Literature Survey on Natural Language Processing Supports Reinforcement Learning

Vinit Bhosale
Masters in Computer Science

Rohit Marathe
Masters in Computer Science

Abstract

The aim of this survey to prove that NLP helps in Reinforcement learning, recently though NLP is used in various complex and graphical games to achieve near-human accuracy in the game. As we perform the experiment considering the tic-tac-toe as our reference game, we come across useful result to support user theory as well as we deep dive in the world of TextWorld to see it's amazing text games and their dependency over NLP.

1 Introduction

Neural systems are frequently improved for execution on a solitary undertaking. Conversely, human knowledge is in a general sense adaptable, and pertinent over a wide assortment of the task. It has been contended this speaks to a basic contrast in the illustrative structure of human insight, what's more, neural systems. In any case, we have contended that this distinction may emerge from the lavishness of human experience, which can be viewed as a huge arrangement of interconnected and commonly supporting assignments. We contend that human-like execution can emerge from profound learning frameworks on the off chance that they are given human-like understanding.

Specifically, regular language guidance is principal to quick human learning in pretty much every unique situation. It gives us both direct guidelines on basic unequivocal assignments, and with the framework to fabricate progressively complex aptitudes. Lake and partners have contended that neural systems are too information hungry to be a decent model for human conduct or to perform complex insight errands without the help of other thinking frameworks [5]. Can this distinction be somewhat clarified basically by the way that profound learning systems are not generally given natural language feedback on tasks?

We explore this question in the simple game-playing context of tic-tac-toe. It is simple enough to train a neural network to play tic-tac-toe using a reinforcement learning algorithm, but suppose we give it in addition to some natural language information about the task. This information may convey useful information about the task structure which is more difficult to obtain from the opaque feedback of a reinforcement learning signal[3]. This might result in more rapid learning, and perhaps in representations that are more suitable for future learning.

In this paper, we will also explore many advancements in this field by researchers as many new applications are coming up with NLP in the reinforcement for more complex games and graphical games like chess, DOTA. Additionally, look into Microsoft TextWorld which is an open-source, extensible engine that both generates and simulates text games. You can use it to train reinforcement learning (RL) agents to learn skills such as language understanding and grounding, combined with the sequential decision.

2 Related Work

There have also been several studies recently focusing on the other benefit of multi-task learning: it can potentially improve performance on a single task more than training on that task alone would [6, e.g.]. In some sense, this can be seen as a more complex form of regularization. Requiring a network to solve two tasks will restrict the representations it forms to those that are beneficial for both tasks. If both tasks convey

useful, distinct information about the world, this could potentially be quite beneficial.

3 Methodology

What is reinforcement learning?

To train a system that interacts (performs actions) with the environment given the observation such that it will receive the maximum accumulated reward at the end.

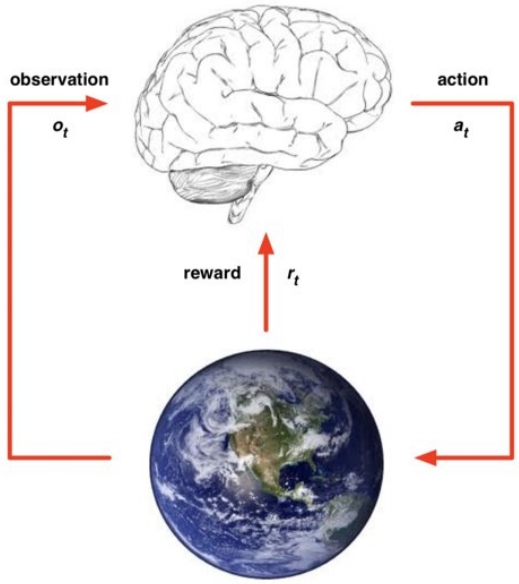


Figure 3. Reinforcement Learning Methodology.

4 Q-learning

One common technique for reinforcement learning is to learn a Q-function, a mapping from (state, proposed action) pairs to the value of the action. These Q-values must satisfy the following recurrence relation called the Bellman equation (for simplicity here we assume rewards depend only on the current state, not the action taken in that state, and that a greedy policy is used which always 2 takes the highest value action):

$$\mathbf{Q(st, at) = R(st+1) + \max a \in A\ Q(st+1, a)[3]}$$

where stand at is the state and action at time t, respectively, and R(s) is the reward for state s.

The Q-learning algorithm does this by playing the game many times and at the end of each move we make in each game, we study the reward we get and use the algorithm above to keep updating the table. Eventually, we will arrive at a set of optimal values. Pasted below is a Wikipedia sourced image of the Q-learning algorithm detailing how we make these updates. After making a move during learning, the Q value for a given state and action is replaced by the new value.

The new value is a sum of two parts. The first part is (1-learning rate)*old value. This is how much of the old value we retain. A learning rate of 0 will mean nothing new will be learned. A learning rate of 1 will mean the old value will be completely discarded.

The second part is the learning rate * (immediate reward for action + discounted estimate of optimal future value). The learning rate as explained above determines how much of the new learned value will be used. The learned value is the sum of immediate reward and discounted estimate of optimal future value. The discount factor determines the importance of future rewards. When set to 0, we will only consider immediate rewards and 1 will make the algorithm take it in full.

5 Architecture Details

5.1 Visual Parser: -

A two-layer neural network that takes as input the board state (3×3, an array consisting of 0s for empty squares, 1s for the network's pieces, and -1s for the opponent's pieces), passes it through a 100 unit hidden layer, and then to its output layer (also 100 units).

5.2 Q-approximator: -

A two-layer neural network that takes as input the "internal representation" from the visual parser, and outputs Q-values for the 9 possible plays on the board.

5.3 Question answerer: -

This system takes a question and the output of the visual parser and produces an answer. We

denote this as a whole by A. It consists of several sub-components: Encoder, Integrator, Decoder

5.3.1 Encoder: -

A simple RNN that encodes the question input to the system (100 hidden units) using word embeddings (20 dimensional, initialized at random, same as those used for the decoder in the output). Its final state is part of the input to the integrator. We denote this by E.

5.3.2 Integrator: -

A single layer neural network (100 units) that takes as input the final state of the encoder and the "internal representation" produced by the visual parser, and produces as output the initial state for the decoder. We denote this by us.

5.3.3 Decoder: -

A simple RNN (100 hidden units) that outputs the answer to the current question as a sequence of word vectors (dimension 20). Its initial state is the output of the integrator, its first input is "GO," and subsequently it receives its previous output as input. We denote this network by D.

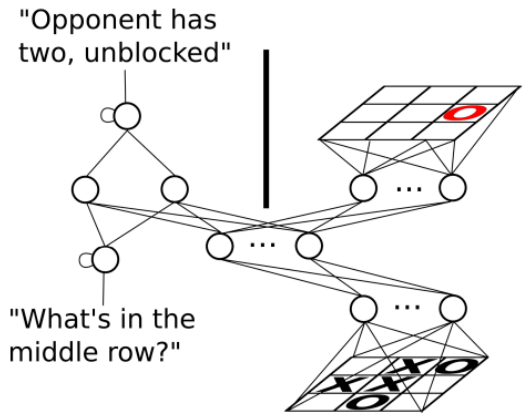


Figure 5. Architectural Design.

6 Data Generation

Because game playing involves interactions between two opponents, data generation is more complicated than on some other tasks. Fortunately, tic-tac-toe is simple, and both the plays and the descriptions could be generated on the fly[3].

6.1 Games: -

We trained the network by playing it against a hand-coded optimal opponent (opening plays were hand-coded, and thereafter the opponent used an efficient set of rules about checking for threats from the opponent, opportunities, etc. later in the game), using the Q-learning procedure stated above for the Q-net.

6.2 Descriptions: -

For this project, we implemented the simplest possible form of linguistic interaction: answering purely descriptive questions. For example:" What's in the third row? My piece, empty square, opponents piece." In total, there were eight possible questions, corresponding to the three rows, three columns, and two diagonals relevant to the task. We asked the network to answer one question (randomly selected from this set of eight questions) before each of its moves.

7 Data Processing

All sentences (input and output) were padded to length 10, input sentences were reversed. Weights and biases were randomly initialized from a normal distribution with a mean 0 and a standard deviation of 0.1. We masked the Q-value output to include only legal plays before taking the max (i.e. only considering playing on squares where there was not already a piece on the board) – this improved learning early on.

We used ϵ -greedy learning with $\epsilon = 0.1$.

We trained all networks by stochastic gradient descent with minibatch size 1 (i.e. a single example at a time).

We varied the learning rate for the Q-learning task, but all results in this paper are presented with a fixed learning rate **(η = 0.001)** for the description task.

8 Experiment

We assessed the system on whether it effectively accomplished flawless execution (no misfortunes) against the ideal rival inside 400

games, and on the off chance that it succeeded, what number of preparing ages it took to do as such (every age comprised of playing 20 games to finishing). We looked at the presentation of the system prepared with portrayals to two examination systems:

8.1 Basic: -

The Q-network without the description task.

8.2 Control: -

The Q-network with a different description task – counting the number of pieces on the board. In this task, there was only a single question ("How many are there?") and responses were of the form ("There are five"). This active control is to test for a possible effect of just having any other task. It's possible that any benefits from the description task would be due to overcoming the vanishing gradients and having learning signals reach the first layer more quickly, having a different auxiliary task that still relies on parsing the board state will fix this.

9 Results

η	basic	control	description
0.05	0%	0%	0%
0.01	50%	40%	**65%**
0.005	5%	10%	**79%**
0.001	0%	**15%**	**15%**

(a) Percent successful runs (runs where optimal performance was reached)

η	basic	control	description
0.05	N/A	N/A	N/A
0.01	14.1	13.8	**11.4**
0.005	19.0	12.5	**9.5***
0.001	N/A	16.0	**14.3**

(b) Mean epochs to success on runs where optimal performance was reached

Table 1: Results across varying learning rates (bold indicates best result in that row, overall best result is starred)

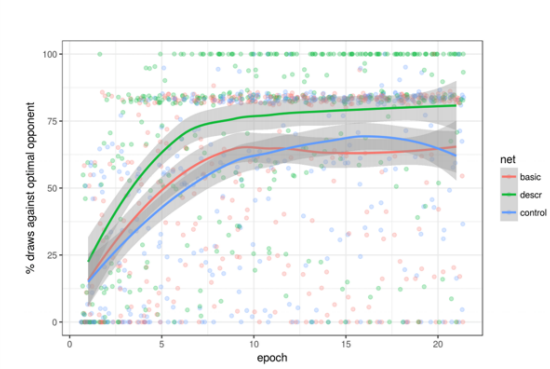


Figure 2: Percent draws vs. training epoch for each network with its optimal hyperparameter settings (average curves and individual data points from each run)

The natural language feedback appears to be beneficial, and comparing to the control task, it seems that its benefits are not attributable solely to allowing the gradients to propagate back more rapidly[3].

Why is the description task beneficial? One possibility is that it conveys information about the structure of the "world" that is not easy to obtain from a reinforcement learning signal.

Having the additional information from the natural language task could inform the system that rows, columns, and diagonals are useful configurations to pay attention to. Without this additional signal, it may be harder for the network to figure out which sets of squares it should and should not pay attention to.

10 Recent Advancement in this area

TextWorld is a sandbox learning environment for training and testing reinforcement learning (RL) agents on text-based games. it enables generating games from a game distribution parameterized by the map size, the number of objects, quest length and complexity, richness of text descriptions, and more. Then, one can sample the game from that distribution. TextWorld can also be used to play existing text-based games[8].

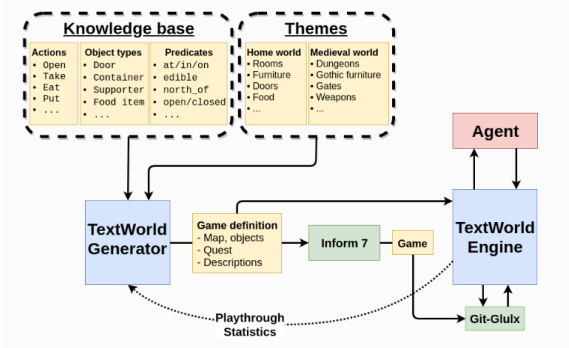


Figure .10. TEXTWORLD ARCHITECTURE.

TextWorld has two main components: a game generator and a game engine. The game generator converts high-level game specifications, such as the number of rooms, number of objects, game length, and winning conditions, into an executable game source code in the Inform 7 language[8]. The game engine is a simple inference machine that ensures that each step of the generated game is valid by using

simple algorithms such as one-step forward and backward chaining.

10.1 Game Observation: -

The game state object contains useful information3 such as:

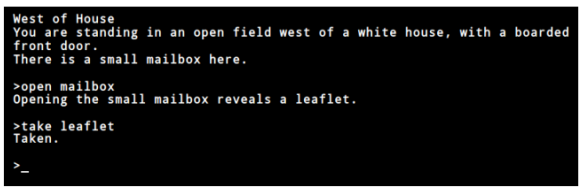
Feedback The interpreter's response to the previous command, i.e., any text displayed on the screen.

Description The description of the current room, i.e., the output of the look command. Inventory The player's inventory, i.e., the output of the inventory command. Location The name of the current room. Score The current score

Game state information is fairly limited when it comes to existing games, whereas games generated with TextWorld can provide much more information if desired. Additional information includes:

Objective Text describing what the player must do to win the game. Admissible Commands A list of commands that are guaranteed (i) to be understood by the interpreter and (ii) to affect the game state or to return information of the game state. Using this information in any way corresponds to playing a choice-based game rather than a parser-based one[8].

Intermediate Reward A numerical value representing how useful the last command was for solving the game. Winning Policy A list of commands that guarantees to win the game starting from the current game state[8]



Introduction to Zork

Text-based games are sequential decision-making problems that can be described naturally by Reinforcement Learning (RL) formalism. In this section, we define some of the terminology found in text-based games, formalize the text-based environment as an RL problem, discuss challenges faced by RL agents in such environments, and show how these challenges motivate the need for a framework like TextWorld. In the following, an "agent" is a model that takes text information as input and outputs text commands to progress through a game

11 Future Directions

Tic-tac-toe is an exceptionally straightforward game, and our depictions were extremely basic, so it was not clear we would perceive any improvement at all. It is energizing to stretch out this to a more extravagant game, (for example, chess or go), where the portrayals could incorporate additionally intriguing configurational data (for example strategic or vital data). This is another fascinating bearing for future research. One of the upsides of utilizing normal language is that it tends to be given by generally gullible end-clients. For instance, one could envision that a progressively unpredictable adaptation of this framework could be valuable if and when fortification learning frameworks are fused in independent vehicles – if a close to mishap happens, the human riders could give input on what the causes were that could enable the framework to adapt all the more quickly to diminish threat later on.

In a general sense, video games offer difficulties that prepackaged games like chess or Go simply don't. They conceal data from players, which means an AI can't see the entire playing field and figure the most ideal next move. There's additionally more data to process and an enormous number of potential moves. OpenAI says that at any one time its Dota 2 bots need to pick between 1,000 distinct activities while preparing 20,000 information focuses that speak to what's going on in the game.

To make their bots, the lab went to a technique for AI known as fortification learning. This is a misleadingly straightforward system that can create complex conduct. Artificial intelligence specialists are tossed into a virtual situation where they show themselves how to accomplish their objectives through experimentation. Developers set what are called

reward capacities (granting bots focuses for things like murdering an enemy), and then they leave the AI agents to play themselves over and over again.

For this new batch of Dota bots, the amount of self-play is staggering. Every day, the bots played 180 years of game time at an accelerated rate. They trained at this place over months. "It starts out random, wandering around the map. Then, after a couple of hours, it begins to pick up basic skills," says Brockman. He says that if it takes a human between 12,000 and 20,000 hours of play to learn to become a professional, that means OpenAI's agents "play 100 human lifetimes of experience every single day."

12 Conclusion

An auxiliary natural language learning task to a reinforcement learning system can accelerate learning on a game-playing task.

The aftereffects of the experiment propose that consolidating natural language guidance might be a productive for system for making complex assignments all the more simple tasks and provides effectively learnable. Content-based games give an effective system in which the agent can figure out how to act in text space. We can see that in TextWorld, a sandbox learning environment for the training and evaluation of RL age. TextWorld is certainly one of the applications where NLP proves to very much helpful as well as the games are more verbose compared to the simple tic-tac-toe.

13 References

[1] Timothy Atkinson, Hendrik Baier, Tara Copplestone, Sam Devlin, and Jerry Swan. The text-based adventure ai competition. arXiv preprint arXiv:1808.01262, 2018.

[2] Marco Baroni, Armand Joulin, Allan Jabri, Germàn Kruszewski, Angeliki Lazaridou, Klemen Simonic, and Tomas Mikolov. Commai: Evaluating the first steps towards a useful general ai. arXiv preprint arXiv:1701.08954, 2017.

[3] Andrew Lampinen Natural Language Learning Supports Reinforcement Learning

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Human-level control through deep reinforcement learning

[5] Ruo Yu Tao, Marc-Alexandre Côté, Xingdi Yuan, and Layla El Asri. "Towards Solving Text-based Games by Producing Adaptive Action Spaces." arXiv preprint arXiv:1812.00855 (2018)

[6] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., And Hassabis, D. Human-level control through deep reinforcement learning. Nature 518, 7540 (2015), 529–533.

[7] Reinforcement Learning: An Introduction. By Richard S. Sutton, Andrew G. Barto

[8] Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore TextWorld: A Learning Environment for Text-based Games