# State-of-the-art reinforcement learning algorithms

# What algorithms are currently used?

- Some of the algorithms we discussed are of historical importance:
  - For instance, you probably don't want to use vanilla policy gradient.
  - They had been superseeded by algorithms that add tweaks, and they are specialized for particular setups - eg. continuous or discrete actions, continuous or discrete states etc.
- These algorithms still fall in the classes we discussed - Q-learning, policy gradient, actor-critic etc.
- Small optimizations in the implementation might make a significant difference in performance
  - E.g. things like memory access patterns
  - Very likely, you don't want to implement them yourself, unless your objective is to improve on them

# Soft Actor-Critic (SAC)

# Motivation

- Traditional actor–critic algorithms maximize **expected return**
- But they can suffer from:
  - Poor exploration
  - Instability in training
- SAC introduces **entropy maximization**
  - Encourages exploration
  - Stabilizes learning

# Entropy in behavior

- You have two algorithms with the same performance but:
  - One does the same thing all the time - it is predictable
  - The other one does random things from time to time (it has *entropy*)
- There are good reasons to prefer the predictable algorithm at *deployment time*
- But in the middle of learning, the algorithm with the entropy has advantages
  - For instance, it takes care of exploration
  - And because it also has performance, it presumably also does an appropriate amount of exploitation as well.

# Soft-actor critic core idea

Maximize both:

1. **Expected reward**
2. **Policy entropy** (randomness in actions)

Objective:

$$J(\pi) = \sum_t \mathbb{E}_{(s_t,a_t)\sim\pi}\left[r(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))\right]$$

- $\alpha$: temperature parameter (trade-off between reward & entropy).

## Key Components

- **Actor (Policy $\pi_\theta$)**: stochastic policy outputs actions.
  - $\pi_\theta(s, a)$ is the probability of choosing action $a$ when in state $s$
  - So the policy is a probability distribution
- **Critic ( $Q_\psi$)**: estimates action-value function.
- **Entropy regularization**: improves exploration.
- **Automatic temperature tuning**: adaptively balances reward vs entropy.

## Update Rules

1. **Critic Update** (soft Bellman backup):

$$Q(s,a) \leftarrow r + \gamma \, \mathbb{E}_{s',a'} \left[ Q(s',a') - \alpha \log \pi(a'|s') \right]$$

2. **Actor Update** (minimizing KL divergence):

$$\nabla_\theta J_\pi(\theta) = \mathbb{E}_{s \sim D, a \sim \pi_\theta} \left[ \alpha \log \pi_\theta(a|s) - Q(s,a) \right]$$

What this thing does is minimizes the Kullback-Leibler divergence between the 1) the policy and 2) a target distribution that assigns higher probability to actions with higher Q-values (expected returns).

By minimizing the KL divergence, the policy learns to imitate this Boltzmann distribution — i.e., to sample actions proportional to their soft value.

This encourages both high-reward and high-entropy behavior.

## Algorithm Steps

1. Initialize replay buffer, actor, critic, and temperature $\alpha$
2. For each step:
    - Sample action from policy $\pi_\theta$
    - Store $(s, a, r, s')$ in replay buffer.
    - Update $Q$-functions using soft Bellman backup.
    - Update policy $\pi_\theta$ to maximize entropy-regularized objective
    - Adjust temperature $\alpha$ automatically.

## Advantages

- Better exploration via entropy maximization
- Sample-efficient (uses off-policy replay buffer)
- Stable training (double critics, entropy regularization)
- Works well in continuous action spaces

## Applications

- Robotics (manipulation, locomotion)
- Continuous control tasks (Mujoco, PyBullet)
- Real-world autonomous systems

## Summary

- **SAC = Actor–Critic + Maximum Entropy RL**
- Learns policies that are both **reward-maximizing** and **stochastic**
- Off-policy, sample-efficient, and stable in practice.

# Proximal Policy Optimization (PPO)

## Motivation

- Policy gradient methods (like REINFORCE, A2C) are:
  - High variance
  - Unstable with large updates
- There were previous algorithms that aimed to improve stability, such as Trust Region Policy Optimization (TRPO)
  - but they were is complex and expensive.
- **PPO**: uses a very simple and cheap trick to improve the stability of training.

## Core Idea

- Prevent policy updates from being too large.
- Use a **clipped surrogate objective** to constrain policy changes.
- What does it mean to clip? Let us say that you are clipping to the $[-0.5, 0.5]$ range:
  - your update is +0.1, return +0.1
  - your update is -0.2, return -0.2
  - your update is +17.5 return +0.5
  - your update is -3067.4 return -0.5

## Clipped Objective

Define probability ratio:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

Clipped surrogate loss:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\Big[\min\left(r_t(\theta)A_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t\right)\Big]$$

- $A_t$ - advantage estimate
- $\epsilon$ - small hyperparameter (e.g. 0.1–0.2)

## Training Procedure

1. Collect rollouts using current policy.
2. Estimate advantages

- Similarly to other policy gradient algorithms
- One good technique is GAE: generalized advantage estimation

3. Optimize clipped loss for several epochs on minibatches.
4. Update value function (critic) and policy (actor).
5. Repeat.

## Advantages

- Stable and reliable performance
- Easy to implement
- Works with high-dimensional continuous actions
- Strong results across many benchmarks (Atari, Mujoco)

## Applications

- Robotics and control (locomotion, manipulation)
- Games (Atari, Go, etc.)
- Sequential decision making in healthcare, finance, energy systems etc.
- Reinforcement learning from human feedback, in training LLMs

## Summary

- **PPO** = policy gradient + clipping for stability
- Balances performance and simplicity
- One of the most widely used RL algorithms today

# Reinforcement Learning with Human Feedback (RLHF)

## Motivation

- Traditional RL requires a **reward function**, but many tasks lack a clear one.
- Human preferences can provide guidance:
  - What outputs are more helpful, safe, or aligned?
- **RLHF** uses human feedback as a training signal.
  - It would be of course easy to just ask the human for the reward.
  - But it is not that simple: humans are very bad at assigning numerical values.

## Core Idea

1. **Pretrain** a model (e.g., language model) on large datasets.
2. **Collect human feedback**:
    - Show multiple outputs for the same prompt.
    - Ask humans which is preferred.
3. **Train a reward model** on this preference data.
4. **Fine-tune the model** with RL to maximize the learned reward.

# RLHF Pipeline

1. **Supervised Fine-Tuning (SFT):**
    - Start with pretrained model.
    - Fine-tune on curated, high-quality examples.
2. **Reward Model (RM):**
    - Learn to predict human preferences.
    - Takes (prompt, output) as input → scalar reward.
3. **RL Optimization (PPO often used):**
    - Fine-tune the model to maximize RM score.
    - Use a KL penalty to keep outputs close to pretrained model.

## Reward Model Training

- Human annotators label which response is **better**
- Reward model trained with pairwise ranking loss:

$$\mathcal{L} = -\log \sigma(R(x, y^+) - R(x, y^-))$$

where $y^+$ is preferred, $y^-$ is not and $\sigma(\cdot)$ is the **logistic sigmoid** function:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

## Policy Optimization (PPO step)

- Policy = language model being trained.
- Objective combines:
  - Reward model score
  - KL penalty against difference from initial policy
- Keeps the model aligned but avoids drifting too far.

## Advantages

- Aligns AI behavior with human preferences
- Works when explicit reward functions are hard to define
- Improves helpfulness, safety, and user satisfaction

## Applications

- Large Language Models (e.g., ChatGPT)
- Conversational AI and assistants
- Content moderation / safe generation
- Robotics tasks with human-in-the-loop training

## Summary

- **RLHF = Pretraining + Human Feedback + RL Fine-Tuning**
- Replaces hand-designed rewards with learned human preference signals.
- Widely used to align large AI models with human values.