

ECE433/COS435 Spring 2024

Introduction to RL

Lecture 3: Formulating the RL Problem (1/2)

1. From Bandit to RL









1.1. Recap: Multi-Arm Bandit

1.2. Contextual Bandit Problems

Contextual bandits extend the basic multi-arm bandit problem by introducing "context" to model the environment's state. In the contextual bandit setting, the decision-making agent faces a situation, described by a context vector \mathbf{c} , where it must choose an action \mathbf{a} from a set of possible actions at each time step, and these actions are associated with rewards r . The challenge here is that the agent has access to certain context or features before making a decision, which influences the expected reward. This problem is a generalization of the multi-armed bandit problem, introducing context (features) to model the environment state.

Example: Product Recommendation

Consider an online advertising scenario where the context is user features (for examples, age, interests, browsing history), and actions are different advertisements to recommend. The reward is whether the user clicks on the ad (1 for click, 0 for no click). The contextual bandit algorithm aims to learn the best ad to show based on the user context to maximize the total number of clicks over time.

Previously liked products <i>State</i>	Recommendation <i>Action</i>	Clicked ? <i>Reward</i>
		
		Click !
		Click !
		

Formulation

Let $A = \{a_1, \dots, a_k\}$ be the set of k actions (or arms) available to the agent. At each time step t , the agent observes a context $c_t \in \mathbb{R}^d$, where d is the dimensionality of the context space. Based on the observed context c_t , the agent selects an action a_t from A . After taking action a_t , the agent receives a reward r_t , which is a random variable with an unknown distribution dependent on both the action taken a_t and the context c_t :

$$r_t \sim P(\cdot | a_t, c_t),$$

We highlight that the distribution of reward conditioned on action and context is unknown.

We assume c_1, c_2, \dots, c_t are independent from each other. As a result, any action chosen or realized reward wouldn't affect future trajectory beyond the current state. This is a key difference between bandit and RL.

The goal of the agent is to maximize the cumulative reward over time. This is often expressed as maximizing the expected reward: $\max \mathbb{E} \left[\sum_{t=1}^T r_t \right]$ where T is the time horizon.

In these problems, the reward distribution is conditioned on the current context, denoted as c_t . Contexts at different time steps are independent from one another.

Introducing contexts to bandit learning makes the decision process more dynamic and reflective of real-world scenarios.

1.3. Upper Confidence Bound (UCB) Methods for Contextual Bandit

Similar to the case of MAB, one can apply the ideas of UCB to contextual bandit, which is to maximizing a sum between reward estimate and exploration bonus:

$$a_t = \arg \max_a \left(\hat{R}_t(a, c_t) + \text{exploration bonus} \right) \quad (1)$$

where \hat{R}_t is a reward prediction function trained by supervised learning based on observations so far $(c_k, a_k, r_k), k \in [t-1]$:

$$\hat{R}_t \leftarrow \operatorname{argmin}_f \sum_{k=1}^{t-1} (r_k - f(a_k, c_k))^2$$

and it kept getting updated as a new sample triplet (a_t, c_t, r_t) is observed.

1.4. Transition to Reinforcement Learning

Contextual bandits are a middle ground between the classic multi-armed bandit problem and the more complex Markov decision processes (MDPs). They are particularly suitable for situations where the context changes dynamically in the environment but **any action only affects the immediate reward distribution and wouldn't affect future dynamics**.

This paradigm is highly relevant in personalized recommendations, online advertising, and medical treatment design, where the algorithm must balance exploration (gathering more information about the environment) and exploitation (making the best decision based on current information).

While bandit problems provide a fundamental understanding of decision-making under uncertainty, reinforcement learning (RL) extends these concepts to more complex scenarios involving sequences of decisions and state transitions.

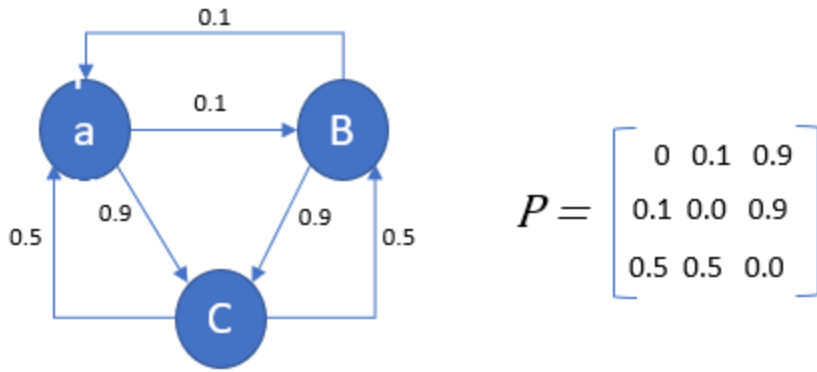
Key differences in RL include:

- **State Transitions:** Actions lead to changes in the environment, influencing future state trajectories and future decisions.
- **Delayed Rewards:** Rewards of an action may not be immediate. One may need to make a sequence of correct moves to receive the final rewards. Decision making with delayed rewards requiring long-term planning.

In the following sections, we will delve deeper into the formulation of reinforcement learning problems, exploring how agents learn to make a sequence of decisions to maximize cumulative rewards in a dynamic environment.

2. Review of Markov Chains

Understanding Markov Chains is crucial for learning Reinforcement Learning (RL). The framework that underlies most of RL is the Markov Decision Process (MDP), a model for decision-making where outcomes are partly random and partly under the control of the decision-maker. Markov Chains are integral to this because they provide a basic mathematical formulation to model the random transitions between different states in an environment.



2.1. Definition of Markov Chains

A Markov chain is a stochastic process that undergoes transitions from one state to another within a finite set of states. It is a fundamental concept in probability theory and forms the basis for understanding Markov Decision Processes (MDPs) in Reinforcement Learning.

In this lecture, let's consider Markov chain with discrete time steps and finite discrete state space. Markov chains can be more generally defined to be over continuous state space over continuous time, but in this course we focus on the discrete case.

Let $S = \{s_1, s_2, \dots, s_n\}$ be the finite set of states, and let the process moves from one state to another at discrete time intervals.

The most critical property of a state in a Markov chain is as follows:

- **Markov Property:** The future state depends only on the current state and not on the sequence of events that preceded it. Mathematically, this is expressed as:

$$P(s_{t+1}|s_t) = P(s_{t+1}|s_1, s_2, \dots, s_t). \quad (2)$$

Here s_t captures all in the information from the history that is relevant for future state trajectory of the Markov chain. The Markov property is often referred to as the "**memoryless property**" because it says that the process would forget all historical information except for the most recent state.

Next we review some Markov chain basics

- **Probability Transition Matrix:**

The dynamics of a Markov chain is governed by its transition matrix P , which contains the probabilities of moving from one state to another. If there are n states, P is an $n \times n$ matrix where each element

$$P_{ij} = \mathbb{P}(s_{t+1} = j | s_t = i)$$

represents the probability of transitioning from state i to state j .

- **Multi-Step Probability Transition:**

$$(P^k)_{ij} = \mathbb{P}(s_{t+k} = j | s_t = i)$$

- **One-Step Conditional Reward Expectation:** Let $r(s)$ be reward of state s . The one-step conditional reward is

$$\mathbb{E}[r(s_{t+1}) | s_t = i] = (Pr)_i,$$

where the matrix-vector product form we denote by $r = [r(s_1), \dots, r(s_n)]^T$.

- **Multi-Step Conditional Reward Expectation:** Let $r(s)$ be reward of state s . The one-step conditional reward is

$$\mathbb{E}[r(s_{t+k})|s_t = i] = (P^k r)_i,$$

where the matrix-vector product form we denote by $r = [r(s_1), \dots, r(s_n)]^T$.

- **Cumulative Visitation Distribution:**

Let the initial state distribution be μ , where μ_i is the probability of starting in state s_i . The state visitation measure $\eta(s)$ for a state s is defined as:

$$\eta(s) = \sum_{t=0}^T \mathbb{P}(s_t = s)$$

Here, s_t denotes the state at time t , and $\mathbb{P}(s_t = s)$ is the probability that the Markov Chain is in state s at time t .

In matrix form, for each state s , the state visitation measure over T steps can be calculated as:

$$\eta(s) = \sum_{t=0}^T \mu^T P^t \mathbf{1}_s$$

where $\mathbf{1}_s$ is the one-hot vector with a 1 in the position corresponding to state s and 0s elsewhere.

- **Chapman-Kolmogorov Equation:** For state transition probabilities over multiple steps.

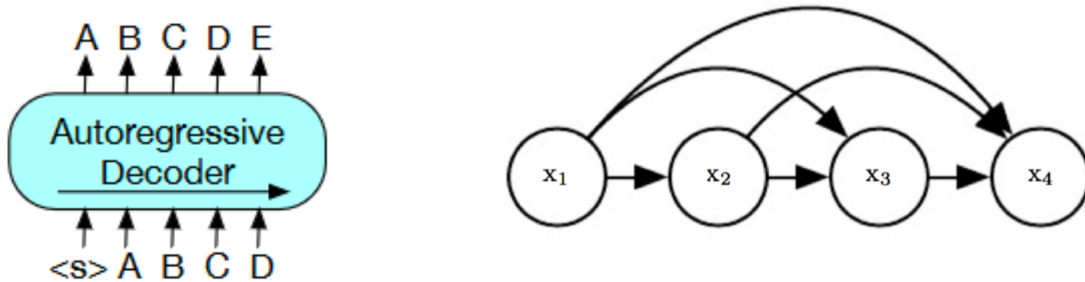
$$\mathbb{P}(s_{t+n+m} = j | s_t = m) = \sum_k \mathbb{P}(s_{t+n} = k | s_t = m) \cdot \mathbb{P}(s_{t+n+m} = j | s_{t+n} = k)$$

or equivalently

$$P_{ij}^{(n+m)} = \sum_k P_{ik}^{(n)} P_{kj}^{(m)}$$

The Chapman-Kolmogorov Equation provides a way to relate the probabilities of a system's state at different times. In its essence, the equation bridges the probability of transitioning from one state to another over different time intervals.

2.2. Example: Autoregressive decoder for text generation



Autoregressive decoders, such as those used in GPT (Generative Pretrained Transformer) models, generate text by predicting each token (like a word) in a sequence based on the tokens that precede it. This process can be modeled as a Markov chain, given that the probability of each token depends on a sequence of previous tokens, rather than just the immediately preceding one.

Discussion:

- Let x_i denote the i -th token to be generated. Can we model x_i as the state and define the transition probability as $P(x_{i+1} | x_i)$, i.e., the probability of token x_{i+1} following token x_i ?
- The above definition of state is not right. Each token is generated conditioned on previous tokens in the context window, not only the most recent way.

To correctly map the autoregressive text generation process to a Markov chain, we need to define state to capture all information from the history that is used for prediction of future tokens.

- State: $s_i = (x_{i-k+1}, x_{i-k+2}, \dots, x_i)$, where k is the size of context window.
- Transition Probability equals the probability of generating token x_{i+1} given the preceding sequence of k tokens:
$$P(s_{i+1} | s_i) = P(x_{i+1} | x_{i-k+1}, x_{i-k+2}, \dots, x_i).$$
- Autoregressive model maps the input sequence of tokens into an embedding vector, which is supposed to capture all information in the generated tokens for prediction of future token. One can also define state to be this embedding vector.

3. Formulation of Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning where an agent learns to make decisions by performing actions in an environment to achieve a goal. The learning process involves the agent interacting with the environment, receiving feedback in the form of rewards, and adapting its actions accordingly.

RL can be viewed as a complex combination of contextual bandit and Markov chains, where the state of environment changes dynamically under the influence of the agent's sequential decisions.

3.1. Basic Concepts

The key elements of reinforcement learning include:

- **State**

A state variable is one of the set of variables that are used to describe the mathematical "state" of a dynamical system. Intuitively, the state of a system describes enough about the system to determine its future behaviour in the absence of any external forces affecting the system.

- **Action:** Choices available to the agent, represented as a . The set of actions available may differ from state to state. In continuous control problem, the actions may be continuous or vector-valued.
- **Reward:** A scalar feedback signal, r , indicating the immediate effect of the agent's action at a current state. Rewards can be random variable with unknown distribution $P(r|s, a)$.
- **Policy:** A strategy or rule followed by the agent, denoted as $\pi : S \mapsto A$, mapping states to actions. More generally, we often allow randomized policies, where π maps any state to a distribution over available actions.
- **Environment:** Agents interact with the environment dynamically: drawing full observations of states from environment, implementing the chosen action in the environment, then the state of environment evolves according to some transition law.
- **Agent:** The learner or decision-maker. The agent can be viewed as an algorithm, interacting with the environment, processing all history of observations and prescribing real-time decisions. Note that agent \neq policy.

3.2. Markov Decision Process

Markov decision processes formally describe an environment for reinforcement learning. Where the environment is **fully observable**, i.e. The current state completely characterises the process.

The RL problem can be formulated as a Markov Decision Process (MDP) with components (S, A, P, R, γ) , where:

- S is a finite set of states.
- A is a finite set of actions.
- P is the state transition probability matrix, $P(s_{t+1}|s_t, a_t)$.
- r is the reward function, $r(s_t, a_t)$.
- T is the horizon, i.e., total number of time steps of the game.
- γ is the discount factor, representing the difference in importance between future and present rewards.

When the agent follows a fixed stationary policy, the Markov decision process reduces to a Markov chain with transition probability

$$P^\pi(s'|s) = P(s'|s, \pi(a))$$

where P^π is the transition probability matrix of the MDP following a fixed policy π .

The agent's objective in an MDP is to learn a policy π^* that maximizes the cumulative reward, often expressed as the expected return. Suppose there is a finite horizon T and the MDP would always terminate at time step T , the policy optimization problem can be expressed as:

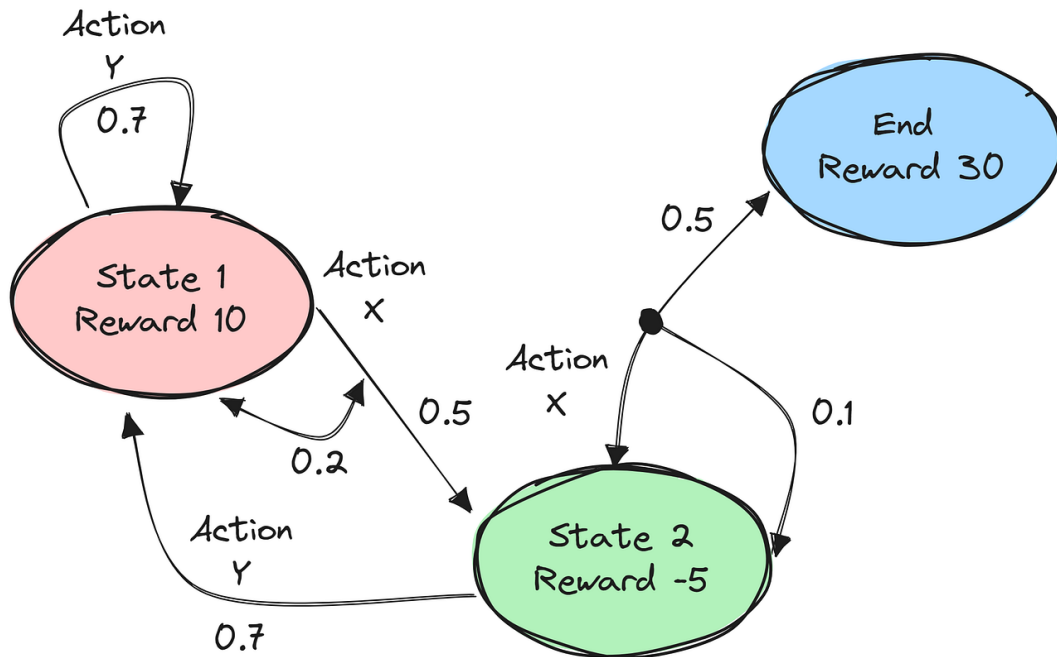
$$\max_{\pi} \mathbb{E}^\pi \left[\sum_{t=0}^T r(s_t, a_t) \right]. \quad (3)$$

Here \mathbb{E}^π means taking expectation over trajectories of the MDP following a fixed policy's transition law P^π .

If $T = \infty$, it doesn't make sense to study/compare infinite values of sum of rewards. In practice, one often pick a **discount factor** $\gamma \in (0, 1)$ and consider the following discounted sum of expected rewards:

$$\max_{\pi} \mathbb{E}^\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]. \quad (4)$$

This discount factor models the decay of future value.



Almost all RL problems can be formalized as MDPs, e.g.

- Classical optimal control primarily deals with continuous MDPs (often physical systems).
- Partially observable problems (we will discuss this topic later) can be converted into MDPs.
- Multi-Arm Bandits are MDPs with one state.
- Contextual bandit are MDPs with independent states.

4. Examples of Reinforcement Learning

4.1. Tetris

Tetris is a popular puzzle video game originally designed and programmed by Soviet software engineer Alexey Pajitnov. It has become one of the earliest prominent testbeds for RL. The game involves manipulating tetrominoes, geometric shapes composed of four square blocks each, as they fall into a playing field. The agent interacts with a grid by placing falling tetrominoes of random shapes. The objective is to clear lines without filling the grid to the top.

Mapping Tetris to a MDP:

States (s):

The current configuration of the grid, the falling piece, and the next upcoming piece.

Actions (a):

$a \in \text{rotate, move left, move right, drop}$

Reward

Rewards (r): Points for each line cleared, penalties for undesirable outcomes.

$$R(s, a) = 1 \quad \text{if a line is cleared}$$
$$R(s, a) = 0 \quad \text{otherwise}$$

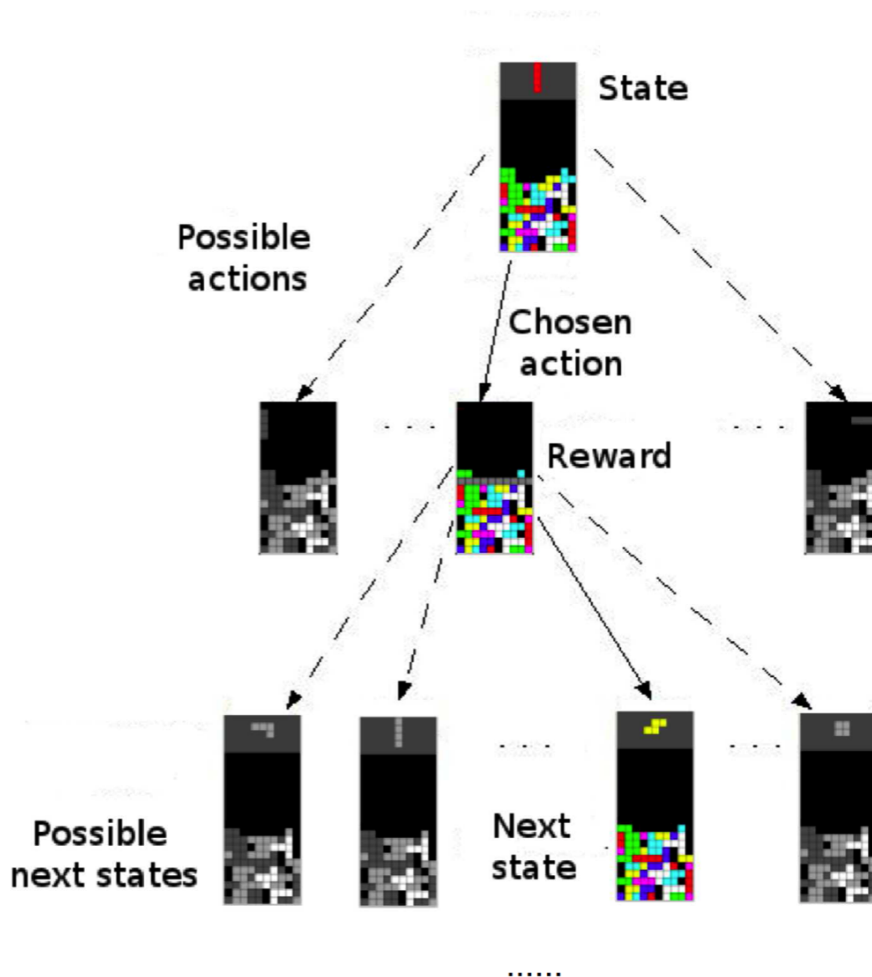
State Transitions:

After an action is executed, the current piece settles, and a new random piece appears.

- For example if the action is to drop the piece and the current state involves a specific grid configuration, the new state s' will reflect the updated grid after the piece has settled and a new piece appears at the top.
- If the height reaches the maximal height+1, the game terminates at an absorbing state.
- The transition is inherently random due to random shape of the new piece.

Let P be the transition probability matrix that documents every (state-action)-to-state transition probability $P(s'|s, a)$.

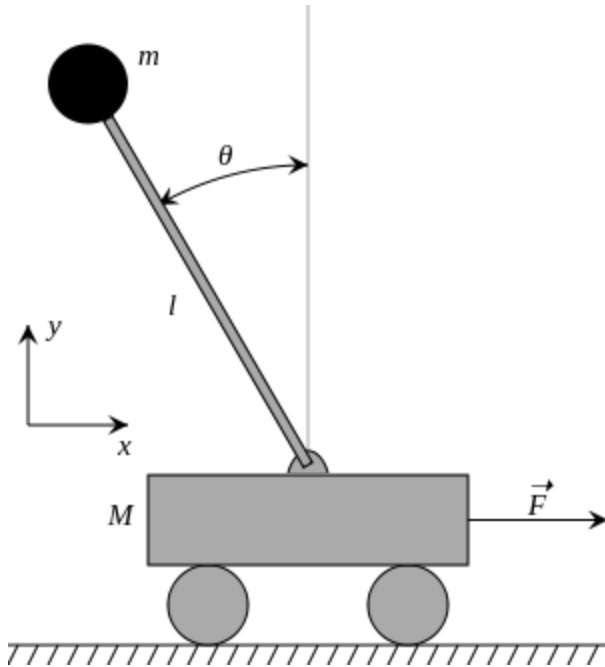
- How big is the matrix P ?
- Is P dense or sparse?



4.2. Inverted Pendulum: Cart-Pole System

The inverted pendulum problem, often exemplified by the cart-pole system, is a classic problem in control theory and reinforcement learning. The Cart-Pole consists of a cart that moves along the horizontal axis and a pole that is anchored on the cart. At every time step, you can observe its position (x), velocity (\dot{x}), angle (θ), and angular velocity ($\dot{\theta}$). These are the observable states. The goal is to apply forces to the cart to keep the pendulum upright.

At any state, the cart only has two possible actions: move to the left or move to the right.



Motion Dynamics of Cart-Pole (Optional)

The dynamics of the cart-pole system can be described by a set of nonlinear differential equations. For simplicity, classical control methods would consider a linear approximation around the unstable equilibrium point (where the pendulum is upright).

Let:

- x be the position of the cart.
- θ be the angle of the pendulum.
- u be the force applied to the cart.
- m be the mass of the pendulum.
- M be the mass of the cart.
- l be the length to the pendulum's center of mass.
- g be the acceleration due to gravity.

The full system dynamics of CartPole is nonlinear, but in robotics/control it is typical to consider linearized approximation to the true dynamics. In the noiseless setting, the linearized equations of motion for the cart-pole system are:

$$\ddot{x} = \frac{1}{M+m} \left(u + ml\dot{\theta}^2 \sin \theta - mg \cos \theta \sin \theta \right)$$

$$\ddot{\theta} = \frac{g \sin \theta - \cos \theta \left(\frac{1}{M+m} \left(u + ml\dot{\theta}^2 \sin \theta \right) \right)}{l \left(\frac{4}{3} - \frac{m \cos^2 \theta}{M+m} \right)}$$

Note: You don't have to know how to derive this. In fact, the purpose of RL is to learn how to control a system without having to know its dynamics equation!)

Mapping CartPole to MDP

The problem is framed as an MDP with the state space, action space, transition probability, and reward function based on the system's dynamics and control objectives.

State

The state of the system at any time t is represented by a vector \mathbf{s}_t that includes:

- x_t : Position of the cart on the track.
- \dot{x}_t : Velocity of the cart.
- θ_t : Angle of the pendulum from the vertical position.
- $\dot{\theta}_t$: Angular velocity of the pendulum.

State-transition

$$\mathbf{s}_{t+1} = F(\mathbf{s}_t, \mathbf{u}_t)$$

where F describes the motion dynamics and \mathbf{u}_t describes the additional force

Action

The actions represent the controls applied to the system.

- In the classical control problem of inverted pendulum, the action \mathbf{a}_t at each time step is a scalar u_t , ie., the force applied to the cart.
- In the OpenAI Gym, the actions were simplified to $\{left, right\}$

Rewards

The reward function is designed to achieve the system's goal, which typically includes:

- A positive reward for each time step the pendulum remains upright.
- A negative reward if the pendulum falls or the cart strays far from the center.
- A penalty for excessive use of force, encouraging energy efficiency.

Objective

The goal is to apply forces \mathbf{u} to keep the pendulum upright ($\theta \approx 0$) and the cart near the track center ($x \approx 0$).

4.3. Youtube recommendation

The YouTube video recommendation system itself uses a combination of user activity data and content data to generate personalized recommendations. User activities are categorized into explicit actions (like rating or liking a video) and implicit actions (like watching a video). The recommendation process also involves ranking candidate videos using various signals that judge the video's quality, the user's unique preferences, and the need for diversity in recommendations

State:

- User's watch history (recently watched videos).
- User's interaction history (likes, dislikes, shares, comments).
- Time spent on different types of videos.
- Demographic information of the user (if available).
- Current session information (duration, number of videos watched).
- Time of the day
- Trending news/memes

Action

- Recommending a specific video.
- Changing the order of video recommendations.
- Selecting videos for the homepage or the "Up next" section.

Reward

- Positive reward for increased user engagement (e.g., longer watch times, likes, shares).
- Negative reward for user disengagement (e.g., skipping videos, closing the app shortly after a recommendation).
- Feedback from user interactions like likes/dislikes on the recommended videos.
- Ethical and Bias Considerations: Ensuring the recommendation system is fair and does not propagate harmful content or biases.
- Long-term user engagement

Takeaways:

There are many ways of mapping real world problems to MDPs. One has to balance between the complexity of the MDP model and how accurately it models the real world. This is hard, part of the problem. Most of the class is going to assume that someone has already done the hard work of defining the MDP. But, when you go out into the real world, this may not be the case. Be forewarned.