Resolving Large Action Spaces in Text-based Games

Bryon Kucharski
bkucharski@umass.edu

Clayton Thorrez
cthorrez@umass.edu

Rakesh Menon
rrmenon@umass.edu

Sasi Kiran Yelamarthi
syelamarthi@umass.edu

Yash Chandak
ychandak@cs.umass.edu

Marc-Alexandre Cote and **Adam Trischler**
macote@microsoft.com
adam.trischler@microsoft.com

Abstract

Reinforcement Learning has been able to achieve a lot of successes in tasks ranging from video-game play to robotics by trying to maximize cumulative reward. However, a large number of challenges still limit the claim for deployment of these models in real-world scenarios. One such challenge is the question of "how to deal with very large action spaces?". As humans, we are faced with multiple decisions at every instant during our daily lives and hence this is an important challenge. However, current state-of-the-art algorithms face an issue with exploring such large action spaces and hence require an exorbitant number of samples for training models. In this paper, we look to tackle the problem of dealing with a large natural language action space, in a text-based game setting, through a combination of action elimination and action generation techniques. We show the efficacy of our approach using the TextWorld(Côté et al., 2018) environment on a set of cooking tasks in a home world.

1 Introduction

Since their invention, video games have become a popular form of entertainment across the world. With the recent advances in computer graphics and graphical processing units, video games are beginning to closely resemble the real life world. Before graphical games dominated the entertainment industry, text-based video games where often played for enjoyment. Text-based games today, while no longer played as often for enjoyment, are being used in the artificial intelligence and natural language processing (NLP) research community. This is because text-based games are considered a type of dialog system, a heavily researched topic in NLP.

2 Background

Traditional dialog systems are evaluated using a BLEU score (Papineni et al., 2002), where a higher BLEU scores means a better model. In some circumstances, this evaluation metric may result in a low score, but the dialog is still correct. An evaluation metric where a goal is in mind, such as a system successfully ordering coffee from a local shop, may be a better metric to evaluate dialog systems.

In text-based games, the player is usually exploring some sort of world, where the description of the world is given to the player as text. The player is also given a quest to complete such as 'pick up the key in the dungeon.' The player must enter text commands to complete this quest for some sort of reward. Using these kind of games for research provides the desired goal-orientated environment to measure performance.

TextWorld (Côté et al., 2018) is a framework developed by Microsoft Research to generate text-based environments and is the main environment used in the scope of this project. Microsoft Research is hosting a competition using this framework to solve various cooking games using reinforcement learning (RL). The player is placed in a variety of different configurations of worlds and is tasked with finding and following recipes in order to prepare meals. The player may need to explore the world to find the ingredients or cut the ingredients before cooking. Our team is given the option to submit to this competition.

3 Related Work

Solving text-based games using reinforcement learning proves to be a difficult challenge mainly due to the large state and action space. Since the environment and actions are text, the size of the state and action space is combinatorial in the size of the vocabulary. In most situations, the vocabulary will be quite large. Many different approaches can be taken in different sub-areas of RL, such as building a model of the environment, generating a knowledge graph, learning the best state representations, encouraging efficient exploration, etc to solve these text-based games.

One approach by (Li et al., 2016) combines the popular LSTM recurrent neural network with the DQN named LSTM-DQN (Mnih et al., 2013a), which has been given as a baseline for the competition. In this work, the LSTM encodes the observation, and the Q-values are predicted for verbs and objects using a feed forward network. Assuming that each action consists of a noun and a verb, the Q-value for each action is predicted as the average of Q-values for the noun and the verb in that action. LSTM-DRQN (Yuan et al., 2018a) aims to improve on the exploration of LSTM-DQN with the use of a bonus reward (different from the reward given by the environment) based on the number of visits to a state. LSTM-DRQN also uses an additional LSTM cell, which takes as input the history from the previous time step, when encoding the action.

The work proposed in this paper takes an approach of action elimination for large action spaces. The most prominent work dealing with action elimination in text based environments is by (Zahavy et al., 2018) which introduces the Action Eliminate Network (AEN). AEN uses contextual bandits to learn the action elimination using a predicted score $e(s, a)$, for a given state s and action a. An Action Elimination network is used to learn the feature expansion required for the bandit, and DQN training method is used for learning the policy. The main draw back of the AEN is that it assumes that the agent has access to all possible permutations of commands in the entire game which may not always be true or is hard to calculate.

(Tao et al., 2018) uses a generative approach to select commands that are admissible at each time step, rather than eliminating actions. Admissible commands are commands that are recognized by the environment and change the game state. Three separate encoder-decoder pointer softmax models are explored to learn actions based on a given dataset. The main drawback lies in the pointer method for selecting words out of the context (current state) which performs a softmax over the whole vocabulary. This is not feasible if the size of the vocabulary is very large, which is usually the case with text-based games.

4 Methodology

Our algorithm is a hybrid approach that reduces the number of actions that the agent has to consider for decision-making through action elimination at multiple levels. It can be outlined with three distinct modules:

- Context-based word elimination
- Admissible command generation
- Learning to control

Here, the first two modules are used for performing action elimination at the level of words and commands respectively. In the first module, we generate words that are important for producing admissible actions in a particular state. Based on the words that we obtain from the first module (which should be a subset of the vocabulary), we try to generate all the admissible commands i.e. all commands that are valid and result in some change of the game environment. Finally, based on the commands generated, we learn a control policy that learns to decide which action needs to be performed at a particular time-step using Q-Learning(Watkins and Dayan, 1992; Mnih et al., 2013b).

4.1 Eliminating Words

This module is responsible for generating all the valid words for a given state. We first define valid words for a state to be the set of all the words that could be used in any valid action at that state. We next define valid action to be any action that is possible at the given state i.e. any action that results in a change of the state.

The main intuition behind this module is that only a few words make sense given the current state. So, we would like to make the subsequent action generation process simpler by allowing it choose from a smaller pool of admissible words rather than the entire vocabulary.

The architecture of the model used is given in Figure 1 on the left side. Firstly, word embeddings are extracted for the current observation and inventory of the player. We currently use GloVe embeddings (Pennington et al., 2014) but later also plan to explore BERT (Devlin et al., 2018) or ELMO (Peters et al., 2018) embeddings.

We then pass these embeddings of the observation and inventory to an encoder LSTM which

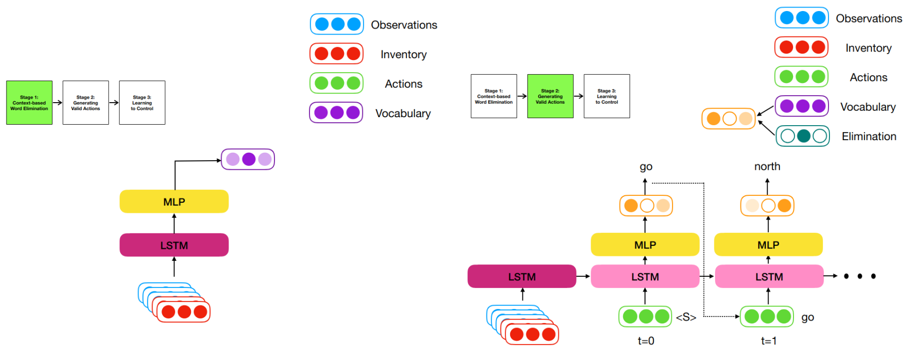
Stage 1: Context-based Word Elimination
Stage 2: Generating Valid Actions
Stage 3: Learning to Control
Observations
Inventory
Actions
Vocabulary
Elimination
MLP
LSTM
go
north
<S>
t=0
t=1

Figure 1: The above figure shows the architecture diagram for the word elimination module (first stage) on the left and the commands generation module (second stage) on the right.

generates a feature representation. This representation is then passed through a decoder MLP to obtain a $|V|$ dimensional vector where each value lies in $(0, 1)$. Here, $|V|$ denotes the size of the vocabulary.

This $|V|$ dimensional vector encodes the probability that each word in the vocabulary is useful at that state. We then threshold this vector to binary values $\{0, 1\}$ where 1 denotes that the word is valid and 0 otherwise and call this the elimination mask. This threshold is obtained based on performance of model on a validation set.

The training data for this task is a dataset used in (Tao et al., 2018) provided by Microsoft Research. This dataset contains a text description for a state and has admissible commands as labels. We modify this dataset by splitting all the admissible commands into valid words for each state. The target vector contains 1s for every valid word thus obtained and 0s elsewhere.

We train this architecture using weighted mean-squared error where weights are inversely proportional to the frequency of occurrence of the words in the whole dataset. This gives larger weight to words which appear rarely and hence eliminates the possibility of the model learning to predict the common words only. This training is done offline i.e. before any RL training.

4.2 Command Generation

4.2.1 Sequence Model

Given the set of all the valid words at a given state, we would now like to generate all the valid actions at that state. Since the number of valid actions is different for different states, we use an approach similar to (Yuan et al., 2018b). Since we need to have a variable number of actions generated, we concatenate all the valid actions for a given state, separated by a delimiter (sep), as a single sequence. This sequence is now made as the target for the decoder to generate.

The architecture for the model is shown in Figure 1 on the right side. Firstly, the current state and inventory of the player are encoded using the pre-trained encoder LSTM from previous stage. This encoder representation is then passed to a decoder LSTM which is trained to generate the target sequence which is the concatenation of all valid actions for that state.

The decoder also has an attention (Bahdanau et al., 2015) over all the hidden states from encoder LSTM. These attention values are combined with the predicted probability vector over vocabulary $|V|$ as a copy mechanism (Yuan et al., 2018b).

At each time step, the output of the LSTM which is a $|V|$ vector is multiplied with a mask which is the modification of elimination mask from previous stage. This elimination mask is modified such that $-\infty$ is placed at all the position with a zero. This ensures that taking softmax at each time step only gives non-zero probabilities to valid words which were not eliminated at previous stage.

The model is trained with loss function similar to (Yuan et al., 2018b). The loss consists of the standard negative log-likelihood term for each term in the sequence. It also contains an orthogonal regularization term which ensures that the decoder hidden states which generate the sentence delimiter (sep) have representation as far apart from each other as possible. This ensures that the sequences generated after the delimiter have some amount of diversity.

4.2.2 Templated Commands

A second approach of command generation is considered where we classify each entity into a type, then fill in the templates to form admissible commands. Each entity can either be a room (r), door (d), container (c), supporter (s), portable object (o), key (k), food (f), or oven (oven). The TextWorld framework provides a list of entities at every time step as well as a list of templated actions for each game. At every time step, the list of entities is assigned a classification label by passing the GloVe embedded entity into a LSTM network and a fully connected layer, then passed through a sigmoid layer for a score. The label with the highest score is assigned to the entity. Then, for each template, the entity type is extracted and filled in based on the list of entities available at that time step. This builds the list of admissible commands to be given to the control module. An example of the templanted command generation is in the Appendix. To build the training data for this model, the training games from the competition were used to extract out every entity and entity type from each state of the game.

4.3 Control Policy

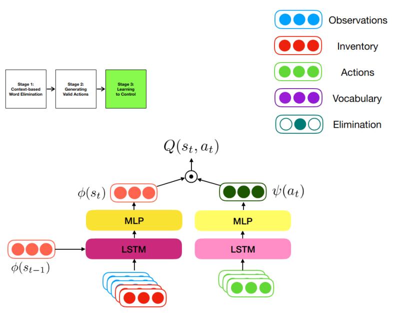

Figure 2: The model architecture used for Q-learning is shown in this figure.

Given the set of admissible commands generated by the agent, we would like to learn a control policy that decides which commands lead to rewards for our task. We consider two common approaches to solving the RL control problem - Actor-Critic and Q-learning.

We briefly describe the details about the two algorithms as well as our modelling choices in the following sub-sections.

4.3.1 Q-Learning

Here, we estimate the $Q(s, a)$ values over the set of admissible actions generated as,

$$Q(s, a) = \phi(s)^T \psi(a)$$

where $\phi(s)$ is the context vector obtained from word elimination stage i.e. c_{ew}. $\psi(a)$ is an GRU that encodes the action into a representation and these parameters of GRU are learned by the RL algorithm. We plan to use Q-learning (Watkins and Dayan, 1992) to learn the parameters, an approach similar to DQN (Mnih et al., 2013b).

We adopt the framework of the Deep Reinforcement Relevancy Network (DRRN) (He et al., 2016). In this approach, $Q(s, a)$ is computed for each feasible action for the current state and the squared temporal difference error is minimized via gradient descent.

$$\text{Loss} = (r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))^2$$

Additionally to prevent action repetition and action cycles, we introduce an additional LSTM which takes in the encoded state and each time step and re-encodes it. This makes each final state encoding aware of information from all past states. This was inspired by Deep Recurrent Q-Learning (Hausknecht and Stone, 2015).

4.3.2 Actor-Critic

Q-learning described in the Section 4.3.1 uses the Q-values to parametrize the policy. An alternate approach is the use of actor-critics (Sutton and Barto, 2018), which is a class of algorithms in RL that aim to explicitly learn the policy and value functions separately while attempting to maximize the cumulative rewards available from the environment

Similar to our architecture in the previous subsection, we encode the each state (s) and each action (a) using GRU encoders to obtain representations $\phi(s)$ and $\psi(a)$ respectively. Following which, we perform a dot product between the state and action encoding to obtain a score function $p(s, a)$,

$$p(s, a) = \phi(s)^T \psi(a)$$

Now, we do this for each admissible action for each state and perform a softmax over the score functions to obtain the policy $\pi(s, a)$.

To perform action sampling, we use the Gumbel-max trick for categorical distributions as proposed in (Jang et al., 2016). In addition, we use a multi-layer perceptron over the state encoding $\phi(s)$ to obtain the value function $V(s) = w^T \phi(s)$.

Finally, our training closely matches that of an existing architecture in A2C (Schulman et al., 2017). We optimize the standard advantage-based policy gradient along with the TD-error for the value function and an additional entropy loss for the policy to induce some stochasticity and aid exploration. Our loss function can be mathematically written as,

$$L_{\text{policy}} = \Big(\sum_{k=t}^{T} \gamma^k r_k - V(s_t)\Big) \log \pi + \pi \log \pi$$

$$L_{\text{value}} = \| \sum_{k=t}^{T} \gamma^k r_k - V(s_t)) \|_2^2$$

$$\text{Loss} = L_{\text{policy}} + c * L_{\text{value}}$$

Here, r_t is the reward obtained for executing action a_t in state s_t. c is a hyper-parameter that we obtain using empirical search. Note, while we mention just π in the above equation due to space constraints, it is indexed as $\pi(s_t, a_t)$.

5 Evaluations

To evaluate the word generator, the precision, recall, and f1 score will be used. For each test state description, the model predicts a score for each word in the vocabulary for every input state description. This score is then thresholded to receive a binary list of words that are most likely to appear in commands. Using the label, a target list is composed and the precision, recall, and f1 score are calculated.

$$f1 = \frac{2 * (precision * recall)}{(precision + recall)}$$

For evaluating the command generation model, we again compute precision, recall and f1 score between the ground truth valid actions and the sequence of actions that are generated by the model.

To evaluate the control policy, we look at the final score the agent achieves in a game. Ideally, the agent will receive the maximum possible score in the shortest amount of time.

6 Baselines

So far we have been attempting to overfit to a very simple environment and have been comparing to an implementation of LSTM-DQN (Mnih et al., 2013b) and an agent which takes random actions at each step. Table 1 shows the performance of these baselines on set of hidden games of the competition.

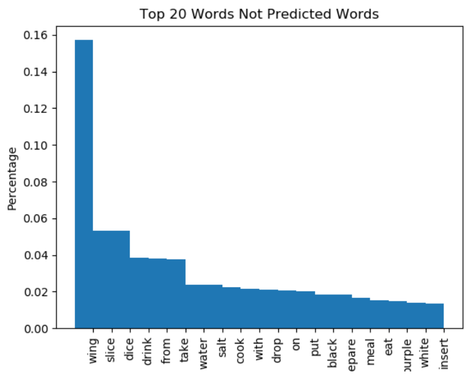

Figure 3: Percentage of top 20 Missing Words (Words not generated when they were supposed to be)

Algorithm	Score (w/o handicap)	Score	Steps
Random Agent	781.5	1563.0	132279
LSTM-DQN	266.5	410.0	214180

Table 1: Performance of baselines on hidden games.

7 Results

Fig 3 shows the performance of the word elimination model on the test set from (Tao et al., 2018). The bar plot shows the top 20 words that are most frequently eliminated when they should not have been. For example, the word 'wing' was eliminated 16 percent of the time when it should have been included in the word set. The performance of this model appears to be acceptable to begin with, but better word embeddings may improve performance and will be explored next.

The command generator module is still currently not performing well with the top performance until now being an f-score of $\sim 10\%$. In spite of this, the model has at least learned the correct ordering of words in each sentence to form a meaningful one i.e. verb should be followed by a noun, etc. An example result of this valid action generation is shown in Appendix A. Because of the low performance, we began exploring the template-based action generation described in Section 4. The TextWorld environment provides the functionality to receive all admissible commands at each time step. The RL results shown are using this handicap, but with the intention of using the template-based action generation instead of the framework to provide a list of admissible commands in future work. We evaluated the RL algorithms, on two TextWorld environments. One very easy where the agent is in one room and just needs to cook and eat a meal, and one a little harder

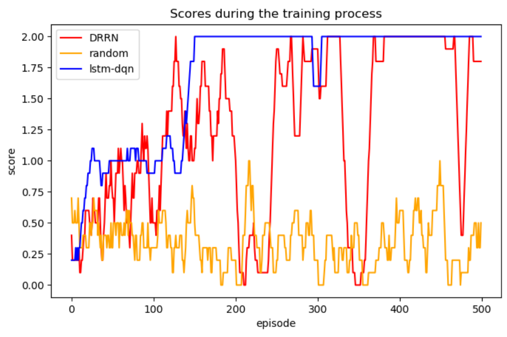

Figure 4: DRRN training curve on a simple 2-step (optimal) cooking task.

where the agent needs to collect a small amount of items before cooking. The training curve for DRRN which learns to overfit to the simple game can be found in Figure 4 along with results for the baselines.

The agents also learn on the medium game but perform worse than the LSTM-DQN baseline and are prone to unlearning as seen in Figure 5. We also experimented with adding randomness to the initial state by performing random actions at the beginning to infer whether the model just memorizes the action sequences. This did not affect the rewards so the model is capable of learning without simply memorizing.

8 Conclusion and Future Work

In summary, we proposed an end-to-end method for training agents for solving text-based games. We encountered a bottleneck in the command generation sequence model which inhibited a full end-to-end model, but began to implement a template-based command generation module to address this issue. We were still able to train the RL agent separately on individual games using a list of admissible commands provided by the framework, but hope to eliminate this in future work. In addition, we hope to use better pretrained embeddings (ELMo or BERT) and develop a method for training the RL algorithms across multiple games at once.

IEEE Conference on Games is holding a paper track for the TextWorld competition. We will continue to work on the templated command generation to try and get and end-to-end model working for this submission. This would also give the option to submit to the competition, but we are unsure if we will submit since we choose a general approach to solve our problem and a hand engineered solution may be best for scoring high on the competition.

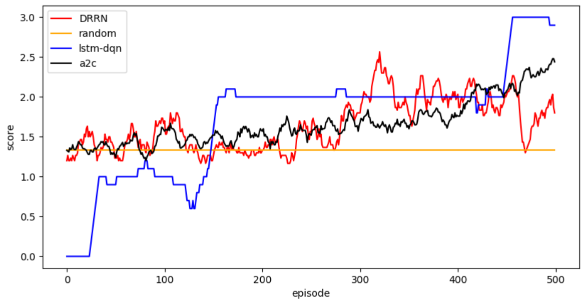

Figure 5: DRRN and A2C on the medium game.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, et al. 2018. Textworld: A learning environment for text-based games. *arXiv preprint arXiv:1806.11532*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Matthew Hausknecht and Peter Stone. 2015. Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposium Series*.

Ji He, , , Jianfeng Gao, , , and Mari Ostendorf. 2016. Deep reinforcement learning with a natural language action space. ACL - Association for Computational Linguistics.

Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. 2016. Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541*.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013a. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013b. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proc. of NAACL*.

John Schulman, Xi Chen, and Pieter Abbeel. 2017. Equivalence between policy gradients and soft q-learning. *arXiv preprint arXiv:1704.06440*.

Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*.

Ruo Yu Tao, Marc-Alexandre Côté, Xingdi Yuan, and Layla El Asri. 2018. Towards solving text-based games by producing adaptive action spaces. *arXiv preprint arXiv:1812.00855*.

Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning*, 8(3-4):279–292.

Xingdi Yuan, Marc-Alexandre Côté, Alessandro Sordoni, Romain Laroche, Remi Tachet des Combes, Matthew Hausknecht, and Adam Trischler. 2018a. Counting to explore and generalize in text-based games. *arXiv preprint arXiv:1806.11525*.

Xingdi Yuan, Tong Wang, Rui Meng, Alissa Sigal, Daqing He, and Adam Trischler. 2018b. Generating diverse numbers of diverse keyphrases. *CoRR*, abs/1810.05241.

Tom Zahavy, Matan Haroush, Nadav Merlis, Daniel J Mankowitz, and Shie Mannor. 2018. Learn what not to learn: Action elimination with deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 3566–3577.

A Appendix - Results on an example

State Description	-= kitchen =- you arrive in a kitchen. a standard one. you can see a fridge. there's something strange about this being here, but you can't put your finger on it. the fridge contains a raw pork chop, a red onion, a white onion, a cilantro and a black pepper. you can see a closed oven. you can see a table. the table is massive. but the thing is empty, unfortunately. oh! why couldn't there just be stuff on it? you can make out a counter. on the counter you can see a banana, a yellow bell pepper, a raw purple potato and a cookbook. you can see a stove. the stove is conventional. but the thing is empty, unfortunately. there is a knife on the floor.
Inventory	a fried yellow potato, an orange bell pepper, a sliced carrot .
Valid words (not eliminated by first module)	banana, bell, black, carrot, chop, cilantro, close, cook, cookbook, counter, dice, drop, eat, examine, fridge, from, insert, into, knife, meal, on, onion, orange, oven, pepper, pork, potato, prepare, purple, put, red, slice, stove, table, take, white, with, yellow
Valid actions (generated by second module)	chop orange bell pepper counter / put orange bell pepper on table put orange bell pepper on counter you / chop orange bell pepper a orange bell pepper / put orange bell pepper on table / put orange bell pepper on table / put orange bell pepper on counter

Table 2: Positive Example of Command Generation

State Description	-= bathroom =- you're now in the bathroom. you can make out a toilet. unfortunately, there isn't a thing on it. you move on, clearly infuriated by your textworld experience. you need an exit without a door? you should try going east.
Inventory	you are carrying nothing.
Valid words (not eliminated by first module)	bathroom, door, east, examine, from, go, put
Valid actions (generated by second module)	a toilet / thing toilet / toilet toilet / on toilet / you on / toilet toilet

Table 3: Negative Example of Command Generation

State Description	-= Kitchen =- The fridge contains a red hot pepper. You make out an oven. Oh wow! Is that what I think it is? It is! It's a table. You see a cookbook and a knife on the table. You can see a counter. On the counter you see a red apple. Were you looking for a stove?!
Entities with classification in brackets (provided by Textworld Framework)	fridge {c}, oven {oven}, counter {s}, red apple {f}, red hot pepper {f}, knife {o}
Template (provided by Textworld Framework)	cook {f} with {oven}, eat {f}, open {c}, put {o} on {s}, dice {f} with {o}
Valid actions (generated by second module)	cook red hot pepper with oven, cook red hot pepper with stove,eat red apple, eat red hot pepper, dice red hot pepper with knife, dice red apple with knife, open fridge, put red hot pepper on counter, put apple on counter

Table 4: Example of Templated Command Generation