Solving Text Games with Machine Learning

Jordan Patterson
University of Victoria
jpatts@uvic.ca

Abstract

This project explores applying traditional machine learning techniques to text-based games, which are a relatively unexplored domain. The algorithms in question are Seq2Seq with Bahdanau attention and PlaNet [1, 2, 3]. Seq2Seq remains unmodified, but PlaNet needed to be transitioned from vision to text. Results are that vanilla Seq2Seq with attention was not powerful enough to fully learn the state space of stochastic test-based games. This could have been due to preprocessing or normalization issues, but more likely just a lack of recurrent units. The PlaNet model was not ready in time to test in the text environment, due to very convoluted source code and unique training requirements. However, based on other researcher, it is suspected that PlaNet would perform exceedingly well in this environment.

1. Introduction

Using AI to solve games has long been an inspiration for the machine learning community. AlphaGo Zero and AlphaStar are two very prominent examples of this, and have achieved fame both inside and outside of the research community [4]. However, despite visual games in practically every genre receiving widespread attention, very little research has been done into text-based games. This is strange, as text games pose a very interesting problem that can be directly applied to real world scenarios. Namely, the ability to piece together a sentence from a bag of words, which is the correct response given the context. Not only would this bolster current translation and generation techniques, it would also help researchers better understand how language is learned.

As a means to this end, Microsoft has recently taken a heavily vested interest into solving Text games, likely for internal benefits. One such creation from this is Textworld, an NLP sandbox providing a variety of text environments to interact with, such as Zork [5]. It is the backbone of this project, sourcing all of the text-based environments for training and testing. These environments are randomly generated each session, and interaction is possible by supplying a command to the environment. Commands are composed of actions and entities, with possible actions being 'chop', 'close', 'cook', 'dice', 'drink', 'drop', 'eat', 'examine', 'go', 'insert', 'inventory', 'lock', 'look', 'open', 'prepare', 'put', 'slice', 'take', and 'unlock'. Some actions are coupled with prepositions 'with', 'into', 'on' and 'from', such as 'chop onion with knife'. Each environment has a number of commands required to complete the scenario, often not required to be done in a specific order. This introduces further entropy, as the optimal path to completion is completely unobservable. With numbers of entities reaching into the double digits in a given session, this is truly a large state space to map.

It may seem simple in concept to apply RNN algorithms to text based games, however this could not be further from the truth. There are several difficult problems that need to be overcome before success can be achieved, which will be the focus of this paper.

1. The tasks require human level inference of a situation. There are 20,200 unique characters or words in the vocabulary for Textworld, and any of them can appear at a given time in the environment. Due to the number of possible combinations of these words in a command, it is not feasible to learn all embeddings beforehand. Therefore, the model needs to infer using prior knowledge if and how a new word it hasn't seen before interacts with its existing data, and how to construct a new command with it. For example, the difference between 'cut yellow potatoe with knife' and 'cut orange carrot with knife' may seem inconsequential to us, as both are vegetables, but to the model it can be a difficult task grouping by adjective as well as noun.

2. Sometimes backtracking is required to complete a scenario. This is notorious in the majority of reinforcement learning agents today, and is a very difficult problem to solve. There are a variety of exploration techniques one can employ, such as Boltzemann and ε-

greedy, however training the model to learn backtracking rather than doing so pseudo-randomly is not easy.

3. The sheer scope of a randomly generated text game could be too large for traditional networks. As evidenced with the prior two points, the number of possible combinations combined with exploring and potentially backtracking an environment create an insane demand for learning capabilities.

These problems have yet to be solved for text games, despite more papers being published in the area recently.

2. Related Work

As stated above, the models used in this paper are from "Sequence to Sequence Learning with Neural Networks" by I. Sutskever et al. and "Learning Latent Dynamics for Planning from Pixels" by D. Hafner et al. Implementation details will be explained in depth later in the paper. To summarize the former, input to Seq2Seq is a float tensor of word2vec vectors which is passed through an encoder. The encoded input is passed alongside the original input to the decoder, which predicts the following word given the current. This is done for t timesteps, where each timestep is a word generated. During training, timesteps is the max number of words in the truth tensor. During testing, timesteps goes to infinity, and continues generating words until the end of sequence character is generated. The latter is far more complicated. PlaNet uses four different models, as explained in depth later in the paper, to essentially lookahead in the state space and choose the optimal action for future reward. This is referred to as 'planning', which is the policy of this network, predicting the best state sequences. As a model-based reinforcement learning algorithm, it is currently one of the most proficient, and was originally used for locomotion based tasks in OpenAI Gym environments [6].

A group that is also tackling this problem is the Microsoft and Mcgill research group, which published "Towards Solving Text-based Games by Producing Adaptive Action Spaces" recently [8]. The idea of their work is using a modified Seq2Seq network for the purpose of generating all possible commands in a given scenario timestep. They achieved good results using cross entropy loss with the valid commands for a scenario as the truth, thus using strong prior knowledge during training. For testing this payed off, as it outperformed previous SOA designs, such as HRED [9]. The ideas that Microsoft are pursuing are very similar to the ones in this paper, which is a good sign that this project is pursuing a good direction from a research perspective. However, their idea of seperating generation of commands and selection of commands is likely a better one until a more powerful model is created that can do both, such as how Alphago Zero combined both action selection and simulated tree search.

3. Architecture

Seq2Seq is designed as an autoencoder, in that its input is encoded into a hidden state before being decoded into the desired translation. Figure 1 showcases the exact specifications of this model. The loss used in this project was Softmax Cross Entropy with a reward and pad masking. Note that this is not the original loss used in the paper, as it has been modified for Textworld. Here pi is the truth values gathered from the Textworld walkthroughs, p is the generated probabilities from the network, and r is the total reward of the entire episode.

$$L_{Seq2Seq} = r + \sum_{i=1}^{N} -pi \log p \tag{1}$$

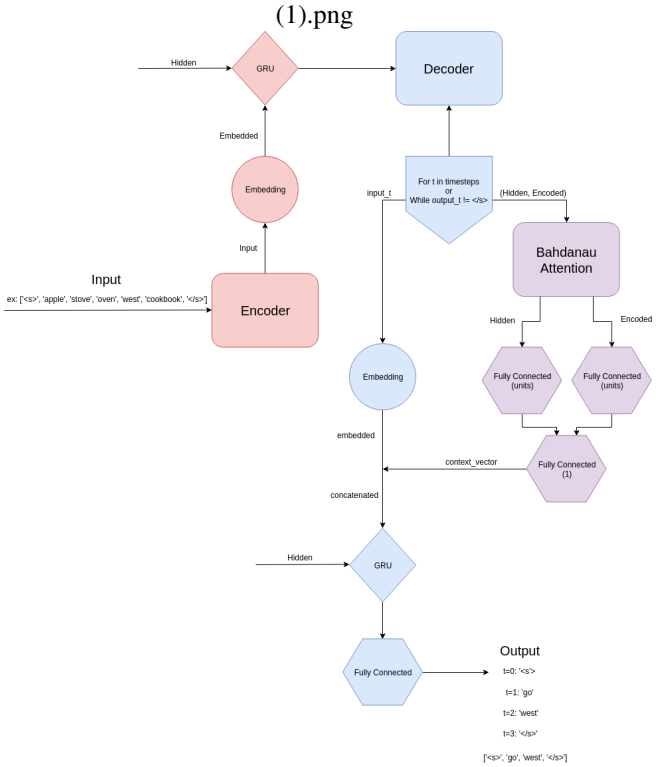

Figure 1. Specification of Seq2Seq Architecture

PlaNet is a combination of several different smaller models, namely the transition, reward, encoder and observation (decoder) models. The encoder changes an observation to a hidden state, while the observation model changes a hidden state to an observation. The reward model takes a hidden state and estimates a reward from the current context. The

transition model predicts future states from the current hidden state, and beliefs. The forward pass of the transition model involves iterating over all the timesteps in a given batch sampled from the experience replay.

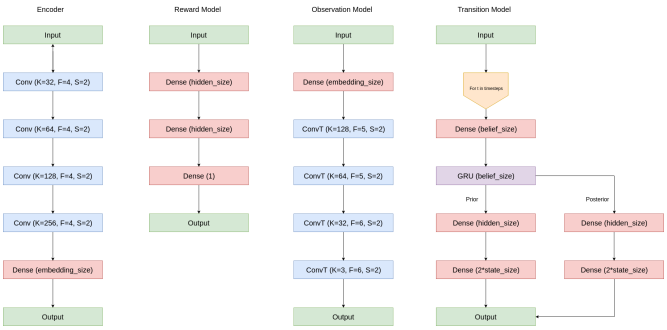

Figure 2. Specification of PlaNet Architecture

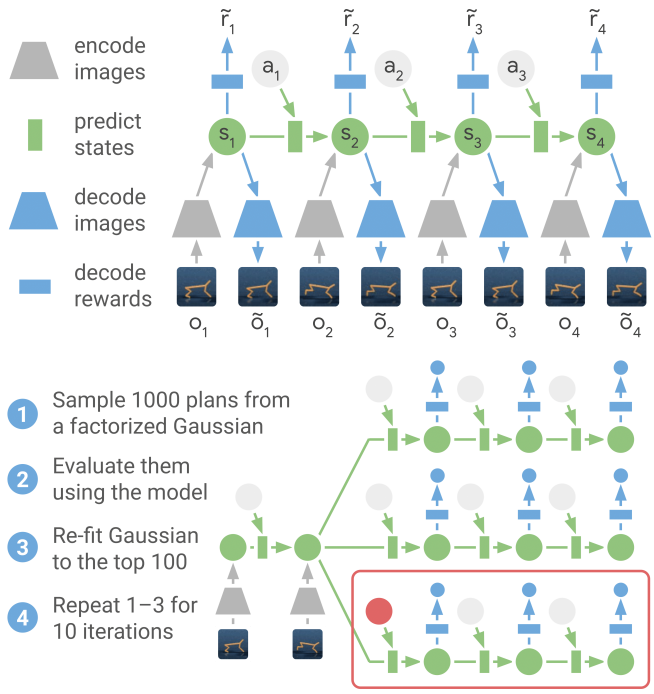

Figure 3. PlaNet High Level Description. Top: Forward Pass. Bottom: Planning [3]

4. Methodology

Data from Textworld is supplied as a text dump, which had to be preprocessed and filtered to be useful. First, all characters besides alphabet letters were removed. Next, a spacy model was used to parse the nouns and adjective-noun pairs from the raw text [7]. Following this the pairs were inserted into a persistent list, which contains all current entities. This list has items removed once used up, such that it is always up to date. For model input, the entity list was vectorized using a word2vec embedding. Three different vector styles were tried, with the best performing unsurprisingly being the simplest. The reason multiple styles needed to be tried is because 'red potatoe' is different from 'potatoe' in Textworld. Thus, adjectives needed to be paired with their corresponding nouns.

The first method used the following syntax: [start, entity, pad, entity, pad, ., end]. By using a pad to separate each entity, the goal was to have the network know [oven, pad, red, potatoe] meant that there are two entities, oven and red potatoe, thus keeping adjective-noun pairs. The following methodology was a twist on this, using the start and end tokens to indicate start and end of adjective-noun pairs. Ex: [start, oven, end, start, red, potatoe, end]. This performed worse than the first methodology, particularly in running time. The last, simplest and best performing methodology was simply listing the entities in order, without separation. Ex: [start, oven, red, potatoe, end]. This shows that the network was able to learn that the adjective came before the noun. Furthermore, the reduced input size meant for much faster training.

The truth, or expected generated vector from the network, was the correct command given the current context. This correct command was parsed from a walkthrough provided by Textworld, and was used exclusively for training. For example, chop red potatoe with knife would be a common command. This truth was then used to teacher force the network, which is feeding the correct word at each timestep, then predicting the following word. This has been shown to provide more stable training, and better yields. The truth values were also used for loss, which was softmax cross entropy added with total reward from the episode. For testing, predictions are done on each prior generated word, and scored on if that command generated a reward.

For training an experience replay was used, which saved model input, truth values and episode reward. Since each step of an episode can have different numbers of entities, and thus varying length of model input, data was padded to the longest sequence in the batch. A mask was used to prevent these padded values from influencing the loss, which was applied to the unreduced output of softmax cross entropy.

5. Results

Although able to learn smaller environments with ease, the Textworld stochastic environment thrown at the Seq2Seq model proved to be too much for it to handle. It consistently learned 2 of the correct words, and their indexes, after hours of training. These are the start and end

tokens, and being static makes this not a great accomplishment. As for more dynamic entities, it was not able to learn the proper mappings between them and commands. It would frequently construct a correct or mostly correct command, except for the wrong context. This shows that it learned which words to generate, but not how they relate to each other. See figure 4 for an example of this behaviour.

```
['<S> take take from <PAD> <PAD> <PAD> <PAD>'] [['<S>' 'go' 'west' '</S>' '<PAD>' '<PAD>' '<PAD>' '<PAD>']]
['<S> take take from <PAD> <PAD> <PAD> <PAD>'] [['<S>' 'go' 'west' '</S>' '<PAD>' '<PAD>' '<PAD>' '<PAD>']]
['<S> take take from <PAD> <PAD> <PAD> <PAD>'] [['<S>' 'go' 'west' '</S>' '<PAD>' '<PAD>' '<PAD>' '<PAD>']]
  3%|
['<S> take take take from take take from'] [['<S>' 'take' 'red' 'hot' 'pepper' 'from' 'counter' '</S>']]
['<S> take take take from take take from'] [['<S>' 'take' 'red' 'hot' 'pepper' 'from' 'counter' '</S>']]
['<S> take take take from take take from'] [['<S>' 'take' 'red' 'hot' 'pepper' 'from' 'counter' '</S>']]
['<S> take take take from take take from'] [['<S>' 'take' 'red' 'hot' 'pepper' 'from' 'counter' '</S>']]
['<S> take take take from take take from'] [['<S>' 'take' 'red' 'hot' 'pepper' 'from' 'counter' '</S>']]
['<S> take take take from take take from'] [['<S>' 'take' 'red' 'hot' 'pepper' 'from' 'counter' '</S>']]
['<S> take take take from take take from'] [['<S>' 'take' 'red' 'hot' 'pepper' 'from' 'counter' '</S>']]
['<S> take take take from take take from'] [['<S>' 'take' 'red' 'hot' 'pepper' 'from' 'counter' '</S>']]
['<S> take take take from take take from'] [['<S>' 'take' 'red' 'hot' 'pepper' 'from' 'counter' '</S>']]
['<S> take take take from take take from'] [['<S>' 'take' 'red' 'hot' 'pepper' 'from' 'counter' '</S>']]
  3%|
['<S> </S> </S> </S> </S> </S> <PAD> <PAD>'] [['<S>' 'drop' 'yellow' 'bell' 'pepper' '</S>' '<PAD>' '<PAD>']]
['<S> </S> </S> </S> </S> </S> <PAD> <PAD>'] [['<S>' 'drop' 'yellow' 'bell' 'pepper' '</S>' '<PAD>' '<PAD>']]
```

Figure 4. Example Output. LHS is generated, RHS is truth

The loss decreased over several hours of training, but eventually stabilized. I believe this is due to the Seq2Seq model reaching its limit for mapping this state space. Since it was originally an algorithm used for straightforward translations, without entropy, adding the immense stochasticity of Textworld is too much for it. Other modified Seq2Seq models that achieved success were much more complex than mine, as evidenced by [8]. Thus it can be fair to assume that adding more GRU layers would yield performance improvements.

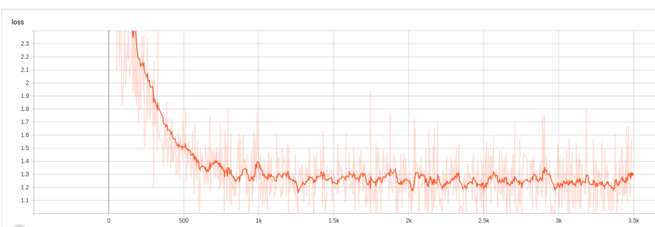

Figure 5. Loss after 3 hours of training

As for the PlaNet model, unfortunately I was unable to convert it to tf.keras in the time of this project. The paper implementations are very convoluted and hard to follow, and thus it was difficult and time consuming to even understand their code, let alone convert it to mine. I managed to get my implementation confirmed working, with correct initial output, up to the observation model forward pass. The reason I stopped here is that a weird shape issue with transposed convolutions was encountered, and I ran out of time to continue. I suspect that PlaNet would significantly outperform Seq2Seq in this task however, as text games are perfect candidates for planning algorithms, with HRED gives this statement factual basis.

6. Conclusion

Initial results of the Seq2Seq model are very promising. Despite huge variance during training (as a result of no normaliation perhaps), it eventually began to converge. During debugging, training on smaller state spaces proved successful as well, with training time increasing exponentially with number of entities. This shows why it was so difficult for the default Seq2Seq to perform well on the full dataset. Future improvements would be to follow [8] and try different use cases for Seq2Seq translation. The overall scope of the project was too large for this scenario, and thus decreasing this would certainly yield better results as a command generator or the like.

Although the PlaNet model was never fully implemented in TensorFlow, working on it really helped me improve my programming skills, as the source code was some of the worst I have seen. I do plan on finishing it this week however, as I think the model conceptually is very sound and I would like to use it in the future. I also think that its less than a day of work from being fully done, and that without the shape and GRUCell issues encountered during conversion, would likely already have been completed. Future steps with this would be to drop in the PlaNet model once its completed, and built up with small examples again, to see if its able to succeed where Seq2Seq failed. Then a comparison with an updated Seq2Seq based on research mentioned earlier would be a good experiment.

References

[1] I. Sutskever et al, *Sequence to Sequence Learning with Neural Networks,* Google, arXiv:1409.3215 , (2014)

[2] D. Bahdanau et al, *NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE,* University Bremen, arXiv:1409.0473, (2016)

[3] D. Hafner et al, *Learning Latent Dynamics for Planning from Pixels,* Google, arXiv:1811.04551, (2019)

[4] D. Silver et al, *Mastering the game of Go without human knowledge,* Google Deepmind, doi:10.1038/nature24270, (2017)

[5] M. Cote et al, *TextWorld: A Learning Environment for Text-based Games,* Microsoft Research, arXiv:1806.11532, (2018)

[6] G. Brockman et al, *OpenAI Gym,* OpenAI, arXiv:1606.01540, (2016)

[7] M. Honnibal et al, *An Improved Non-monotonic Transition System for Dependency Parsing,* Association for Computational Linguistics, 1373–1378, (2015)

[8] I. Serban et al, *Towards Solving Text-based Games by Producing Adaptive Action Spaces,* Microsoft Research, arXiv:1812.00855, (2018)

[9] I. Serban et al, *Building End-To-End Dialogue Systems Using Generative Hierarchical Neural Network Models,* University of Montreal, arXiv:1507.04808, (2016)