State Prediction in TextWorld with a Predicate-Logic Pointer Network Architecture

Corentin Sautier[1,2] , **Don Joven Agravante**[1] , **Michiaki Tatsubori**[1]
[1]IBM Research [2]MINES ParisTech - PSL Research University, Paris, France
corentin.sautier@ibm.com, don.joven.r.agravante@ibm.com, mich@jp.ibm.com

Abstract

Our work builds toward AI agents that can take advantage of data and deep learning while making use of the structure, extensibility and explainability of logical models with automated planning. We believe such agents have the potential to surpass many of the inherent limitations of current RL-based agents. Towards this long term goal, we aim to improve the capability of deep learning as a front-end to produce logical state representations required by planners. Specifically, we are interested in text-based games where we use Transformers to translate the unstructured textual description in the game into logical states. To improve the neural network architecture for this problem setting, we propose to augment the Transformer with a pointer network in a two-staged architecture. Our results show a clear improvement over a baseline Transformer network.

1 Introduction

In recent years, a lot of research focus in automated decision making has been given to Reinforcement Learning (RL), notably thanks to new frameworks simplifying the implementation [Brockman *et al.*, 2016]. RL algorithms usually require very limited knowledge about their environment, such as a list of admissible commands and a reward function. However, usual RL algorithms have several well-known limitations such as requiring long training on expensive hardware [OpenAI, 2018], difficulty in scaling up [Espeholt *et al.*, 2019], and they are often deemed unsafe for real-life applications [Dulac-Arnold *et al.*, 2019], especially outside the range of their training domain. On the other hand, classical planning is well-known to provide solutions that are consistent, more explainable, and with optimality guarantees for certain problems. However, most of the techniques in automated planning require domain knowledge that has been carefully handcrafted by an expert. This is usually written in a logical form such as the Planning Domain Definition Language (PDDL). This particular limitation has been overcome in RL, in part due to its integration of deep learning which can be used with raw observations such as images or plain text. Although deepRL methods are now common, there isn't much research on leveraging deep learning techniques for automated planning.

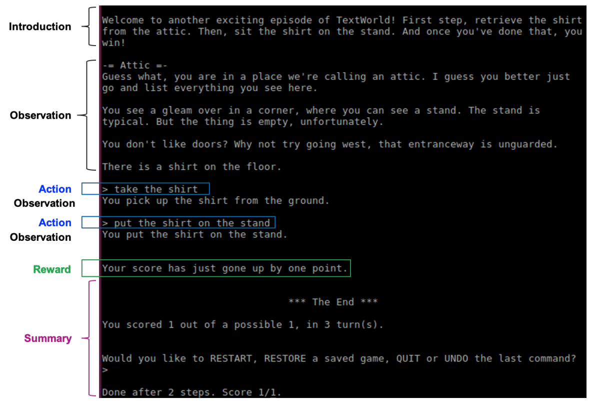
Our general research direction is that of expanding the usability of Automated Planners on environments providing raw observations by leveraging advances made in deep learning. Towards this, we concentrate on the task of transforming raw text into logic statements compatible with planners using supervised learning. To achieve this, we design semantic parsers to convert such natural language into a predicate-logic structure, such as the *states* of the PDDL. This is a similar problem setting to that of [Agravante and Tatsubori, 2020], where they showed how this semantic parser can be used together with *external knowledge* to complete the PDDL and "play" the TextWorld game.

Although the direction is promising, one of the main issues shown in [Agravante and Tatsubori, 2020] is that the Transformer neural network was not able to achieve sufficient generalization capabilities and overfit on the training set. Our focus is on this particular problem – that is to improve the semantic parser that will specifically be applied to obtaining the goal state. In this paper we propose a semantic parser with a two-staged architecture based on a Transformer Network and a novel multi-head Pointer Layer, to predict a goal state from its natural language description.

2 Problem Setting and Task

We are using the TextWorld [Côté *et al.*, 2018] game which is a gym environment [Brockman *et al.*, 2016], specifically designed to combine RL and Natural Language Processing (NLP). TextWorld is an environment in which every interaction is done through text. The game starts with an introduction of a task, followed by a textual observation of the surroundings of the player. Each action, also inputted by text, prompts a new observation, which usually returns the success or failure of the action, and in some situations a new observation. Figure 1 shows an example of a game. Our main interest in TextWorld is the *introduction* text, which describes the quest/objective of the game. We would like to parse this text into a predicate logic form that is suitable for the PDDL

The Planning Domain Definition Language (PDDL) is the de-facto standard for automated planners. It is a complete formal description of a planning problem mainly consisting of the initial state, goal state and action templates. An agent that is able to transcribe problems into the PDDL would be



```
Welcome to another exciting episode of TextWorld! First step, retrieve the shirt
from the attic. Then, sit the shirt on the stand. And once you've done that, you
win!

-= Attic =-
Guess what, you are in a place we're calling an attic. I guess you better just
go and list everything you see here.

You see a gleam over in a corner, where you can see a stand. The stand is
typical. But the thing is empty, unfortunately.

You don't like doors? Why not try going west, that entranceway is unguarded.

There is a shirt on the floor.

> take the shirt
You pick up the shirt from the ground.

> put the shirt on the stand
You put the shirt on the stand.


Your score has just gone up by one point.


                              *** The End ***

You scored 1 out of a possible 1, in 3 turn(s).


Would you like to RESTART, RESTORE a saved game, QUIT or UNDO the last command?
>

Done after 2 steps. Score 1/1.
```

Figure 1: Example of a TextWorld game

an interesting but very daunting research challenge. Here, we concentrate on a small but interesting piece of this – learning to transcribe the goal state. The challenge in TextWorld is to parse the *introduction* text into the PDDL goal state.

The PDDL is based on first-order/predicate logic. A PDDL state is usually defined as a conjunction of every true proposition at a given time. Each proposition is composed of a predicate function and its arguments – the objects. The arity of a predicate refers to the number of objects required as the input. For example, the proposition: **on(stand, shirt)**, is composed of the predicate *on*, with arity of 2. The two objects are *stand* and *shirt*. In our setup, there are 10 possible predicates, with arities of 1 or 2.

Our main task is similar to that of [Agravante and Tatsubori, 2020] where they predict the PDDL goal state given the *introduction* text that describes the quest. The main challenge here is to effectively form all the propositions by predicting the predicates and the associated objects for each of these predicates.

3 Deep Neural Semantic Parsing Model

Before detailing our model in the subsections, we first give an intuition and overview as follows. The state can be expressed as a set of every true proposition, and each proposition is composed of a predicate and a list of objects, subsequently defining a two-stages generation process as shown in Fig. 2. Our work assume the knowledge of all predicates and their arities.

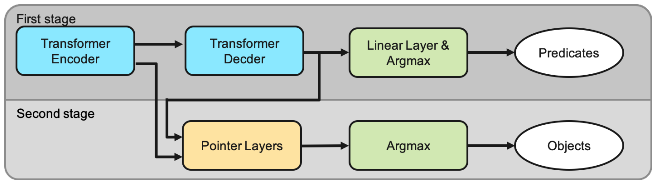


Figure 2: Our two-staged deep neural semantic parsing model

The first stage is the predicate decoding, which our model does in a structure of a Transformer Sequence-to-Sequence. The second stage is, for each of those predicates generated by the first stage, to supply them with the correct number of objects. To do that, instead of relying on a dictionary of every word that could fit in, we will use Pointer layers to predict the index of the correct words from the input sentence, to put in each of the expected objects of the predicate.

For example, if the introduction contains *Make sure to unlock the chest*, we will want to generate the proposition *unlocked(chest)*. The first stage must learn to predict the proposition *unlocked* and the the second stage must return the index of *chest* in the sentence. In our example it is the 6th word.

In practice, since some objects are multi-words (e.g., *wooden chest*), each pointer also predicts the number of words it refers to. For predicate of arity greater than one, the need to supply multiple objects is solved by having multiple Pointer Layers. For example, the predicate *in*, requires two objects, and will use two different pointer layers, for the object contained, and the object containing.

The main difficulty is that those Pointer layers must use in-context information about the decoded predicate, to select relevant objects. If for instance the introduction contains *Make sure the chest is unlocked and contains a shirt*, and we are in the process of decoding *unlocked(chest)*, the network must understand that in this context, the relevant word is *chest*, even though *shirt* might also be relevant in another context.

Transformer Network

A Transformer Network [Vaswani *et al.*, 2017], as opposed to a Recurrent Neural Network (RNN) does not need to sequentially pass the words (or any other sequence) through gates, giving each word the same potential no matter its position. This allows for bigger sequences, as well as an easier internal representation of the words, and the resulting Neural Networks significantly outperform RNNs in many complex NLP tasks [Devlin *et al.*, 2018].

A Transformer Network, is an encoder-decoder model. An input text is encoded after embedding, using stacked multi-head self-attention layers, and decoded with a similar structure of stacked multihead attention using the output sequence as an auto-regressive input, and the encoded vector as an internal attention mechanism [Vaswani *et al.*, 2017].

Before explaining our Pointer Network, we recall some of the important underlying mechanisms of the Transformer's attention which we use later on.
Let's define $X = x_1, x_2, ..., x_n$ an input sequence of vector (e.g. the output of the encoder) and $Y = y_1, y_2, ..., y_m$ a partial output sequence (e.g. the embedded sequence of already decoded words). Those all have the same hidden dimension d. We define model parameters matrices $W^Q_{d\times d}$, $W^K_{d\times d}$ and $W^V_{d\times d}$. The sequences $Q = q_1, q_2, ..., q_m$, $K = k_1, k_2, ..., k_n$ and $V = v_1, v_2, ..., v_n$ are obtained by calculating $Q = W^Q \cdot Y$, $K = W^K \cdot X$ and $V = W^V \cdot X$. The vectors in Q, K, V are named queries, keys and values, respectively.[1] The names come from the role we want to give them:

- A query represents what we look for in the sequence

[1] A bias is usually also added in those equations, but we left it out for clarity

(i.e., it will encode what sort of information we need to find next).

- A key encodes what information is actually contained by this sequence element (i.e., what a word from the input sentence means, as well as its syntactic role).
- A value represents what information should be remembered of this sequence element if it is deemed useful.

The attention can be interpreted as the sum of the value of each sequence element pondered by its relative importance to a given query, and is calculated as follows:

$$\forall\, i \in [1,m],\ A_i = \sum_{j=1}^{n} softmax(\frac{q_i^T \cdot K}{\sqrt{d}})_j \times v_j, \qquad (1)$$

where A_i is the attention to the i-th element of the sequence.

Using different matrix parameters, multiple of those attentions can be stacked to obtain a *multihead attention* layer. The intuition is that it allows heads to attend to different sub-spaces, allowing for a more diverse and robust model [Vaswani *et al.*, 2017]. And in the specific case where X = Y, this layer becomes a self-attention layer, whose result has exactly the same dimensionality as its input, allowing to stack as many of those layers as needed. A standard Transformer network is composed of multiple self-attention layers, as well as one multihead attention linking the input to the shifted output [Vaswani *et al.*, 2017].

Multihead Pointer Layer

A Pointer Layer [Vinyals *et al.*, 2015], in the field of NLP, is a Softmax applied over a vector of score of length of the input sentence, implicitly giving *probabilities* that a word is interesting for a given purpose. The score is hinting the importance of a word relatively to the other words of the sentence.

Pointer Networks have been successfully used for NLP with RNN [See *et al.*, 2017] but since the creation and general adoption of Transformer Networks after 2017 [Vaswani *et al.*, 2017], they have rarely been used since the technology developped for an RNN was not directly applicable after a multihead attention. An important reason why it was difficult is that the output of a Transformer decoder does not contain information about single words, and forget even the size of the input text, hence referring to it simply would require tricks such as fixed-size buffers with padding, with no real consistency guarantee. We could also directly apply a Pointer Layer after the encoder, but such a Pointer would not contain any conditional information about the current decoding step, hence pointing to interesting words from the input sentence regardless of the context, which is not our objective here.
We believe a correct Pointer Layer formulation must satisfy those criteria:

- The output must have the same length as the input sentence
- It must use information about each word to compute the score, and not some pooling over the whole sentence or similar tricks
- The information must be contextualized to the decoding process

Our model satisfies all those criteria. Using the same concepts and notations as previously, we believe that what we want is the relative importance of each word to a given query. Mathematically, this gives:

$$\forall\, i \in [1,m],\ p_i = softmax(\frac{q_i^T \cdot K}{\sqrt{d}}), \qquad (2)$$

where p_i is the pointing vector over the input sentence, computed at the i-th decoding step.

Note that this formula cannot be stacked, however, it can be rewritten in a multihead formulation, provided we compute a pooling of the scores of each head before the Softmax operation. If we define $nhead$ to be the number of heads, and add another index for the heads, it becomes:

$$\forall\, i \in [1,m],\ p_i = softmax(m_{j=1}^{nhead}(\frac{q_{j,i}^T \cdot K_j}{\sqrt{d}})), \qquad (3)$$

where $m_{j=1}^{nhead}$is a pooling function of elements in $[i, nhead]$. Contrasting Eq.(1) and Eq.(3) shows how to adapt Transformer layers into Pointer layers.

4 Related Work

Our problem setting is adapted from [Agravante and Tatsubori, 2020] but related works show that it is also applicable to other settings. For example, some recent work has focused on adapting a pointer layer to a transformer network [Esmaeilzadeh *et al.*, 2019], or [Aksenov *et al.*, 2019]. However, their implementations differ from ours significantly, both technically and in the use case. They rely on the architecture of the Pointer Generator Network [See *et al.*, 2017], with an ad hoc linear layer to adapt a Transformer Memory to the required shape of the Pointer Layer, and the use case is text summarization, for which the pointer mechanisms had already proved to be of great use [See *et al.*, 2017]. The closest work that can be linked to ours would be [He *et al.*, 2019], specifically using a pointer layer to augment a semantic parser working with logical form. However, their architecture uses the pointing mechanism as a copy and paste, in a sequence decoding, when we use it in a nested two-staged decoding process, making use of the knowledge of the predicates.

Using nested decoding in a semantic parser was done very successfully by [Dong and Lapata, 2016] in their Seq2Tree, applied to code and query generation, using however LSTM, as it was the main NLP technology in research at the time. A very similar approach using Transformers was proposed by [Sun *et al.*, 2019] in their TreeGen network, but the difference in intermediary decoding step compared to LSTM made them chose a different and more complicated architecture than the original Seq2Tree.

5 Experiments and Results

The TextWorld games are custom generated with the *house* theme, quest length of 5, world size of 5 and containing 10 objects. We made sure to use different seeds for each game.

We gathered a dataset of 67992 introduction text from randomly generated TextWorld games and obtained goal state for them by reading into each game file. Among those games,

Introduction text	Predicted PDDL goal state
It's time to explore the amazing world of TextWorld! Here is your task for today. First off, if it's not too much trouble, I need you to move west. After that, take a trip west. Then, take the latchkey from the kitchenette. After that, venture east. Okay, and then, place the latchkey into the locker in the studio. Alright, thanks!	at (locker, studio); at-agent (studio); in (latchkey, locker); not (locked (locker)); opened (locker)
You are now playing a fast paced round of TextWorld! Here is how to play! Your first objective is to retrieve the Henderson's passkey from the shelf in the garage. If you have picked up the Henderson's passkey, insert the Henderson's passkey into the Henderson's gateway's lock to unlock it. Then, open the Henderson's gateway. And then, venture west. Then, make it so that the Henderson's gateway within the shower is shut. That's it!	at-agent (shower); not (locked (henderson's gateway's)); not (opened (henderson's gateway's))
I hope you're ready to go into rooms and interact with objects, because you've just entered TextWorld! Here is how to play! First thing I need you to do is to unlock the formless gateway with the formless key. And then, ensure that the formless gateway in the steam room is open. After that, make an attempt to venture west. After that, recover the hat from the attic. After you have picked up the hat, you can deposit the hat into the spherical locker. Got that? Good!	at (spherical locker, attic); at-agent (attic); in (hat, locker); not (locked (locker)); opened (locker)

Table 1: Example of generated goal states from our model – the first two are perfect predictions, and the third has a few mistakes highlighted.

	Accuracy	BLEU score
Seq2Seq baseline (seed 1)	0.872	0.869
Seq2Seq baseline (seed 2)	0.709	0.751
Our Model (seed 1)	0.903	0.900
Our Model (seed 2)	**0.939**	**0.918**
Our Model (seed 3)	0.936	0.917
Our Model (seed 4)	0.927	0.913

Table 2: Results

13032 goal states contained objects not included in the introduction, that we discarded, as sometimes the game include hidden or implied proposition in the goal state, that are not actually necessary, or predictable, bringing the total dataset to 54960 elements, that we split in 47100 training examples and 7860 testing examples (roughly one sixth).

We trained both a Sequence-to-Sequence (Seq2Seq) baseline, and our model for 1000 epochs, performing an evaluation on the complete testing set every 50 epochs, and only report the best epoch results.

We used two metrics: the first is the accuracy, describing how many goal states were perfectly predicted among the testing set, and the second is the BLEU score. The BLEU score is used to give a more local information about correctness at the *words level*. On the other hand, the accuracy being the proportion of the problems entirely solved, it only counts the perfect predictions, and is thus less biased.

In Table 1, the first two results show the capacity of the network to understand that intermediary states will then be canceled and do not appear in the goal – the *latchkey* is placed in the *locker* so we no longer *carry* it, and the *gateway* we *opened* must be *closed* again. We picked the third example to show a mistake the network can make: it doesn't find that *spherical* is an adjective of *locker*. We found this task to be difficult, because *spherical* is not openable like the *locker*, and it can be an adjective for many different types of objects. As a result, the prediction correctly inferred that a *locker* must be opened, but those results might not be specific enough to solve the problem.

Table 2 shows that our model performs better than a Seq2Seq baseline in both the accuracy and BLEU metrics. This performance is also consistent in training runs with different random seeds. In investigating the deficiency of the Seq2Seq baseline, it is interesting that we haven't found examples where a predicate had an inappropriate number of objects. This suggests that this is probably an easy rule to *learn*. We can infer from this observation that the benefits of our model doesn't come from the nesting alone, but also from the pointing mechanism.

6 Conclusion

We have built a semantic parser to predict logic forms usable by an automatic planner, using a nested architecture stacking a Transformer Network and custom Pointer Layers. We have found that using a pointing mechanism to augment a Seq2Seq network, by allowing it to refer directly to input words can still be an effective solution, even with Transformer Networks, as it can significantly outperform a network without this mechanism.

The main reason for the difference between the two models is the difficulty to refer to objects, as in the case of the Seq2Seq, it consists of picking the correct word in a huge dictionary (562 words here) with even a few of those words never seen during the training, whereas the pointing mechanism has neither this scale issue, nor the discrepancy between training and testing.

Subsequently, we presented an architecture allowing this pointing mechanism to benefit from the decoding memory, so that the pointing mechanism gives relevant words in the current decoding context. Finally we were able to use an easy to implement, two-staged nested architecture with a Transformer network, inspired by successful work such as Seq2Tree [Dong and Lapata, 2016] or TreeGen [Sun *et al.*, 2019] and paving the way for future work in semantic parsing with nested syntax, such as code generation.

References

[Agravante and Tatsubori, 2020] Don Joven Agravante and Michiaki Tatsubori. Learning to set planning goals for textworld by deep neural semantic parsing. *The 34th Annual Conference of the Japanese Society for Artificial Intelligence*, JSAI2020:3F5ES205–3F5ES205, 2020.

[Aksenov *et al.*, 2019] Dmitrii Aksenov, Julian Moreno-Schneider, Peter Bourgonje, Robert Schwarzenberg, Leonhard Hennig, and Georg Rehm. Abstractive text summarization based on language model conditioning and locality modeling. *ArXiv*, abs/1904.00788, 2019.

[Brockman *et al.*, 2016] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016.

[Côté *et al.*, 2018] Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. TextWorld: A Learning Environment for Text-based Games. *CoRR*, abs/1806.11532, 2018.

[Devlin *et al.*, 2018] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[Dong and Lapata, 2016] Li Dong and Mirella Lapata. Language to Logical Form with Neural Attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33–43, Berlin, Germany, August 2016.

[Dulac-Arnold *et al.*, 2019] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of Real-World Reinforcement Learning. *arXiv*, 2019.

[Esmaeilzadeh *et al.*, 2019] Soheil Esmaeilzadeh, Gao Xian Peh, and Angela Xu. Neural abstractive text summarization and fake news detection. *ArXiv*, abs/1904.00788, 2019.

[Espeholt *et al.*, 2019] Lasse Espeholt, Raphaël Marinier, Piotr Stanczyk, Ke Wang, and Marcin Michalski. SEED RL: Scalable and Efficient Deep-RL with Accelerated Central Inference. abs/1910.06591, 2019.

[He *et al.*, 2019] Xuanli He, Quan Tran, and Gholamreza Haffari. A pointer network architecture for context-dependent semantic parsing. In *Proceedings of the The 17th Annual Workshop of the Australasian Language Technology Association*, pages 94–99, Sydney, Australia, 4–6 December 2019. Australasian Language Technology Association.

[OpenAI, 2018] OpenAI. OpenAI Five, 2018.

[See *et al.*, 2017] Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[Sun *et al.*, 2019] Zeyu Sun, Qihao Zhu, Yingfei Xiong, Yican Sun, Lili Mou, and Lu Zhang. TreeGen: A Tree-Based Transformer Architecture for Code Generation. *CoRR*, abs/1911.09983, 2019.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008, 2017.

[Vinyals *et al.*, 2015] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems*, volume abs/1506.03134, 2015.