CS F317: Reinforcement Learning - Course Project

A Simple Deep Reinforcement Learning Dialogue System

Group RL14

December 1, 2021

1 Motivation

Dialogue agents have a wide range of applications in the real world. Current dialogue generation models based on neural networks tend to be shortsighted, with responses that ignore influence on future outcomes. Applying deep reinforcement learning techniques facilitates the modelling of future rewards in dialogue generation, making this an area of interest.

2 Introduction

The selected research paper is **SimpleDS: A Simple Deep Reinforcement Learning Dialogue System, authored by Heriberto Cuayahuitl** [1]. This paper is implemented in the restaurant domain. It performs action selection directly from raw text of the last system along with noisy user responses. The major problem with an RL-based dialogue system is that all the elements, including dialogue state, action, rewards, or penalties, have to engineered. This paper aims to increase the degree of automation in feature engineering through the use of Deep Reinforcement Learning.

3 Implementation Details

Reinforcement learning is more unstable when neural networks are used to represent the action values. Training such a network requires a lot of data, but it is not guaranteed to converge on the optimal value function. There are situations where the network weights can oscillate or diverge due to the high correlation between actions and states. To solve this problem, we use Experience Replay.

3.1 Algorithm: Deep Q-Learning with Experience Replay

Experience Replay is a replay memory technique used in reinforcement learning where we store the agent's experiences at each time step, pooled over many episodes into a replay memory.
The experiences:
$e_t = (s_t, a_t, r_t, s_t + 1)]$
are pooled into the data-set
$D = e_1, ..., e_n$

```
Initialize network Q
Initialize target network Q^
Initialize experience replay memory D
Initialize the Agent to interact with the Environment
while not converged do

      /* Sample phase
      Є ← setting new epsilon with Є-decay
      Choose an action a from state s using policy Є-greedy(Q)
      Agent takes action a, observe reward r, and next state s'
      Store transition (s, a,r, s', done) in the experience replay memory D

            if enough experience in D then
            /* Learn phase
            Sample a random minibatch of N transitions from D
            for every transition ($s_i$,$a_i$,$7_i$, $s'_i$,$done_i$) in minibatch do
                  if $done_i$ then
                     $y_i = r_i$
                  Else
                     $y_i = r_i + \gamma \max_{a' \in A} Q\hat{}(s'_i, a')$
                  End
            End
      Calculate the loss £ =1/N ∑(for i=0 till i=N-1)($(Q(s_i,a_i)-y_i)^2$
      Update Q using the SGD algorithm by minimizing the loss of £
      Every C steps, copy weights from Q to Q^
      End
end
```

Figure 1: Deep Q-Learning with Experience Replay

During the inner loop of the algorithm, we usually sample the memory randomly for a minibatch of experience and use this to learn off-policy, or we apply Q-learning updates. This solves the issue of autocorrelation leading to unstable training by making the problem more like a supervised learning problem. After performing experience replay, the agent selects and executes an action according to an -greedy policy. Since using histories of arbitrary length as inputs to a neural network can be difficult, our Q-function instead works on a fixed-length representation of histories produced by a function ϕ [2].
At the end of every iteration of the while loop, it

- Calculates the loss

$$\pounds = \sum_{i=0}^{n-1} (Q(S_i, a_i) - y_i)^2$$

- Updates Q using the SGD algorithm by minimizing the loss of £
- Every C steps, copies the weights from Q to the target network, Q*

3.2 Environment Details: States, Actions, Rewards

States: 100 word-based features in English, Spanish, and German used in a Restaurant domain. States are numerical representations of the last system and noisy user inputs.

Action Space: Actions are dialogue acts, 35 dialogue acts over a restaurant domain (for all three languages) which includes : i) 2 salutations, ii) 7 apologies, iii) 9 requests, iv) 7 explicit confirmations, v) 7 implicit confirmations, vi) 1 retrieve information, vii) 2 provide information.
Slot Values were defined for easier understanding

- Food : Italian, Indian, Mexican, Chinese, Japanese
- Price : Cheap, Expensive
- Area : North, South, East, West, Center

Transition Function: The state transition function is based on a numerical vector pointing to the last system and user responses. The system responses are 0 if absent and one if present. The latter corresponds to the confidence level [0..1] of user responses. SimpleDS targets an extensible and straightforward dialogue system, a template for language generation, and confidence scores generated uniformly at random (words with scores under a threshold were distorted).

Rewards: The rewards system is based on the fact that the more human-like the conversation, the better the rewards. The reward function is defined by:

$$R(s, a, s0) = (CR \times w) + (DR \times (1 - w)) - DL$$

where CR is the number of positively confirmed slots divided by the slots to ensure.

W is weight over the confirmation reward (CR). We used w=0.5; DR is a data-like probability of having observed action a in state s, and DL is used to encourage efficient interactions. We used DL=0.1. The DR scores are derived from the same statistical classifier above, which allows us to make statistical inferences over actions given states (Pr(a|s)).

3.3 Tools and Frameworks Used

Client-Server Architecture: SimpleDS is a computational framework that aims at using deep reinforcement learning in the training of dialogue systems. In particular, SimpleDS selects dialogue from raw text. The system makes use of Java and JavaScript to run under a client-server architecture. The learning agent (in JavaScript) acts as the client, and the environment (in Java) as the server.

ConvNetJS: The learning agent is built on ConvNetJS [3], a Javascript library used to train deep learning models. The library is an implementation of neural networks. It supports common neural network modules, different cost functions, and image processing convolutional networks. In the current context, we use the ConvNetJS tool to implement the Deep Q-Learning with Experience Replay algorithm.

4 Results of Original Implementation

In the original paper, the model is trained using 3000 simulated dialogues, and the learning parameters used are as follows:

- Experience Replay Size: 1,000
- Discount factor: 0.7
- Minimum Epsilon: 0.001
- Batch Size: 32
- Burning Steps: 100
- Optimizing Algorithm: Stochastic Gradient Descent

Using this configuration, we train three separate models for the three language datasets used in the paper, namely, English, German and Spanish. During the training, the intermediate results, including the average reward, average actions used and epsilon, is outputted every 100 steps.
Based on these results, the policy is updated accordingly for the remaining steps. Therefore, each language's final results are the learnt interaction policy and the logged performance for every 100 steps. The performance of these models with respect to their average reward over the number of experiences during this phase is shown in Figures 2-4.

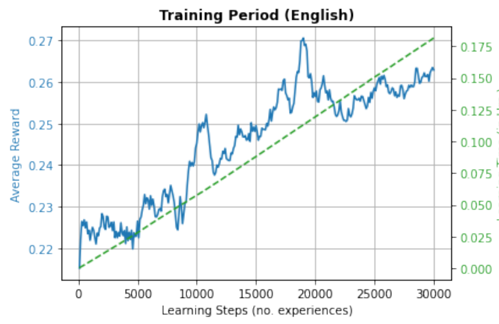


Figure 2: Avg. Reward over Time for Training Period (English)

Figure 2 highlights the change in average reward over the number of experiences during the training period for the agent when trained on simulated dialogues in English. The average reward shows an upward trend as the number of experiences increases during this period, starting from 0.22 and ending close to 0.26.
Figure 3 and Figure 4 are similarly made for when the agent is trained on German and Spanish dialogues, respectively instead. In the case of all three languages, a clear upward trend in the average reward is seen during the training period as the number of experiences increases indicating that the agent is 'learning' to maximise the reward by successfully interacting with the user with a minimum number of actions.
Once the trained models are obtained, we can test the models based on more user simulated dialogues. Here, we use 300 user-simulated dialogues during the test period. During the test period the model will be using

i)	2 salutations	Hello,
ii)	7 apologies	Sorry, I didn't get the food type.,
iii)	9 requests	How can I help you?,
iv)	7 explicit confirmations	Did you say *food food?,
v)	7 implicit confirmations,	Okay, *price food in the *area.
vi)	1 retrieve information,	Let me see.
vii)	2 provide information	Restaurant X is an excellent choice. It is located in Y.

Table 1: Action Spaces

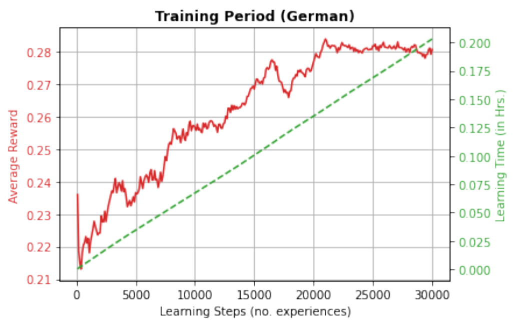


Figure 3: Avg. Reward over Time for Training Period (German)

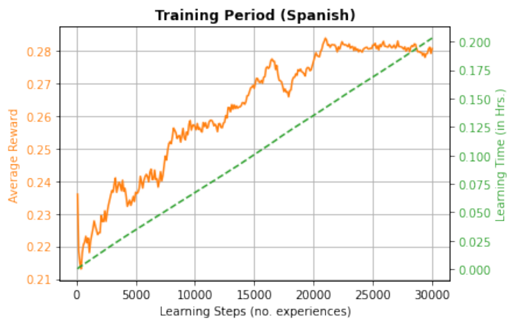


Figure 4: Avg. Reward over Time for Training Period (Spanish)

the policy learnt during the training period. Figures 5-7 illustrate the variation of average reward (every 100 steps) over the number of experiences during the test period.
The most prominent feature of these graphs is that the average reward starts at a higher point at the beginning of the period than it did during training and this is presumably a result of the learnt interaction policy during training. In this case, the average reward of the agents trained on English and Spanish dialogues do not show any trend during the test period while the average reward for the agent trained on German dialogues shows an upward trend, rising from an average reward of 0.26 to 0.27 over the 3000 steps.

5 Proposed Improvements

5.1 Word Embedding

Our dialogue agent needs to be as natural as possible, and adapt to the intricacies of natural language. For

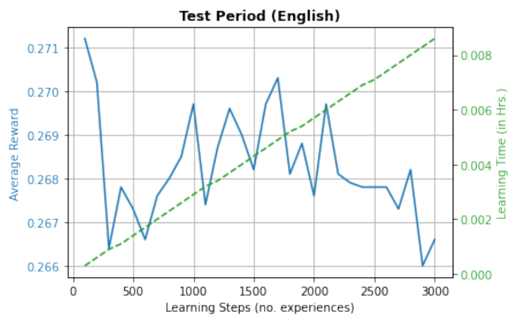


Figure 5: Avg. Reward over Time for Test Period (English)

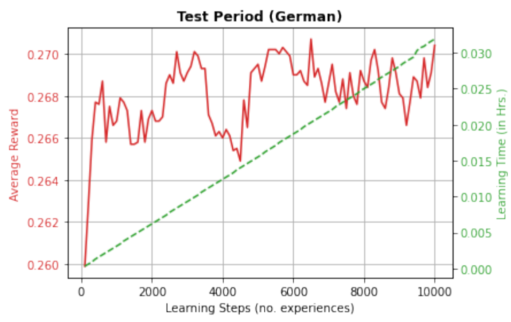


Figure 6: Avg. Reward over Time for Test Period (German)

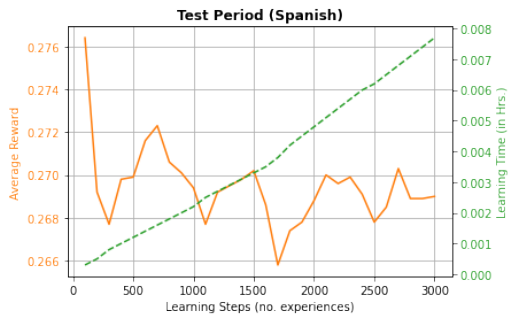


Figure 7: Avg. Reward over Time for Test Period (Spanish)

example, instead of asking for "I want Indian food", the user could type "I fancy Indian food". But as 'fancy' is not in the vocabulary of the agent (81 words manually entered), it won't work properly. We propose to solve this problem using word embedding, which essentially aims to deal with synonyms [4]. Word Embedding is representation of words as vectors in an n-dimensional space such that words with similar meanings, frequently used words, etc. are closer together. So when, someone types words with similar meaning in a given context, the agent is still able to correctly identify the variables. Pre-trained word vectors from the Stanford GloVe project are used for this purpose [5].

5.2 Discount Factor and Minimum Epsilon

The performance of an RL agent is often summarized by the total discounted reward obtained, given by:
$R = R_0 + \gamma R_1 + \gamma^2 R_2 + ...$
The use of discounting ensures that the total return is always convergent. It is therefore a valid measure regardless of the limiting behavior of the system (periodic, recurrent, absorbing, etc.). Moreover, the discounting mechanism formalizes the intuitive notion that, other things being equal, it is preferable to get the (positive) rewards as quickly as possible and to delay the punishment (the negative rewards) as much as possible. The objective of the learning agent is to determine a policy that maximizes the expected total discounted reward.
The discount factor, γ is a real value $\in$ [0, 1], cares for the rewards agent achieved in the past, present, and future. In different words, it relates the rewards to the time domain. If γ= 0, the agent cares for the first reward only. If γ = 1, the agent cares for all future rewards.
In reinforcement learning, the agent tries to discover its environment. The concept of exploiting what the agent already knows versus exploring a random action is called the exploration-exploitation trade-off.
Epsilon (ϵ) parameter is the ratio for this exploration-exploitation trade-off. The epsilon parameter introduces randomness, forcing trials of different actions. This helps in not getting stuck in a local optimum.
If $\epsilon = 0$, we never explore but always exploit the knowledge we already have whereas $\epsilon = 1$ forces the algorithm to always take random actions and never use past knowledge.
We propose to vary the values of the Minimum Epsilon (ϵ) and Discount Factor (γ) used to find the parameters that will maximise the reward.

5.3 Hyperparameter Optimization

Another improvement we propose is related to hyperparameter optimization of the neural network attributes. Optimizers allow us to alter these attributes (such as weights, learning rate) in an attempt to increase the accuracy of the model and reduce losses. These optimizers typically aim to minimize the cost function associated with a neural network.
The parameters of the neural network that the paper implements have been optimized using the Stochastic Gradient Descent algorithm. In SGD optimizers, the first derivative of the cost function is computed by taking one point at a time. While it is computationally faster than other methods, its frequent updates make it noisy, and it may also take longer to converge to a minimum of the cost function. Moreover, a learning rate needs to be chosen that remains constant throughout the parameter updates, and making the correct choice of learning rate may be difficult.
As a result, we propose the use of some newer optimization techniques that may be able to overcome the disadvantages of the Stochastic Gradient Descent algorithm. In particular, we propose the use of optimizers that do not use constant learning rates. Such optimizers, like AdaGrad (Adaptive Gradient Descent) [6] and AdaDelta [7], change the learning rate adaptively with iterations, eliminating the need to monitor and manually tune the learning rate, as would need to be done when using SGD for optimization.

6 Experiments

6.1 Word Embedding

The average reward over time of the agent when trained on the pre-defined data from manually entered vocabulary is shown in Figure 8. In this case, we find that word embeddings does not provide a significant improvement in average reward. However, as shown in Figure A3 and A4 in the appendix, when trying with words out of vocabulary (manually entered list of 81 words), Word embedding-disabled model takes random values, whereas with word-embedding enabled, the agent is able to interpret words outside the pre-defined vocabulary. For example, when the user asks for Sicilian food (not defined in the vocabulary), the agent that was trained without word embeddings takes random values for the kind of food, but the one trained on word embeddings is able to match Sicilian to Italian food.

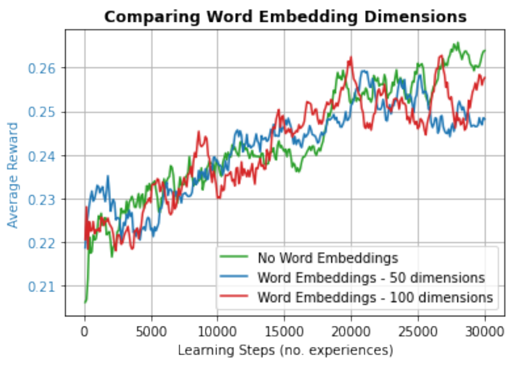


Figure 8: Comparison of Avg. Reward over Time for Training Period with and without Word Embedding

6.2 Discount Factor and Minimum Epsilon

We train the models for the English language data set using seven different values of Discount Factor (γ). The intermediate results are outputted every 100 steps, and the policy is updated accordingly. Figure 9 show the models' average reward over the number of experiences for three values of γ

There was a clear upward trend in performance of the model when γ=0.7, but the highest average reward is obtained when γ=0.1. For all the values, a sharp increase is observed in the average reward after 20,000 steps. A possible reason could be over fitting. Overall, γ=0.05 and γ=0.1 seem to have a higher average reward than higher values of γ. The maximum average reward is greater than 0.375 for γ=0.05 and γ=0.1, compared to the highest value of 0.26 for γ=0.7.

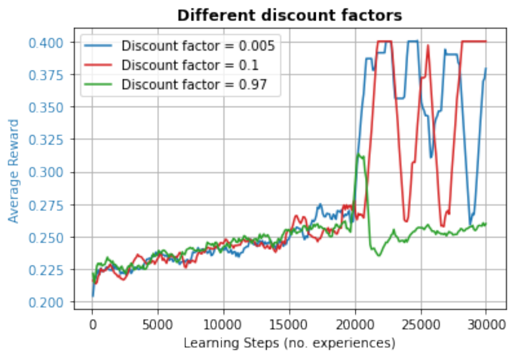


Figure 9: Comparison of Avg. Reward over Time for Training Period with Variation in Discount Factor

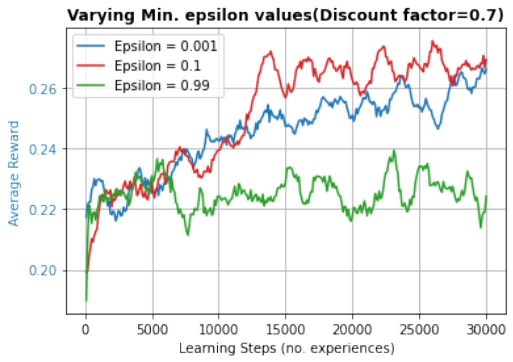


Figure 10: Comparison of Avg. Reward over Time for Training Period with Variation in Minimum Epsilon Values

Again, we test the models based on 300 user-simulated dialogues, this time varying minimum epsilon values. The results are observed in Figure 10. As the minimum ϵ approaches 0, we observe an upward trend in performance, and higher average rewards than those for higher values of the minimum ϵ. The highest average reward is greater than 0.26, and is achieved when the minimum ϵ=0.1.

6.3 Hyperparameter Optimization

We train the models for the English language dataset using three different optimizers - Stochastic Gradient Descent (SGD), Adaptive Gradient (AdaGrad), and AdaDelta. The training is done on 3000 user-simulated dialogues and similar to the training of the original model, the intermediate results are outputted every 100 steps, and the policy is updated accordingly.

Figure 11 shows comparisons of the models' average reward over the number of experiences for the three optimization algorithms in the training period. The average reward in the training period is the lowest in the case of SGD, higher for AdaGrad, and the highest for AdaDelta, after a sharp rise at around 20,000 learning steps. The maximum average reward is greater than 0.26 for SGD, greater than 0.27 for AdaGrad, and greater than 0.40 for AdaDelta.

Figure 12 depicts the average rewards in the testing period for the three optimizing algorithms. The average rewards for both AdaDelta and AdaGrad optimizers are significantly higher than that of the SGD optimizer. The average reward also follows a slight upward trend in the case of AdaGrad. For the AdaDelta optimizer, the average reward rises quickly and remains constant at 0.425 continually after a certain number of learning steps. The highest values for the average rewards for SGD, AdaGrad, and AdaDelta optimizers are 0.2375, 0.2685, and 0.4250, respectively.

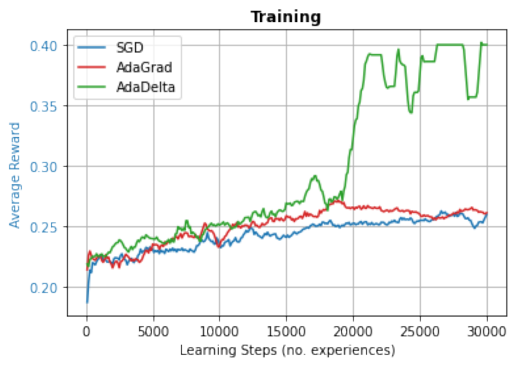


Figure 11: Comparison of Optimizing Algorithms for Agent Training

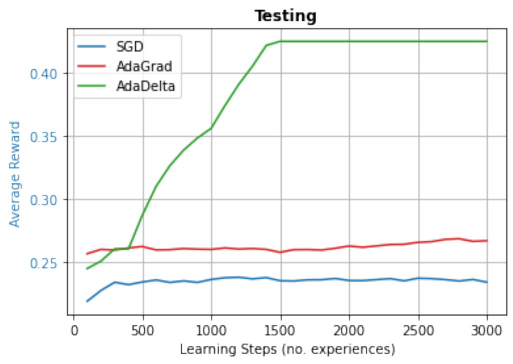


Figure 12: Comparison of Optimizing Algorithms for Agent Testing

7 Results of Proposed Improvement

The final model that we arrive at has the following parameters:

- Experience Replay Size = 1,000
- Discount factor = 0.1
- Minimum Epsilon = 0.1
- Batch Size = 32
- Burning Steps = 100
- Optimizing Algorithm = Adaptive Gradient Algorithm

We choose the discount factor to be 0.1, owing to the overall higher average reward observed in the experiments. For similar reasons, we opt to use a minimum epsilon value of 0.1 in our final model. The optimizing algorithm used is the Adaptive Gradient Algorithm (AdaGrad). Although the AdaDelta optimizer gave a higher average reward during training and testing, the average reward showed a sharp rise early in the testing phase and remained constant at 0.4250, running infinitely. This could indicate overtraining of the neural network, and for this reason, we choose to make use of the AdaGrad optimizer in our final model. The performance of the final model in the training and testing phases is depicted Figure 13 and Figure 14.

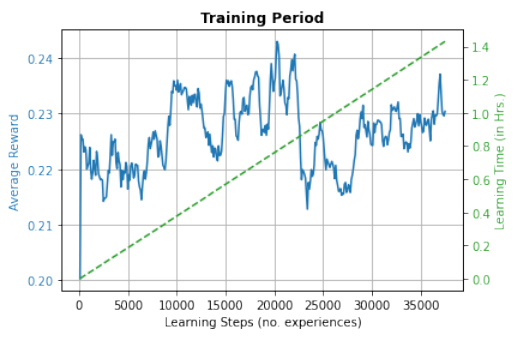


Figure 13: Avg. Reward over Time for Training Period

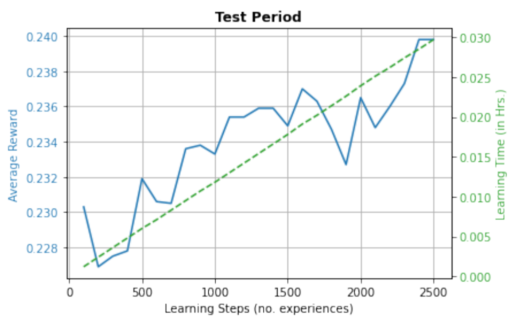


Figure 14: Avg. Reward over Time for Testing Period

8 Major Findings & Conclusions

The objective of this work was to explore the extent of automation possible in creating RL agents and subsequently evaluate the performance of these agents. Furthermore, the addition of word embeddings was able to improve the usability of the application by providing support for a greater range of natural language. By experimenting with other parameters such as discount factor and the optimizing algorithm, the average reward of the agent could be further maximised. Overall, we feel that there is great scope for the automation of training of RL agents across domains.

9 Work Distribution

Complete Team:

- Devanshi Gupta - 2019A7PS1265H
- Grahithaa Sarathy - 2018B3A70895H
- Saandra Nandakumar - 2018B3A70843H
- Saloni Singh - 2019A7PS1261H
- Zaeem Ansari - 2019A7PS0057H

Task	Done By
Ideation and selection of paper	Complete Team
Original Paper Implementation	Saandra Nandakumar
Word Embeddings	Zaeem Ansari
Discount Factor and Min. Epsilon	Saloni Singh
Hyperparameter Optimization	Grahithaa Sarathy
Final Model Implementation	Devanshi Gupta
Report Writing and Formatting	Complete Team

References

[1] Heriberto Cuayáhuitl. Simpleds: A simple deep reinforcement learning dialogue system. In *Dialogues with social robots*, pages 109–118. Springer, 2017.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[3] Andrej Karpathy. Convnetjs: Deep learning in your browser (2014). *URL http://cs. stanford. edu/people/karpathy/convnetjs*, 5, 2014.

[4] Heriberto Cuayáhuitl, Seunghak Yu, Ashley Williamson, and Jacob Carse. Deep reinforcement learning for multi-domain dialogue systems. *arXiv preprint arXiv:1611.08675*, 2016.

[5] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[6] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

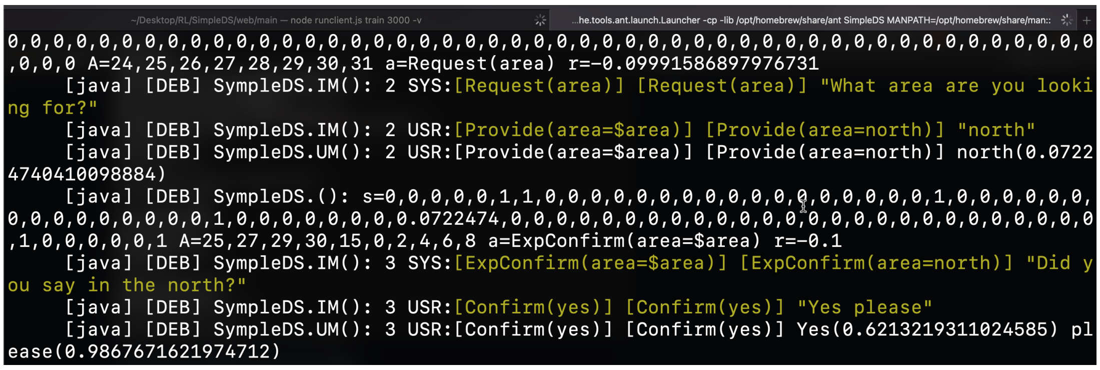
[7] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

10 Appendix

10.1 Code

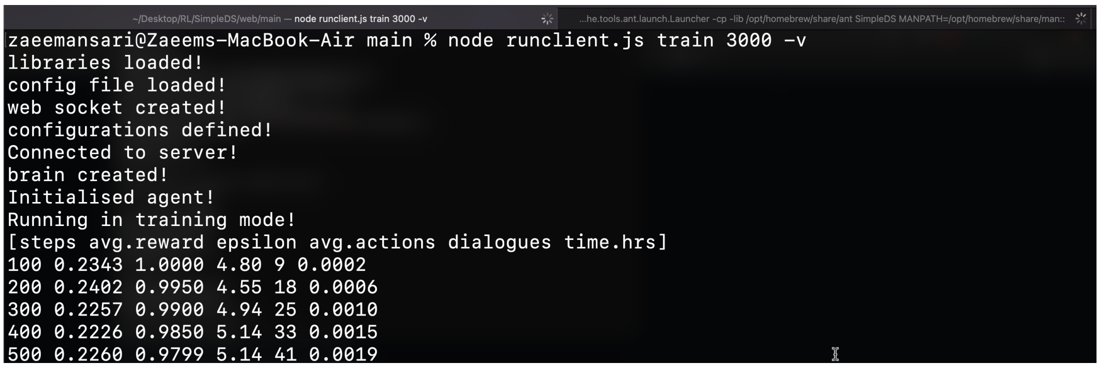
The code used in this project can be found at https://github.com/saandra02/DRL-dialogue

10.2 Selected Screenshots


~/Desktop/RL/SimpleDS/web/main — node runclient.js train 3000 -v
...he.tools.ant.launch.Launcher -cp -lib /opt/homebrew/share/ant SimpleDS MANPATH=/opt/homebrew/share/man::

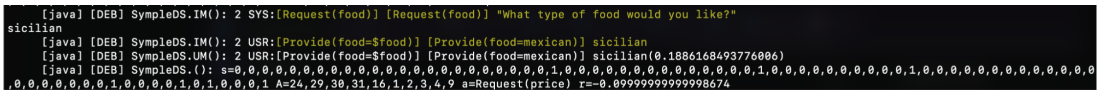
```
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0 A=24,25,26,27,28,29,30,31 a=Request(area) r=-0.09991586897976731
     [java] [DEB] SympleDS.IM(): 2 SYS:[Request(area)] [Request(area)] "What area are you looki
ng for?"
     [java] [DEB] SympleDS.IM(): 2 USR:[Provide(area=$area)] [Provide(area=north)] "north"
     [java] [DEB] SympleDS.UM(): 2 USR:[Provide(area=$area)] [Provide(area=north)] north(0.0722
4740410098884)
     [java] [DEB] SympleDS.(): s=0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0.0722474,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,1,0,0,0,0,0,1 A=25,27,29,30,15,0,2,4,6,8 a=ExpConfirm(area=$area) r=-0.1
     [java] [DEB] SympleDS.IM(): 3 SYS:[ExpConfirm(area=$area)] [ExpConfirm(area=north)] "Did y
ou say in the north?"
     [java] [DEB] SympleDS.IM(): 3 USR:[Confirm(yes)] [Confirm(yes)] "Yes please"
     [java] [DEB] SympleDS.UM(): 3 USR:[Confirm(yes)] [Confirm(yes)] Yes(0.6213219311024585) pl
ease(0.9867671621974712)
```

Figure A1. Training - Server Side


~/Desktop/RL/SimpleDS/web/main — node runclient.js train 3000 -v
...he.tools.ant.launch.Launcher -cp -lib /opt/homebrew/share/ant SimpleDS MANPATH=/opt/homebrew/share/man::

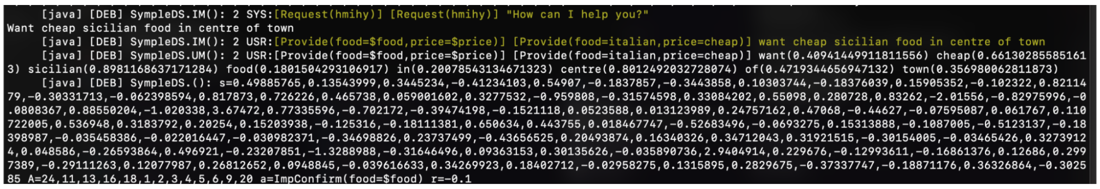
```
zaeemansari@Zaeems-MacBook-Air main % node runclient.js train 3000 -v
libraries loaded!
config file loaded!
web socket created!
configurations defined!
Connected to server!
brain created!
Initialised agent!
Running in training mode!
[steps avg.reward epsilon avg.actions dialogues time.hrs]
100 0.2343 1.0000 4.80 9 0.0002
200 0.2402 0.9950 4.55 18 0.0006
300 0.2257 0.9900 4.94 25 0.0010
400 0.2226 0.9850 5.14 33 0.0015
500 0.2260 0.9799 5.14 41 0.0019
```

Figure A2. Training - Client Side



```
     [java] [DEB] SympleDS.IM(): 2 SYS:[Request(food)] [Request(food)] "What type of food would you like?"
sicilian
     [java] [DEB] SympleDS.IM(): 2 USR:[Provide(food=$food)] [Provide(food=mexican)] sicilian
     [java] [DEB] SympleDS.UM(): 2 USR:[Provide(food=$food)] [Provide(food=mexican)] sicilian(0.1886168493776006)
     [java] [DEB] SympleDS.(): s=0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,1,0,0,0,0,1,0,1,0,0,0,1 A=24,29,30,31,16,1,2,3,4,9 a=Request(price) r=-0.09999999999998674
```

Figure A3. Agent Trained without Word Embedding in Interactive Mode (Random Value)



```
     [java] [DEB] SympleDS.IM(): 2 SYS:[Request(hmihy)] [Request(hmihy)] "How can I help you?"
Want cheap sicilian food in centre of town
     [java] [DEB] SympleDS.IM(): 2 USR:[Provide(food=$food,price=$price)] [Provide(food=italian,price=cheap)] want cheap sicilian food in centre of town
     [java] [DEB] SympleDS.UM(): 2 USR:[Provide(food=$food,price=$price)] [Provide(food=italian,price=cheap)] want(0.40941449911811556) cheap(0.66130285585161
3) sicilian(0.8981168637171284) food(0.1801504293106917) in(0.20078543134671323) centre(0.8012492032728074) of(0.4719344656947132) town(0.356980062811873)
     [java] [DEB] SympleDS.(): s=0.49885765,0.13543999,0.3445234,-0.41234103,0.54907,-0.1837857,-0.3443858,0.10303744,-0.18376039,0.15905352,-0.102322,0.82114
79,-0.30331713,-0.062398594,0.817873,0.726226,0.465738,0.059001602,0.3277532,-0.959808,-0.31574598,0.33084202,0.55098,0.280728,0.83262,-2.01556,-0.82975996,-0
.0808367,0.88550204,-1.020338,3.67472,0.77335596,-0.702172,-0.39474198,-0.1521118,0.0523588,0.013123989,0.24757162,0.47068,-0.44627,-0.07595087,0.061767,0.110
722005,0.536948,0.3183792,0.20254,0.15203938,-0.125316,-0.18111381,0.650634,0.443755,0.018467747,-0.52683496,-0.0693275,0.15313888,-0.1087005,-0.5123137,-0.18
398987,-0.035458386,-0.022016447,-0.030982371,-0.34698826,0.23737499,-0.43656525,0.20493874,0.16340326,0.34712043,0.31921515,-0.30154005,-0.03465426,0.3273912
4,0.048586,-0.26593864,0.496921,-0.23207851,-1.3288988,-0.31646496,0.09363153,0.30135626,-0.035890736,2.9404914,0.229676,-0.12993611,-0.16861376,0.12686,0.299
7389,-0.29111263,0.12077987,0.26812652,0.0948845,-0.039616633,0.34269923,0.18402712,-0.02958275,0.1315895,0.2829675,-0.37337747,-0.18871176,0.36326864,-0.3025
85 A=24,11,13,16,18,1,2,3,4,5,6,9,20 a=ImpConfirm(food=$food) r=-0.1
```

Figure A4. Agent Trained with Word Embedding in Interactive Mode (Synonymous Value)