Natural Language Guided Reinforcement Learning for Playing Snake in Arbitrary Dimensions

Alexander Robert Seutin
Department of Computer Science
Stanford University
Stanford, CA
aseutin@cs.stanford.edu

Abstract

Humans receive complex and nuanced feedback when learning to play games. They also have the ability to pose questions for even more specific advice, allowing them to learn quickly and efficiently - exploring complex strategies from the very first time they play. In most reinforcement learning examples, computers are expected to learn on their own without such feedback leading to inefficient exploration. Therefore, balancing exploration versus exploitation has been one of the main difficulties in reinforcement learning. In this project, I attempt to train an agent to play snake in an arbitrary number of dimensions - a task that would be impractical by conventional means. Learning is improved mainly via user given natural language responses to agent produced questions in order to estimate the value function before training. The agent then takes this information and quickly learns to play the game in a low number of dimensions before extrapolating and breaking down larger worlds into lower dimensional slices.

1 Motivation

Modern virtual assistants have make significant strides in recent years, becoming far more practical than they once were. However, after speaking with several engineers on the Siri team, I learned of the many constraints that they face in their work. Due to privacy laws, corporations cannot store user PPI long term. Any information sent to company servers must be wiped within a fraction of a second. Assistants are therefore limited in their ability to learn about user tendencies, to form a model of their historical needs and patterns, and build on what they have learned before. Due to these findings, I have been very interested in emotionally intelligent virtual assistants - an area that has yet to be fully tackled due to the mentioned limitations as well as the fact that there is simply more demand for practical assistants. Building an emotionally intelligent assistant might require more research upfront, but I believe it would pay dividends in the future. Tackling such a problem in the given timeframe of this project would be unreasonable, but I wanted to choose a project that might be a step in that general direction. At a high level, I set out to understand how an agent might learn about the world with the help of the user (through an interactive dialogue), remember that information in its own internal model of the world, use that information to solve a problem in a small setting, and finally to be able to extrapolate and solve more difficult problems given what it has already learned. In my background research, I found interesting papers on reinforcement learning agents that could take in natural language advice and learn to play a game more efficiently using that information. A great example can be found in [**?**]. This team taught an agent to play Atari games by giving it advice such as ”climb the ladder” to help direct it’s exploration.
There seemed to be little efforts, however, to solve the opposite problem: to have an agent request information from the user as it encounters objects in the environment that it has never seen before. Initially, such an agent would require hand-holding from the user to be there to answer all of its

questions, but a more refined and developed version might be able to query its internal knowledge graph to answer them.

2 Introduction and Background

I tackled this problem in several steps. As previously mentioned, I would need to create an agent with the ability to interact with the world in a guided way, using human input in the form of natural language and generating a model of the world to form a memory that it could eventually use in future applications. To do so, I had the agent first play a game without training - recording any unique tokens it encounters in the environment. In Snake, this task was simple. The agent had only 4 items to recognize: the snakes head, the snakes body, the food items, and the background. In my version of the game, the world was represented as a matrix where each cell contained an enum with a value between 0 and 3 (for each of the items). In other applications of these approaches, we could imagine an initial layer of neural architecture able to recognize items in something like an image. This model might label items in the world in a similar way as I did here. To avoid doing so, and to constrain the problem to what was directly relevant, I gave the agent access to a mapping between these enums and a word (ie. mapping[1] = FOOD). The agent would then ask the user what it thought of each of these items and generate a sentiment score for each (stored for later use). I regarded this approach as a way of training the agent to play snake in 0 dimensions - giving it the ability to roughly estimate the value function of a single cell environment in the game.

In a second stage, I implemented baselines and oracles to measure performance in various environments. I then developed several deep Q-learning algorithms that could learn to play snake on their own without help from the user, limiting change in as many variables as possible to get a true measure of relative performance. After finding the best one, I decided to compare various pre-processing methods to alter that data and generalize learning - some of which utilized the estimated value function of each cell calculated by the users feedback. Finally, I attempted to get the agent's trained in small environements to play in higher dimensions.

Some of the big questions that I to answer were how an agent might handle large worlds in which rewards are few and far between, how much a relatively simple neural network could be improved by user guidance, and how much I could generalize training and performance.

3 Assumptions

Here I will be clarifying some of my general goals and assumptions that directed my decision making as I approach this problem. Before starting, I was operating with the idea that a more general way of representing states would help the model perform better when extrapolating and using the trained model in new contexts later on. Due to this, I experimented with various ways of preprocessing the state with this in mind, hoping to show that I could achieve the same results in this altered environment while also improving scalability. In the same vain, I wanted to use models that did not have a memory whenever possible ie. not use any RNNs. This was done because I believed that a human playing this game could play just as well starting from an arbitrary point and wanted the model to be able to perform well without needing to know what happened several moves in the past. I did however, need to capture some degree of movement so I settled by embedding several frames into one larger frame as my input.

4 Game

The agent I will be training will be evaluated on its performance in Snake. As I implemented this environment myself, I will describe how it is played here. In this version of snake, the snake is only able to move straight or in a direction orthonormal to the one it is currently traveling in. If the player attempts to move backwards, nothing happens - the snake will continue to travel in its current direction. At any given time, there is exactly one piece of food in the world. Each time it is eaten (ie. the snake head lands on that tile), it reappears in a new location that was previously empty. If the snake eats something, its tale grows by 1 unit (tile) in length. If the snake ever runs into itself or

a wall the game ends. The score in the game is the length of the snake. The game can also be played with any world size and in any number of dimensions.

5 Baseline and Oracle

To establish certain benchmarks in performs I then implemented several tears of baselines and oracles. The first was an agent that acts entirely randomly in order to establish what the lowest possible expected reward should be in various size worlds. The second tear was an agent that acts randomly without taking any action that will immediately kill it. These agents would tend to run around the world endlessly, so I added a criteria that the game would stop if no food was eaten for more than $world\ size^{world\ dimension} * 2$ time steps ie. the snake would starve. Without this addition, it would have been impossible to train any of my agents as they would not be able to validate rewards due to episodes going on endlessly. The reason I chose $world\ size^{world\ dimension} * 2$ time steps is because $world\ size^{world\ dimension}$ is the number of tiles in the world. In rare cases it is possible for a player to need to travel more than this many tiles in order to reach the next piece of food (because it has to avoid its tail) but it should be on the same order or magnitude. To be safe, I picked a cutoff that was would not limit the snakes performance if it reached the higher levels, while also not detrimentally slowing down training times.// To create the oracles, I did something very similar. First, I created an agent that always moved in the direction that brought it as close as possible to the food (greedy model). Next, I created an agent that picked the move that brought it closest to the food without immediately killing it. The results of those tests can be seen in tables 1 and 2.

Table 1: Baseline Results

Dimension	Size	Random	RandomNonLethal
2	5	0.1 +- 0.36	1.78 +- 0.41
2	10	0.0 +- 0.0	1.68 +- 0.46
3	5	0.1 +- 0.3	1.78 +- 0.41
4	5	0.02 +- 0.13	1.82 +- 0.43
5	5	0.0 +- 0.0	1.8 +- 0.4

Table 2: Oracle Results

Dimension	Size	Oracle	OracleNonLethal
2	5	10.92 +- 3.34	23.74 +- 3.09
2	10	29.2 +- 11.70	547.16 +- 65.09
3	5	14.58 +- 6.47	80.44 +- 17.72
4	5	17.18 +- 7.93	217.86 +- 77.54
5	5	18.9 +- 10.97	577.0 +- 80.85

6 Architecture

I will first describe the architect of each model that was used. Next, I will explain the various processing steps I used in combination with these models. Finally, I will explain how I combined the two in my actual experiments

6.1 Models

Four models were implemented: an RNN for sentiment analysis, a linear model for DQN, a two hidden layer network for DQN, and a convolutional network for DQN.

RNN:
Using 50 dimensional pre-trained glove word vectors from Stanfords NLP library, I created an embedding matrix. Using this matrix, words could be mapped to their word vectors for training and testing. Sequences were padded so everything would be a uniform length and losses were masked so predictions for the NULL words at the end of shorter sequences would not adversely affect training. Sentiments were trained using movie reviews from the Stanford Sentiment Treebank and

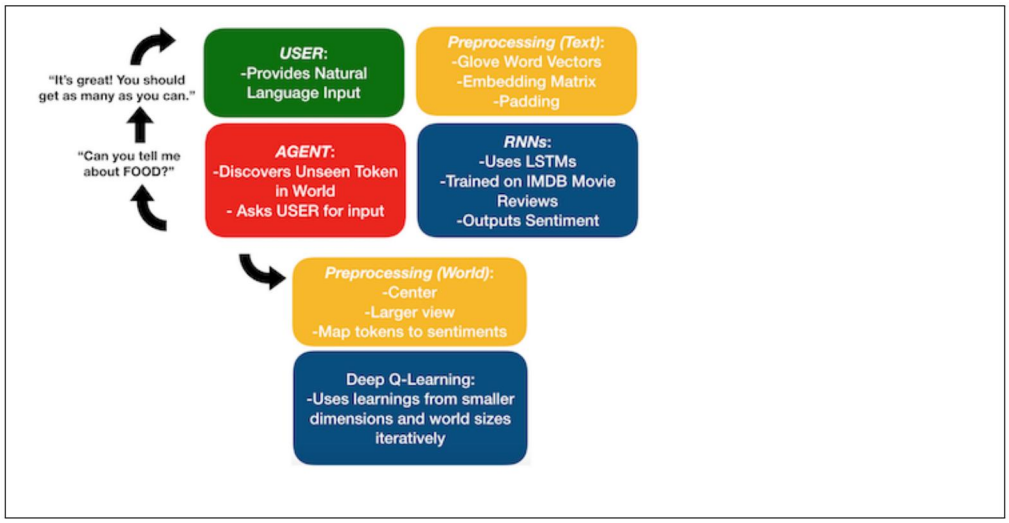

Figure 1: High Level Overview of Architecture of Models

predictions were made by unrolling the inputs.
Important practical note: When using the results from this while testing the DQN models, I created default sentiment values stored so as to stay consistent and compare the models as accurately as possible. I had a high sentiment for food and a neutral one for everything else.
Linear model:
This model is a simple linear predictor built on top of a DQN class provided by CS 234 starter code. Embedded (4 time steps) input frames are flattened and fed into a single fully connected layer.
Two layer Q-learning:
This model is a secondary baseline test. It is a neural network with two hidden layers also built on top of the same DQN class as the linear model. It contains two fully connected hidden layers - the first is size 512 and the second is 256.
Convolutional Q-learning:
Finally the convolutional model has the following architecture: Convolution layers with 16 outputs, a kernel size of 3, and stride of 1 followed by a second convolutional layer with 32 outputs neurons, a kernel size of 3, and a stride of 1, and then one additional fully connected hidden layer with 512 neurons.

6.2 Extra Steps

I will be referring to each of the steps I am about to describe in section 7.
There were several different preprocessing steps I performed in order to try to improve both the speed of learning and generalizability of the resulting models.

1 - Centering:
This preprocessing step is relatively simple. Some of the centered models have had their inputs mapped to a larger world (it is twice the original size in order to allow the snakeHead to be placed in the middle no matter where it lies in the actual original world). This was done in order to make the observed states be more general and to allow the resulting model to be able to play in any larger world with the same dimension (by only observing through a window).

2 - Epsilon rewards:
This step utilizes the sentiment scores by adding a small reward within the range $[-\epsilon; +\epsilon]$ where ϵ is a very small value (0.05 in this case). It is calculated by taking the previous and current states and mapping each cell to their estimated value based on user provided sentiment scores (in default case this means everything is mapped to 0 other than food items which are mapped to 1). We then look at the difference matrix between those states: $S_t - S_{t-1}$. A score is calculated by iteratively moving

out one step away form the snake head pos and finding the sum of the neighbors at that time step. That sum is weighted by $1/2^{steps\ away}$. The score is then scaled to fit within the desired range. It will be positive when the snake has moved closer to the food item and negative when it has moved further. This step is meant to counteract the fact that rewards can be few and far between in larger worlds making training very slow. This also takes advantage of the user provided sentiment scores.

3 - Playing in higher dimensions:
This refers to using a trained model from a lower dimensional world and playing in a higher dimension. I will provide an example here. After training a model in 2 dimensions, it can be applied to 3 dimensions by splitting up the 3d world into many 2 dimensional slices (twice as many slices as the provided world_size parameter). Then, many separate games can be played at once and the action with the highest probability will be played (mapped back to the corresponding action in 3d of course). The idea behind this is that most slices will produce low probabilities for actions if they do not see a snakeHead or a food but in slices where these do appear and the model can be more certain, it will predict the correct action with higher certainty. I also took this a step further by trying the highest average probability and even by projected the snake-head into each slice to create a state that is more similar to the environment the model was trained on.

7 Experiments and Results

As expected, the most powerful model performs the best, but the results are still very mediocre and only slightly improved over the baselines (see figure 2).
I then used the convolutional model as my benchmark to improve upon for all future experiments.

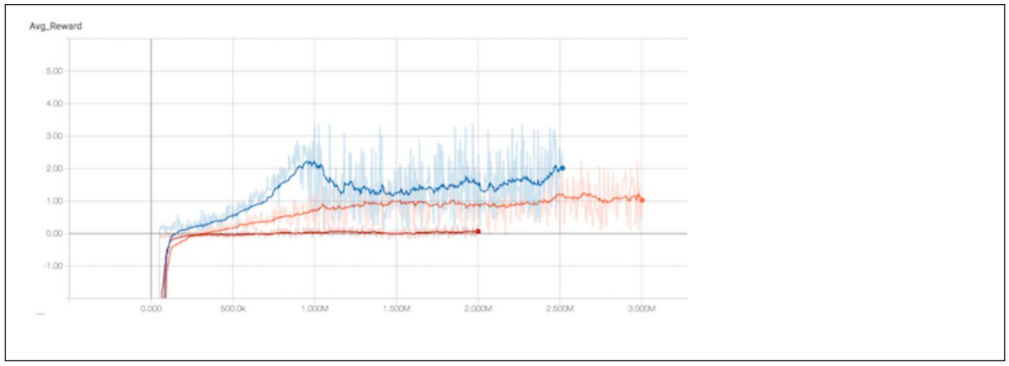

Figure 2: Training Results of Preliminary Models (Avg Reward over Number of Episodes (Red: Linear Model; Orange: Two Layer Model; Blue: Convolutional Model)

It seemed that centering might help speed up learning and improve generalizability, so I tried running the same convolutional model on the centered inputs. Unfortunately, as you can see by figure 3, this actually hurt performance and slowed down training. My theory is that due to the larger input size, it is harder for this relatively simple model to make sense of it. To make matters worse, there is more that changes in the world now from state to state as the whole world is shifting around rather than just one cell within it.

I then decided to try adding small amounts of feedback to help speed up training. I implemented the changes seen in figure 4. This significantly helped performance. Unfortunately, the sentiment epsilon rewards did not do as well as the same model without the sentiment scores did.

I went on to try using the sentiment scores in several other ways but ran into issues. Initially, I mapped the inputs the their estimated value function (sentiment score) directly and fed that into the network. Unfortunately, this blinded the network as it could no longer differentiate between cells that had the same sentiment score, making it impossible to learn. I also changed the initial bias for all cells in the input but did not work well either.

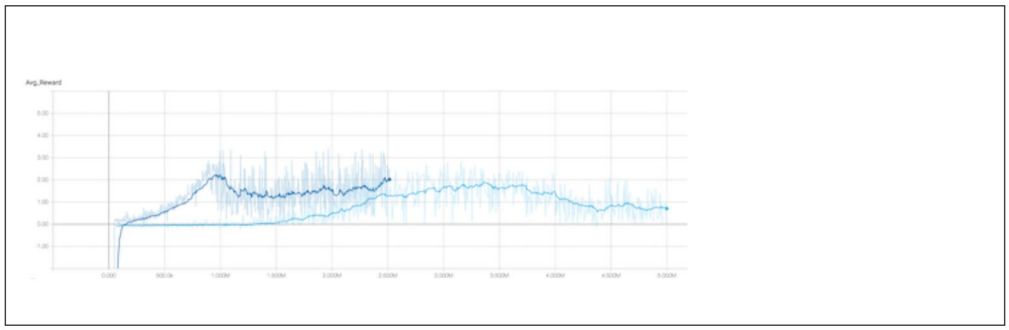

Figure 3: Training Results on Centered Represenation of the World versus Standard Representation Using a the same Convolutional Neural Network as Before (Dark Blue: Basic; Light Blue: Centered)

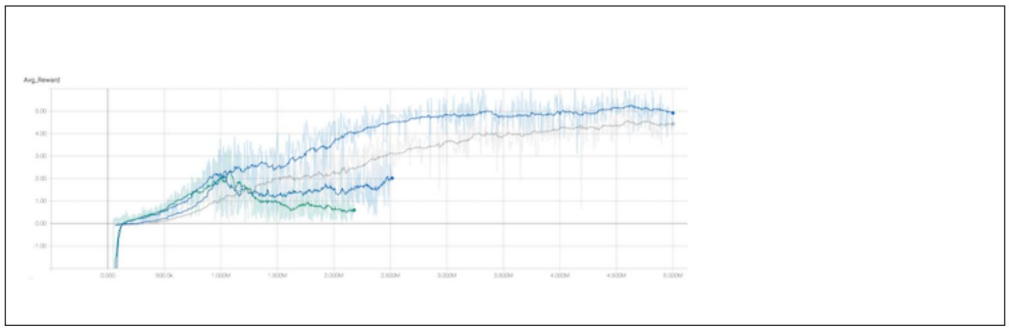

Figure 4: Graph of Average Rewards of Convolutional Model on Standard Representation of World Using Different Epsilon Reward Rules (Blue (Lower): No epsilon rewards; Blue (Upper): Negative epsilon for impossible action, Negative epsilon for no food; Grey: Negative epsilon for impossible action, Negative epsilon for no food, Sentiment epsilon reward; Green: Negative epsilon for no food)

I then put it a lot of time to adapting the lower dimensional models to a higher dimensional world. Disappointingly, this did not lead anywhere interesting.

8 Conclusion

8.1 Take Aways

Overall, despite not finding a major breakthrough, I believe that this work was valuable in exploring some very useful ideas. I found the results to be very surprising. With further research, I hope that this work could lead to exciting developments in this area.

8.2 Next Steps and Analysis of Methods

As an immediate next step, I would have liked to have fine tuned the base model that I was using more to be sure that I had the best possible baseline to build on top of. Similarly, I would have liked to have done far more complete testing regarding the epsilon/cookie crumb trail reward system using the sentiment scores, as I still believe that that has a lot of potential to help training. With more time and compute power, I think it could have been useful to analyze how these approaches scale to larger worlds better as well. It is possible that I have already developed something that scales far better than the basic approaches, but I was not able to test that as the drop-off in speed is so huge as the worlds get even marginally bigger (ex: going from dimension 2 size 5 to dimension 3 size 10 is the difference between a word of size 25 and 1000). Even the baselines has trouble completing enough tests to get statistically significant results on the larger sized worlds. This is exactly why smarter more scalable solutions should be developed to these types of problems. While snake might not be

Table 3: Baseline Results of Models in World of Size 5 and Dimension 2 (*: negative epsilon rewards for impossible actions (moving backwards), negative reward if no food eaten.; **: refer to 6.2; ***: negative epsilon rewards for impossible actions (moving backwards), negative reward if no food eaten, epsilon reward based on sentiment.)

Model	Reward
Linear Model	0.61 +/- 0.21
2 layer model	0.91 +/- 0.25
Conv Model	2.02 +/- 0.35
Conv Model* (adjusted rewards)	7.96 +/- 0.57
Conv Model Centered**	0.68 +/- 0.21
Conv Model Sentiment Epsilon Rewards***	5.70 +/- 0.51

an important issue, other exploration based problems are.
Now seeing as each model took up to 12 hours to train, and I had limited compute resources and time, these approaches were not feasible in the time frame I had available, but I think it would be very worthwhile to do in the future as the possible implications of this type of work are far reaching. I stand by my choice to test these methods with a simpler algorithm, as it allowed me to constrain the scope of the project more narrowly. That being said, I think it would also be interesting to test this with more advanced approaches.
Furthermore, I believe that there could be a lot more done with regards to smart initialization and training that I was not able to fully explore. Rather than simply training a 2d model and running it on slices of a 3d world, I would want to train a 2d model, then initialize the weights of the networks that take in that section of the world with the same values, and then retrain to see how much more quickly the 3d model is able to converge on a solution. I am curious to see if this would help prevent it form wandering around aimlessly so long at the beginning before it has developed a good enough strategy to constantly get at least some reward. Developing the lower dimensional model to play in higher dimensions would be a great proof of concept for what I am trying to explore. It seems to me that intuitively, this type of approach should have a similar advantage to the one that an RNN provides over a window based approach in language modeling. Rather than training many neurons or sections of the network to essentially relearn to do the same thing, it makes more sense to just train a smaller piece to perfect a subtask and then unroll it across each slice of the higher dimensional game. Similarly, I would like to test the centered approach further as this model could even more directly be applied to larger worlds. If the model has been trained to look at a window of size 5x5 and play snake using that input, it could theoretically play a game with any size world since it need only be centered on the snake head. The only downside would be that it will not always be able see the food, but there could be potential work around to this like training two networks and having this one kick in only when the food is within the view. // Finally, I am very interested in other ways one might use natural language responses to agent generated queries in training feedback. Examples beyond what I considered attempting here might include periodically performing actions for the user and asking them for step by step feedback while doing so. Ultimately, if this could be combined with something like [**?**], to develop more of a back and forth between user and agent, we would have something really interesting.

References

Andrea Zanette, Rahul Sarkar. Information directed reinforcement learning. Technical report, Department of Computer Science, Stanford University, Stanford, CA, 2016.

Lampinen, Andrew. 224n project: Natural language learning supports reinforcement learning. Technical report, Department of Psychology, Stanford University, New Brunswick, MA, 2016.

Ramesh, Giovanni Campagna Rakesh. Deep almond: A deep learning-based virtual assistant. Technical report, Department of Computer Science, Stanford University, Stanford, CA, 2016.

Russell Kaplan, Christopher Sauer, Alexander Sosa. Beating atari with natural language guided reinforcement learning. Technical report, Department of Computer Science Stanford University, Atlanta, GA, 2016.

Shane Griffith, Kaushik Subramanian, Jonathan Scholz Charles L. Isbell and Thomaz, Andrea. Policy shaping: Integrating human feedback with reinforcement learning. Technical report, College of Computing Georgia Institute of Technology, New Brunswick, MA, 2016.