Non-Markovian Control Policies for Text-based Games

Surya Bhupatiraju, Simanta Gautam

***Abstract*— Recent work by Narasimhan et. al. (2015) employed a deep reinforcement learning framework to learn control policies for text-based games, where all interactions within the game are through text. This previous framework was Markovian in that the agents selected actions from only the current textual description. However, games in general do not always exhibit this Markov property because relevant information from past descriptions could affect the current action. Our work deals with the non-Markovian nature of these games by preserving relevant information over many time steps using experience replay memory and memory networks. We compare the results of our architecture with the framework in Narasimhan et. al. on custom games with quests that require memory from previous game states.** [1]

I. Introduction

Text-based games involve a player interacting with elements of a virtual world only through text. A player is given a quest as a textual description, and attempts to complete this quest by navigating in the virtual world with natural language commands to take actions. In particular, the player must learn to perform the right actions on relevant objects by understanding the textual descriptions provided during the game. Since the player cannot observe the underlying state directly, she must understand the text to choose relevant actions.

In creating an autonomous agent for such games, we adopt a reinforcement learning framework as done by Narasimhan et. al (2015) [9]. Using reward signals from the game environment , the agent learns to approximate action-value function $Q(s, a)$, which denotes the expected reward over time by taking action a at state s, and performing optimally thereafter. The agent can use the action-value function to form a policy by considering the values of $Q(s, a)$ across all actions a at any given s.

In this work, we parameterize the action-value function using a deep recurrent neural network with a memory network component. The network consists of three modules, the first and third of which are are adapted from [9]. The first module converts textual descriptions into vector representations using Long Short-Term Memory (LSTM) networks [6]. The second module chooses representations of states in previous time steps that are relevant to the current state representation using memory networks [10]. Finally, the third module scores approximates $Q(s, a)$ across all possible actions given the current state representation computed by the first component and the relevant memories selected by the second component.

The agent will learn a good approximation of action-value function by backpropagating the game feedback through all three connected modules in the network. In particular, we use Deep Q-learning with experience replay and priority sampling to train the neural network [8].

To evaluate our model, we create a custom Multi-User Dungeon (MUD) game that involves remembering descriptions from previous timesteps to complete quests [2]. The game environment is based on Evennia, an open-source library for building online text-based games. It is a Python framework that allows one to build new games by writing a batch file that describes the environment with details of the text-based game, including various objects and actions. In addition, the game engine keeps track of an internal game state, providing both textual descriptions of the current state, as well as reward signals.

Since our work involves extending the LSTM-DQN model used in [9] with an integrated memory component, we use the results from that work as our baselines. In particular, we compare our LSTM-Mem-DQN model with the previous LSTM-DQN model on our custom game to show that our model finds better control policies by extracting relevant memories from previous time steps.

II. Related Work

Learning control policies from textual environments is a challenging problem and one that is gaining attention in the NLP community. Games in particular present a rich domain to conduct controlled language analysis; the various states presented and the transitions between them are well understood and can be controlled. Much of past work in this area has involved inferring suitable control policies when provided fully observable game states or given game-related documents and precompiled traces of the game [4][3][1]. Work by Narasimhan et. al and Mnih et. al aimed to jointly infer the state representation from textual or visual information and learn control policies [9][8][7].

In addition to learning novel control policies, there have also been a number of recent efforts to capture long-term structure within sequences using recurrent models. The notion of memory was investigated in number of ways. Steinbuch and Piske [11] and Taylor [12] considered memory that performed nearest-neighbor operations of stored input vectors and then fit parametric models to the retrieved sets, which is abstractly close to how we utilize memory. More work on associative memory was explored in the 1990s, and most recently, the Neural Turing Machine (NTM) by Graves et. al, [5] shows considerable promise for deploying memory to agents. The NTM uses a continuous memory representation and allows for both content and address-based access. Our paper uses the work done in the End-to-end

[1]For the complete code repository, please follow this link: `https://github.mit.edu/simanta/TBGames`

Memory Networks paper [10] because the model is simpler than in the NTM and because we will only write memory sequentially and will not require operations like sharpening.

Our work is an extension of the work by Narasimhan et. al in that we study control policies for text-based games. As aforementioned, we adopt the overall reinforcement learning framework, including the deep Q-network and LSTMs used for language understanding, but we augment the model with memory networks to aid the agent in making more informed decisions. Previously, memory networks have largely been used in isolated question-answering language systems, but we hope that it is extendable for general purpose language processing. Throughout this paper, we demonstrate the overall model and argue for its efficacy.

III. Background

A. Markovian framework

In the approach by Narasimhan et. al., the automatic player is able to learn extensive control policies for completing almost all quests in the various games. As a simple example, the agent may be placed in one of several rooms and be tasked with navigating to the kitchen and drinking the water by emitting textual commands to the game environment, such as 'go east' and 'drink water'. In these games, the agent is provided with the description of the current room and the quest description at every time step, and a reward is given if the quest is completed at this time step.

While it works well, the framework was limited in that it was unable to consistently solve quests that required knowledge of prior interactions. In other words, this previous framework was Markovian because the agent only made decisions based on the current description and disregarded prior experiences. However, games in general are non-Markovian in that players must often incorporate past experiences and previous interactions with the game environment to move forward. This is especially in extensive role-playing games or puzzle games when prior knowledge needs to be incorporated with the information being presented currently.

B. Memory Networks

When the agent is able to consider prior information, more informed decisions can be made and more difficult quests can be solved. To this end, we strive to improve on the Markovian constraint of the framework by providing the agent access to memories of previous interactions. We would like the agent to learn to select relevant information from previous timesteps given the current textual description. However, integrating a memory component into the framework is difficult because we would like to be able to backpropagate the game feedback to the memory component so that an agent can better learn to select relevant memories.

To resolve this issue, we adapt the single memory hop architecture from Sukhbaatar et. al (2015) [10]. This allows us to integrate the memory component into the existing framework so that the game feedback can be backprogated through the entire framework. In this way, the agent can learn to select particular memories that can be combined with the current state representation to improve controller. The overall model is described in full detail in Section IV.

IV. Model

One of the main contributions of this work is the LSTM-Mem-DQN model, which is presented in Figure 2. As is described by Narasimhan et. al., the stream of words observed in state s is propagated through a number of LSTMs, the hidden states are then mean-pooled, and an overall sentence embedding is produced as a vector representation v_i. In the previous model, this v_i was the input to an action scorer, which then decides what action to take at that particular state to yield the highest reward. This entire model is trained by a squared loss with regards to the reward received, and the parameters in the LSTMs and in the action scorer are updated to move towards some local minima.

The goal is that we want to append relevant information to v_i such that the action scorer is able to make a more informed decision about which action to take. This extra information can be produced in a number of ways, but a natural approach is to incorporate previous memories. As such, one copy of v_i is copied to the next part of the model, while the other copy is used to decide which memory to select.

First, v_i is passed into a MemIn module that selects up to 20 previous state representations $x_1, \ldots, x_{20}$ from experience replay where an x_i may be padded with zeros if a particular x_i is not contained in the same episode. These memories are converted to memory vectors m_i of the same dimension computed by embedding each x_i in a continuous space by propagating x_i through a single-layer feedforward neural network with one fully connected linear layer and a ReLU nonlinearity. We compute that match between v_i and each memory m_i by computing the dot product of v_i and m_i followed by a softmax:

$$p_i = \text{Softmax}(v_i^T m_i)$$

where $\text{Softmax}(z_i) = e^{z_i} / \sum_j e^{z_j}$. In this way, we compute the probability vector p over the inputs. Intuitively, this layer encodes the input memory representation.

Next, p is propagated through the MemOut module where we retrieve the same cached set of transitions $x_1, \ldots, x_{20}$ and compute the response vector:

$$o = \sum_i p_i c_i$$

Because the function from input to output is smooth and differentiable, we can easily compute gradients and backpropagate through it [10]. We can interpret o as a *mixture* of the previous transitions where we weigh relevant memories more and others less. This mixture is then concatenated with v_i and this new embedding is passed forward to the action scorer, where an action can be selected.

However, due to time constraints, we were unable to implement the entire memory network attachment, and instead defer to a simpler, more naive memory augmentation depicted in Figure 3.

The model described by this architecture performs a similar notion of 'splitting' v_i into two separate tracks. In the memory track, v_i is fed into a 2-layer feedforward neural network, where the output layer has size 20. The ith in the output layer represents the ith previous transition stored in experience replay; when we compute an `argmax` of the output, we are choosing which transition from the experience replay we are sampling; this ith transition is chosen by the subsequent layer, converted into a bag-of-words representation and then appended to v_i to by scored in the same fashion by the action scorer. We note that this also allows for non-Markovian control policies, but we find that in practice, this approach does not work as well as we thought it would.

V. Experimental Setup

To test the new augmented model, we constructed a custom version of the game world, as well as a number of different experiments to benchmark the performance of the original framework by Narasimhan et. al. on games that required past knowledge to successfully complete, as well as test our new models.

In designing the first experiment, we noted that a quest description was given to the agent at every timestep in the previous work. However, in a realistic setting, it may be likely that the agent is given the quest description in only the first state, and must remember that description as its playing the game. Therefore, we our first experiment involves the same Home World game used in Narasimhan et. al., except that we do not give the quest description as a part of the state representation in every timestep. Instead, we give the quest description on the first state, and only the room descriptions on states thereafter.

For our second experiment, we made changes to the Home World game itself. In particular, we added a few new objects to the game without changing the architecture of the rooms. These changes are described in detail in the following subsection, and the overall architecture of the custom world is shown in Figure 1.

A. Custom Game Environment

We designed a different game than in Narasimhan et. al. to require the agent to recall text descriptions from previous states to find the optimal policy. In this new game, the agent is randomly put in one of four rooms and given the text description of that room as the initial state representation. From there, the agent needs to go to the Living Room to watch the TV, where it will be given the quest description. The quests involves either drinking the soda, eating the apple, napping on the couch, or sleeping on the sofa.

In this game, we incur a penalty using a reward signal of -0.5 if the agent drinks the soda when the quest is to eat the apple, and vice versa, or if the agent sleeps on the bed when the quest is to nap on the couch, and vice versa. By changing the existing reward function in this way, it is more valuable for the agent to watch the TV to retrieve the exact quest description, as opposed to arbitrarily trying actions on different objects.

After watching the TV to retrieve the quest description, the agent must also remember that quest description in the choosing optimal actions in future states. Therefore, this custom game is a good environment to determine whether the memories selected by the agent at each timestep are actually relevant to optimally solving the quest. Ideally, we would like this particular memory to be recalled more frequently, or weighed more heavily when considering what to do any particular timestep.

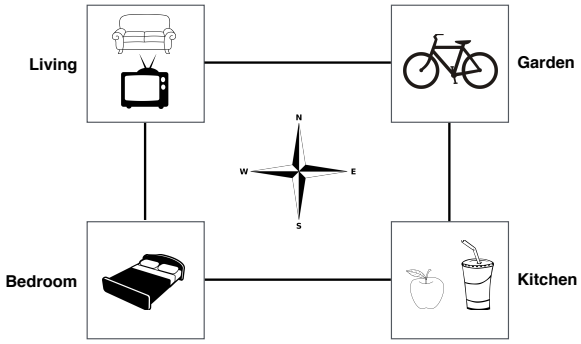
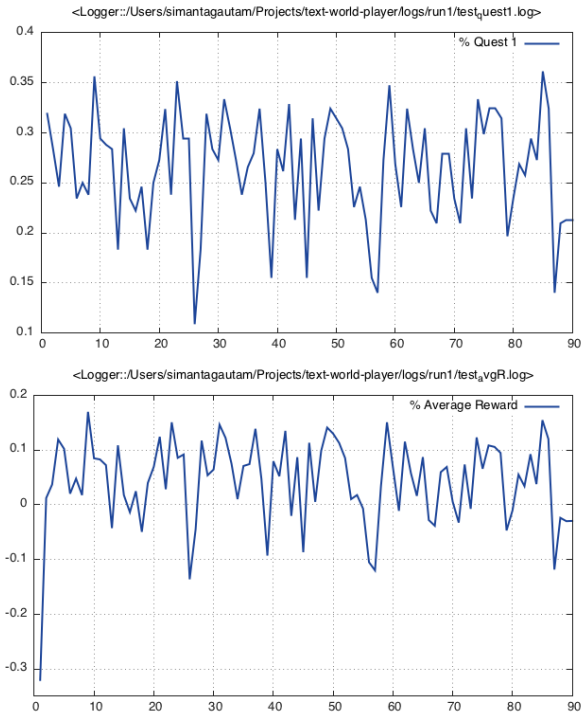


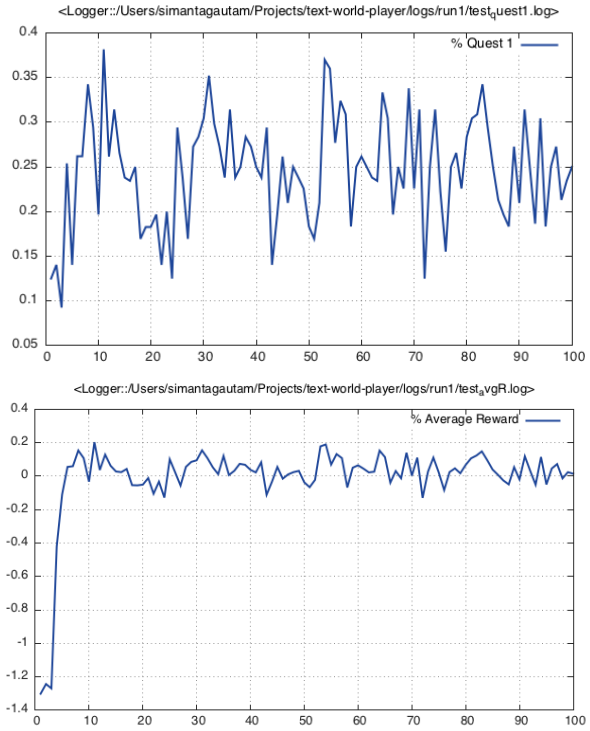
Fig. 1. Our custom game environment.

VI. Results

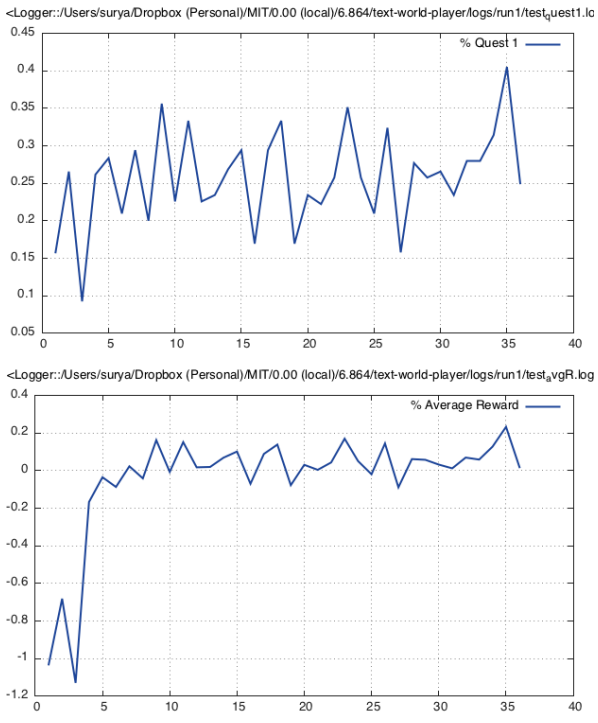
As aforementioned, we run a number of experiments using the more naive model. While less robust than the full memory network, this model provides insight into the difficult of learning non-Markovian control policies and the care with which these must implemented.



These two plots represent the average quest completion rate and the average reward accumulated over test epochs as the iterations increases of the first experiment run on Narasimhan et. al's model.



These two plots represent the same set of data where the second experiment in our custom game world is run on Narasimhan et. al's model.



Finally, these two plots represent the same set of data where the second experiment of only showing the quest once is run on our naive memory framework. The two remaining plots represent the same set of data where the first experiment is run on our naive memory framework; because this data looks very similar, we omit it for brevity.

VII. Discussion

The graphs show a number of results and also demonstrate the difficult in constructing proper memory-acquisition routines. In the case of the first experiment run on the framework by Narasimhan et. al, we see that the average quest completion rate amortizes to about 0.25, because in one-fourth of the possible quests and starting configurations, the agent can solve the task in one step, i.e. doing the right action in the same room. The rest of the time, the agent will act randomly, which is the noise that we see and what contributes to the fluctuations. Because the quest completion rate is relatively low, we see that the average reward is consequently quite low and hovers only slightly above 0.

The following two graphs also show similar, but slightly improved performance. We see that the average quest completion rate is still approximately 0.25, while the average reward accumulated is now slightly higher on average. We see the same behavior happening in the last set of graphs, where we run the second experiment of the custom game environment using our naive memory model. Surprisingly, we see very similar performance; the average quest completion seems to be slightly higher than one-fourth on average, and the average reward accumulated seems only slightly higher. In the plots where we run the custom world using Narasimhan et al's framework, we see similar but slightly worse performance, which is a reasonable conclusion. The game should do no better than our implementation because it should identically as it did the in the experiment, because the quest is only shown once and then forgotten.

Overall, we do not see the performance gains that we expect to, and it is made clear by the various metrics collected. Indeed, adding the memory components in the designated fashion is not helpful to selecting reasonable actions and may actually inhibit the learnability of the task. Furthermore, there are inherent limitations with using a static feedforward neural network to selecting relevant memory; even if we backpropagate errors, the particular indices or index that we select will be dynamic as we encounter more states, and this is not something the neural network will capture.

VIII. Future Steps

There is now considerable reason to believe that fully implemented memory networks will yield much more favorable results. They have the built-in notion of deciding what memories are important to store. For instance, if we continually do invalid actions (such as trying to eat an apple when the agent is in the bedroom), we do not want to store such transitions and have those be considered. Ideally, we would want our memory module to prioritize high-impact transitions, such as the quest given when the agent watches the TV. This is something that the memory network, especially with multiple

memory hops, can theoretically achieve; the naive approach shows no such promise. This is ongoing work and work in this direction is already being made, but was unfortunately unable to come to fruition.

IX. Conclusion

In this work, we looked at extending the neural network model in Narasimhan et. al. to incorporate memory from previous states. The previous work parameterized the action-value function using a deep recurrent neural network, which was trained with game feedback. However, the game sequences were formulated as an MDP and constrained the autonomous player to learn only Markovian control policies. Our work involved integrating a memory component to the previous framework that could be trained to select relevant memories with game feedback, which allows the agent to find non-Markovian control policies.

To this end, we designed two versions of the framework, LSTM-Simple-DQN and LSTM-Mem-DQN, each with an integrated memory component. The former framework was the first approach which used a naive memory selection module, while the latter framework uses a memory network to allow the game feedback to backpropagate into the memory module. Due to the time constraints for this project, the implementation of our LSTM-Mem-DQN model was not complete. However, there is considerable ongoing work in incorporating K-hop memory networks into the existing module, and we believe there is considerable promise with this approach; we've learned that selecting memories to augment control policies is a challenging and nuanced problem.

X. Acknowledgments

Thank you to Karthik Narasimhan and Regina Barzilay for guidance and assistance for this project.

References

[1] SRK Branavan, D. Silver, and R. Barzilay. 2011a. Learning to win by reading manuals in a monte-carlo framework. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1, pages 268277. Association for Computational Linguistics.

[2] P. Curtis. 1992. Mudding: Social phenomena in text-based virtual realities. High noon on the electronic frontier: Conceptual issues in cyberspace, pages 347374.

[3] J. Eisenstein, J. Clarke, D. Goldwasser, D. Roth. 2009. Reading to learn: Constructing features from semantic abstracts. In Proceedings of the Conference on Empirical Methods in Natural Language Processing, pages 958967, Singapore, August. Association for Computational Linguistics.

[4] P. Gorniak and D. Roy. 2005. Speaking with your sidekick: Understanding situated speech in computer role playing games. In R. Michael Young and John E. Laird, editors, Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference, June 1-5, 2005, Marina del Rey, California, USA, pages 57 62. AAAI Press.

[5] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. arXiv preprint: 1410.5401, 2014.

[6] S. Hochreiter and J. Schmidhuber. 1997. Long short-term memory. Neural computation, 9(8):17351780

[7] J. Koutnk, G. Cuccu, J. Schmidhuber, and F. Gomez. 2013. Evolving large-scale neural networks for visionbased reinforcement learning. In Proceedings of the 15th annual conference on Genetic and evolutionary computation, pages 10611068. ACM.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. 2015. Human-level control through deep reinforcement learning. Nature, 518(7540):529533, 02.

[9] K. Narasimhan, T. Kulkarni, and R. Barzilay. Language understanding for text-based games using deep reinforcement learning. CoRR, abs/1506.08941, 2015.

[10] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus. End-to-End Memory Networks. CoRR, abs/1503.08895, 2015.

[11] K. Steinbuch and U. Piske. Learning matrices and their applications. IEEE Transactions on Electronic Computers, 12:846862, 1963.

[12] W. K. Taylor. Pattern recognition by means of automatic analogue apparatus. Proceedings of The Institution of Electrical Engineers, 106:198209, 1959.

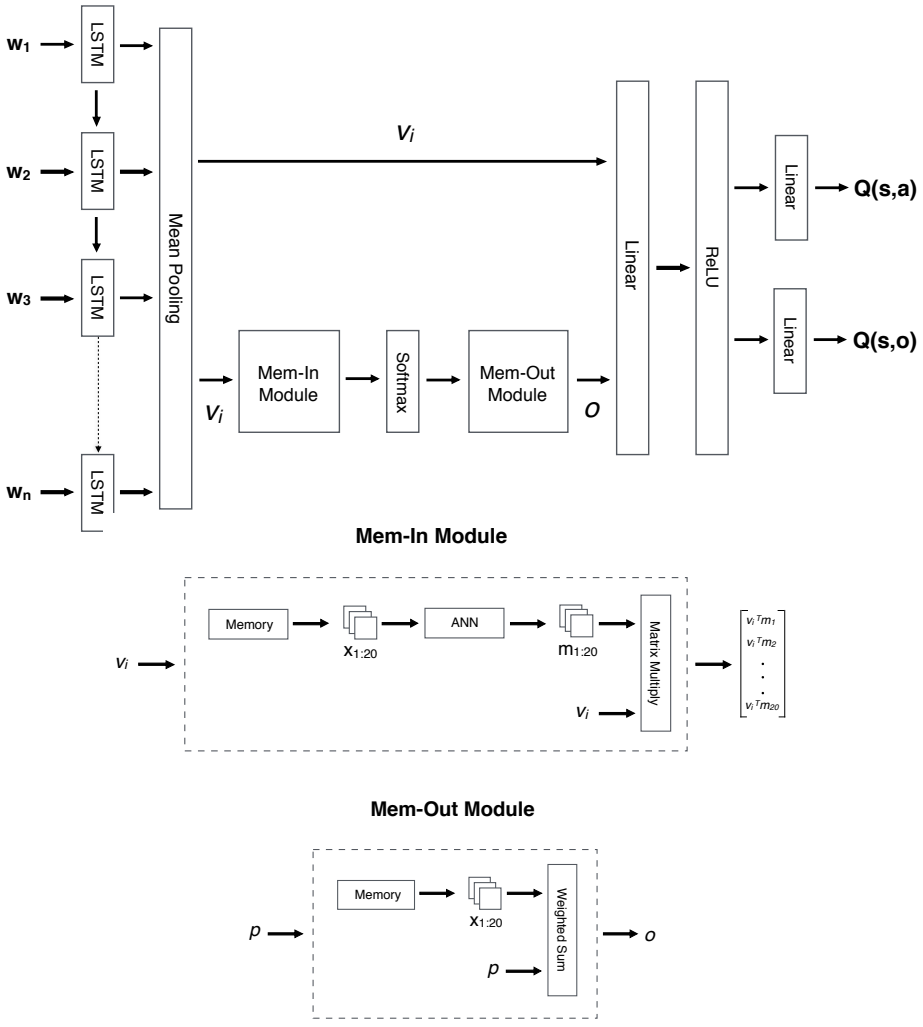


Fig. 2. LSTM-Mem-DQN Model

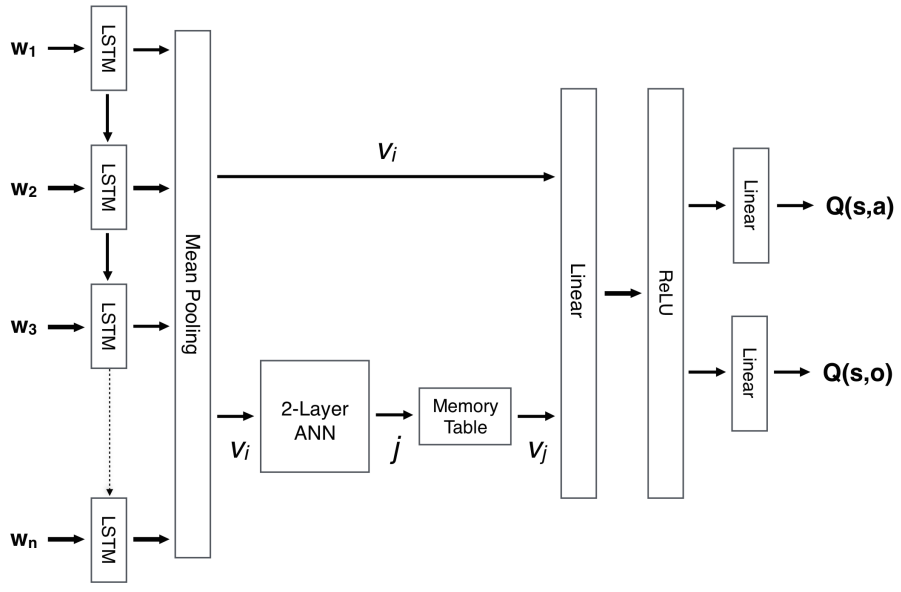


Fig. 3. LSTM-Simple-DQN Model