Reinforcement Learning

Lara J. Martin (she/they)

https://laramartin.net/neurosymbolic-text-gen/

Modified from slides by Cassandra Kent, Yejin Choi, Bill Yuchen Lin, & Valentina Pyatkin

Learning Objectives

Define a Markov Decision Process

Distinguish between model-based and model-free RL

Compare and contrast the effects of exploration and exploitation

Extend Q-tables to real-world scenarios

Determine how RL is applied to language modeling

Reinforcement Learning

Learning by interacting with an environment

An algorithm for approximating a solution to a Markov Decision Process (MDP)

- “Stochastic sequential decision process” [1]

[1] https://www.sciencedirect.com/science/article/abs/pii/S0927050705801720

MDP Problem Definition

State space (S) All states the agent can be in, with **initial state** s_0

Actions ($A(s)$) Set of applicable actions agent can execute in each state s

Transition model ($T(s,a,s')$)

Function describing how actions change states;
modeled as probability distribution $P(s' \mid s, a)$

Reward function ($R(s)$) The value of being in state s

Discount factor (γ) How much to prioritized current rewards over future rewards $0 \leq \gamma \leq 1$

It's solution? A policy; $\pi: S \rightarrow A$

MDP Definition via Example

Consider a complex task environment: *Dungeons & Dragons* battle



Screencap from Netflix's Stranger Things via https://www.thetimes.co.uk/article/dungeons-and-dragons-fans-enter-the-land-of-popularity-j0q53lkcw

MDP D&D

An MDP has 5 components: $(S, A(s), T(s, a, s'), R(s), \gamma)$

State S:

- Character stats, health, injuries, inventory, location, skills
- Monster stats, health, location
- Not too bad, all of this is finite and known to players

Actions $A(s)$:

- Any combination of weapons/spells in inventory used on a certain # of monsters
- Spells to help team
- But also…using items
- Starts to get complicated if you include saying things as the character, but we can ignore this for now



https://www.pngmart.com/files/23/D20-Transparent-PNG.png

MDP D&D

An MDP has 5 components: $(S, A(s), T(s, a, s'), R(s), \gamma)$

Transition function $T(s, a, s')$:

- Some sort of model of the rules of the game
  - E.g., What happens when I cast fireball on this goblin?

Reward $R(s)$:

- What do we need to define this for our agent?
- The experience we get for slaying the monsters?
  - What if we wanted to spare them or they got away?
- Did we have any other objectives?
  - E.g., Did we get the magical amulet?
- Note that these rewards don't occur at every step, just at the end!

Discount factor γ: No problem, just a parameter



https://www.pngmart.com/files/23/D20-Transparent-PNG.png

How can we solve this type of MDP?

How can we avoid difficult-to-specify transition functions?

- The world is our transition function!
- We can try an action, and see what happens
- No need to specify everything

How can we avoid difficult-to-specify reward functions?

- Similar idea as above
- Find a way to observe a reward signal
- No need to formulate a function

Intuition Behind RL

Learning by **trial-and-error**

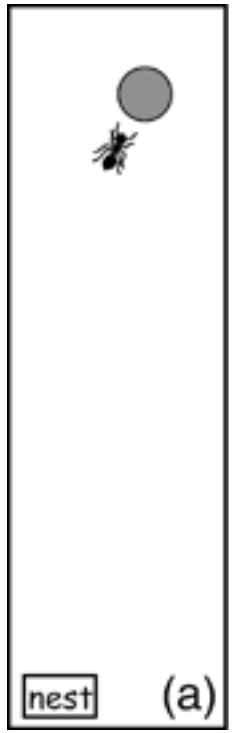
1. Try doing something
2. Leverage the real world or a simulator to observe the transition function and reward function
3. Record the results
4. Repeat, focusing more on what worked in past trials







Analogy: Ants

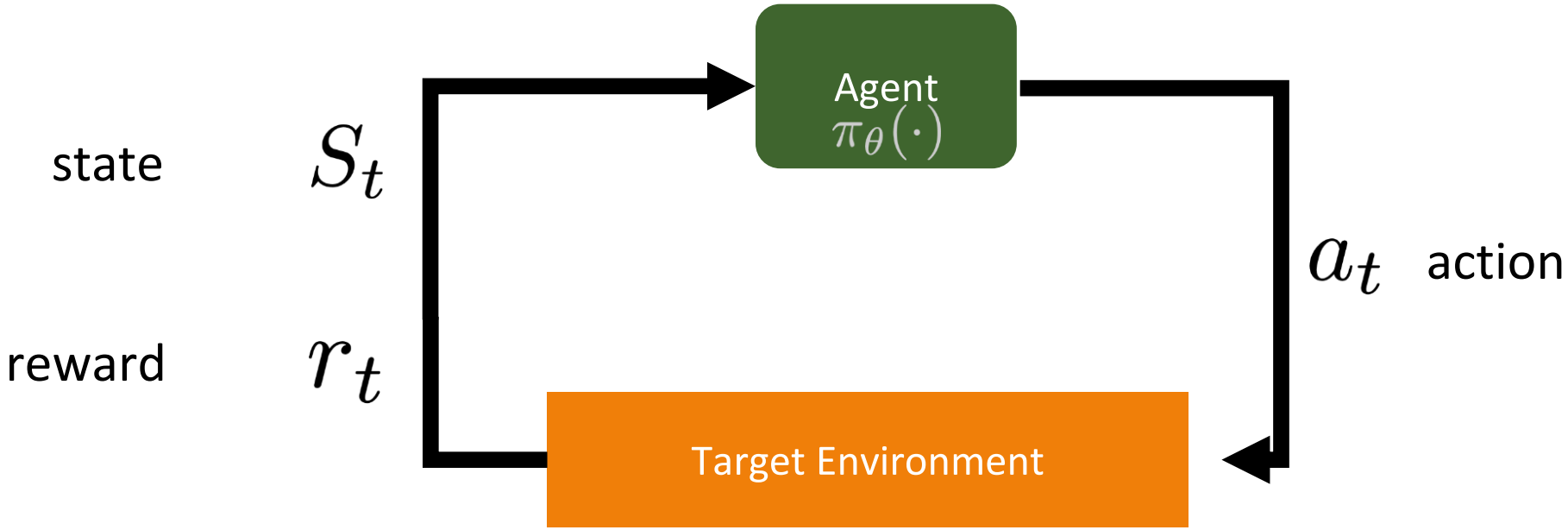


https://resjournals.onlinelibrary.wiley.com/doi/10.1111/j.1365-3032.2008.00658.x



https://i.makeagif.com/media/6-14-2016/3eCU2f.gif

Reinforcement Learning



$a_t \sim \pi_\theta(S_t)$: policy

High-Level RL Algorithm

Goal: Compute a policy $\pi(s) \rightarrow a$

1. Start in initial state s_t (t = current time)
2. Observe the reward r
3. Pick an action a_t to execute
4. Executing a_t will cause the agent to go to state s_{t+1}
5. Repeat steps 1-4, until we have a lot of data
6. Compute the best policy $\pi(s)$ we can given what we've observed

We call this a **trial**

Model-Based vs. Model-Free

Model-Based RL: learn $T(s, a, s')$, then compute $\pi(s)$

Interacting with the environment produces samples of the transition function and the reward function

Build **explicit** models of T and R

- $p(s'|s, a) = (\#\ times\ doing\ a\ in\ s\ led\ to\ s')/(\#\ times\ tried\ a\ in\ s)$
- $R(s) = average\ reward\ for\ s$

Compute $\pi(s)$ with value/policy iteration

Model-Based vs. Model-Free

Model-Free RL: don't try to learn $T(s, a, s')$, but compute $\pi(s)$ directly

Interacting with the environment produces samples of the transition function and the reward function

We **only** care about T and R as a means of computing utility and an optimal policy

Directly compute optimal policy from samples of T and R

Model-Based vs. Model-Free

Model-based RL

- Useful if we need to know the transition function
- Many model-based approaches
- Not really used with language

Model-Free RL

- More straightforward approach if all you want is a policy
- Approach we will focus on in detail
- Commonly used model-free approaches: Q-learning & policy optimization (PPO, A2C)

D&D Q-Table Example

	Use Sword	Move forward	Move back	Heal
S1 *Have sword* *Monster alive*	5	2	0	0
S2 *No sword* *Monster alive*	0	0	10	11
S3 *Have sword* *Monster dead*	0.001	2	1	5
S4 *No Sword* *Monster dead*	0	1	0	3

The Q-Learning Algorithm

1. Start in initial state s_t (t = current time)
2. Observe the reward r
3. Pick an action a_t to execute 
4. Executing a_t will cause the agent to go to state s_{t+1}
5. Update Q-value for $Q(s_t, a_t)$
6. Repeat steps 1-5, until we have a lot of data
7. Compute $\pi(s)$ from $Q(s, a)$ $\pi^*(s) = \arg\max_a Q^{\pi^*}(s, a)$

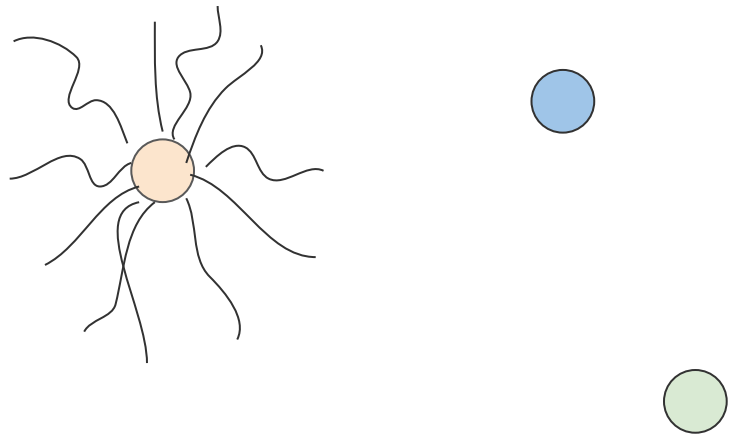
Selecting Actions during Trials

Option 1: Random Actions

Always pick an action at random

This provides the extensive **exploration** of the state space

Start
Medium reward
High reward



Selecting Actions during Trials - Example

Scenario: We just moved to a new city for college. We need to learn a policy for ourselves to get from our apartment to campus.

Option 1: Random Actions

Always pick an action at random

Every day, we'll take a random turn at every intersection.

What's going to happen in this case?

Selecting Actions during Trials

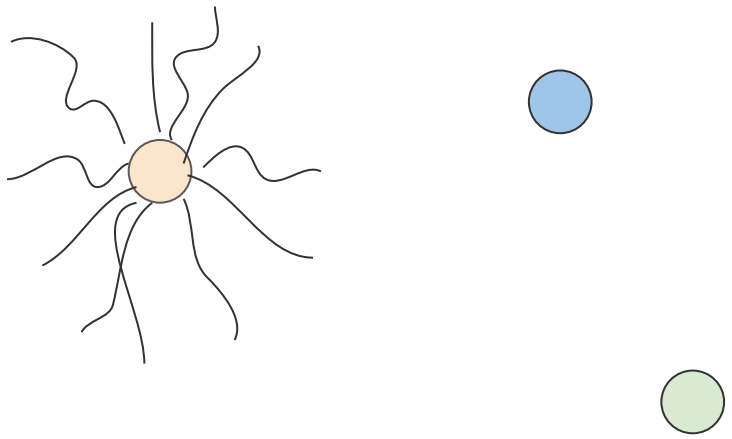
Option 1: Random Actions

Always pick an action at random

This provides the extensive **exploration** of the state space

- Eventually cover the state-action space
- May not see the same states multiple times (problem if we have stochastic actions)
- Biased towards seeing states near start state more often

Start
Medium reward
High reward



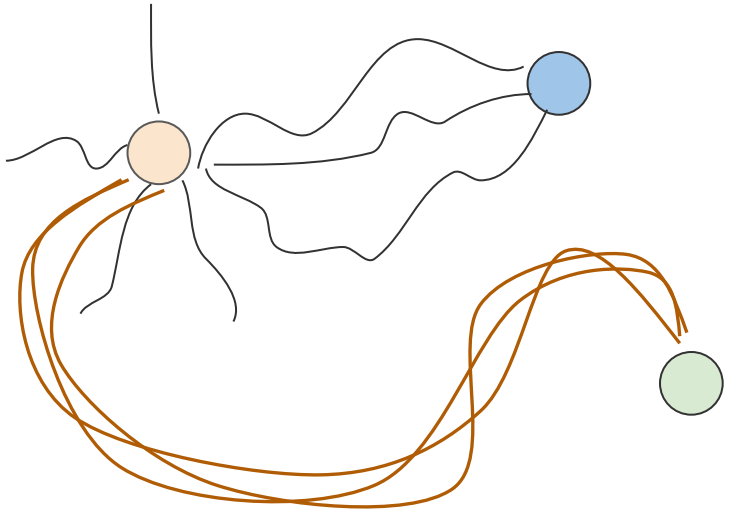
Selecting Actions during Trials

Option 2: Greedy Actions

Always pick action $a = \underset{a}{\text{argmax}}\, Q(s, a)$

This provides **exploitation** of the Q-values and rewards we have seen in past trials

Start
Medium reward
High reward



Selecting Actions during Trials - Example

Scenario: We just moved to a new city for college. We need to learn a policy for ourselves to get from our apartment to campus.

Option 2: Greedy Actions

Always pick action $a = \underset{a}{\operatorname{argmax}}\, Q(s, a)$

As soon as we find a route through random wandering, then every day we'll take the turn that got us the best result in the past

What's going to happen in this case?

Selecting Actions during Trials

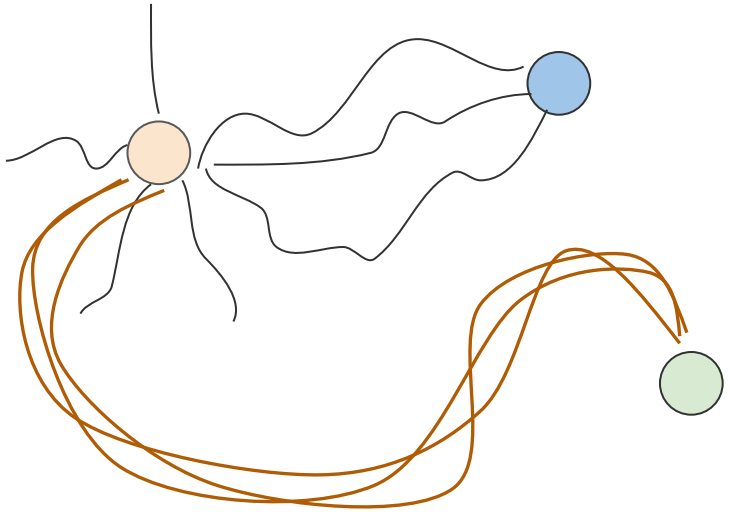
Option 2: Greedy Actions

Always pick action $a = \underset{a}{\text{argmax}}\, Q(s, a)$

This provides **exploitation** of the Q-values and rewards we have seen in past trials

- Starts by exploring randomly
- Over time, bias towards states with reward
- May focus in on sub-optimal reward states
- May focus on repeating sub-optimal policies

Start
Medium reward
High reward



https://www.youtube.com/watch?v=hx_bgoTF7bs&t=90s

Suboptimal Paths



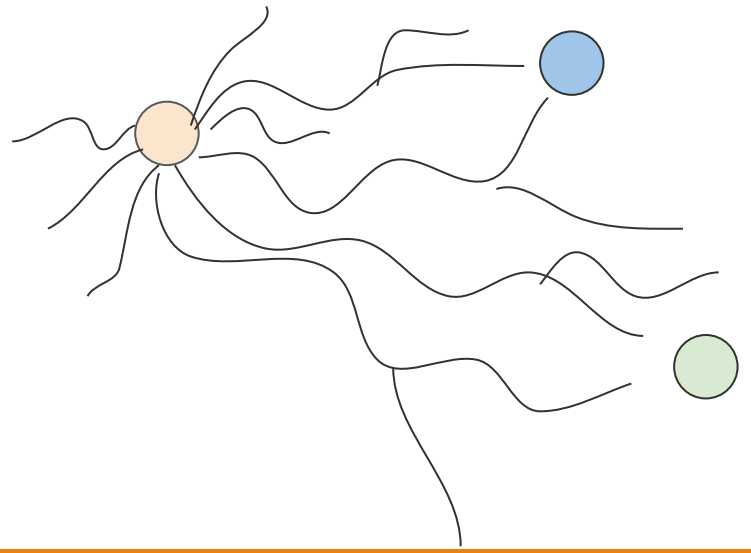
Selecting Actions during Trials

Option 3: ε-Greedy Action Selection

With probability ε, pick a random action
Otherwise, pick action $a = \underset{a}{\text{argmax}}\, Q(s, a)$

Simple way to trade off **exploration** and **exploitation**

Start
Medium reward
High reward



Selecting Actions during Trials - Example

Scenario: We just moved to a new city for college. We need to learn a policy for ourselves to get from our apartment to campus.

Option 3: ε-Greedy Action Selection

With probability ε, pick a random action
Otherwise, pick action $a = \underset{a}{\text{argmax}}\, Q(s, a)$

What's going to happen in this case?

Every day, we'll usually take turns that gave us the best results in the past, but sometimes we'll try something new or retry something less efficient

Selecting Actions during Trials

Option 3: ε-Greedy Action Selection

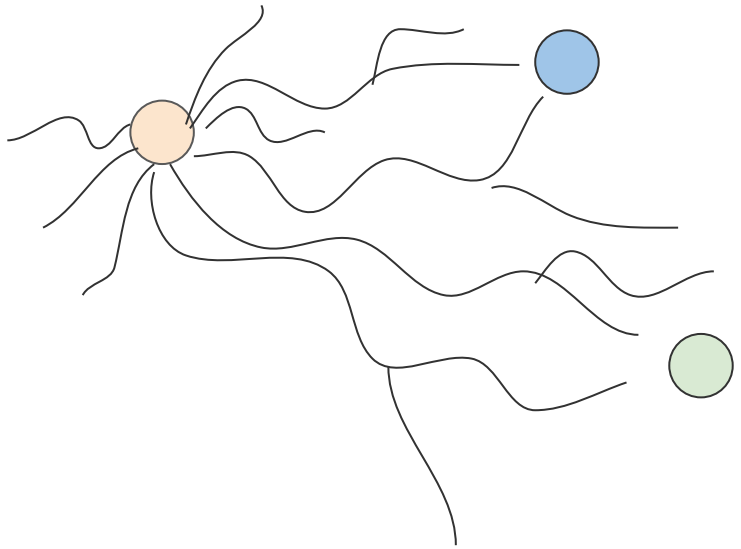
With probability ε, pick a random action
Otherwise, pick action $a = \underset{a}{\text{argmax}}\, Q(s, a)$ Simple way to trade off **exploration** and **exploitation**

Can explore at any time

Tends to move towards rewards (i.e. spend time exploring relevant areas of state space)

GLIE: Greedy in the Limit of Infinite Exploration – Guaranteed to converge to optimal $\pi(s)$**!**

Start
Medium reward
High reward

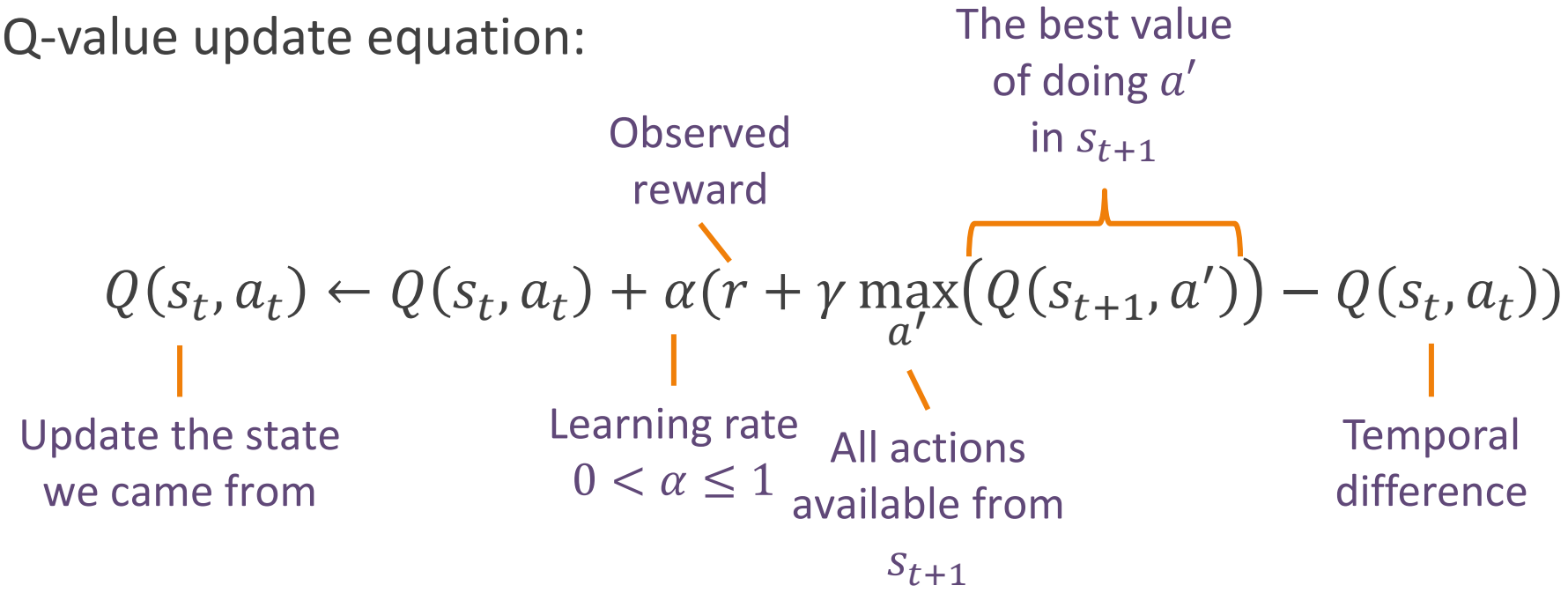


The Q-Learning Algorithm

1. Start in initial state s_t (t = current time)
2. Observe the reward r
3. Pick an action a_t to execute
4. Executing a_t will cause the agent to go to state s_{t+1}
5. Update Q-value for $Q(s_t, a_t)$ 
6. Repeat steps 1-5, until we have a lot of data
7. Compute $\pi(s)$ from $Q(s, a)$ $\pi^*(s) = \arg\max_a Q^{\pi^*}(s, a)$

Updating Q-Values

The Q-value update equation:



Updating Q-Values

The Q-value update equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(\underbrace{r + \gamma \max_{a'}\big(Q(s_{t+1}, a')\big)}_{\text{New measurement of } Q(s_t, a_t)} - Q(s_t, a_t))$$

We're updating the Q-Value based on its observed change in value

Updating Q-Values

The Q-value update equation (for terminal states):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r + \gamma \max_{a'}(Q(s_{t+1}, a')) - Q(s_t, a_t))$$

0

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r - Q(s_t, a_t))$$

Why not set this directly to the observed reward?

- We can set $Q(s_t, a_t) = r$ if our rewards are deterministic
- We should use the update equation if our rewards are stochastic!

What are we “Learning” in Q-Learning?

Instead of specifying a complex transition function and reward function, we are learning a table of Q-values:

	Use Sword	Move forward	Move back	Heal
S1 *Have sword* *Monster alive*	5	2	0	0
S2 *No sword* *Monster alive*	0	0	10	11
S3 *Have sword* *Monster dead*	0.001	2	1	5
S4 *No Sword* *Monster dead*	0	1	0	3

Why a lookup table?

Learning a table is better than specifying it by hand

Easy to interpret

But can we learn a more concise representation?

We can use function approximation to approximate the values in the Q-value table.

Function Approximation for Q-Learning

Function approximation: using a parameterized representation to generate approximate values instead of using a table of exact values

Example: use a weighted linear function to approximate Q-values

$$Q_{\boldsymbol{\theta}}(s,a) = \theta_1 f_1(s,a) + \theta_2 f_2(s,a) + \theta_3 f_3(s,a) + \cdots + \theta_n f_n(s,a)$$

Approximate Q-value

Weight parameter, which we can learn using linear regression or online learning

Feature calculated from state and action

Function Approximation for Q-Learning

What does this do for us?

We don't have to store a table of size $|S|$ x $|A|$

We can calculate Q-functions for states and actions that we **haven't seen before**!

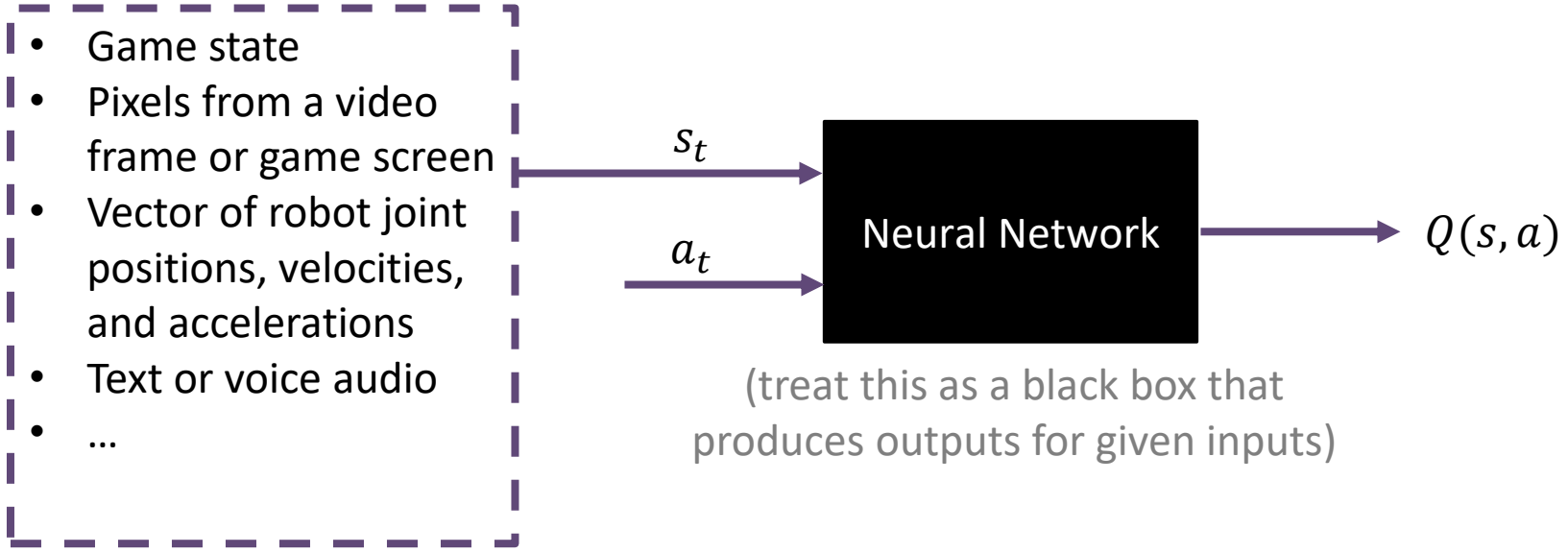
$$Q_{\boldsymbol{\theta}}(s,a) = \theta_1 f_1(s,a) + \theta_2 f_2(s,a) + \theta_3 f_3(s,a) + \cdots + \theta_n f_n(s,a)$$

Function approximation **generalizes** to unexplored states and actions.

Getting Deep: Modern Function Approximation

Instead of a linear function, we can use a deep neural network as a function approximator for the Q-values.

Goal: learn a neural network to replace the Q-value table:



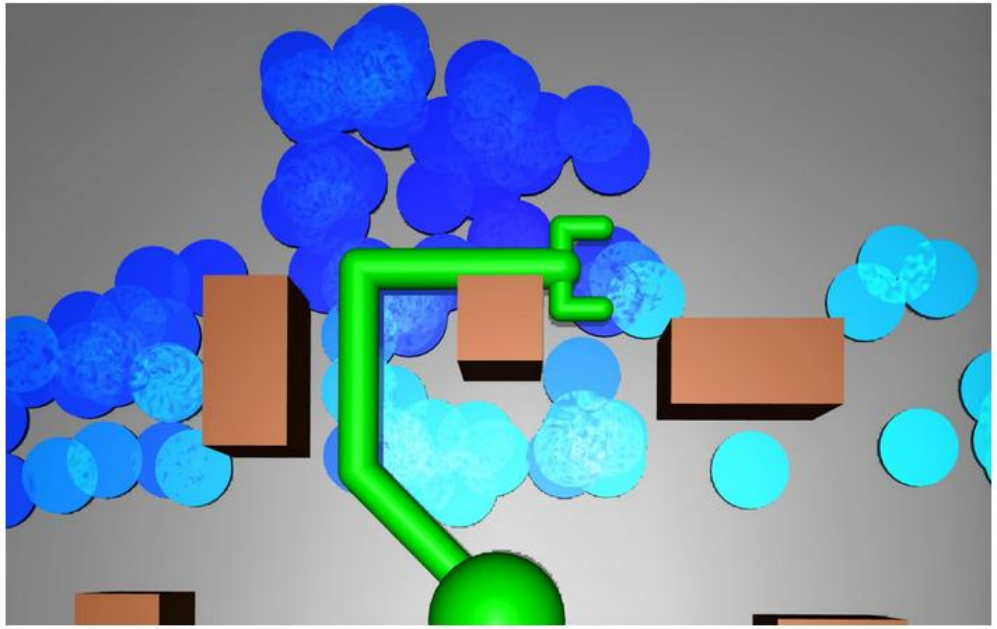
Why is Deep Learning Effective for RL?

Advantage of neural networks for function approximation:

Can be designed *without* the assumption that nearby states should generalize to similar values

Models are large enough to encode complex relationships between agent and environment

Example on right: similar colored positions are states that are “close” to each other

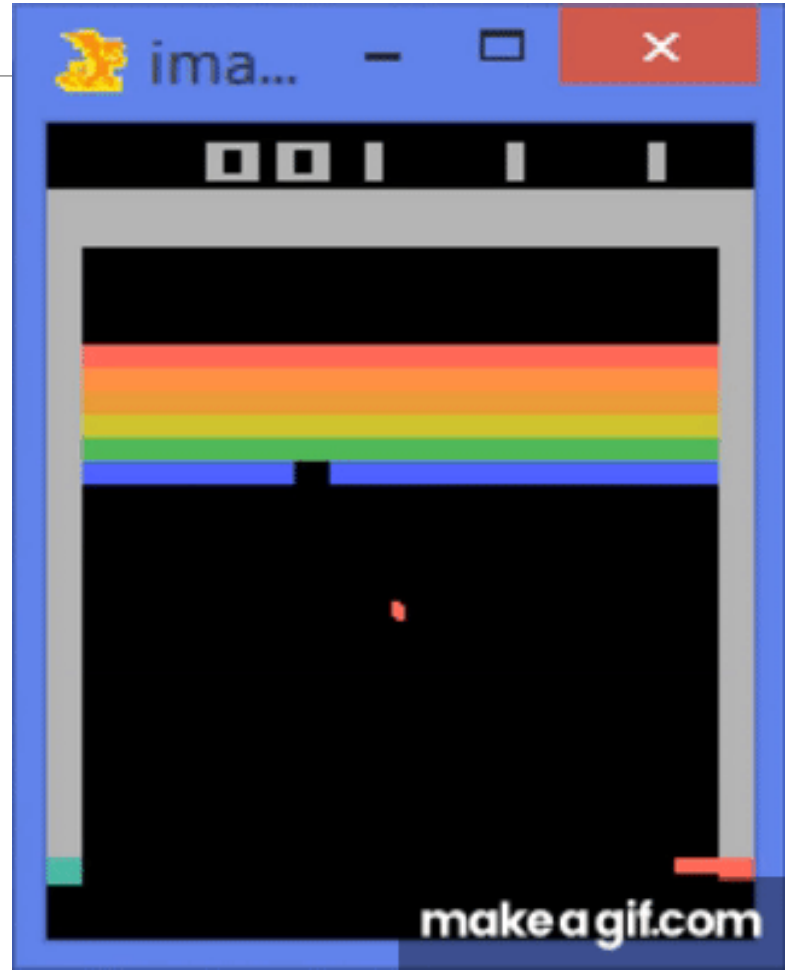


A distance metric learned by a neural network [3]. **Lighter blue → more distant.** The agent, which was trained to grasp objects using the robotic arm, takes into account obstacles and arm length when it measures the distance between two states.

From https://distill.pub/2019/paths-perspective-on-value-learning/

RL for Game Playing

RL has been used for game playing for a while but it wasn't until recently that it has been used for language modeling

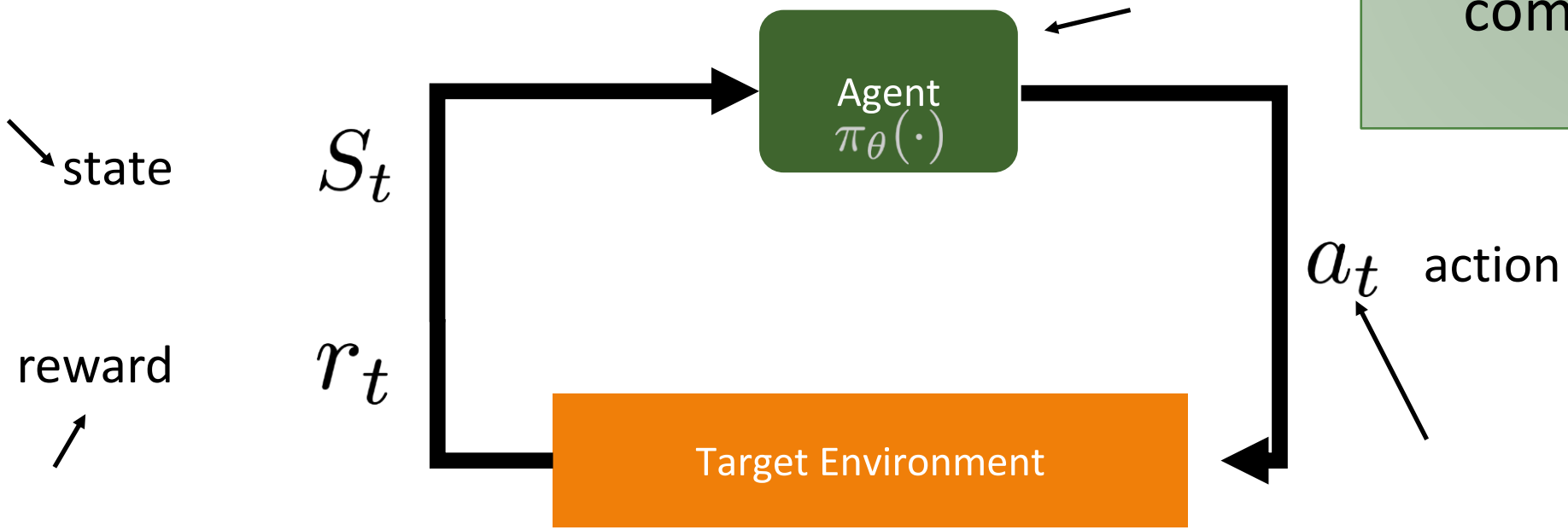


https://www.youtube.com/watch?v=V1eYniJ0Rnk

Playing Atari with Deep Reinforcement Learning. Volodymyr Mnih, et al. 2013

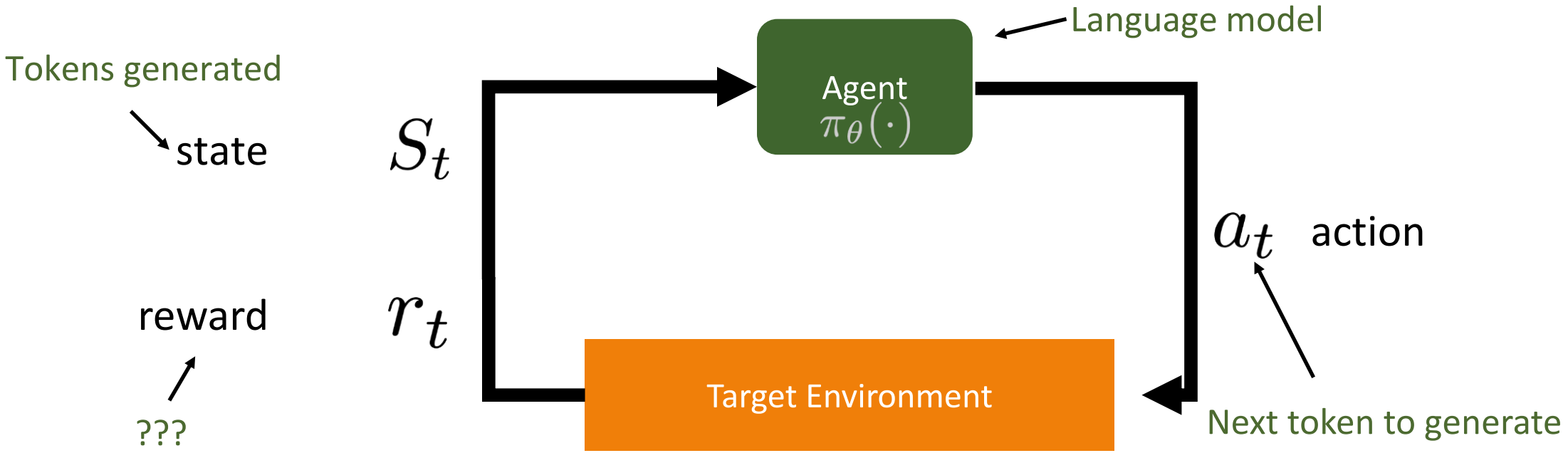
Think-Pair-Share

If we were using RL with language models, what would these four components be?



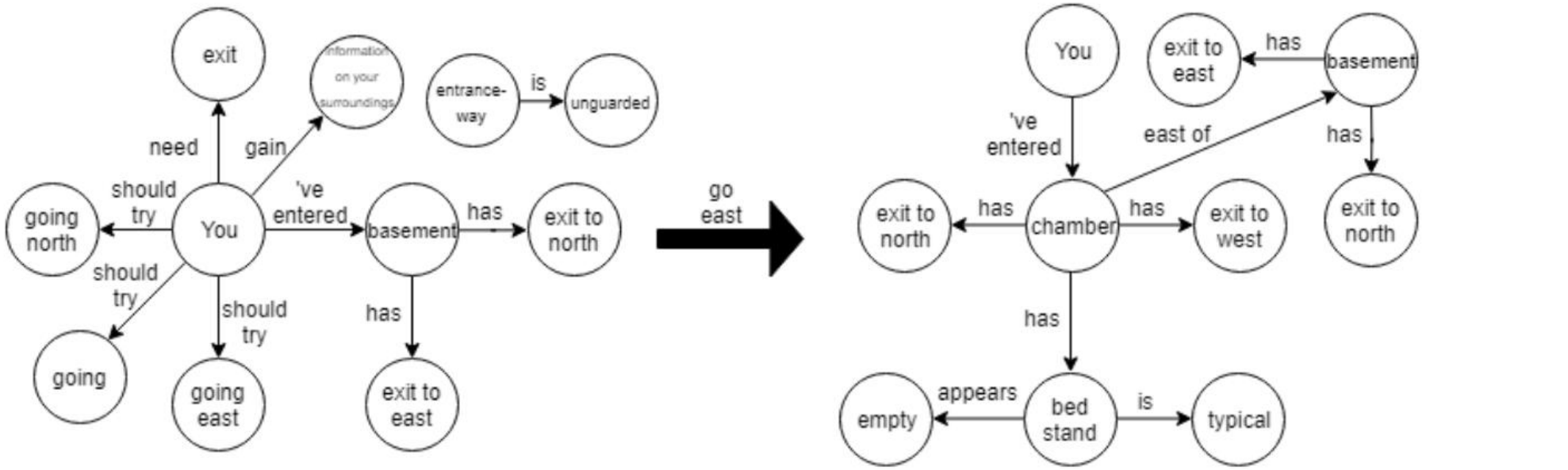
$a_t \sim \pi_\theta(S_t)$: policy

RL in the Context of Language Models…



$a_t \sim \pi_\theta(S_t)$: policy

RL for IF Playing

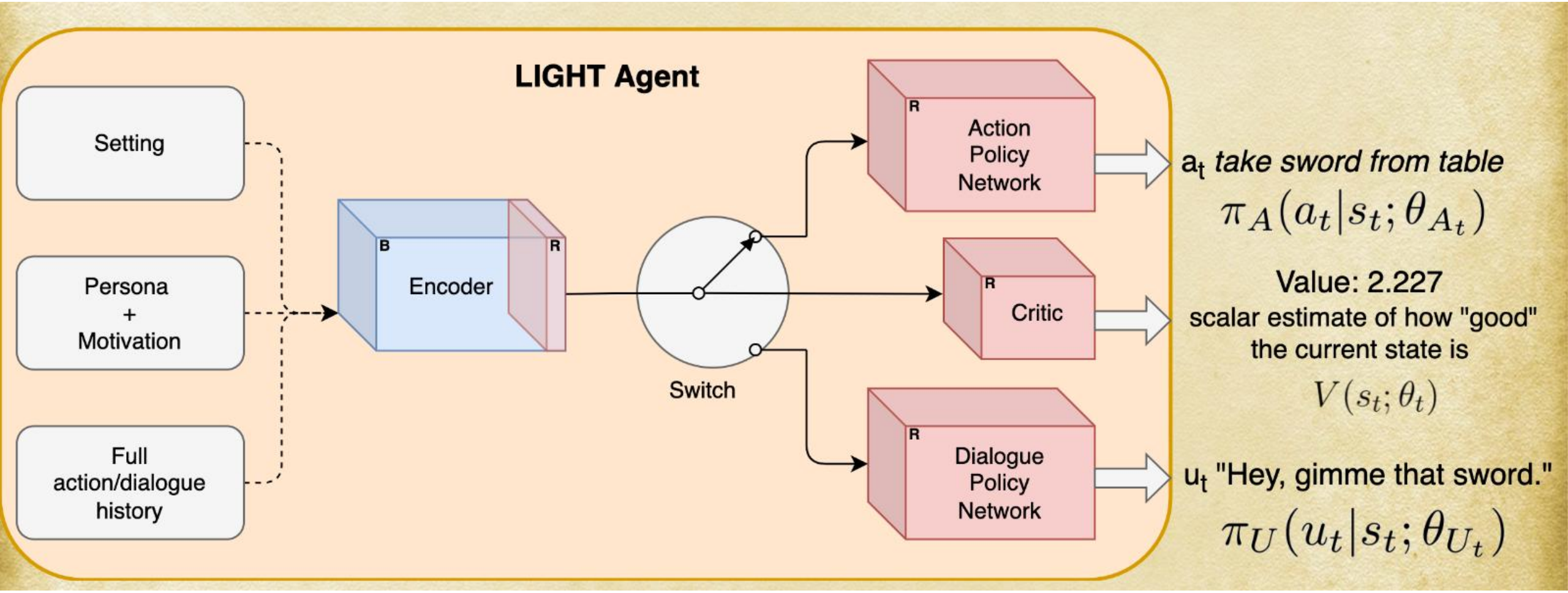


You've entered a basement. You try to gain information on your surroundings by using a technique you call "looking." You need an unguarded exit? You should try going east. You don't like doors? Why not try going north, that entranceway is unguarded.

You've entered a chamber. You can see a bed stand. The bed stand is typical. The bed stand appears to be empty. There is an exit to the north. Don't worry, it is unblocked. There is an unblocked exit to the west.

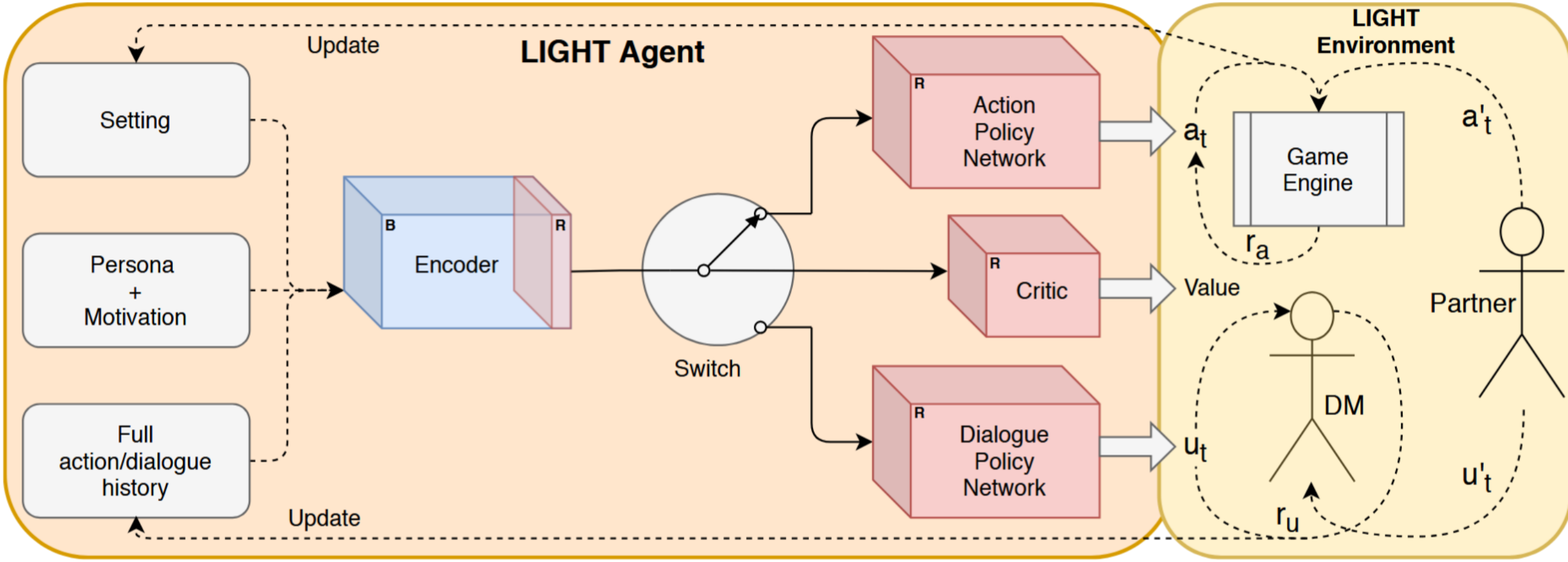
Playing Text-Adventure Games with Graph-Based Deep Reinforcement Learning (Ammanabrolu & Riedl, NAACL 2019)

RL for Action & Dialog

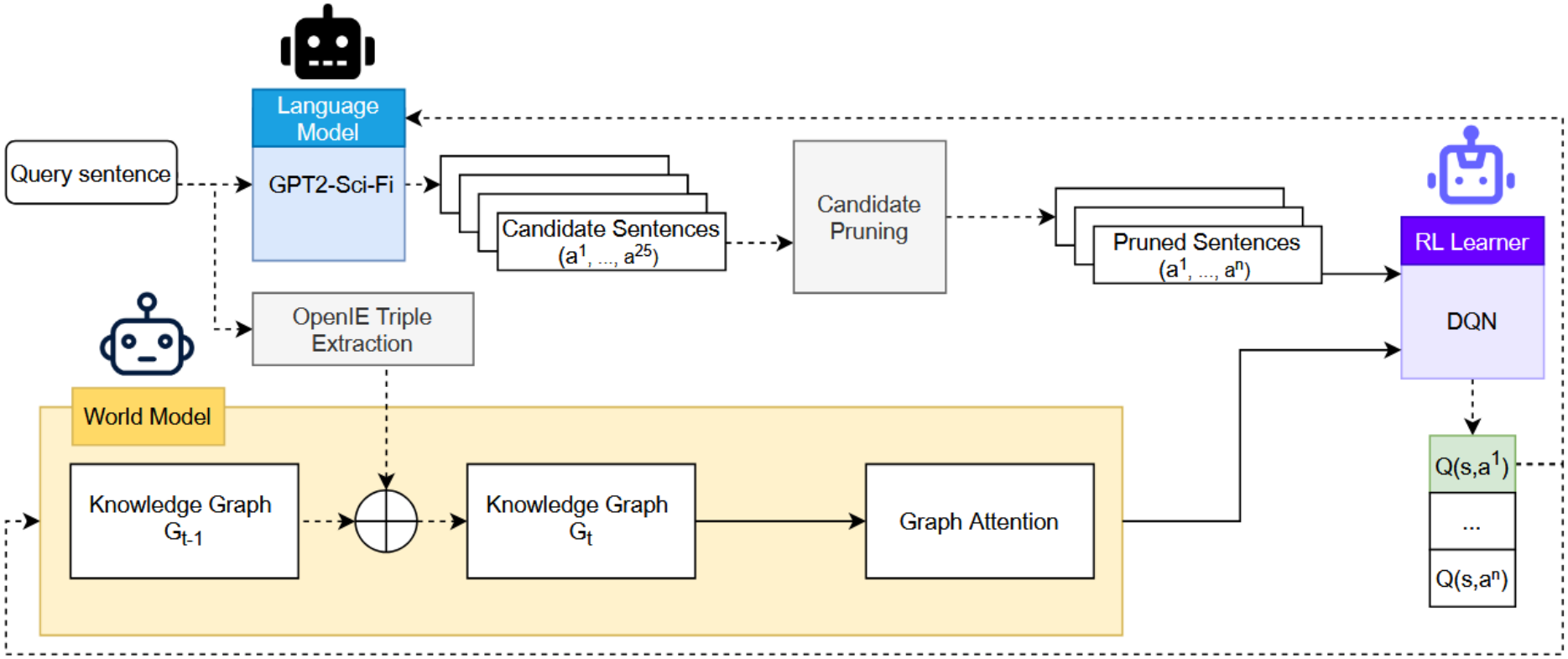


How to Motivate Your Dragon: Teaching Goal-Driven Agents to Speak and Act in Fantasy Worlds (Ammanabrolu et al., NAACL 2021)

RL for Action & Dialog



RL for Story Generation



Goal-Directed Story Generation: Augmenting Generative Language Models with Reinforcement Learning. Amal Alabdulkarim, et al. 2019

Reinforcement Learning from Human Feedback

arxiv in Sep 2019
NeurIPS 2020

Fine-Tuning Language Models from Human Preferences

Daniel M. Ziegler* **Nisan Stiennon*** **Jeffrey Wu** **Tom B. Brown**
Alec Radford **Dario Amodei** **Paul Christiano** **Geoffrey Irving**
OpenAI
{dmz,nisan,jeffwu,tom,alec,damodei,paul,irving}@openai.com

arxiv in Sep 2020
NeurIPS 2020

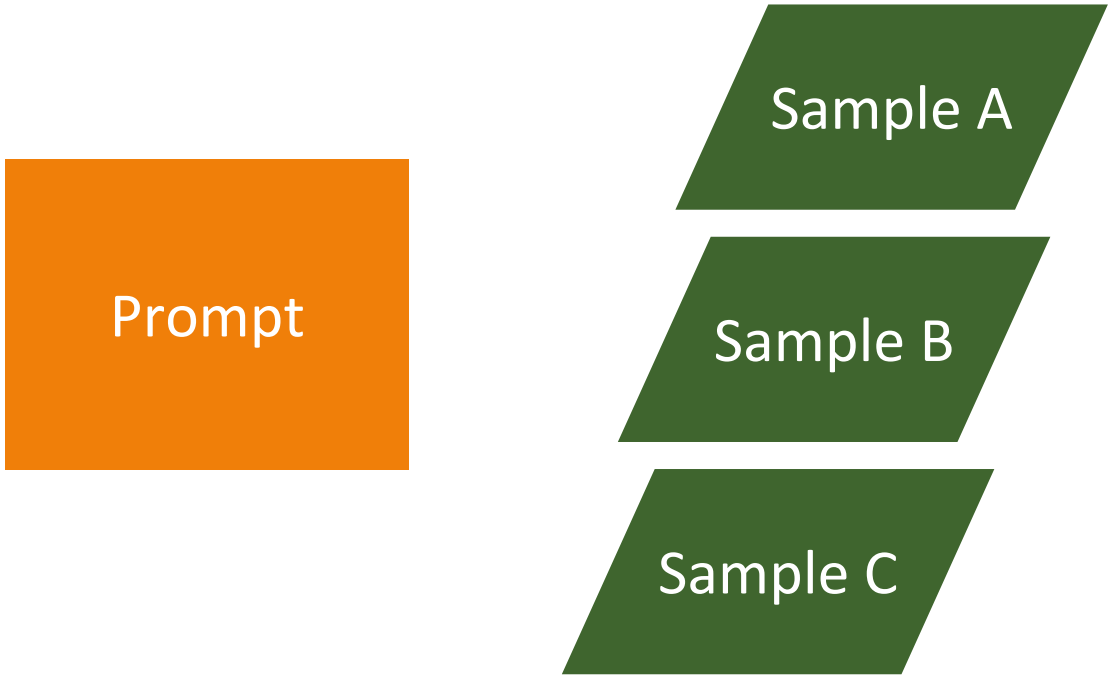
Learning to summarize from human feedback

Nisan Stiennon* **Long Ouyang*** **Jeff Wu*** **Daniel M. Ziegler*** **Ryan Lowe***
Chelsea Voss* **Alec Radford** **Dario Amodei** **Paul Christiano***
OpenAI

Human Preferences

A set of sampled completions for a prompt

Triples



$$D = \{x^i, y_w^i, y_l^i\}$$

Prompt

Preferred Response

Dispreferred Response

👩‍💻 Can you help me write a resignation letter to my current employer, while leaving on good terms and expressing gratitude for the opportunities provided?

Example: Annotation

🤖 Here are two responses from the chatbot. (Please scroll down on the content to see the entire response if it is too long)

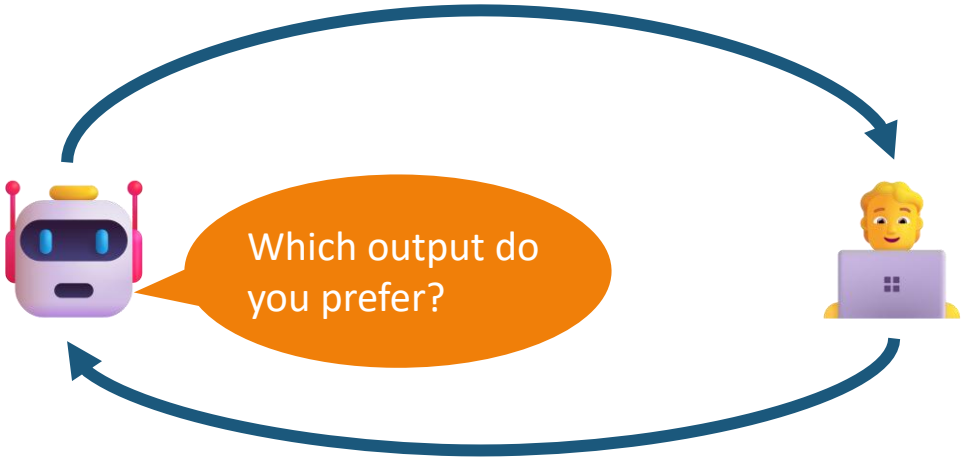




Annotator needs to choose whether they prefer A or B.

But..

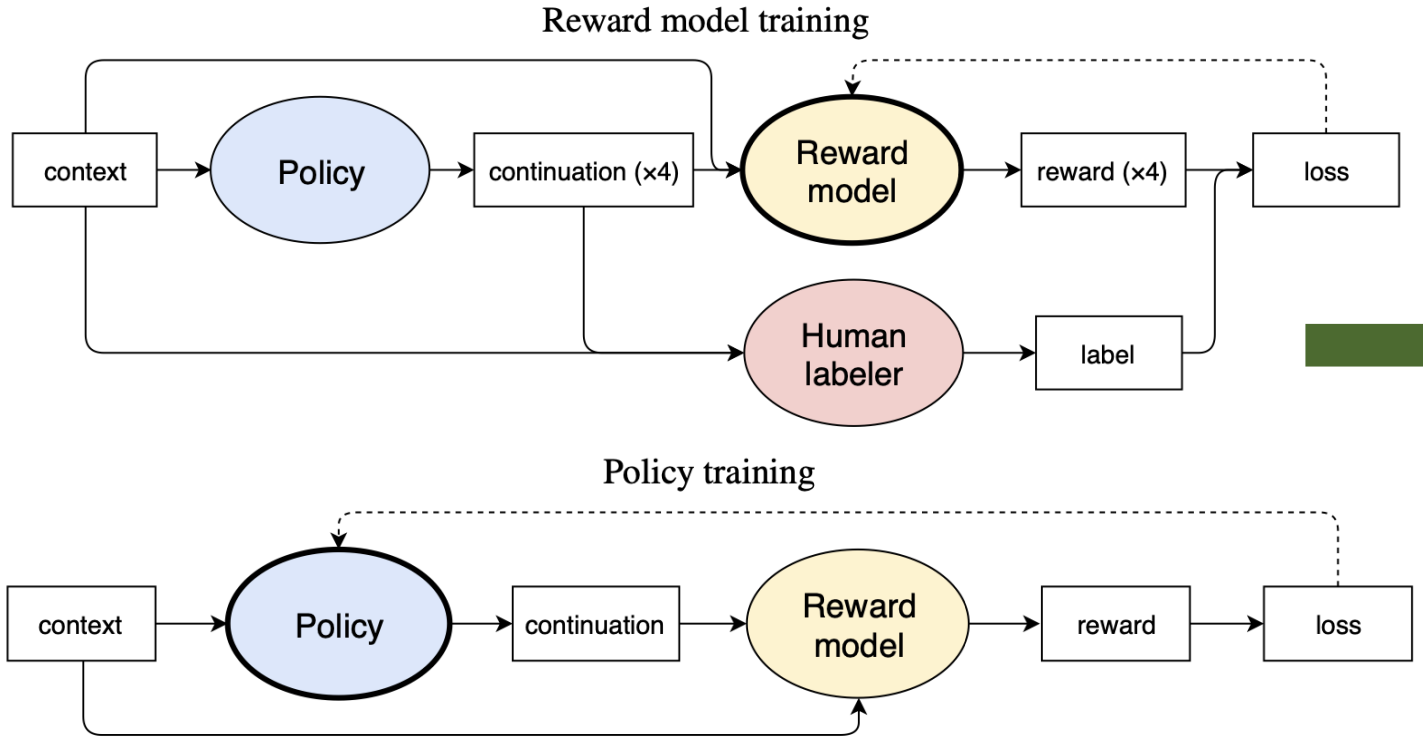
How do we get feedback for the reward while training our RL model?



Having a human in the loop is very expensive!

But..

How do we get feedback for the reward while training our RL model?



Instead: train a Reward Model (RM) on preference data to predict preferences!

Ziegler et al., 2019 "Fine-Tuning Language Models from Human Preferences"

Reward Modeling

$$p(y_w > y_l|x) = \frac{\exp(r(x, y_w))}{\exp(r(x, y_w)) + \exp(r(x, y_l))}$$

Train on preference data.

Minimizing negative log likelihood.

Bradley-Terry Model

equivalent to

$$\mathcal{L}_R(\phi, D) = -\mathbb{E}_{(x,y_w,y_l)\sim D}[\log \sigma(r(x, y_w) - r(x, y_l))]$$

Train an LLM with an additional layer to minimize the neg. log likelihood

Fun Facts about Reward Models

Trained for 1 epoch (to avoid overfitting)!

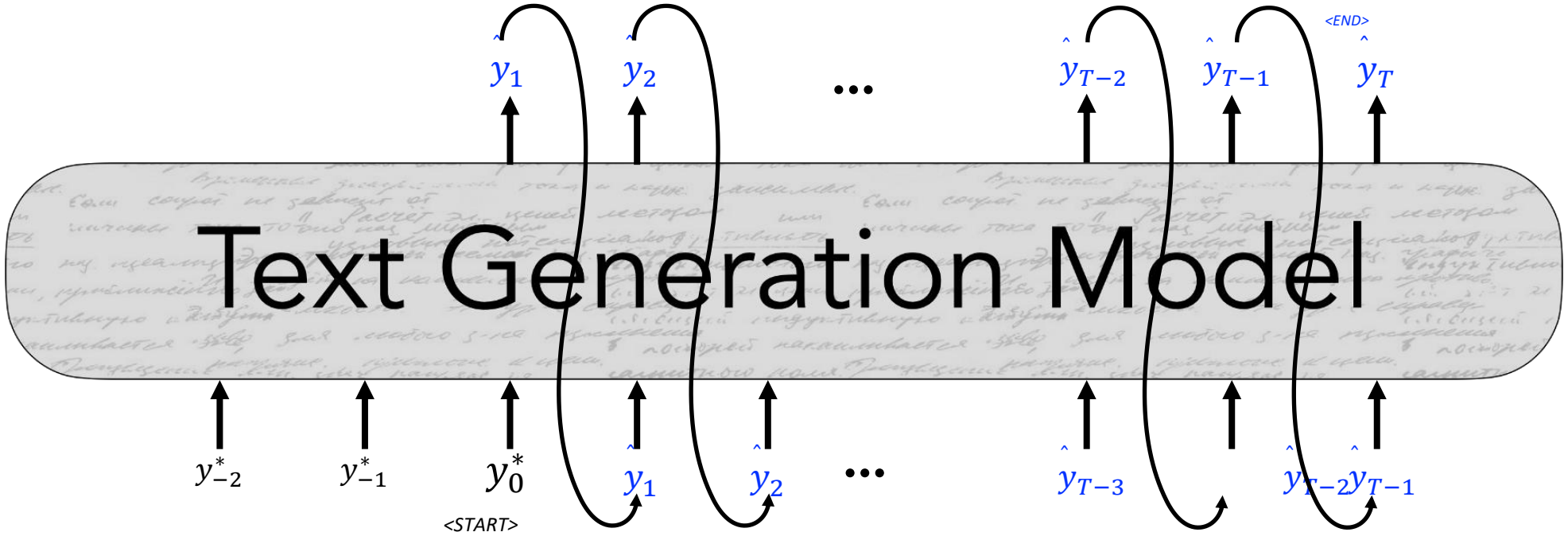
Evaluation often only has 65% - 75% agreement

Lambert et al., 2023

REINFORCE

Sample a sequence from your model, score the sequence, and use the score to train the model.

$$L_{RL} = -\sum_{t=1}^{T} r(\hat{y}_t)\log P(\hat{y}_t|\{y^*\};\{\hat{y}\}_{<t})$$



REINFORCE

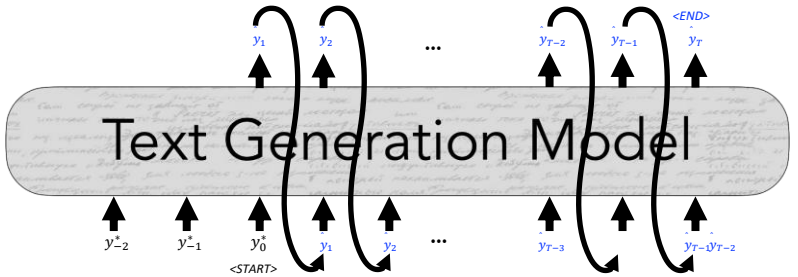
- Sample a sequence from your model, score the sequence, and use the score to train the model.

Next time, increase the probability of this sampled token in the same context.

$$L_{RL} = -\sum_{t=1}^{T} r(\hat{y}_t) \log P(y_t | \{y^*\}; \{\hat{y}\}_{<t})$$

... but increase it more if I get a higher reward from the reward function.

- $r(\cdot)$: Your reward model
- y^* :Input sequence given to the model
- $\hat{y}$:The sequence sampled from the model given y^*



Summary of Policy Gradient for RL

REINFORCE Update:

$$\theta_{t+1} := \theta_t + \alpha \frac{1}{m} \sum_{i=1}^{m} R(S_i) \nabla_{\theta_t} \log p_{\theta_t}(S_i)$$

Simplified Intuition: good actions are reinforced and bad actions are discouraged

Williams, 1992

Summary of Policy Gradient for RL

REINFORCE Update:

$$\theta_{t+1} := \theta_t + \alpha \frac{1}{m} \sum_{i=1}^{m} R(S_i) \nabla_{\theta_t} \log p_{\theta_t}(S_i)$$

If: Reward is high/positive

Then: maximize this

Simplified Intuition: good actions are reinforced and bad actions are discouraged

Williams, 1992

Summary of Policy Gradient for RL

REINFORCE Update:

$$\theta_{t+1} := \theta_t + \alpha \frac{1}{m} \sum_{i=1}^{m} R(S_i) \nabla_{\theta_t} \log p_{\theta_t}(S_i)$$

If: Reward is negative/low

Then: minimize this

Simplified Intuition: good actions are reinforced and bad actions are discouraged

Williams, 1992

Policy

We have: Reward Model

Next step: learn a **policy** to maximize the reward (minus KL regularization term) using the reward model

$$\max_{\pi_\theta} \mathbb{E}_{x\sim D, y\sim\pi_\theta(y|x)}[r_\phi(x,y)] - \beta\mathbb{D}_{KL}[\pi_\theta(y|x)||\pi_{ref}(y|x)]$$

Sampling from policy

Reward given prompt and sampled generation

KL-divergence between original model's generation and the sampled generation

Policy

We have: Reward Model

Next step: learn a **policy** to maximize the reward (minus KL regularization term) using the reward model

$$\max_{\pi_\theta} \mathbb{E}_{x\sim D, y\sim\pi_\theta(y|x)}[r_\phi(x,y)] - \beta\mathbb{D}_{KL}[\pi_\theta(y|x)||\pi_{ref}(y|x)]$$

Sampling from policy

Reward given prompt and sampled generation

Should be high!

KL-divergence between original model's generation and the sampled generation

Should be low!