

# Deep Reinforcement Learning for Dialog-Based System – Software Development Report

40107299

## Contents

1.	SYSTEM SPECIFICATION .....	2
1.1	Data Model .....	2
1.2	Function Definitions.....	2
1.3	Error Conditions.....	3
2.	DESIGN .....	4
2.1	User Interface Design.....	4
2.2	Software System Design.....	5
3.	IMPLEMENTATION AND TESTING .....	9
3.1	Machine.....	9
3.2	Languages, Packages and Libraries.....	9
3.3	Important Functions and Algorithms.....	10
3.4	Components.....	13
3.5	Testing.....	15

## 1. SYSTEM SPECIFICATION

### 1.1 Data Model

Reinforcement learning is concerned with how agents take actions in an environment in order to maximise a cumulative future reward. Deep reinforcement learning is formulating the problem using a deep network.

In short, the system is a reinforcement learning agent which replies to the user when the user enters an input. The learning agent utilises a deep Q-learning neural network in order to select the reply at a given time.

Where the input to the neural network is a float array vector of size 2,  $[S_1, S_2]$ , where  $S_1$  is a float representing the current sentence entered by the user and  $S_2$  is the float representing the previous input sentence.

These inputs are taken from a list of possible inputs for that given stage of the conversation.

The actions selected by the system will be an integer between 0 and 21 from the action space  $[0...21]$ , where each integer corresponds to a different reply  $[A_0...A_{21}]$  within the possible answers array. The integer selected during testing is the maximum Q-value. The Q-values for a given state (input) is a float array, where each float represents the "Q-Value" for an action  $[Q_0...Q_{21}]$ . The Q-value is an estimate by the system of cumulative future reward, where the reward is an integer sent to the agent whenever it receives an action, which is positive whenever the correct answer is selected.

Each component is further described in *Section 3.4* and the important functions and calculations are further described in *Section 3.3*.

### 1.2 Functions

The implementation of important functions is described in *Section 3.3*.

The main functions of the system will be as follows:

1. The main function or the end goal of the system is to select the correct response to send back to the user for a given input. This is aided by the following functions of the system.

2. Calculate the Q-values for a given state (context of the conversation and current input) and the possible actions (replies that the chatbot can send).
3. Store the state, Q-value, action taken and reward for each step in the conversation into a replay memory. Use the replay memory to update the Q-Values.
4. Calculate and reduce the loss between the newly calculated Q-values from the replay memory and the Q-values currently estimated i.e. learn how to correctly reply for a given state.

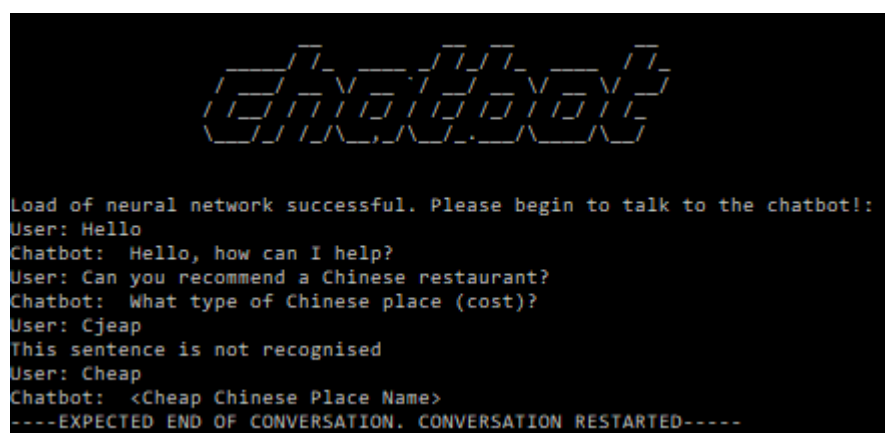
### 1.3 Error Conditions

Here details the error conditions of the system.

- *This sentence is not recognised*

While testing the system, if you enter a full word sentence where every word is not in the word corpus, the following error will appear: “This sentence is not recognised.” This is because there is no word vector representation learned for this so when the environment gets the state, no current state is returned –thus when this state is input into the neural network it is not recognised.

This is a hand-coded “try and except” error message and is not output by the agent. To work around this, the previous state is kept the same so the conversation can resume from the same point in the conversation flow, as shown below where the user has mistakenly typed the word “cheap” as “Cjeap”, where “Cjeap” is not in the word corpus.



```
Load of neural network successful. Please begin to talk to the chatbot!:
User: Hello
Chatbot: Hello, how can I help?
User: Can you recommend a Chinese restaurant?
Chatbot: What type of Chinese place (cost)?
User: Cjeap
This sentence is not recognised
User: Cheap
Chatbot: <Cheap Chinese Place Name>
----EXPECTED END OF CONVERSATION. CONVERSATION RESTARTED-----
```

Image 1: Incorrect Sentence Entered

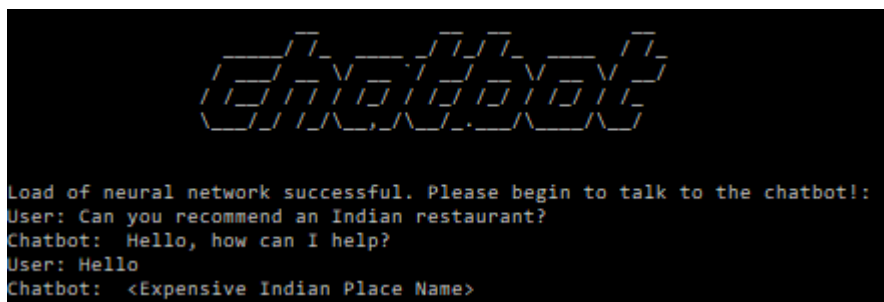
- *Incorrect Conversation Flow*



*Image 2: Expected Conversation Flow*

This is not necessarily an error, but if the conversation does not follow the exact flow shown above then incorrect expected outputs occur. This is because the agent trained to a fixed conversation flow.

An example of such a conversation is shown below:



*Image 3: Example of Unexpected Conversation Flow*

- *No Checkpoint Found*

This error occurs whenever the testing mode is run before the training mode. This means there is no checkpoint file to load within the appropriate folder. The user will be informed that no checkpoint could be found with the following error message: "Could not find checkpoint." However, the user can still chat to the chatbot, but the neural network will only have initialised Q-values, similar to what it would look like at the beginning of training; so the selected replies by the chatbot will be completely random.

## 2. DESIGN

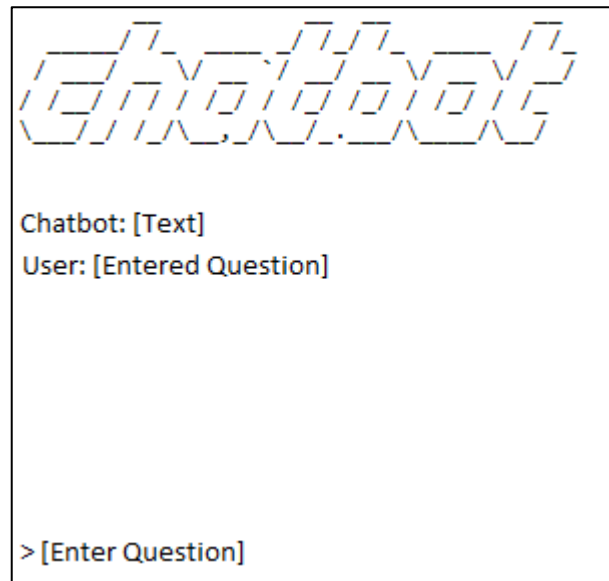
### 2.1 User Interface Design

The user interface will just be based on the command line. The command line screen will clear upon opening; a simple header will display to show to the user that the chatbot has been opened. From there, the user will use the command line input to input text to the chatbot like the majority of other

chatbots or like a simple messaging service. This means it will be quite intuitive for a user to use and will reduce the burden of testing.

The message entered by the user will be displayed to the screen and the chatbot agent will instantaneously find an optimal response which will be displayed to the screen underneath the user's question.

An example of what the chatting screen will look like is displayed below:



*Image 4: Chatbot Window Design*

## *2.2 Software System Design*

In *Section 2.2*, the architecture and data flow are described along with the role of each component within the system.

### *2.2.1 Architecture and Hyper-Parameters*

The architecture for this system is a multilayer fully-connected neural network. It consists of an input layer, 2 hidden layers and an output layer. The input layer has an input size of 2 – this is the size of a state: previous question and current question. Both hidden layers have 20 nodes each and the output layer is of size 22 (the number of possible actions). The hidden layers use the rectified linear unit (ReLU) activation function.

Hyper-parameters are variables which determine the structure and how the network is trained. The other hyper-parameters for the neural network are as follows:

Learning rate: 0.01

Epsilon: 0.1

Discount factor: 0.95

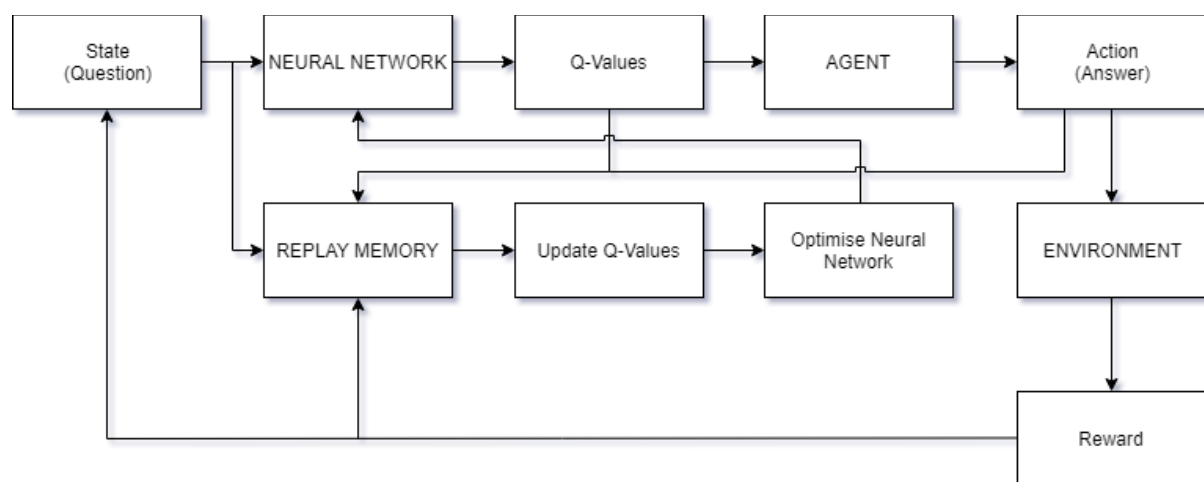
Replay memory size: 1000

Batch Size: 50

Learning steps: 1,000,000

Weight Initialisation: `tf.truncated_normal_initializer` with a standard deviation of 0.1

### 2.2.2 Data Flow

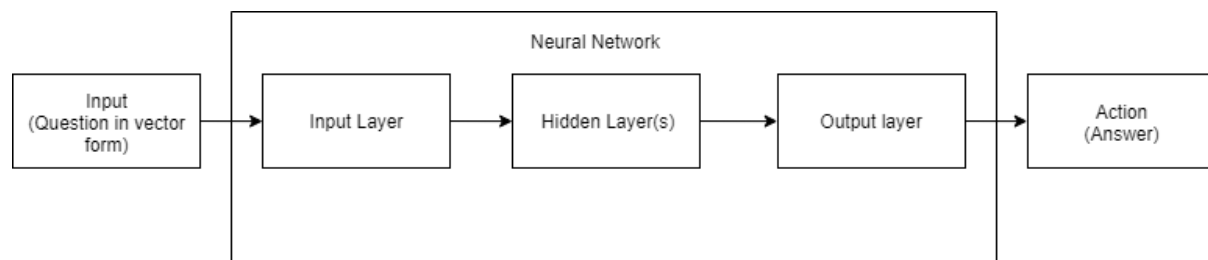


*Image 5: Data Flow in System between Components (components described in Section 2.2.2)*

The image above shows how the agent learns. The outer loop around the edge of the diagram shows the agent chatting and exploring the environment. The environment generates a conversation which is used to create the states which the neural network observes and estimates Q-values for; and this state is also passed to the replay memory in the inner loop in the middle of the diagram. Based on the Q-values estimated by the neural network, an action is selected and this action is passed to the environment. The environment then checks if this action was correct, then issues a reward. This reward and action is passed to the replay memory; this information is added into the same iteration as the state from earlier in the loop.

The remainder of the inner loop is activated when the replay memory is full – the Q-values are updated in a backwards sweep using the rewards that have been stored during training. After this,

the neural network is optimised to try to teach the neural network to select the correct action at a given state by updating the Q-values.



*Image 6: Data Flow in Neural Network*

This image shows briefly how data flows through the neural network. The state, which is a text input represented as numbers, is used as the input and passed into the input layer. This is then passed to the hidden layers which contain the activation function. These hidden layers then link to an output layer, where an action is selected.

### 2.2.3 Components

This section gives a brief description of the main components and the role of each component. How each component was implemented is described in *Section 3.4*.

#### 2.2.3.1 Environment

The environment in this system is a chatbot environment set in a domain of selecting a restaurant. This means the agent will have to explore questions that a customer would ask a customer service representative regarding a restaurant. During training, the environment randomly generates a conversation from the list of questions for the agent to explore and learn how to best respond to the given questions over time.

The environment also contains the reward function. This means that when the environment outputs a vector state based on the next sentence in the conversation array, the agent selects a reply and then, based on this reply to the given question, the environment sends the following information back to the agent:

- 1) Whether the answer was correct or incorrect.
- 2) If this is the end of the conversation.

This information is used by system as described below.

### 2.2.3.2 *Replay Memory*

The replay-memory contains several states from the chatbot environment i.e. several questions in vector format. For a given state the Q-values, the action taken and whether it was the final state in that episode are stored. This information is passed to the replay memory by the agent for every iteration (every input) of the training loop, however the Q-values passed to the replay-memory are updated by the replay-memory itself to pass the reward back to earlier states with a discounted value. The replay-memory is used to optimise the neural network once it is full by sending the calculated Q-values for the given states into the neural network; and then it is emptied again.

This also contains random batch sampling as using a replay-memory with random batch sampling helps to stabilise the training of the neural network.

### 2.2.3.3 *Deep Neural Network*

Neural networks are a means of doing machine learning, where the first layer takes in the input and the final layer produces the output of the neural network. “Deep” implies that the neural network will consist of multiple layers; more than 1 hidden layer.

The neural network estimates the Q-values for a given state i.e. it determines which reply/action is optimal to give to a user when a question is input. This state is generated by the environment.

The specific architecture of this neural network is described in *Section 2.2.1*

### 2.2.3.4 *Agent*

The agent performs the interactions with the environment to explore it and develop the optimal Q-learning policy i.e. maximize its rewards. The agent in this case gains reward by selecting correct responses (reward function described in *Section 3.3.3*) to the inputs sent to it by the environment during training. After training, the agent should be able to select the correct action every time a learned state is sent to it.

The agent selects actions either randomly, or by selecting the highest Q-value estimated by the neural network. This action selection process of the agent is described in *Section 3.3.5*.

The technical aspects of the agent are described in *Section 3.4.4*.



### 3. IMPLEMENTATION AND TESTING

#### 3.1 Machine

The neural network was trained and tested on a machine with a 2.30GHz CPU. The machine also has 8GB RAM, which is far more than sufficient for the small replay memory required for this program. The machine's operating system was Windows 10.

#### 3.2 Languages, Packages and Libraries

The system was implemented and tested with the following software and library versions:

Language, Package or Library	Description
Python 3.6 with the following imports: <ul style="list-style-type: none"><li>- OS</li><li>- time</li><li>- sys</li><li>- timeit</li></ul>	Python was the language used for the implementation. OS was used for navigating folders, time was used for time delays on the user interface, and sys was used to clear the command window for the user interface.  From timeit, default_timer was imported. This is a built in function to measure execution time. Default_timer measures the wall clock time.
Tensorflow 1.4.0	Tensorflow is an open-source machine learning framework. This provided the tools for implementing the neural network.
Gensim 3.1.0	Gensim is a library which contains Word2Vec with additional functionality built in. This was used to build the data model.
NLTK 3.2.5	From NLTK, word_tokenize was imported. This was used for tokenizing the word in the sentences.
Pandas 0.22.0	Pandas is a package for data structures. It was used to read the CSV containing the conversations.
NumPy 1.14.1	NumPy is a library that adds support for large, multi-dimensional arrays and matrices as well as functions. It was used for the multiple arrays and matrices contained in this program as well as many mathematical functions.
argparse 1.1	This is used for parsing to the command-line. It was used

	specifically to specify training or testing.
Matplotlib 2.1.2	Matplotlib is a Python 2D plotting library. It was used to plot the average reward over time for the results segment of the dissertation.

*Table 1: Languages, packages and libraries used*

### 3.3 Important Functions and Algorithms

This section details how essential elements of the system were implemented.

#### 3.3.1 Q-Values

The Q-values for a given state is a vector with a value for each action in the action space. This is an indication of the future reward the agent can receive by selecting this action for this given state. The Q-values are initialised to a low value at the beginning.

The new Q-values are calculated within the “update\_q\_values” method. When the replay memory (Section 3.4.2) is full, the replay memory goes backwards updating the Q-values for each state. If the current state was the end of the conversation, the Q-value is just the reward. Otherwise, it is calculated with this formula (put simply):  $Q\text{-value} = \text{reward} + \text{discount factor} * \max Q\text{-value for next state}$ .

```
if end_episode:
```

```
    value = reward
```

```
else:
```

```
    value = reward + self.discount_rate * np.max(self.q_values[curr + 1])
```

The neural network is then passed a random batch of these Q-values at each time of optimization. The neural network then calculates the loss (Section 3.4.2) to learn how to estimate better Q-values.

#### 3.3.2 Loss

The loss is implemented in one line within the neural network. The loss is calculated is L2-Regression – the mean-squared error. This error measures the error between the Q-values that are calculated by the neural network and the Q-values that are input from the replay memory during training.

Reducing this loss will help the neural network estimate “more correct” Q-values in order to select the correct action.

### 3.3.3 *Reward Function*

A conversation is generated for each training step of the system. A conversation consists of a greeting, an answer and the relevant information to find the answer e.g. a “cheap Italian restaurant” would consist of “Italian” and “cheap” to find it.

For a conversation, if the agent selects a correct answer for any stage of the conversation, then a reward of +0.2 is returned to encourage the system to select this action again for the given state. For a completely correct conversation, a reward is issued of 1. The reward is discounted by 0.95 for each step backwards so that the system learns the entire sequence correctly. Otherwise, if the agent selects an incorrect answer, no reward is issued i.e. a reward value of 0.

The implementation simply adds this to an integer which is returned, along with a Boolean which signals the end of episode. When the end of episode Boolean is issued, a new conversation is generated and the process begins again.

### 3.3.4 *Optimization*

For optimization, a fixed-size batch of 50 indices is randomly selected from the replay memory of size 1000 when the replay memory is completely full. For these batch indices, the Q-values and states are returned from their relevant arrays. These 50 corresponding states and Q-values are input into the neural network for optimization. The neural network tries to reduce the loss (*Section 3.3.2*).

For this system, the selected optimizer was RMSPropOptimizer, which is used to minimise the loss.

### 3.3.5 *Action Selection*

The size of the action space (the number of actions that can be selected) is determined by how many values are in the reply columns in the conversations CSV file. In this case, the reply columns are the replies to the greetings, the place and the final answer.

When the function is called to get the action, there is a 10% chance to select a random action and a 90% chance to select the highest estimated action (action which the system believes is the most likely to return the highest reward); the max Q-value value from  $A_0$ - $A_{21}$ . When a random action is selected, a reduced action set is used; the agent was allowed to select an answer from the appropriate column. However, when selecting the highest action value, the entire action set is still used.

For testing, the highest estimated action value is selected.

### 3.3.6 *State*

The state is the input to the neural network. This is a numerical representation of the current conversation. For this system, the state consists of the word vector of the current input and the previous input.  $State = [Current\ Input, Previous\ Input]$ , where the inputs are float word vectors.

Both the previous input and the current input are represented by 1 word vector each, therefore the state is a float array of size 2. This should be increased as it may be possible to find 2 words with the same or very similar vector representation – the neural network would find it difficult to distinguish this. This was not done as the system would need to be re-tuned to find the optimal hyper-parameters. A better solution would be to use an LSTM.

The functionality implemented in the `get_state` class is actually supposed to sum and average each word in the input sentence to calculate the state. However, the input “sentences” used in this system are actually all one word long e.g. “Hello”, “Chinese” and “Expensive”. This allows for expansion by implementing the use of full sentences.

The Word2Vec model is created in the `create_model` class. This class adds the relevant needed words to a corpus, tokenises this corpus then generates a Word2Vec model.

### 3.3.7 *Conversation Generation*

The environment generates a conversation for the agent to explore:

Conversation=[Greeting, Place, Cost, Answer]

The environment first selects a random place and cost by selecting an integer between 0 and the number of elements in each. An example result is: “0” representing “Italian” and “1” representing

“expensive”. The environment takes these integers and calculates the index of the relevant answer with the following line of code:

```
answer_n = ((len(cost)*place_n) + (cost_n+1)) - 1
```

In this case, it returns the integer “1” for an expensive Italian place. There is only one greeting so it is just added every time to the conversation array (index 0). Note that the answer is not actually sent by the environment to the agent as an input, it is only used for verifying the agent’s final answer.

### 3.4 Components

This section details how each of the components were implemented. These are the 4 main classes of the main body of code. The data flow between each component is highlighted by *Image 5* in *Section 2.2.2*.

#### 3.4.1 Environment

The environment creates the conversation for the agent to interact with during training. It does this by selecting a greeting, a place and a cost randomly – then it calculates the appropriate final answer the system should retrieve (*Section 3.3.7*). This is then stored in an array.

The environment also contains the functionality to return a state (described in *Section 3.3.6*) as well as defining the functionality for the agent taking a “step” in the environment. A step in the environment takes in the action and the current question and outputs the reward and whether the end episode has been reached. This function is further described in *Section 3.3.3*.

#### 3.4.2 Replay Memory

The replay memory contains storage space for the previous states of the environment along with the corresponding Q-values, actions and rewards observed at that time. It contains arrays for all of the data the size of the current step iteration of training after the replay memory has been wiped (where the arrays are emptied). The discount rate is also initialized here.

As described in *Section 3.3.1*, the Q-values are updated in a backwards sweep. This occurs in the replay memory when the replay memory is full. After, the memories are thrown away as the replay memory is reset.

When a batch is needed for optimization, the replay memory generates 50 random indices from the memory. The states and the corresponding Q-values observed at these indices are fed into the neural network by taking the values from the arrays at those indices.

### 3.4.3 *NeuralNetwork*

The neural network was created with TensorFlow layers. This class contains the TensorFlow session; this session is run whenever there is a call to optimize. The optimization function is also contained in the neural network class. This gets the batch from the replay memory (*Section 3.4.2*) and runs one optimization step and returns the current loss. This loss is reduced by the optimizer (*Section 3.3.2*).

This session can also be run to return the values that the neural network estimates for a state, by passing in the state with a `feed_dict`.

Finally, the neural network class also contains the functionality to save and load itself using a saver. This is used at the end of training to save the neural network to be loaded during testing. This is saved into a checkpoint directory, from which it can be restored during testing.

The neural network class also contains the merge function for TensorBoard graphs.

### 3.4.4 *Agent*

The agent takes in a value of either “training” or “testing” so it knows which mode to load.

During training; initially, the replay and the environment are initialised here along with hyper-parameters such as the learning rate and the epsilon greedy value (chance to take a random action). These values are not used when testing is selected.

This class also implements the action selection function, as described in *Section 3.3.5*. For testing, there is a different method implemented for selecting the action, where the maximum Q-value is selected.

Importantly, the run method is defined in this class, which, during training, runs the environment and uses the neural network to decide which actions to take in each step of the environment. The agent passes the state to the neural network to estimate Q-values for action selection. It sends this action to the environment which sends back the reward and end of episode signal.

All of this information is passed to the replay memory by the agent. The agent then checks if the replay memory is full; so that the Q-values can be updated and optimization can begin. The agent then resets the replay memory.

When there is an end of episode observed, the agent calls for a new conversation to be generated, and resets the conversation step counter and reward for the episode, and also increments the episode counter.

Finally, at the end of training, the neural network is saved. Also, it is worth noting that the agent also keeps stores rewards for printing statistics during training.

If the testing argument is passed in instead of training, the saved neural network is restored. The user can then chat to the chatbot. The agent will then reply by first getting the state from the environment class. This state is then passed to into the neural network by the agent; the agent then observes the maximum value which tells the agent what is the best reply to give to the user based on its previous experiences. This action is just an integer index value within an array, so to print it there is an array with the corresponding outputs the value represents; which is printed to the screen.

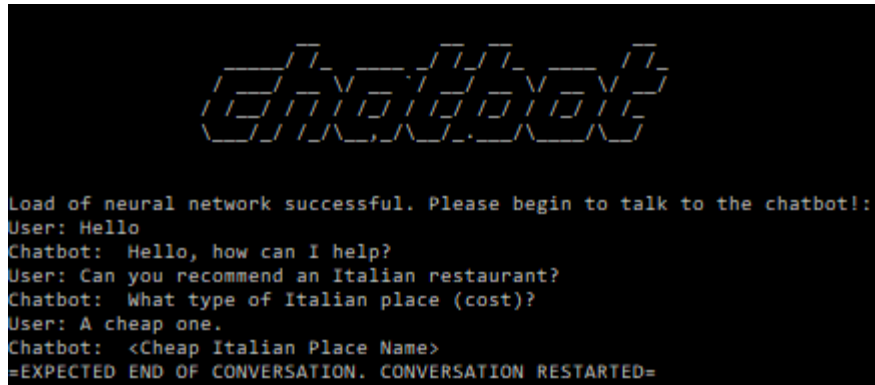
When the user enters another input, the previous input is also loaded so that both of these can be used to create the state.

The number of questions asked is tracked. Since the agent was only trained on fixed conversation lengths, it's not suitable for any longer conversations. So, when the user has entered 3 inputs and the agent has issued 3 replies, a message is displayed that the end of conversation is expected. The current question counter is reset and the previous question, so that a new conversation may take place.

### *3.5 Testing*

The system is tested by having a human tester input to the system and test for expected outputs. This means the system is run in "testing mode" instead of "training mode". Due to the limited size of possible inputs and action outputs, all sequences can be tested to ensure that the correct answer output occurs.

Here is an example screenshot of a full test conversation:



```
Load of neural network successful. Please begin to talk to the chatbot!:
User: Hello
Chatbot: Hello, how can I help?
User: Can you recommend an Italian restaurant?
Chatbot: What type of Italian place (cost)?
User: A cheap one.
Chatbot: <Cheap Italian Place Name>
=EXPECTED END OF CONVERSATION. CONVERSATION RESTARTED=
```

*Image 7: Full Example Dialogue*

This is repeated for every possible valid conversation sequence, as learned during the training phase. For the current implementation of the system, all conversation sequences returned the correct result.