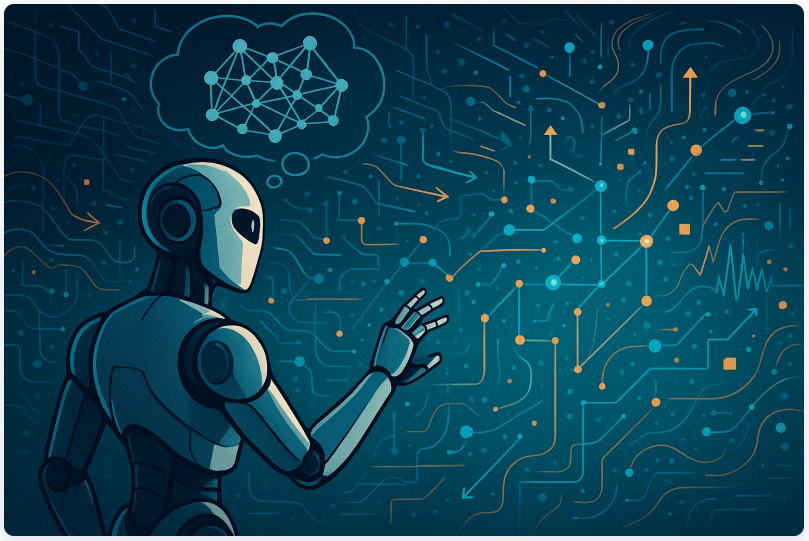
Reinforcement Learning Through a Probabilistic Lens

METIS 2025: Trends in Machine Learning

Salem Lahlou, MBZUAI, https://la7.lu

Section 1: Introduction & Motivation

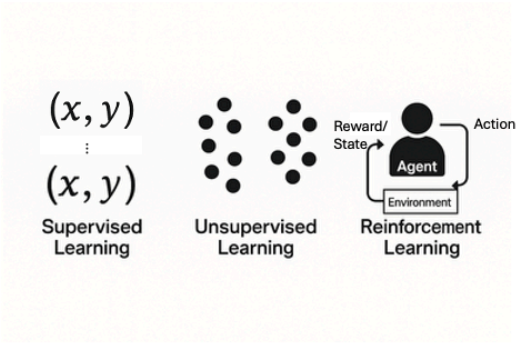
The Quest for Intelligent Agents & Planning



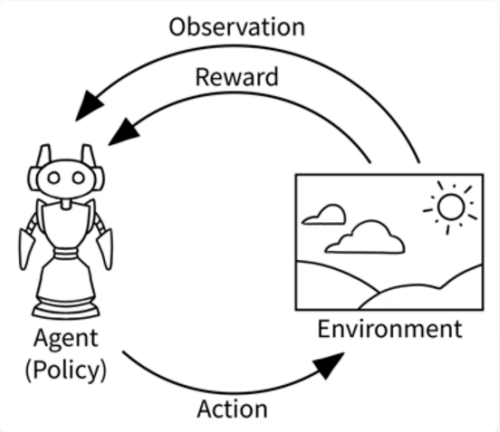
- The grand ambition: Creating machines that can perceive, reason, learn, **plan**, and act intelligently in complex, dynamic environments.
- How do we build systems that can make good sequences of decisions to achieve goals? This involves **planning** – figuring out a course of action.
- Machine Learning (ML) provides the tools for systems to learn from experience, often to improve their planning and decision-making capabilities.
- Reinforcement Learning (RL) is a powerful paradigm for learning optimal **plans** or **policies** through trial and error, guided by rewards, directly addressing this quest for intelligent, planning agents.

What is Machine Learning? Types of Learning Tasks

- **Machine Learning:** Algorithms that enable computer systems to learn from data and improve their performance on a specific task without being explicitly programmed for each case.
- **Major Types of Learning Tasks:**
  - **Supervised Learning:** Learning from labeled data ($\{(\mathbf{x}_i, y_i)\}$). Goal is to learn a mapping $f : \mathbf{X} \to Y$, where f aims to minimize some loss $L(f(x_i), y_i)$.
  - **Unsupervised Learning:** Learning from unlabeled data ($\{\mathbf{x}_i\}$). Goal is to find patterns, structure, or representations in the data.
    - *Examples: Clustering, dimensionality reduction (PCA), anomaly detection.*
  - **Reinforcement Learning:** Learning to make sequences of decisions by interacting with an environment to achieve a long-term goal.
    (More on this next!)



What is Reinforcement Learning?



Example: An Urban Delivery Robot deciding whether to move, charge, or deliver based on its location, battery, and package status to maximize successful deliveries.

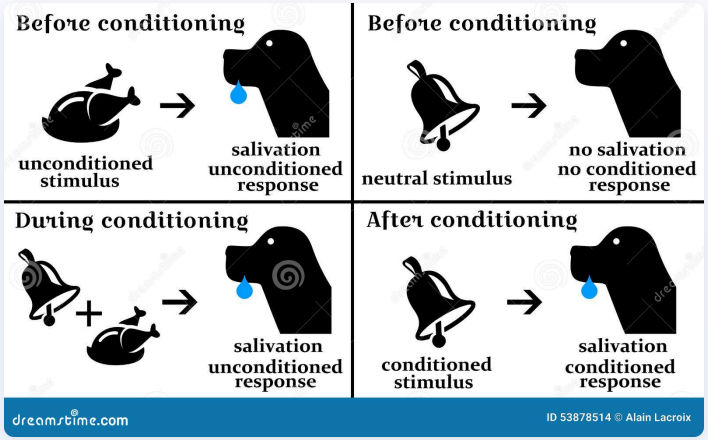
- An agent learns to make a sequence of decisions by interacting with an environment to achieve a long-term goal.
- **Core components:** Agent, Environment, State (S), Action (A), Reward (R).
- The agent observes the state, takes an action, receives a reward, and transitions to a new state.
- The goal is to learn a **policy** (a strategy for decision-making) that maximizes the cumulative reward:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_t \gamma^t r(s_t, a_t) \right]$$

RL's Roots: Pavlov & Learning by Association

Ivan Pavlov (1890s-1900s): Classical Conditioning

- **The Famous Experiment:** Pavlov, a Russian physiologist, was initially studying digestion in dogs. He noticed that dogs would begin to salivate not just when they saw food, but also at the sight of the lab assistant who brought the food, or even the sound of their footsteps.
- **The Discovery:** He systematically showed that a neutral stimulus (like a bell) could become associated with an unconditioned stimulus (food, which naturally causes salivation).

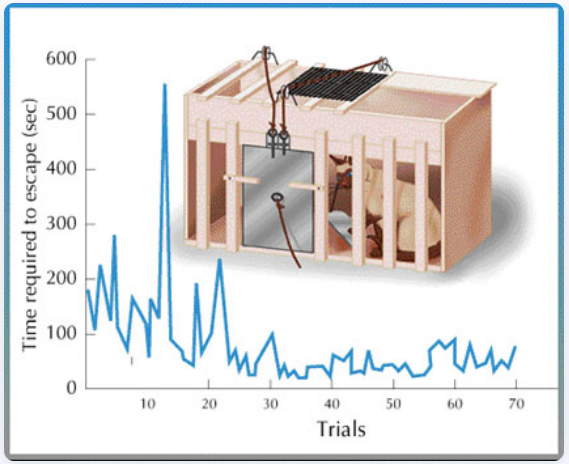


- This demonstrated **associative learning** – an organism can learn to associate one stimulus with another and thus learn to predict an upcoming event (the food).

RL's Roots: Thorndike & The Law of Effect

Edward Thorndike (1898-1911): The Law of Effect

- **Puzzle Box Experiments:** Thorndike placed cats in "puzzle boxes". To escape and get a food reward, the cats had to perform a specific action, like pulling a lever or stepping on a platform.
- **Observation:** Initially, cats performed many random actions. Over successive trials, the time it took to escape decreased as they gradually learned the connection between the correct action and the satisfying outcome (escape and food).



- **The Law of Effect (1911):** Thorndike formulated this principle:

> “ Responses that produce a satisfying effect in a particular situation become more likely to occur again in that situation, and responses that produce a discomforting effect become less likely to occur again in that situation. ”

- This was the first formal articulation of the **reinforcement principle**. It directly states that actions are strengthened or weakened by their consequences. This is the absolute core of how RL agents learn.

RL's Roots: Skinner & Shaping Behavior

B.F. Skinner (1930s-1950s): Operant (Instrumental) Conditioning

- **Active Learning:** Unlike Pavlov's classical conditioning (which focused on existing reflexes and associations), Skinner studied how **entirely new behaviors** could be learned and shaped when an animal *actively operates* on its environment.
- **The "Skinner Box":** A chamber where animals (e.g., rats, pigeons) could perform a specific action (like pressing a lever or pecking a disk) to receive a reward (food pellet) or avoid a punishment.
- **Shaping & Schedules of Reinforcement:**
  - Skinner went beyond Thorndike's simple trial-and-error. He demonstrated how **complex behaviors** could be built up through "shaping" – reinforcing successive approximations of the desired behavior.
  - He extensively studied **schedules of reinforcement**, discovering that *how and when* rewards are delivered dramatically affects the rate of learning, persistence of behavior, and response patterns. This has direct parallels to reward design and exploration strategies in modern RL.
- **Extinction:** He also showed that when reinforcement stops, the learned behavior gradually diminishes (extinguishes) – a concept relevant to understanding how learned associations can fade.

RL's Roots: Early Mathematical Ideas

Andrey Markov (Early 1900s): Markov Chains & Property

- Developed the theory of "Markov chains" – sequences of possible events (states) where the probability of transitioning to any future state depends *only* on the current state, not on the sequence of events that preceded it.
- **Markov Property:** $P(S_{t+1}|S_t, S_{t-1}, \ldots, S_0) = P(S_{t+1}|S_t)$.
- **Significance:** Provided the core mathematical framework (Markov Decision Processes, or MDPs) for modeling sequential decision-making problems.

Von Neumann & Morgenstern (1944): Utility Theory

- In their book "Theory of Games and Economic Behavior," they formalized the concept of rational decision-making under uncertainty.
- **Expected Utility Maximization:** Proposed that rational agents make choices as if they are maximizing the expected value of some "utility" function U associated with outcomes.
- **Significance:** This underpins the objective function in RL, where agents aim to maximize the expected sum of (discounted) rewards R, with rewards serving as a proxy for utility.

Why a Probabilistic Lens for RL?

- **Principled Handling of Uncertainty:** RL problems are inherently uncertain. Probability theory allows us to explicitly model and reason about:
  - Stochasticity in the environment (e.g., the robots movement might not always succeed, or traffic conditions might vary).
  - Partial observability (not knowing the true state perfectly).
  - Uncertainty in the learned model or value functions (which we will categorize as aleatoric and epistemic).
- **Unifying Framework:** Connects RL to broader statistical concepts like Bayesian inference and graphical models, providing a common language for diverse algorithms.
- **Enhanced Algorithms:** Leads to more robust algorithms, better exploration strategies (e.g., MaxEnt RL encourages exploring diverse behaviors), and a deeper understanding of complex behaviors.
- **Foundation for Advanced Topics:** Crucial for understanding Bayesian RL, control as inference, and many modern RL applications that deal with complex uncertainties.

Traditional vs. Probabilistic Reinforcement Learning: A Comparison

- **Traditional RL Perspective:**
  - Exploration primarily through ϵ-greedy or similar strategies
  - Focus on finding a single optimal policy
- **Probabilistic RL Perspective:**
  - Explicit reasoning about distributions over states, actions, and returns
  - Exploration naturally emerges through uncertainty reasoning
  - Often learns distribution over policies (or stochastic policies)
  - Makes the connection between control and inference explicit

Talk Roadmap

1. ML Fundamentals (Probabilistic View, including types of uncertainty)
2. Core RL Principles (featuring the Urban Delivery Robot example)
3. RL Through a Probabilistic Lens (Control as Inference, MaxEnt RL)
4. Recent Developments (RLHF, LLMs, Model-Based, Offline, MARL)
5. Conclusion

Probability: The Language of Uncertainty

- Machine learning, at its core, is about making sense of data and making predictions or decisions in the face of uncertainty.
- Probability theory provides the mathematical language and tools to:
  - Quantify uncertainty.
  - Represent beliefs about unknown quantities.
  - Update beliefs in light of new evidence.
- **What do these numbers we call "probabilities" actually represent?**

What Do We Mean by "Probability"? (Warm-Up 1/4)

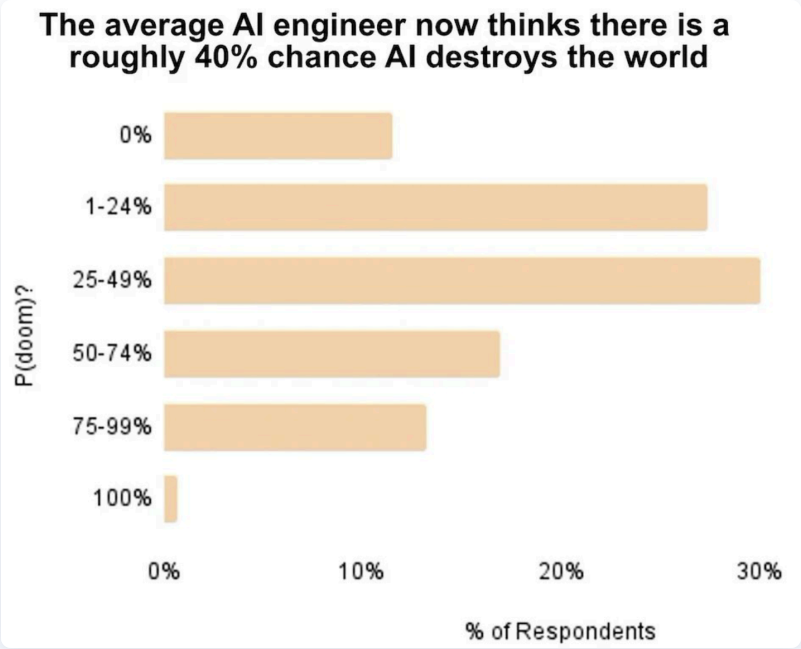
We use probabilities daily. But what do they *really* signify?
Consider these scenarios. Which interpretation of probability feels more natural for each?

1. **Football Analytics:** "Based on the shot's angle and distance, the probability of this player scoring a goal is 0.067."

   - Is this about long-run frequencies or a belief about this specific kick?

What Do We Mean by "Probability"? (Warm-Up 2/4)

2. **Existential Risk:** An expert states, "The probability of an unaligned superintelligence causing an existential catastrophe by 2070 is 99%."



- Can we repeat this event to get a frequency? Or is it a degree of belief?

What Do We Mean by "Probability"? (Warm-Up 3/4)

3. **Medical Diagnosis:** After an ultrasound, a doctor says, "There's an 80% probability it's a girl."

   - Is this based on how often the test is right, or confidence in this specific image?

What Do We Mean by "Probability"? (Warm-Up 4/4)

4. **The Metro Gamble:** A fare evader thinks, "The probability of an inspector being on this train is only 4%, and the penalty to pay if I get caught is only 20 times the ticket price, so it's worth the risk."

   - Is this 4% based on repeated observations or a one-time belief?

Different Flavors of Probability (1/2)

Let's revisit our examples and think about the *nature* of the probability in each:

1. **Football Goal (6.7%):**
    - **Informed by past data:** The 6.7% likely comes from analyzing thousands of *similar enough* kicks (how often they resulted in a goal).
2. **AI Existential Risk (99%):**
    - **A statement of belief:** This reflects an expert's current level of confidence or reasoned judgment about a complex, unique future event. There's no history of "AI doom trials" to count frequencies from.

Different Flavors of Probability (2/2)

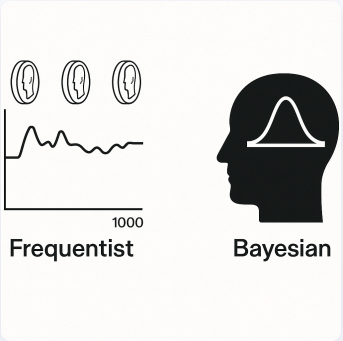
3. **Ultrasound Gender (80%):**
   - **Based on past accuracy:** The 80% likely reflects how often such ultrasound readings have correctly predicted gender in previous, similar cases (e.g., "in 100 *similar* readings, the prediction was correct 80 times").
4. **Metro Inspector (4%):**
   - **Based on observed frequency:** The 4% is most naturally understood as an estimate from past observations (e.g., "inspectors were present on 4 out of the last 100 trains I took/observed"). It describes how often one might expect to encounter an inspector over many train rides.

We see two main "flavors" emerging: probabilities based on long-run frequencies of repeatable events, and probabilities representing a degree of belief about specific, sometimes unique, situations (even if informed by data). Now, let's give these formal names...

Interpretations of Probability

Frequentist Interpretation

- Defines probability as the long-run relative frequency of an event in a large number of identical, repeatable trials.



Subjectivist (Bayesian) Interpretation

- Defines probability as a **degree of belief** or confidence that a rational agent assigns to an uncertain proposition or event.
- Beliefs are updated using Bayes' theorem as new information becomes available.

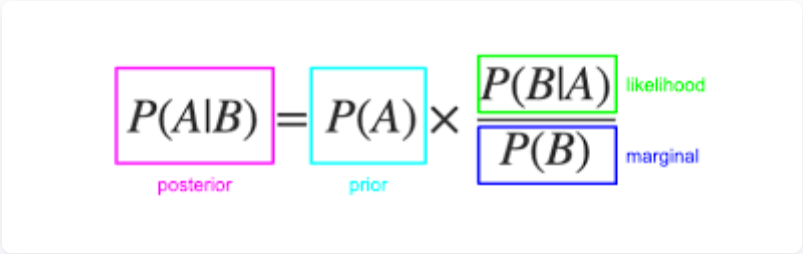
The **Subjectivist Interpretation** is central to Bayesian statistics and the probabilistic approach to ML and RL we will explore.

Cox's theorem (1946): for degrees of (subjective) beliefs to be rational and coherent, they have to follow Kolmogorov's axioms of probability (e.g., for a countable sequence of disjoint events $A_1, A_2, \ldots$, $P(\cup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} P(A_i)) \rightarrow$ **Both interpretations ultimately adhere to the same mathematical rules.**

Core Concepts in Probability

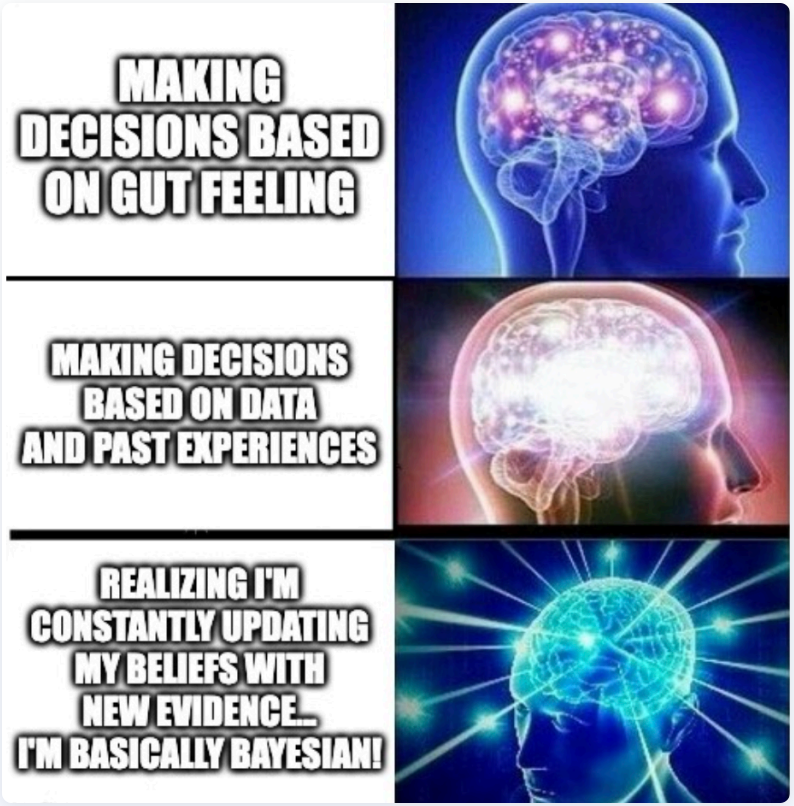
- **Random Variable (X):** A variable whose value is a numerical outcome of a random phenomenon.
  - *Example (Urban Delivery Robot): The `next_state` after an action is a random variable due to the 20% slip chance. The `battery_level` might also be modeled with noise in a more complex scenario.*
- **Conditional Probability:** $P(A|B) = P(A \cap B)/P(B)$. The probability of A given B has occurred.
- **Bayes' Rule:** $P(H|E) = \frac{P(E|H)P(H)}{P(E)}$
  - $P(H|E)$: Posterior probability of hypothesis H given evidence E.
  - $P(E|H)$: Likelihood of evidence E given hypothesis H.
  - $P(H)$: Prior probability of hypothesis H.
  - $P(E)$: Marginal likelihood of evidence E (normalizing constant).

Bayesian Inference: Learning as Belief Update



- Start with a **prior belief** $P(H)$ about a hypothesis (e.g., parameters of a model, the true value of a state).
- Observe **data (evidence)** E.
- Use the **likelihood** $P(E|H)$ (how probable is the data given the hypothesis) to update beliefs.
- Result: **Posterior belief** $P(H|E) = \frac{P(E|H)P(H)}{P(E)}$, a refined understanding of the hypothesis.
- This iterative process of updating beliefs is fundamental to how Bayesian systems learn.
  - *Robot Example (Urban Delivery Robot): It might initially have a uniform prior over which paths are good. After some trial and error (observing rewards and transitions), it updates its belief about the value of different paths (states/actions), effectively learning from experience.* This is a core concept we'll see in RL.

Embrace Bayesianism!



Supervised Learning: A Probabilistic Perspective

- **Goal:** Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, learn a function to predict y from $\mathbf{x}$.
- **Probabilistic Approach:** Model the conditional probability $p(y|\mathbf{x}; \theta)$, where θ are model parameters.
- **Maximum Likelihood Estimation (MLE):** Find θ that maximizes the likelihood of observing the training data.
  - $\theta_{MLE} = \arg\max_\theta \prod_i p(y_i|\mathbf{x}_i; \theta) = \arg\max_\theta \sum_i \log p(y_i|\mathbf{x}_i; \theta)$.
  - **Connection to Loss Functions:** Minimizing a loss function is often equivalent to MLE under a specific probabilistic model.
    - **Squared Error Loss (Regression):** Corresponds to MLE with a Gaussian likelihood $p(y|\mathbf{x}; \theta) = \mathcal{N}(y|f(\mathbf{x}; \theta), \sigma^2)$. Minimizing $\sum (y_i - f(\mathbf{x}_i; \theta))^2$.
    - **Cross-Entropy Loss (Classification):** Corresponds to MLE with a Categorical/Bernoulli likelihood (e.g., softmax output). Minimizing $-\sum y_i \log p(y_i|\mathbf{x}_i; \theta)$.

MAP Estimation

- **Maximum A Posteriori (MAP) Estimation:** Incorporates a prior distribution $p(\theta)$ over the parameters.
  - $\theta_{MAP} = \arg\max_\theta p(\theta|\mathcal{D}) = \arg\max_\theta [\log p(\mathcal{D}|\theta) + \log p(\theta)]$.
  - The prior $\log p(\theta)$ acts as a **regularization term**.
    - Example: Gaussian prior $p(\theta) \sim \mathcal{N}(0, \sigma_p^2 I)$ leads to L2 regularization (weight decay).
    - Example: Laplacian prior leads to L1 regularization.

Understanding Uncertainty in ML: Why It's Crucial

When our models make predictions, a single point estimate (like "70% chance of rain") often isn't enough. **Quantifying the uncertainty** around that prediction is vital:

- **Reliability & Trust:** How much should we trust this prediction? Is it a confident 70% or a very shaky 70%?
- **Informed Decision-Making:**
  - High uncertainty in a medical diagnosis might lead to more tests.
  - In autonomous driving, uncertainty about an object could trigger cautious maneuvers.
- **Active Learning:** Models can identify areas of high uncertainty to request more data, learning efficiently.
- **Exploration in RL:** Uncertainty about the environment or value of actions can guide an RL agent to explore more effectively.

Two primary types of uncertainty help us categorize and address this: **Aleatoric** and **Epistemic**.

Aleatoric vs. Epistemic: What's the Difference?

Let's consider an example: Throwing a standard six-sided die.

- The outcome is uncertain.
- **Question for you:** If we want to model the uncertainty of the die's outcome, is the uncertainty due to:
  - (a) The inherent randomness of the die throwing process?
  - (b) Our lack of knowledge?

(Image prompt: generate an image of a mathematician throwing a six-sided die, moroccan painting style)

The Answer: It depends on our assumptions!

- **If we assume the die is fair and the throw is "random" from our perspective (we don't model the detailed physics):** The uncertainty is **Aleatoric**. It's inherent randomness we can't reduce by knowing more about the die itself, even if we throw it an infinite number of times (beyond it being fair).
- **If we believe we *could* perfectly model the physics** (initial position, velocity, spin, air resistance, surface properties): Then any uncertainty in the outcome would be due to our lack of perfect knowledge of these initial conditions or the model parameters. This would be **Epistemic**. With a perfect physics model and perfect inputs, the outcome would be deterministic. With a certain number of throws, we could *infer* the outcome of the next throw.

This highlights how the type of uncertainty can depend on the scope and fidelity of our model.

Defining and Leveraging Uncertainty Types

Aleatoric Uncertainty (Data/Inherent Noise)

- Irreducible uncertainty inherent in the data-generating process or the system's natural stochasticity. Cannot be reduced even with infinite data of the *same kind*.
- **Sources:** Measurement noise, inherent randomness (like quantum effects, or our die roll example if simplified), unobserved influences (latent variables).
- **RL Impact:** Part of the environment's transition dynamics $P(s'|s,a)$. The agent learns a policy that is robust to or accounts for this inherent randomness.

Epistemic Uncertainty (Model/Knowledge Uncertainty)

- Uncertainty due to our lack of knowledge about the best model or its parameters. Reducible with more (diverse) data or a better model class.
- **Sources:** Limited training data (especially in certain regions of the input space), model misspecification,.
- **Example:** If a robot hasn't explored an area, its estimate of state values there will have high epistemic uncertainty.
- **RL Impact:** Crucial for **driving exploration**. High epistemic uncertainty about $Q(s,a)$ or the model $P(s'|s,a)$ can incentivize the agent to try those uncertain state-action pairs to gain more information.

More on uncertainty

Why Distinguish?

- **Aleatoric:** Informs us about fundamental predictability limits. We design systems to be robust to it.
- **Epistemic:** Tells us where our model is ignorant. We can actively reduce it by collecting more data in those areas (active learning, exploration in RL) or improving the model.

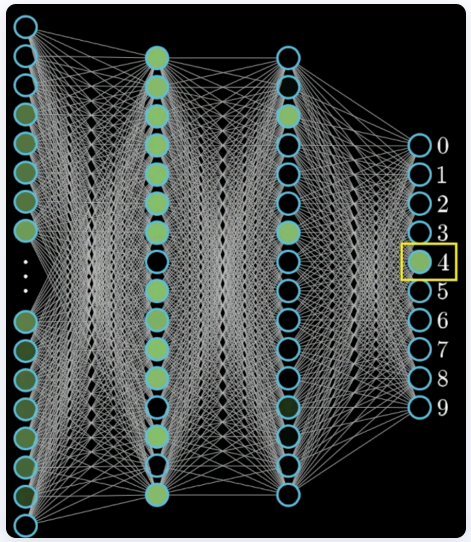
More details in "DEUP: Direct Epistemic Uncertainty Prediction" - Salem Lahlou*, Moksh Jain*, Hadi Nekoei, Victor Ion Butoi, Paul Bertin, Jarrid Rector-Brooks, Maksym Korablyov, Yoshua Bengio, TMLR 2023

Neural Networks: Powerful Function Approximators

At their core, Neural Networks (NNs) are computational models inspired by the structure of biological neural networks. They are powerful tools for learning complex, non-linear functions.

Basic Structure:

- Composed of layers of interconnected nodes (neurons).
- **Input Layer:** Receives the input data (e.g., state s in RL).
- **Hidden Layers:** Perform intermediate computations.
- **Output Layer:** Produces the final output (ideally the target).
- **Weights ($\mathbf{W}$) and Biases ($\mathbf{b}$):** Parameters associated with connections and neurons, learned during training.
- **Activation Functions (σ):** Introduce non-linearity (e.g., ReLU, sigmoid, tanh), allowing NNs to learn complex patterns.



3B1B

The general form for a feedforward network can be expressed as:

$$f(x;\theta) = \sigma_L(W_L\sigma_{L-1}(W_{L-1}\dots\sigma_1(W_1x+b_1)\dots+b_{L-1})+b_L)$$

where θ represents all weights and biases.

Training Neural Networks

The goal of training a neural network is to find the optimal parameters θ (weights and biases) that allow it to perform a desired task accurately.

1. Loss Function (L):

- **Supervised Learning Examples:**
  - Mean Squared Error (MSE) for regression: $L = \frac{1}{N}\sum(y_i - f(\mathbf{x}_i;\theta))^2$
  - Cross-Entropy Loss for classification.
- **Reinforcement Learning Examples (for value functions) - We will see this later today!**
  - Mean Squared Bellman Error for TD learning.

2. Gradient Descent:

- An iterative optimization algorithm used to minimize the loss function L.
- Parameters are updated in the direction opposite to the gradient of the loss: $\theta_{\text{new}} = \theta_{\text{old}} - \eta\nabla_\theta L$ where η is the learning rate.

3. Backpropagation Algorithm:

- An efficient algorithm for computing the gradient $\nabla_\theta L$ of the loss function with respect to *all* parameters θ in the network.

The Power of NNs: Universal Approximation

Universal Approximation Theorem:

- A foundational result stating that a feedforward neural network with a single hidden layer, a finite number of neurons, and an appropriate non-linear activation functions (e.g., sigmoid, ReLU), can approximate any continuous function on compact subsets of $\mathbb{R}^n$ to any desired degree of accuracy, given enough neurons
- **Implication:** Theoretically, NNs are expressive enough to represent arbitrarily complex functions, including intricate value functions or policies in RL. (Note: This doesn't guarantee they can be *learned* efficiently or generalize well from finite data).

Intelligence requires planning

- Supervised learning excels at one-shot predictions (y from x). We learn a mapping.
- However, the **quest for intelligent agents** often involves more than just prediction. It involves **planning** and making a **sequence of decisions** over time.
- In these scenarios, actions have consequences that affect future states and future rewards. The data is not i.i.d.; it's a stream of experience.
- How do we learn optimal sequences of actions when the environment is complex, uncertain (both aleatoric and epistemic!), and feedback (rewards) might be delayed or sparse?
- This is where **Reinforcement Learning (RL)** comes in. RL formalizes the problem of an agent learning to make optimal decisions in an environment to achieve a long-term cumulative goal. It's about learning *how to act* or *how to plan*.

How NNs are revolutionizing RL

Revolutionary Impact on Reinforcement Learning (Deep Reinforcement Learning - Deep RL):

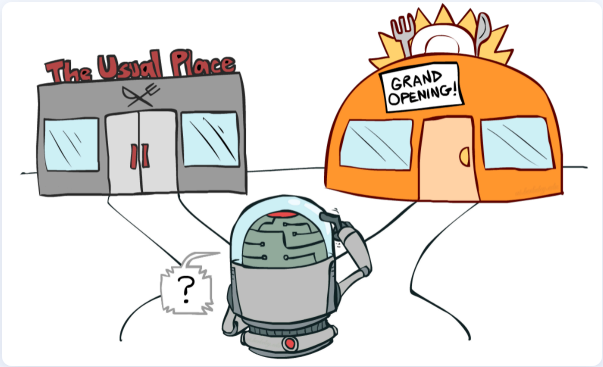
- **Scaling to High-Dimensional Inputs:** NNs can process raw, high-dimensional inputs like images (e.g., pixels from Atari games in DQN) or complex sensor data, where traditional tabular RL methods are infeasible.
- **Automatic Feature Extraction:** Deep NNs (especially Convolutional NNs for images) can learn relevant features from raw data, reducing the need for manual feature engineering.
- **Generalization:** They can generalize from states seen during training to similar, unseen states, which is crucial for large or continuous state spaces.
- **End-to-End Learning:** Enable learning complex behaviors directly from raw inputs to actions.
- **Breakthroughs:** This combination led to major successes like DQN mastering Atari games, AlphaGo defeating human champions, and advancements in robotics and other complex control tasks.

The Reinforcement Learning Problem

- **Agent:** The learner and decision-maker.
- **Environment:** Everything outside the agent; it reacts to the agent's actions and presents new situations.
- **State (S_t):** A representation of the environment at time t.
- **Action (A_t):** A choice made by the agent in state S_t.
- **Reward (R_{t+1}):** A scalar feedback signal received after transitioning from S_t to S_{t+1} due to action A_t.
- **Policy (π):** The agent's behavior; a mapping from states to actions (or distributions over actions).
  - $\pi(a|s) = P(A_t = a|S_t = s)$
- **Goal:** Learn a policy π^* that maximizes the **expected cumulative discounted reward** (Return).
  - Return: $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$
  - Discount factor $\gamma \in [0, 1]$: trades off immediate vs. future rewards.

The Exploration-Exploitation Dilemma

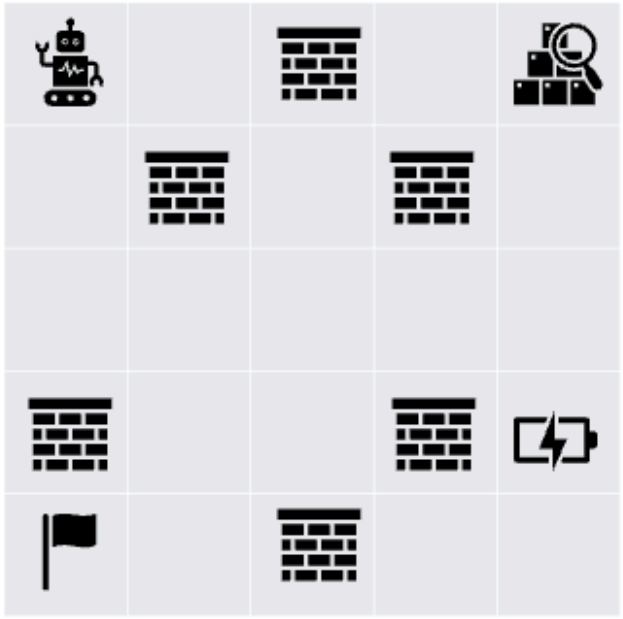
- **The Core Tension in RL:**
  - **Exploitation:** Maximize immediate rewards based on current knowledge
  - **Exploration:** Gather information to potentially find better strategies
- **Urban Delivery Robot Example:**
  - **Exploitation:** Always take shortest known path to package and delivery
  - **Exploration:** Try alternative routes to potentially discover faster paths
- **Why This Matters:**
  - Too much exploitation → Suboptimal behavior, may miss better solutions
  - Too much exploration → Excessive "wandering," inefficient learning
  - Finding this balance is critical for effective RL
- **The Probabilistic View** will give us principled tools to address this challenge through uncertainty quantification.



Core Characteristics & Challenges in RL

1. **Optimization:**
   - The ultimate goal is to find a strategy (policy) that leads to the **best possible long-term outcomes**.
   - This involves maximizing a cumulative utility function.
2. **Delayed Consequences (Credit Assignment):**
   - Actions taken now can have significant impacts on rewards received later.
   - Unlike supervised learning, feedback (rewards) can be sparse and temporally distant from the actions that caused them.
3. **Exploration vs. Exploitation:**
   - The agent must learn about its environment by interacting with it.
   - It cannot know what would have happened if it took a different action (counterfactual) without trying.
4. **Generalization:**
   - The ability to perform well in novel states or situations not encountered during training is crucial.
   - This allows knowledge to be transferred across related but different scenarios.
   - A key aspect of intelligence that distinguishes true learning from memorization, especially vital for Deep RL.

The Urban Delivery Robot MDP (5x5 Grid)



States
Robot position (x, y)
Battery level (from 0 to 5)
Package status (0 or 1)

Actions
Up (0)
Down (1)
Right (2)
Left (3)

Rewards
Succesful delivery: +100
Action cost: -1
Wall collision: -10
Battery depleted: -50

Transitions
0.8 probability of moving to the intended cell (except if wall)
0.2 probability of staying put
1 move = 1 unit of battery consumed
Landing on charging cell = battery level becomes 5
Landing on pickup cell without package = package status becomes 1

Episode ends
If succesful delivery
OR
If battery depleted

Markov Decision Processes (MDPs)

- A mathematical framework for modeling decision-making in situations where outcomes are partly random and partly under the control of a decision-maker.
- **Markov Property:** The future is independent of the past given the present.
  - $P(S_{t+1}|S_t, A_t, S_{t-1}, A_{t-1}, \ldots, S_0, A_0) = P(S_{t+1}|S_t, A_t)$
  - The current state S_t captures all relevant information from the history.
  - *Robot Example (Urban Delivery Robot): Its next state depends only on its current (location, battery, package status) and chosen action, not how it got there.*
- **Components of an MDP:**
  - $\mathcal{S}$: A finite set of states.
  - $\mathcal{A}$: A finite set of actions (or $\mathcal{A}_s$ for actions available in state s).
  - $P(s'|s, a)$: State transition probability function. $P(S_{t+1} = s'|S_t = s, A_t = a)$.
  - $R(s, a, s')$: Reward function. Expected reward after transitioning from s to s' due to action a.
  - γ: Discount factor, $\gamma \in [0, 1]$.

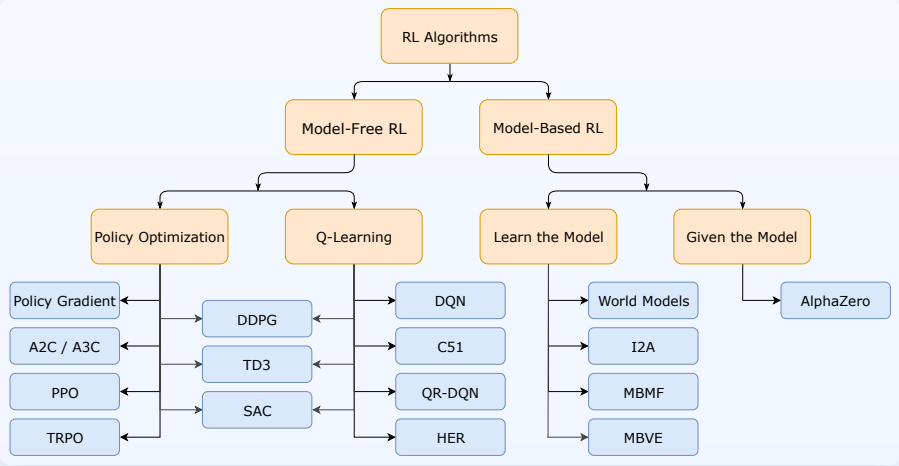
Policies and Value Functions

- **Policy (π):** A mapping from states to probabilities of selecting each possible action.
  - Deterministic policy: $a = \pi(s)$.
  - Stochastic policy: $\pi(a|s) = P(A_t = a|S_t = s)$.
  - *Robot Example (Urban Delivery Robot): A policy might tell it to move "Up" if its battery is high and it doesn"t have the package, and "Right" if it has the package and is near the delivery location.*
- **State-Value Function ($V^\pi(s)$):** The expected return starting from state s and following policy π.
  - $V^\pi(s) = E_\pi[G_t|S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s]$.
  - Measures "how good" it is to be in state s under policy π.
- **Action-Value Function ($Q^\pi(s, a)$):** The expected return starting from state s, taking action a, and thereafter following policy π.
  - $Q^\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a]$.
  - Measures "how good" it is to take action a in state s under policy π.

These concepts form the foundation for different families of RL algorithms

Reinforcement Learning Algorithm Taxonomy

- **Policy-Based Methods:**
  - Directly learn the policy function $\pi(a|s)$
  - Examples: REINFORCE, PPO, SAC (actor component)
  - **Advantages:** Can learn stochastic policies, natural for continuous actions
  - **Challenges:** Often high variance, sample inefficient without value functions
- **Value-Based Methods:**
  - Learn value functions ($V(s)$ or $Q(s,a)$), derive policy
  - Examples: Q-learning, DQN, SARSA
  - **Advantages:** Often more sample efficient, lower variance
  - **Challenges:** Harder for continuous actions, typically produce deterministic policies



Disclaimer

The next 10 slides are math-heavy.

If

- this is not your cup of tea, and you like it that way,
- of you are already familiar with the theory of MDPs,

then let's meet again in Slide 56.

If

- like me, you need to (think you) understand every mathematical detail to (have the impression you) understand algorithms

then give me your attention for the next few minutes, it's going to be worth it 🙂

Bellman Equations

Recursive relationships that decompose value functions into immediate reward plus discounted value of the next state.

- **Bellman Expectation Equation for V^π:**
  - $V^\pi(s) = E_\pi[R_{t+1} + \gamma V^\pi(S_{t+1}) | S_t = s]$
  - $V^\pi(s) = \sum_a \pi(a|s) \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^\pi(s')]$.
  - *Robot Example (Urban Delivery Robot): The value of being at (0,0) is the average (over its possible actions and their outcomes) of the immediate reward plus the discounted value of where it ends up.*
- **Bellman Expectation Equation for Q^π:**
  - $Q^\pi(s,a) = E_\pi[R_{t+1} + \gamma Q^\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]$
  - $Q^\pi(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \sum_{a'} \pi(a'|s') Q^\pi(s',a')]$.
  - Can also be written as: $Q^\pi(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^\pi(s')]$.

Proof of Bellman Expectation Equation

The value function $V^{\pi}(x)$ for a state x under a **deterministic policy** $\pi(x)$ can be derived as follows:

$$
\begin{aligned}
V^{\pi}(x) &= \mathbb{E}_{\pi}[G_0 \mid X_0 = x] && \text{(Definition of state-value function)} \\
&= \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_k \mid X_0 = x\right] && \text{(Definition of discounted return } G_0\text{)} \\
&= \mathbb{E}_{\pi}[R_0 + \gamma \sum_{k=0}^{\infty} \gamma^k R_{k+1} \mid X_0 = x] && \text{(Extracting the first reward } R_0\text{)} \\
&= \mathbb{E}_{\pi}[R_0 \mid X_0 = x] + \gamma \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{k+1} \mid X_0 = x\right] && \text{(Linearity of expectation)} \\
&= r(x, \pi(x)) + \gamma \mathbb{E}_{X_1 \sim p(\cdot|x,\pi(x))}\left[\mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{k+1} \mid X_1 = x'\right]\right] && \text{(Def. of } r(x,a) = \mathbb{E}[R_0|X_0 = x, A_0 = a]\text{;} \\
& && \text{Law of Total Exp. on future rewards, conditioning on } X_1\text{)} \\
&= r(x, \pi(x)) + \gamma \sum_{x' \in \mathcal{S}} p(x' \mid x, \pi(x)) \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{k+1} \mid X_1 = x'\right] && \text{(Expanding the expectation over } X_1\text{)} \\
&= r(x, \pi(x)) + \gamma \sum_{x' \in \mathcal{S}} p(x' \mid x, \pi(x)) \mathbb{E}_{\pi}\left[\sum_{j=0}^{\infty} \gamma^j R_j \mid X_0 = x'\right] && \text{(Stationarity: future rewards from } X_1 = x' \text{ are like rewards from a new} \\
& && \textit{start at} X_0 = x'\text{, re-indexing sum } k \to j \text{ for } R_{k+1} \to R_j \text{ from } X_1\text{)} \\
&= r(x, \pi(x)) + \gamma \sum_{x' \in \mathcal{S}} p(x' \mid x, \pi(x)) V^{\pi}(x') && \text{(Definition of value function } V^{\pi}(x')\text{)}
\end{aligned}
$$

The proofs in the stochastic policy case, and for Q-values, are similar.

Banach Fixed-Point Theorem

A crucial mathematical tool for proving convergence in dynamic programming and RL is the Banach Fixed-Point Theorem.

> **Banach Fixed-Point Theorem:**
> Given a complete normed vector space $\mathcal{X}$ (e.g., $\mathbb{R}^n$ with a norm $\|.\|$), and a mapping $f : \mathcal{X} \to \mathcal{X}$.
> If f is a **contraction mapping**, meaning for all $x, y \in \mathcal{X}$:
>
> $$\|f(x) - f(y)\| \leq \eta \|x - y\|$$
>
> for some constant $0 \leq \eta < 1$ (the contraction factor),
> then:
>
> 1. f has a **unique fixed point** x^* such that $f(x^*) = x^*$.
> 2. For any initial $x_0 \in \mathcal{X}$, the sequence $x_{k+1} = f(x_k)$ converges to this unique fixed point x^*.

This theorem is key to showing that iterative applications of Bellman operators converge to the true value functions.

Bellman Expectation Operator ($\mathcal{T}^\pi$) is a Contraction

Let V be a vector of state values in $\mathbb{R}^{|\mathcal{S}|}$. The Bellman expectation operator for a policy π is defined as:

$$(\mathcal{T}^\pi V)(s) = r(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) V(s')$$

Or in vector form: $\mathcal{T}^\pi V = \mathbf{r}^\pi + \gamma \mathbf{P}^\pi V$.

Theorem: The Bellman expectation operator $\mathcal{T}^\pi$ is a contraction mapping under the max-norm ($\|V\|_\infty = \max_s |V(s)|$) with contraction factor γ.

Proof

Theorem: The Bellman expectation operator $\mathcal{T}^\pi$ is a contraction mapping under the max-norm ($\|V\|_\infty = \max_s |V(s)|$) with contraction factor γ.

Proof:
For any two value vectors $V, V' \in \mathbb{R}^{|\mathcal{S}|}$:

$$
\begin{aligned}
\|\mathcal{T}^\pi V - \mathcal{T}^\pi V'\|_\infty &= \|(\mathbf{r}^\pi + \gamma \mathbf{P}^\pi V) - (\mathbf{r}^\pi + \gamma \mathbf{P}^\pi V')\|_\infty \\
&= \|\gamma \mathbf{P}^\pi (V - V')\|_\infty \\
&= \gamma \max_{s \in \mathcal{S}} \left| \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s))(V(s') - V'(s')) \right| && \text{(Def. of operator and max-norm)} \\
&\leq \gamma \max_{s \in \mathcal{S}} \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) |V(s') - V'(s')| && \text{(Triangle inequality, } \gamma \geq 0\text{)} \\
&\leq \gamma \max_{s \in \mathcal{S}} \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) \|V - V'\|_\infty && \text{(Def. of max-norm: } |V(s') - V'(s')| \leq \|V - V'\|_\infty\text{)} \\
&= \gamma \|V - V'\|_\infty \max_{s \in \mathcal{S}} \underbrace{\sum_{s' \in \mathcal{S}} P(s'|s, \pi(s))}_{=1} && \text{(Factoring out constant; probabilities sum to 1)} \\
&= \gamma \|V - V'\|_\infty
\end{aligned}
$$

Since $0 \leq \gamma < 1$, $\mathcal{T}^\pi$ is a contraction. By Banach's theorem, iterative application $V_{k+1} = \mathcal{T}^\pi V_k$ converges to a unique fixed point, which is V^π.

Policy Iteration Algorithm

Policy Iteration (PI) is an algorithm that finds an optimal policy by alternating between two steps:

1. **Initialize:** Start with an arbitrary policy π_0.
2. **Repeat** for $k = 0, 1, 2, \ldots$, until the policy is stable and no longer improves:
   - **(a) Policy Evaluation:**
     - Given the current policy π_k, compute its state-value function V^{π_k}.
     - This is done by solving the Bellman expectation equation:

       $$V^{\pi_k}(s) = r(s, \pi_k(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi_k(s)) V^{\pi_k}(s') \quad \forall s \in \mathcal{S}$$

       (This can be solved iteratively using $V_{j+1} = \mathcal{T}^{\pi_k} V_j$ which converges because $\mathcal{T}^{\pi_k}$ is a contraction, or by solving a system of linear equations).
   - **(b) Policy Improvement:**
     - Improve the policy by creating a new policy π_{k+1} that acts greedily with respect to V^{π_k}:

       $$\pi_{k+1}(s) = \arg\max_{a \in \mathcal{A}} \left(r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V^{\pi_k}(s') \right)$$

       This is equivalent to choosing a that maximizes $Q^{\pi_k}(s, a)$.

Convergence of Policy Iteration

Theorem: Policy Iteration is guaranteed to converge to an optimal policy V^* and π^* in a finite number of iterations for a finite MDP.

Proof Outline:

1. **Policy Evaluation Converges:**
   - We have already seen that PE converges because $\mathcal{T}^\pi$ is a contraction.
2. **Policy Improvement Theorem:**
   - The theorem states that $V^{\pi_{k+1}}(s) \geq V^{\pi_k}(s)$ for all $s \in \mathcal{S}$.
   - If π_k is not optimal, then there must be at least one state s where $V^{\pi_{k+1}}(s) > V^{\pi_k}(s)$, implying a strict improvement.

Bellman Optimality Equations

For the optimal policy π^*, the value functions satisfy the Bellman optimality equations.

- **Optimal State-Value Function ($V^*(s)$):** $V^*(s) = \max_a Q^*(s,a)$.
  - $V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$.
- **Optimal Action-Value Function ($Q^*(s,a)$):**
  - $Q^*(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')]$.
  - $Q^*(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$.
- If we know $Q^*(s,a)$, the optimal policy is to choose the action that maximizes $Q^*(s,a)$:
  - $\pi^*(s) = \arg\max_a Q^*(s,a)$.

Bellman Optimality Operator ($\mathcal{T}^*$) & Value Iteration Convergence

The Bellman optimality operator $\mathcal{T}^*$ is defined as:

$$(\mathcal{T}^* V)(s) = \max_{a \in \mathcal{A}} \left(r(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V(s') \right)$$

Theorem: The Bellman optimality operator $\mathcal{T}^*$ is a contraction mapping under the max-norm with contraction factor γ.

Proof

Theorem: The Bellman optimality operator $\mathcal{T}^*$ is a contraction mapping under the max-norm with contraction factor γ.

Proof Sketch: For any $V, V' \in \mathbb{R}^{|\mathcal{S}|}$:

$$
\begin{aligned}
\|\mathcal{T}^*V - \mathcal{T}^*V'\|_\infty &= \max_{s\in\mathcal{S}} \left| \max_{a\in\mathcal{A}} \left(r(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s') \right) - \max_{a'\in\mathcal{A}} \left(r(s,a') + \gamma \sum_{s'} P(s'|s,a')V'(s') \right) \right| \\
&\leq \max_{s\in\mathcal{S}} \max_{a\in\mathcal{A}} \left| \left(r(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s') \right) - \left(r(s,a) + \gamma \sum_{s'} P(s'|s,a)V'(s') \right) \right| \\
&= \max_{s\in\mathcal{S}} \max_{a\in\mathcal{A}} \left| \gamma \sum_{s'} P(s'|s,a)(V(s') - V'(s')) \right| \\
&\leq \gamma \max_{s\in\mathcal{S}} \max_{a\in\mathcal{A}} \sum_{s'} P(s'|s,a)|V(s') - V'(s')| \\
&\leq \gamma \max_{s\in\mathcal{S}} \max_{a\in\mathcal{A}} \sum_{s'} P(s'|s,a)\|V - V'\|_\infty \\
&= \gamma\|V - V'\|_\infty \underbrace{\max_{s\in\mathcal{S}} \max_{a\in\mathcal{A}} \sum_{s'} P(s'|s,a)}_{=1} \\
&= \gamma\|V - V'\|_\infty
\end{aligned}
$$

Trick: Using $\left|\max_x f(x) - \max_x g(x)\right| \leq \max_x |f(x) - g(x)|$

Value Iteration

1. The optimal value function V^* is a fixed point of $\mathcal{T}^*$, i.e., $V^* = \mathcal{T}^* V^*$. (This is Bellman's Optimality Equation).
2. Since $\mathcal{T}^*$ is a contraction mapping with $0 \leq \gamma < 1$, by Banach's Fixed-Point Theorem:
   - $\mathcal{T}^*$ has a unique fixed point. This unique fixed point must be V^*.
   - The sequence generated by **Value Iteration**, $V_{k+1} = \mathcal{T}^* V_k$, converges to V^* for any initial V_0.
3. **Algorithm**:
   - Initialize $V \in \mathbb{R}^{|\mathcal{S}|}$ however you want (zeros for example)
   - Repeat until there is no much change:
     - $V \leftarrow \mathcal{T}^* V$
   - Extract the optimal policy π^*:
     - For each state s:
       - $\pi^*(s) = \arg\max_a \left[r(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s') \right]$

We have now derived two algorithms (PI and VI) to provably solve RL problems when the world model is known !

Model-Free RL: Temporal Difference (TD) Learning

When the model ($P(s'|s,a)$, $R(s,a,s')$) is unknown, we learn directly from experience samples (s,a,r,s').

Temporal Difference (TD) Learning:

- Learns from incomplete episodes by "bootstrapping": updating value estimates based on other learned estimates: often more sample efficient than methods that wait for full episode returns.
- Does not require waiting until the end of an episode to update values.
- **TD(0) Update for State-Values ($V^\pi(S_t)$):**
  After observing a transition $(S_t, A_t, R_{t+1}, S_{t+1})$:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{TD Target}} - V(S_t) \right]$$

  - α: Learning rate.
  - **TD Error ($\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$):** The difference between the TD target and the current estimate $V(S_t)$. The update moves $V(S_t)$ towards the TD target.

Model-Free RL: Q-Learning for Action-Values

Q-Learning (Off-Policy TD Control):

- Learns the optimal action-value function, $Q^*(s, a)$, directly.
- **Update Rule for Action-Values ($Q(S_t, A_t)$):**
  After observing a transition $(S_t, A_t, R_{t+1}, S_{t+1})$:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t) \right]$$

  - The agent learns about the greedy policy (by using $\max_{a'} Q(S_{t+1}, a')$ in the target) while potentially following a different behavior policy to collect data (off-policy).
  - Once $Q^*(s, a)$ is learned, the optimal policy is to choose a that maximizes $Q^*(s, a)$.

Deep Q-Learning

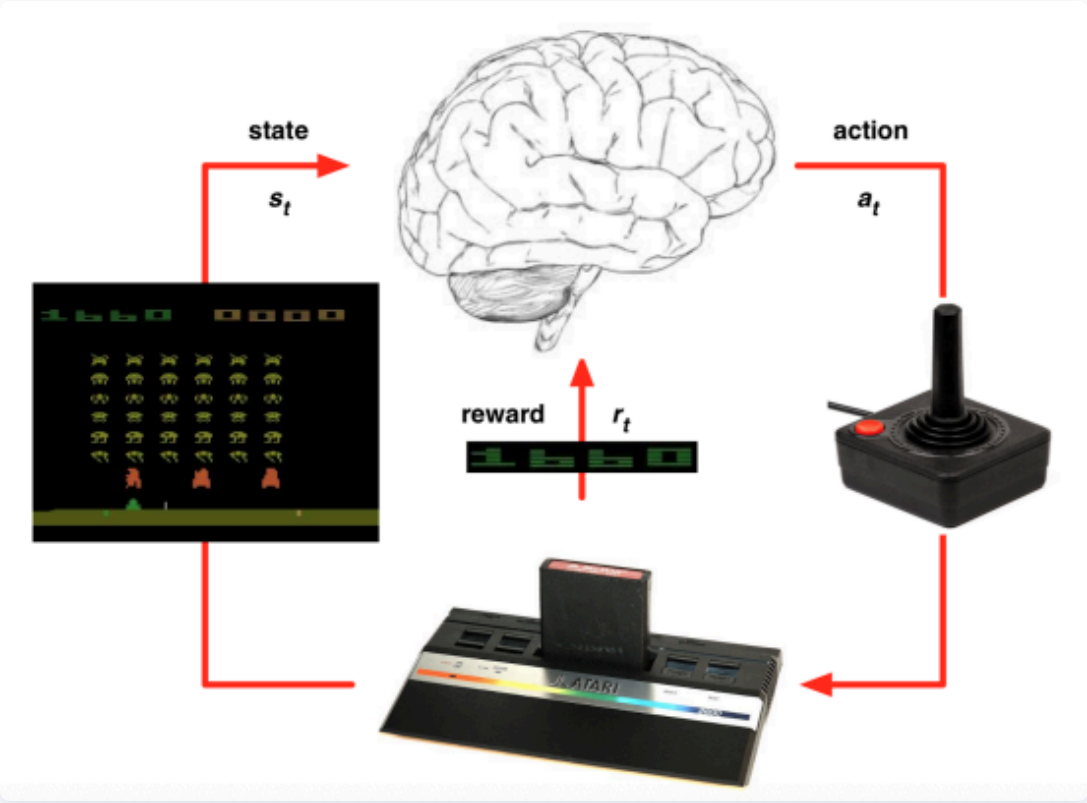
Function Approximation with Neural Networks:

- In complex problems with large state/action spaces, representing $V(s)$ or $Q(s, a)$ as tables is infeasible.
- **Neural Networks** can be used as powerful function approximators:
  - State-value approximation: $V(s; \theta) \approx V^*(s)$
  - Action-value approximation: $Q(s, a; \theta) \approx Q^*(s, a)$
- The parameters θ of the neural network are updated using methods like gradient descent to minimize the error between the network's output and the TD targets.
  - For V-values: Minimize $((R_{t+1} + \gamma V(S_{t+1}; \theta)) - V(S_t; \theta))^2$
  - For Q-values: Minimize $((R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a'; \theta)) - Q(S_t, A_t; \theta))^2$
- This allows RL to scale to problems with continuous or very large discrete state spaces.

Exploration Strategies: Examples

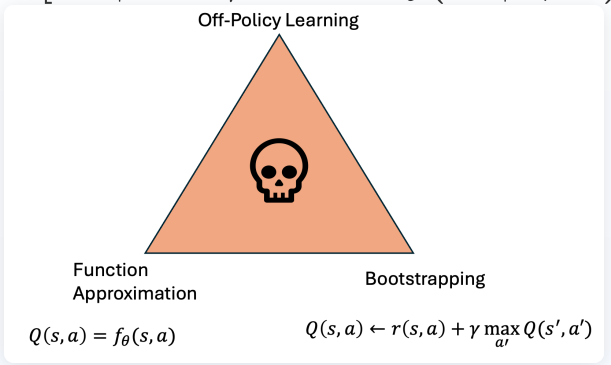
- **Simple Strategies:**
  - **ϵ-greedy:** With probability $1 - \epsilon$, choose the greedy action (exploit); with probability ϵ, choose a random action (explore). ϵ can be annealed over time.
  - **Optimistic Initialization:** Initialize Q-values to high values to encourage trying all actions initially.
- **More Advanced Strategies (often leveraging uncertainty):**
  - **Probability Matching / Thompson Sampling (Posterior Sampling):** Maintain a posterior distribution over Q-values (or model parameters). Sample an action according to its probability of being optimal.
  - **Intrinsic Motivation / Curiosity:** Add internal rewards for visiting novel states or for actions that lead to surprising outcomes (high prediction error of a learned dynamics model).
  - **MaxEnt RL (discussed later):** Entropy term in the objective naturally encourages exploration.

The Atari games milestone



Q-Learning: Challenges & The Deadly Triad

- **Q-Learning Update Rule (recap):**
  - $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)].$



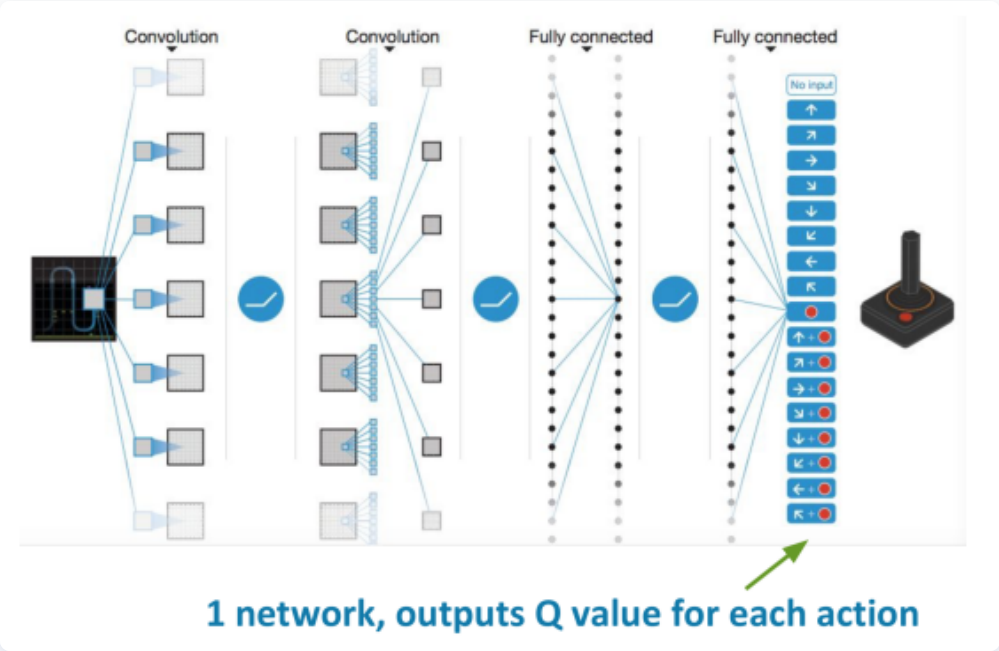
- **Challenges with Function Approximation (e.g., Neural Networks):**
  - When Q-functions are approximated (e.g., $Q(s, a; \theta)$), convergence is not always guaranteed, especially with off-policy learning and bootstrapping.
  - **The Deadly Triad:** The combination of:
    1. **Function Approximation** (e.g., linear, neural networks)
    2. **Bootstrapping** (updating estimates based on other estimates)
    3. **Off-policy training** (learning about a policy different from the one used to generate data)

    ...can lead to instability and divergence of value estimates.
  - DQN introduced techniques (experience replay, target networks) to mitigate these issues for deep NNs.

Deep Q-Networks (DQN)

- Combines Q-Learning with deep neural networks to approximate $Q^*(s, a; \theta)$.
- Addresses challenges of using NNs with RL (non-stationarity, correlated samples, deadly triad).
- **Innovations:**
  1. **Experience Replay:** Store transitions (s, a, r, s') in a replay buffer $\mathcal{D}$. Sample minibatches randomly from $\mathcal{D}$ to train the Q-network. Breaks correlations, reuses data.
  2. **Target Network:** Use a separate target Q-network $Q(s, a; \theta^-)$ for calculating TD targets. Its weights θ^- are periodically copied from the online Q-network θ or slowly tracked. Stabilizes training.
- **Loss Function (for updating θ):**
  - $L(\theta) = E_{(s,a,r,s') \sim \mathcal{D}}[(y - Q(s, a; \theta))^2]$, where target $y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$.
- Achieved human-level performance on Atari games from raw pixel inputs.

DQN Architecture Example



- **Input:** State representation (e.g., raw pixels from a game screen, or features like robots (x,y,battery,package) status).
- **Network Body:** Typically convolutional layers for image inputs, followed by fully connected layers.
- **Output:** A vector of Q-values, one for each possible discrete action in the current state.
- The action with the highest Q-value is chosen by the greedy policy.

Policy Gradient Methods: Overview

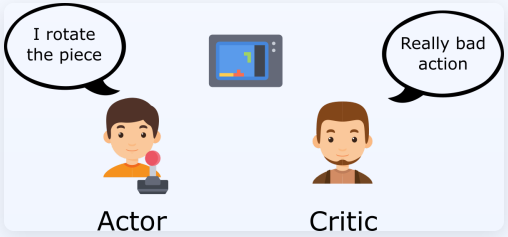
- Directly learn the parameters θ of a policy $\pi(a|s;\theta)$.
- Optimize an objective $J(\theta) = E_{\tau\sim\pi_\theta}[\sum_t R(s_t, a_t)]$.
- Update parameters by ascending the gradient $\nabla_\theta J(\theta)$.
- **Policy Gradient Theorem:**
  - $\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta}[(\sum_t \nabla_\theta \log \pi(A_t|S_t;\theta))(\sum_t R(S_t, A_t))]$.
  - Often use G_t (return from time t) or $Q^\pi(S_t, A_t)$ or Advantage $A^\pi(S_t, A_t) = Q^\pi(S_t, A_t) - V^\pi(S_t)$ as the weighting factor.
  - $\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi(A_t|S_t;\theta) A^\pi(S_t, A_t)]$.
- **REINFORCE Algorithm:** A basic Monte Carlo policy gradient method.

Actor-Critic Methods & Types (A2C/A3C)

- Combine policy gradients with learned value functions.
  - **Actor:** The policy $\pi(a|s;\theta)$, decides which action to take.
  - **Critic:** The value function (e.g., $V(s;\mathbf{w})$ or $Q(s,a;\mathbf{w})$), evaluates the actions taken by the actor.
- The critic estimates the return or advantage, providing a lower variance gradient signal for the actor.
- Actor update: $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi(A_t|S_t;\theta)\delta_t$.
- Critic update (e.g., for V): $\mathbf{w} \leftarrow \mathbf{w} + \beta\delta_t \nabla_{\mathbf{w}} V(S_t;\mathbf{w})$, where $\delta_t = R_{t+1} + \gamma V(S_{t+1};\mathbf{w}) - V(S_t;\mathbf{w})$.
- **Advantage Actor-Critic (A2C):** Synchronous version, waits for all actors to finish before updating.
- **Asynchronous Advantage Actor-Critic (A3C):** Uses multiple parallel actors with local copies of the model, updating a global model asynchronously. Can be more efficient.

Method	Training Time	Mean	Median
DQN	8 days on GPU	121.9%	47.5%
Gorila	4 days, 100 machines	215.2%	71.3%
D-DQN	8 days on GPU	332.9%	110.9%
Dueling D-DQN	8 days on GPU	343.8%	117.1%
Prioritized DQN	8 days on GPU	463.6%	127.6%
A3C, FF	1 day on CPU	344.1%	68.2%
A3C, FF	4 days on CPU	496.8%	116.6%
A3C, LSTM	4 days on CPU	623.0%	112.6%

Table 1. Mean and median human-normalized scores on 57 Atari games using the human starts evaluation metric. Supplementary Table S3 shows the raw scores for all games.



Proximal Policy Optimization (PPO)

- A state-of-the-art policy gradient method known for its stability and sample efficiency.
- Aims to take the largest possible improvement step on the policy per iteration without stepping too far and causing performance collapse.
- **Clipped Surrogate Objective:**
  - Let $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ be the probability ratio.
  - $L^{CLIP}(\theta) = E_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$.
  - $\hat{A}_t$ is an estimator of the advantage function.
  - The clipping discourages large policy updates.
- **Full PPO Objective (often includes):**
  - $L^{PPO}(\theta) = E_t[L^{CLIP}(\theta) - c_1 L^{VF}(\theta) + c_2 S[\pi_\theta](s_t)]$.
    - $L^{VF}(\theta)$: Value function error term (e.g., squared error $(V_\theta(s_t) - V_t^{target})^2$).
    - $S[\pi_\theta](s_t)$: Entropy bonus to encourage exploration.

From Traditional to Probabilistic RL

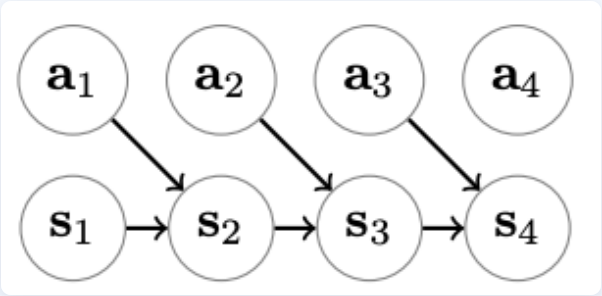
- **Limitations of Traditional RL Approaches:**
  - Exploration is often heuristic (e.g., ϵ-greedy)
  - Handling uncertainty is typically implicit
  - Standard approaches can converge to brittle deterministic policies
  - The "deadly triad" issues with function approximation
- **What a Probabilistic Perspective Offers:**
  - Principled treatment of different uncertainty types (aleatoric vs. epistemic)
  - Explicit modeling of beliefs and their updates
  - Natural exploration through information-seeking behavior
  - Theoretical connections to other fields (Bayesian inference, information theory)
  - More robust, stochastic policies with built-in exploration
- **How can we formally frame RL as probabilistic inference?**

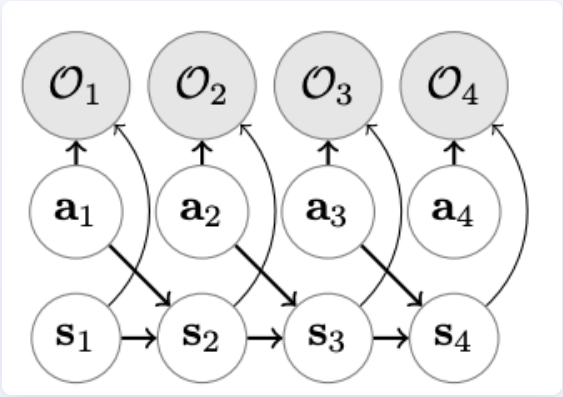
Standard RL vs. MaxEnt RL Objectives

- **Standard RL Objective:** Maximize expected cumulative reward.
  - $J(\pi) = E_{\tau \sim \pi}[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)]$.
  - Focuses solely on exploiting known high-reward paths once discovered.
  - Can lead to deterministic, potentially brittle policies.
- **Maximum Entropy (MaxEnt) RL Objective:** Maximize expected cumulative reward AND policy entropy.
  - $J(\pi) = E_{\tau \sim \pi}[\sum_{t=0}^{T} \gamma^t (r(s_t, a_t) + \alpha H(\pi(\cdot|s_t)))]$.
  - $H(\pi(\cdot|s_t)) = -E_{a \sim \pi(\cdot|s_t)}[\log \pi(a|s_t)]$ is the policy entropy.
  - Temperature α balances reward vs. entropy (randomness/exploration).
  - Encourages policies that are not only high-rewarding but also explore diverse behaviors and are less predictable.
  - Leads to "softer" policies that assign non-zero probability to multiple good actions.

Control as Probabilistic Inference: The Core Idea

- **Traditional View:** RL maximizes rewards. Inference finds probable explanations.
- **Probabilistic View:** Frame RL as an inference problem in a specific graphical model.
  - Goal: Infer actions that are *likely to be optimal*.
- Introduce binary **optimality variables** O_t for each time step t.
  - $O_t = 1$ signifies that the state-action pair (s_t, a_t) is part of an "optimal" trajectory.
  - We need a way to define $P(O_t = 1 | s_t, a_t)$ based on rewards. (More on this next!)
- The overall task becomes computing the posterior probability of actions given that they are optimal: $P(a_{1:T} | s_{1:T}, O_{1:T} = 1)$.





Defining an "Optimality Score"

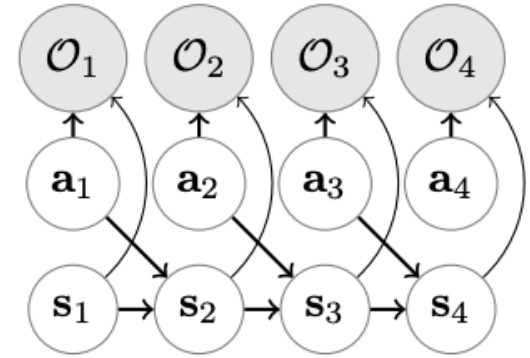
- We assign a **score (or potential)** for this $O_t = 1$ event, based on the immediate reward $r(s_t, a_t)$:

$$P(O_t = 1 \mid s_t, a_t) = \exp\left(\frac{r(s_t, a_t)}{\alpha}\right)$$

  *(Note: We use the "P" notation here by convention from the literature. Think of this as an **unnormalized score** $\notin [0, 1]$ that reflects preference.)*

- **Interpreting this Score:**
  - **Relative Preference:** Higher rewards $r(s_t, a_t)$ lead to exponentially higher scores, strongly preferring better actions.
  - **Not a Standalone Probability:** This score doesn't have to be ≤ 1. Its role is to contribute to a larger model. Properly normalized probabilities (that sum to 1) will emerge later when we consider all alternative actions and trajectories.
- **Role of Temperature ($\alpha > 0$):**
  - Controls the "softness" of preference:
    - Small α: Scores become very sensitive to reward differences (near-deterministic preference for the best).
    - Large α: Scores are less sensitive, allowing more stochasticity and exploration.

The Graphical Model for Control as Inference



- **Model Components:**
  - States s_t, Actions a_t.
  - Dynamics: $P(s_{t+1}|s_t, a_t)$.
  - Optimality variables O_t.
  - Prior over actions $P(a_t|s_t)$ (can be uniform if not specified).
- **Joint Probability (Trajectory $\tau = (s_1, a_1, \ldots, s_T, a_T)$):**
  - $P(\tau, O_{1:T} = 1) = P(s_1) \prod_{t=1}^{T} P(a_t|s_t) P(s_{t+1}|s_t, a_t) P(O_t = 1|s_t, a_t)$.
  - $P(\tau|O_{1:T} = 1) \propto P(s_1) \left(\prod_{t=1}^{T} P(a_t|s_t) P(s_{t+1}|s_t, a_t) \right) \exp\left(\sum_{t=1}^{T} r(s_t, a_t)/\alpha \right)$.
- The most probable trajectory under this posterior is one that is dynamically feasible and has high cumulative reward.

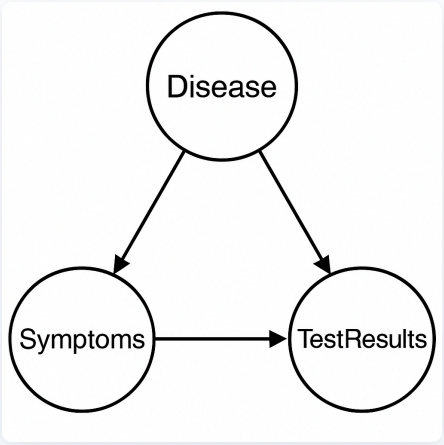
Aside: Probabilistic Graphical Models (PGMs)

- **Recap:** In the first part of our talk, we discussed how probability helps us quantify uncertainty and update beliefs (e.g., using Bayes' Rule for a few variables).
- **PGMs: Scaling Up Probabilistic Reasoning:**
  - PGMs are a visual language for representing complex probabilistic relationships among many random variables.
  - **Nodes:** Represent random variables (e.g., states s_t, actions a_t, optimality O_t).
  - **Edges (Arrows/Lines):** Represent probabilistic dependencies.
    - *Directed edges:* Indicate influences (e.g., $S_t, A_t \to S_{t+1}$).
- **Why use them?**
  - **Structure:** Clearly visualize problem assumptions.
  - **Modularity:** Break down complex systems.
  - **Efficient Reasoning:** Enable algorithms like *message passing* for inference.

Our RL problem, with states, actions, and optimality, is modeled as such a PGM!

Aside: Inference in PGMs – Asking Questions

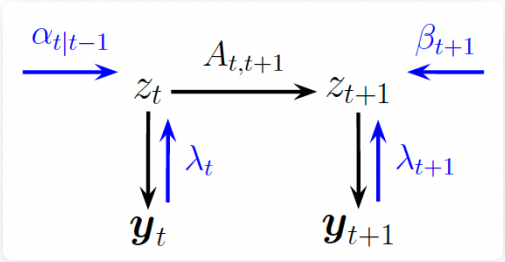
- Once we have a PGM, we use it to answer questions – this is **probabilistic inference**.
- **Typical Goal:** Compute the probability distribution of some *hidden* (unobserved) variables, given some *observed* variables (evidence).
  - Example: $P(\text{Disease}|\text{Symptoms}, \text{TestResults})$
  - In our RL context: $P(\text{Actions}|\text{States}, \text{Optimality } O_{1:T} = 1)$



- **Common Inference Tasks:**
  - **Marginal Inference:** $P(X_i|E)$ – Probability of a specific variable X_i.
  - **Maximum A Posteriori (MAP) Inference:** $\arg\max_X P(X|E)$ – Most likely configuration.
- **The Challenge:** Brute-force inference (summing/maximizing over all possibilities) is often computationally intractable.
- **Solution:** Efficient algorithms like **Message Passing**.

Aside: Message Passing for PGM Inference

- **The Intuition:** Variables in the PGM "talk" to their neighbors.
  - They exchange "messages" (summaries of information or beliefs)
  - Each message a node sends summarizes what it has learned from all *other* neighbors



- **How it Works (Conceptually):**
  - Local computations at each node based on received messages and local factors
  - Messages are passed along the graph's edges
  - Eventually, each node computes its updated belief (posterior probability)
- **Main Algorithms:**
  - **Belief Propagation (Sum-Product):** Computes exact marginals in tree-structured graphs
  - **Max-Product:** Finds MAP configuration
  - The "backward messages" $\beta_t(s_t)$ we'll see in RL are a specific type of message

Backward Messages & The Emergence of Soft Values

- **Goal Recap:** We want to find optimal actions by computing $P(a_t|s_t, O_{1:T} = 1)$. This requires understanding the value of states through backward message passing.
- **Backward Messages ($\beta_t(s_t)$):** These messages quantify $P(O_{t:T} = 1|s_t)$, the probability of all *future steps being optimal* (from time t to T), given current state s_t.
- **Recursive Calculation:**

$$\beta_t(s_t) = \sum_{a_t} P(a_t|s_t) \underbrace{P(O_t = 1|s_t, a_t)}_{\exp(r(s_t,a_t)/\alpha)} \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t) \underbrace{\beta_{t+1}(s_{t+1})}_{\exp(V_{t+1}(s_{t+1})/\alpha)}$$

  (Here, $P(a_t|s_t)$ is a prior policy, e.g., uniform. We define the soft value $V_t(s_t) = \alpha \log \beta_t(s_t)$).

- **This Recursion as a Soft Value Update:**
  Substituting $\beta_t(s_t) = \exp(V_t(s_t)/\alpha)$ and taking $\alpha \log(\cdot)$ of both sides, we get:

$$V_t(s_t) = \alpha \log \sum_{a_t} P(a_t|s_t) \exp\left(\frac{r(s_t, a_t)}{\alpha} + \log \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t) \exp\left(\frac{V_{t+1}(s_{t+1})}{\alpha}\right)\right)$$

  This equation for $V_t(s_t)$ is a form of the **soft Bellman equation**.

The Soft Bellman Connection

Recall: Standard Bellman Backup
(e.g., Bellman Expectation with $\gamma = 1$)

$$V^{\pi}(s) = \sum_{a} \pi(a|s) \left(R_{sa} + \mathbb{E}_{s'}[V^{\pi}(s')] \right)$$

Our Soft Value Update (from previous slide):

$$V_t(s_t) = \alpha \log \sum_{a_t} P(a_t|s_t) \exp\left(\frac{Q_{soft}(s_t, a_t)}{\alpha} \right)$$

where the **soft action-value** is:

$$Q_{soft}(s_t, a_t) = r(s_t, a_t) + \underbrace{\alpha \log \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t) \exp\left(\frac{V_{t+1}(s_{t+1})}{\alpha} \right)}_{\text{Soft expected value of next state}}$$

Optimal Policy (π^*):

- The optimal policy $\pi^*(a_t|s_t) = P(a_t|s_t, O_{t:T} = 1)$ is derived from the terms that make up $\beta_t(s_t)$:

$$\pi^*(a_t|s_t) \propto P(a_t|s_t) \cdot \exp\left(\frac{r(s_t, a_t)}{\alpha} \right) \cdot \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t) \exp\left(\frac{V_{t+1}(s_{t+1})}{\alpha} \right)$$

- This simplifies using the soft Q-value:

$$\pi^*(a_t|s_t) \propto P(a_t|s_t) \exp\left(\frac{Q_{soft}(s_t, a_t)}{\alpha} \right)$$

Benefits of the Control as Inference View

- **Unified Framework:** Provides a common mathematical language for RL, control, and probabilistic modeling.
- **Principled Derivation of Algorithms:** Many RL algorithms (Value Iteration, Policy Iteration, some policy gradients) can be re-derived as inference procedures.
- **Natural Exploration:** Entropy term in MaxEnt RL (from variational inference) encourages exploration.
- **Robustness:** MaxEnt policies can be more robust to model inaccuracies and noise.
- **Compositionality & Hierarchy:** PGMs offer tools for building more complex, structured models (e.g., hierarchical RL).
- **Inverse RL:** If rewards are unknown, they can be inferred by assuming expert trajectories are samples from $P(\tau|O_{1:T} = 1)$.
- **Partially Observed MDPs (POMDPs):** Inference naturally extends to hidden states.
- **Multi-modality:** Can learn multi-modal policies (multiple ways to solve a task)

Maximum Entropy Reinforcement Learning

- **Objective:** Maximize the standard RL objective augmented with an entropy term.
  - $J(\pi) = \sum_{t=0}^{T} E_{(s_t,a_t)\sim\rho_\pi}[r(s_t,a_t) + \alpha H(\pi(\cdot|s_t))]$.
  - $H(\pi(\cdot|s_t)) = -E_{a\sim\pi(\cdot|s_t)}[\log \pi(a|s_t)]$ is the entropy of the policy at state s_t.
- **Achieving this Objective: The Probabilistic Inference Connection:**
  - The principles of MaxEnt RL are deeply connected to the **control as probabilistic inference** framework.
  - The **soft value functions (V_{soft}, Q_{soft})** and **soft Bellman equations** that emerge from that inference process (e.g., using $P(O_t = 1|s_t, a_t) = \exp(r/\alpha)$ and backward messages) are precisely the value functions for policies optimizing this reward + entropy objective.
  - The **optimal stochastic policy** derived in that framework, typically a **Boltzmann policy** of the form $\pi^*(a|s) \propto \exp(Q_{soft}(s,a)/\alpha)$, naturally maximizes cumulative reward while also maintaining high entropy, fulfilling the MaxEnt objective.
  - The **temperature parameter** α is the same crucial element in both: it balances exploitation (reward) and exploration (entropy) in the objective, and it dictates the stochasticity of the optimal policy and the "softness" of the value functions.

Comparing Traditional and Probabilistic RL

Aspect	Traditional RL	Probabilistic/MaxEnt RL
Policy Type	Often deterministic	Naturally stochastic
Exploration	Typically separate mechanism (e.g., ϵ-greedy)	Integrated via entropy/uncertainty
Objective	Maximize expected return	Maximize return + entropy/information
Uncertainty	Implicit or absent	Explicitly modeled
Sample Efficiency	Often requires more samples	Can be more efficient with good priors
Robustness	May find brittle solutions	Often more robust to perturbations
Computation	Generally lighter	Often more computational overhead
Examples	DQN, A3C	Soft Actor-Critic

The probabilistic approach often trades increased computational complexity for better exploration, robustness, and uncertainty handling.

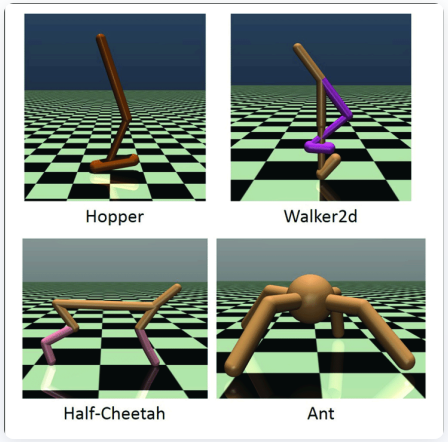
Summary: Soft Value Functions in MaxEnt RL

- **Soft State-Value Function ($V_{soft}(s)$):**
  - $V_{soft}(s_t) = E_{a_t \sim \pi}[Q_{soft}(s_t, a_t) - \alpha \log \pi(a_t|s_t)]$.
- **Soft Action-Value Function ($Q_{soft}(s, a)$):**
  - Bellman equation for Q_{soft}:
    - $Q_{soft}(s_t, a_t) = r(s_t, a_t) + \gamma E_{s_{t+1} \sim P}[V_{soft}(s_{t+1})]$.
- **Optimal Soft Policy:** $\pi^*(a|s) \propto \exp(Q_{soft}(s, a)/\alpha)$.
  - This is a Gibbs/Boltzmann distribution. Higher Q-values get exponentially more probability.

Soft Actor-Critic (SAC): Overview

- An **off-policy actor-critic** algorithm based on the MaxEnt RL framework.
- Aims for high sample efficiency and stability.
- **Key Components:**
  1. **Stochastic Actor (Policy π_ϕ):** Learns a policy that maximizes reward + entropy.
  2. **Critic(s) (Soft Q-functions $Q_{\theta_1}, Q_{\theta_2}$):** Estimate soft Q-values.
  3. **Value Function (V_ψ):** Often implicitly defined or learned separately (original SAC had a separate V-net).
  4. **Replay Buffer:** For off-policy learning.
  5. **Target Networks:** For stabilizing Q-function and Value function learning.

Examples



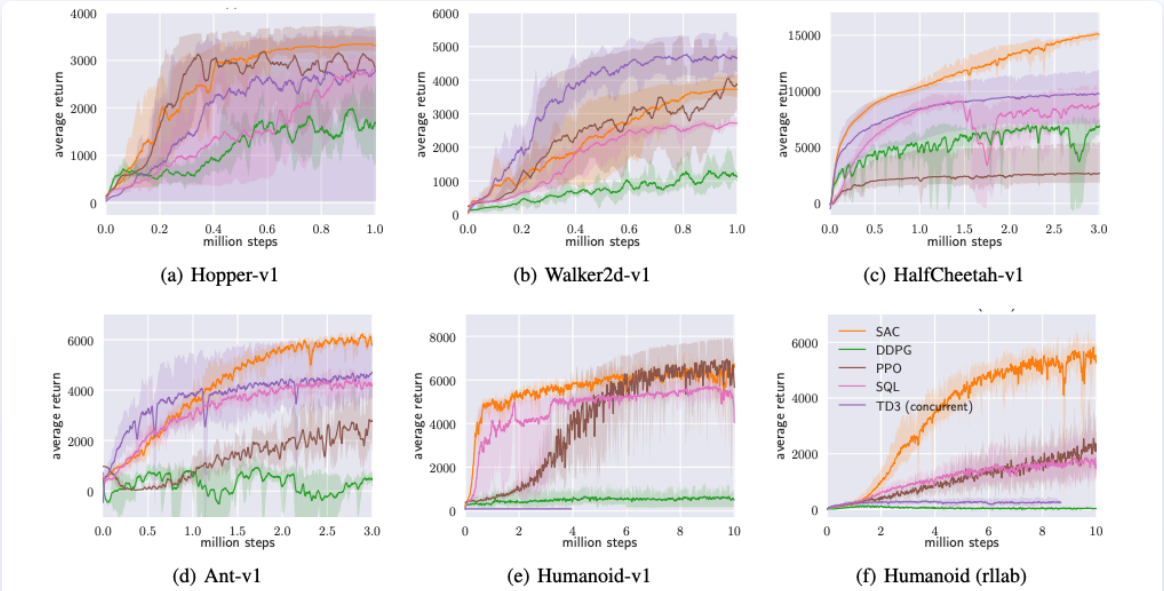


Figure 1. Training curves on continuous control benchmarks. Soft actor-critic (yellow) performs consistently across all tasks and outperforming both on-policy and off-policy methods in the most challenging tasks.

From Theory to Advanced Applications

- **How Probabilistic Foundations Enable Modern Advances:**
  - **Uncertainty-aware decision making** → Critical for real-world robustness
  - **Probabilistic policies** → Better exploration for complex environments
  - **Inference framework** → Natural extensions to preference learning, offline data
- **Modern Applications:**
  - Model-based RL benefits from explicit uncertainty in dynamics models
  - Multi-agent RL requires reasoning about other agents' uncertain behaviors

Reinforcement Learning from Human Feedback

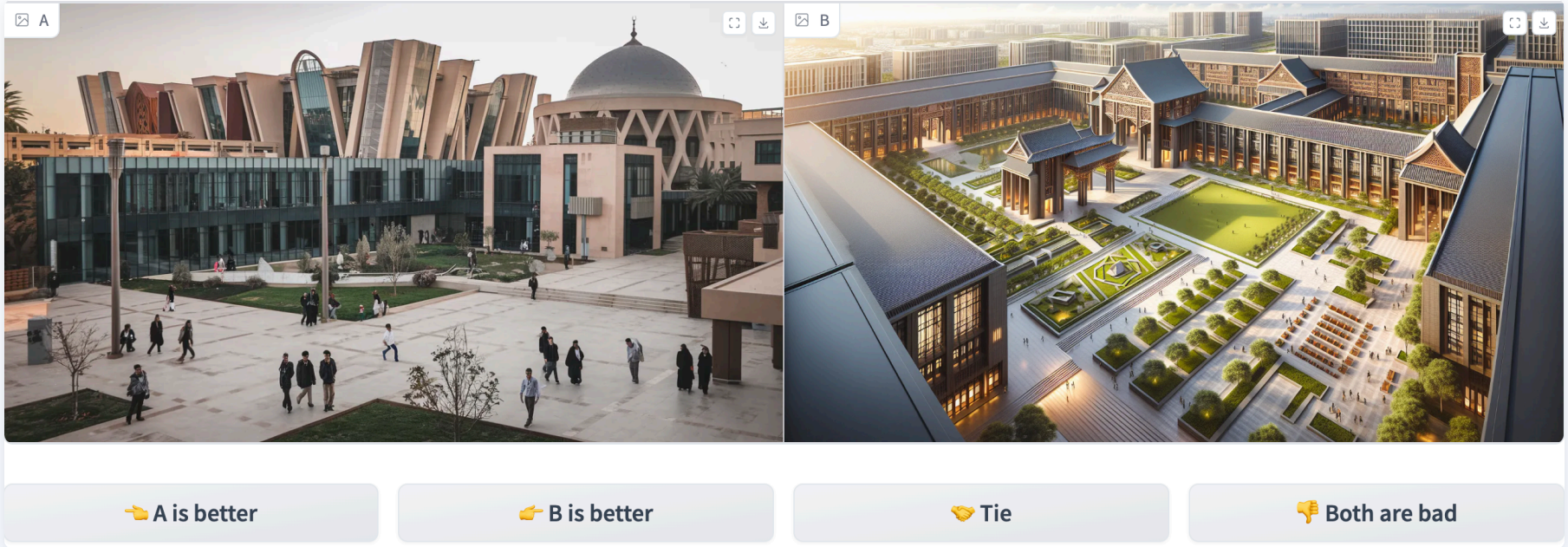
- A technique to align powerful models (especially LLMs) with human preferences.
- **Core Idea:** Use human feedback to train a reward model, then use RL to optimize the base model against this learned reward.
- **Typical RLHF Process for LLMs:**
  1. **Supervised Fine-Tuning (SFT):** Fine-tune a pre-trained LLM on a dataset of high-quality human-written demonstrations (e.g., desired responses to prompts).
  2. **Reward Modeling (RM):**
     - Collect human preference data: Present humans with multiple model outputs for the same prompt and ask them to rank them or choose the best.
     - Train a separate reward model to predict human preference scores given a prompt and a response. Input: (prompt, response), Output: scalar reward.
  3. **RL Optimization:** Use an RL algorithm (commonly PPO) to fine-tune the SFT model.

LLMs and Reinforcement Learning

- **Alignment:** RLHF is a primary method for aligning LLMs to be helpful, harmless, and honest.
- **Improving Capabilities / Agency:**
  - **Tool Use:** Training LLMs to use external tools (calculators, search engines, APIs) via RL by rewarding successful tool invocation and task completion.
  - **Interactive Agents:** Developing LLM-based agents that can plan and act in simulated or real environments (e.g., web navigation, game playing) using RL to learn policies.
  - **Dialogue Systems:** Optimizing dialogue flow, coherence, and user satisfaction using RL.
- But how to pick the right scores/rewards?

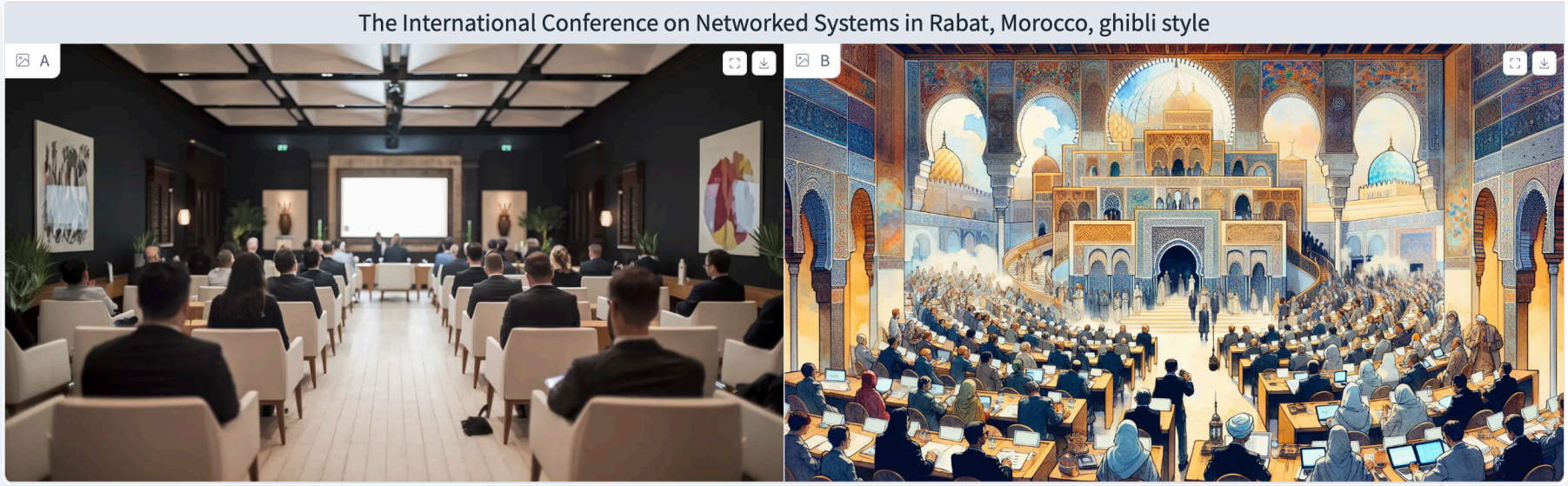
Preferences are more natural (1/2)

Prompt: **Mohammed VI Polytechnic University Rabat campus**



Preferences are more natural (2/2)

Prompt: **The International Conference on Networked Systems in Rabat, Morocco, ghibli style**

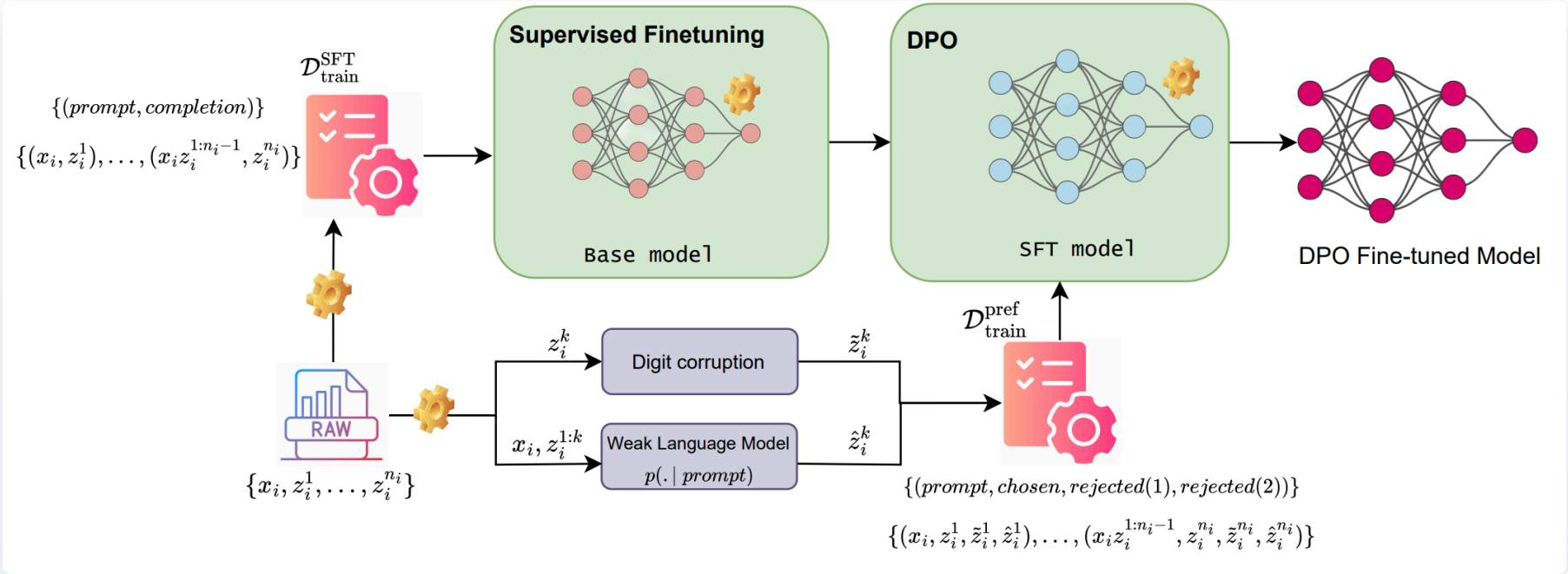


https://lmarena.ai/ (Chatbot Arena (formerly LMSYS): Free AI Chat to Compare & Test Best AI Chatbots)

Preference Optimization (e.g., DPO)

- **Direct Preference Optimization (DPO):** An alternative to the RM+RL pipeline in RLHF.
- **Core Idea:** Directly optimize the policy to satisfy human preferences, bypassing the explicit reward modeling step.
- Derives a loss function directly from preference pairs (y_w, y_l) (winner, loser responses for a prompt x).
- The DPO loss encourages the policy π_θ to assign higher probability to preferred responses y_w and lower probability to dispreferred responses y_l, relative to a reference policy π_{ref} (often the SFT model).
- $L_{DPO}(\pi_\theta; \pi_{ref}) = -E_{(x,y_w,y_l)\sim\mathcal{D}}[\log \sigma(\beta \log \frac{\pi_\theta(y_w|x)}{\pi_{ref}(y_w|x)} - \beta \log \frac{\pi_\theta(y_l|x)}{\pi_{ref}(y_l|x)})]$.
  - β is a temperature parameter.
- **Advantages:** Simpler than the multi-stage RLHF pipeline, can be more stable.

Preference Optimization on Reasoning Traces



PORT: Preference Optimization on Reasoning Traces - Salem Lahlou*, Abdalgader Abubaker*, Hakim Hacid, NAACL 2025

Examples and Results

prompt (GSM8K question + answer start)	chosen	rejected(1)	rejected(2)
Natalia sold clips to 48 of her friends in April, and then she sold half as many clips in May. How many clips did Natalia sell altogether in April and May? Natalia sold 48/2 = 24 clips in May.	Natalia sold 48+24 = 72 clips altogether in April and May.	Natalia sold 25+98 = 12 clips altogether in April and May.	Natalia sold 24 clips in April.

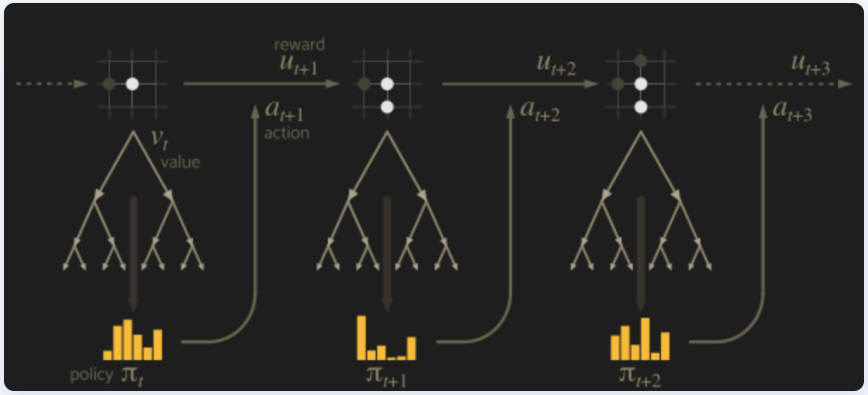
Model	GSM8K	AQuA	ARC	LastLetterConcat
Base model	54.66	31.50	76.11	16.67
SFT	55.43	30.71	75.60	17.34
DPO (ours)	**58.91**	**35.04**	76.02	**18.67** (+12%)

Model	GSM8K	AQuA	ARC
Base model	54.66	31.50	76.11
SFT on AQUA	54.89	31.50	75.68
DPO - digit corr.	**57.70**	**37.40** (+18.73%)	**76.88**
DPO - Llama-7B	55.57	33.86	76.71

Model-Based Reinforcement Learning

- **Core Idea:** Learn a model of the environment's dynamics $P(s'|s,a)$ and rewards $R(s,a,s')$, then use this learned model for planning or policy learning.
- **Contrast with Model-Free RL:** Model-free methods learn policies or value functions directly from experience without explicitly learning a model.

- **Advantages:**
  - **Sample Efficiency:** Can be much more sample efficient if the model is accurate, as simulated experience from the model is "cheaper" than real-world interaction.
  - **Interpretability:** The learned model can sometimes provide insights into the environment.
- **Disadvantages/Challenges:**
  - **Model Error:** If the learned model is inaccurate (model bias), the policy learned from it can be suboptimal or even catastrophic (compounding errors).
  - **Computational Cost:** Learning and using a model can be computationally intensive.

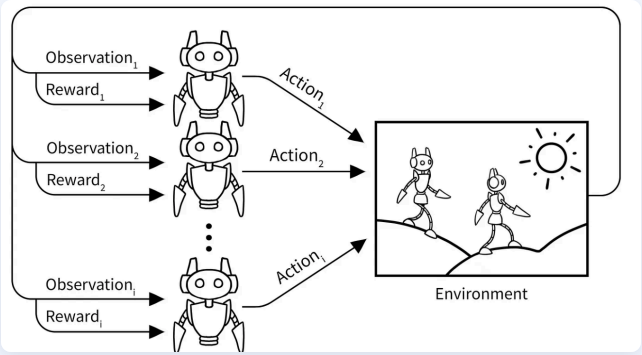


Offline Reinforcement Learning (Batch RL)

- **Core Idea:** Learning policies from a fixed, previously collected dataset of experience, without any further interaction with the environment.
- **Motivation:** Many real-world scenarios where active data collection is expensive, risky, or unethical (e.g., healthcare, robotics, autonomous driving from existing logs).
- **Main Challenge:**
  - **Counterfactual Queries:** Cannot query the environment for what *would have happened* if a different action was taken.
- **Key Ideas & Methods:**
  - **Policy Constraint / Regularization:** Constrain the learned policy to stay close to the behavior policy that generated the data.
  - **Importance Sampling (with care):** Can be used but suffers from high variance with large policy deviations.

Multi-Agent Reinforcement Learning (MARL)

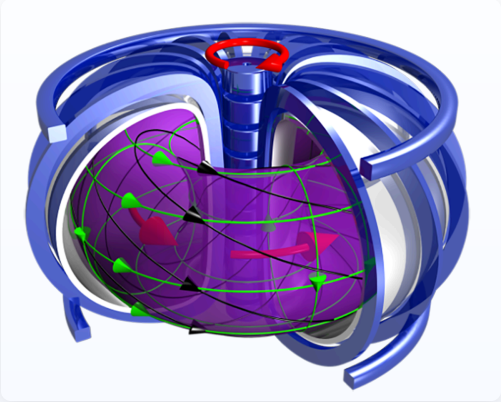
- **Core Idea:** Multiple agents learning simultaneously in a shared environment, where each agent's actions can affect others and the overall system dynamics.
- **Applications:** Robotics teams, autonomous vehicle coordination, game playing (e.g., StarCraft, Dota), economic modeling.
- **Main Challenges:**
  - **Non-Stationarity:** From any single agent's perspective, the environment is non-stationary because other agents are also learning and changing their policies.
  - **Credit Assignment:** Difficult to determine which agent contributed to a team reward or failure.
  - **Scalability:** The joint action space grows exponentially with the number of agents.
  - **Coordination vs. Competition:** Agents might need to cooperate, compete, or a mix of both.



RL in the Real World: Controlling Fusion Plasma

The Quest: Sustainable Fusion Energy

- **Tokamaks:** Toroidal devices that use powerful magnetic fields to confine and heat plasma to millions of degrees Celsius – hot enough for fusion to occur.
- **Goal:** A clean, virtually limitless energy source.



The Grand Challenge: Plasma Control

- **Extreme Conditions:** Plasma at >100 million °C must be precisely shaped and positioned, avoiding contact with vessel walls.
- **Instabilities:** Plasma is inherently unstable and can be lost in milliseconds if not perfectly controlled.
- **High-Dimensional Actuation:** Dozens of magnetic coils must be adjusted thousands of times per second.
- **Complexity:** Different plasma shapes (configurations) are needed for research and optimizing fusion performance. Designing controllers for each is a massive engineering effort with traditional methods.

Deep RL Tames the Tokamak

How RL Was Used (Degrave et al. (Deepmind), Nature 2022):

- **The Agent:** A deep reinforcement learning algorithm (an actor-critic model) learned to control all 19 magnetic coils of the Tokamak à Configuration Variable (TCV) in Switzerland.
- **Training:**
  - The RL agent was trained in a **high-fidelity simulator** that modeled plasma dynamics.
  - **Reward Function:** Defined by high-level objectives: achieving specific plasma shapes, current, position, and maintaining stability. Penalties for hitting constraints or undesired states.
  - The agent learned a **neural network** mapping sensor readings and target shapes to coil voltage commands.

Is This "Really Real World"? Absolutely.

- **Physical Deployment:** The learned controller was directly deployed on the **actual TCV tokamak** ("zero-shot" transfer from simulation).

Challenges and Open Problems in RL

- **Sample Efficiency:** Many algorithms still require vast amounts of data. *How can we learn effectively from fewer interactions?*
- **Exploration vs. Exploitation:** Efficiently exploring large state-action spaces remains hard. *Developing smarter exploration strategies is key.*
- **Generalization & Transfer Learning:** Policies learned for one task often don"t transfer well to even slightly different tasks or environments. *How to create more adaptable agents?*
- **Reward Design:** Crafting good reward functions is often difficult and crucial for success (reward engineering). *Can we learn rewards or use preferences more effectively?*
- **Safety and Robustness:** Ensuring learned policies are safe, reliable, and robust to adversarial perturbations or unexpected situations. *Critical for real-world deployment.*
- **Scalability:** Applying RL to very high-dimensional state/action spaces or long-horizon problems.
- **Interpretability:** Understanding why an RL agent makes certain decisions. *Opening the "black box".*
- **Credit Assignment:** Determining which actions in a long sequence were responsible for good/bad outcomes, especially with sparse rewards.

Research Platforms: Minigrid & BabyAI

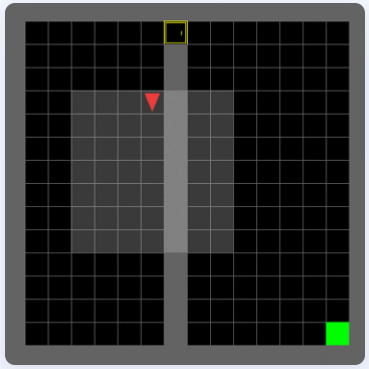
MiniGrid - Chevalier-Boisvert, ..., Lahlou, et al., Neurips 2023

- Lightweight & customizable grid-worlds.
- For general RL research:
  - Goal-oriented tasks.
  - Hierarchical missions.
- Tunable difficulty for curriculum learning.

BabyAI - Chevalier-Boisvert, ..., Lahlou, et al., ICLR 2018

- Built on Minigrid.
- Focus: **Grounded language learning**.
- Tasks via synthetic natural language instructions (e.g., "put the ball next to the box").
- Studies how agents connect language to actions & perception.

Flexible platforms for investigating complex RL challenges like planning, memory, and language understanding, studying exploration, sample efficiency, generalization, safety, etc..



Section 6: Conclusion & Future Directions

Summary: RL Through a Probabilistic Lens

- We've journeyed from fundamental ML concepts to core RL principles, emphasizing a probabilistic interpretation.
- Viewing RL as **probabilistic inference** (Control as Inference, MaxEnt RL like SAC) offers:
  - A unifying theoretical framework.
  - Principled ways to handle uncertainty and derive algorithms.
  - Enhanced exploration and robustness.
- This perspective underpins powerful algorithms like Soft Actor-Critic.

Future Directions

- **Deeper Integration of Probabilistic Models:** More sophisticated Bayesian methods, better uncertainty quantification in deep RL (especially distinguishing and using aleatoric/epistemic).
- **Causal RL:** Incorporating causal reasoning for better generalization, intervention understanding, and out-of-distribution robustness.
- **RL for Scientific Discovery:** Using RL to design experiments, discover materials, control complex systems (e.g., fusion reactors, chemical synthesis). This is at the core of my current research on **Generative Flow Networks**, which are *cousins* of RL.
- **Foundation Models for RL:** Leveraging large pre-trained models (like LLMs or Vision Transformers) for better representations, priors, or even as components of RL agents (e.g., for planning or reward specification).
- **Addressing Societal Impact:** Ensuring fairness, transparency, and safety as RL systems become more capable and deployed in the real world. Developing ethical guidelines and robust auditing mechanisms. Awareness of inherent biases (remember the generated photos about Morocco from earlier?).

Thank You!

Questions?

Contact: Salem Lahlou, https://la7.lu

References (1/2)

- https://www.freecodecamp.org/news/bayes-rule-explained/
- https://m0nads.wordpress.com/2022/01/22/muzero/
- https://medium.com/data-science/multi-agent-deep-reinforcement-learning-in-15-lines-of-code-using-pettingzoo-e0b963c0820b
- https://www.iaea.org/bulletin/
- https://unlocked.microsoft.com/laliga-beyond-stats/
- https://blog.biocomm.ai/2023/11/30/12696/

References (2/2)

- An introduction to apprenticeship learning by teaching an agent how to play a video game
- Reinforcement Learning and Control as Probabilistic Inference: Tutorial and Review
- Probabilistic Machine Learning, Advanced topics
- Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor
- Magnetic control of tokamak plasmas through deep reinforcement learning
- Asynchronous Methods for Deep Reinforcement Learning
- PORT: Preference Optimization on Reasoning Traces
- Minigrid & Miniworld: Modular & Customizable Reinforcement Learning Environments for Goal-Oriented Tasks
- BabyAI: A Platform to Study the Sample Efficiency of Grounded Language Learning
- DEUP: Direct Epistemic Uncertainty Prediction

Computational Cost Comparison

- **Q: How much more computationally expensive are probabilistic methods?**
- **A:** Typically 1.2-2x the computational cost of traditional methods:
  - SAC vs. DDPG: ~1.5x training time per iteration
  - Additional costs from:
    - Computing entropy/KL terms
    - Maintaining distribution parameters
    - Often requiring larger networks
  - However, may converge in fewer environment interactions, saving time overall

When to Use Traditional vs. Probabilistic RL

- **Q: When would standard RL be preferable to probabilistic approaches?**
- **A:** Standard RL might be better when:
  - Computational resources are severely constrained
  - The environment has very limited stochasticity
  - The task requires a deterministic policy by design
  - Exploration is easily guided by shaped rewards
  - The problem is simple enough that exploration issues are minimal

Scaling to High-Dimensional Problems

- **Q: How do these methods scale to very high-dimensional problems?**
- **A:** Scaling considerations:
  - Factorized distributions help manage dimensionality (e.g., diagonal covariance Gaussians)
  - Function approximation (neural networks) handles high-dimensional state spaces
  - For extremely high-dimensional action spaces:
    - Auto-regressive policies can help
    - Hierarchical approaches decompose the problem
    - Latent variable models provide dimensionality reduction

Software and Implementation Resources

- **Q: What software libraries/frameworks support probabilistic RL well?**
- **A:** Recommended resources:
  - **General RL libraries with MaxEnt implementations:**
    - Stable Baselines3, RLlib, torchrl
  - **Probabilistic programming:**
    - TensorFlow Probability, Pyro, PyMC

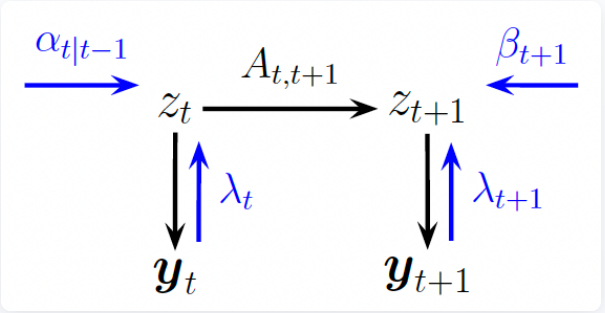
Justifying the Exponential Optimality Score

- **Why This Choice Works So Well:**
  - This specific definition is powerful because of its direct impact on trajectory probabilities:
    - The individual scores, $\exp(r(s_t, a_t)/\alpha)$, for each step in a trajectory multiply together.
    - This means the overall score (or unnormalized probability) for an entire trajectory becomes proportional to $\exp\left(\sum_t \frac{r(s_t,a_t)}{\alpha}\right)$.
  - **Crucially, this links directly to the goal of RL: trajectories with higher cumulative (summed) rewards are exponentially more likely to be considered optimal.**
- **Underlying Principle:**
  - This effective exponential form is also consistent with the **Principle of Maximum Entropy**.
  - This principle suggests that if you want to translate rewards into scores in the most "unbiased" way (specifically, by wanting the log of the score to be proportional to the reward), the exponential form naturally emerges (Boltzmann/Gibbs distribution).

Aside: Example of Message Passing - HMM Chain

Consider variables:

- z_t: Hidden state at time t
- y_t: Observation at time t



Forward-Backward Message Passing:

- Forward messages: $\alpha_t(j) = P(z_t = j | y_{1:t})$
  - Belief about current state given past observations
  - Computed recursively: $\alpha_t(j) \propto \sum_i \alpha_{t-1}(i) \cdot p(z_t = j | z_{t-1} = i) \cdot p(y_t | z_t = j)$
- Backward messages: $\beta_t(j) = P(y_{t+1:T} | z_t = j)$
  - Likelihood of future observations given current state
  - Computed recursively: $\beta_{t-1}(i) = \sum_j \beta_t(j) \cdot p(y_t | z_t = j) \cdot p(z_t = j | z_{t-1} = i)$
- Combining messages: $P(z_t = j | y_{1:T}) \propto \alpha_t(j) \cdot \beta_t(j)$

In our RL context: backward messages represent expected future optimality/rewards.

Stochastic Dynamics & Variational Inference

- Exact inference is often intractable for stochastic dynamics and continuous spaces.
- **Variational Inference:** Approximate the true posterior $P(\tau|O_{1:T}=1)$ with a simpler, tractable distribution $q(\tau;\phi)$ parameterized by ϕ.
  - Typically, $q(\tau;\phi) = P(s_1)\prod_t q(a_t|s_t;\phi)P(s_{t+1}|s_t,a_t)$. Here $q(a_t|s_t;\phi)$ is our learned policy.
- **Objective:** Minimize $KL(q(\tau;\phi)||P(\tau|O_{1:T}=1))$.
  - Equivalent to maximizing the **Evidence Lower Bound (ELBO)**
- More details in "Reinforcement Learning and Control as Probabilistic Inference: Tutorial and Review" by Sergey Levine