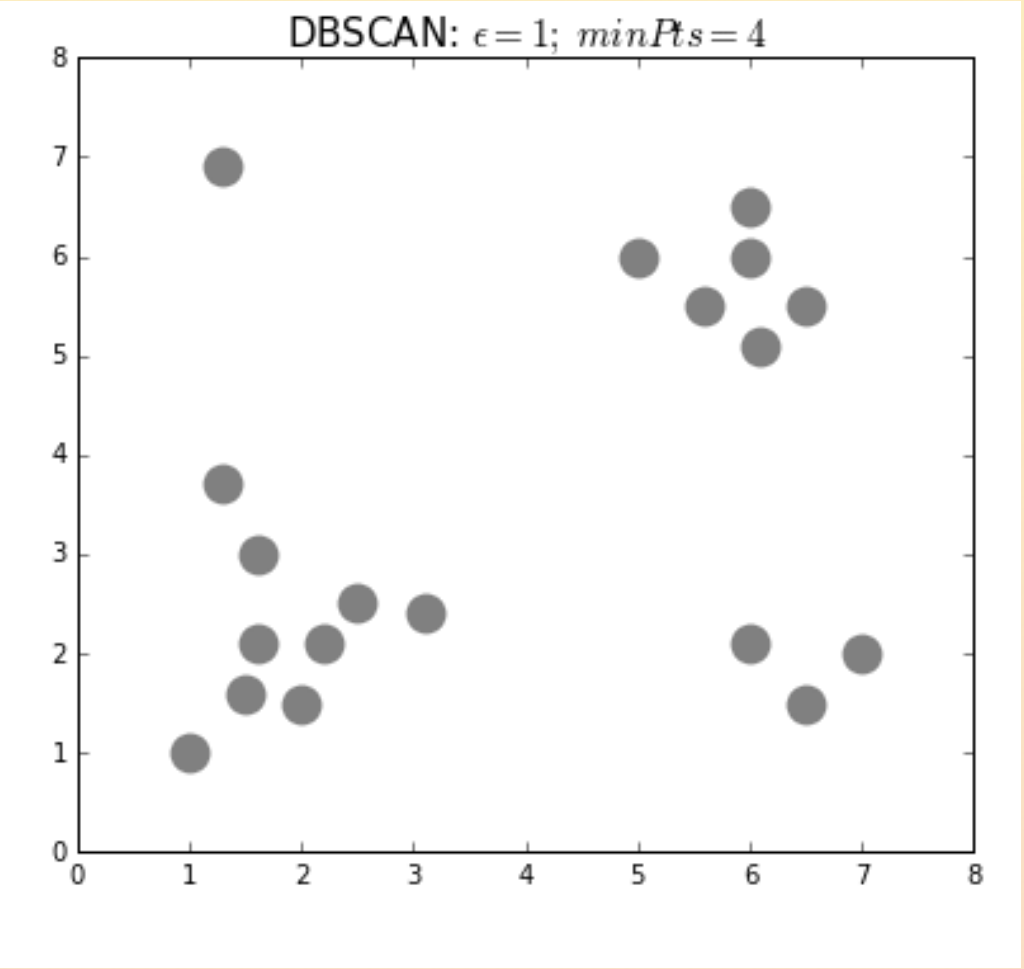


Animation by dashee87

- Pros: Flexible cluster shapes (covariance matrices).

Clustering with DBSCAN

- Clusters are defined as dense zones
  - Observations with no close neighbours treated as noise
- Define minimum number of elements in cluster and size of neighbourhood
- Iteratively investigate neighbourhood of all points belonging to cluster
  - Convergence when all points either in a cluster or labelled as noise
- Pros: no need to specify number of clusters, treat outliers as noise, any shape
- Cons: points at the border may be assigned to different clusters in each execution, curse of dimensionality



Animation by dashee87

Dimensionality Reduction

- Curse of dimensionality.
  - How many samples do we need to estimate f^*, depending on assumptions on its regularity?

Dimensionality Reduction

- Curse of dimensionality.
  - How many samples do we need to estimate f^*, depending on assumptions on its regularity?
  - f^* constant $\rightarrow$ need only 1 sample
  - f^* linear $\rightarrow$ need d samples

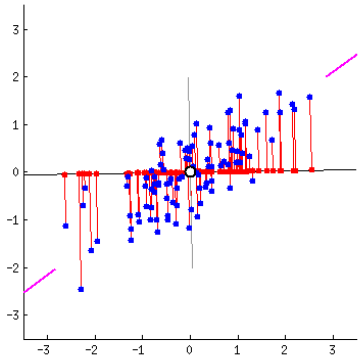
Dimensionality Reduction

- Curse of dimensionality.
  - How many samples do we need to estimate f^*, depending on assumptions on its regularity?
  - f^* constant $\rightarrow$ need only 1 sample
  - f^* linear $\rightarrow$ need d samples
  - If f^* is Lipschitz, it can be demonstrated that $n \sim \epsilon^{-d}$
- Computational efficiency.
- Data visualization.
- Linear Methods
  - Principal Component Analysis (PCA).
- Non-Linear Methods
  - t-SNE, UMAP

Dimensionality reduction: PCA

- Principal Component Analysis: find orthornormal basis where dimensions are linearly uncorrelated
  - Found iteratively finding the direction (linear combination of features) explaining the most variance

- Principal components are the eigenvectors of the data covariance matrix
  - Can be found by Singular Value Decomposition (SVD)
- Somehow analogous to finding axes of ellipsoid
  - Features with different units $\longrightarrow$ arbitrariety (scale them first)
- Can retain a few dimensions: dimensionality reduction
  - Drop directions least explaining the variance
  - Retain the dimensions that explain most of the variance



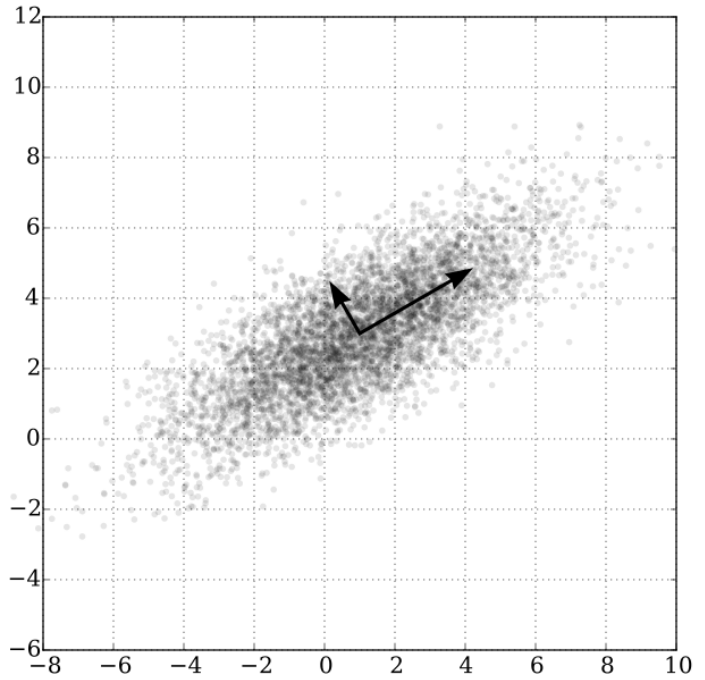


Image by Nicoguaro on Wikipedia and by builtin

Dimensionality reduction: PCA

- Iteratively, find the first one, then find the next one conditioned on being orthogonal to the previous one
  - Or simply do Single Value Decomposition (SVD)
- SVD: 2D case (multivariate is just iteratively the same)
  - find the best linear fit: this shows the direction of maximum variance in the dataset
  - the eigenvector is the direction of that line
  - the eigenvalue expresses how much the data set is spread out on that line

- Steps for PCA
  - Standardize each variable
  - Compute covariance matrix
  - Compute eigenvectors of covariance matrix
  - Order them by eigenvalue
  - Select components you want to keep
  - Transform data in the new coordinate system

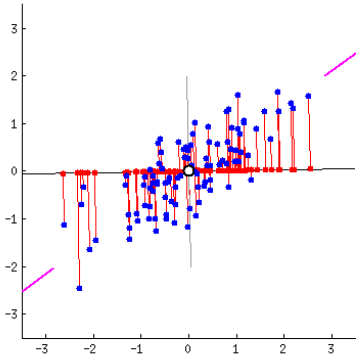


Image by builtin

t-SNE & UMAP

- t-SNE (t-Distributed Stochastic Neighbor Embedding)
  - Minimizes the Kullback-Leibler divergence between the joint probabilities of the original data and of their low-dimensional embedding
  - Focusses on preserving local structure
  - Good for data visualization but not always for actual tasks
- UMAP (Uniform Manifold Approximation and Projection)
  - Builds a weighted graph of the data and optimizes its layout in lower dimensions
  - Preservese both local and global strucutre
  - Often faster, strongly adopted for visualization

- Recent work hints that the two algorithms can be "morphed" into each other via "repulsion/attraction" one-dimensional parameter

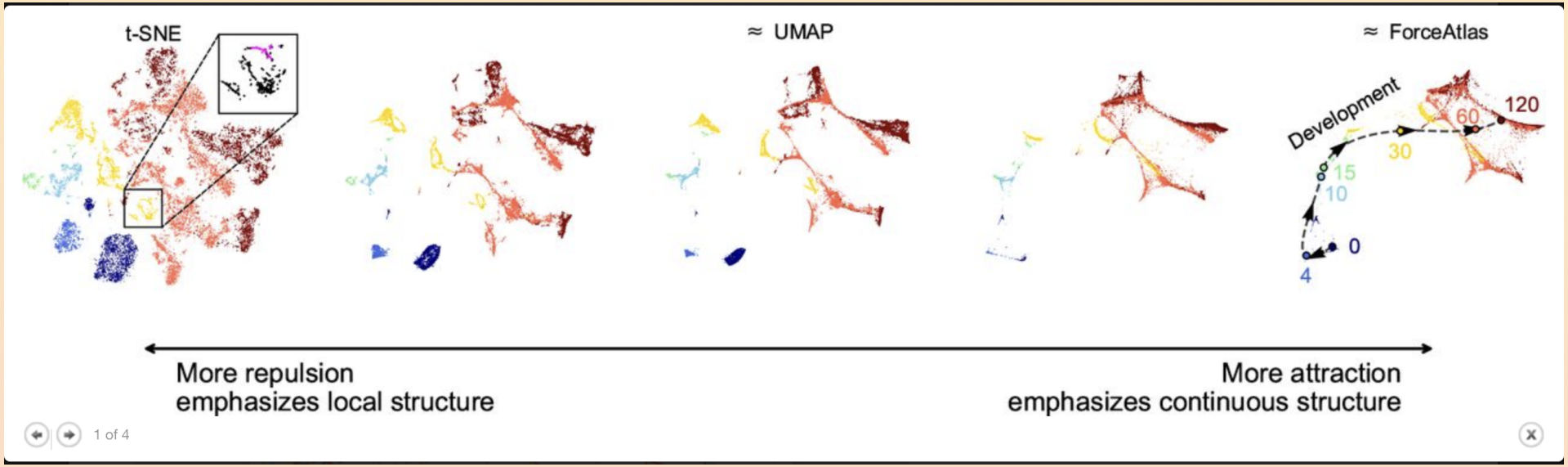


Image from Damrich et al, 2024: 10.1101/2024.04.26.590867

Autoencoders...

- Learn the data itself passing by a lower-dimensional intermediate representations
  - Capture data generation features into a lower-dimensional space
- Can use for anomaly detection
  - Spot objects that are different from those you have trained on (see e.g. anomaly detection in the CMS Muon chambers 1808.00911)
- Can sample from the latent space to obtain random samples (generative AI)
- Can denoise data, learn features, reduce dimensionality

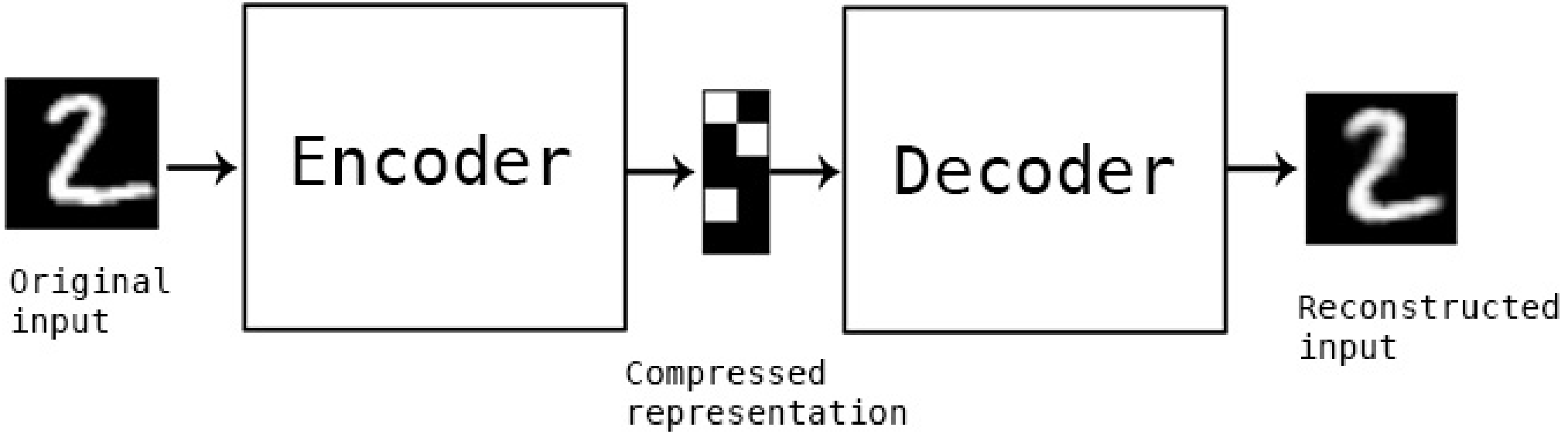


Figure from...

...and Variational autoencoders

- Learn a space of continous representations of the inputs

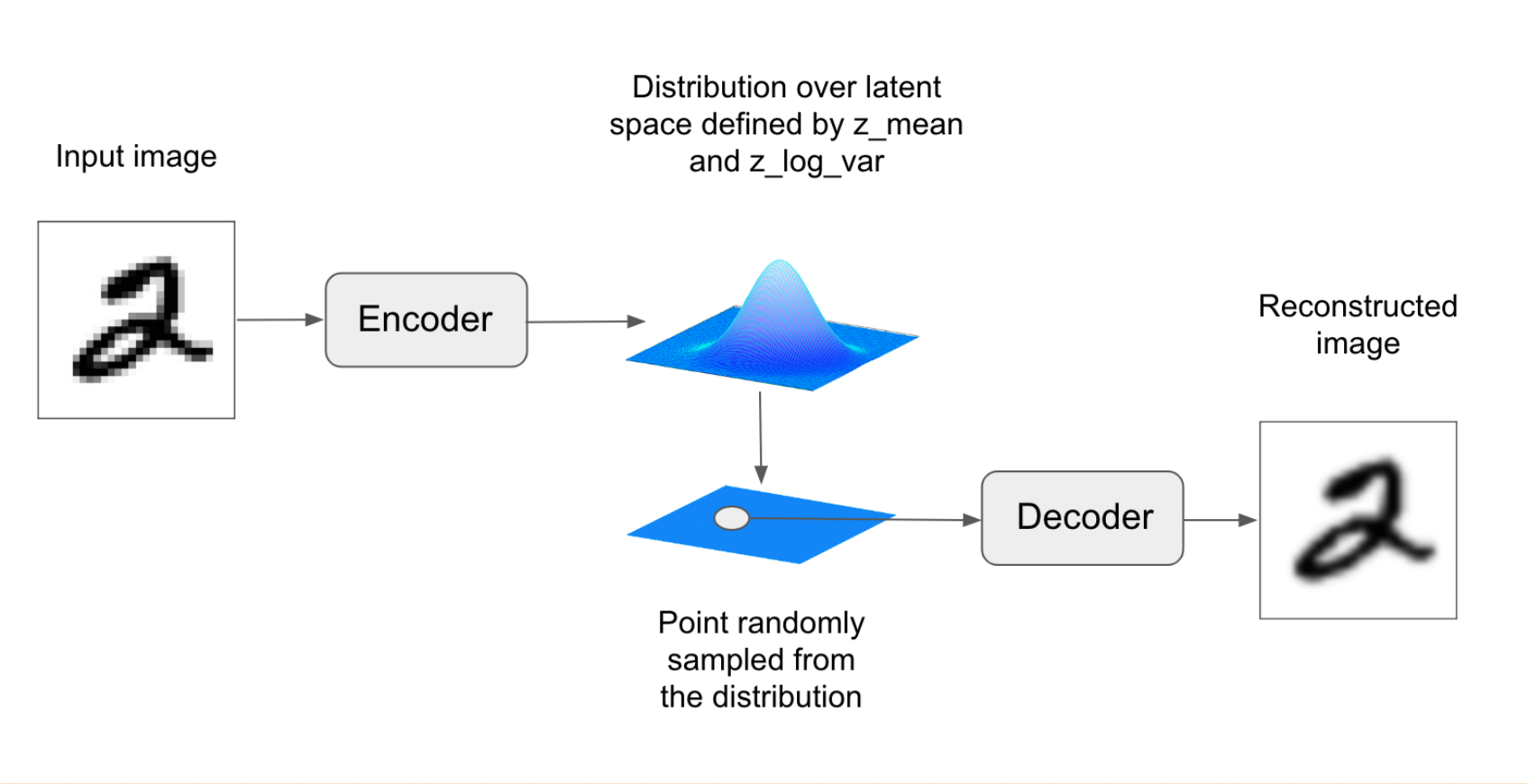
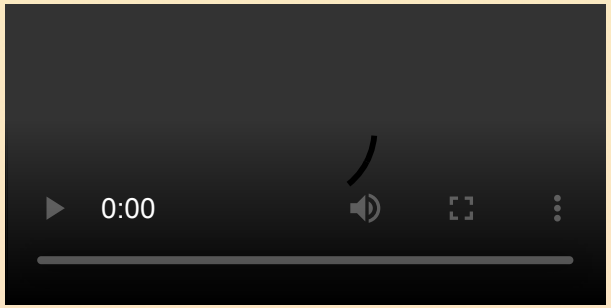


Image from Chollet and Allaire, Deep Learning With R

Reinforcement Learning

Reinforcement learning...

- An agent learns to make sequential decisions by interacting with an environment.
  - Maximize cumulative reward over time.
  - A process of trial and error + delayed reward



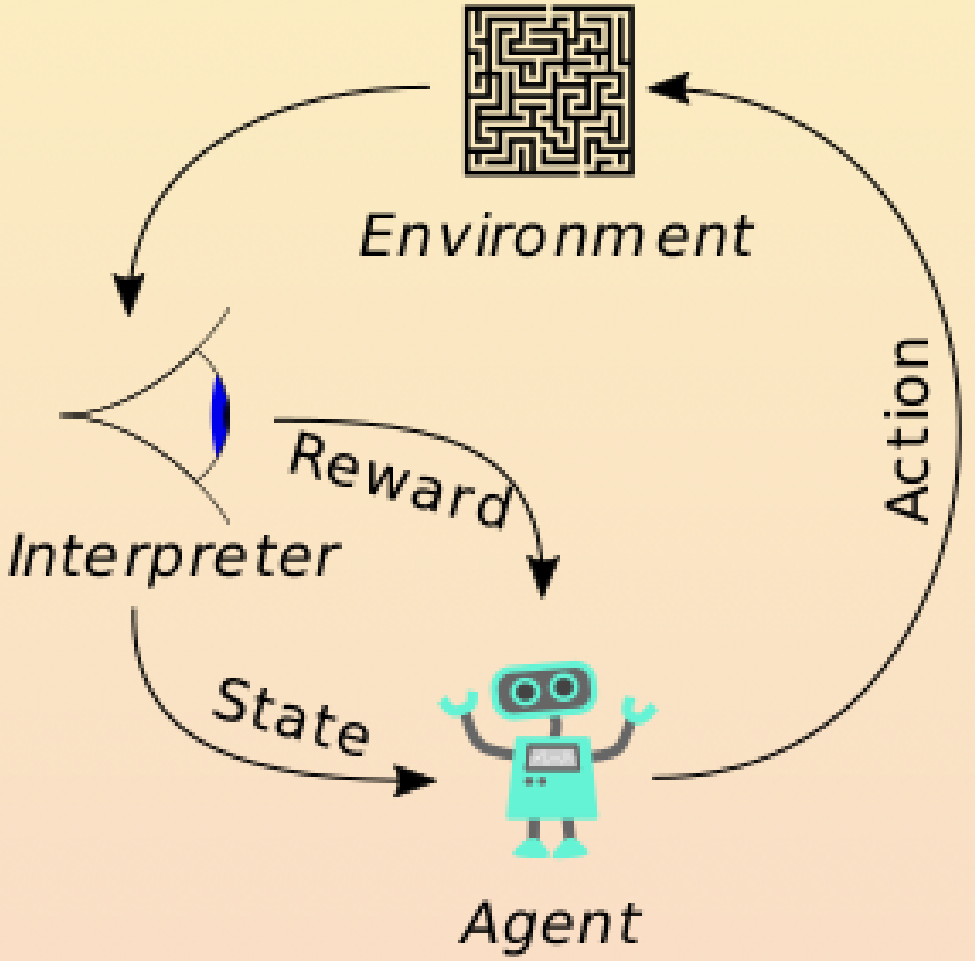


Image by Megajuice on Wikimedia, video from YouTube

...in Physics

- "Particle Physics Model Building with Reinforcement Learning" (2103.04759)
  - Reward models consistent with the observed quark properties

charges	$\mathcal{Q} = \begin{array}{c\|ccc\|ccc\|ccc\|cc} & Q_1 & Q_2 & Q_3 & u_1 & u_2 & u_3 & d_1 & d_2 & d_3 & H & \phi \\ \hline q & 6 & 4 & 3 & -2 & 2 & 4 & -3 & -1 & -1 & 1 & 1 \end{array}$
$\mathcal{O}(1)$ coeff.	$(a_{ij}) \simeq \begin{pmatrix} -1.975 & 1.284 & -1.219 \\ 1.875 & -1.802 & -0.639 \\ 0.592 & 1.772 & 0.982 \end{pmatrix}$ $(b_{ij}) \simeq \begin{pmatrix} -1.349 & 1.042 & 1.200 \\ 1.632 & 0.830 & -1.758 \\ -1.259 & -1.085 & 1.949 \end{pmatrix}$
VEV, Value	$v_1 \simeq 0.224\,, \qquad \mathcal{V}(\mathcal{Q}) \simeq -0.598$

charges	$\mathcal{Q} = \begin{array}{ccc\|ccc\|ccc\|cc} Q_1 & Q_2 & Q_3 & u_1 & u_2 & u_3 & d_1 & d_2 & d_3 & H & \phi \\ \hline 1 & 2 & 0 & -1 & -3 & 1 & -3 & -5 & -4 & 1 & 1 \end{array}$
$\mathcal{O}(1)$ coeff.	$(a_{ij}) \simeq \begin{pmatrix} -0.601 & 1.996 & 0.537 \\ -0.976 & -1.498 & -1.156 \\ 1.513 & 1.565 & 0.982 \end{pmatrix}$ $(b_{ij}) \simeq \begin{pmatrix} 0.740 & -1.581 & -1.664 \\ -1.199 & -1.383 & 0.542 \\ 0.968 & 0.679 & -1.153 \end{pmatrix}$
VEV, value	$v_1 \simeq 0.158\,, \qquad \mathcal{V}(\mathcal{Q}) \simeq -0.621$

Table from 2103.04759

Core Reinforcement Learning concepts

- The Agent is the learner/decision-maker
- The agend acts with the Environment, an external system
- The environment is in a certain State s at each time
- The agent can execute an Action a, which affects the environment
- The agent receives a feedback signa, the Reward r, indicating how good the action was

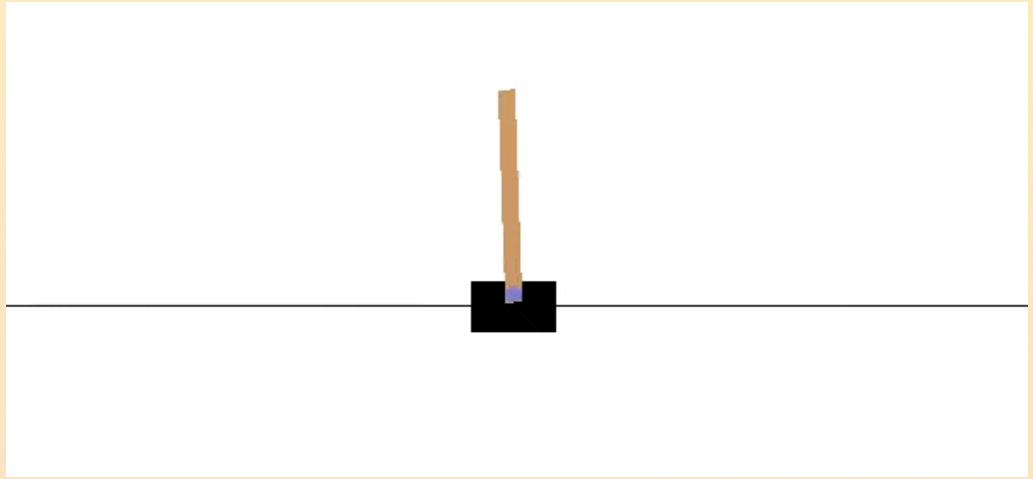
Markov Decision Process (MDP)

- Often build a finite-state machine and solve with MDPs
  - Often difficult in real environments (e.g. continuous variables instead of finite states)

- MDP defined by:

$$(\mathcal{S}, \mathcal{A}, P, R, \gamma)$$

  - $\mathcal{S}$: State space
  - $\mathcal{A}$: Action space
  - $P(s' \mid s, a)$: Transition probabilities
  - $R(s, a)$: Reward function
  - $\gamma \in [0, 1]$: Discount factor for future rewards



Animation from paperspace

Policies and Value

- Policy π: a mapping from state to action (deterministic) or state to action probabilities (stochastic).

$$\pi(a \mid s)$$

- Value function $V^\pi(s)$: Expected return (sum of discounted rewards) starting from state s, following policy π.

$$V^\pi(s) = \mathbb{E}_\pi\Big[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_0 = s\Big]$$

- Action-value function (Q-function) $Q^\pi(s, a)$: Expected return starting from state s, taking action a, then following π

$$Q^\pi(s, a) = \mathbb{E}_\pi\Big[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_0 = s, a_0 = a\Big]$$

Bellman Equations

- Bellman Expectation Equation for V^π

$$V^\pi(s) = \mathbb{E}_{a\sim\pi}\left[R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V^\pi(s')\right]$$

- Bellman Optimality Equation for V^*

$$V^*(s) = max_a\left(R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) V^*(s')\right)$$

- Similarly, for Q-functions

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s' \mid s,a) \max_{a'} Q^*(s',a')$$

The Bellman Equation is often used to solve stochastic optimal control problems.

Dynamic Programming and Value Iteration

- If the MDP model P and R are known and the state space is not too large
  - Value Iteration: Iteratively apply Bellman updates to converge to V^* (and therefore Q^*)
  - Policy Iteration: Alternate between policy evaluation and policy improvement steps
- If the environment is large or unknown, model-free methods are preferred

Model-Free Methods: Q-Learning & SARSA

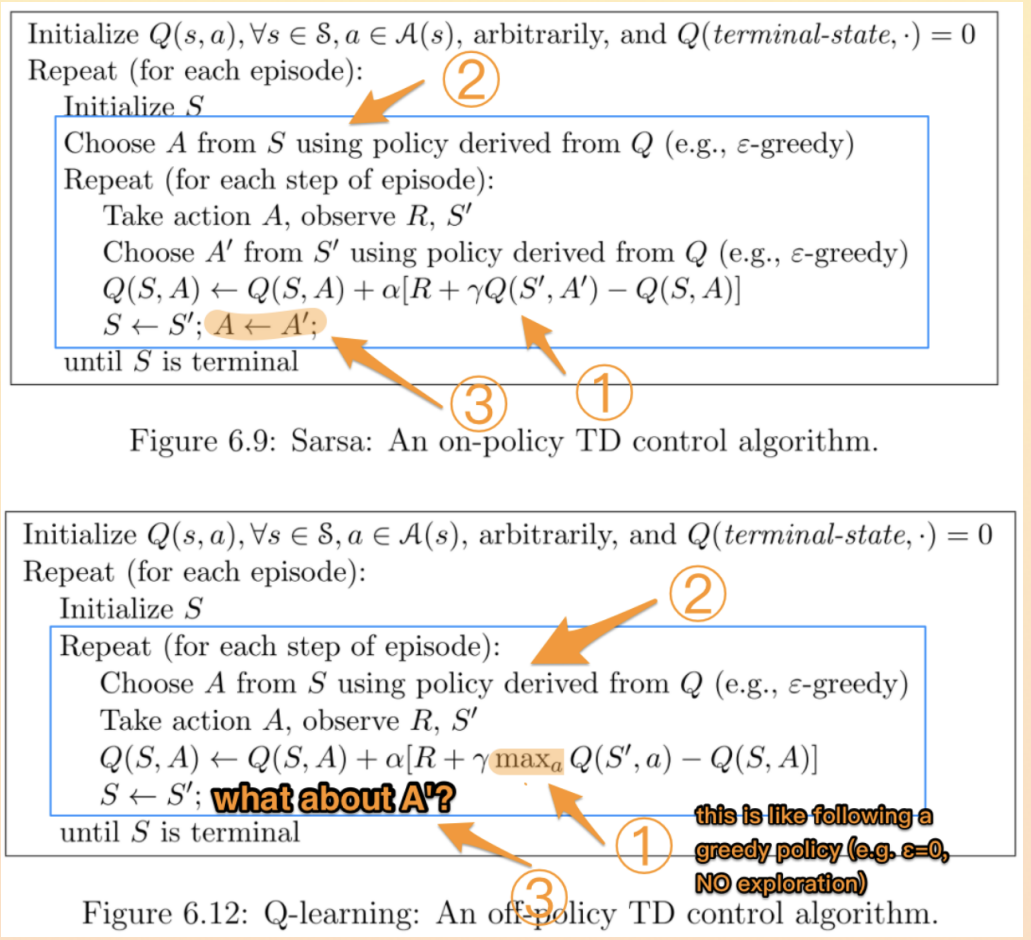
- Q-Learning:
  - Off-policy algorithm: Learns the optimal policy regardless of the agent's behavior policy.
  - Update rule (tabular):

$$Q(s,a) \leftarrow Q(s,a) + \alpha\Big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\Big]$$

- SARSA:
  - On-policy algorithm: Learns the value of the policy being carried out.
  - Update rule (tabular):

$$Q(s,a) \leftarrow Q(s,a) + \alpha\Big[r + \gamma Q(s',a') - Q(s,a)\Big]$$

where $a' \sim \pi(\cdot \mid s')$



Animation from paperspace

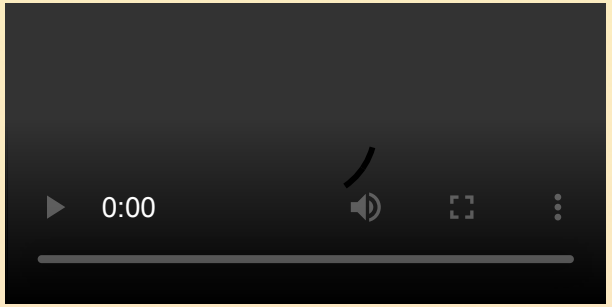
Other paradigms

- Temporal Difference (TD) Learning
  - Learn value functions from incomplete rollouts by bootstrapping from current estimates (i.e. update prediction before looking at the final outcome)
  - $TD(0)$: One-step update
  - $TD(\lambda)$: a larger proportion of credit from a reward can be given to more distant states and actions (multistep look-ahead)
  - Requires less computation than Monte Carlo methods and often converges faster
- Policy Gradient Methods
  - Instead of learning a value function and deriving a policy, directly learn the parameters (\theta) of a policy (\pi_\theta(a|s)).
  - Maximize expected return (\mathcal{J}(\theta)).
  - Gradient Ascent (REINFORCE, actor-critic methods)

$$\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{J}(\theta)$$

Deep Reinforcement Learning

- Combine Deep Neural Networks with Reinforcement Learning
- Deep Q-Network (DQN) Approximate $Q(s, a)$ with a neural net
  - Surrogate
  - Experience replay
  - Target networks
- Actor-Critic Architectures: Use separate networks for policy (actor) and value function (critic).

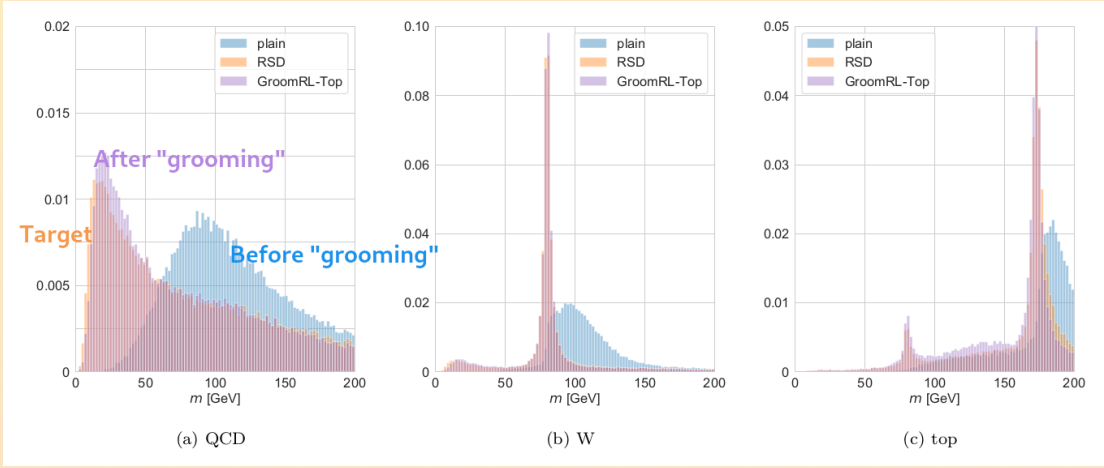


Video by DeepMind

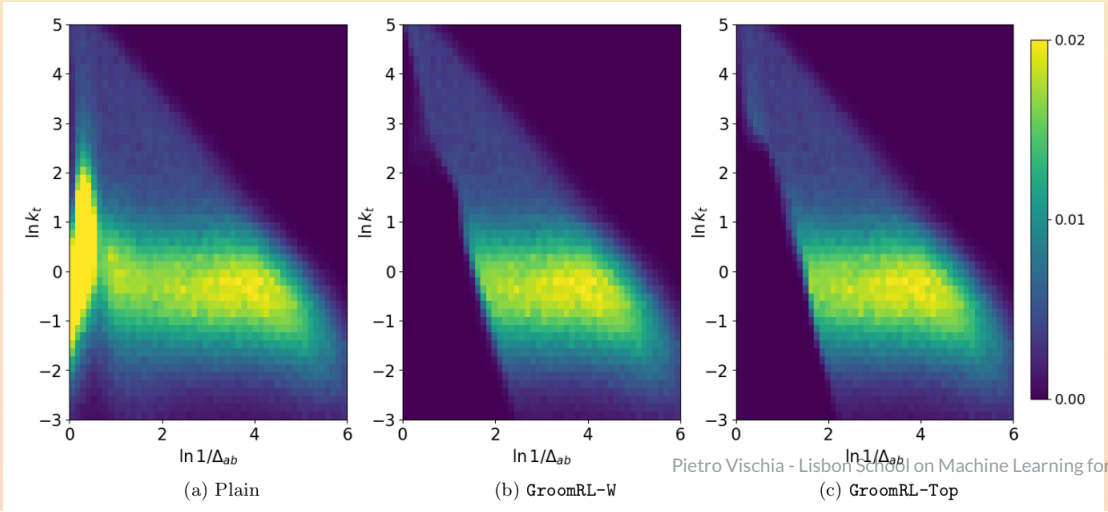
Deep Q Learning in particle physics

- Boosted objects decay to collimated jets reconstructed as a single jet
- Fat jet grooming: remove soft wide-angle radiation not associated with the underlying hard substructure

- Resulting Lund diagrams match with expectations from first principles



(a) QCD (b) W (c) top



(a) Plain (b) GroomRL-W (c) GroomRL-Top

Images from 1903.09644

Challenges in RL

- Sample Efficiency: Training can require large amounts of data
- Exploration vs. Exploitation: Balancing trying new actions vs. capitalizing on known rewards
- Partial Observability: Agents often don't see the entire environment
- Function Approximation: Instability in deep architectures
- Reward Shaping: Designing reward functions that lead to desired behavior

Now: exercise 4, and data challenge