Language Understanding for Text-based Games using Deep Reinforcement Learning

Karthik Narasimhan, Tejas D Kulkarni, Regina Barzilay

Presented by Mark Chang

Outline

Introduction

- Learning the control policies for text-based games.
- A deep **reinforcement learning** framework
  - learn state representations (semantics of text)
  - learn action policies (rules of game)

Introduction

- MUD(Multi-User Dungeon) Game
  - A text-based multiplayer real-time virtual world.
  - Player read descriptions of the world state.
  - Player take actions by natural language commands.

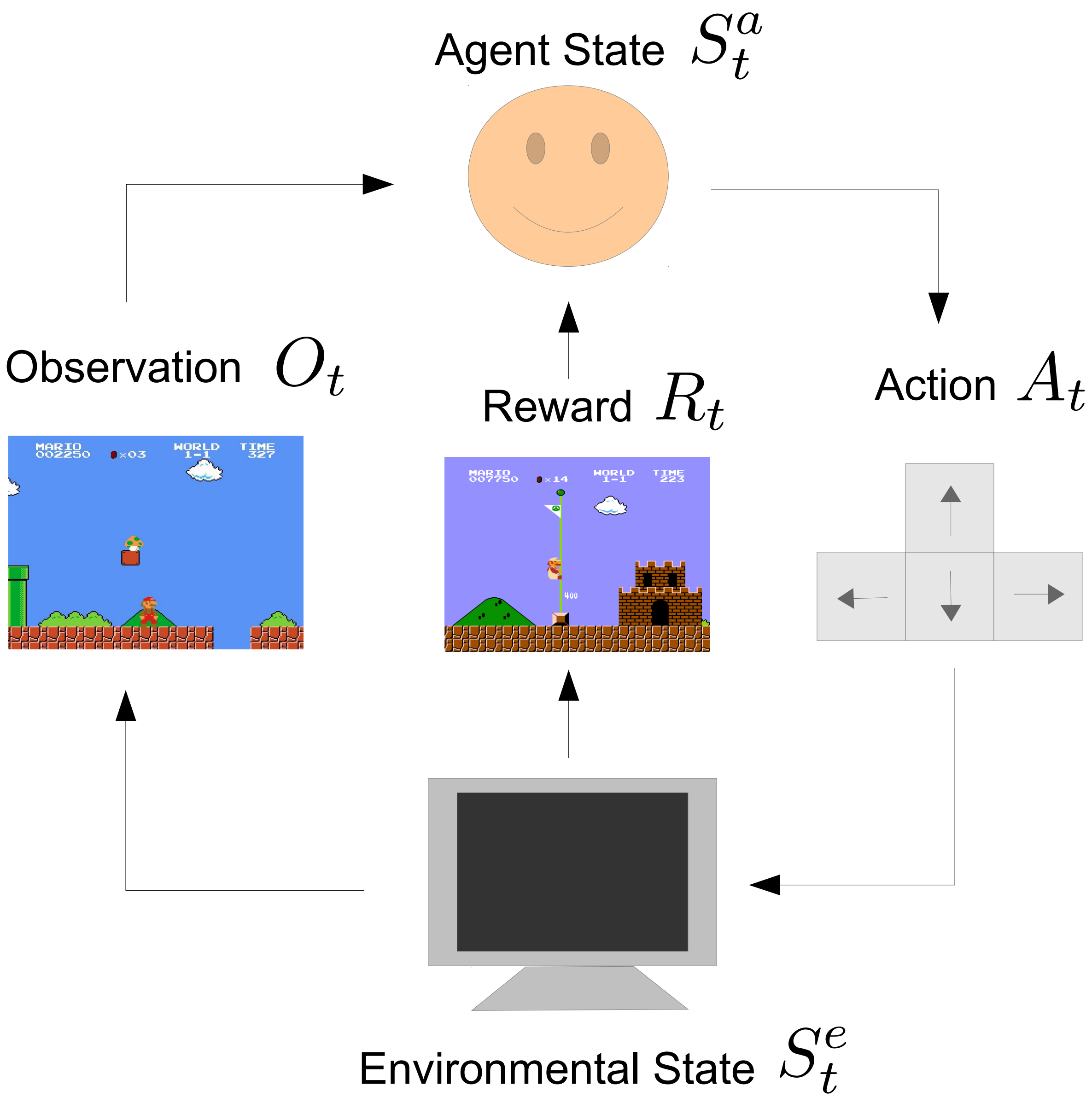
```
You are standing very close to the bridge’s eastern foundation.
If you go east you will be back on solid ground.
The bridge sways in the wind.

>  Go east
```

Background

- Reinforcement Learning
- Q-Learning
- Deep Q-Network
- Long Short-Term Memory

Reinforcement Learning


Agent State S_t^a
Observation O_t
Reward R_t
Action A_t
MARIO 002250 ×03 WORLD 1-1 TIME 327
MARIO 007750 ×14 WORLD 1-1 TIME 223
400
Environmental State S_t^e

- Rules of the game are unknown.
- No supervisor, only a reward signal.
- Feedback is delayed.
- Agent’s actions affect the subsequent data it receives.

Reinforcement Learning

- Markov Decision Process $< S, A, P, R, \gamma >$
  - S is a finite set of states
  - A is a finite set of actions
  - P is a state transition probability matrix
    $P^a_{ss'} = P[S_{t+1} = s' \mid S_t = s, A_t = a]$
  - R is a reward function
    $R_s = E[R_{t+1} \mid S_t = s]$
  - γ is a discount factor

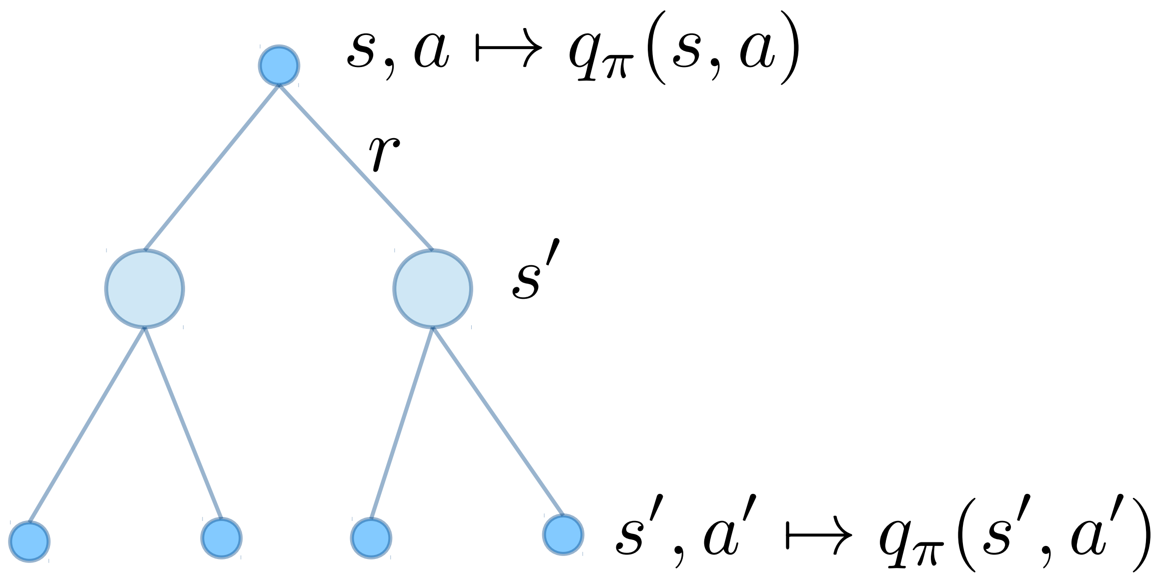
Reinforcement Learning

- Return : total discounted reward from time-step t.
  - $G_t = R_{t+1} + \gamma R_{t+2} + ... + R_{\text{final}}$
- Policy : Agent Behavior
  - Deterministic policy: $\pi(s) = a$
  - Stochastic policy: $\pi(a \mid s) = P[A_t = a \mid S_t = s]$
- Value function: Expected return starting from state s.
  - state-value function: $v_\pi(s) = E_\pi[G_t \mid S_t = s]$
  - action-value function: $q_\pi(s, a) = E_\pi[G_t \mid S_t = s, A_t = a]$

Reinforcement Learning

- Bellman Equation for Action-Value Function

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss}^a \sum_{a' \in A} \pi(a' \mid s') q_\pi(s',a')$$

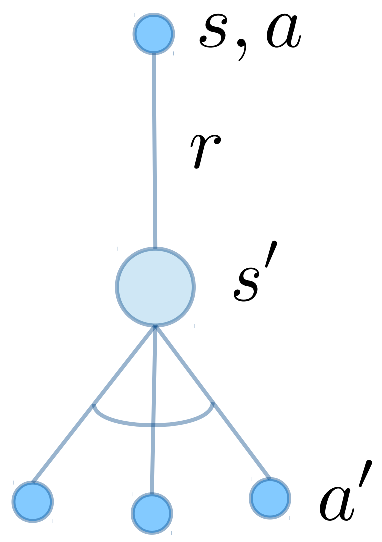


Q-Learning

- A model-free, off-policy technique to learn an optimal Q(s, a) for the agent.

$$Q_{i+1}(s,a) \leftarrow Q_i(s,a) + \alpha(R + \gamma \max_{a'} Q_i(s',a') - Q_i(s,a))$$

- model-free: using sampling instead of complete model
- off-policy: learning optimal policy independently of agent's actions
- $Q(s,a)$: Estimation of $q(s,a)$
- α : Step Size (Learning Rate)



Q-Learning

- How to choose A in $Q(s,a)$?
- ϵ - Greedy Exploration
  - ensuring continual exploration
  - With probability 1 - ϵ choose the greedy action
  - With probability ϵ choose an action at random

$$\pi(a \mid s) = \begin{cases} \frac{\epsilon}{m} + 1 - \epsilon & \text{if } a = \underset{a'}{\operatorname{argmax}}\, Q(s,a') \\ \frac{\epsilon}{m} & \text{otherwise} \end{cases}$$

Deep Q-Network

- Problem with $Q(s,a)$:
  - too many states/actions to store in memory
  - too slow to individually learn their value
- Solution :
  - Approximate $Q(s,a)$ using a neural network

    Q-Network : $Q(s,a;\theta)$

Deep Q-Network

- Loss Function:

  reduce the discrepancy between

  - the predicted value : $Q(s, a; \theta_i)$
  - the expected Q-value : $r + \gamma \max_{a'} Q(s', a'; \theta_{i-1})$

$$L(\theta_i) = E_{s,a}[(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i))^2]$$

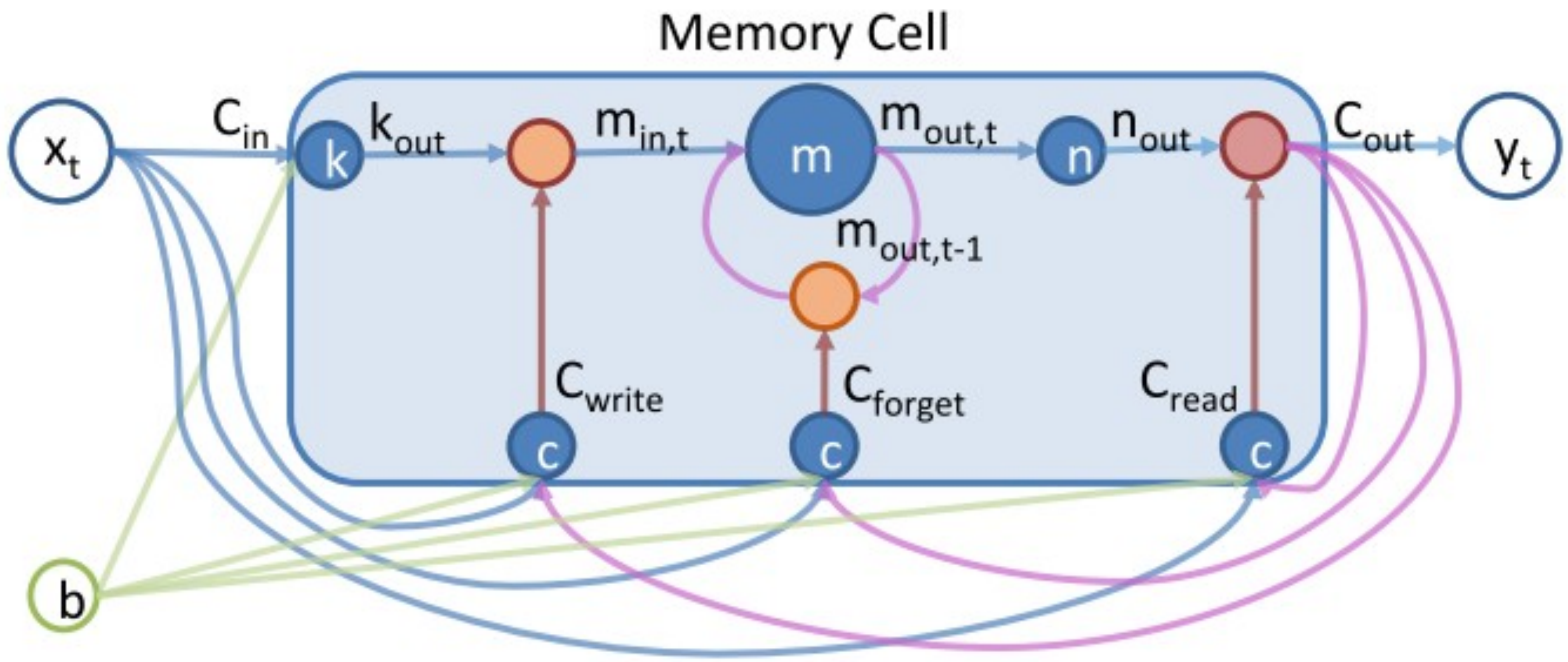
- Gradient of Loss Function:

$$\nabla_{\theta_i} L(\theta_i) = E_{s,a}[(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)) \nabla_{\theta_i} Q(s, a; \theta_i)]$$

  - previous iteration θ_{i-1} is fixed

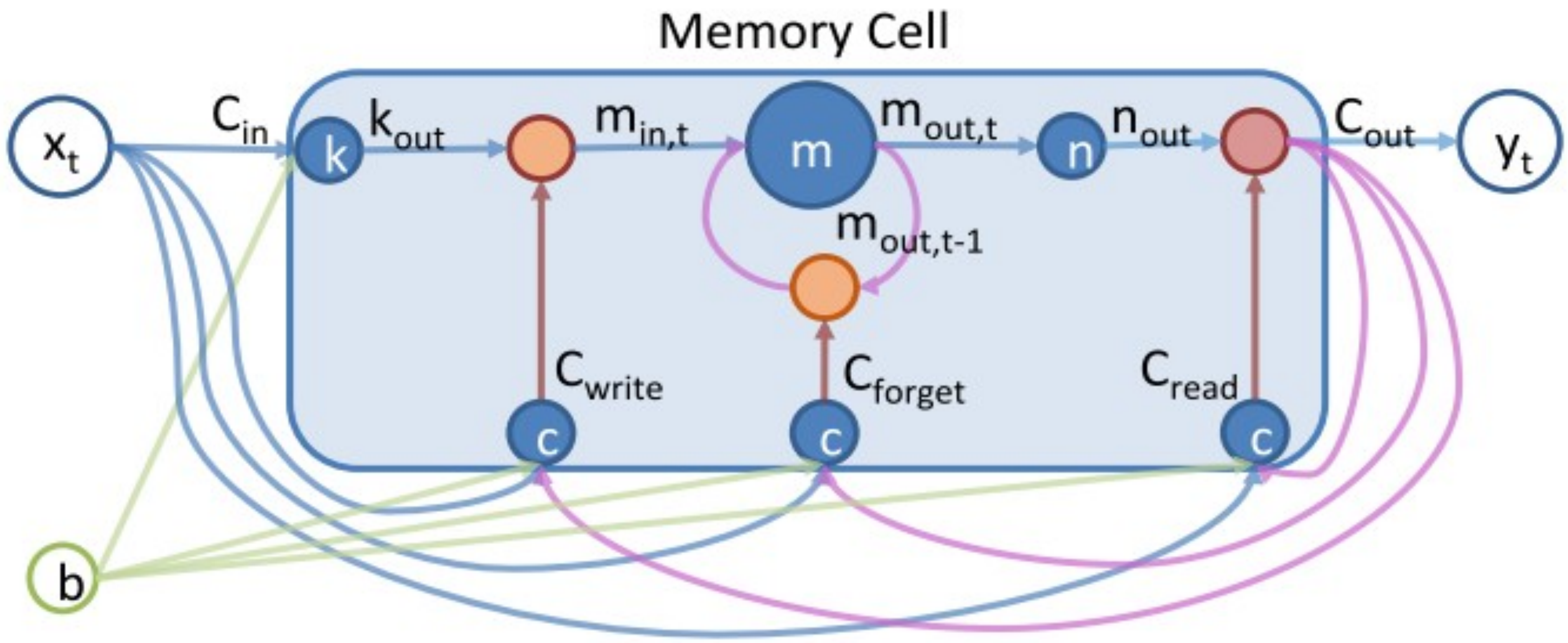
Long Short-Term Memory

- recurrent neural networks with memory
- able to connect and recognize long-range patterns between words in text

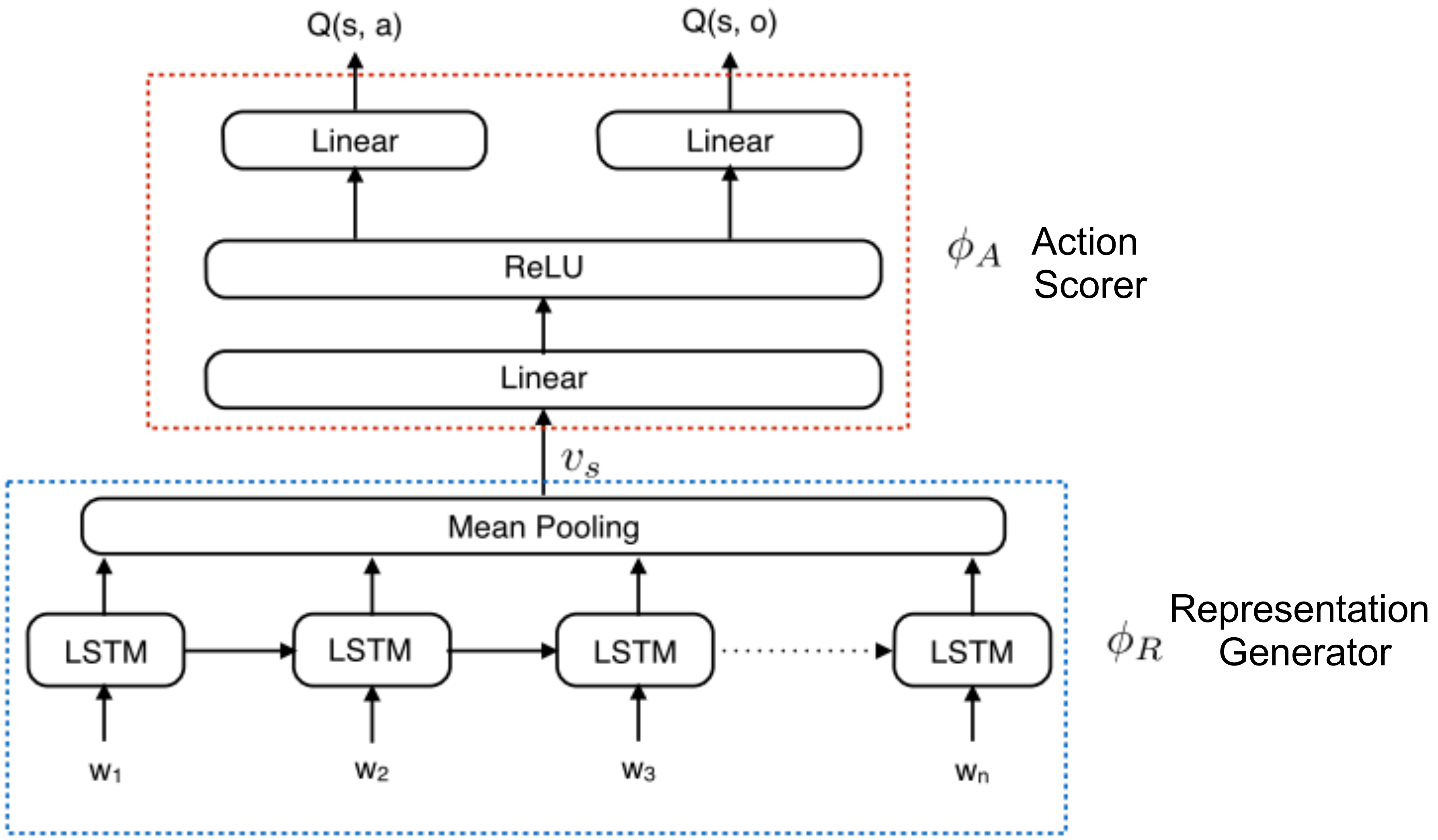


Long Short-Term Memory

- Write: $m_{in,t} = k_{out} C_{write}$
- Forget: $m_{out,t} = \mathrm{m}_{in,t} + C_{forget} \mathrm{m}_{out,t-1}$
- Read: $C_{out} = \mathrm{n}_{out} \mathrm{C}_{read}$



Deep Reinforcement Learning Framework

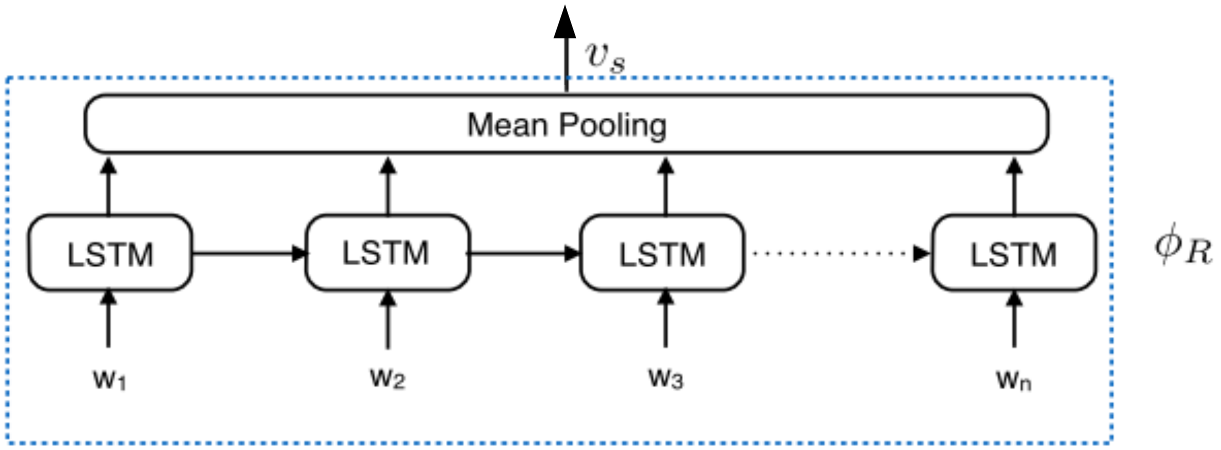


Game Representation

- Represented by the tuple $< H, A, T, R, \Psi >$
- H : the set of all possible game states.
- $A = \{(a, o)\}$: all commands (action-object pairs).
- $T(h' \mid h, a, o)$: stochastic transition function between game states.
- $R(h, a, o)$: reward function.
- $\Psi : H \rightarrow S$: converts game state into a textual description S, since the game state is hidden.

Representation Generator ϕ_R

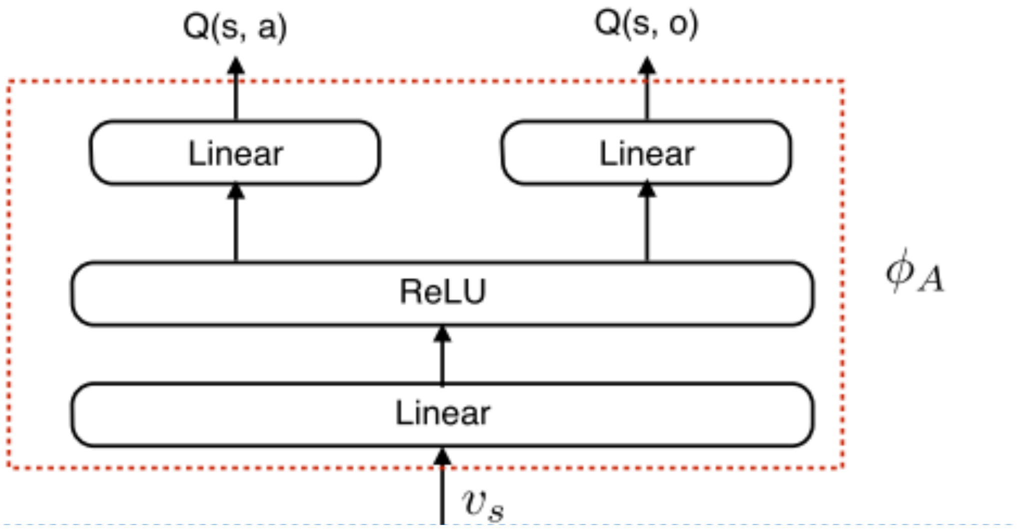
- Reads displayed raw text $w_1, w_2, ..., w_n$
- converts it to a vector representation v_s
- LSTM : $x_k = f(w_1, ..., w_k)$
- mean pooling layer : $v_s = \frac{1}{n} \sum x_k$



v_s

Action Scorer ϕ_A

- multi-layered neural network
- given the current state representation v_s , produces scores for actions.
- only consider commands with one action and object : (a, o)



Parameter Learning

- learning the parameters θ_R, θ_A by using stochastic gradient descent with RMSprop
- keep track of the agent's previous experiences in a memory D
- sample a random transition $(\hat{s}, \hat{a}, s'r)$ from D, and updating the parameters.

$$L(\theta_i) = E_{\hat{s},\hat{a}}[(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(\hat{s}, \hat{a}; \theta_i))^2]$$

- prioritized sampling to the transition with reward $r > 0$

Parameter Learning

```
3:  for $episode = 1, M$ do
4:      Initialize game and get start state description $s_1$
5:      for $t = 1, T$ do
6:          Convert $s_t$ (text) to representation $v_{s_t}$ using $\phi_R$
7:          if $random() < \epsilon$ then
8:              Select a random action $a_t$
9:          else
10:             Compute $Q(s_t, a)$ for all actions using $\phi_A(v_{s_t})$
11:             Select $a_t = \text{argmax}\ Q(s_t, a)$
12:         Execute action $a_t$ and observe reward $r_t$ and new state $s_{t+1}$
13:         Set priority $p_t = 1$ if $r_t > 0$, else $p_t = 0$
14:         Store transition $(s_t, a_t, r_t, s_{t+1}, p_t)$ in $\mathcal{D}$
15:         Sample random mini batch of transitions $(s_j, a_j, r_j, s_{j+1}, p_j)$ from $\mathcal{D}$,
              with fraction $\rho$ having $p_j = 1$
16:         Set $y_j = \begin{cases} r_j & \text{if } s_{j+1} \text{ is terminal} \\ r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta) & \text{if } s_{j+1} \text{ is non-terminal} \end{cases}$
17:         Perform gradient descent step on the loss $\mathcal{L}(\theta) = (y_j - Q(s_j, a_j; \theta))^2$
```

Experiment & Result

- Two World:
  - Home world : hand-cracted simple testcase
  - Fantasy World : real MUD game testcase
- Evaluation:
  - the cumulative reward
  - the fraction of quests completed by the agent
- Baseline:
  - Random agent
  - Bag-of-words
  - bigram

Experiment & Result

- Home world :
  - Mimic a typical house with 4 rooms.
  - Each room contains a representative object.
  - Text : Description of the character's state and that the quest.

- Example:

  quest:

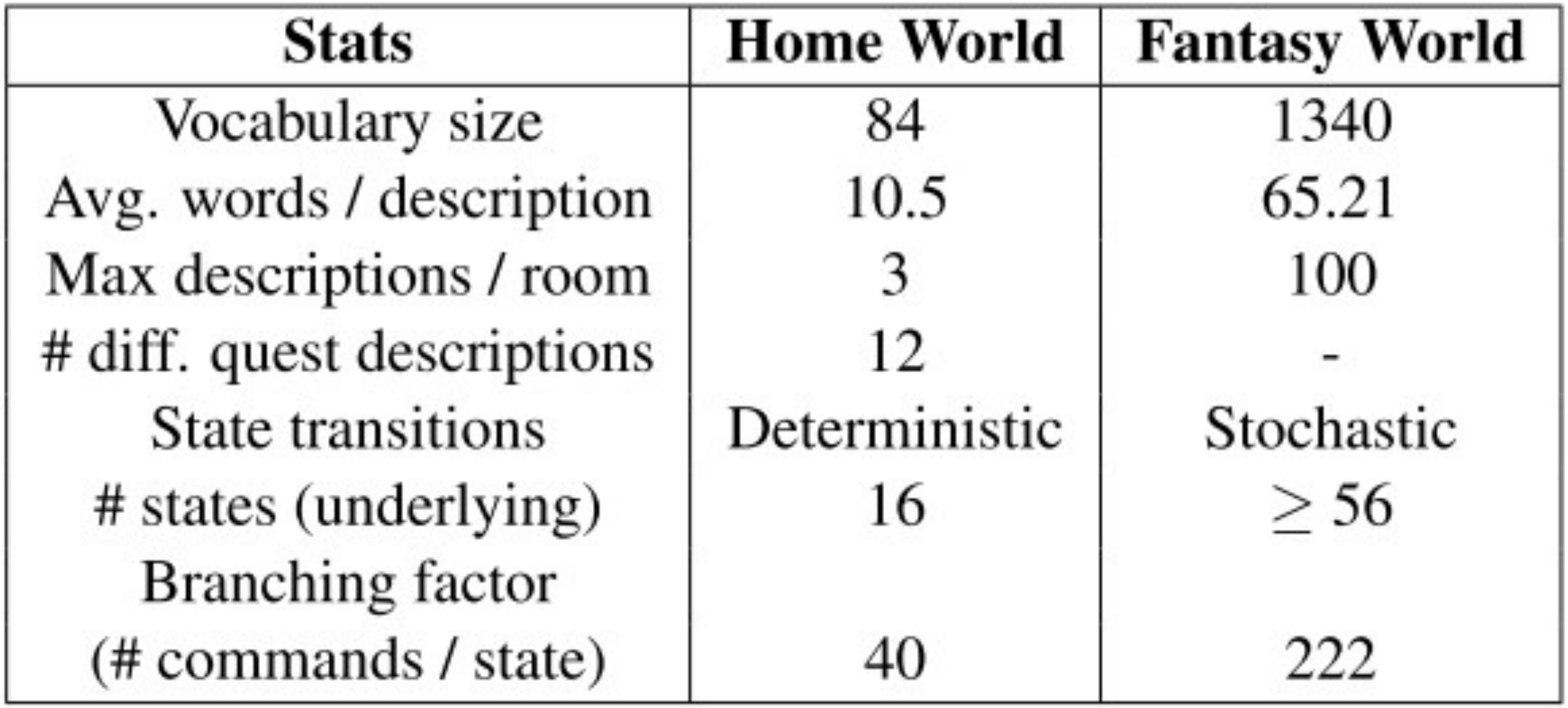
    Not you are sleepy now but you are hungry now.

  command:

    go to kitchen

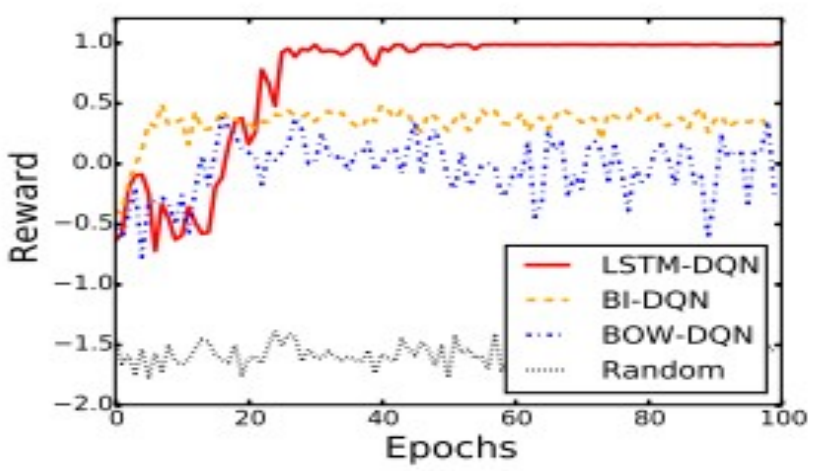
    eat apple

Experiment & Result

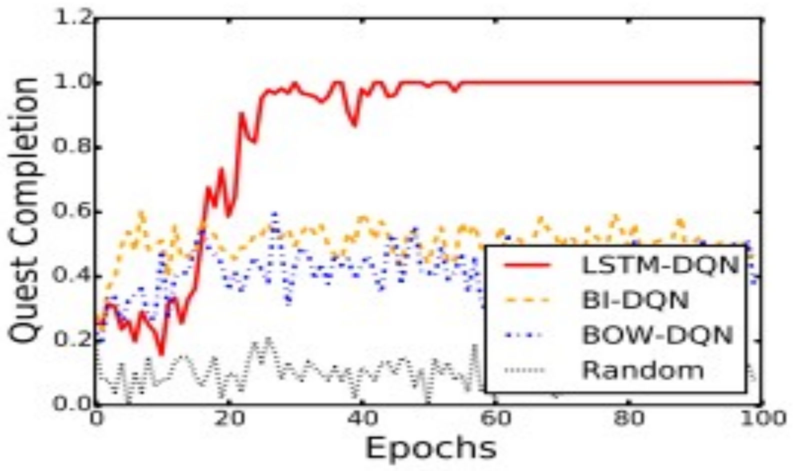


Stats	Home World	Fantasy World
Vocabulary size	84	1340
Avg. words / description	10.5	65.21
Max descriptions / room	3	100
# diff. quest descriptions	12	-
State transitions	Deterministic	Stochastic
# states (underlying)	16	≥ 56
Branching factor (# commands / state)	40	222

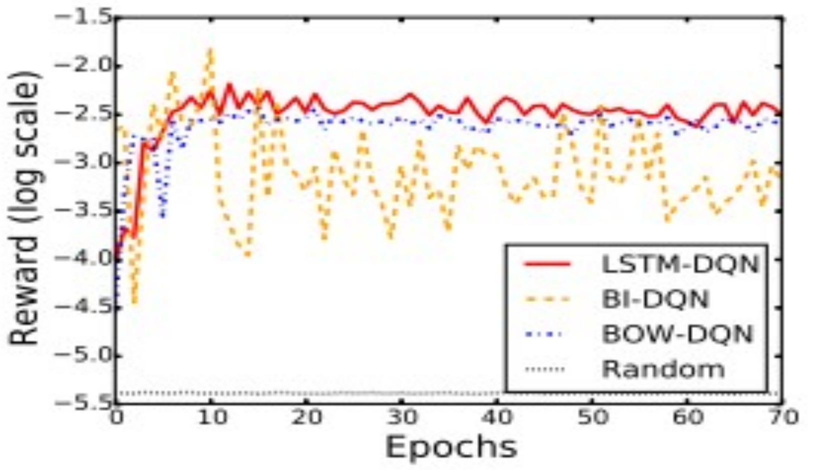
Experiment & Result



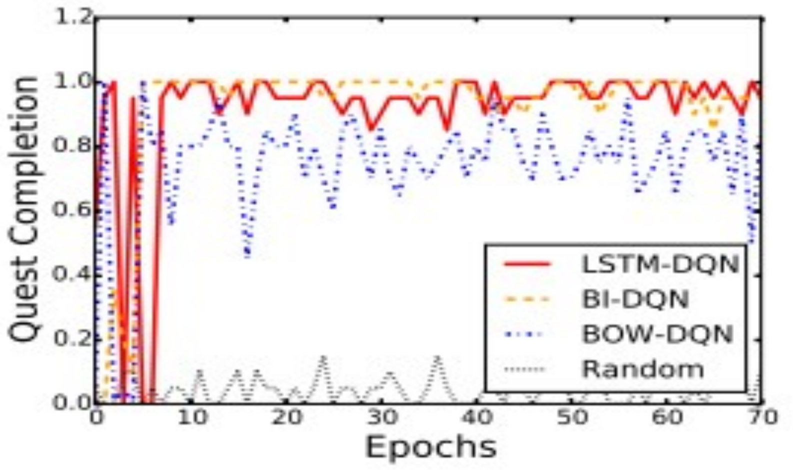
Reward (Home)



Quest completion (Home)

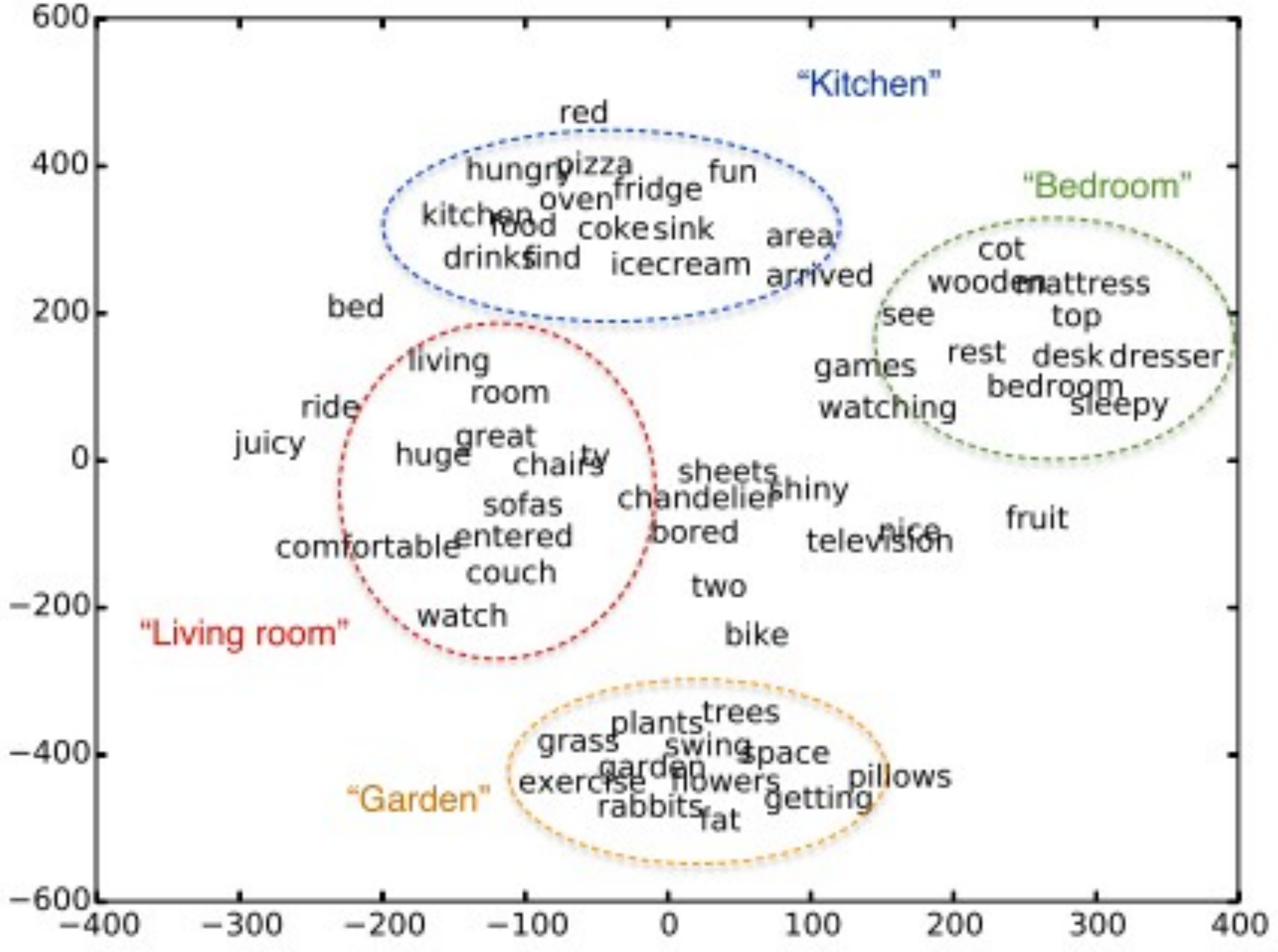


Reward (Fantasy)



Quest completion (Fantasy)

Experiment & Result



Conclusion

- a deep reinforcement learning framework
- jointly learn state representations and action policies using game rewards as feedback.
- mapping text descriptions into vectors that capture the semantics of the game states.