Project Report

Study on Deep Reinforcement Learning with a Natural Language Action Space applied to Text-Based Games

Teammates: Raique Pereira, Aswin Shriram Thiagarajan, Anand Krishnamoorthy

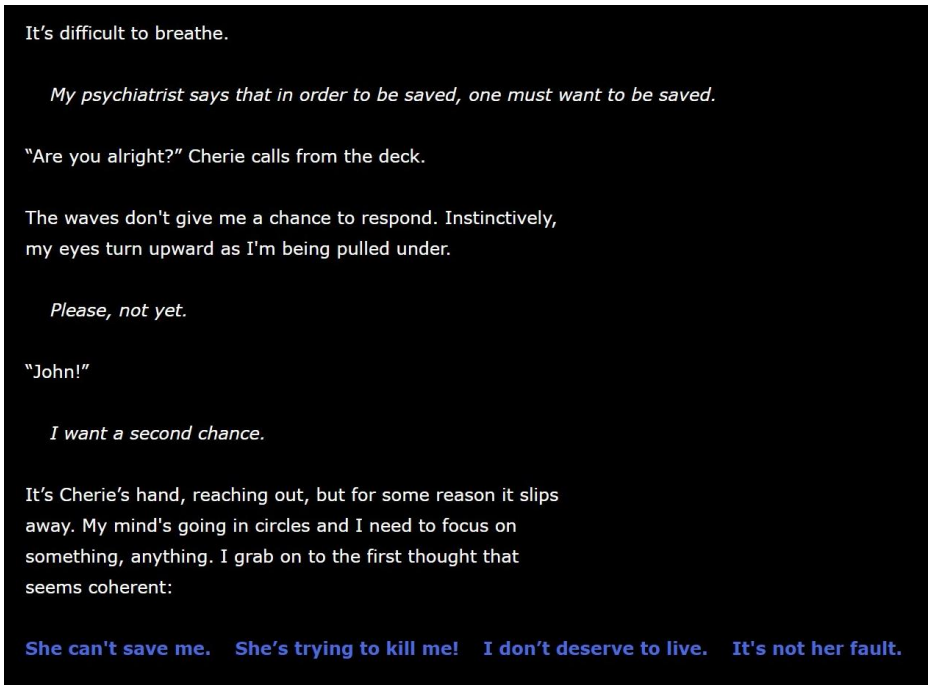
Problem Description:

The project is on the theoretical track and is essentially a study of the intersection between reinforcement learning and natural language processing. The project itself is centered around understanding the Deep Reinforcement Learning algorithms for learning control policies on text-based strategy games. We attempt to study 3 papers to solve this problem statement of text-based games with higher dimensional features.

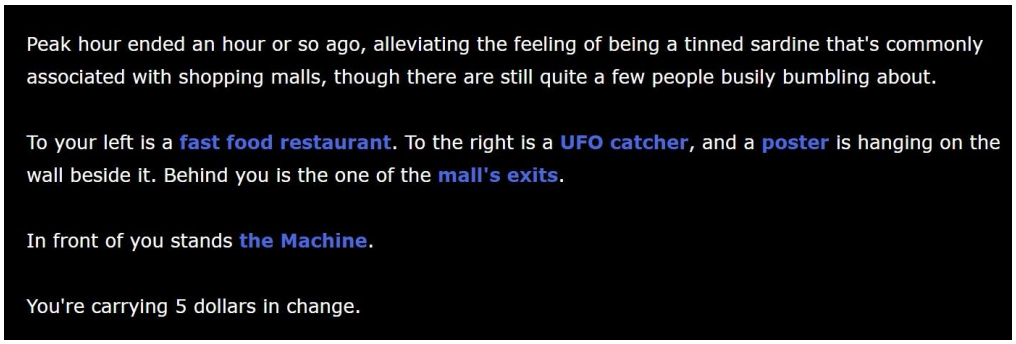
- ★ The LSTM-DQN network.
- ★ The DRRN architecture
- ★ The SSAQN network

The text-based games explored were: Saving John (SJ), Machine of Death (MoD) and Fantasy world (in evennia framework). At the end of the project, we hope to have a clear understanding of LSTM-DRL architecture and hopefully analyze the results obtained from the SSAQN implementation.

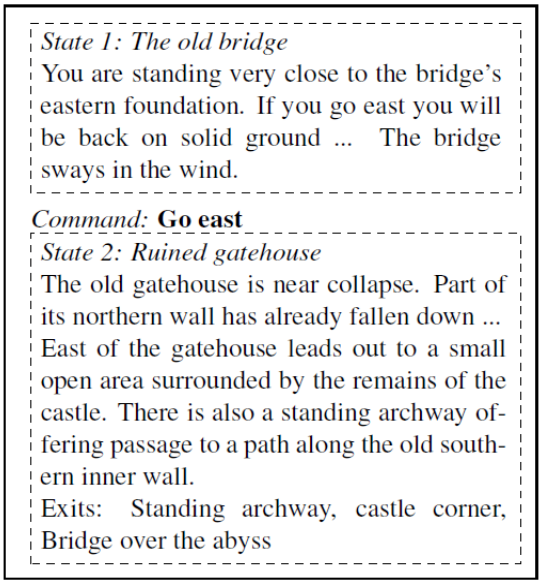
- Saving John:



- Machine of Death:



- Fantasy World:



Motivation:

- ★ Application of RL in a practical real world application
- ★ Explore language in RL domains
- ★ Looking beyond simple finite grid world domains
- ★ Complicated state and action space

Problem Statement:

Use of Deep Reinforcement algorithms to learn text-based state-action space and interact effectively in a natural language environment. The crux of the problem is to learn the language representation and take decisions based on the actions that optimize the overall returns. Since it is a gaming environment (text-based) the agent's goal is to finish the game with maximum rewards.

Text game is a sequential decision-making task with both input and output spaces given in natural language.

MDP Formulation (Text game):

Let us define a text game as a tuple $G = \{H, H_t, S, A, D, T, R\}$, where

- H is a set of game states, H_t is a set of terminating game states, $H_t \subseteq H$,
- S is a set of possible state descriptions,
- A is a set of possible action descriptions,
- D is a function generating text descriptions, $D : H \rightarrow (S \times 2^A)$,
- T is a transition function, $T : (H \times A) \rightarrow H$,
- R is a reward function, $R : (S_t, A_t, S_{t+1}) \rightarrow R$.

Input: sequences of text (game state space)
Learn: Q-value of the state-action pair - Q(s, a)
Output: Identify best action-object pair.

- **Research papers studied:**
  Language Understanding for Text-based Games using Deep Reinforcement Learning
  Deep Reinforcement Learning with a Natural Language Action Space
  Baselines for Reinforcement Learning in Text Games

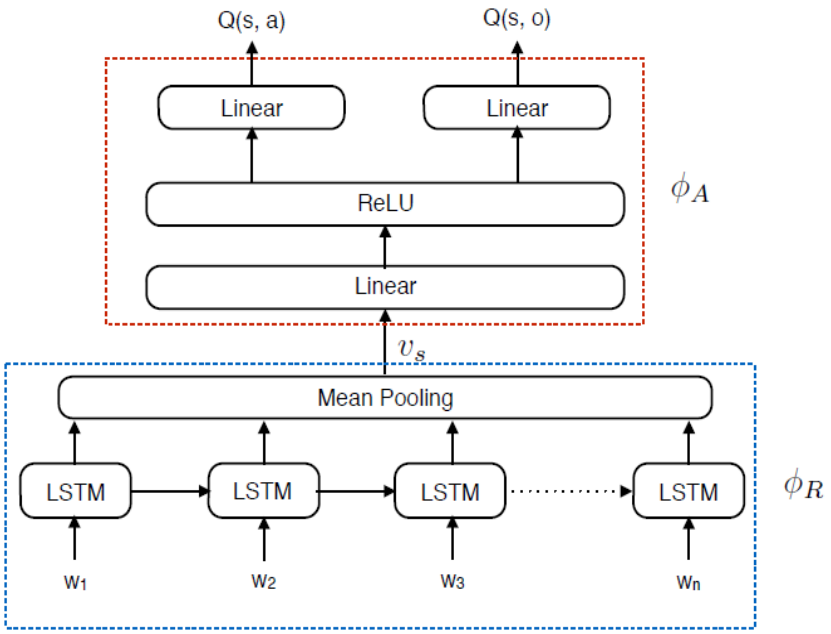
- **Reference** **Code:**
  SSAQN Implementation Code

- **Environments(Text based Games):**
  - **Saving John and Machine of Death:**
    - https://github.com/jvking/text-games
    - https://github.com/MikulasZelinka/text-games

- **Packages/Libraries/Platforms**
  Python, TensorFlow, PyFiction
  Local machines and Google CoLab GPU/TPU

Algorithms:

- **LSTM-DQN:**



The NN contains two modules:

- ★ One converts textual descriptions into vector representations that act as proxies for states (LSTM). Also called Representation Generator ($\varnothing_R$).
- ★ The second module of the network scores the actions given the vector representation computed by the first. Also called Action Generator ($\varnothing_A$).

The representations learnt can be reused across games, speeding up learning and faster convergence of Q-values.

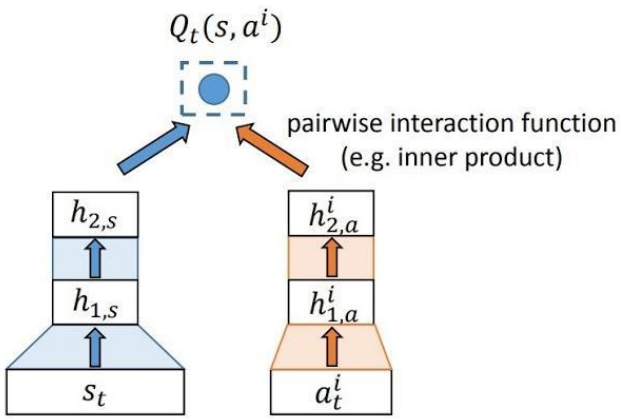
- **Representation Generator ($\varnothing_R$):**
  - ★ The LSTM network takes in word embeddings w_k from the words in a description s and produces output vectors x_k at each step.
  - ★ Mean pooling layer computes the element-wise mean over the output vectors x_k.

- **Action Scorer ($\varnothing_A$):**
  - ★ Input is v_s (hidden state space representation).
  - ★ Outputs are scores for actions - Q(s, a).
  - ★ Commands consist of one action (e.g. eat) and one argument object (e.g. apple).
  - ★ Predict both action and object for each state using the same network.
  - ★ Q(s, a, o) = Average of Q(s, a) and Q(s, o).

Algorithm 1 Training Procedure for DQN with prioritized sampling

Initialize experience memory $\mathcal{D}$
Initialize parameters of representation generator (ϕ_R) and action scorer (ϕ_A) randomly
for $episode = 1, M$ **do**
    Initialize game and get start state description s_1
    for $t = 1, T$ **do**
        Convert s_t (text) to representation v_{s_t} using ϕ_R
        if $random() < \epsilon$ **then**
            Select a random action a_t
        else
            Compute $Q(s_t, a)$ for all actions using $\phi_A(v_{s_t})$
            Select $a_t = \text{argmax}\ Q(s_t, a)$
        Execute action a_t and observe reward r_t and new state s_{t+1}
        Set priority $p_t = 1$ if $r_t > 0$, else $p_t = 0$
        Store transition $(s_t, a_t, r_t, s_{t+1}, p_t)$ in $\mathcal{D}$
        Sample random mini batch of transitions $(s_j, a_j, r_j, s_{j+1}, p_j)$ from $\mathcal{D}$, with fraction ρ having $p_j = 1$
        Set $y_j = \begin{cases} r_j & \text{if } s_{j+1} \text{ is terminal} \\ r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta) & \text{if } s_{j+1} \text{ is non-terminal} \end{cases}$
        Perform gradient descent step on the loss $\mathcal{L}(\theta) = (y_j - Q(s_j, a_j; \theta))^2$

- **Deep Reinforcement Relevance Network (DRRN):**



(c) DRRN

It is an improvement to LSTM-DQN architecture, for enhanced Natural Language Understanding which can handle continuous action spaces and stochastic environments. Following are the salient features of this network architecture:

★ Dual learning representations for state and action spaces.

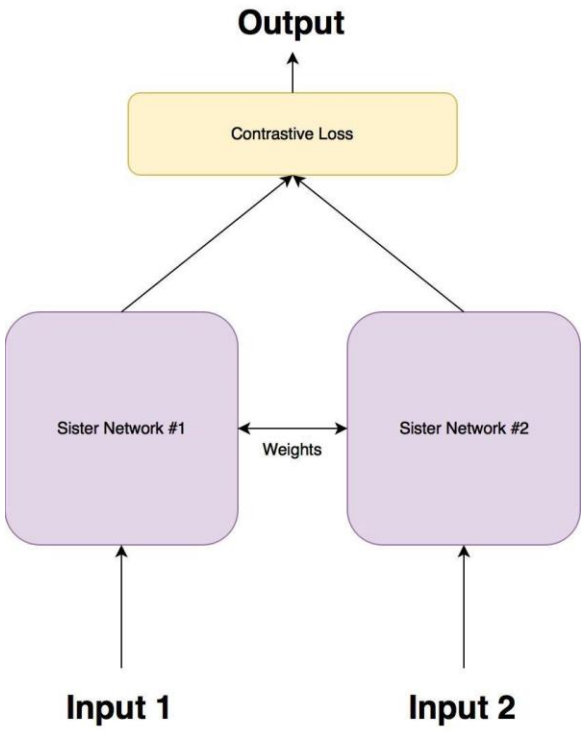
★ State and action space is unbounded.

★ Final Q-value is the inner product of inner state representations of respective vectors.

Algorithm 1 Learning algorithm for DRRN

Initialize replay memory $\mathcal{D}$ to capacity N.
Initialize DRRN with small random weights.
Initialize game simulator and load dictionary.
for $episode = 1, \ldots, M$ **do**
  Restart game simulator.
  Read raw state text and a list of action text from the simulator, and convert them to representation s_1 and $a_1^1, a_1^2, \ldots, a_1^{|\mathcal{A}_1|}$.
  for $t = 1, \ldots, T$ **do**
    Compute $Q(s_t, a_t^i; \Theta)$ for the list of actions using DRRN forward activation (Section 2.3).
    Select an action a_t based on probability distribution $\pi(a_t = a_t^i | s_t)$ (Equation 2)
    Execute action a_t in simulator
    Observe reward r_t. Read the next state text and the next list of action texts, and convert them to representation s_{t+1} and $a_{t+1}^1, a_{t+1}^2, \ldots, a_{t+1}^{|\mathcal{A}_{t+1}|}$.
    Store transition $(s_t, a_t, r_t, s_{t+1}, A_{t+1})$ in $\mathcal{D}$.
    Sample random mini batch of transitions $(s_k, a_k, r_k, s_{k+1}, A_{k+1})$ from $\mathcal{D}$.
    Set $y_k = \begin{cases} r_k & \text{if } s_{k+1} \text{ is terminal} \\ r_k + \gamma \max_{a' \in A_{k+1}} Q(s_{k+1}, a'; \Theta)) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_k - Q(s_k, a_k; \Theta))^2$ with respect to the network parameters Θ (Section 2.4). Back-propagation is performed only for a_k even though there are $|\mathcal{A}_k|$ actions at time k.
  end for
end for

- **Siamese State Action Q-Network (SSAQN):**



The final algorithm we explored is called a Siamese State Action Q-Network (SSAQN). The main objective of the algorithm was to create an agent that could generalize (transfer learning) to unseen text-games, consequently, playing multiple games at once. Moreover, the architecture is quite simpler and operates more efficiently than the DRRN architecture.

This neural network model employs a siamese architecture, where two branches - state and action branches - share both embedded and LSTM layers used for generating representational features. This means that the weights for both these layers are shared by state and action input data. Differentiation between the two branches only occurs in a dense layer prior to an interaction function. This model architecture provides the flexibility to provide arbitrary length actions (provided in DRRN but not in LSTM-DRL).

In more detail - the first layer in the architecture deals with preprocessing and tokenization of both the state and action branches and is then converted from words to their vector representation using word embeddings. These word embeddings are then passed through an LSTM layer where the sequence based features of the input are recognized and the inner state of each gated cell is updated. Within this layer, inputs of arbitrary lengths are bounded to a fixed vector size of the length of LSTM units. Following the shared embedded and LSTM layer, we now have two dense layers - one for states and one for actions. Lastly, we then apply a cosine similarity interaction function to the state and action dense activations, resulting in the final Q-value. This Q-value is then utilized in back propagation where the loss function is defined as simply the mean squared error (MSE) of the last estimated Q-value and the target Q-value and gradient descent is performed with an RMSProp optimizer.

In essence, this algorithm has three key takeaways that separates it from the previous two; first the network accepts two text descriptions (state and action) of arbitrary length - this was not provided in LSTM-DRL. This siamese architecture means that the embedding and LSTM layers are the same for both state and action input - their weights are shared. Lastly, the interaction function of inner vectors of state and action is realised by a normalized dot product, commonly referred to as cosine similarity.

In this paper, there are four distinct testing scenarios the agent was run against. The first was simply training and testing the agent against a single text-based game. The second scenario was done to see if transfer of learning was effective by pre-training the agent against five different games and testing it on its expected game. The last scenario dealt with playing multiple games at once - essentially each agent was trained and evaluated on a single instance on all six games at once. In each training step, all the games were presented successively to the agent.

Problem Inspiration/Related Work:

There was an implementation of SSAQN available on Github. We used the code as a baseline to implement the algorithm on SJ and MoD. We modified the NN architecture in the implementation and observed the results. We also carried out various hyperparameter tuning on the algorithm. We have plotted a few of the noticeable ones as results.

Challenges:

- ★ We had to research a lot to identify research papers which address the issue of learning control policies on text-based strategy games.
- ★ We faced challenges to successfully interact with the environment (SJ and MoD). However, then we came across the SSAQN implementation and it helped us interact with the SJ and MoD environments.
- ★ Making changes to the available SSAQN implementation was challenging.
- ★ Several hours of training time for the text-based games (especially MoD).
- ★ Analysis of the state/action embedding proved to be challenging.

Results:

We executed the code and also plotted the learning results of the SSAQN agent on SJ and MoD to better understand the learning curve (Rewards vs Episodes).

- ★ As inferred by the paper, we also noticed SJ achieving peak performance in fewer episodes. The learning curve has a similar learning pattern to the results shown in the paper.
- ★ We found more variance in the Machine of Death than suggested by the paper. However, the general trend in the curves remains the same.
- ★ SJ leans to reach the goal state in fewer steps.
- ★ State/action representation gets better after every episode.

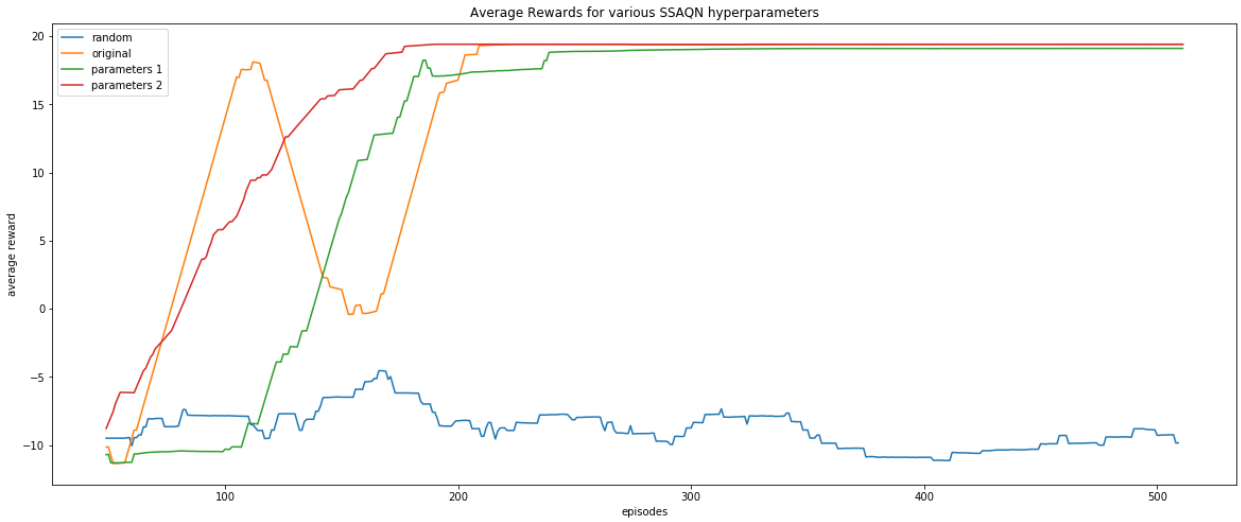


Figure 1: Hyperparameters average reward for SSAQN algorithm running on machine of saving john simulation.

	Original Parameter	Parameters 1	Parameters 2
Embedding Dimensions	16	16	32
LSTM Dimensions	32	32	64
Batch Size	64	32	64
Gamma	0.95	0.9	0.95
Epsilon Decay	0.99	0.9	0.99
Dense Dimensions	8	8	8
Prioritized Fraction	0.25	0.25	0.25

Table 1 : Hyperparameters used for SSAQN for Saving John Simulator.

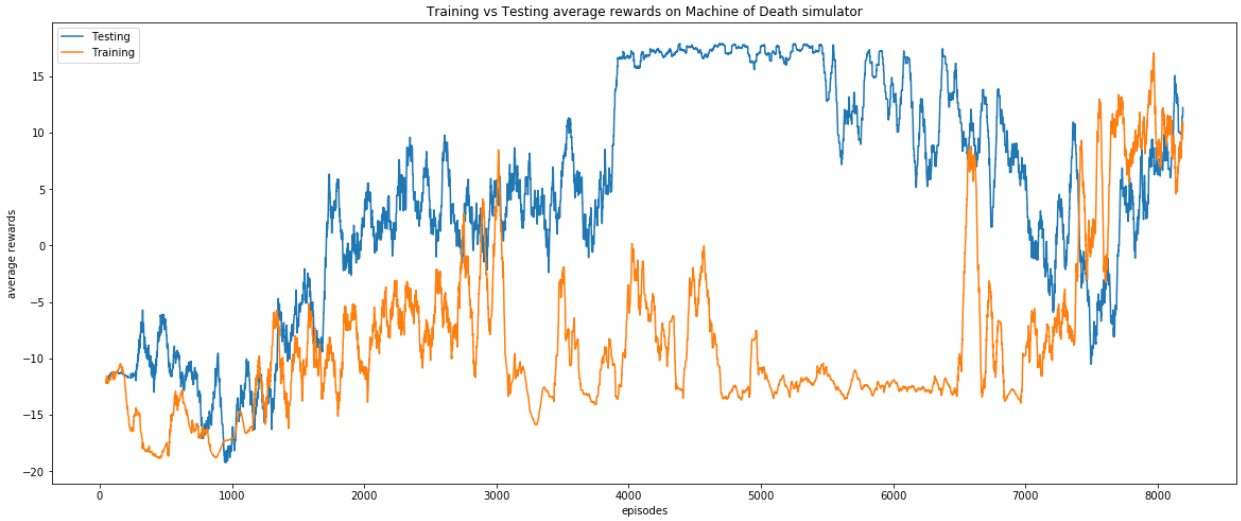


Figure 2 : Average reward for training and testing run of SSAQN.

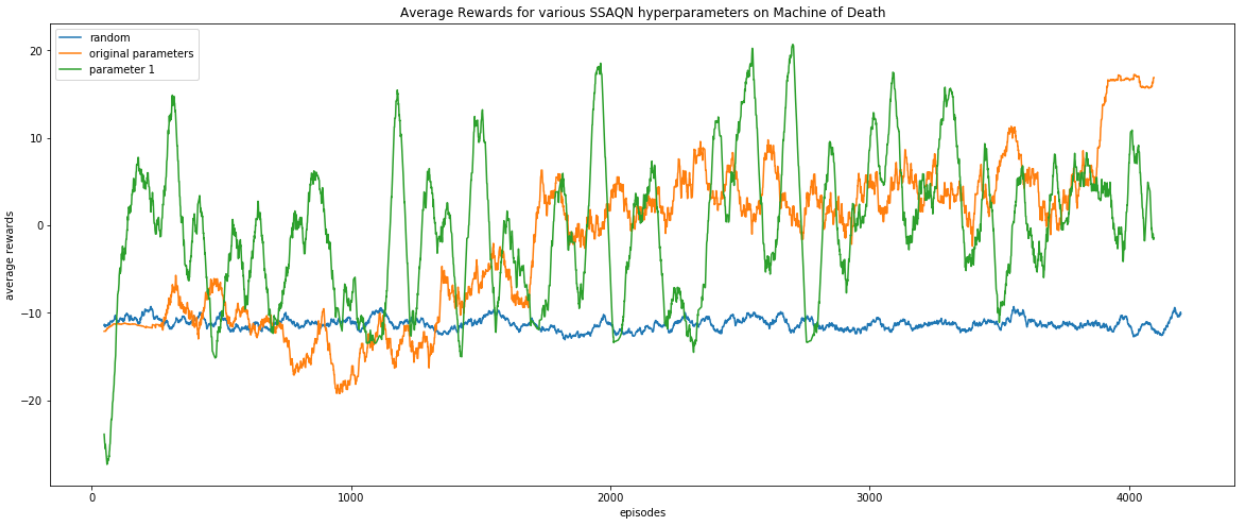


Figure 3 : Hyperparameters average reward for SSAQN algorithm running on machine of death simulation

	Original Parameters	Parameters 1
Embedding Dimensions	16	32
LSTM Dimensions	32	64
Batch Size	64	64
Gamma	0.95	0.95
Epsilon Decay	0.99	0.99
Dense Dimensions	8	8
Prioritized Fraction	0.25	0.25

Table 2: Hyperparameters used for SSAQN for Machine of Death Simulator.

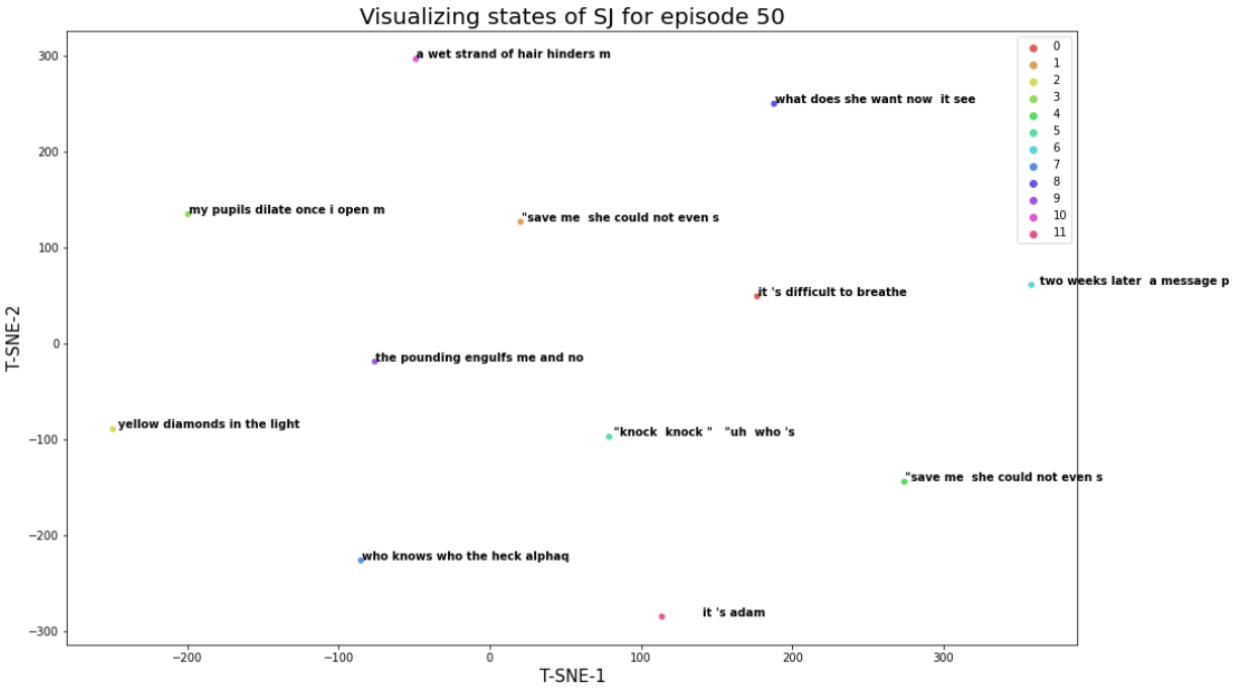


Figure 4 : Visualizing state representation using T-SNE plots for SJ for episode 50

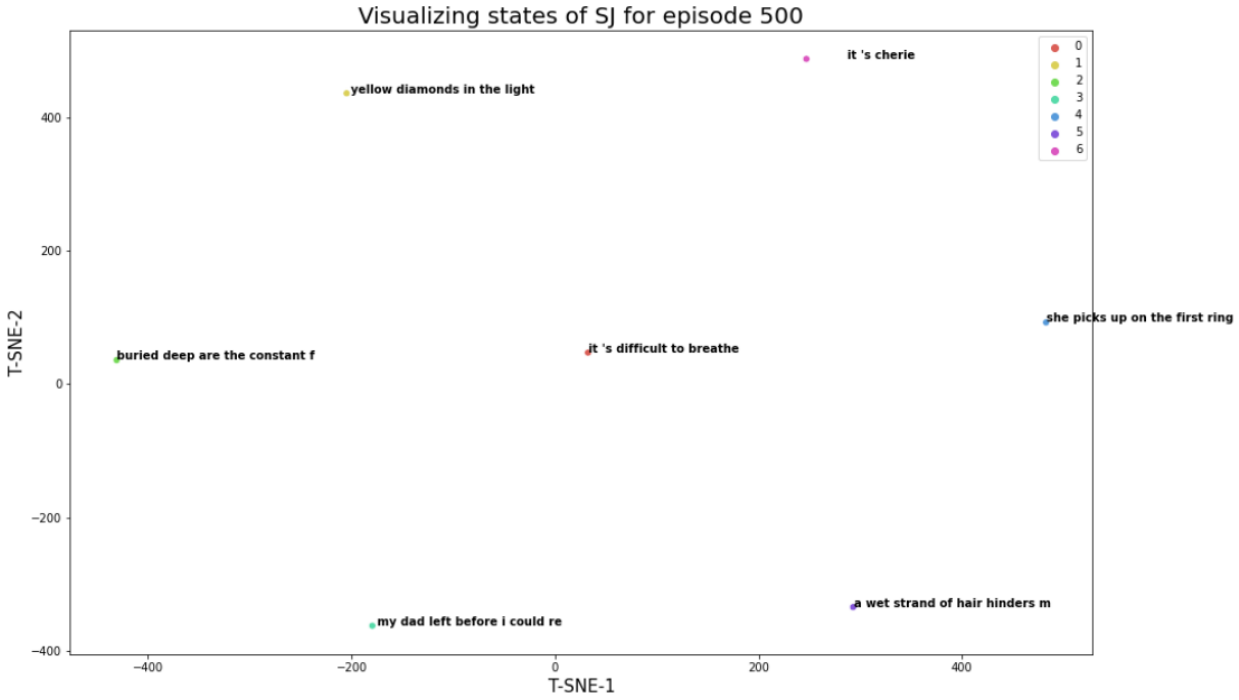


Figure 5 : Visualizing state representation using T-SNE plots for SJ for episode 500

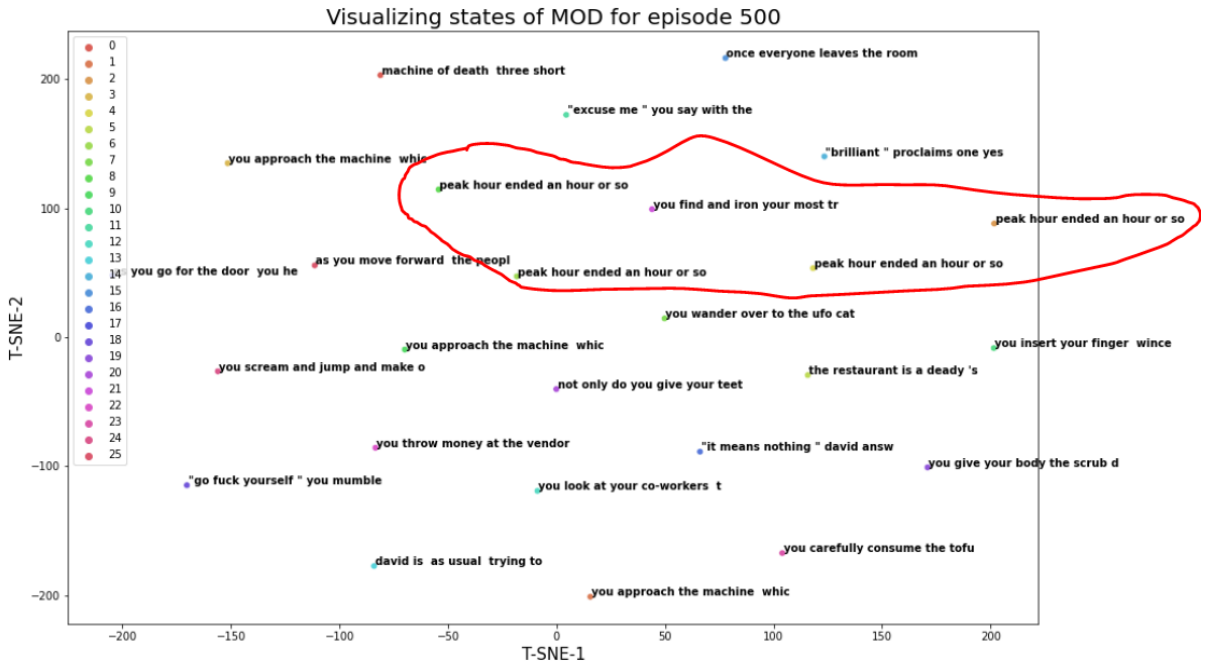


Figure 6 : Visualizing state representation using T-SNE plots for MOD for episode 500

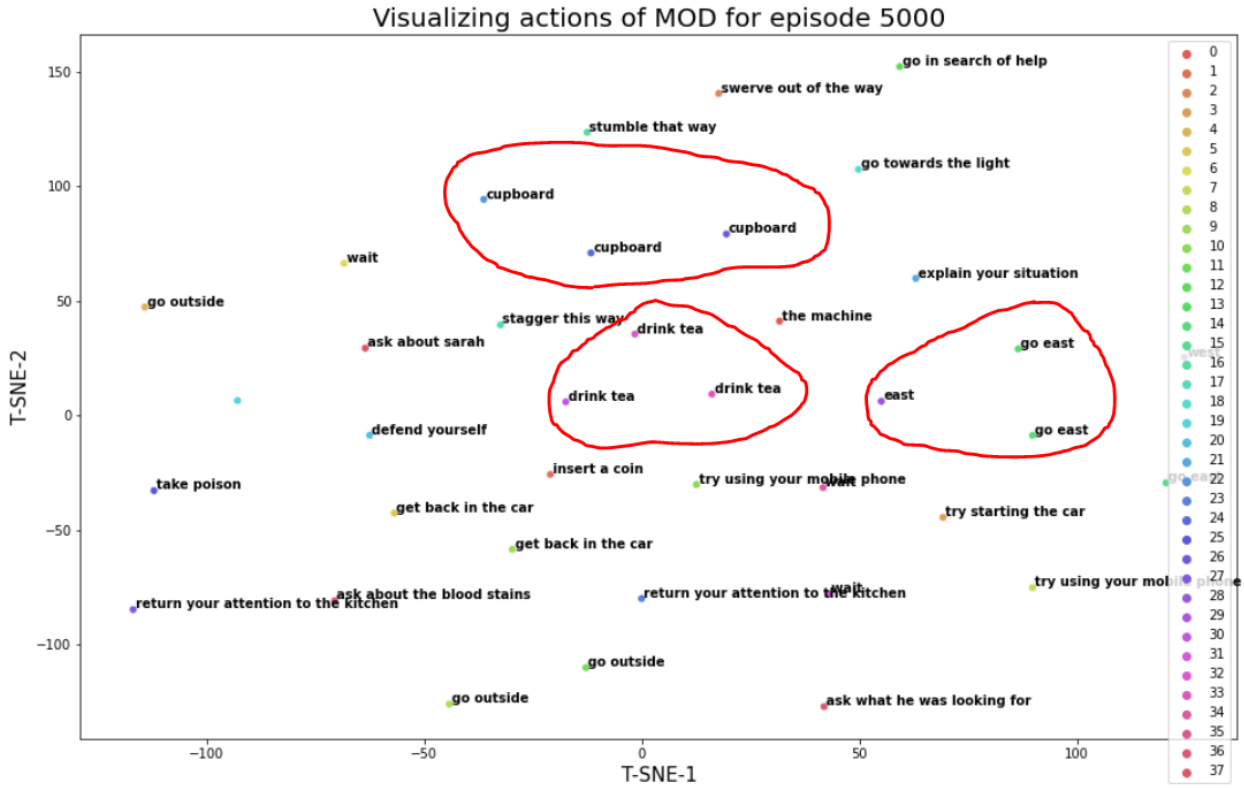


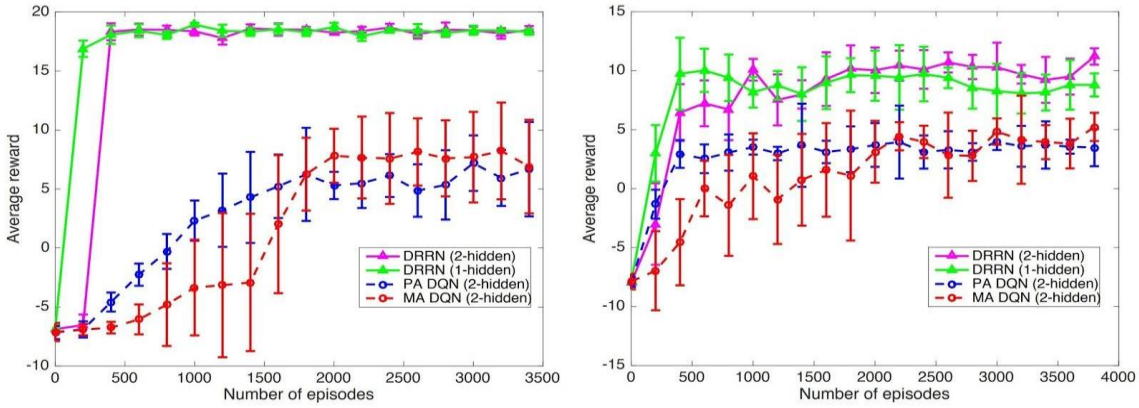
Figure 7 : Visualizing state representation using T-SNE plots for MOD for episode 5000

Analysis:

The difference between LSTM-DQN approach and the Siamese network approach is the action space length. The former uses a fixed set action space whereas the latter is used to expand the scope to a continuous horizon. Hence, the paper from Microsoft concluded with their DRR network performing better than a conventional DQN.

The algorithms discussed in this project are of increasing generalizability. That is,

- ★ The Random agent follows a random policy
- ★ The Bag of Words-DQN agent uses primitive state features and computes the Q-values of a state-action pair
- ★ The LSTM-DQN agent uses better state representations but limited action space and computes the Q-values of a state-action pair
- ★ The DRRN agent uses better state and action representations from separate networks to compute them, applicable to continuous action space and computes the Q-values of a state-action pair
- ★ The SSAQN agent uses better state and action representations from same shared network to compute them, applicable to continuous action space and computes the Q-values of a state-action pair



(a) Game 1: "Saving John" (b) Game 2: "Machine of Death"

Figure 4: Learning curves of the two text games.

Moreover, our results indicate that SSAQN does indeed seem to perform at least as well DRRN and outperform LSTM-DQN ; the Saving John results show convergence at around 100 episodes, which is comparable to DRRN from figure 4. Similarly, for machine of death, SSAQN performed at least as well as DRRN from 2000 episodes - 6500 episodes. It does seem that SSAQN has higher variance than DRRN, as shown in simulating it on Machine of Death. This higher variance, we may be the reason for the sudden drop from 6500 episodes to 7500 episodes.

The results from modifying the embedding and lstm layer dimensions from 16 and 32 respectively to 32 and 64 respectively indicate that a higher dimension was able to better represent these features, leading to a more efficient convergence. This makes sense because in DRRN the embedding and lstm layers for state-action spaces are separated, meaning learning for each space occurs separately. In SSAQN, however,

the weights for both embedded and lstm are shared and utilized by both state and actions, suggesting that these layers have to represent information about both state and actions - not simply state or action. A higher dimensional lstm and embedding layer would allow for this to happen, hence, a convergence more efficient than its predecessor.

The SJ game is more deterministic. From figure 4 and 5 we can observe that the number of steps taken to arrive at the goal for SJ in episode 50 is higher compared to the steps taken to arrive at the goal in episode 500. Since the number of steps are very few (less than 15) there is nothing much to infer from the state embedding visualization using T-SNE plots.

The MoD game is stochastic, and fewer steps is not necessarily better. Analysis of MoD is much harder since the states the agent lands at various episodes greatly differ at each episode. However, from figure 6 and 7 we can infer that the similar state/action embedding gets better after each episode. In figure 6, similar states are not grouped together because at episode 500 the agent hasn't been able to learn good representations. But, at episode 5000(figure 7), similar states/actions are grouped together. (east and go east are grouped together)

Conclusion:

The results denote the effectiveness of learning the state-action representation is the key to human-like performance (or beyond) and also helps in achieving generalization over multiple games. We were able to improve the performance by tuning the hyper-parameters, which is evident to the fact that there is still room for improvement in the selection of hyper-parameters.

It is still a challenge to visualize the progress in learning the state and actions representations because direct comparison between different episodes is not feasible due to the large state space and the stochastic nature of the games .

Future Work:

We can possibly extend the implementation of SSAQN:

- To Accommodate Bi-directional LSTM, Attention architecture for learning state-action representations.
- Changing the interaction function to perform some complex non-linear mappings (using another DNN) to estimate the Q-values.
- Using Evolutionary algorithms to boost effective search of hyper-parameters.