Semantic Parsing for Planning Goals as Constrained Combinatorial Contextual Bandits

Anonymous ACL submission

Abstract

We are working towards AI planning systems with natural language interfaces. In this paper, we tackle the semantic parsing problem of learning to set the logical goals of the planning system based on a natural language description of the task. The current state of the art in semantic parsing is to use supervised learning with deep neural networks but this needs a lot of labelled data made by domain experts. To reduce this need, we additionally use a reward signal that comes from completing the AI planning task. We formalize this as a constrained combinatorial contextual bandit problem. The context is created by using a deep neural network for feature extraction and the constrained combinatorial nature of the task can be used to increase the efficiency of learning. We show this result theoretically with our lower regret bound and then experimentally in our extension of the TextWorld problem.

1 Introduction

AI planning systems are designed to solve problems that can be formulated as finding a series of actions from an initial state to a goal state. This is a generic problem such that these systems are deployed in many different applications, e.g., game playing agents (Perez-Liebana et al., 2019), simple customer support agents (Nothdurft et al., 2015), enterprise applications (Sohrabi, 2019), robotics (Thomason et al., 2015), etc. In this paper, we use the example of an autonomous agent/robot. Although AI planning systems are powerful, they need to be programmed in a logic-base language such as the Planning Domain Definition Language (PDDL) (Fox and Long, 2003). Creating this logic specification is non-trivial and it is a well-known bottleneck that prevents the further application of AI planning systems. Towards reducing this burden, our task in this paper is enabling the use of natural language for specifying the PDDL goals as shown in Fig. 1.

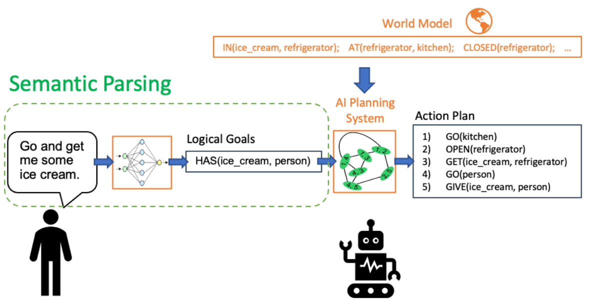

World Model
IN(ice_cream, refrigerator); AT(refrigerator, kitchen); CLOSED(refrigerator); ...
Semantic Parsing
Go and get me some ice cream.
Logical Goals
HAS(ice_cream, person)
AI Planning System
1, 2
1, 3
1, 4
4, 3
2, 5
4, 5
Action Plan
1) GO(kitchen)
2) OPEN(refrigerator)
3) GET(ice_cream, refrigerator)
4) GO(person)
5) GIVE(ice_cream, person)

Figure 1: The semantic parsing task consists of translating a user request into logical goals. The goals interface into in an AI planning system that uses a world model to create an action plan.

Our task is a specific example of semantic parsing - the general task of processing a given text into its meaning, which is represented in a *logical/symbolic form*. Modern semantic parsing methods are based on deep learning (Dong and Lapata, 2016) which has produced great results in recent years. With this success, several semantic parsing applications have been introduced into the literature. A useful taxonomy arises from the different output logic forms. This output is usually a type of programming language, for example, Prolog (Dong and Lapata, 2016); SQL (Zhong et al., 2017; Dong and Lapata, 2018); Python (Yin and Neubig, 2017; Dong and Lapata, 2018); or Lisp (Liang et al., 2017). Of these examples, our task is closest to that of (Zhong et al., 2017) where a natural language *question* is translated into an *SQL querry* which then uses an existing *database* to produce the *answer*. A useful analog of each part can be seen in Fig. 1. Here, a natural language *quest* is translated into a *PDDL goal* which then uses an existing *world model and AI planning system* to produce the *optimal sequence of actions*. Similarly, creating the semantic parsing labels (whether SQL or PDDL goal) requires domain expertise which is relatively expensive to obtain. There is a cheaper but less informative label that can be obtained by checking the final output – checking the final answer in the case of SQL or running the actions and obtaining a goal completion or *reward* signal for our task. We seek to use this reward signal as a way to reduce the amount of logic labels required for training a useful system. Although there is a conceptual similarity between these tasks, there are two important differences that we use to our advantage - the temporal structure of the output and the rich logical structure of the world model.

To solve our task, we seek an approach that can leverage the labelled logic form (PDDL goal) when available but also make use of the *reward* label of the final system output which are cheaper to get in practice. A good starting point was used in (Zhong et al., 2017), where they improved a supervised model with a kind of *one-step* reinforcement learning (RL) by using policy gradient updates. In the RL literature, this *one-step* setting is also called the *multi-armed bandit* problem setting (Sutton and Barto, 2018). The basic bandit problem can be augmented in many ways to describe different tasks. Here, we add a *context* which corresponds to our *quest* and then we use the logical structure of our output *PDDL goals* together with the *world model* to impose a structure which is *combinatorial* and *constrained*.

The contribution of this paper is twofold: First, we propose the formalization of our semantic parsing task as a constrained combinatorial contextual bandit. Theoretically, we show that we can reduce the regret which means learning is guaranteed to be more efficient. Practically, we introduce the CCLin-UCB algorithm for this task. Our second contribution, is a scalable dataset for our task. Our task is common in the robotics literature in many different forms but there has been no large scale dataset suitable for large deep learning models. Here, we extend the existing TextWorld benchmark (Côté et al., 2018) by providing PDDL interfaces and we outline how this can be used similarly to Fig. 1.

In the following sections, we start with a background on semantic parsing and how deep learning is used. Next, we introduce the constrained combinatorial contextual bandits for our semantic parsing task. Then we introduce our extension of TextWorld with PDDL planning and finally we demonstrated the results of our method on this.

2 Semantic Parsing

Semantic parsing converts a given input of natural language text, represented as $\mathbf{x}$, into an output logic form, represented as $\mathbf{y}$. In our task, $\mathbf{x}$ comes from the quest description and $\mathbf{y}$ is from the PDDL goal statement. Both of the vectors, $\mathbf{x}$ and $\mathbf{y}$, have a sequential nature.

To get $\mathbf{x}$ and $\mathbf{y}$, a common preprocessing step is *tokenization*. This involves dividing the input (or output) text into tokens, each one is typically a separate word or a punctuation. A *dictionary* then transforms these tokens into corresponding integer IDs such that $\mathbf{x}$ and $\mathbf{y}$ are sequences of integers representing the original input and output. This whole preprocessing step is reversible meaning that given an arbitrary sequence of integers, one can get a corresponding input or output text.

Once we have $\mathbf{x}$ and $\mathbf{y}$, the main problem of semantic parsing consists of finding a function, g, such that

$$\mathbf{y} = g_\theta(\mathbf{x}), \tag{1}$$

where θ denotes the parameters of g. Learning the function then amounts to finding θ and this can be done in two different ways: supervised and bandits. These are detailed in the following subsections.

2.1 Supervised Learning of Semantic Parsing

Supervised learning has shown to be very effective for the semantic parsing problem but it requires a sufficiently large, diverse and labelled dataset, $\{\mathbf{x}, \mathbf{y}^{gt}\}$, where a ground truth label, $\mathbf{y}^{gt}$, is provided for each $\mathbf{x}$. The problem of finding g can then be set up as:

$$\theta^* = \arg\min_\theta \mathcal{L}(\mathbf{y}^{gt}, g_\theta(\mathbf{x})), \tag{2}$$

where the problem is to find the model's optimal parameters, θ^*, by minimizing some loss function, $\mathcal{L}$, that measures the mismatch between predicted $\mathbf{y}$ and the ground truth label $\mathbf{y}^{gt}$.

The key to successfully applying deep learning on the problem of Eq.(2) is to find a suitable deep neural network architecture for g_θ. For semantic parsing, sequential models (e.g. LSTM) are the most suitable (Dong and Lapata, 2016). This means that the neural network is modeling the conditional probability, $p(y^i|y^{i-1},\ldots,y^0,\mathbf{x})$, where i denotes the index of the value within the sequence. In the current state of the art, the best sequential models are variations on the *Transformer* (Vaswani et al., 2017). Because of this, we use it as our base model.

For the rest of this paper, it is useful to recall that the Transformer has an encoder-decoder structure (Vaswani et al., 2017). This means that the input sequence is first *embedded* into a vector with continuous values and then *encoded* into a *context vector*. The output tokens are then *decoded* in an auto-regressive manner based on the context vector and the previously generated output tokens. For more details, such as the loss function and optimization algorithm, we refer the reader to (Vaswani et al., 2017).

2.2 Bandits for Semantic Parsing

In our particular semantic parsing setting, we can obtain additional information by using the output logic form in some process, h, to obtain a reward, r. That is:

$$r = h(\mathbf{y}). \tag{3}$$

In our task depicted by Fig. 1, h consists of running the AI planning system and getting the action plan, the robot executing the plan and finally the human gives a scalar approval rating which is the reward, r. Additional rewards can also be obtained by other information during the whole process of h. Another example of this is using SQL (Zhong et al., 2017), where h consists of querying the database to get an answer and then a scalar reward r is returned based on the correctness of the answer and errors in the querying process.

In this reinforcement learning (RL) paradigm, we want to use rewards, r, as information to improve the function that is generating $\mathbf{y}$ instead of labels, $\mathbf{y}^{gt}$, which were needed for the supervised case. Using RL terminology, we are treating $\mathbf{y}$ as an *action* and g as a policy function. The problem is then to maximize the cumulative rewards R such that:

$$\theta^* = \arg\max_{\theta} \mathbb{E}\left[R \mid g_\theta\right]. \tag{4}$$

This is the bandit problem (Sutton and Barto, 2018) where we need to explore the space of possible g to find one that gives us the most rewards. Intuitively, the solution is to do a *trial-and-error* process, where we try different policies and update it based on the rewards produced. Furthermore, the problem in Eq.(4) is already a contextual bandit problem because the policy, g, takes the input $\mathbf{x}$ which is an additional context for the task.

The RL literature has several ways to solve the optimization problem of Eq.(4). Perhaps the most common one is to use a policy gradient such that:

$$\nabla_\theta \mathbb{E}\left[R \mid g_\theta\right] \approx \frac{1}{N}\sum_{n=1}^{N} \nabla_\theta log(g_\theta) R, \tag{5}$$

where N is the number of samples to approximate the gradient. This method was also used in (Zhong et al., 2017) where $N = 1$.

Although it is appealing to train the agent with only rewards, note that it comes at the computational cost in doing the exploration. This results in the so called *curse of dimensionality* (Sutton and Barto, 2018). Specifically, the action space $\mathbf{y}$ is usually too large to be explored without a good starting point. The next subsection discusses a practical method for handling this.

2.3 On Combining Supervised Learning and Bandit Algorithms

The previous subsections showed two different learning methods with different pros and cons - supervised learning requires labels, $\mathbf{y}^{gt}$ and is very efficient while bandits only require rewards, r, but then an expensive exploration process is needed. Each of the learning methods can be used separately but it is more practical to use them together.

In fact, the method detailed in (Zhong et al., 2017) is one such instance wherein the available labels for supervised learning are first used. When this model converges, further training is carried out in the RL setting. Another possible improvement comes with the popularity and availability of pre-trained language models such as BERT (Devlin et al., 2019). These can be used in place of supervised learning as the starting point for learning with bandits or it may even be used as the starting point for supervised learning and then RL training is used as the final step.

3 Constrained Combinatorial Contextual Bandits

We model our learning problem as a combinatorial online optimization called "combinatorial semi-bandit" (Wang and Chen, 2018; Wen et al., 2015), which is defined by a triple $(E, \mathcal{A}, P)$, where (1) $E = \{1, \ldots, L\}$ is a set of L arms, called the *ground set*, (2) $\mathcal{A} \subseteq \{A \subseteq E : |A| \leq K\}$ is a family of subsets of E with up to K arms, where $K \leq L$, and P is a probability distribution over the weights $\mathbf{w} \in \mathbb{R}^L$ of the arms in the ground set E. (3) $\mathbf{w} : E \rightarrow \mathbb{R}$ is a *weight function* that assigns each arm e in the ground set E a real number. The total weight of all arms in a set $A \subseteq E$ is defined as: $f(A, \mathbf{w}) = \sum_{e\in A} \mathbf{w}(e)$, which is a linear functional of $\mathbf{w}$. For our PDDL goal semantic parsing task, A is the set of facts in the logical goal state and E is the set of all possible facts to be considered in forming this goal. The final PDDL goal state is simply the conjunction over A.

At each iteration t, the agent chooses $A^t \in \mathcal{A}$, gains $f(A^t, \mathbf{w}_t)$, and observes the weights of all the chosen arms in iteration t, $\{(e, \mathbf{w}_t(e)) : e \in A^t\}$. The agent goal is to maximize the expected cumulative return in n-iterations $\mathbb{E}\left[\sum_{t=1}^{n} f(A^t, \mathbf{w}_t)\right]$. We assume that the agent knows a generalization matrix $\Phi \in \mathbb{R}^{L\times d}$. With ϕ_e the transpose of the e-th row of Φ, and refer to it as the *context vector*. In our case, this should be a vector representation of the *quest*. We can get this by inputting the given text into a pre-trained deep neural network as a *feature extractor*. That is, the context vector is some intermediate result in a middle layer of the neural network. Moreover, instead of generic pre-trained models, it is also possible to use the result of the supervised learning method described previously. Continuing, we define $\theta^* = \arg\min_{\theta} \|\bar{\mathbf{w}} - \Phi\theta\|_2$, with the mean weight is denoted by $\bar{\mathbf{w}} = \mathbb{E}[\mathbf{w}]$ and $\bar{\mathbf{w}} = \Phi\theta^*$. Note that in our setting there are some arms that could be explored just once since the environment is going to inform the agent $A_i^t \in \{A^t \cap A'\}$ the not allowed arms, with A' the set of all not allowed arms. Intuitively, disallowed arms correspond to *absurd facts*. This error is usually caught in *type-checking* by using the given PDDL world model. For example, given the predicate *eaten* which takes an object of type *food* as an input, then the fact *(eaten kitchen)* is absurd since *kitchen* is not a type of food. To continue, we define $R_t = f(A^{opt}, \mathbf{w}_t) - f(A^t, \mathbf{w}_t)$ as the regret in episode t. With $A^{\text{opt}} \in \arg\max_{A\in\mathcal{A}} f(A, \mathbf{w}) = \arg\max_{A\in\mathcal{A}} \sum_{e\in A} \mathbf{w}(e)$ and the *expected cumulative regret* of the learning algorithm in n episodes as $R(n) = \sum_{t=1}^{n} \mathbb{E}\left[R_t|\bar{\mathbf{w}}\right]$.

The pseudocode of CCLinUCB is given in Algorithm 1 where λ is a regularization parameter, σ controls the decrease rate of the covariance matrix, and c controls the exploration. In each episode t, Algorithm 1 consists of three steps. First, for each $e \in E$, it computes an upper confidence bound (UCB) $\hat{\mathbf{w}}_t(e)$. Second, it computes A^t based on $\hat{\mathbf{w}}_t$. Finally, it updates the mean vector $\bar{\theta}_{t+1}$ and a covariance matrix Σ_{t+1} based on Kalman filtering. Note that our proposed algorithm is a modification of CombLINUCB proposed in (Wen et al., 2015), where we remove in an iterative manner the not allowed arms. We show in the following, how our modification is positively impacting the regret. Assuming that $\|\phi_e\|_2 \le 1, \forall e \in E$ $\mathbf{w}_t(e) \in [0, 1]$ and the stochastic arm weights $\{\mathbf{w}(e)\}_{e\in E}$ are statistically independent under P, we have the following upper bound on $R(n)$ when CCLINUCB is applied:

Algorithm 1 CCLinearUCB

```
Input: Combinatorial structure $(E, \mathcal{A})$, generalization matrix $\Phi \in \mathbb{R}^{L\times d}$, algorithm parameters $\lambda, \sigma, c > 0$
Initialize $\Sigma_1 \leftarrow \lambda^2 I \in \mathbb{R}^{d\times d}$ and $\bar{\theta}_1 = 0 \in \mathbb{R}^d$
for all $t = 1, 2, \ldots, n$ do
    $A^t = \arg\max_{A\in\mathcal{A}} \sum_{e\in A} \langle \phi_e, \bar{\theta}_t\rangle + c\sqrt{\phi_e^T \Sigma_t \phi_e} \quad \forall e \in E$
    Choose set $A^t$, observe both $R_t$ and $A_i^t \in A'$
    remove the not allowed arms $A = A \backslash A_i^t$
    Update $\bar{\theta}_{t+1}$ and $\Sigma_{t+1}$,
    for all $e \in A^t$ do
        $\Sigma_{t+1} = \Sigma_t - \frac{\Sigma_t \phi_e \phi_e^T \Sigma_t}{\phi_e^T \Sigma_t \phi_e + \sigma_t^2}$
        $\bar{\theta}_{t+1} = \left[I - \frac{\Sigma_t \phi_e \phi_e^T}{\phi_e^T \Sigma_t \phi_e + \sigma^2}\right] \bar{\theta} + \left[\frac{\Sigma_t \phi_e}{\phi_e^T \Sigma_t \phi_e + \sigma^2}\right] \langle \phi_e, \bar{\theta}_t\rangle$
    end for
end for
```

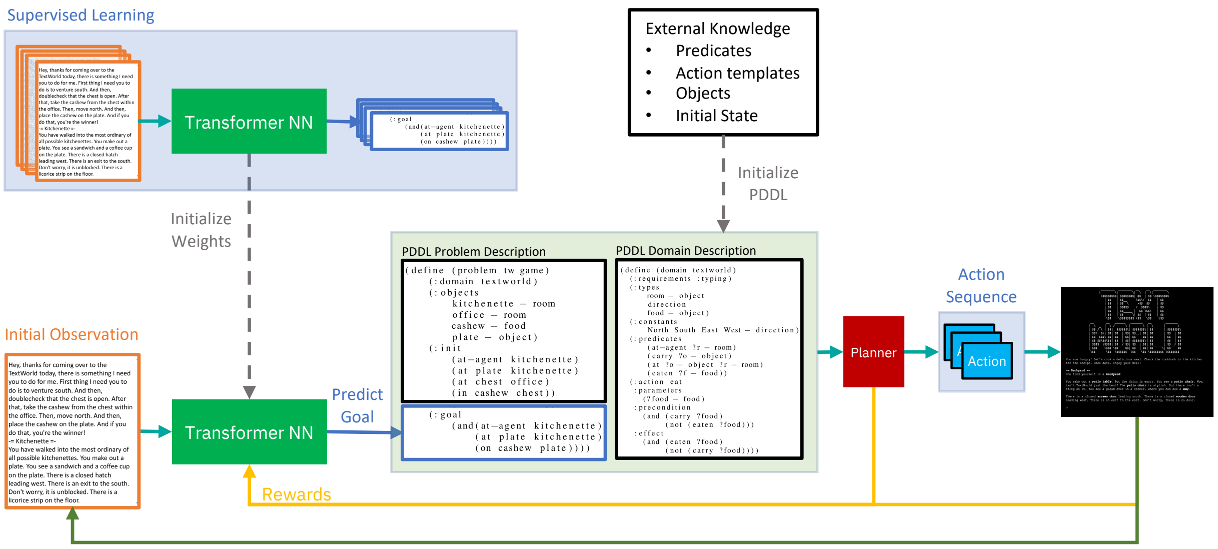
Theorem 1 *For any $\lambda, \sigma > 0$, any $\delta \in (0, 1)$, and any c satisfying*

$$c \ge \frac{1}{\sigma}\sqrt{d\ln\left(1 + \frac{n(K-K')\lambda^2}{d\sigma^2}\right) + 2\ln\left(\frac{1}{\delta}\right)} + \frac{\|\theta^*\|_2}{\lambda},$$

we have,

$$R(n) \le \; 2c(K-K')\lambda\sqrt{\frac{dn\ln\left(1 + \frac{n(K-K')\lambda^2}{d\sigma^2}\right)}{\ln\left(1 + \frac{\lambda^2}{\sigma^2}\right)}} + K' + n(K-K')\delta. \tag{6}$$

This Theorem is showing that we can reduce the regret bound from $\tilde{O}((K)d\sqrt{n})$ to $\tilde{O}((K-K')d\sqrt{n})$ when using CCLINUCB algorithm instead of the classical CombLINUCB. The proof is a direct application of the step done in CombLINUCB.



Hey, thanks for coming over to the TextWorld today, there is something I need you to do for me. First thing I need you to do is to venture south. And then, doublecheck that the chest is open. After that, take the cashew from the chest within the office. Then, move north. And then, place the cashew on the plate. And if you do that, you're the winner!
-= Kitchenette =-
You have walked into the most ordinary of all possible kitchenettes. You make out a plate. You see a sandwich and a coffee cup on the plate. There is a closed hatch leading west. There is an exit to the south. Don't worry, it is unblocked. There is a licorice strip on the floor.

```
(define (problem tw_game)
    (:domain textworld)
    (:objects
        kitchenette − room
        office − room
        cashew − food
        plate − object)
    (:init
        (at−agent kitchenette)
        (at plate kitchenette)
        (at chest office)
        (in cashew chest))
```

```
    (:goal
        (and(at−agent kitchenette)
            (at plate kitchenette)
            (on cashew plate))))
```

```
(define (domain textworld)
  (:requirements :typing)
  (:types
      room − object
      direction
      food − object)
  (:constants
      North South East West − direction)
  (:predicates
      (at−agent ?r − room)
      (carry ?o − object)
      (at ?o − object ?r − room)
      (eaten ?f − food))
  (:action eat
   :parameters
    (?food − food)
   :precondition
    (and (carry ?food)
         (not (eaten ?food)))
   :effect
    (and (eaten ?food)
         (not (carry ?food))))
```

Figure 2: Framework of our TextWorld game-playing agent that uses a Neural Network to set the goals of an AI planning system.

4 The TextWorld PDDL benchmark

Although our task in Fig. 1 is common in the robotics literature, there isn't a common dataset. In this paper, we propose to extend an existing benchmark called TextWorld (Côté et al., 2018) which has an MIT License. This is a type of text-based game where the player can only interact by text - that is all observations and actions are in natural language. By creating PDDL world models from the underlying game logic, we can repurpose it to be exactly the same as our task. An overview of our framework for TextWorld is shown in Fig. 2. Specifically, we are using the *custom* games of TextWorld which can generate the widest variety of *quests*. For each game instance, the *quest* is given at the start which is suitable for our purpuse.

4.1 Generating the TextWorld PDDL files

The Planning Domain Definition Language (PDDL) is the de-facto standard for planners and the AI planning research community (Fox and Long, 2003). The PDDL is composed of 2 parts - the domain and problem. First, the domain description models the general aspects of all the possible TextWorld game instances. It corresponds to the *physics* or *rules* of TextWorld. This is done by defining **predicates** which are templates for logical facts and **action templates** which decompose an action into parameters, preconditions and effects. Second, the problem description models the specifics of each TextWorld game instance. This is done by specifying the **objects** in the world, the **initial state** and the **goal state**. Both the initial and goal states are described by using logical operators along with the defined predicates and objects.

For reference, Figure 2 shows the domain file and a problem file, but due to space only a few relevant examples of each part are shown. Please refer to the supplementary material for full examples. The domain file is common throughout and was extracted only once from the TextWorld game engine. Meanwhile, the problem description can be parsed from a TextWorld game's JSON file which is produced together with the corresponding game instance (Côté et al., 2018). Given both the domain and problem descriptions, standard AI planning systems can be used to solve TextWorld games. In this case, the generated PDDL planning problems are single-agent, discrete time, finite discrete space, fully observable, deterministic, static, non-numeric, non-durative, with non-conditional unit-duration actions, using predicates with typed arguments.

With our proposed *TextWorld PDDL*, one can effectively generate an arbitrarily large dataset of PDDL corresponding to generated TextWorld games. These allow us to tackle specific problems by learning to predict parts of the PDDL while assuming that the rest is known. In this paper we specifically learn to predict the goal state from the given natural language quest.

4.2 The TextWorld PDDL agent

All of our learning framework and examples of the important inputs/outputs are shown in Figure 2. The TextWorld quest in orange border and the output goal in blue border are actual examples from TextWorld and our PDDL interface. The given domain knowledge of the PDDL in black border is a shortened version due to space. With a model trained from supervised learning, we can already use it to play the TextWorld game as shown in Figure 2. Then by using rewards from the TextWorld game itself or deriving negative rewards from errors in the planner, we can refine our model in the bandit setting.

5 Experimental setup

The TextWorld games (Côté et al., 2018) we use for our datasets are custom generated with the house theme, quest length of 5, world size of 5 and containing 10 objects. The random seeds for generating TextWorld games are controlled to be uniquely in only one of the dataset splits (training and testing datasets). For experiments with supervised learning, we started with a *small* dataset size of 4984 games for training and 100 games for testing. For a larger scale experiment with the Transformer, we then used a *big* dataset with 48565 training set games and 9701 games for testing. These are the games left after removing some generated games where there is an unreachable goal(even with expert play). The Transformer Neural Network is a standard implementation using the base parameters of (Vaswani et al., 2017). We optimize this model with stochastic gradient descent using a learning rate of 0.001, batch size of 4, momentum of 0.9, over 100 epochs. For planning, we used the Fast Downward planner (Helmert, 2006) with an implementation of the A^* search algorithm and the landmark-cut heuristic. For comparison, we also trained an RNN model (the GRU with a hidden state size of 128) in a sequence-to-sequence structure.

For supervised learning, The main metric we use is the percentage of *solved* games in the test set which requires a perfect translation of the goal. We also evaluate the semantic parsing task by using the BLEU score. This metric is a good indicator of partially correct translations even if it is less suitable for our task.

In Reinforcement Learning (RL), TextWorld games are known to pose significant challenges (Côté et al., 2018; Adolphs and Hofmann, 2019). This is why all previous works used *intermediate* rewards such that a reward is given after each *correct* action. In contrast, all our experiments use the default *sparse* reward case wherein a reward is only given when the goal is completed (i.e. after a series of actions). This significantly increases the difficulty of the learning task such that it highlights the strengths of our approach. In the first set of experiments, we want to find out how much a simple contextual bandit approach can improve our fully supervised model. To do this, we took the best supervised learning result and used a policy gradient method (Williams, 1992) to refine the result. Our framework's planner rewards give -0.1 whenever there is any planner *error*. The Textworld game gives a reward of 1 if it is completed otherwise 0. A learning rate of 0.0001 was used for the policy gradient updates. In the second set of experiments, we test a more principled contextual bandit approach that we proposed - CCLinUCB. Since our motivation here is in this algorithm's usefulness without our TextWorld supervised learning dataset, we instead use a generic pre-trained model. For this, we chose DistilBERT (Sanh et al., 2019), which is a lightweight version of the well-known BERT (Devlin et al., 2019). We use this directly as a feature extractor to get a context vector such that it doesn't use any TextWorld data for fine-tuning. To limit the combinatorial complexity in our experiments, we use only 20 games. However, this already provides a big combinatorial challenge. Furthermore, in addition to the previously mentioned rewards, we are also able to give a reward that is proportional to the number of reachable goals. This takes inspiration form the *goal-count heuristic*. This is possible in this case since we impose some structure on the bandit arms instead of treating it as a sequence translation.

6 Results and Discussions

Before we present our main results, we point the interested reader to our supplementary materials where we have a small validation experiment to compare our CCLinUCB against a standard LinUCB algorithm. This showed that CCLinUCB converged faster to a reward of about 2.5 compared to LinUCB at only around 2.

Since our TextWorld PDDL dataset is also novel in the literature, we start with supervised learning experiments which are summarized by Table 1. We present this to show the difficulty of the task and as a reference comparison later on. Similar to many Natural Language Processing (NLP) tasks, the Transformer outperforms the RNN model. In

Model	Dataset	BLEU		Solved %	
		train	test	train	test
GRU	small	0.6398	0.3098	0.00	0.00
Tr	small	0.9999	0.6787	99.92	20.00
Tr	big	1.0000	0.7944	100.00	38.19

Table 1: Supervised learning results

Learning Method	Reward	BLEU	Solved %
LSTM DQN	0.0	—	0.0
SL	0.42	0.80	41.0
SL + PG	0.76 ± 0.01	0.84 ± 0.01	61.2 ± 1.92
LM + CCLinUCB	0.91 ± 0.02	—	60.0 ± 7.82

Table 2: Reinforcement learning results

this case, the GRU is unable to solve any games but it has many partially correct translations as indicated by the BLEU score. The Transformer performed extremely well in the training set where it was able to solve almost all the games. However, it did not do as well in the test set where it was only able to complete 20% of the games. One might think that this is due to a lack of diversity in the training set due to a relatively small number of training games, however we can see a similar problem on the Transformer model on the bigger dataset. With this, it still performed well in training and there is a small but significant improvement on the test set. However, the performance gap is still significant. Another possible reason for this might be overfitting to the training set, but this was not the case from investigating our loss curves where both training and testing loss levelled off. Rather, this gap illustrates the main difficulty of TextWorld which shows the need to incorporate a type of *common sense* for better generalization (Murugesan et al., 2020). Practically, this can be thought of as adding much more knowledge into the world model portion and then utilizing this to further guide the learning phase. However, this research direction is orthogonal to our paper and it is out of our scope.

The final results for our contextual bandit experiments are summarized in Table 2. Similar to the supervised learning case, our main metric is still the percentage of solved games. To complement this, we also have the reward that the final trained agent can obtain. The reward has been scaled such that the maximum here is 1.0. We also keep the BLEU score for the supervised model to show how it improves with fine-tuning by policy gradient updates. Under the learning methods, we also include the most common benchmark for text-adventure games which is the LSTM-DQN of (Narasimhan et al., 2015). Its performance in TextWorld even with intermediate rewards is bad since it has very simple action pruning (Adolphs and Hofmann, 2019). In the sparse reward case that we tackle here, we observed that it did not learn anything with 0 reward throughout resulting in 0% solved games. Although this isn't a great comparison because our work uses much more domain knowledge, what this benchmark demonstrates is the difficulty of the *end-to-end learning* task if no domain knowledge was used. To our knowledge, we are also the first to tackle and solve sparse reward games for TextWorld. The second learning method is supervised learning (SL) as another baseline for comparison. The difference in the result here compared to Table 1 is due to the smaller testing set but larger training set. The third learning method starts from our best SL method and fine-tunes it with policy gradient (PG) updates. Our results show that all of the relevant metrics are improved by this method. Notably, the number of solved games increased by about 20% while the BLEU score increased by only an average of about 0.04. This result shows how the main deficiency of our supervised models were addressed by the agent as it has to simply correct a few key words (small BLEU score increase) but there is a large increase in solved games. The last learning method is our proposed CCLinUCB. Our results show that the rewards are significantly higher. In our main metric which is the solved percentage, it shows that CCLinUCB is comparable but slightly lower than the supervised learning with policy gradient refinement. Furthermore, the higher standard deviation can indicate the need for more training iterations and computation budget. However, recall that we do not require labelled data (used in SL) but instead we start from a generic language model (LM) which is understood to have a much worse performance than a specifically trained SL model. With this consideration, we believe that our result is significant and shows the efficacy of our method.

7 Related Work

In the broader scope of literature (outside our specific task), a related group of works is in trying to improve AI planning systems with deep learning. Probably the closest work here is (Miglani, 2019). The difference is that they concentrate on generating the action templates which is orthogonal to our task of generating the goal state. To do this, (Miglani, 2019) builds on the work of (Feng et al., 2018) which uses deep RL for extracting the action sequences from text. The line of work which connects planning and natural language by leveraging machine learning also traces back to the works before the popular rise of deep learning such as (Branavan et al., 2012). Compared to these, recent hardware has made both deep learning and bandit algorithms potentially viable solutions.

Another important part of the related literature is deepRL research for text-based games. Perhaps the first paper to investigate this is (Narasimhan et al., 2015) where an LSTM neural network learns representations of natural language within a DQN RL framework. Since then, text-based games have increasingly gained research attention. Notably, TextWorld (Côté et al., 2018) was proposed as a new testbed and a competition was held with some pre-generated games. The work of (Adolphs and Hofmann, 2019) is a technical write-up of the second place entry in that competition. Compared to these deep RL methods which stem from an *end-to-end learning* approach, our work is very different and it can be seen as starting from a *modelling* approach in classical planning but then we relax this by learning the important parts relevant to our problem. Interestingly, recent approaches in deepRL for text-games have started to relax the *end-to-end learning* philosophy by incorporating concepts from automated planning such as modeled *rules* (Adolphs and Hofmann, 2019) or knowledge graph structures (Ammanabrolu and Riedl, 2019; Ammanabrolu and Hausknecht, 2020; Zelinka et al., 2020; Adhikari et al., 2020).

8 Conclusion

We described the semantic parsing task for producing PDDL which allows automated planners to be used with natural language. We implemented a benchmark and dataset for this that we call *TextWorld PDDL*. We showed a baseline performance of the Transformer on this and demonstrated how to use this model with an automated planner to play the TextWorld game. We then proposed the CCLinUCB algorithm in the contextual bandit learning paradigm to allow for learning from rewards instead of a fully labelled dataset. We believe that this work also opens the door for integrating more deep learning techniques with classical planning which is best for applications where a large portion of the task can be formally modelled.

Ethics Statement

We believe this work contributes toward more ethical AI in two ways. First, we have an *interpretable* logic in between the neural network semantic parsing and robotic planning system. Second, we advocate for training constraints for the semantic parsing system which limits the possible errors. In this work, we don't have a sensitive context or application area.

References

Ashutosh Adhikari, Xingdi Yuan, Marc-Alexandre Côté, Mikuláš Zelinka, Marc-Antoine Rondeau, Romain Laroche, Pascal Poupart, Jian Tang, Adam Trischler, and Will Hamilton. 2020. Learning dynamic belief graphs to generalize on text-based games. *Advances in Neural Information Processing Systems*, 33.

Leonard Adolphs and Thomas Hofmann. 2019. LeDeepChef: Deep Reinforcement Learning Agent for Families of Text-Based Games. *CoRR*, abs/1909.01646.

Prithviraj Ammanabrolu and Matthew Hausknecht. 2020. Graph constrained reinforcement learning for natural language action spaces. In *International Conference on Learning Representations*.

Prithviraj Ammanabrolu and Mark Riedl. 2019. Playing text-adventure games with graph-based deep reinforcement learning. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3557–3565, Minneapolis, Minnesota. Association for Computational Linguistics.

S.R.K. Branavan, Nate Kushman, Tao Lei, and Regina Barzilay. 2012. Learning High-Level Planning from Text. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 126–135, Jeju Island, Korea. Association for Computational Linguistics.

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. 2018. TextWorld: A Learning Environment for Text-based Games. *CoRR*, abs/1806.11532.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages

4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Li Dong and Mirella Lapata. 2016. 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.

Li Dong and Mirella Lapata. 2018. Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 731–742, Melbourne, Australia. Association for Computational Linguistics.

Wenfeng Feng, Hankz Hankui Zhuo, and Subbarao Kambhampati. 2018. Extracting Action Sequences from Texts Based on Deep Reinforcement Learning. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 4064–4070.

Maria Fox and Derek Long. 2003. PDDL2. 1: An extension to PDDL for expressing temporal planning domains. *Journal of artificial intelligence research*, 20:61–124.

Malte Helmert. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246.

Chen Liang, Jonathan Berant, Quoc Le, Kenneth D. Forbus, and Ni Lao. 2017. Neural Symbolic Machines: Learning Semantic Parsers on Freebase with Weak Supervision. In *Proc. 55th Annual Meeting of the Association for Computational Linguistics*, pages 23–33, Vancouver, Canada.

Shivam Miglani. 2019. NL to PDDL: One-Shot Learning of Planning Domains from Natural Language Process Manuals. Masters thesis, Delft University of Technology.

Keerthiram Murugesan, Mattia Atzeni, Pavan Kapanipathi, Pushkar Shukla, Sadhana Kumaravel, Gerald Tesauro, Kartik Talamadupula, Mrinmaya Sachan, and Murray Campbell. 2020. Text-based RL agents with commonsense knowledge: New challenges, environments and baselines. *CoRR*, abs/2010.03790.

Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. 2015. Language Understanding for Text-based Games using Deep Reinforcement Learning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1–11, Lisbon, Portugal.

Florian Nothdurft, Gregor Behnke, Pascal Bercher, Susanne Biundo, and Wolfgang Minker. 2015. The interplay of user-centered dialog systems and ai planning. In *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 344–353.

Diego Perez-Liebana, Jialin Liu, Ahmed Khalifa, Raluca D Gaina, Julian Togelius, and Simon M Lucas. 2019. General video game ai: A multitrack framework for evaluating agents, games, and content generation algorithms. *IEEE Transactions on Games*, 11(3):195–214.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

Shirin Sohrabi. 2019. Ai planning for enterprise: Putting theory into practice. In *IJCAI*, pages 6408–6410.

Richard Sutton and Andrew Barto. 2018. *Reinforcement Learning: An Introduction*, second edition. MIT press.

Jesse Thomason, Shiqi Zhang, Raymond J Mooney, and Peter Stone. 2015. Learning to interpret natural language commands through human-robot dialog. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*. Citeseer.

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

Siwei Wang and Wei Chen. 2018. Thompson sampling for combinatorial semi-bandits. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5114–5122. PMLR.

Zheng Wen, Branislav Kveton, and Azin Ashkan. 2015. Efficient learning in large-scale combinatorial semi-bandits. In *International Conference on Machine Learning*, pages 1113–1122.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450, Vancouver, Canada. Association for Computational Linguistics.

Mikuláš Zelinka, Xingdi Yuan, Marc-Alexandre Côté, Romain Laroche, and Adam Trischler. 2020. Building dynamic knowledge graphs from text-based games.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *CoRR*, abs/1709.00103.