

A Curiosity-Driven Approach for Generating Textual Descriptions of Environments

Xinyun Zhang



Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-142

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-142.html>

May 19, 2021

Copyright © 2021, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

A Curiosity-Driven Approach for Generating Textual Descriptions of Environments

by

Xinyun(Lily) Zhang

A dissertation submitted in partial satisfaction of the
requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Trevor Darrell, Chair
Professor Daniel Klein

Spring 2021

**A Curiosity-Driven Approach for Generating Textual Descriptions of
Environments**

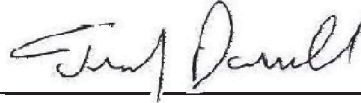
by Xinyun(Lily) Zhang

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences,
University of California at Berkeley, in partial satisfaction of the requirements for the
degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:



Professor Trevor Darrell
Research Advisor

5/18/21

(Date)



Professor Daniel Klein
Second Reader

5/19/21

(Date)

A Curiosity-Driven Approach for Generating Textual Descriptions of Environments

Copyright 2021
by
Xinyun(Lily) Zhang

Abstract

A Curiosity-Driven Approach for Generating Textual Descriptions of Environments

by

Xinyun(Lily) Zhang

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Trevor Darrell, Chair

Generating textual descriptions of environments in reinforcement learning can be difficult, especially in the absence of explicit signal for recognizing the relationships within them. In this work, we propose a curiosity-driven approach for collecting the trajectories necessary for generating textual descriptions of environments. We formulate the task of finding the exploration policy as a two-player game between the policy and a forward model which predicts state transitions. In addition, we propose a meta-training scheme allowing them to adapt to changes in the environment. We also propose a text generation scheme that helps to generate natural language descriptions from trajectories. Finally, we evaluate our model in the RTFM domain [65], in which two monster and weapon pairs exist. The agent is expected to learn a policy that helps it to figure out all the monster and weapon relation pairs through interaction with the environment and generate a descriptive natural language document that summarizes all the environment dynamics.

Acknowledgement

I would like to thank my research advisors, Professors Trevor Darrell and Daniel Klein, for the opportunity to work in Berkeley Artificial Intelligence Research Lab and providing me the resources and opportunities for research during my time in UC Berkeley. I would especially like to thank my mentors, Rudy Corona and Nicholas Tomlin, for the guidance and support in all my research endeavours.

Contents

Contents	ii
List of Figures	iv
1 Introduction	1
2 Related Work	3
2.1 Language Grounding	3
2.2 Learning by Interaction for Embodied Agents	3
2.3 Curiosity-Driven Exploration	4
2.4 Meta Learning	4
3 Task	5
3.1 Summary of the Task in Zhong et al. 2020	5
3.2 Our Task	5
4 Model for Learning an Exploration Policy	7
4.1 Forward Model	7
4.2 Policy Generator	7
4.3 Meta Training and Model Architecture	8
5 Model for Text Generation	11
5.1 Data Collection	11
5.2 Architecture and Training of the Seq2Seq Model	11
6 Experimental Setup for Learning an Exploration Policy	13
6.1 Environment	13
6.2 Visualization of the Environment	14
6.3 Dataset for the Forward Model	14
6.4 Training Details	14
6.5 Baseline Methods for Overall Training	15
6.6 Baseline Methods for Forward Model	15

7	Experimental Setup for Text Generation	16
7.1	Dataset	16
7.2	Baseline Methods	16
7.3	Details of the Model	17
7.4	Evaluation Metric	17
8	Experiments	18
8.1	Performance of Model for Learning an Exploration Policy	18
8.2	Performance of Model for Text Generation	22
9	Conclusion	26
	Bibliography	27

List of Figures

3.1	RTFM Environment [65]	6
8.1	Graph of the number of monster/weapon pairs learnt by the model within 300 steps using extrinsic vs intrinsic reward. The model with extrinsic reward performs better than the model with intrinsic reward.	18
8.2	Graph of the intrinsic reward of plugging pre-trained forward model in policy generator. Reward first increases and later converges, which complies with the fact that the number of monster/weapon pairs learnt by the model within 300 steps using intrinsic reward fails to increase after a short time.	19
8.3	Graph of the number of monster/weapon pairs learnt by the model within 300 steps using jointly training of forward model and policy generator. The number first decreases and later converges at a low level, which is worse than the performance of the model with pre-trained forward model plugged in policy generator.	20
8.4	Graph of the intrinsic reward of jointly trained forward model and policy generator. Reward first increases and later converges.	20
8.5	Graph of the error of forward model over time. The number of pickup error stays constant, which meets our expectation. The number of pickup error also stays constant, which is unexpected, since we expect the number to decrease over time.	21
8.6	Example of successful killing. At $t=1$, the monster is killed in real environment, and it is also killed in the model prediction.	24
8.7	Example of unsuccessful killing. At $t=1$, the monster is killed in real environment, but in the model prediction the monster is still alive.	25

Chapter 1

Introduction

Environment dynamics are defined as the non-stationary elements in the environment that are essential to decision making. For instance, imagine a healthcare robot is expected to pick up a specific kind of medicine for a patient. The robot needs to know which person needs which kind of medicine in order to pick up the correct one, and what the patient needs can vary over time due to his or her health conditions. Similarly, in a game environment, a player is expected to pick up a specific weapon to kill a monster. The player needs to know which weapon can kill the monster in order to complete the task, and the weapon that can kill the monster can vary in different rounds of the game. In our work, we propose a method that can generate textual descriptions of environment dynamics in a game setting.

Generating textual descriptions of environment dynamics can be useful for real-world many applications. For example, the document can serve as a guide for an agent in game, which helps with generalization to new environment instances [65]. It can also be used to generate a safety guide for humans when a robot explores a dangerous environment, such as a wild forest, or a battlefield.

Prior work on generating textual descriptions has been limited to recognizing elements in fixed environments. For instance, image [58, 62, 63, 23, 30, 10] and video captioning [56, 64, 60, 61, 29] have been studied for years, but have been limited to recognizing key elements in a static setting. There is a clear need for a method for generating textual descriptions of a dynamic environment from an agent's trajectories.

We propose a learning framework for generating textual descriptions of environment dynamics from an agent's trajectories, which includes an exploration policy to collect informative trajectories, and a text generator to produce the natural language description. Environment dynamics are defined as the key relationships that need to be identified in order for the agent to accomplish the goal. For instance, in our environment, environment dynamics is defined as all the monster and weapon killing relation pairs in the environment. Since it's not always possible to define a specific reward function for every environment, we would like the agent to explore the environment in a self-supervised manner, so that this policy can be easily generalized to new environments. To do so, We assume the agent is curious, which is an intrinsic behavior of humans, and propose an approach for the agent to learn an exploration

policy. We also propose a meta-training scheme to help with the learning, which helps the agent quickly learn given a novel set of dynamics. We aim to show that the learnt exploration policy successfully recognizes key dynamics of the environment. Finally, we propose a method to generate the natural language description of environment dynamics from the agent’s trajectories. We have demonstrated that our text generation approach outperforms the baseline approach by generating documents that have higher BLEU scores.

Chapter 2

Related Work

2.1 Language Grounding

Language grounding refers to finding correspondences between language and non-linguistic modalities, such as images, videos, games, etc. It is essential to many tasks, such as image captioning [58, 62, 63, 23, 30, 10], video captioning [56, 64, 60, 61, 29], visual question answering [1, 32, 15], video question answering [28], instruction following [3, 40, 14, 5], instruction generation [9], policy learning given text that conveys domain knowledge [7, 39], game playing [59], etc. In [65], the agent learns a policy that can be generalized to new environment instances given a text description about environment dynamics and a goal, which combines high-level instruction following and learning from domain knowledge. Inspired by Zhong’s work, we aim to do the reverse of the task, in which the agent is expected to generate a text description about the environment dynamics by interacting with the environment.

2.2 Learning by Interaction for Embodied Agents

Work on embodied agents has made huge progress in recent years. Early works used logical representations of the environment without real-world perception [33, 8], and assumed no interaction with the environment. Later works used RGB cameras to capture real world images or simulated environments to enable more realistic perception [50, 16]. Recent works start to treat agents not as passive executors, but as active learners that can interact with the environment, such as [66, 18, 38], or humans [55, 48]. In our work, we enable the agent to learn by interacting with a simulated game environment.

2.3 Curiosity-Driven Exploration

Curiosity-driven exploration is a well-studied topic in the reinforcement learning. An overview of the topic was given in [41, 42]. When the extrinsic reward is extremely sparse, or it is not possible to construct a shaped reward function, the agent’s intrinsic motivation or curiosity to discover novel states becomes essential. Within the curiosity-driven exploration framework, agents are encouraged to take actions that result in states that are surprising. A variety of intrinsically motivated reinforcement learning algorithms are proposed in recent years, including [36, 6, 22, 44, 49], etc. We choose to adapt the curiosity-driven learning approach proposed by [44], which generates an intrinsic reward signal based on the difficulty of predicting the consequence of the agent’s own actions. The harder it is to predict the resulting state, the higher the reward is, since the difficulty of predicting the resulting state indicates the level of surprise brought by the action.

2.4 Meta Learning

Meta learning has been an active research area for recent years, which has been successfully applied to many problems, including few-shot learning [57, 53, 46], continual learning [52, 47, 37], unsupervised learning [34], curiosity-driven learning [2], and neural architecture search [67, 19]. Common meta learning approaches include learning an optimization scheme for updating model parameters [4, 31], learning good parameter initialization for fast adaption [12, 13, 24], and learning through external memory updates [27]. We choose to adapt the external memory approach proposed by [27], and use the hidden state of an LSTM as the external memory to support meta learning.

Chapter 3

Task

3.1 Summary of the Task in Zhong et al. 2020

In the Read to Fight Monsters (RTFM) task proposed by Zhong et al. [65], the agent is presented with text that summarizes environment dynamics, the goal, and environment observations. Environment dynamics include all the monster and weapon relationships (E.x. Weapon 1 can kill Monster 1, Weapon 2 can kill Monster 2) and team assignment of each monster (E.x. Monster 1 belongs to Group 1, Monster 2 belongs to Group 2). the written goal contains the target group that we want the agent to destroy. By reading the document, the agent needs to perform the subsequent actions of identifying the target team, identifying the monsters that belongs to the team, identifying which monster is in the world, identifying which weapon is effective in killing the monster, picking up the appropriate weapon, and killing the monster. In each episode, the environment consists of a sample of groups, monsters and weapons. To ensure the agent does not learn by memorizing, the environment dynamics differ every episode.

3.2 Our Task

Our task is the reverse version of the RTFM task. As opposed to having an agent complete the goal by reading the document, we would like to have an agent that navigates through the environment and then generates the document that describes the environmental dynamics through interaction with the environment (more precisely, by learning to pick up different weapons to try battling with the monsters). We remove the team assignment since it's not possible for an agent to learn in an episode, and we also remove the goal since the goal should be specified by the game maker. Therefore, our target document only consists of the environment dynamics, which are all the monster and weapon pairs.

Mathematically, we have a 4×4 grid world, an agent, two monsters denoted by M_1 and M_2 , and two weapons denoted by W_1 and W_2 . The document D summarizes the killing relationships, which are W_1 kill M_1 and W_2 kill M_2 , and W_1, W_2, M_1, M_2 are different in

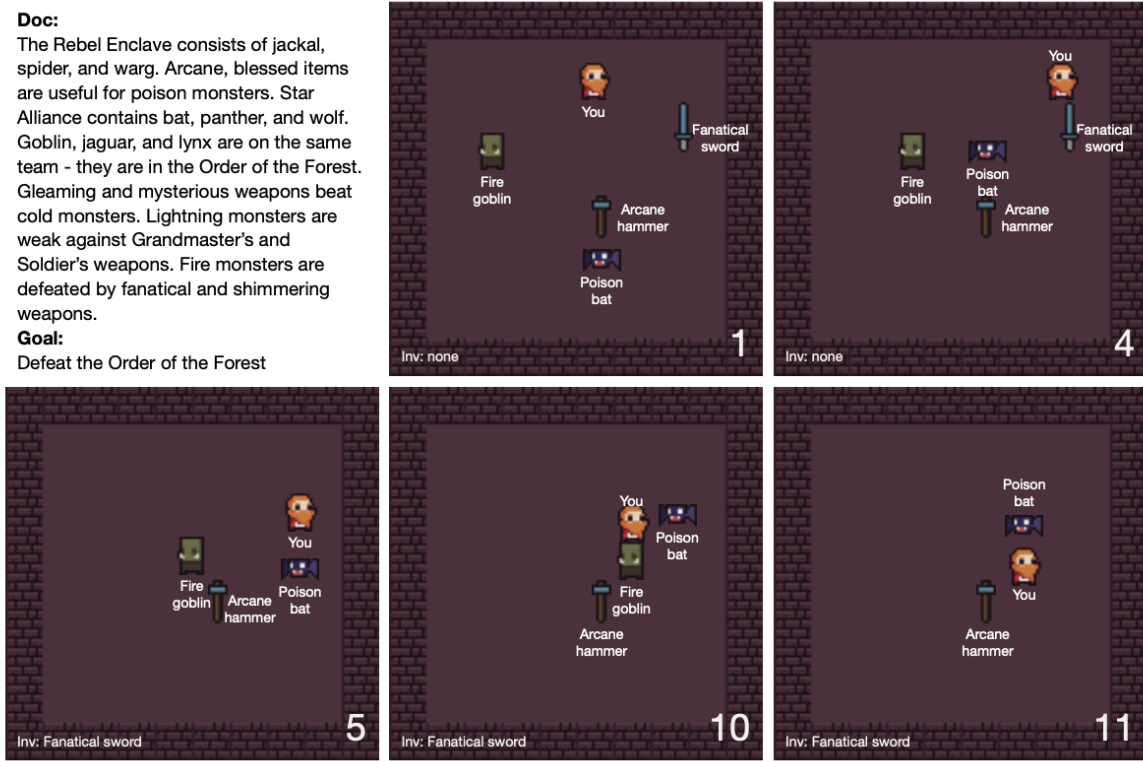


Figure 3.1: RTFM Environment [65]

each episode. At each timestep t , A is given environment observation o_t , which summarizes the locations of the agent, monsters and weapons. It is also given its inventory i_t , which contains the weapons the agent currently holds (which is limited to 1 in our case), and the agent's position p_t . Given the agent's trajectories $\{o_t, i_t, p_t\}_0^t$, we expect to generate the text that describes the relationships W_1 kills M_1 , W_2 kills M_2 in natural language.

Our task consists of two subtasks:

1. first, use self-supervised method to learn exploration policy for environment dynamics
2. then, collect the agent's trajectories using the exploration policy learnt in task 1, and then transform agent trajectories to textual description of environment dynamics

Chapter 4

Model for Learning an Exploration Policy

We propose a model similar to GAN [17], which consists of a policy generator and a forward model that acts as the discriminator. Let (s, a) denote a state-action pair, with state space S and action space A . Define policy generator as: $\pi: S \rightarrow A$. Define forward model as: $\mu: S \times A \rightarrow S$. Define *diff* as a measure of difference between two states. Define *re* as the true state output. Define θ as the parameters for policy generator, and ϕ as the parameters for forward model. We frame the learning process as:

$$\min_{\theta} \max_{\phi} - \text{diff}(\text{re}(s, \pi(s)) \parallel \mu(s, \pi(s)))$$

4.1 Forward Model

The forward model takes in a state-action pair (s, a) and predicts the next state s' . In order to minimize the difference between the real state and the predicted state, the state output should be as similar to the real state as possible. In other words, the forward model tries to output the next state precisely.

4.2 Policy Generator

The policy generator takes in a state s and predicts the best action the agent should take a . The action should help to maximize the difference between the real state and the predicted state by the forward model, which means that it should output the action that confuses the forward model the most, so that the forward model doesn't know what the next state is after taking this action. In other words, the forward model serves as a measure of how well the agent knows about current environmental dynamics, and the policy generator tries to generate a policy which guides the agent to explore the remaining environmental dynamics that the forward model doesn't know about.

4.3 Meta Training and Model Architecture

Our goal is to make the learning process within each environment dynamics faster and faster. Therefore, we propose a meta-learning scheme to train both the forward model and the policy generator.

Motivation for Meta Training

There are two levels of learning happening. The lower level of learning is instance learning. Within a single set of dynamics, the model should learn all the weapon and monster pairs by remembering past experience. At the higher level of learning, the model should learn to learn each environment dynamics faster whenever it encounters a new set of dynamics. Therefore, the problem can be framed as a meta-learning problem. Within each dynamics, we use LSTM hidden states to remember all the past experience. Between every set of dynamics, the model updates its weights to learn faster whenever it encounters a new set of dynamics.

Data Collection for Forward Model

We frame the training of the forward model as a supervised learning problem. We begin with data collection. Using a random exploration policy, we collect all trajectories happen within the same environment dynamics as a single data point, which is a $M \times N$ matrix, where M = number of dynamics, N = number of episodes per dynamics \times number of timesteps per episode. Let each state be a tuple of $s_t = (o_t, i_t, p_t)$. Let each trajectory be a tuple of (s_{t-1}, a_t, s_t) . For every environment dynamics, we start with an empty list, and subsequently append the trajectory happen at each timestep to the list. For each data point, the list of (s_{t-1}, a_t) is the input, and the list of (s_t) is the label.

Algorithm 1 Data Collection Algorithm for Forward Model

```

INITIALIZE dataset = []
for every environment dynamics do
  data = []
  for every episode do
    for every timestep do
       $s_{t-1} = [o_{t-1}, i_{t-1}, p_{t-1}]$ 
      take action
       $s_t = [o_t, i_t, p_t]$ 
      data +=  $[(s_{t-1}, a_t), s_t]$ 
    end for
  end for
  dataset += [data]
end for

```

Architecture of the Forward Model

We first concatenate a_t with s_{t-1} , and then process the combined vector through an LSTM and two additional linear layers to get the target output_state.

Training the Forward Model

We reset the hidden state of LSTM for each data point. In other words, we reset it for each environmental dynamics.

We frame the measurement of the difference between real and predicted observation as a multi-label classification problem. We represent both the original and the target observation as binary vectors, with 1 indicating some monster or weapon is present and 0 otherwise. We frame that of the inventory in the same way. We use BCE with logits loss as a measurement of error for both observation and inventory, and the loss are denoted by L_{obs} and L_{inv} respectively. Since the agent can only be on one of the grid cells at a time, we frame the measurement of the difference between real and predicted agent’s position as a single-label classification problem. The input is the relative original position, and the label is the absolute target position. We use the cross entropy loss, and the loss is denoted by L_{pos} . We calculate the total loss by adding up the three separate losses from observation, inventory, and the agent’s position, which is denoted by L_{total} , so that:

$$L_{total} = L_{obs} + L_{inv} + L_{pos}$$

Architecture of the Policy Generator

The policy generator implements actor critic [26], using a LSTM with a MLP feature extractor. The MLP feature extractor consists of a shared layer, and non-shared layers for the value network and the policy network.

Training the Policy Generator

We consider two settings of training. In the first setting, We first fully train the forward model, and then meta-train the policy generator. In the second setting, we jointly train the forward model and the policy generator from scratch. At each environmental dynamics, we initialize a zero hidden state. At each timestep, we feed last state, hidden state and action into the forward model to get a prediction of current state, and an updated hidden state. Then we get the difference between the prediction and true current state, and award the policy generator this value.

Algorithm 2 Meta-Training Algorithm for Policy Generator

```
for every environmental dynamics do  
  hidden_state = []  
  for every episode do  
    for every timestep do  
       $s_{t-1} = [o_{t-1}, i_{t-1}, p_{t-1}]$   
      take action  
       $s_t = [o_t, i_t, p_t]$   
      intrinsic_reward, hidden_state = diff(forward_model( $s_{t-1}$ ,  $a_t$ , hidden_state),  $s_t$ )  
      agent_reward = intrinsic_reward  
    end for  
  end for  
end for
```

Chapter 5

Model for Text Generation

We use a Seq2Seq model [54] for transforming the agent trajectories to text description. The model typically consists of an encoder for encoding context and a decoder for generating the resulting sequence.

5.1 Data Collection

We collected agent trajectories using a pre-trained model from the RTFM repository. Each data point consists of all the steps with the same environment dynamics, which share the same wiki document that describes the environment dynamics. Each step is recorded with the observation, the inventory, and the positions of every cell relative to the position of the agent, normalized by the width and height of the grid world.

5.2 Architecture and Training of the Seq2Seq Model

We first fuse the observation o_t and inventory i_t through a fusion module, and then feed the fused vector through the encoder, and use the output hidden state as the initial hidden state for the decoder. For each wiki, we remove the last word and add a “START” token in the beginning as input to the decoder, using the original wiki as the output target. We use teacher forcing in training, and greedy decoding in evaluation.

Fusion Module

A fusion module is in charge of fusing all the given information and producing a dense vector, in this case it should process the observation and inventory. In the original work [65], it also processes the wiki and goal. We try using two versions of fusion modules: the fusion module of the CNN baseline model and that of the FiLM baseline model from [65], with the removal of wiki document that describes environment dynamics and goal as input.

The fusion module of the CNN baseline model consists of 5 layers of convolutions with

channels of 16, 32, 64, 64, and 64, with residual connections from the 3rd layer to the 5th layer. The initial inputs to the network are the concatenation of observation and inventory, and the agent’s relative positional feature. At each convolutional layer, we feed in the output from the previous layer and the agent’s relative positional feature. The inventory is encoded using a bidirectional LSTM before being fed into the network.

The fusion module of the FiLM baseline replaces the convolutional layers in the CNN model with a FiLM layer from [45], which has been shown to be highly effective for visual reasoning. FiLM layer can be used to transform a neural network’s architecture, conditioned on an arbitrary input. In [65], the combined text which consists of the document D that describes environment dynamics, inventory i , and goal G are used to modulate observation o and position p features. Since the agent doesn’t know D and G , we used i to modulate o and p features.

We choose not to use the fusion module in `txt2 π` , which is the third option of model in [65], because it focuses on capturing three-way interactions between the goal, document describing environment dynamics, and environment observations. However, we don’t want to feed in the goal and the wiki document, so the model isn’t suitable for our case.

Chapter 6

Experimental Setup for Learning an Exploration Policy

This section describes the details of the experimental setup for learning an exploration policy for environment dynamics.

6.1 Environment

The environment we evaluate on is similar to the environment of the 'Read to Fight Monster' task described in [65], with stationary monsters and no group assignments. In the 4×4 grid world, two monsters (M_1, M_2), two weapons (W_1, W_2) are present. Each weapon can be used to kill one of the monsters respectively. An agent is presented in the environment. The agent has a inventory, which can hold one weapon. There are five actions available for the agent: LEFT, RIGHT, UP, DOWN, PICKUP. If an agent is on a cell with a weapon on it, it can choose to pick it up or not. If the agent chooses to pick up a new weapon with an old weapon already in its backpack, the agent will automatically drop the old weapon and replace it with the new one. If the agent steps on the same cell with a monster with an appropriate weapon in its inventory, it will automatically kill the monster. If it encounters a monster with an empty backpack, or with an inappropriate weapon in its backpack, nothing will happen. If the agent attempts to step off the grid, nothing will happen.

The environment is built in Stable Baselines [21], a deep reinforcement learning framework based on OpenAI Baselines [11]. For each environment dynamics we collect 300 timesteps across episodes, and each episode within it terminates either when the agent discovers all the weapon and monster relationship pairs, which are 4 pairs in this case (W_1 can kill M_1 , W_2 can kill M_2 , W_1 cannot kill M_2 , W_2 cannot kill M_1), or when the maximum number of 30 steps is exceeded. The number of pairs of relationships discovered by the agent is tracked by a knowledge vector. The knowledge vector is initialized with a zero vector of length 4, and if a relationship is discovered by the agent, the corresponding component of the vector will be set to 1. In the settings when external rewards are used, the agent gets a reward of

+1 whenever a component of the knowledge vector is set to 1.

6.2 Visualization of the Environment

We use PyGame, a game programming python library, to visualize the environment. In order to create a realistic setting, we render images of monsters, weapons and a robot onto the grid world. Visualization helps with qualitatively evaluating the model.

6.3 Dataset for the Forward Model

To ensure each episode has the same length, we modified the termination condition a little bit. Different from the environment in training setting, each episode only terminates when it reaches the maximum number of steps of 30, but not when it discovers all the killing relationship pairs. We conduct the experiments with 2 settings. In the first setting, Each 300 steps, which consists of 10 episodes, is collected as a single data point with the same environment dynamics. In the second setting, each 600 steps, which consists of 20 episodes is collected as a single data point. Due to memory limits, the dataset is not collected at once. In the 300 steps setting, we collect 8000 data points each time, while in the 600 steps setting, we collect 4000 data points each time. We recollect new dataset every 30 training epochs.

Class imbalance problem: We’ve noticed that there is an imbalance between the number of successful killing events and the number of unsuccessful killing events(the events when the agent attempts to kill a monster with the wrong weapon and doesn’t succeed). Initially, since we only force an episode to contain at least one killing event by ignoring those that don’t have one, the number of killing events far exceed non-killing events. To address this issue, we alternate between gathering episodes with killing events and episodes with non-killing events, which ensures a balance of killing and non-killing events in our dataset.

We also ensure that the environment dynamics that have appeared in the training set do not appear in the test set.

6.4 Training Details

For training the forward model, we use standard supervised learning approach with the ADAM optimizer [25].

We integrate the forward model as part of the environment and use it to calculate the intrinsic reward for the exploration policy at every timestep. In the joint training setting, we update it every environment dynamics with the ADAM optimizer. We use different deep reinforcement learning algorithms in Stable Baselines to train the policy network, including A2C [35], Double-DQN [20] and PPO [51]. We use a custom policy network to integrate an LSTM network to process the sequence of data, and tune it by modifying the number of

layers for the policy network and value network respectively. The final number of layers is 64 for both the policy network and value network.

6.5 Baseline Methods for Overall Training

Model with Extrinsic Reward: This baseline model rewards the agent extrinsically. Whenever the agent identifies a new killing relation pair, we give the agent +1 reward.

Model with Intrinsic Reward Only + Separate Training: This model rewards the agent intrinsically. At each step, we reward the agent with the loss returned by the forward model, which measures the difference between the predicted state and the real state. We first fully train the forward model, and then plug it in the policy generator without further training.

Model with Intrinsic Reward Only + Joint Training: This model also rewards the agent intrinsically in the same way as the previous model does. The difference is that, instead of plugging in a fully-trained forward model to the policy generator, we jointly train both the models from scratch. Within each episode, we update the hidden state of both the policy generator and the forward model. Between each environment dynamics, we update the parameters of both models.

6.6 Baseline Methods for Forward Model

To test whether the number of episodes is enough for the model to learn the environment dynamics, We compare the performance of learning from 10 episodes vs learning from 20 episodes.

Chapter 7

Experimental Setup for Text Generation

This section describes the details of the experimental setup for generating text descriptions from trajectories.

7.1 Dataset

We conduct our experiment with a training dataset consisting of 40000 data points, collected using the RTFM github repo with a pre-trained txt2 π model, which is the model with best performance in [65] that captures interactions between observations from the environment, goal and document using FiLM² layers, which is also proposed in [65]. Using the best-performing model to generate trajectories ensures the collected trajectories contain useful information. Each data point consists of trajectories with the same environment dynamics. Data points have varying lengths, so we pad the data with zeros to the same maximum length in the dataloader. We extract observation ('name'), inventory ('inv'), descriptive document ('wiki') and the relative position matrix ('rel_pos') from the trajectories.

The test dataset contains 200 data points. We also ensure the environment dynamics that have appeared in training set do not appear in the test set.

7.2 Baseline Methods

Model with Zero Hidden State: This baseline model removes the effect of the encoder, and consequently removes the effect of the trajectories.

Model with Hidden State from the Trajectories: This model first process the trajectories via the encoder, and uses the hidden state from encoder as the initial hidden state for the decoder.

7.3 Details of the Model

For the model with hidden state from the trajectories, we first feed the observation, inventory and the relative position matrix via the fusion module, and get a resulting vector of size $\text{batch_size} \times 1 \times 400$. Then we process the resulting vector via the LSTM encoder, getting a hidden state. Finally, we feed the wiki document with 'START' token added in the front and last word removed via the LSTM decoder, and initialize the hidden state as the hidden state from encoder. The encoder only consists of an LSTM layer, while the decoder consists of a word embedding layer, an LSTM layer, a linear layer that transforms the vector into the size of the vocabulary (263 in this case, which is measured by extracting the vocabulary from the RTFM github repo), and then a softmax function which gives us the word with the highest probability.

For the model with zero hidden state, we set the hidden state to a tuple of zero tensors before passing to the decoder, so that the decoder gets no information about the trajectories.

7.4 Evaluation Metric

The performance of the model is measured with BLEU score, which is a method for the automatic evaluation of machine translation [43]. The goal of the BLEU score is to compare machine translations against human-generated translations. BLEU score ranges from 0 to 1. The higher the score is, the more similar the predicted sequence is to the target sequence.

Chapter 8

Experiments

8.1 Performance of Model for Learning an Exploration Policy

Performance of Model with Extrinsic Reward vs Intrinsic Reward

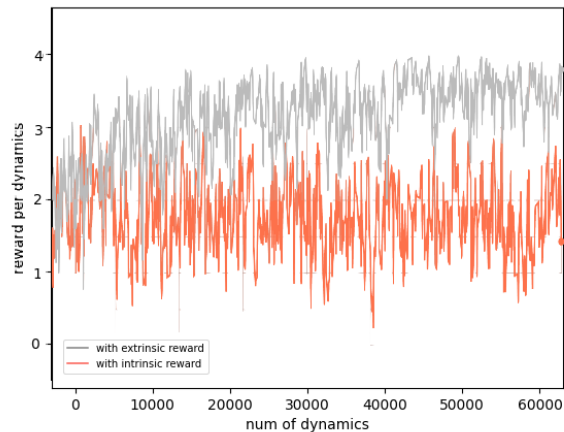


Figure 8.1: Graph of the number of monster/weapon pairs learnt by the model within 300 steps using extrinsic vs intrinsic reward. The model with extrinsic reward performs better than the model with intrinsic reward.

We compare the performance of the model with extrinsic reward with the one with intrinsic reward only. For the model with intrinsic reward, We choose the one with pre-trained forward model plugged in policy generator over the jointly trained one, since the first one achieves better experimental result. In the experiment, we record how many pairs

of monster, weapon relation can be discovered within 300 steps, which consists of 10 episodes, each with a maximum of 30 steps. Environment dynamics remains the same within the 300 steps. We expect the model with extrinsic reward to outperform the one with intrinsic reward, and the result confirms that. The result is confirmed by Figure 8.1. We can observe that as the number of training dynamics increased, the number of monster and weapon pairs discovered by the agent within 300 steps finally converge to 4, which is the maximum number of monster and weapon pairs in a 2 agents \times 2 weapons setting with extrinsic reward. However, the number of monster and weapon pairs discovered by the agent does not increase over time in the setting with intrinsic reward, which is left for further diagnosis in the next experiments.

Intrinsic Reward over Time

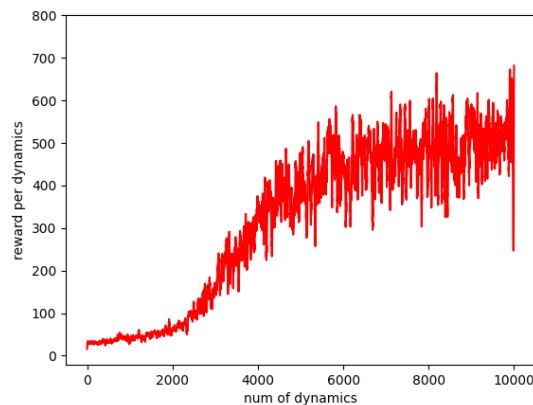


Figure 8.2: Graph of the intrinsic reward of plugging pre-trained forward model in policy generator. Reward first increases and later converges, which complies with the fact that the number of monster/weapon pairs learnt by the model within 300 steps using intrinsic reward fails to increase after a short time.

We graph the intrinsic reward over time to see the overall trend. If the model with intrinsic reward works ideally, the intrinsic reward should increase over time. Figure 8.2 shows that the intrinsic reward initially increases over time, but then stays constant later. This complies with the fact that the number of monster and weapon pairs identified by the agent within a fixed time length does not increase over time.

Performance of the Joint Training Model

In the previous settings, we plug in the fully-trained forward model in the policy generator. In this experiment, we jointly train both the forward model and the policy generator from

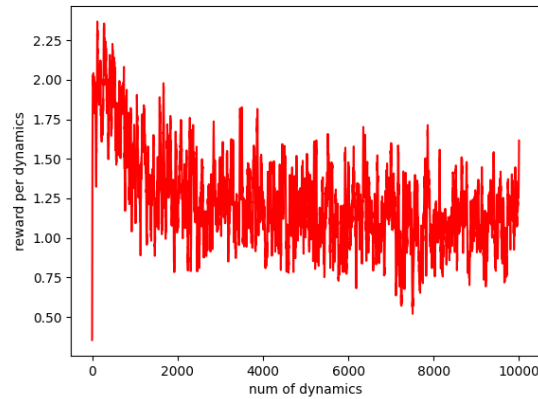


Figure 8.3: Graph of the number of monster/weapon pairs learnt by the model within 300 steps using jointly training of forward model and policy generator. The number first decreases and later converges at a low level, which is worse than the performance of the model with pre-trained forward model plugged in policy generator.

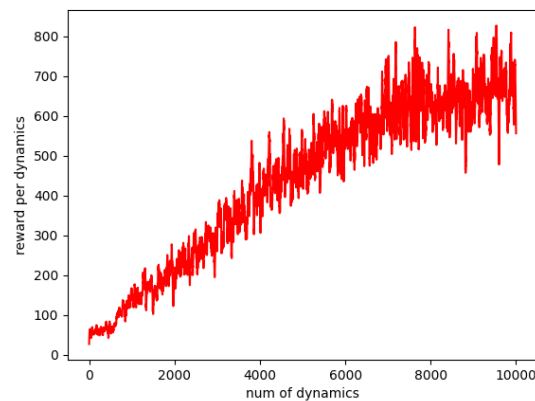


Figure 8.4: Graph of the intrinsic reward of jointly trained forward model and policy generator. Reward first increases and later converges.

scratch, updating the parameters of each model every environment dynamics. To our surprise, the number of monster and weapon pairs identified by the agent first decreases over time, and then converge at a low level. Intrinsic reward in this setting follows similar pattern to the one in the previous setting, in which it first increases and then stays constant after convergence.

Quantitative Performance of the Forward Model

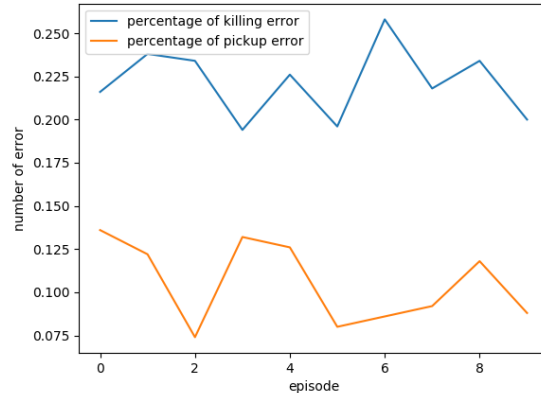


Figure 8.5: Graph of the error of forward model over time. The number of pickup error stays constant, which meets our expectation. The number of pickup error also stays constant, which is unexpected, since we expect the number to decrease over time.

We want to check whether the reason for causing the above unexpected result is with the forward model. We fully train a forward model, and then measure its error rate in killing events and pickup events over the episodes with the same environment dynamics. We expect the pickup error rate to stay constant over time, since the effect of PICKUP does not vary across dynamics and the model should already know "PICKUP" action means picking up a weapon after enough training. On the other hand, we expect the killing error rate to decrease over episodes, since the killing relationships do vary across dynamics. At the beginning, the model does not know about any environment dynamics, and consequently, is expected to make errors on whether a weapon is able to kill a monster. After the model has viewed some episodes, in which it learns which weapon can kill which monster (if a weapon can be used to kill a monster, the monster will disappear if the agent steps on the monster with the weapon in its backpack), it should make fewer killing prediction mistakes. Figure 8.5 shows the pickup and killing error rate of a fully-trained forward model with 10 episodes per environment dynamics. Pickup error rate remains stable as expected, while killing error rate does not decrease over time, which is unexpected.

One concern that this result raises is that 10 episodes may not be enough for the model to learn the environment dynamics. To test this hypothesis, we fully trained the forward model with 20 episodes per environment dynamics. The experimental result shows that the killing error stays around 0.24 and pickup error rate stays around 0.06, both are nearly constant without a decreasing trend, which is similar to the previous setting. The result indicates that the forward model needs further diagnosis.

Qualitative Performance of the Forward Model

Besides quantitative evaluation, we also qualitatively evaluate the performance of the model. We render observations at each timestep using PyGame. The result complies with the quantitative evaluation, which shows that the agent can pick up the weapon almost every time with "PICKUP" action. However, the killing error does not decrease very much as the number of episodes increases. Most of the time, killing is completed successfully. However, sometimes the model can foresee a successful killing before viewing any other episodes, which is unexpected. Sometimes the agent holds an appropriate weapon but the model prediction doesn't kill the monster after it views the same killing happened in previous episodes, which is also unexpected. The result is shown in figs. 8.6 and 8.7

8.2 Performance of Model for Text Generation

We evaluate the performance of the models quantitatively and qualitatively. The experimental result indicates that the model with hidden state from the trajectories outperforms the model with zero hidden state. The BLEU score of the best performing model with hidden state from the trajectories is 0.65, while that of the best performing model with zero hidden state is 0.38.

Qualitatively, the model with hidden state from the trajectories has a roughly consistent performance in terms of the weapon and monster pairs, while the model with zero hidden state has very inconsistent performance, some are good but some are bad. The inconsistent performance can be explained by the fact that a model without any information given from the trajectories is likely to select random weapon and monster pairs, although it can predict the correct structure since the structure of the wiki document is fixed. The results are shown below:

Example of good performance from zero hidden state model:

prediction: gleaming beat cold, shimmering beat fire, grandmasters beat lightning, blessed beat poison, panther are order of the forest, panther are rebel enclave, panther are star alliance.

groundtruth: blessed beat cold, shimmering beat fire, grandmasters beat lightning, gleaming beat poison, panther are order of the forest, jaguar are rebel enclave, wolf are star alliance.

BLEU score: 0.7198173100035995

Example of bad performance from zero hidden state model:

prediction: dagger, panther are rebel enclave, panther are star alliance.

groundtruth: gleaming beat cold, blessed beat fire, grandmasters beat lightning, shimmering

beat poison, wolf are order of the forest, panther are rebel enclave, jaguar are star alliance.
BLEU score: 0.11018078282769957

Example of performance from the trajectories' hidden state model:

prediction: gleaming beat cold, shimmering beat fire, grandmasters beat lightning, blessed beat poison, wolf are order of the forest, wolf are rebel enclave, wolf are star alliance.

groundtruth: shimmering beat cold, gleaming beat fire, grandmasters beat lightning, blessed beat poison, jaguar are order of the forest, panther are rebel enclave, wolf are star alliance.

BLEU score: 0.7198173100035995

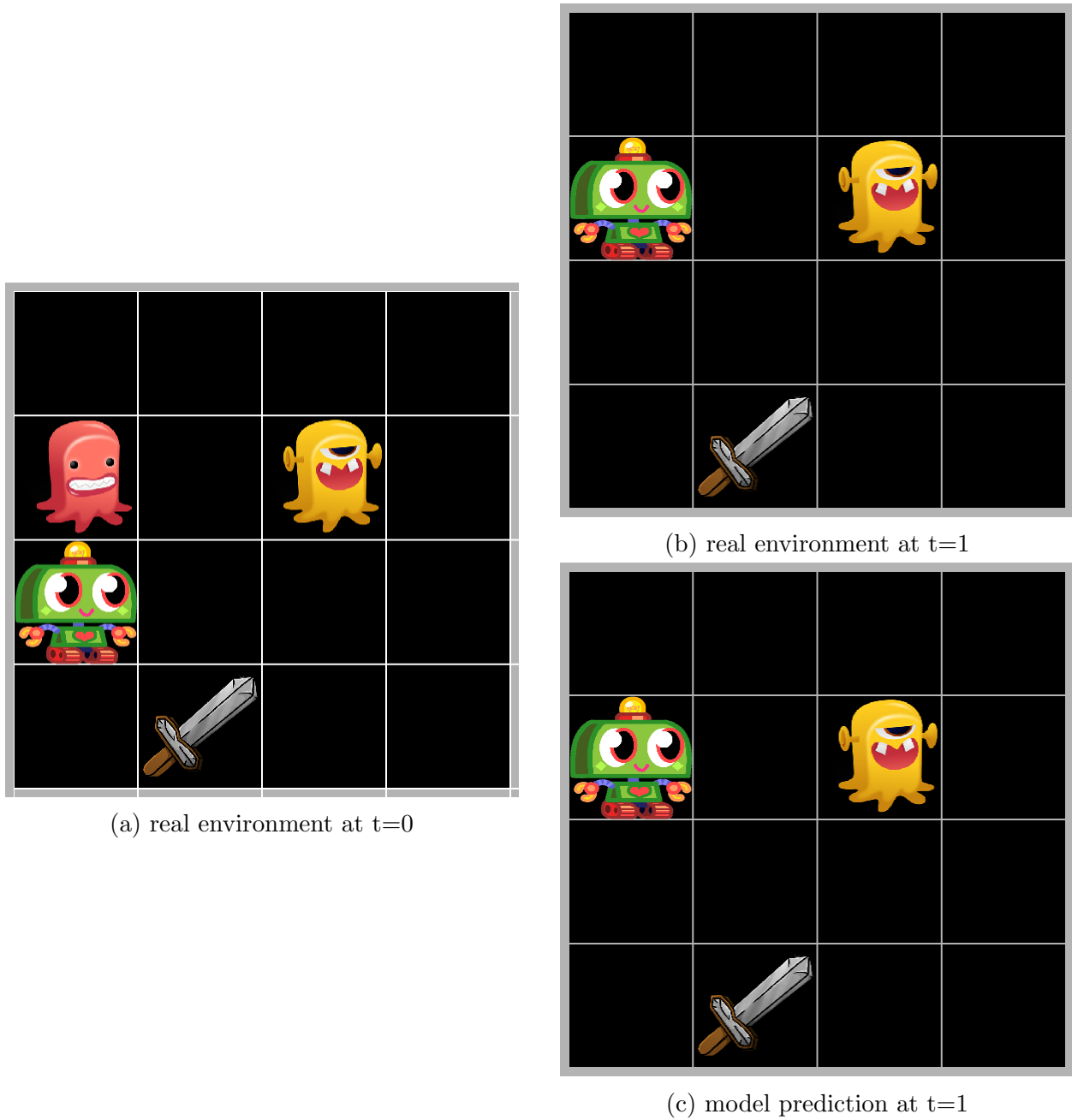


Figure 8.6: Example of successful killing. At $t=1$, the monster is killed in real environment, and it is also killed in the model prediction.

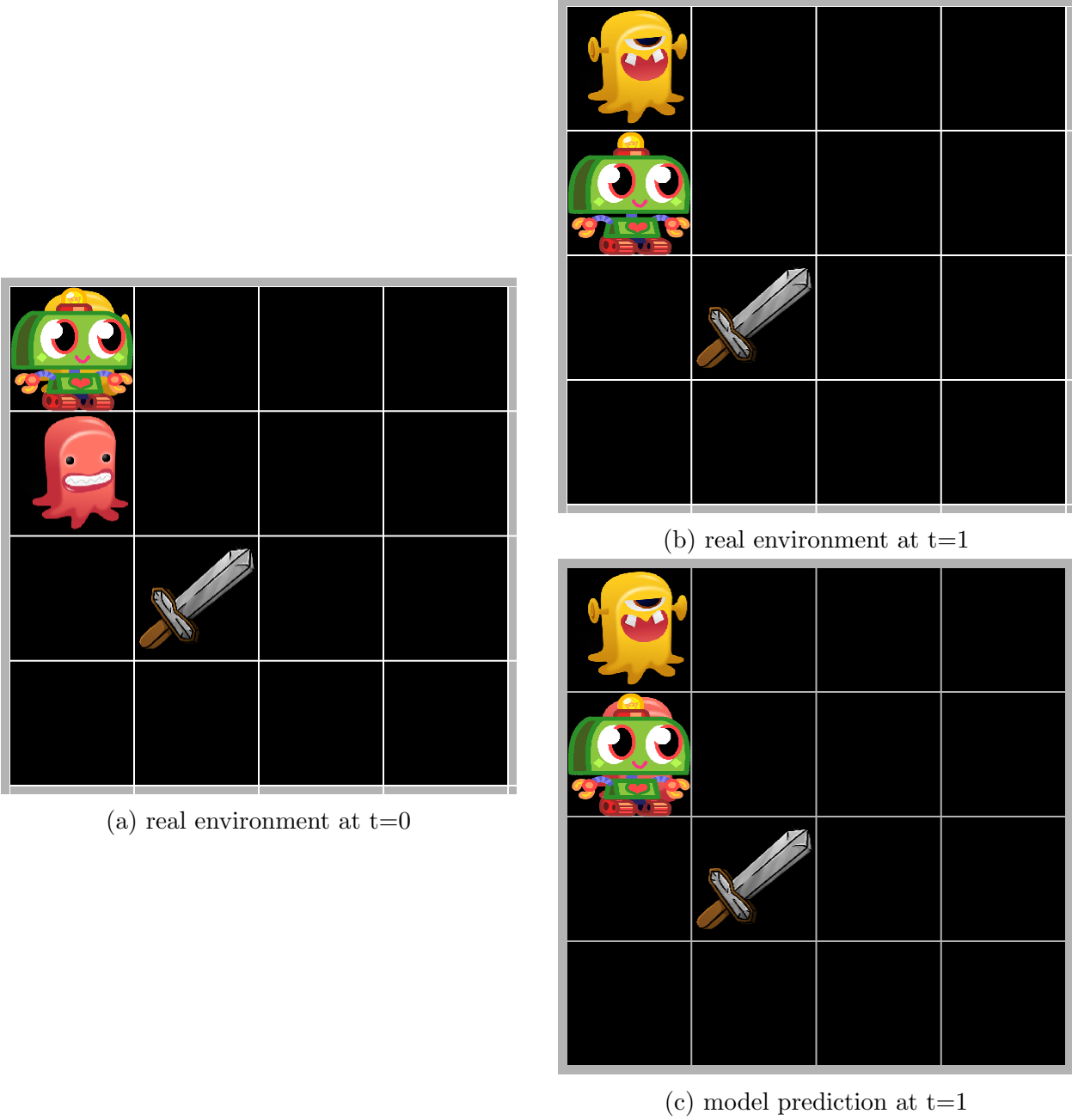


Figure 8.7: Example of unsuccessful killing. At t=1, the monster is killed in real environment, but in the model prediction the monster is still alive.

Chapter 9

Conclusion

We proposed a curiosity-driven approach for generating textual descriptions of environment dynamics, which is a task that hasn't been tackled before. In detail, we framed the task of learning an exploration policy for trajectory collection as a two-player game, and proposed a self-supervised learning approach to solve it, which allows generalization to new environments by eliminating the need of designing a reward function for each specific environment. We also proposed a framework to transform agent trajectories to textual descriptions, which performs more consistently and reaches a higher BLEU score than the baseline method. Further directions for future work include enabling end-to-end learning of the whole process.

Bibliography

- [1] Aishwarya Agrawal et al. “VQA: Visual Question Answering”. In: *ICCV 2015*.
- [2] Ferran Alet et al. “Meta-Learning Curiosity Algorithms”. In: *ICLR 2020*.
- [3] Peter Anderson et al. “Vision-and-Language Navigation: Interpreting Visually-Grounded Navigation Instructions in Real Environments”. In: *CVPR 2018*.
- [4] Marcin Andrychowicz et al. “Learning to Learn by Gradient Descent by Gradient Descent”. In: *NIPS 2016*.
- [5] Dzmitry Bahdanau et al. “Learning to Understand Goal Specifications by Modelling Reward”. In: *ICLR 2019*.
- [6] Marc G. Bellemare et al. “Unifying Count-Based Exploration and Intrinsic Motivation”. In: *NIPS 2016*.
- [7] S.R.K. Branavan, David Silver, and Regina Barzilay. “Learning to Win by Reading Manuals in a Monte-Carlo Framework”. In: *JAIR 2012*.
- [8] David L. Chen and Raymond J. Mooney. “Learning to Interpret Natural Language Navigation Instructions from Observations”. In: *AAAI 2011*.
- [9] Andrea F. Daniele, Mohit Bansal, and Matthew R. Walter. “Navigational Instruction Generation as Inverse Reinforcement Learning with Neural Machine Translation”. In: *HRI 2017*.
- [10] Chaorui Deng et al. “Length-Controllable Image Captioning”. In: *ECCV 2020*.
- [11] Prafulla Dhariwal et al. *OpenAI Baselines*. <https://github.com/openai/baselines>. 2017.
- [12] Chelsea Finn, Pieter Abbeel, and Sergey Levine. “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks”. In: *ICML 2017*.
- [13] Chelsea Finn, Kelvin Xu, and Sergey Levine. “Probabilistic Model-Agnostic Meta-Learning”. In: *NeurIPS 2018*.
- [14] Daniel Fried et al. “Speaker-Follower Models for Vision-and-Language Navigation”. In: *NIPS 2018*.
- [15] Peng Gao et al. “Question-Guided Hybrid Convolution for Visual Question Answering”. In: *ECCV 2018*.

- [16] Xiaofeng Gao et al. “VRKitchen: an Interactive 3D Environment for Learning Real Life Cooking Tasks”. In: *ICML 2019 Workshop on Reinforcement Learning for Real Life*.
- [17] Ian J. Goodfellow et al. “Generative Adversarial Nets”. In: *NIPS 2014*.
- [18] Daniel Gordon et al. “IQA: Visual Question Answering in Interactive Environments”. In: *CVPR 2018*.
- [19] Yiming Yang Hanxiao Liu Karen Simonyan. “DARTS: Differentiable Architecture Search”. In: *ICLR 2019*.
- [20] Hado van Hasselt, Arthur Guez, and David Silver. “Deep Reinforcement Learning with Double Q-learning”. In: *AAAI 2016*.
- [21] Ashley Hill et al. *Stable Baselines*. <https://github.com/hill-a/stable-baselines>. 2018.
- [22] Rein Houthoofd et al. “VIME: Variational Information Maximizing Exploration”. In: *NIPS 2016*.
- [23] Lun Huang et al. “Attention on Attention for Image Captioning”. In: *ICCV 2019*.
- [24] Mirantha Jayathilaka. “Enhancing Generalization of First-Order Meta Learning”. In: *ICLR 2019*.
- [25] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *ICLR 2015*.
- [26] Vijay R. Konda and John N. Tsitsiklis. “Actor Critic Algorithms”. In: *NIPS 1999*.
- [27] Brenden M. Lake. “Compositional Generalization through Meta Sequence-to-Sequence Learning”. In: *NeurIPS 2019*.
- [28] Thao Minh Le et al. “Hierarchical Conditional Relation Networks for Video Question Answering”. In: *CVPR 2020*.
- [29] Jie Lei et al. “MART: Memory-Augmented Recurrent Transformer for Coherent Video Paragraph Captioning”. In: *ACL 2020*.
- [30] Guang Li et al. “Entangled Transformer for Image Captioning”. In: *ICCV 2019*.
- [31] Ke Li and Jitendra Malik. “Learning to Optimize”. In: *ICLR 2017*.
- [32] Jiasen Lu et al. “Hierarchical Question-Image Co-Attention for Visual Question Answering”. In: *NIPS 2016*.
- [33] Cynthia Matuszek, Dieter Fox, and Karl Koscher. “Following Directions Using Statistical Machine Translation”. In: *HRI 2010*.
- [34] Luke Metz et al. “Meta-Learning Update Rules for Unsupervised Representation Learning”. In: *ICLR 2019*.
- [35] Volodymyr Mnih et al. “Asynchronous Methods for Deep Reinforcement Learning”. In: *ICML 2016*.

- [36] Shakir Mohamed and Danilo J. Rezende. “Variational Information Maximisation for Intrinsically Motivated Reinforcement Learning”. In: *NIPS 2015*.
- [37] Anusha Nagabandi et al. “Learning to Adapt in Dynamic, Real-World Environments Through Meta-Reinforcement Learning”. In: *ICLR 2019*.
- [38] Tushar Nagarajan and Kristen Grauman. “Learning Affordance Landscapes for Interaction Exploration in 3D Environments”. In: *NeurIPS 2020*.
- [39] Karthik Narasimhan, Regina Barzilay, and Tommi Jaakkola. “Grounding Language for Transfer in Deep Reinforcement Learning”. In: *JAIR 2018*.
- [40] Daniel Nyga et al. “Grounding Robot Plans from Natural Language Instructions with Incomplete World Knowledge”. In: *CoRL 2018*.
- [41] Pierre-Yves Oudeyer, Frdric Kaplan, and Verena V. Hafner. “Intrinsic Motivation Systems for Autonomous Mental Development”. In: *Evolutionary Computation 2007*.
- [42] Pierre-Yves Oudeyer and Frederic Kaplan. “What is Intrinsic Motivation? A Typology of Computational Approaches”. In: *Frontiers in Neurorobotics 2009*.
- [43] Kishore Papineni et al. “BLEU: a Method for Automatic Evaluation of Machine Translation”. In: *ACL 2002*.
- [44] Deepak Pathak et al. “Curiosity-driven Exploration by Self-supervised Prediction”. In: *ICML 2017*.
- [45] Ethan Perez et al. “FiLM: Visual Reasoning with a General Conditioning Layer”. In: *AAAI 2018*.
- [46] Mengye Ren et al. “Meta-Learning for Semi-Supervised Few-Shot Classification”. In: *ICLR 2018*.
- [47] Samuel Ritter et al. “Been There, Done That: Meta-Learning with Episodic Recall”. In: *ICML 2018*.
- [48] Homero Roman Roman et al. “RMM: A Recursive Mental Model for Dialogue Navigation”. In: *EMNLP Findings 2020*.
- [49] Nikolay Savinov et al. “Episodic Curiosity through Reachability”. In: *ICLR 2019*.
- [50] Manolis Savva et al. “Habitat: A Platform for Embodied AI Research”. In: *ICCV 2019*.
- [51] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].
- [52] Maruan Al-Shedivat et al. “Continuous Adaptation via Meta-Learning in Nonstationary and Competitive Environments”. In: *ICLR 2018*.
- [53] Jake Snell, Kevin Swersky, and Richard S. Zemel. “Prototypical Networks for Few-Shot Learning”. In: *NIPS 2017*.
- [54] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. “Sequence to Sequence Learning with Neural Networks”. In: *NIPS 2014*.

- [55] Jesse Thomason et al. “Improving Grounded Natural Language Understanding through Human-Robot Dialog”. In: *ICRA 2019*.
- [56] Subhashini Venugopalan et al. “Sequence to Sequence – Video to Text”. In: *ICCV 2015*.
- [57] Oriol Vinyals et al. “Matching Networks for One Shot Learning”. In: *NIPS 2016*.
- [58] Oriol Vinyals et al. “Show and Tell: A Neural Image Caption Generator”. In: *CVPR 2015*.
- [59] Sida I. Wang, Percy Liang, and Christopher D. Manning. “Learning Language Games through Interaction”. In: *ACL 2016*.
- [60] Xin Wang, Yuan-Fang Wang, and William Yang Wang. “Watch, Listen, and Describe: Globally and Locally Aligned Cross-Modal Attentions for Video Captioning”. In: *NAACL 2018*.
- [61] Xin Wang et al. “Video Captioning via Hierarchical Reinforcement Learning”. In: *CVPR 2018*.
- [62] Kelvin Xu et al. “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”. In: *ICML 2015*.
- [63] Quanzeng You et al. “Image Captioning with Semantic Attention”. In: *CVPR 2016*.
- [64] Haonan Yu et al. “Video Paragraph Captioning Using Hierarchical Recurrent Neural Networks”. In: *CVPR 2016*.
- [65] Victor Zhong, Tim Rocktaschel, and Edward Grefenstette. “RTFM: Generalising to Novel Environment Dynamics via Reading”. In: *ICLR 2020*.
- [66] Yuke Zhu et al. “Visual Semantic Planning using Deep Successor Representations”. In: *ICCV 2017*.
- [67] Barret Zoph and Quoc V. Le. “Neural Architecture Search with Reinforcement Learning”. In: *ICLR 2017*.