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 and PlaNet [?, ?]. Seq2Seq remains unmodified, but PlaNet needed to be transitioned from vision to text. Results are that Seq2Seq was not quite powerful enough to learn the entire domain space of random test-based games. This could have been due to preprocessing or normalization issues. The PlaNet model was not ready in time to test in the text environment, due to very convoluted source code and unique training requirements.

1. Introduction

Using AI to solve games has long been an inspiration for the field. AlphaGo Zero and AlphaStar are two very prominent examples of this, and have achieved fame both inside and outside of the research community. 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 techniques, it would also help researchers better understand how language is learned.

Textworld is an NLP sandbox, providing a variety of text environments to interact with, such as Zork [3]. 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 red apple 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, an LSTM encoder is used on each word vector input to the model, and then an LSTM decoder is ran on the encoded data until an end token is generated, signalling completition of the task. The latter

3. Architecture

PlaNet is a combination of several different smaller models, namely the transition, reward, encoder and decoder models. The encoder changes an observation to a hidden state, while the decoder 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.

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. 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 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 they are static, which shows how it learned them. 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.

6. Conclusion

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 absolute garbage. In the end, it worked up to the encoder part of the model, as there was a bug involving shape and transposed convolutions that I couldnt fix in time. 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. As for my Seq2Seq Textworld model, it exhibited with the smaller MVP examples that it was very capable of learning, and that a full text-based game was simply too large of an observation space. 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.

References