Language Technologies Institute

Carnegie Mellon University

Multimodal Machine Learning

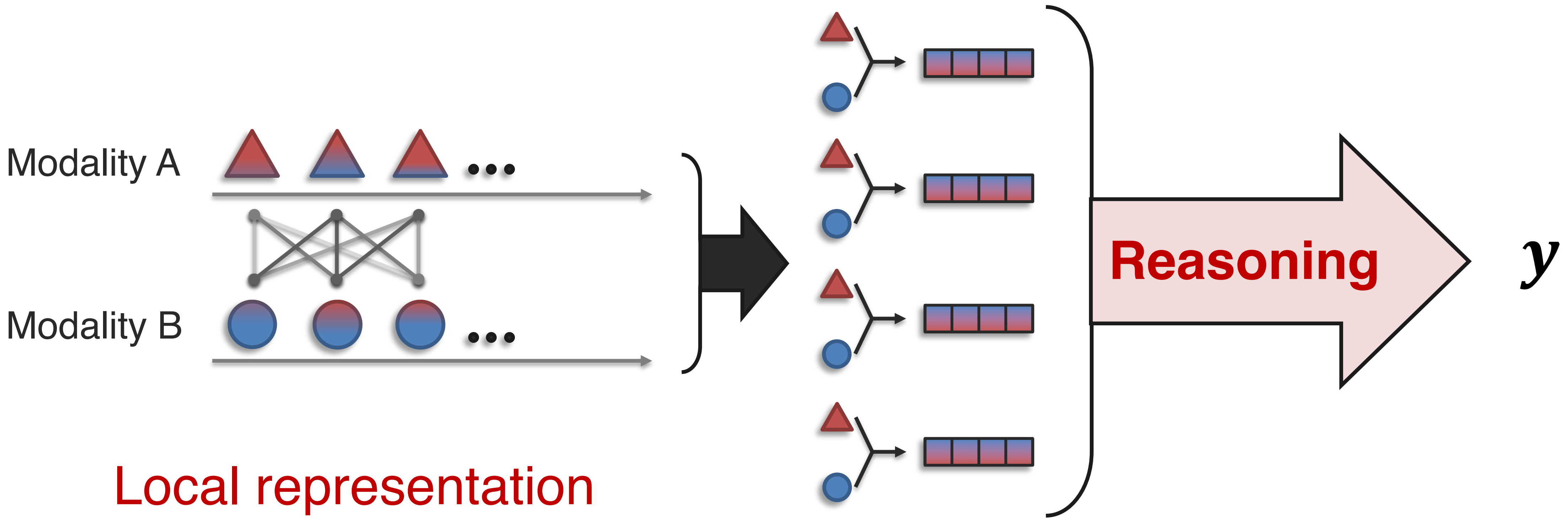
Lecture 7.1: Reasoning 2
Interaction + Structure Learning

Paul Liang

** Original course co-developed with Tadas Baltrusaitis.*
Spring 2021 edition taught by Yonatan Bisk

Reasoning

Definition: Combining knowledge, usually through multiple inferential steps, exploiting multimodal alignment and problem structure.



Local representation

+ Aligned representation

The Challenge of Compositionality

Definition: Combining knowledge, usually through multiple inferential steps, exploiting multimodal alignment and problem structure.

(a) some plants surrounding a lightbulb

(b) a lightbulb surrounding some plants

CLIP, ViLT, ViLBERT, etc.
All random chance

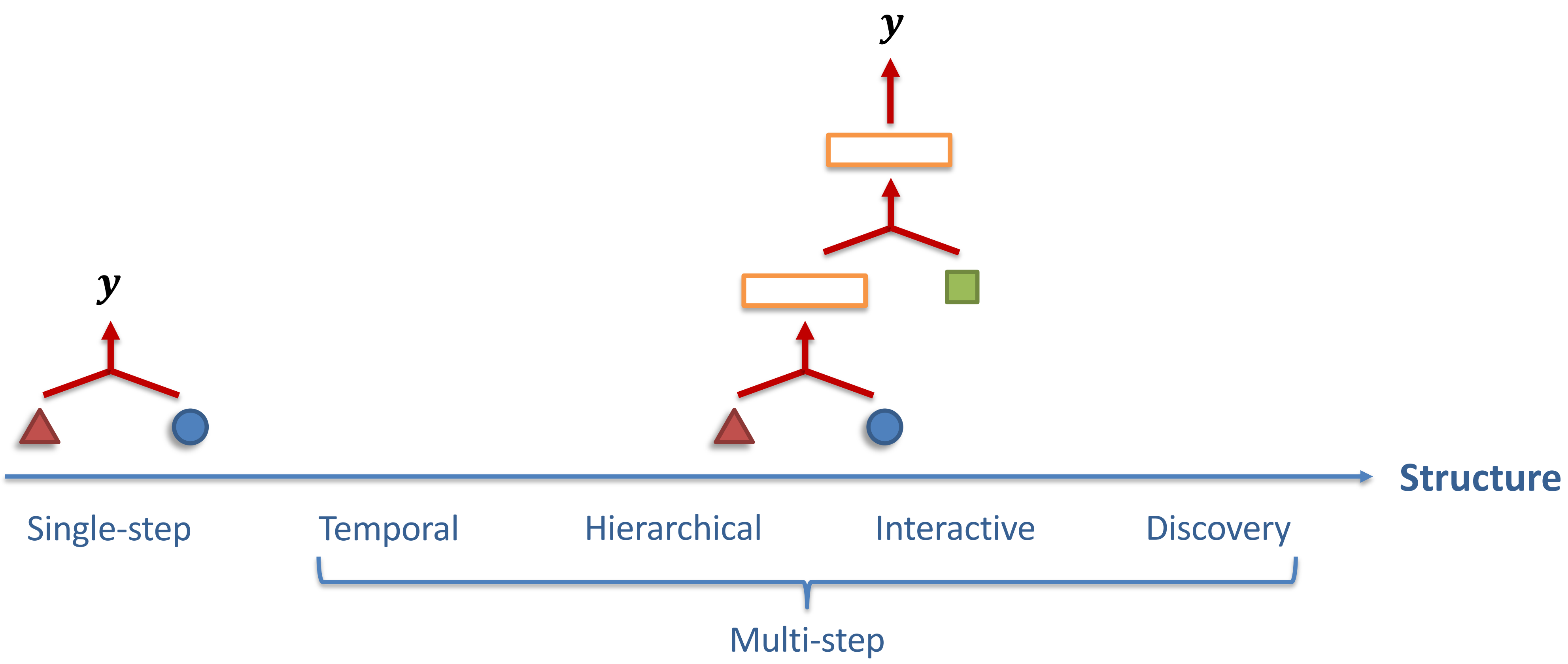
Compositional Generalization
to novel combinations outside of training data

1. Structure: <subject> <verb> <object>
2. Concepts: ‘plants’, ‘lightbulb’
3. Inference: ‘surrounding’ – spatial relation
4. Knowledge: from humans!

[Thrush et al., Winoground: Probing Vision and Language Models for Visio-Linguistic Compositionality. CVPR 2022]

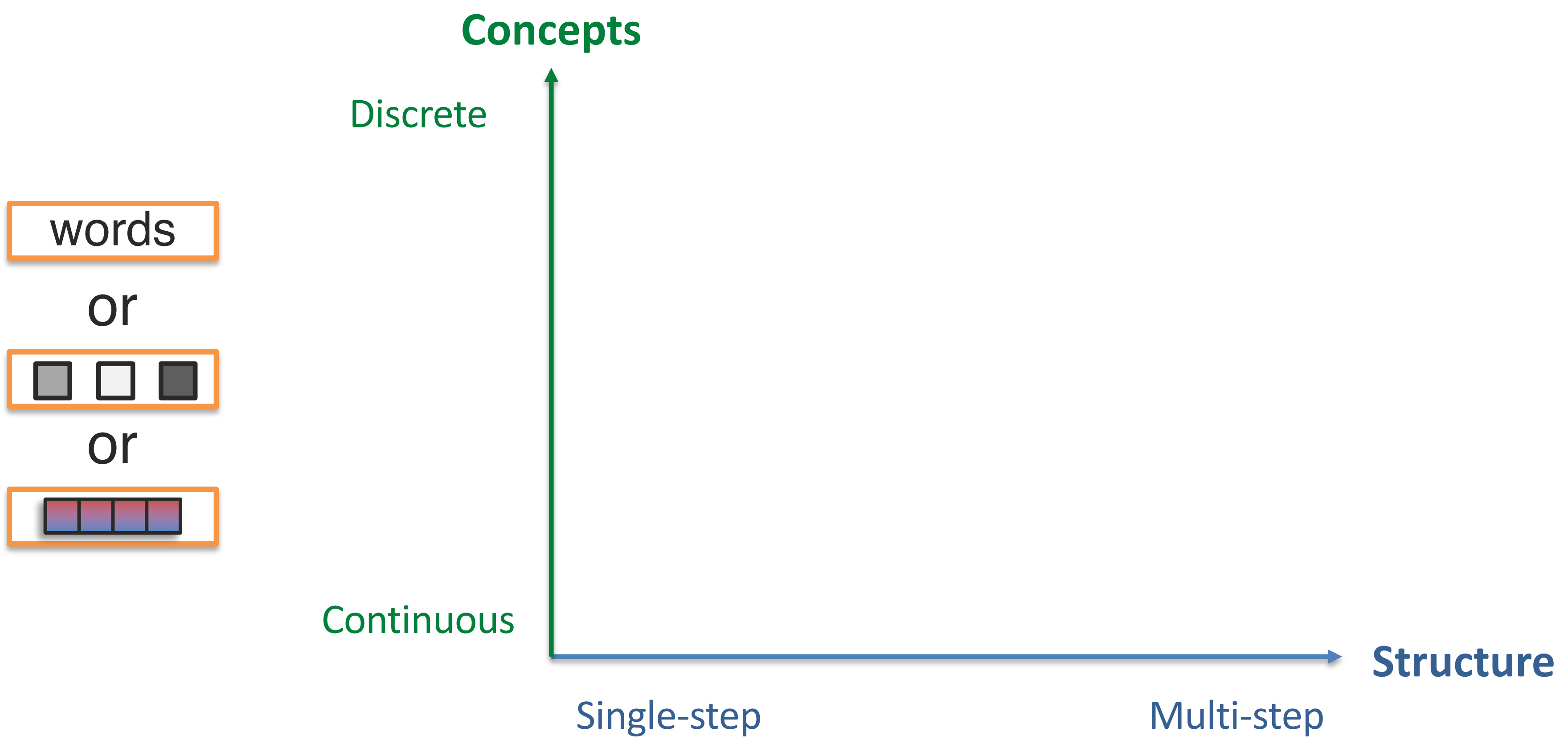
Sub-Challenge 3a: Structure Modeling

Definition: Defining or learning the relationships over which reasoning occurs.



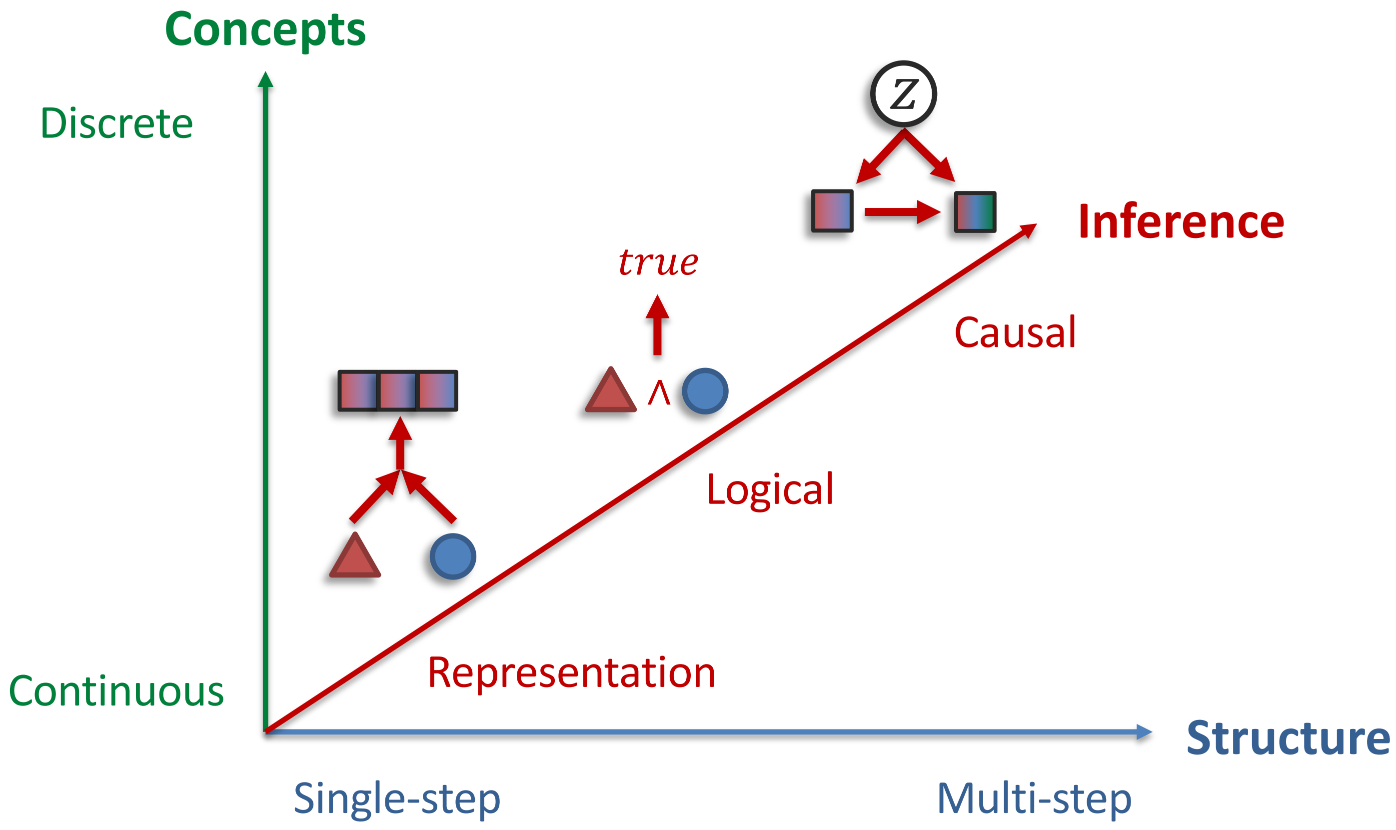
Sub-Challenge 3b: Intermediate Concepts

Definition: The parameterization of individual multimodal concepts in the reasoning process.



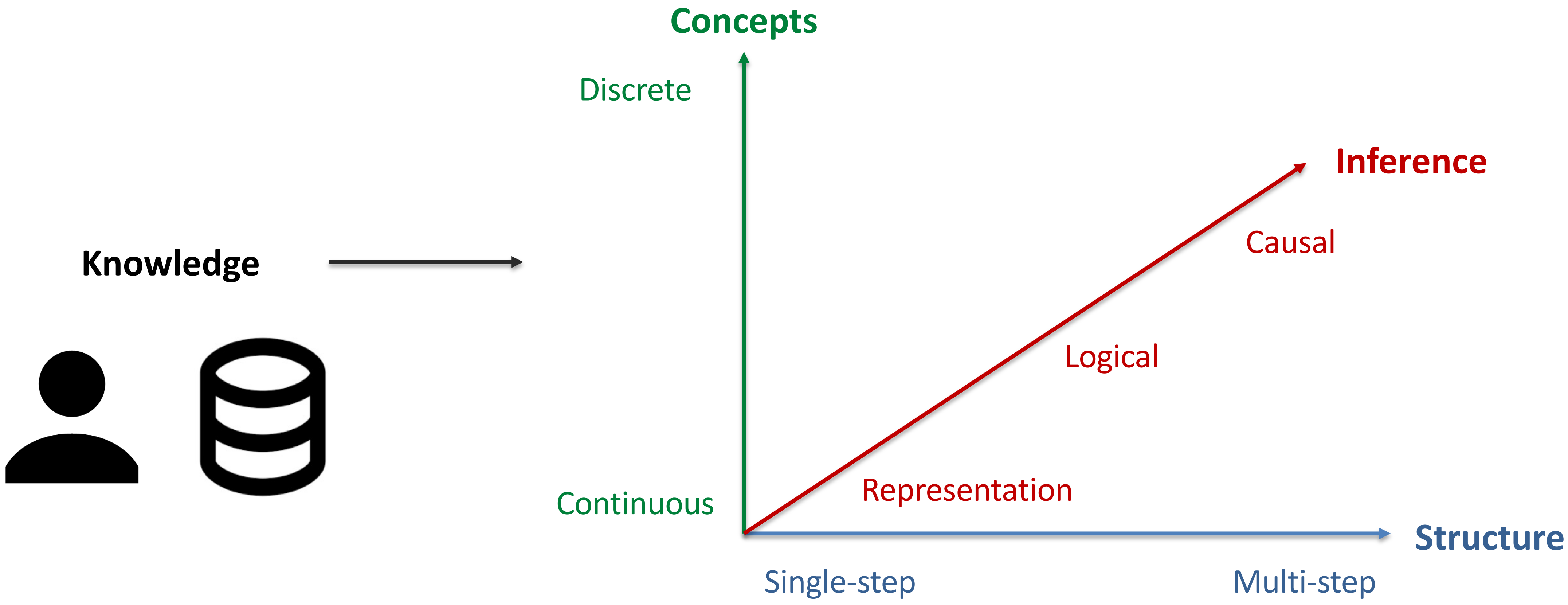
Sub-Challenge 3c: Inference Paradigm

Definition: How increasingly abstract concepts are inferred from individual multimodal evidences.



Sub-Challenge 3d: External Knowledge

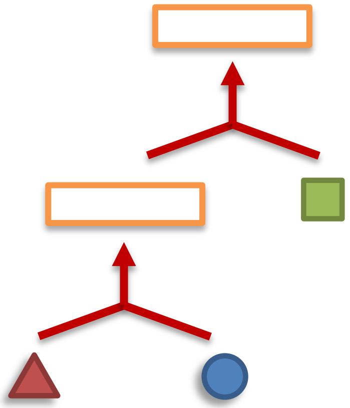
Definition: Leveraging external knowledge in the study of structure, concepts, and inference.



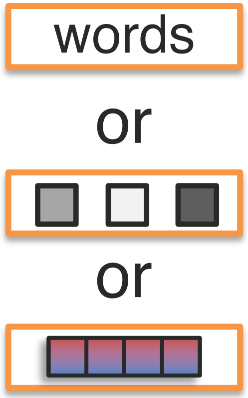
Reasoning

Definition: Combining knowledge, usually through multiple inferential steps, exploiting multimodal alignment and problem structure.

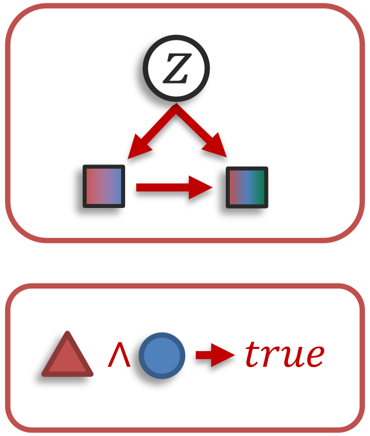
Ⓐ **Structure modeling**



Ⓑ **Intermediate concepts**



Ⓒ **Inference paradigm**



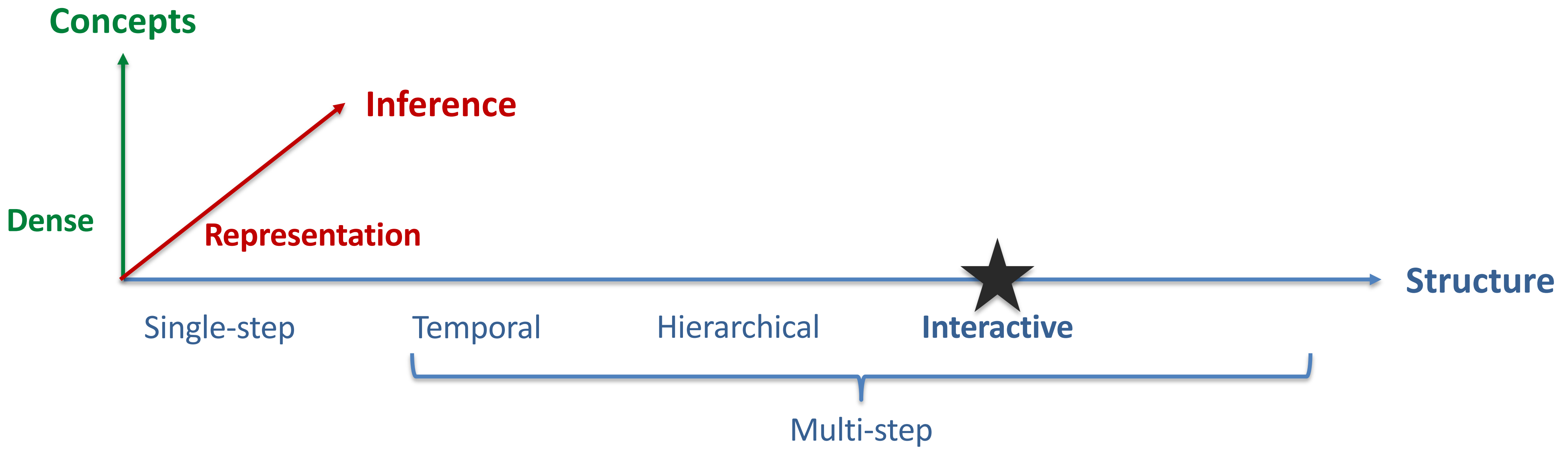
Ⓓ **External knowledge**

Roadmap for Next 3 Lectures

Definition: Combining knowledge, usually through multiple inferential steps, exploiting multimodal alignment and problem structure.

	(A) Structure modeling	(B) Intermediate concepts	(C) Inference paradigm	(D) External knowledge
Last Thursday	Temporal Hierarchical	Continuous		
Today	Interactive Discovery			
Thursday		Discrete	Causal Logical	Knowledge Commonsense

Sub-Challenge 3a: Structure Modeling

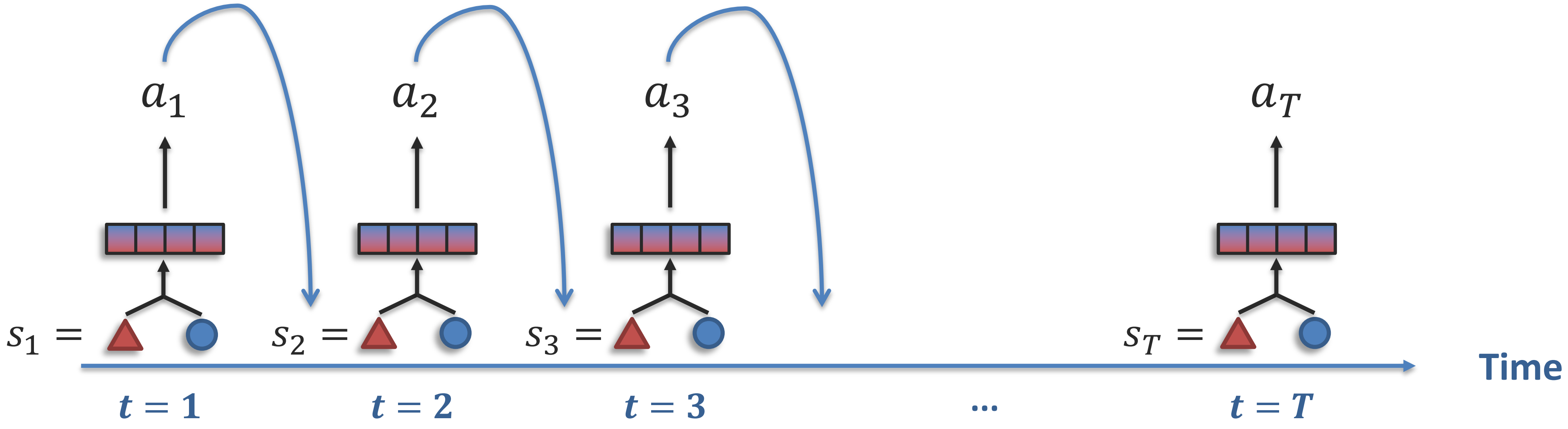


Interactive Structure

Structure defined through interactive environment

Main difference from temporal - actions taken at previous time steps affect future states

Integrates multimodality into the reinforcement learning framework

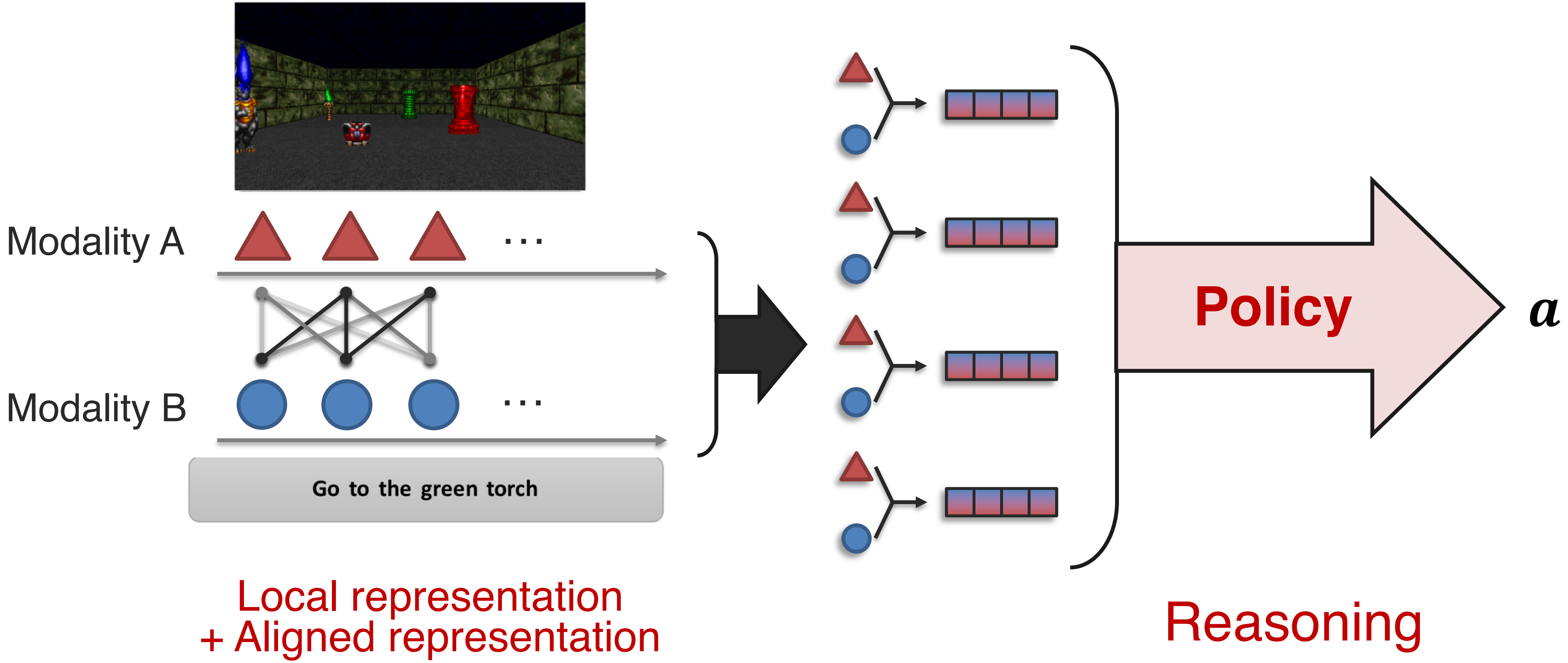


[Luketina et al., A Survey of Reinforcement Learning Informed by Natural Language. IJCAI 2019]

Interactive Structure

Structure defined through interactive environment

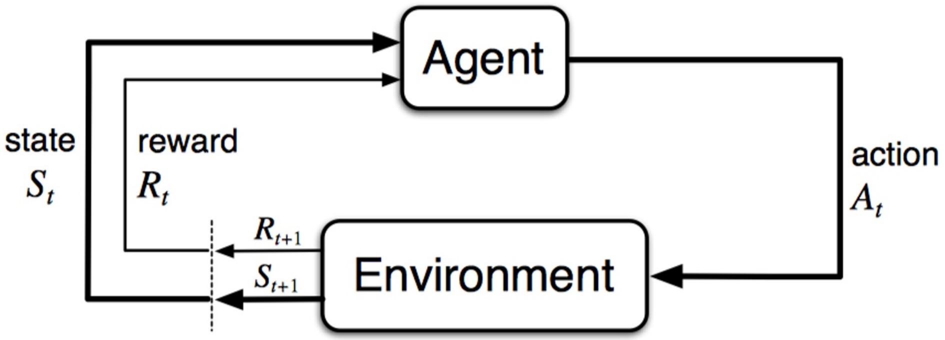
Main difference from temporal - actions taken at previous time steps affect future states

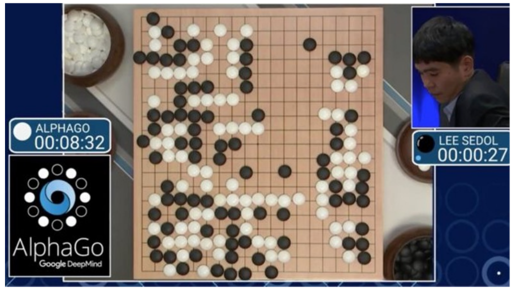


Learning a Policy – RL basics

Reinforcement learning

- Introduction to RL
- Markov Decision Processes (MDPs)
- Solving known MDPs using value and policy iteration
- Solving unknown MDPs using function approximation and Q-learning

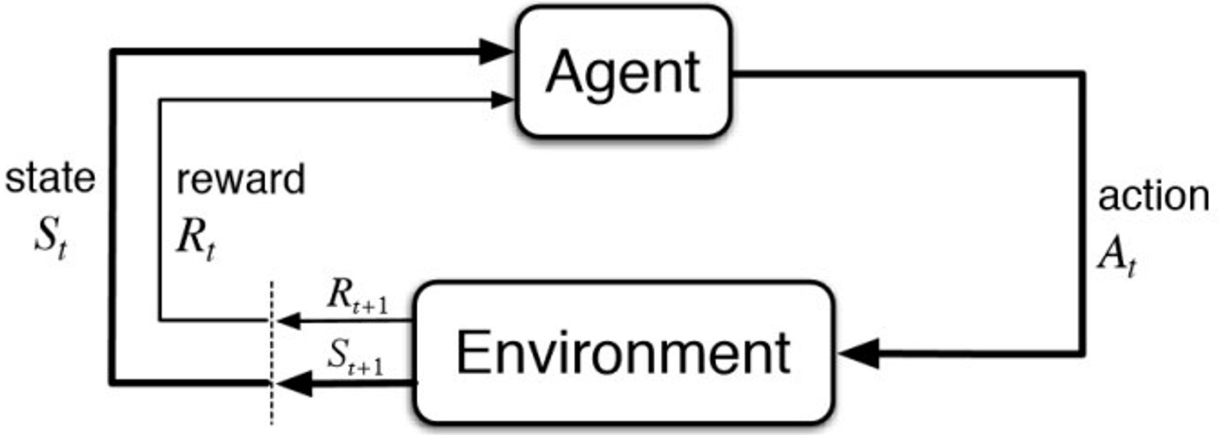




Learning a Policy – RL basics

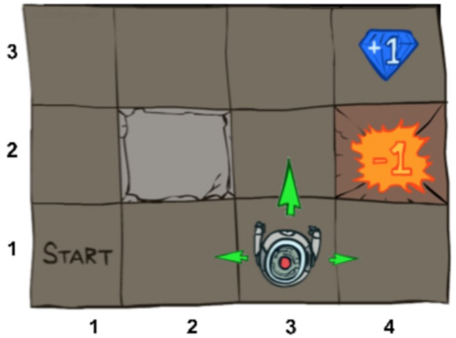
An MDP is defined by:

- Set of states S.
- Set of actions A.
- Transition function $P(s'|s,a)$.
- Reward function $r(s,a,s')$.
- Start state s_0.
- Discount factor γ.
- Horizon H.

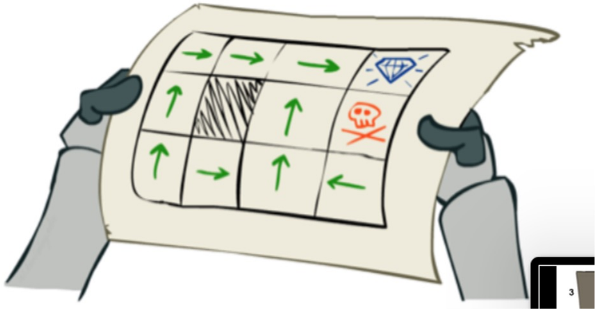


Return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$



π:



Policy: $\pi(a|s) = \Pr(A_t = a | S_t = s) \quad \forall t$

Goal: $\arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t R_t | \pi\right]$

RL vs Supervised Learning

Reinforcement Learning

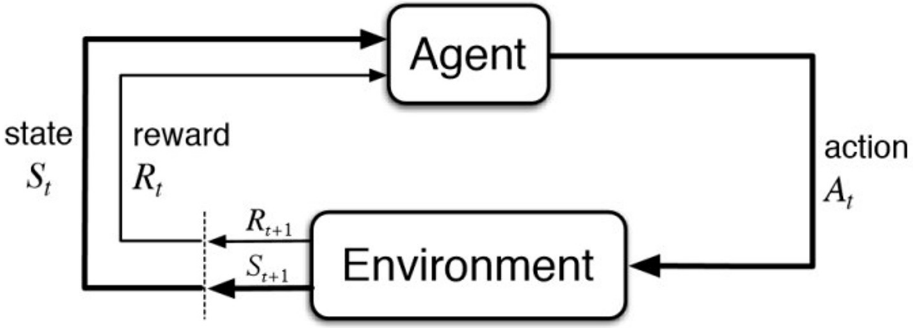
- Sequential decision making
- Maximize cumulative reward
- Sparse rewards
- Environment maybe unknown

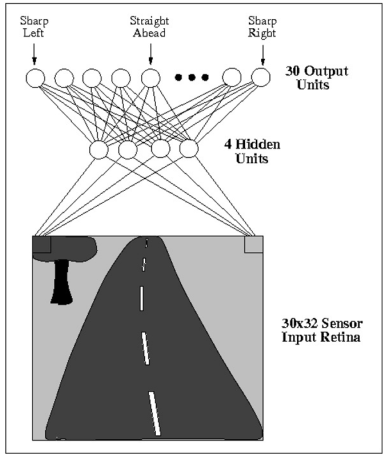
Supervised Learning

- One-step decision making
- Maximize immediate reward
- Dense supervision
- Environment always known

Intersection Between RL and Supervised Learning

Imitation learning





Obtain expert trajectories (e.g. human driver/video demonstrations):

$$s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, \ldots$$

Perform supervised learning by predicting expert action

$$D = \{(s_0, a_0^*), (s_1, a_1^*), (s_2, a_2^*), \ldots\}$$

But: distribution mismatch between training and testing

Hard to recover from sub-optimal states

Sometimes not safe/possible to collect expert trajectories

State and Action Value Functions

Definitions

- Definition**:** the **state-value function** $V^{\pi}(s)$ of an MDP is the expected return starting from state s, and following policy

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[G_t | S_t = s\right]$$

Captures long term reward

- Definition: the **action-value function** $Q^{\pi}(s,a)$ is the expected return starting from state s, taking action a, and then following policy

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[G_t | S_t = s, A_t = a\right]$$

Captures long term reward

Optimal State and Action Value Functions

Definitions

- Definition: the **optimal state-value function** $V^*(s)$ is the maximum value function over all policies

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

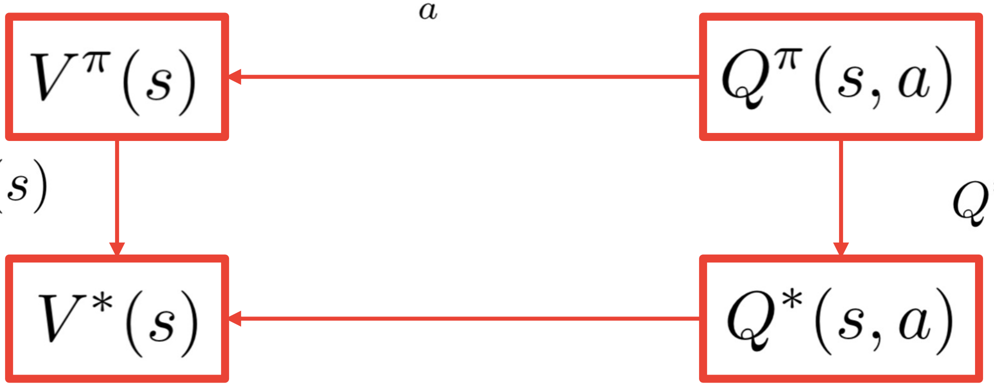
- Definition: the **optimal action-value function** $Q^*(s, a)$ is the maximum action-value function over all policies

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a)$$

Relationships Between State and Action Values

State value functions **Action value functions**

$$V^\pi(s) = \sum_a \pi(a|s) Q^\pi(s,a)$$



$$V^*(s) = \max_\pi V^\pi(s)$$

$$Q^*(s,a) = \max_\pi Q^\pi(s,a)$$

$$V^*(s) = \max_a Q^*(s,a)$$

Obtaining the Optimal Policy

Optimal policy can be found by maximizing over Q*(s,a)

$$\pi^*(a|s) = \begin{cases} 1, & \text{if } a = \arg\max_a \; Q^*(s,a) \\ 0, & \text{else} \end{cases}$$

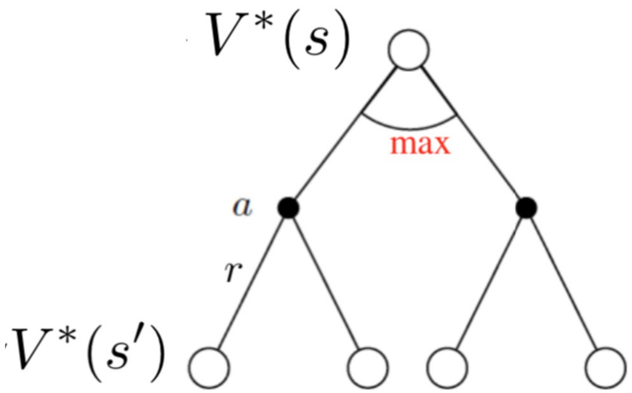
Optimal policy can also be found by maximizing over V*(s')
with **one-step look ahead**

$$\pi^*(a|s) = \begin{cases} 1, & \text{if } a = \arg\max_a \mathbb{E}_{s'} \left[r(s,a,s') + \gamma V^*(s') \right] \\ 0, & \text{else} \end{cases}$$

$$\pi^*(a|s) = \begin{cases} 1, & \text{if } a = \arg\max_a \left[\sum_{s'} p(s'|s,a)(r(s,a,s') + \gamma V^*(s')) \right] \\ 0, & \text{else} \end{cases}$$

Bellman Optimality for State Value Functions

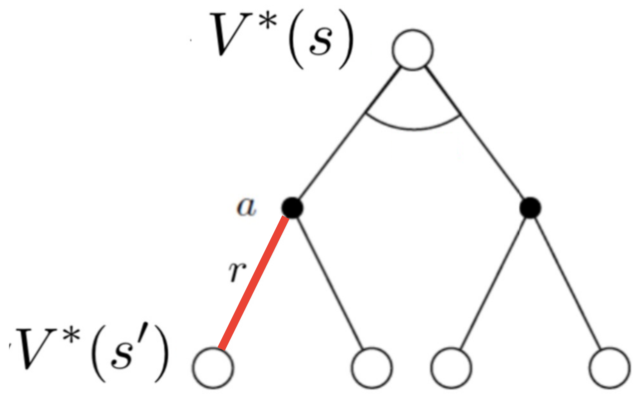
Recursive definition



$$V^*(s) = \max_a Q^*(s, a)$$

Bellman Optimality for State Value Functions

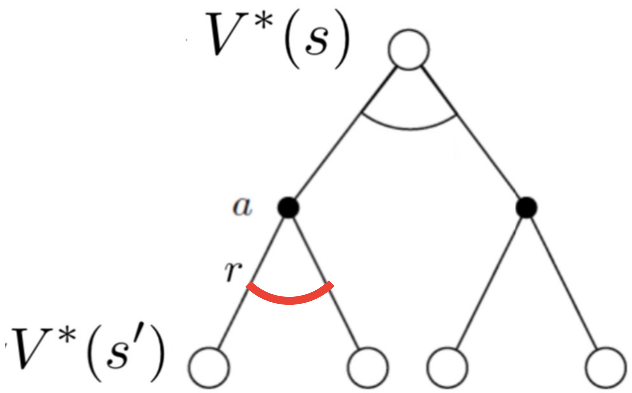
Recursive definition



$$V^*(s) = \max_a Q^*(s,a)$$
$$= \max_a \mathbb{E}_{s'}\left[r(s,a,s') + \gamma V^*(s')\right]$$

Bellman Optimality for State Value Functions

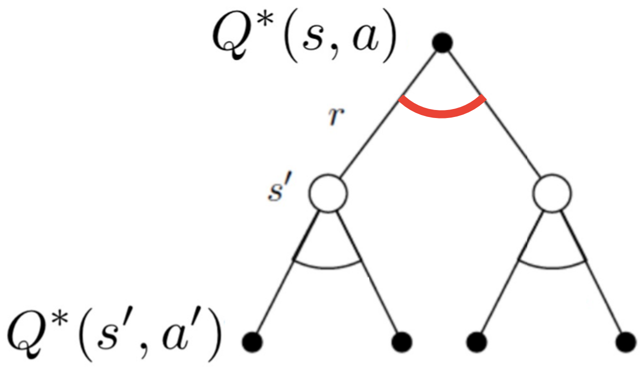
Recursive definition



$$
\begin{aligned}
V^*(s) &= \max_a Q^*(s,a) \\
&= \max_a \mathbb{E}_{s'}\left[r(s,a,s') + \gamma V^*(s')\right] \\
&= \max_a \left[\sum_{s'} p(s'|s,a)(r(s,a,s') + \gamma V^*(s'))\right]
\end{aligned}
$$

Bellman Optimality for Action Value Functions

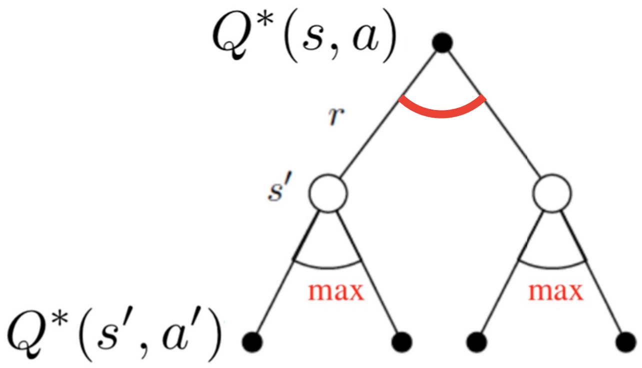
Recursive definition



$$Q^*(s,a) = \mathbb{E}_{s'}\left[r(s,a,s') + \gamma V^*(s')\right]$$

Bellman Optimality for Action Value Functions

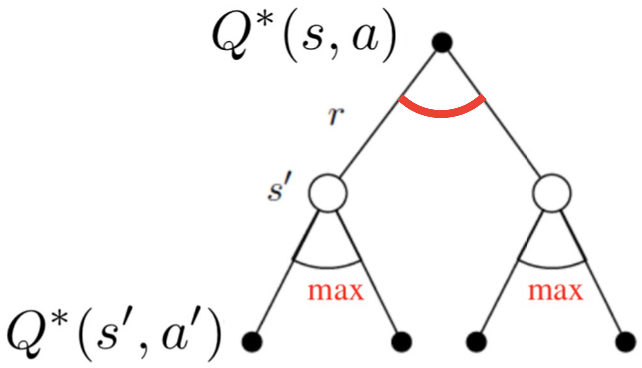
Recursive definition



$$Q^*(s,a) = \mathbb{E}_{s'}\left[r(s,a,s') + \gamma V^*(s')\right]$$
$$= \mathbb{E}_{s'}\left[r(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right]$$

Bellman Optimality for Action Value Functions

Recursive definition



$$Q^*(s,a) = \mathbb{E}_{s'}\left[r(s,a,s') + \gamma V^*(s')\right]$$
$$= \mathbb{E}_{s'}\left[r(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right]$$
$$= \sum_{s'} p(s'|s,a)\left(r(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right)$$

Solving the Bellman Optimality Equations

Recursive definition

$$V^*(s) = \max_a \left[\sum_{s'} p(s'|s,a)(r(s,a,s') + \gamma V^*(s')) \right]$$

Solve by iterative methods

$$V^*_{[k+1]}(s) = \max_a \left[\sum_{s'} p(s'|s,a)(r(s,a,s') + \gamma V^*_{[k]}(s')) \right]$$

[Slides from Fragkiadaki, 10-703 CMU]

Value Iteration

Algorithm:

Start with $V_0^*(s) = 0$ for all s.

For k = 1, … , H:

For all states s in S:

$$V_k^*(s) \leftarrow \max_a \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma V_{k-1}^*(s') \right)$$

[Slides from Fragkiadaki, 10-703 CMU]

Value Iteration

Algorithm:

Start with $V_0^*(s) = 0$ for all s.

For k = 1, … , H:

For all states s in S:

$$V_k^*(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$$

$$\pi_k^*(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$$

Find the best action according to one-step look ahead

This is called a value update or Bellman update/back-up

Repeat until policy converges. Guaranteed to converge to optimal policy.

[Slides from Fragkiadaki, 10-703 CMU]

Q-Value Iteration

$Q^*(s, a)$ = expected utility starting in s, taking action a, and (thereafter) acting optimally

Bellman Equation:

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

Q-Value Iteration:

$$Q^*_{k+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q^*_k(s',a'))$$

[Slides from Fragkiadaki, 10-703 CMU]

Summary: Exact Methods

Fully known MDP
states
transitions
rewards

→ Bellman optimality equations: $Q^*(s, a)$ — **Q-value iteration**; $V^*(s)$ — **Value iteration**

→ Bellman expectation equations: $Q^\pi(s, a)$ — **Q-policy iteration**; $V^\pi(s)$ — **Policy iteration**

Repeat until policy converges. Guaranteed to converge to optimal policy.

Limitations:
Iterate over and storage for all states and actions: requires small, discrete state and action space
Update equations require fully observable MDP and known transitions

Unknown MDPs?

$Q^*(s, a)$ = expected utility starting in s, taking action a, and (thereafter) acting optimally

Bellman Equation:

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

Q-Value Iteration:

$$Q^*_{k+1}(s,a) \leftarrow \boxed{\sum_{s'} P(s'|s,a)}(R(s,a,s') + \gamma \max_{a'} Q^*_k(s',a'))$$

This is problematic when do not know the transitions

[Slides from Fragkiadaki, 10-703 CMU]

Tabular Q-learning

- Q-value iteration: $Q_{k+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q_k(s',a'))$
- Rewrite as expectation: $Q_{k+1} \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$

[Slides from Fragkiadaki, 10-703 CMU]

Tabular Q-learning

- Q-value iteration: $Q_{k+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q_k(s',a'))$
- Rewrite as expectation: $Q_{k+1} \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$
- (Tabular) Q-Learning: replace expectation by samples
  - For an state-action pair (s,a), receive: $s' \sim P(s'|s,a)$ simulation and exploration
  - Consider your old estimate: $Q_k(s,a)$
  - Consider your new sample estimate:

$$\text{target}(s') = r(s,a,s') + \gamma \max_{a'} Q_k(s',a')$$

$$\text{error}(s') = \left(r(s,a,s') + \gamma \max_{a'} Q_k(s',a') - Q_k(s,a) \right)$$

[Slides from Fragkiadaki, 10-703 CMU]

Tabular Q-learning

learning rate
↓

$$Q_{k+1}(s,a) = Q_k(s,a) + \alpha \ \mathrm{error}(s')$$
$$= Q_k(s,a) + \alpha \left(r(s,a,s') + \gamma \max_{a'} Q_k(s',a') - Q_k(s,a) \right)$$

Key idea: implicitly estimate the transitions via simulation

Tabular Q-learning

Bellman optimality

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right]$$

Algorithm:

Start with $Q_0(s,a)$ for all s, a.

Get initial state s

For k = 1, 2, … till convergence

Sample action a, get next state s’

If s’ is terminal:

$$\text{target} = r(s,a,s')$$

Sample new initial state s’

else:

$$\text{target} = r(s,a,s') + \gamma \max_{a'} Q_k(s',a')$$

$$Q_{k+1}(s,a) = Q_k(s,a) + \alpha\left(r(s,a,s') + \gamma \max_{a'} Q_k(s',a') - Q_k(s,a)\right)$$

$$s \leftarrow s'$$

[Slides from Fragkiadaki, 10-703 CMU]

Tabular Q-learning

Algorithm:

Start with $Q_0(s, a)$ for all s, a.

Get initial state s

For k = 1, 2, … till convergence

Sample action a, get next state s’

If s’ is terminal:

$$\text{target} = r(s, a, s')$$

Sample new initial state s’

else:

$$\text{target} = r(s, a, s') + \gamma \max_{a'} Q_k(s', a')$$

$$Q_{k+1}(s, a) = Q_k(s, a) + \alpha \left(r(s, a, s') + \gamma \max_{a'} Q_k(s', a') - Q_k(s, a) \right)$$

$$s \leftarrow s'$$

- Choose random actions?
- Choose action that maximizes $Q_k(s, a)$ (i.e. greedily)?
- ε-Greedy: choose random action with prob. ε, otherwise choose action greedily

[Slides from Fragkiadaki, 10-703 CMU]

Exploration and Exploitation

Poor estimates of Q(s,a) at the start:

Bad initial estimates in the first few cases can drive policy into sub-optimal region, and never explore further.

$$\pi(s) = \begin{cases} \max_a \hat{Q}(s,a) & \text{with probability } 1-\epsilon \\ \text{random action} & \text{otherwise} \end{cases}$$

Gradually decrease epsilon as policy is learned.

[Slides from Fragkiadaki, 10-703 CMU]

Tabular Q-learning

Algorithm:

Start with $Q_0(s, a)$ for all s, a.

Get initial state s

For k = 1, 2, … till convergence

Sample action a, get next state s'

If s' is terminal:

$$\text{target} = r(s, a, s')$$

Sample new initial state s'

else:

$$\text{target} = r(s, a, s') + \gamma \max_{a'} Q_k(s', a')$$

$$Q_{k+1}(s, a) = Q_k(s, a) + \alpha \left(r(s, a, s') + \gamma \max_{a'} Q_k(s', a') - Q_k(s, a) \right)$$

$$s \leftarrow s'$$

Tabular: keep a |S| x |A| table of Q(s,a)

Still requires small and discrete state and action space

How can we generalize to unseen states?

- ε-Greedy: choose random action with prob. ε, otherwise choose action greedily

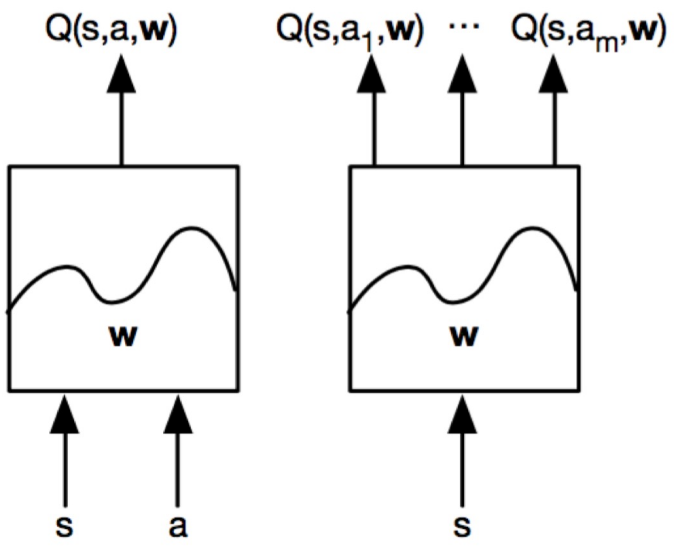
[Slides from Fragkiadaki, 10-703 CMU]

Deep Q-learning

Q-learning with function approximation to **extract informative features** from **high-dimensional** input states.

Represent value function by Q network with weights **w**

$$Q(s, a, \mathbf{w}) \approx Q^*(s, a)$$



+ high-dimensional, continuous states
+ generalization to new states

[Slides from Fragkiadaki, 10-703 CMU]

Deep Q-learning

- Optimal Q-values should obey Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q(s',a')^* \mid s,a\right]$$

- Treat right-hand $r + \gamma \max_{a'} Q(s',a',\mathbf{w})$ as as a target
- Minimize MSE loss by stochastic gradient descent

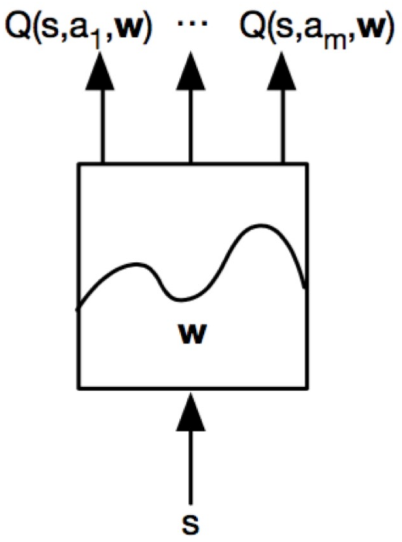
$$l = \left(r + \gamma \max_{a} Q(s',a',\mathbf{w}) - Q(s,a,\mathbf{w})\right)^2$$

Deep Q-learning Challenges

- Minimize MSE loss by stochastic gradient descent

$$l = \left(r + \gamma \max_a Q(s', a', \mathbf{w}) - Q(s, a, \mathbf{w}) \right)^2$$

- Converges to Q* using table lookup representation
- But diverges using neural networks due to:
  - Correlations between samples
  - Non-stationary targets



Deep Q-learning: Experience Replay

- To remove correlations, build data-set from agent's own experience

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
$\ldots$
$s_t, a_t, r_{t+1}, s_{t+1}$

$\rightarrow \quad s, a, r, s'$

exploration, epsilon greedy is important!

- Sample random mini-batch of transitions (s,a,r,s') from **D**

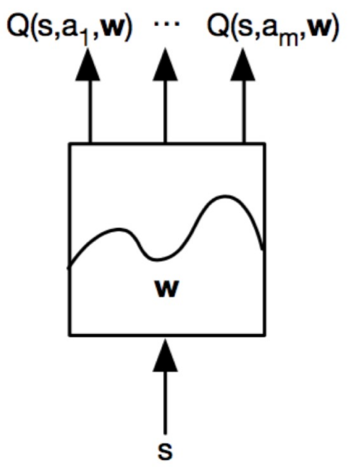
Deep Q-learning: Fixed Q-targets

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
$\ldots$
$s_t, a_t, r_{t+1}, s_{t+1}$

- Sample random mini-batch of transitions (s,a,r,s') from D
- Compute Q-learning targets w.r.t. old fixed parameters **w-**

- Optimize MSE between Q-network and Q-learning targets

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a'; w_i^-)}_{\text{Q-learning target}} - \underbrace{Q(s, a; w_i)}_{\text{Q-network}}\right)^2\right]$$



- Use stochastic gradient descent
- Update **w-** with updated **w** every ~1000 iterations

Value-based and Policy-based RL

- Value Based
  - Learned Value Function
  - Implicit policy (e.g. ε-greedy)

State value functions

$$V^{\pi}(s)$$

$$V^{*}(s)$$

Action value functions

$$Q^{\pi}(s,a)$$

$$Q^{*}(s,a)$$

$$\pi^*(a|s) = \begin{cases} 1-\epsilon, & \text{if } a = \arg\max_a \mathbb{E}_{s'}\left[r(s,a,s') + \gamma V^*(s')\right] \\ \epsilon, & \text{else} \end{cases}$$

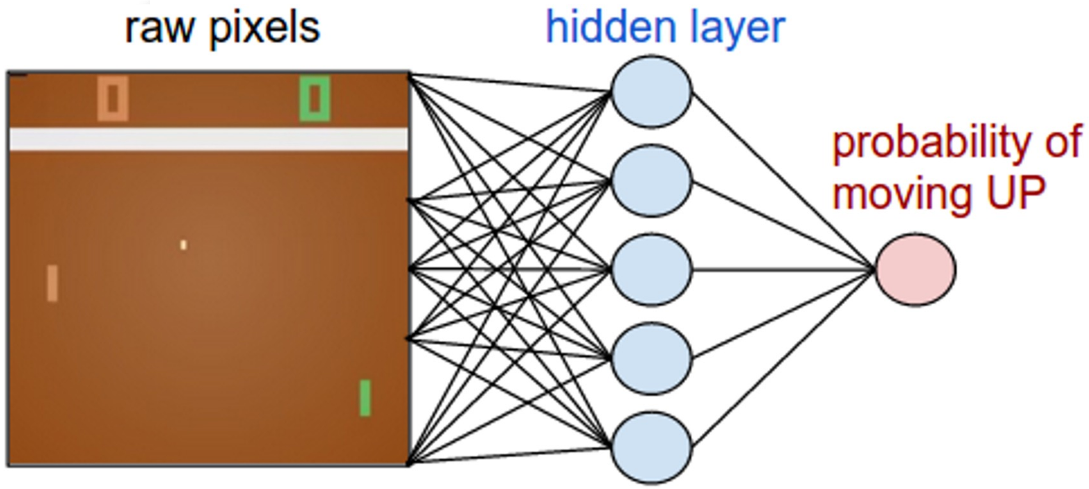
$$\pi^*(a|s) = \begin{cases} 1-\epsilon, & \text{if } a = \arg\max_a \; Q^*(s,a) \\ \epsilon, & \text{else} \end{cases}$$

[Slides from Fragkiadaki, 10-703 CMU]

Value-based and Policy-based RL

- Value Based
  - Learned Value Function
  - Implicit policy (e.g. ε-greedy)
- Policy Based
  - No Value Function
  - Learned Policy

$$\pi_\theta(s, a) = \mathbb{P}\left[a \mid s, \theta\right]$$



[Slides from Fragkiadaki, 10-703 CMU]

Can we Directly Learn the Policy?

- Often π can be simpler than Q or V
  - E.g., robotic grasp

Q(s,a) and V(s) very high-dimensional
But policy could be just 'open/close hand'

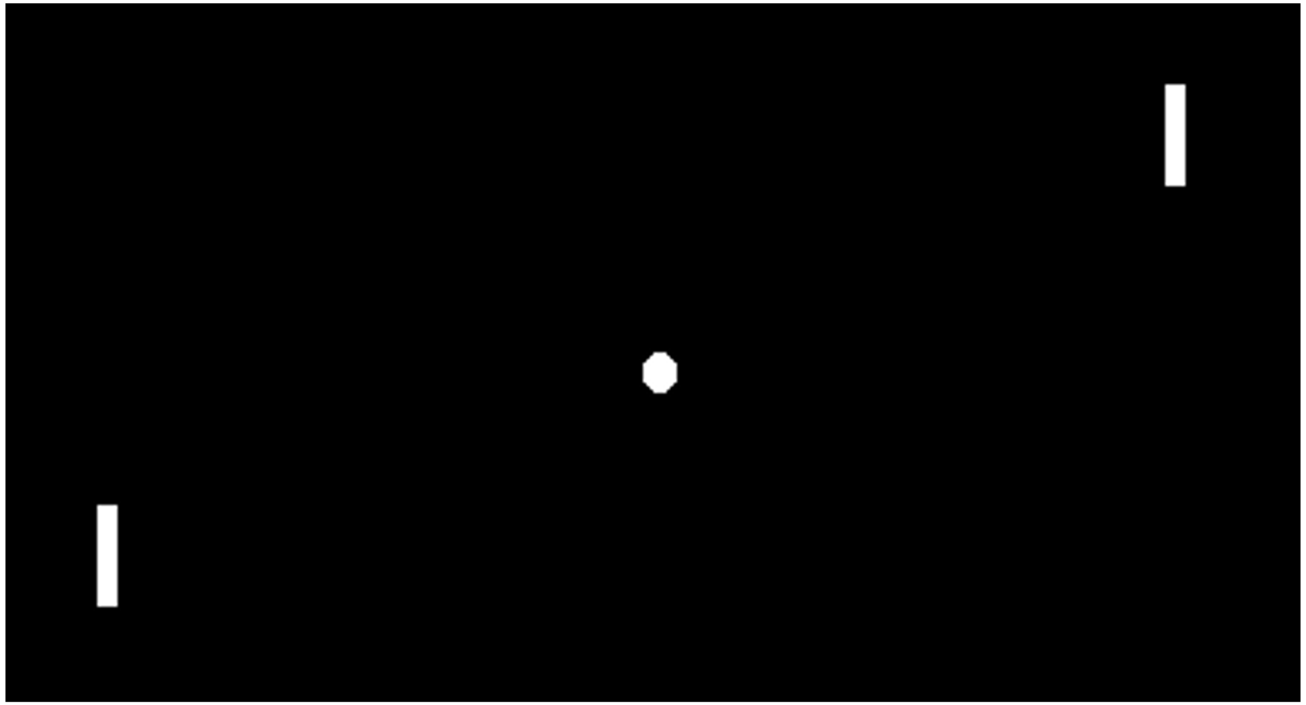
- V: doesn't prescribe actions
  - Would need dynamics model (+ compute 1 Bellman back-up)
- Q: need to be able to efficiently solve $\arg\max_a \; Q^*(s,a)$
  - Challenge for continuous / high-dimensional action spaces

$$\pi^*(a|s) = \begin{cases} 1-\epsilon, & \text{if } a = \arg\max_a \mathbb{E}_{s'}\left[r(s,a,s') + \gamma V^*(s')\right] \\ \epsilon, & \text{else} \end{cases}$$

$$\pi^*(a|s) = \begin{cases} 1-\epsilon, & \text{if } a = \arg\max_a \; Q^*(s,a) \\ \epsilon, & \text{else} \end{cases}$$

[Slides from Fragkiadaki, 10-703 CMU]

Pong from Pixels

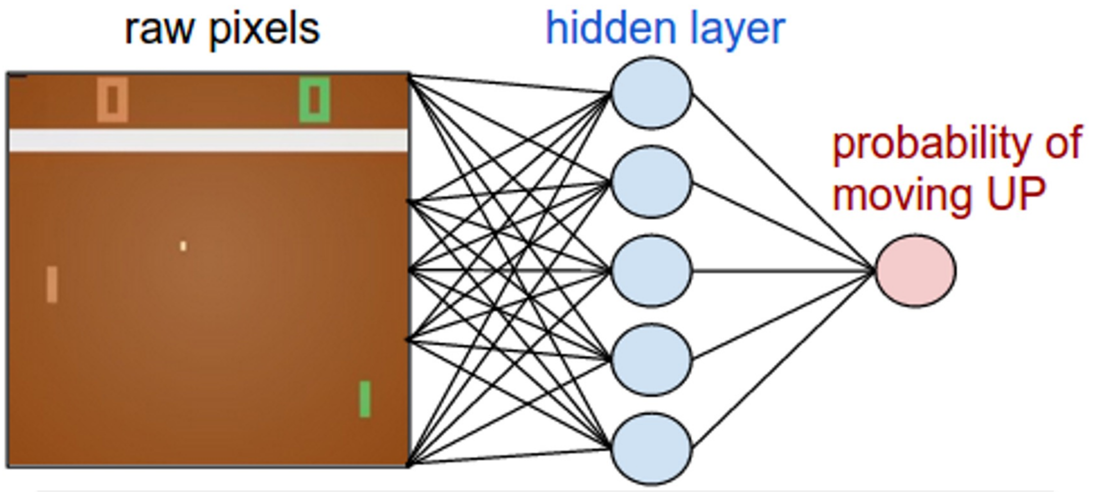


Pong from Pixels

e.g.,

height width

[80 x 80]
array of



Network sees +1 if it scored a point, and -1 if it was scored against.
How do we learn these parameters?

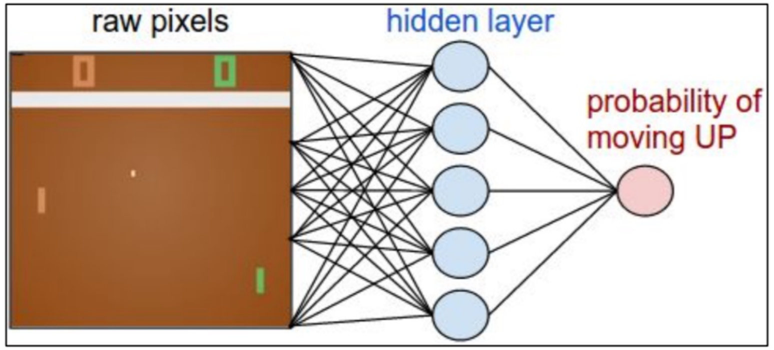
Pong from Pixels

Suppose we had the training labels…
(we know what to do in any state)

(x1,UP)
(x2,DOWN)
(x3,UP)
...

maximize:

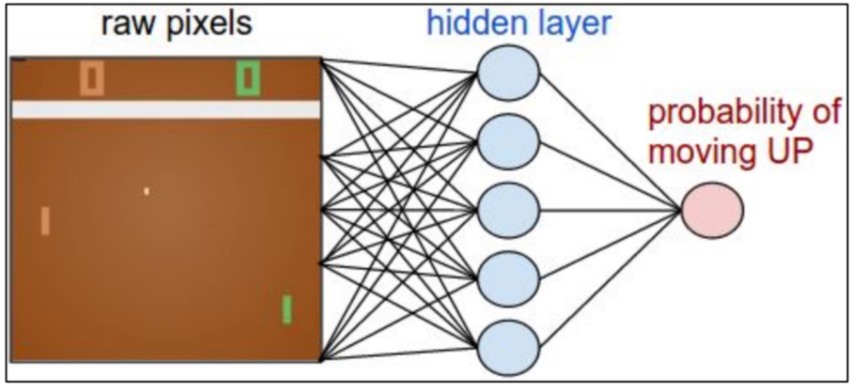
$$\sum_i \log p(y_i \mid x_i)$$



[Slides from Karpathy]

Pong from Pixels

Except, we don't have labels...

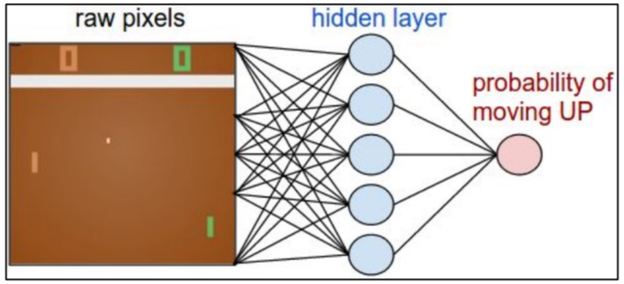


Should we go UP or DOWN?

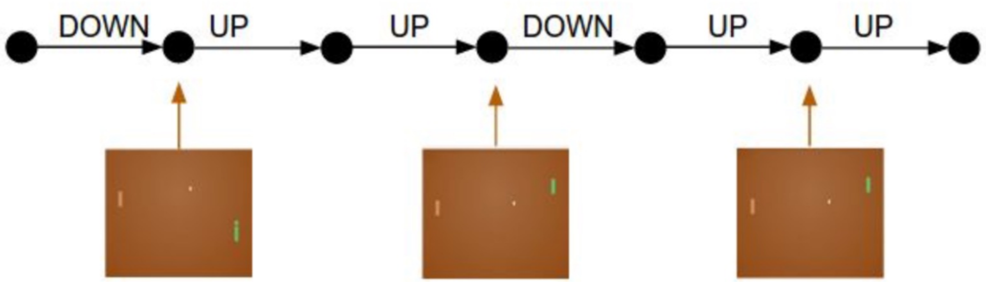
[Slides from Karpathy]

Pong from Pixels

Let’s just act according to our current policy...



Rollout the policy and collect an episode



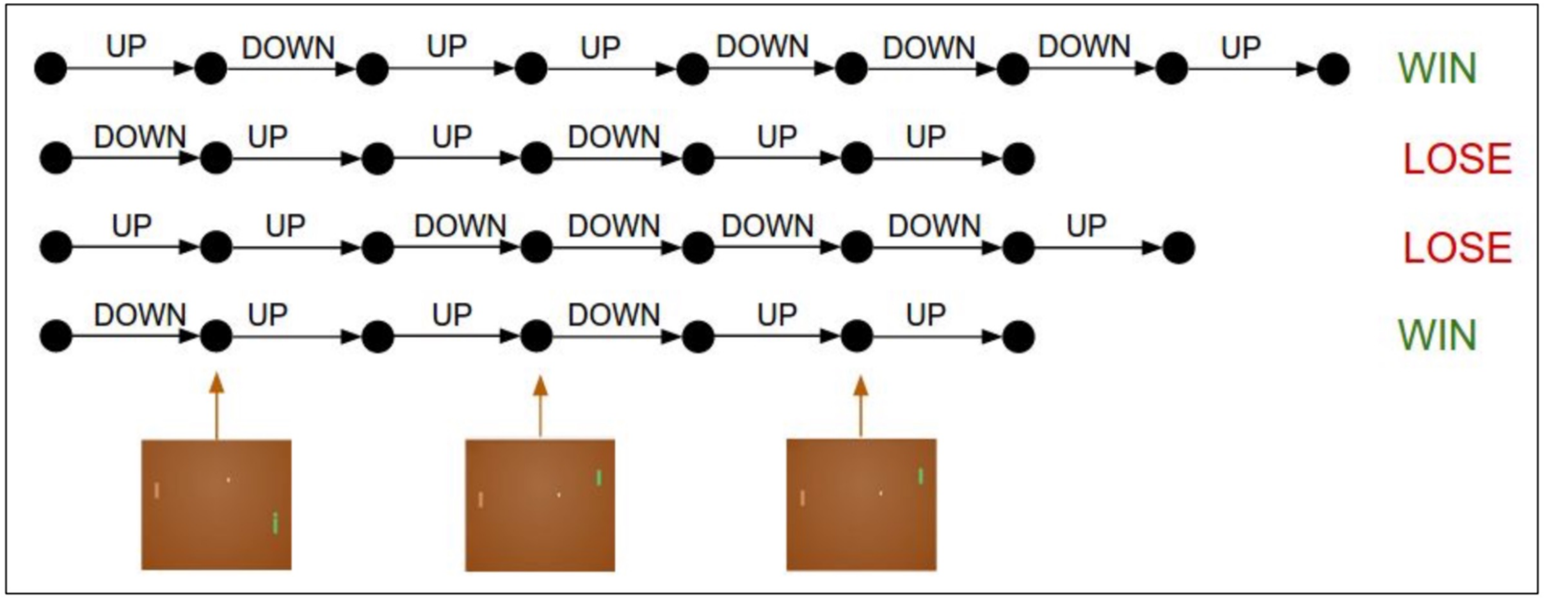
WIN

[Slides from Karpathy]

Pong from Pixels

Collect many rollouts...

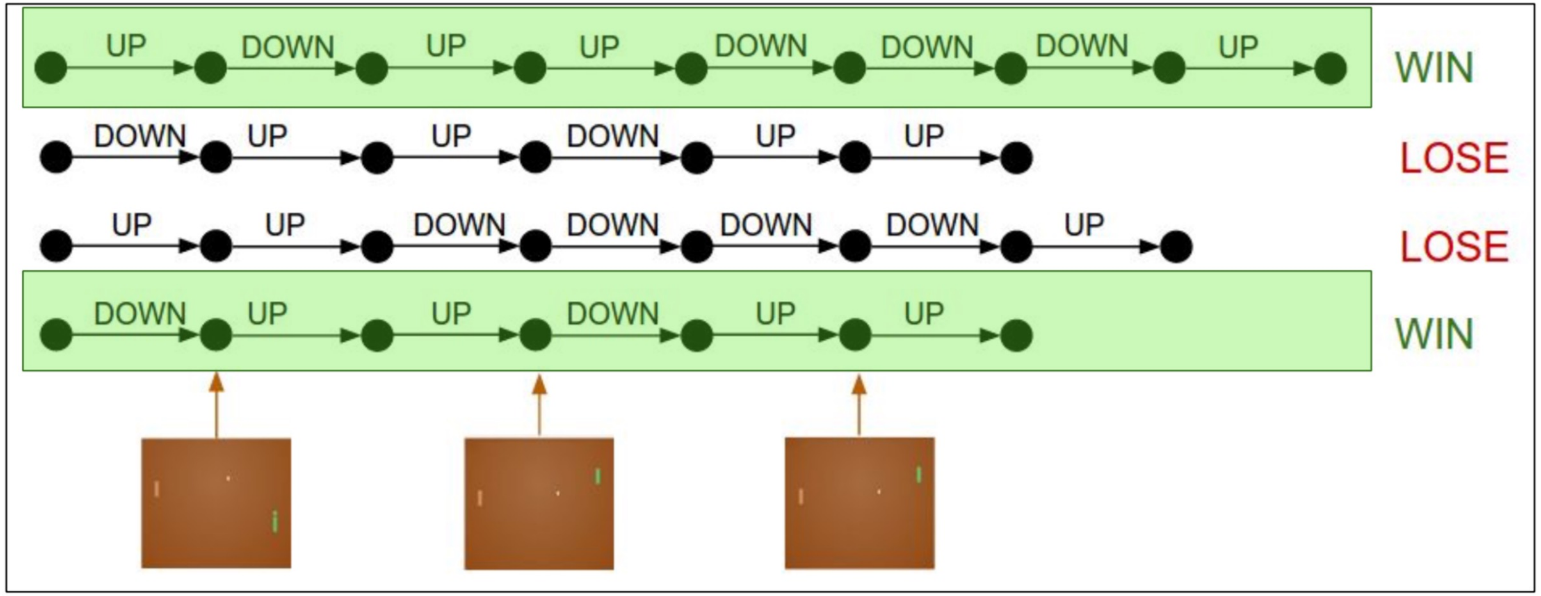
4 rollouts:



[Slides from Karpathy]

Pong from Pixels

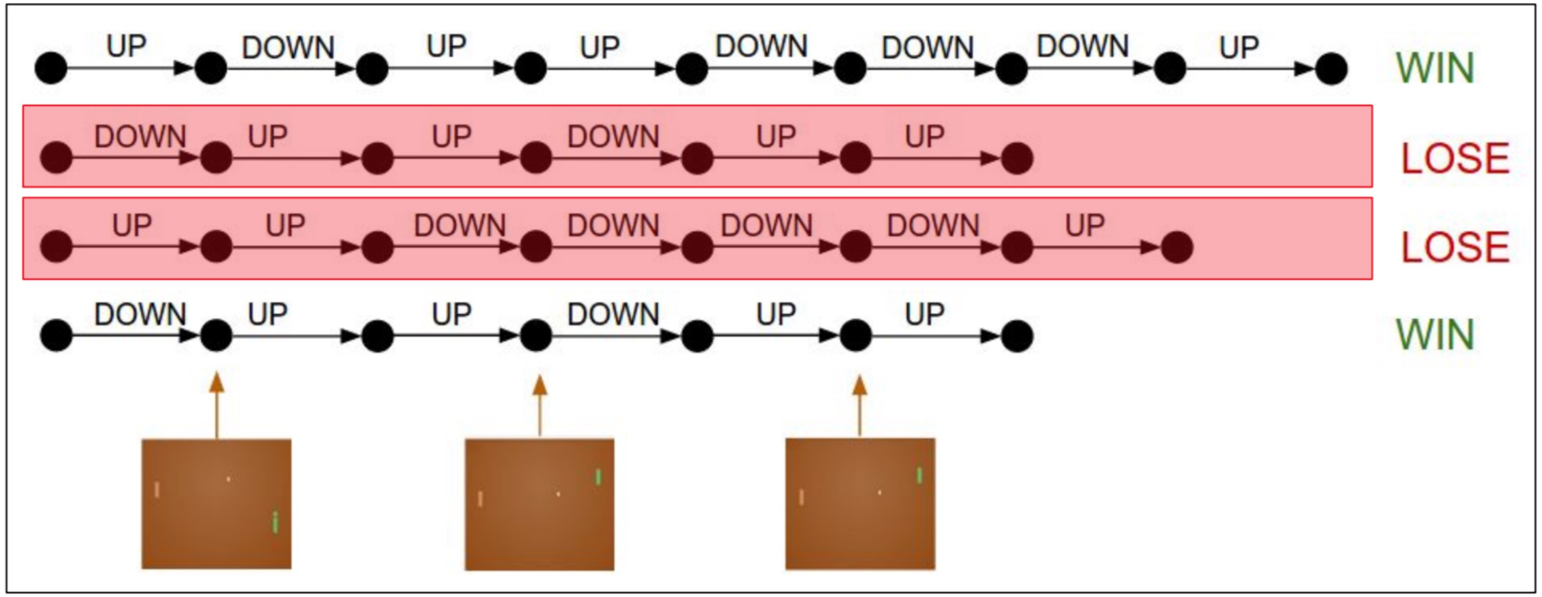
Not sure whatever we did here, but apparently it was good.



[Slides from Karpathy]

Pong from Pixels

Not sure whatever we did here, but it was bad.



[Slides from Karpathy]

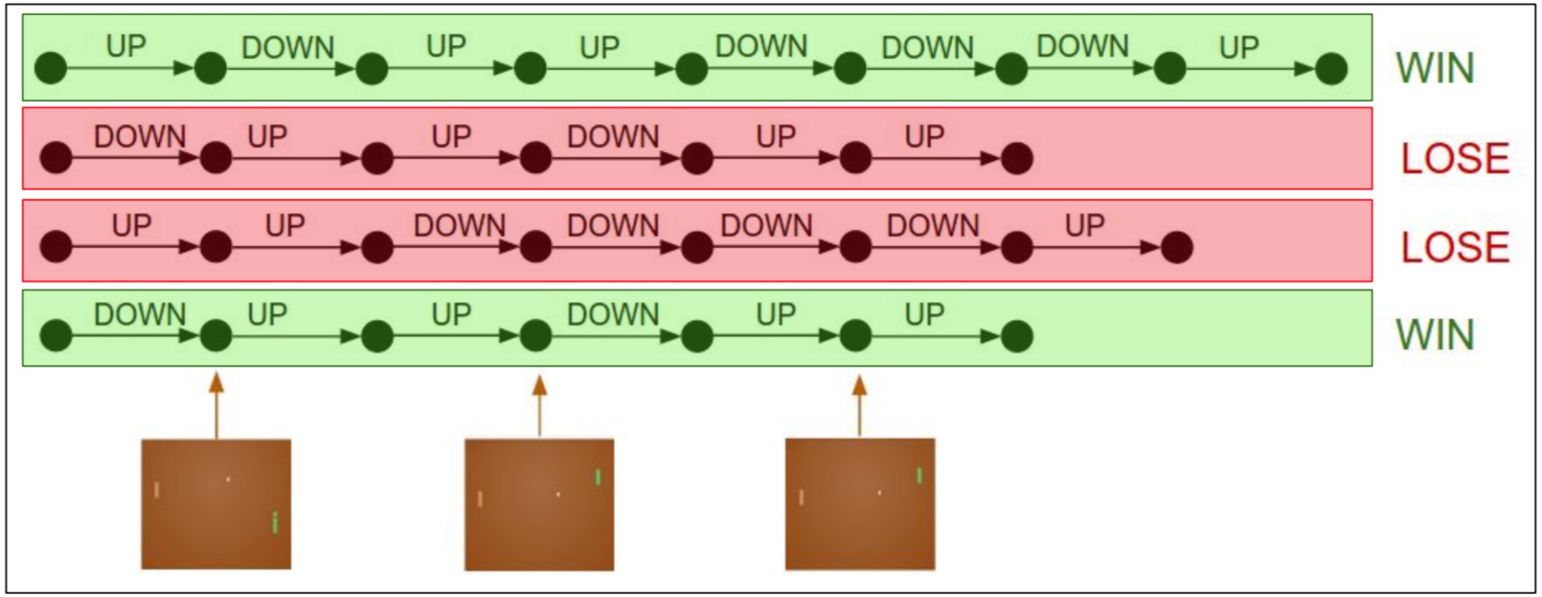
Pong from Pixels

Pretend every action we took here was the correct label.

maximize: $\log p(y_i \mid x_i)$

Pretend every action we took here was the wrong label.

maximize: $(-1) * \log p(y_i \mid x_i)$



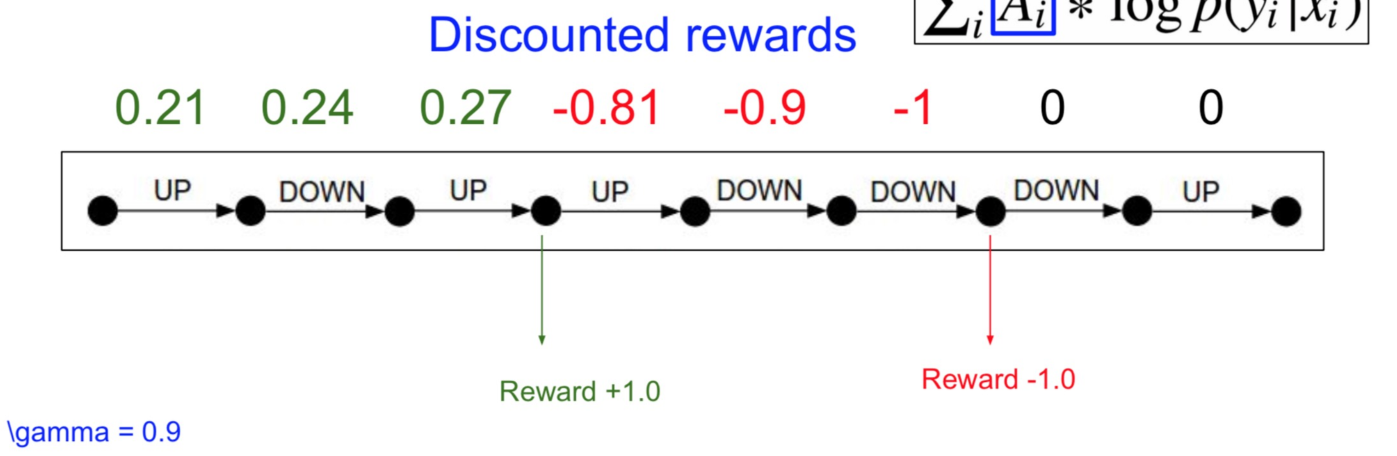
[Slides from Karpathy]

Pong from Pixels

Discounting

Blame each action assuming that its effects have exponentially decaying impact into the future.

$$\sum_i \boxed{A_i} * \log p(y_i | x_i)$$

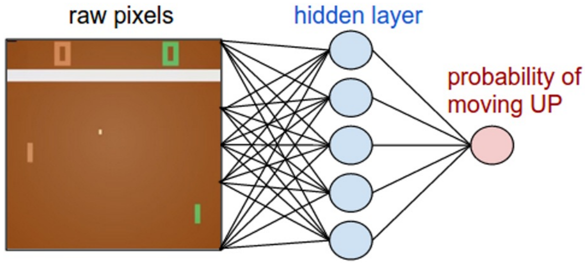


[Slides from Karpathy]

Pong from Pixels

$\pi(a \mid s)$

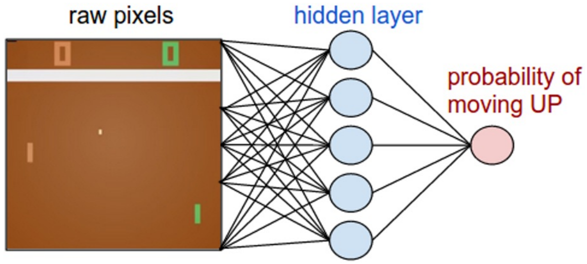
1. Initialize a policy network at random



[Slides from Karpathy]

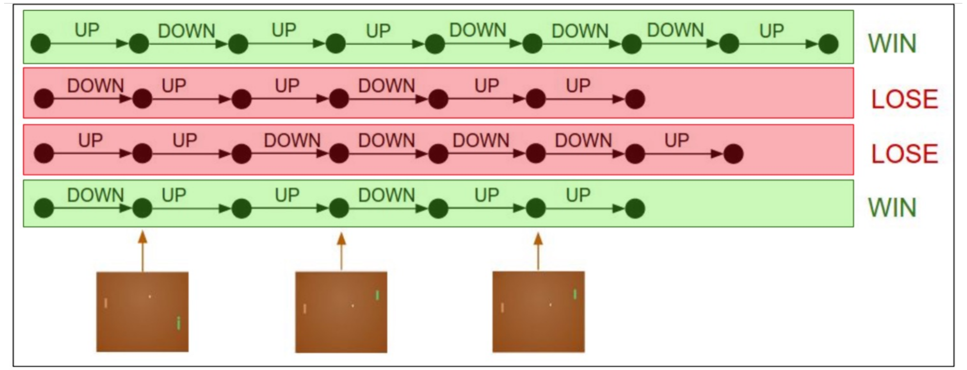
Pong from Pixels

$\pi(a \mid s)$




```
1.  Initialize a policy network at random
2.  Repeat Forever:
3.      Collect a bunch of rollouts with the policy
```

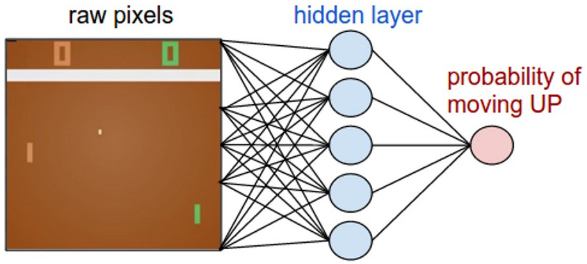
epsilon greedy!



[Slides from Karpathy]

Pong from Pixels

$\pi(a \mid s)$




```
1.  Initialize a policy network at random
2.  Repeat Forever:
3.      Collect a bunch of rollouts with the policy
4.      Increase the probability of actions that worked well
```

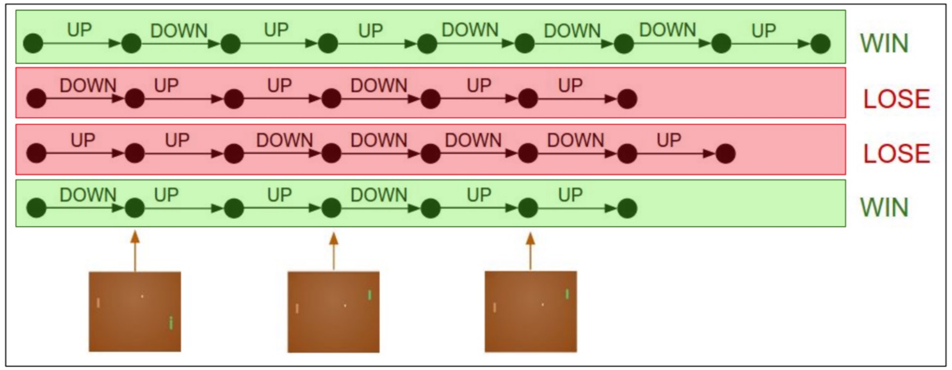
epsilon greedy!

Pretend every action we took here was the correct label.

maximize: $\log p(y_i \mid x_i)$

Pretend every action we took here was the wrong label.

maximize: $(-1) * \log p(y_i \mid x_i)$

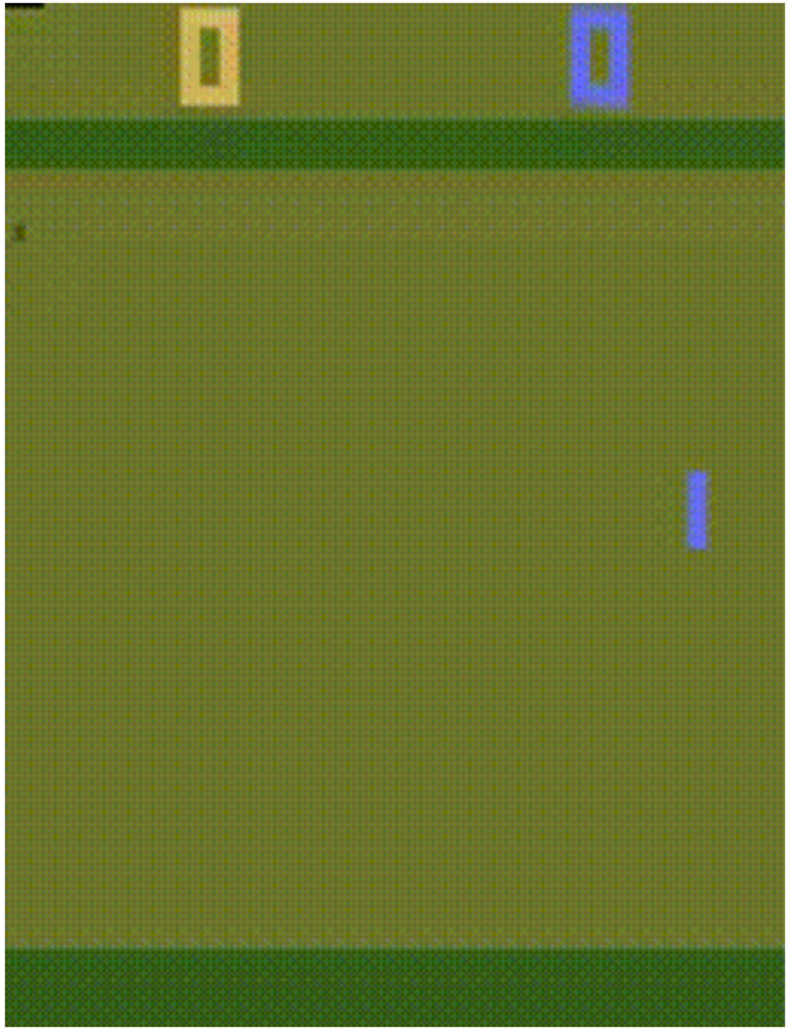


$$\sum_i A_i * \log p(y_i | x_i)$$

Does not require transition probabilities
Does not estimate Q(), V()
Predicts policy directly

[Slides from Karpathy]

Pong from Pixels



[Slides from Karpathy]

Policy Gradients

Why does this work?

```
1.  Initialize a policy network at random
2.  Repeat Forever:
3.      Collect a bunch of rollouts with the policy
4.      Increase the probability of actions that worked well
```

$$\sum_i A_i * \log p(y_i | x_i)$$

[Slides from Karpathy]

Policy Gradients

Formally, let's define a class of parameterized policies $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$

For each policy, define its value:

$$J(\theta) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi_\theta\right]$$

Policy Gradients

Writing in terms of trajectories $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$

Probability of a trajectory

$$p(\tau;\theta) = \pi_\theta(a_0|s_0)p(s_1|s_0,a_0)$$
$$\times\ \pi_\theta(a_1|s_1)p(s_2|s_1,a_1)$$
$$\times\ \pi_\theta(a_2|s_2)p(s_3|s_2,a_2)$$
$$\times\ \ldots$$
$$= \prod_{t\geq 0} p(s_{t+1}|s_t,a_t)\pi_\theta(a_t|s_t)$$

Reward of a trajectory

$$r(\tau) = \sum_{t\geq 0}\gamma^t r_t$$

$$J(\theta) = \mathbb{E}\left[\sum_{t\geq 0}\gamma^t r_t | \pi_\theta\right] = \mathbb{E}_{\tau\sim p(\tau;\theta)}\left[r(\tau)\right]$$

Policy Gradients

Formally, let's define a class of parameterized policies $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$

For each policy, define its value:

$$J(\theta) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi_\theta\right] = \mathbb{E}_{\tau\sim p(\tau;\theta)}\left[r(\tau)\right]$$

We want to find the optimal policy $\theta^* = \arg\max_\theta J(\theta)$

How can we do this?

Gradient ascent on policy parameters

REINFORCE Algorithm

Expected reward: $$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)} [r(\tau)]$$
$$= \int_\tau r(\tau)p(\tau;\theta)\ d\tau$$

REINFORCE Algorithm

Expected reward: $J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\right]$

$$= \int_{\tau} r(\tau) p(\tau;\theta)\ d\tau$$

$$p(\tau;\theta) = \prod_{t \geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

Now let's differentiate this: $\nabla_\theta J(\theta) = \int_{\tau} r(\tau) \nabla_\theta p(\tau;\theta)\ d\tau$

Intractable

REINFORCE Algorithm

Expected reward: $J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)} \left[r(\tau) \right]$

$$= \int_{\tau} r(\tau) p(\tau;\theta) \; d\tau$$

$$p(\tau;\theta) = \prod_{t \geq 0} p(s_{t+1}|s_t, a_t) \pi_\theta(a_t|s_t)$$

Now let's differentiate this: $\nabla_\theta J(\theta) = \int_{\tau} r(\tau) \nabla_\theta p(\tau;\theta) \; d\tau$

Intractable

However, we can use a nice trick: $\nabla_\theta p(\tau;\theta) = p(\tau;\theta) \dfrac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} = p(\tau;\theta) \nabla_\theta \log p(\tau;\theta)$

REINFORCE Algorithm

Expected reward: $J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\right]$

$$= \int_{\tau} r(\tau) p(\tau;\theta)\ d\tau$$

$$p(\tau;\theta) = \prod_{t \geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

Now let's differentiate this: $\nabla_\theta J(\theta) = \int_{\tau} r(\tau) \nabla_\theta p(\tau;\theta)\ d\tau$

Intractable

However, we can use a nice trick: $\nabla_\theta p(\tau;\theta) = p(\tau;\theta)\dfrac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} = p(\tau;\theta)\nabla_\theta \log p(\tau;\theta)$

If we inject this back:

$$\nabla_\theta J(\theta) = \int_{\tau} \left(r(\tau)\nabla_\theta \log p(\tau;\theta)\right) p(\tau;\theta)\ d\tau$$

$$= \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\nabla_\theta \log p(\tau;\theta)\right]$$

REINFORCE Algorithm

Can we compute these without knowing the transition probabilities?

We have: $$p(\tau;\theta) = \prod_{t \geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

REINFORCE Algorithm

Can we compute these without knowing the transition probabilities?

We have: $$p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

Thus: $$\log p(\tau;\theta) = \sum_{t\geq 0} (\log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t))$$

REINFORCE Algorithm

Can we compute these without knowing the transition probabilities?

We have: $$p(\tau;\theta) = \prod_{t \geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

Thus: $$\log p(\tau;\theta) = \sum_{t \geq 0} (\log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t))$$

And when differentiating: $$\nabla_\theta \log p(\tau;\theta) = \sum_{t \geq 0} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Doesn't depend on transition probabilities

REINFORCE Algorithm

Can we compute these without knowing the transition probabilities?

We have: $$p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

Thus: $$\log p(\tau;\theta) = \sum_{t\geq 0} (\log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t))$$

And when differentiating: $$\nabla_\theta \log p(\tau;\theta) = \sum_{t\geq 0} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Doesn't depend on transition probabilities

Therefore when sampling a trajectory, we can estimate gradients:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p(\tau;\theta)}\left[r(\tau)\nabla_\theta \log p(\tau;\theta)\right] \approx \sum_{t\geq 0} r(\tau)\nabla_\theta \log \pi_\theta(a_t|s_t)$$

Policy Gradients

Gradient estimator:

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} r(\tau)\nabla_\theta \log \pi_\theta(a_t|s_t)$$

Interpretation:

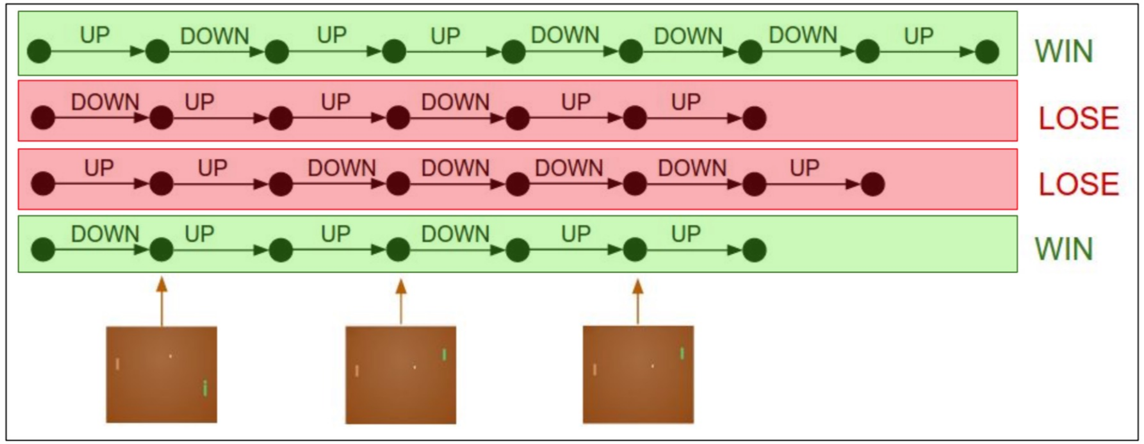
- If **r(trajectory)** is high, push up the probabilities of the actions seen
- If **r(trajectory)** is low, push down the probabilities of the actions seen

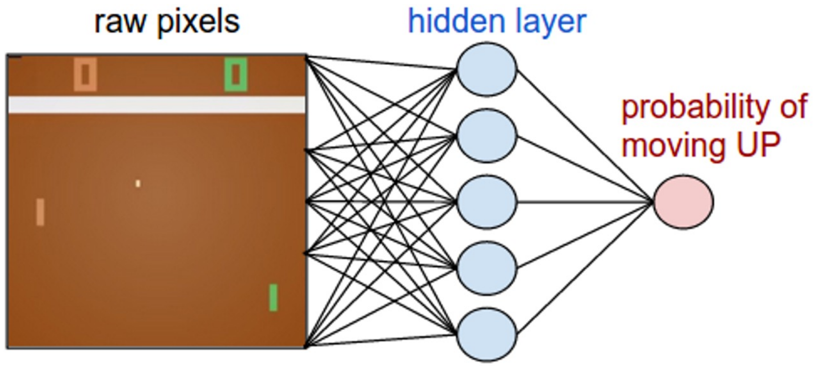
Pretend every action we took here was the correct label.

maximize: $\log p(y_i \mid x_i)$

Pretend every action we took here was the wrong label.

maximize: $(-1) * \log p(y_i \mid x_i)$





$$\sum_i A_i * \log p(y_i|x_i)$$

Policy Gradients

Gradient estimator: $$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} r(\tau)\nabla_\theta \log \pi_\theta(a_t|s_t)$$

Interpretation:

- If **r(trajectory)** is high, push up the probabilities of the actions seen
- If **r(trajectory)** is low, push down the probabilities of the actions seen

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta}), \forall a \in \mathcal{A}, s \in \mathcal{S}, \boldsymbol{\theta} \in \mathbb{R}^n$
Initialize policy weights $\boldsymbol{\theta}$
Repeat forever:
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$ following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  For each step of the episode $t = 0, \ldots, T-1$:
    $G_t \leftarrow$ return from step t
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G_t \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t,\boldsymbol{\theta})$

epsilon greedy

[Slides from Fragkiadaki, 10-703 CMU]

Policy Gradients

Gradient estimator: $$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

Interpretation:

- If **r(trajectory)** is high, push up the probabilities of the actions seen
- If **r(trajectory)** is low, push down the probabilities of the actions seen

Might seem simplistic to say that if a trajectory is good then all its actions were good. But in expectation, it averages out!

However, this also suffers from high variance because credit assignment is really hard - can we help this estimator?

[Slides from Fragkiadaki, 10-703 CMU]

Variance Reduction with a Baseline

Problem: The raw reward of a trajectory isn't necessarily meaningful. E.g. if all rewards are positive, you keep pushing up probabilities of all actions.

What is important then? Whether a reward is higher or lower than what you expect to get.

Idea: Introduce a baseline function dependent on the state, which gives us an estimator:

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} (r(\tau) - b(s_t)) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

e.g. exponential moving average of the rewards.

[Slides from Fragkiadaki, 10-703 CMU]

Actor-Critic Methods

A better baseline: want to push the probability of an action from a state, if this action was better than the expected value of what we should get from that state

Recall: **Q and V - action and state value functions!**

We are happy with an action **a** in a state **s** if **Q(s,a) - V(s)** is large.
Otherwise we are unhappy with an action if it's small.

Using this, we get the estimator:

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)\right) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

Actor-Critic Methods

Problem: we don't know Q and V - can we learn them?

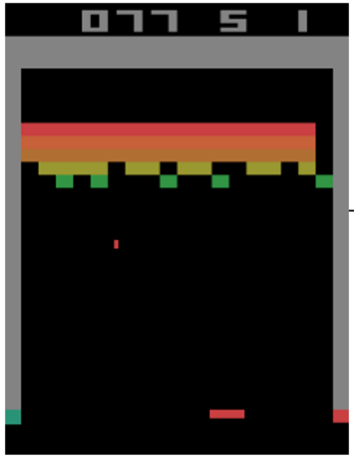
Yes, using Q-learning! We can combine Policy Gradients and Q-learning by training both an **actor** (the policy) and a **critic** (the Q function)

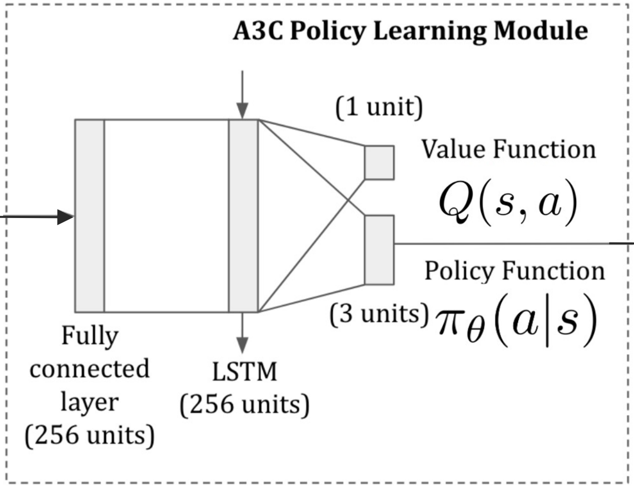
Exploration + experience replay
Decorrelate samples
Fixed targets

Critic: evaluates how good the action is

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}\left[\left(r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i)\right)^2\right]$$

Q-learning target — Q-network





$\pi_\theta(a|s)$

Actor: decides what actions to take

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)\right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Variance reduction with a baseline

[Minh et al., Asynchronous Methods for Deep Reinforcement Learning. ICML 2016]

Summary: RL Methods

Value iteration
Policy iteration
(Deep) Q-learning

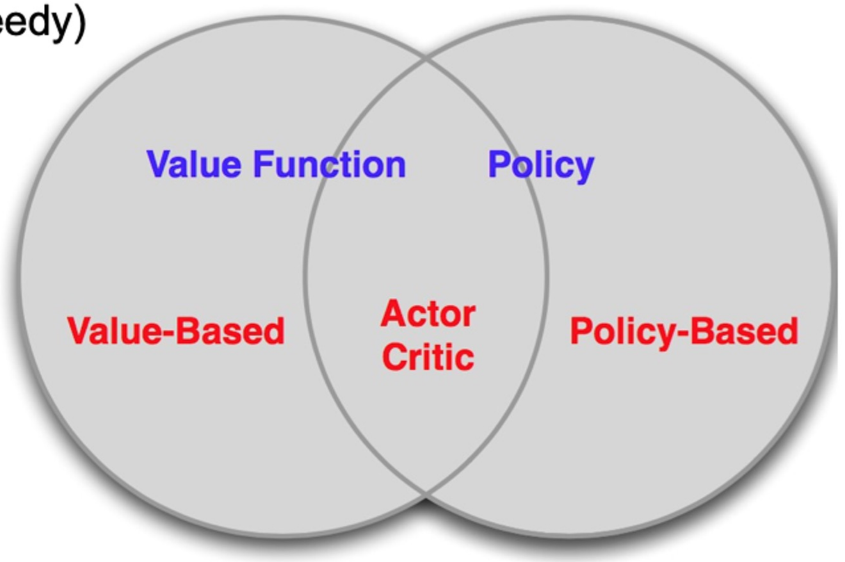
- Value Based
  - Learned Value Function
  - Implicit policy (e.g. ε-greedy)

Policy gradients

- Policy Based
  - No Value Function
  - Learned Policy

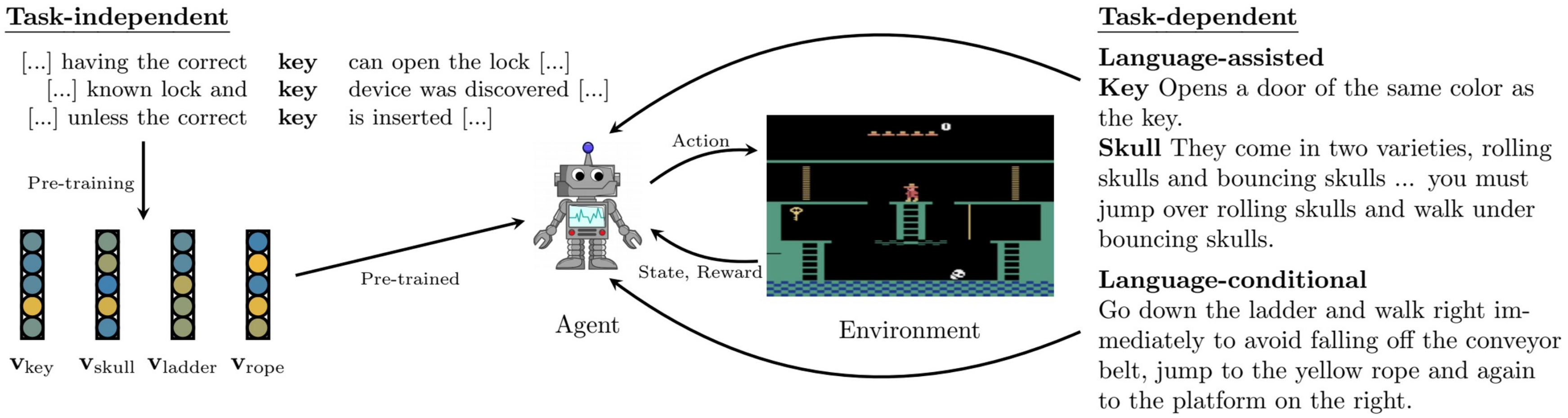
Actor (policy)
Critic (Q-values)

- Actor-Critic
  - Learned Value Function
  - Learned Policy



[Slides from Fragkiadaki, 10-703 CMU]

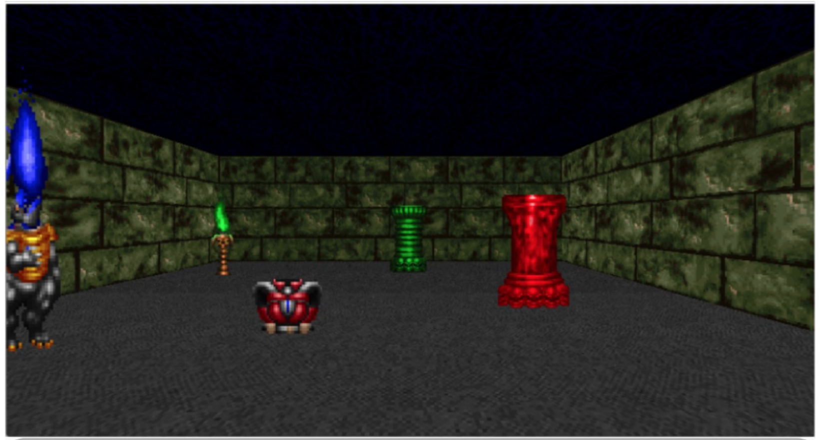
Back to Reasoning: Interactive Reasoning



[Luketina et al., A Survey of Reinforcement Learning Informed by Natural Language. IJCAI 2019]

Language-conditional RL: Instruction Following

- Navigation via instruction following



Go to the green torch

Train

Go to the short red torch
Go to the blue keycard
Go to the largest yellow object
Go to the green object

Test

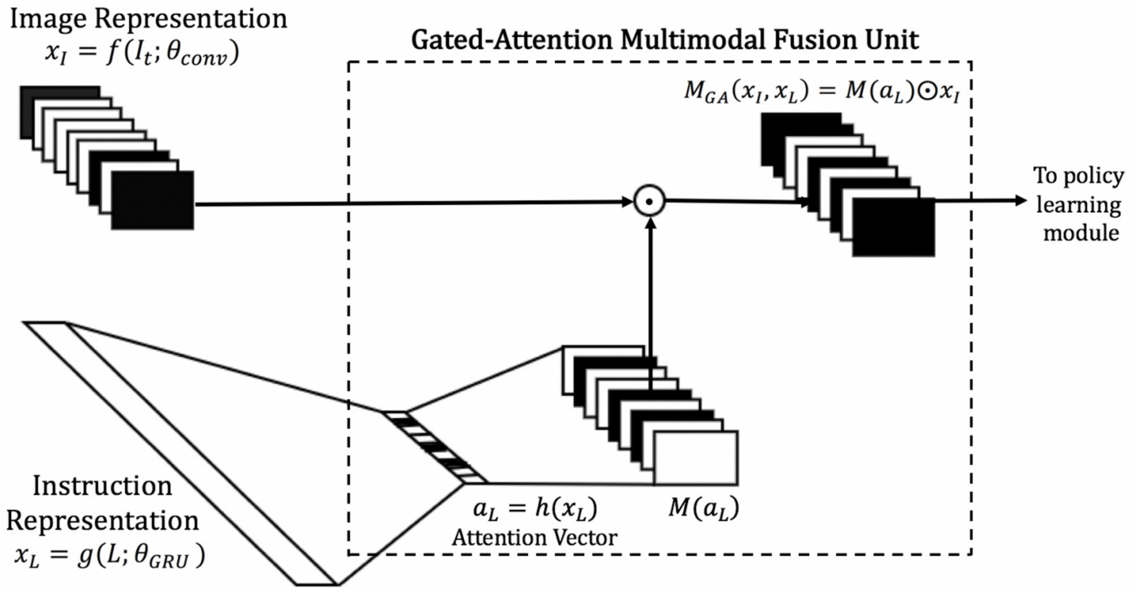
Go to the tall green torch
Go to the red keycard
Go to the smallest blue object

Fusion
Alignment
Ground language
Recognize objects
Navigate to objects
Generalize to unseen objects

[Misra et al., Mapping Instructions and Visual Observations to Actions with Reinforcement Learning. EMNLP 2017]
[Chaplot et al., Gated-Attention Architectures for Task-Oriented Language Grounding. AAAI 2018]

Language-conditional RL: Instruction Following

- Gated attention via element-wise product

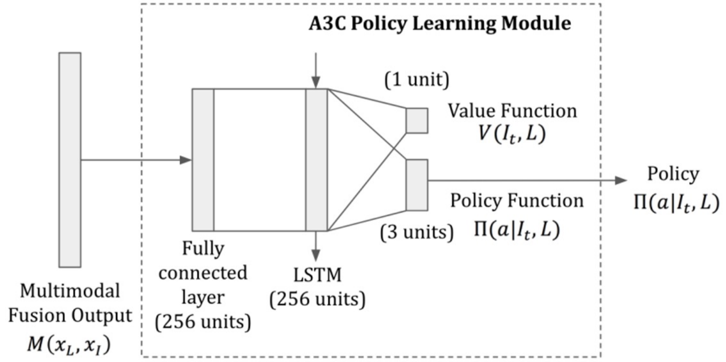


Fusion
Alignment
Ground language
Recognize objects

[Chaplot et al., Gated-Attention Architectures for Task-Oriented Language Grounding. AAAI 2018]

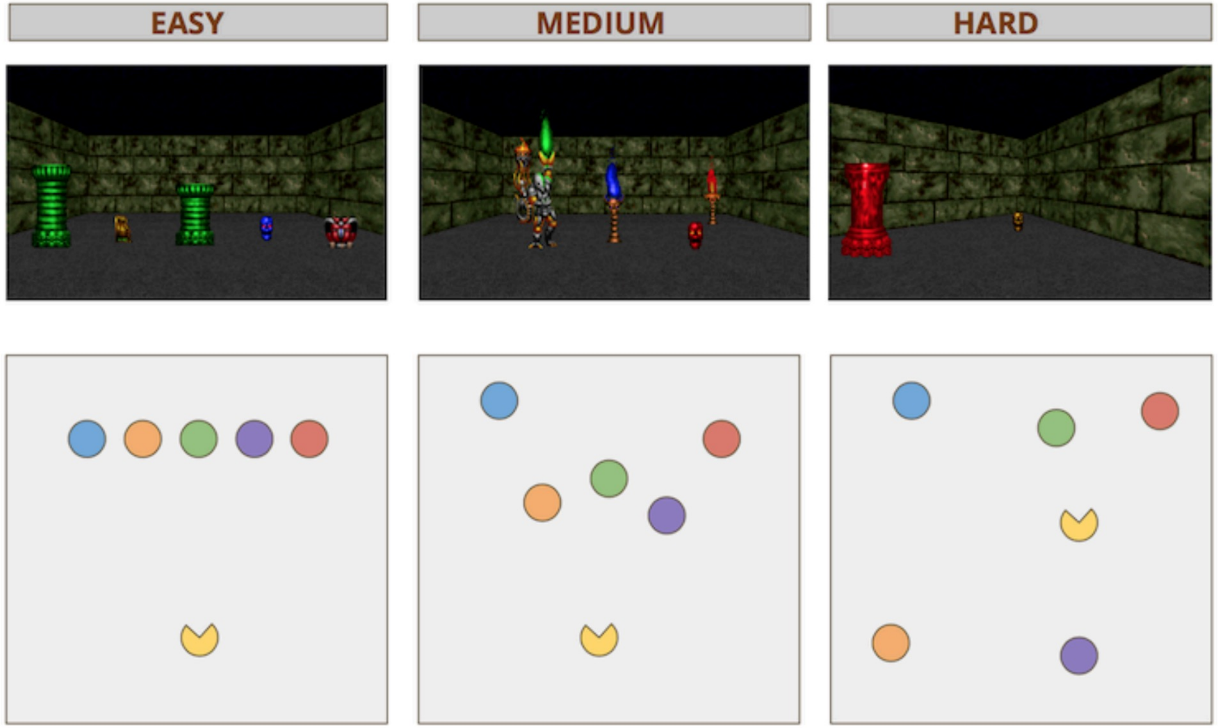
Language-conditional RL: Instruction Following

- Policy learning
  - Asynchronous Advantage Actor-Critic (A3C) (Mnih et al.)
    - uses a deep neural network to parametrize the policy and value functions and runs multiple parallel threads to update the network parameters.
    - use **entropy regularization** for improved exploration
    - use **Generalized Advantage Estimator** to reduce the variance of the policy gradient updates (Schulman et al.)



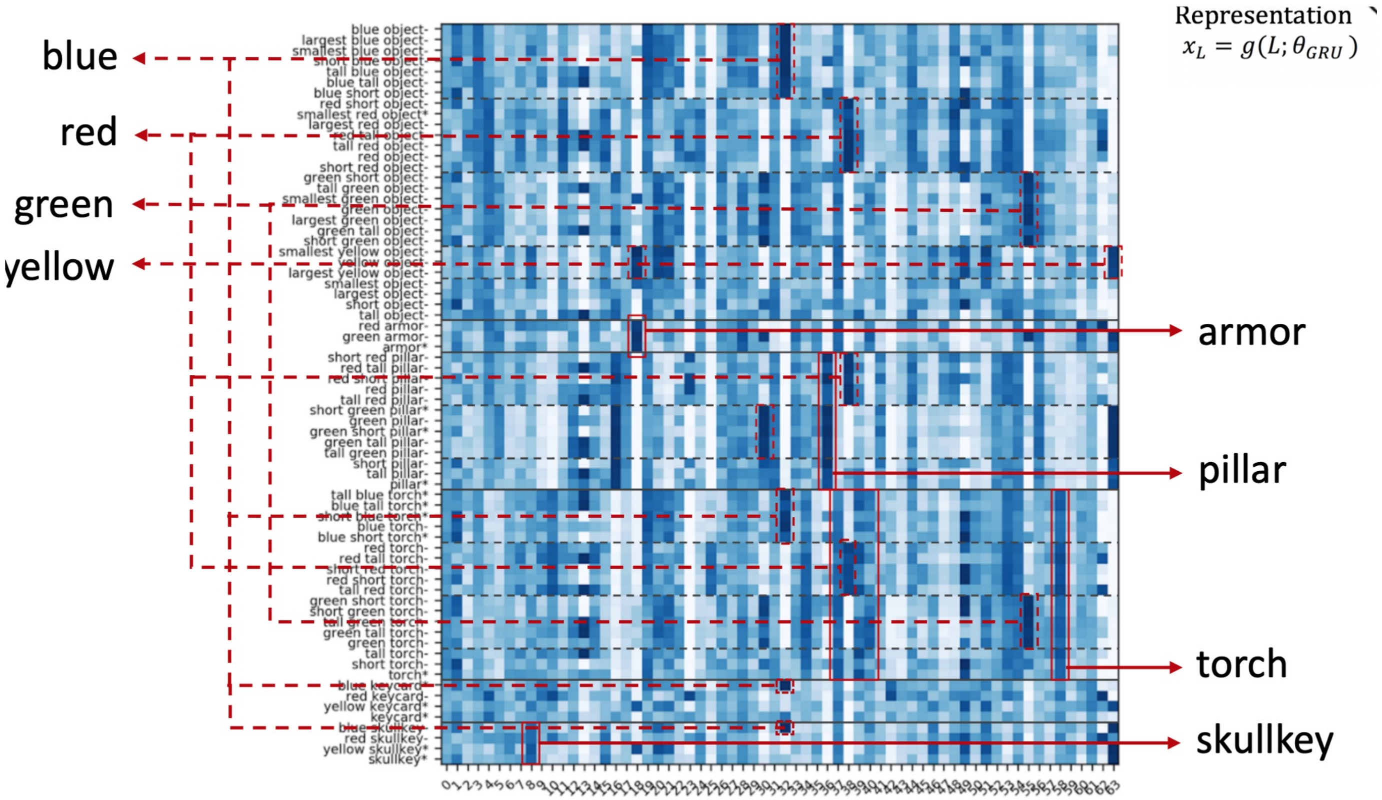
[Chaplot et al., Gated-Attention Architectures for Task-Oriented Language Grounding. AAAI 2018]

Language-conditional RL: Instruction Following



[Chaplot et al., Gated-Attention Architectures for Task-Oriented Language Grounding. AAAI 2018]

Language-conditional RL: Instruction Following



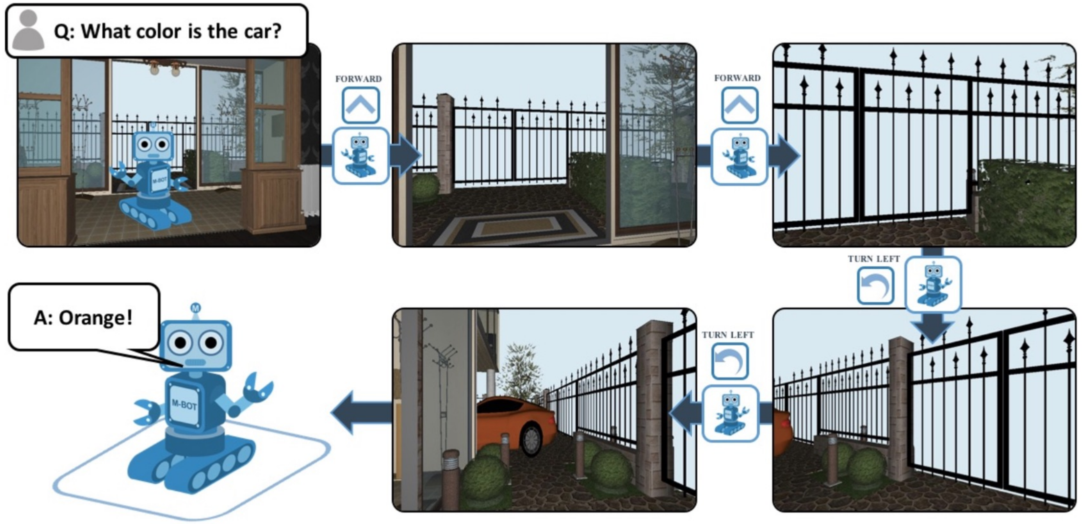
Grounding is important for generalization

blue armor, red pillar -> blue pillar

[Chaplot et al., Gated-Attention Architectures for Task-Oriented Language Grounding. AAAI 2018]

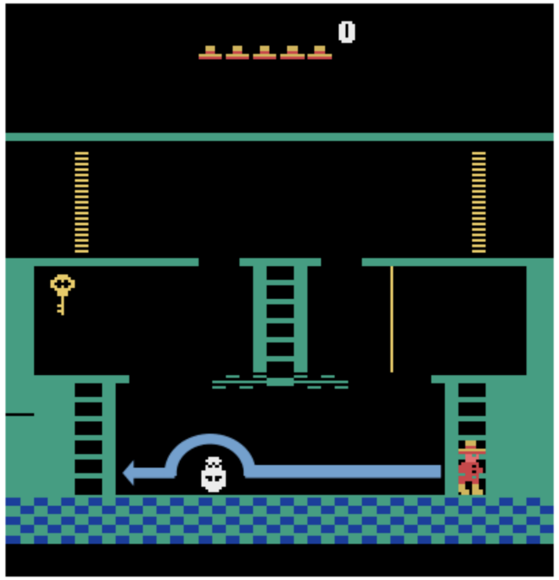
Language-conditional RL: Embodied QA

- Embodied QA: Navigation + QA



[Das et al., Embodied Question Answering. CVPR 2018]

Language-assisted RL: Reward Shaping



Montezuma's revenge

Sparse, long-term reward problem
General solution: reward shaping via auxiliary rewards

Natural language for reward shaping

"Jump over the skull while going to the left"

from Amazon Mturk :-(
asked annotators to play the game and describe entities

Intermediate rewards to speed up learning

[Goyal et al., Using Natural Language for Reward Shaping in Reinforcement Learning. IJCAI 2019]

Language-assisted RL: Domain knowledge

- Learning to read instruction manuals

The natural resources available where a population settles affects its ability to produce food and goods. Build your city on a plains or grassland square with a river running through it if possible.

Figure 1: An excerpt from the user manual of the game Civilization II.

[Branavan et al., Learning to Win by Reading Manuals in a Monte-Carlo Framework. JAIR 2012]

Language-assisted RL: Domain knowledge

- Learning to read instruction manuals

The natural resources available where a population settles affects its ability to produce food and goods. Build your city on a plains or grassland square with a river running through it if possible.

Map tile attributes:
- Terrain type (e.g. grassland, mountain, etc)
- Tile resources (e.g. wheat, coal, wildlife, etc)

City attributes:
- City population
- Amount of food produced

Unit attributes:
- Unit type (e.g., worker, explorer, archer, etc)
- Is unit in a city ?

1. Choose **relevant** sentences
2. Label words into **action-description**, **state-description**, or **background**

[Branavan et al., Learning to Win by Reading Manuals in a Monte-Carlo Framework. JAIR 2012]

Language-assisted RL: Domain knowledge

- Learning to read instruction manuals

- Phalanxes are twice as effective at defending cities as warriors. ✔
- Build the city on plains or grassland with a river running through it. ✔
- You can rename the city if you like, but we'll refer to it as washington.
- There are many different strategies dictating the order in which advances are researched

Relevant sentences

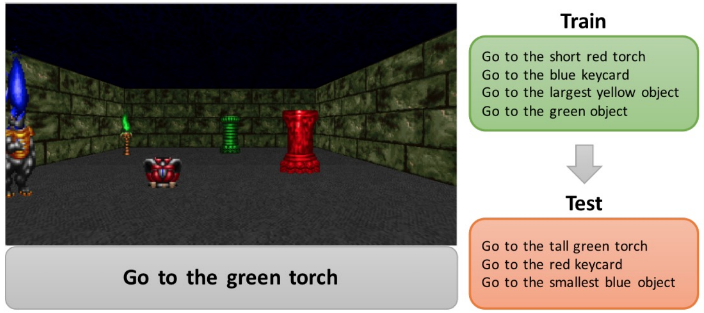
- After the road is built, use the settlers to start improving the terrain.
  S S S A A A A A
- When the settlers becomes active, chose build road.
  S S S A A A
- Use settlers or engineers to improve a terrain square within the city radius
  A S ✘ A A S A ✘ S S S S

A: action-description
S: state-description

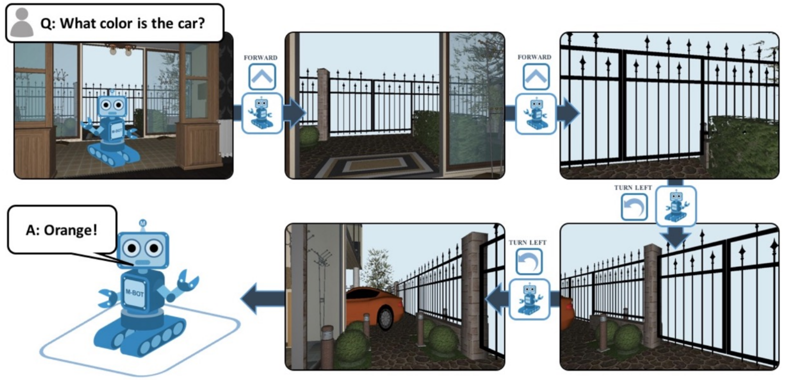
[Branavan et al., Learning to Win by Reading Manuals in a Monte-Carlo Framework. JAIR 2012]

Summary: Interactive Reasoning

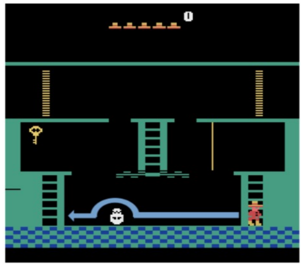
Instruction following



Embodied learning



Reward shaping



"Jump over the skull while going to the left"

Domain knowledge

The natural resources available where a population settles affects its ability to produce food and goods. Build your city on a plains or grassland square with a river running through it if possible.

Figure 1: An excerpt from the user manual of the game Civilization II.

[Luketina et al., A Survey of Reinforcement Learning Informed by Natural Language. IJCAI 2019]

Summary

Definition: Combining knowledge, usually through multiple inferential steps, exploiting multimodal alignment and problem structure.

	(A) Structure modeling	(B) Intermediate concepts	(C) Inference paradigm	(D) External knowledge
Last Thursday	Temporal Hierarchical	Continuous		
Today	Interactive			

RL basics

Summary: RL Methods

Epsilon greedy + exploration
Experience replay
Decorrelate samples
Fixed targets

- Value Based
  - Learned Value Function
  - Implicit policy (e.g. ε-greedy)

Value iteration
Policy iteration
(Deep) Q-learning

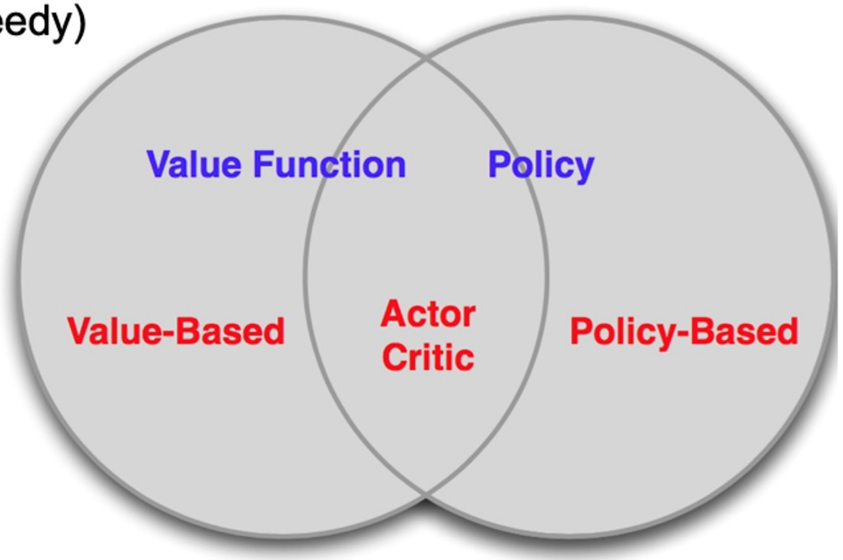
- Policy Based
  - No Value Function
  - Learned Policy

Policy gradients

Variance reduction with a baseline

- Actor-Critic
  - Learned Value Function
  - Learned Policy

Actor (policy)
Critic (Q-values)



[Slides from Fragkiadaki, 10-703 CMU]

Summary

Definition: Combining knowledge, usually through multiple inferential steps, exploiting multimodal alignment and problem structure.

	(A) Structure modeling	(B) Intermediate concepts	(C) Inference paradigm	(D) External knowledge
Last Thursday	Temporal Hierarchical	Continuous		
Today	Interactive			
Thursday	Discovery	Discrete	Causal Logical	Knowledge Commonsense