SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING and COMPUTER SCIENCE

CSC4006 – RESEARCH AND DEVELOPMENT PROJECT

Dissertation Cover Sheet

A signed and completed cover sheet must accompany the submission of the Research Dissertation submitted for assessment.

Work submitted without a cover sheet will **NOT** be marked.

Student Name: Keelan Graham-Sands Student Number: 40107299

Project Title: Deep Reinforcement Learning for Dialog-Based System

Supervisor: Vien Ngo

Declaration of Academic Integrity

Before submitting your dissertation please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the Student Project Handbook
2. Contains full acknowledgement of all secondary sources used (paper-based and electronic)
3. Does not exceed the specified page limit
4. Is clearly presented and proof-read
5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.

By submitting your dissertation you declare that you have watched the video on plagiarism at http://www.qub.ac.uk/directorates/sgc/learning/WritingSkillsResources/Plagiarism/ and are aware that it is an academic offence to plagiarise. You declare that the submission is your own original work. No part of it has been submitted for any other assignment and you have acknowledged all written and electronic sources used.

Student's signature	*Date of submission*
Keelan Graham-Sands	04/05/2018

Deep Reinforcement Learning for Dialog-Based System

A dissertation submitted in partial fulfilment of

the requirements for the degree of

MASTER OF ENGINEERING in Software Engineering

in

The Queen's University of Belfast

by

Keelan Graham-Sands

04/05/2018

ACKNOWLEDGEMENTS

I would like to thank my supervisor Vien Ngo, who was very helpful and a continuous source of invaluable advice throughout the duration of this project; as well as providing me with initial papers and learning resources for me to get started.

ABSTRACT

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

This paper presents a deep reinforcement learning system within the domain of recommending a restaurant for a user. This system implements Q-learning in order to train it. This system keeps track of the conversation using a combination of the word vector of the current input and a word vector of the previous input in order to select the response at a given time. This system is trained on a small dataset and it trains fully in around 40 minutes. The results of this paper conclude that the agent was successfully able to navigate the proposed environment, however it remains to be proven how far this approach can go.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Background and Motivation

Today, many large companies are trying to develop dialogue systems including chatbots as they are becoming more ubiquitous in society today (Serban et al., 2017). These dialogue systems can provide technical support, assist with sales or simply talk with a human about general topics (Serban et al., 2017). In most cases, chatbots use messenger apps to communicate with customers. This is particularly useful as messenger apps are among the most popular apps today (Nusca, 2017) and therefore implementing a chatbot within a messenger app will be intuitive to many users. Chatbots can often find a perfect response to a user's query with just one or a few basic questions. Chatbots are easy to use as they are very intuitive, and they tend to be faster than calling a company representative.

Currently, most chatbots have scripted dialogue management i.e. the chatbots rely on a script to know what to say next to the user, which leaves the conversation flow to be very limited and the dialogue very short. Many companies are hoping to develop bots to have natural conversations indistinguishable from human ones, and many are claiming to be using Natural Language Processing and Deep Learning techniques to make this possible (Britz, 2016). On top of this, this scripting type of implementation is not scalable across domains, as you will be required to write a new script for each domain.

Motivated by the drawbacks of initial systems, this paper proposes a deep reinforcement dialog system that learns policies from input questions in text format. To achieve this, an agent is used to explore a chatting environment and the agent receives the information whether the previous response it has selected resulted in a reward i.e. the correct response was selected.

The entire process of creating the agent is formulated as one sequential decision-making problem under partial observability. The partial observation comes from the fact that the agent cannot understand and percept fully the assumed context of the conversation and the intention of the human. Technically, this would make the response/search space (in texts) grow exponentially. Therefore, deep reinforcement learning is used to scale up to practical problems.

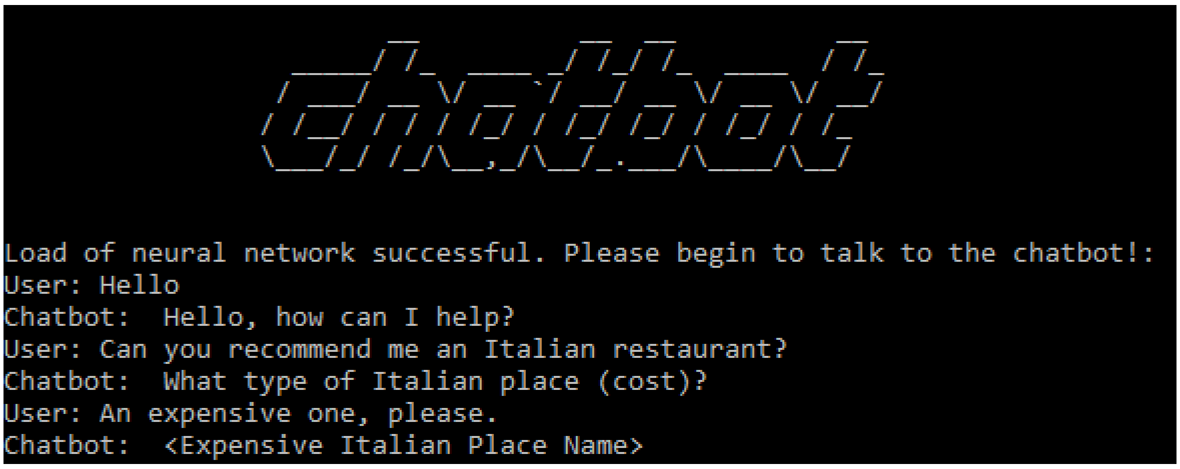
Reinforcement learning allows learning via a scalar reward (Mnih et al., 2013) which is a result of taking actions. The goal of reinforcement learning is to learn good policies for sequential decision problems by optimizing a cumulative future reward signal (Sutton and Barto, 1998). Within action selection research, scholars have adapted the reinforcement learning model (Cuayáhuitl, 2016) to select actions because then dialogue design can be treated as optimization problem with the performance improving over time as opposed to having substantial amounts of hand-labelled data.

The rest of the paper is organized in the following fashion: The remainder of Section 1 describes the specific problem in detail and explores some previous related work. Section 2 presents the technical aspects of the system and how the approach was selected along with the respective algorithms. Section 3 contains the results of the investigation as well as the conditions upon which the system was executed. These results are then analyzed. Finally, section 4 contains the summary of the project work with the findings and suggestions for future work.

1.2 Problem Setting

The setting of this project will be to build a deep reinforcement learning agent in the domain of recommending a restaurant for a customer. The dialogue agent will be able to greet the user and suggest a restaurant to a user based on the type of food the customer is looking for as well as the cost range.

Here is an example 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 me an Italian restaurant?
Chatbot:  What type of Italian place (cost)?
User: An expensive one, please.
Chatbot:  <Expensive Italian Place Name>
```

Figure 1: Example Conversation

For humans, it is clear to see that the user here is looking for an expensive Italian restaurant to eat food – there is no learning required by humans in order to understand what is being asked.

The problem is that the chatbot must learn all the steps leading up to selecting the correct reply. It must know that the user is looking for an Italian place and an expensive place to eat and use this information to select a correct overall response. This is where the deep Q-learning neural network is used. The chatbot above has selected the correct responses based on previous experiences from interacting with conversations generated by the environment and updating this neural network to learn the correct responses.

1.3 Related Work

Below, a summary of some previous work is presented which are related to the work to be carried out by this research.

Mnih et al. (2013) used reinforcement learning and to play various Atari games. The model used is a deep convolutional neural network trained with Q-learning. The inputs in this case were images from the game environment, as opposed to a question/text input changed into vectors.

This paper greatly showed the transferability of deep reinforcement learning agents from one domain to another, as the same method was applied across seven Atari games with no adjustment of the architecture or learning algorithm and it managed to outperform previous approaches in six of the seven games – including approaches such as Sarsa and Contingency. This implementation even managed to surpass human experts on three of the games. It was the first demonstration of a real deep learning model that could adapt to different environments without human interference.

SimpleDS (Cuayáhuitl, 2016) is a publicly available dialog system trained with deep reinforcement learning which implements the same Q-learning algorithm set out by Mnih et al. (2015), which will be implemented in this solution. The system performs action selection directly from the raw text of the last system and user responses. The domain selected by Cuayahuitl is searching for a restaurant, like this program. The results of this paper suggest that within the restaurant domain, it is possible that reasonable interactions can be generated using the proposed approach. However, in the system SimpleDS, the learning agent is made using JavaScript and the environment is made with Java. The research carried out will be mostly an extension of this by implementing a deep Q-learning dialogue agent using Python and TensorFlow.

Su et al. (2016) used deep reinforcement learning to allow users to simulate an interview by using the predicted state and action pair to generate an interview question. The state tracking model in this investigation was made using Python modules and the deep reinforcement learning-based action prediction model was a modified version of SimpleDS. This system uses Word2Vec to produce word vectors. Deep reinforcement learning was used to learn the relation between dialog states and actions and this system also implements deep Q-learning with experience replay. When the system was evaluated, it was noted that an encouraging result was obtained for dialog state tracking, action selection and interview question generation.

Serban et al. (2017) implemented a deep reinforcement learning chatbot called MILABOT which was capable of conversing with humans on popular small talk topics through both speech and text. This applied reinforcement learning to real interactions to select the best responses. It experimented with different policies and it was observed that a policy based on Q-Learning performed the best among all policies experimented with. This chatbot was developed for the Amazon Alexa Prize competition and managed to significantly outperform other systems in terms of user score for the Q-learning implementation.

To sum up the previous work; these papers suggest that implementing a deep reinforcement learning agent is scalable across domains which addresses previous issues of chatbots. It also shows encouraging signs for chatbot implementation and in the case of SimpleDS, it shows that a similar problem can be represented by deep reinforcement learning.

The technical aspects of the related work are described in *Section 2.1*.

2. TECHNICAL SECTIONS

2.1 Related Work

This section will detail the implementations used by those from the previous work in *Section 1.1*. The architectural components and hyper-parameters are described further on in this section.

The work carried out by DeepMind (Mnih et al., 2013) will not consist of a very similar architecture to the proposed system. This is since for a game there are substantially more states than for this small system as the states in the DeepMind paper are based on images frame from the game. This resulted in DeepMind using very large architectures with convolutional neural nets to process the images. For example, in the DeepMind paper, they used a very large replay memory with 1,000,000 states which did a batch update every step. This is also partly due to the resources that DeepMind possesses. It is also worth noting that DeepMind used RMSProp for the optimizer of the system.

The most suitable system to observe would be SimpleDS (Cuayáhuitl, 2016) due to the similarities between SimpleDS and the proposed system. For SimpleDS, it consisted of 35 actions and up to 100 word-based features; the architecture consisted of up to 100 nodes on the input layer (input size), two hidden nodes of size 40 and 35 output nodes (actions). The hyper-parameters selected by

SimpleDS are as follows: replay memory size 10,000, a discount factor of 0.7, minimum epsilon 0.01, batch size 32 and the number of learning steps are set to 10,000.

Comparing the solution proposed by this paper and that proposed by SimpleDS, the neural network used in the proposed paper will be smaller due to the smaller number of actions and input dialogues (this system's architecture is further described in *Section 3.1.2*). However, proportionally the batch size will be larger in this system since SimpleDS encourages efficient interactions by limiting the action space size (limiting the number of actions the system can take at a particular time). In this case if the batch size was too small, it may take a long time for the system to select a state with a correct answer during random selection as the system has a larger amount of actions to select from at a given time. This could be fixed by calculating the estimated error between the newly calculated Q-values and the Q-values estimated by the neural network, where a high estimation error early would generally imply that a correct sequence has been taken as the Q-value will appear much higher than the estimated value. This is left as future work (*Section 4.2*).

Due to how the environment generates sequences, the number of learning steps would have to be increased. This is since there is only a maximum of 3 states input per episode (the size of 1 conversation). This is also based on the fact that SimpleDS uses a reduced action set and this system only occasionally uses a reduced action set (see *Section 2.5*). This means during training it may take some time for the system to observe a correct sequence and when it comes to optimization, the batch may not select an instance where a correct sequence occurred.

The rest of the hyper-parameters described by SimpleDS are quite suitable for this system.

2.2 Problem Definition

Following on from *Section 1.1*, this paper details a learning agent in the environment of a chatbot in a restaurant domain. When the user inputs a sentence (*Section 2.2.1*), the agent will select the correct response (*Section 2.2.2*) for that given question based on previous experiences it has had with that question.

Here is a simple view of the problem:

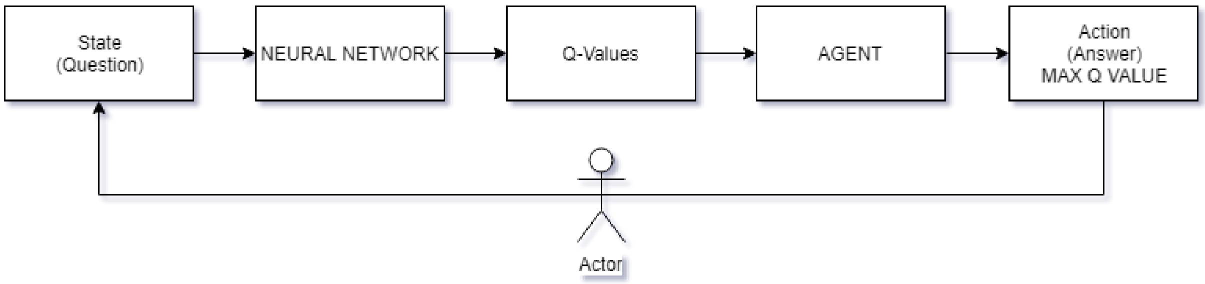
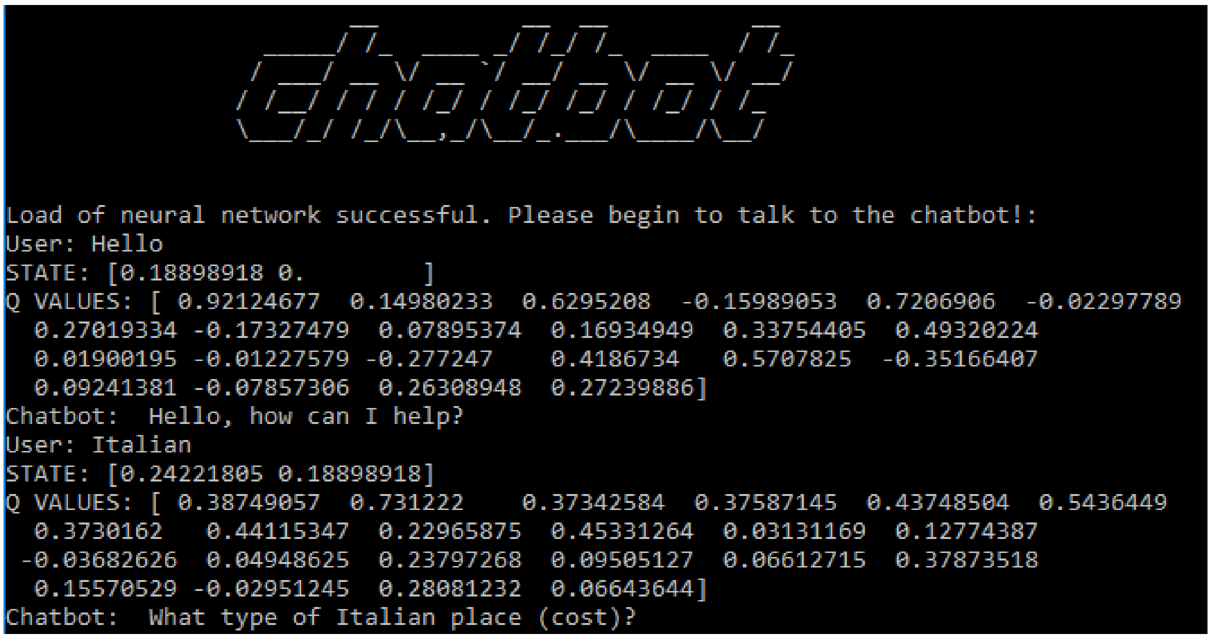


Figure 2: Basic Overview of the problem

Below is an image (*Figure 3*) from the testing the chatbot that shows how briefly how the system selects a response. A question is input by the user and the agent must learn to select the correct answer. The vector labelled “STATE” is the vector representation of the input question and the vector labelled “Q VALUES” is the estimated Q-values for each action – where a higher value represents a higher chance of reward. The chatbot then selects the highest action by selecting the highest Q-Value, where each action (number in the Q-values vector) represents a different response.



```
Load of neural network successful. Please begin to talk to the chatbot!:
User: Hello
STATE: [0.18898918 0.        ]
Q VALUES: [ 0.92124677  0.14980233  0.6295208  -0.15989053  0.7206906  -0.02297789
  0.27019334 -0.17327479  0.07895374  0.16934949  0.33754405  0.49320224
  0.01900195 -0.01227579 -0.277247    0.4186734   0.5707825  -0.35166407
  0.09241381 -0.07857306  0.26308948  0.27239886]
Chatbot:  Hello, how can I help?
User: Italian
STATE: [0.24221805 0.18898918]
Q VALUES: [ 0.38749057  0.731222    0.37342584  0.37587145  0.43748504  0.5436449
  0.3730162   0.44115347  0.22965875  0.45331264  0.03131169  0.12774387
 -0.03682626  0.04948625  0.23797268  0.09505127  0.06612715  0.37873518
  0.15570529 -0.02951245  0.28081232  0.06643644]
Chatbot:  What type of Italian place (cost)?
```

Figure 3: How the chatbot selects a reply

During training, the system tries to find the correct response for all the states. For example, if the system selects the action value corresponding to “Hello, how can I help?” for the input “Hello” – it is correct. The problem is how will the system ensure that the correct response is selected the next time “Hello” is input by a user? A deep reinforcement learning Q-network is implemented to solve this.

2.2.1 Inputs and State Tracking

The input to the system is a question or a sentence, S.

During training, the system gets these inputs from a CSV file. This CSV file has 6 headers; 3 pairs of inputs and outputs. The pairs were inputs and replies for greetings, places, and the cost (where the reply to this is the final answer). This CSV file is viewable in *Section 3.1.1.* The inputs consist of:

- 1 greeting (Hello).
- 7 places (Italian, Chinese, Japanese, Pizza, American, Indian, Kebab).
- 2 cost specifiers (Cheap, Expensive).

The environment generates a conversation of length 3 for the agent to interact with. To do this the environment selects a greeting, then a place and a cost e.g. "Italian" and "expensive". Then the environment selects the relevant final answer (an expensive Italian restaurant). Conversation = [Greeting, Place, Cost, Answer], where "Answer" is stored just for checking the final response. This means the environment will chat to the agent with the following conversation flow:

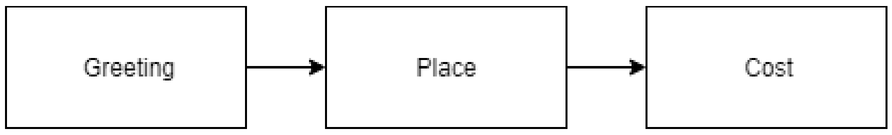


Figure 4: Conversation Flow

The system will input the greeting, then the place (Italian) then the cost (cheap). The expected output will then be a cheap Italian restaurant after the user inputs the cost.

As the system cannot interpret the input as text, the input must be changed into a numerical vector to be input into the system. Word embeddings can be used in a neural network for text data (Brownlee, 2017). This is achieved in this system by using Word2Vec (Mikolov et al., 2013), where each word in a corpus is mapped to a corresponding word vector. Word2Vec creates vectors that are distributed numerical representations of word features, such as the context of the individual words in the sentence. For this implementation, the context is not used, just the numerical vector is important.

Using the CSV which contained the inputs and actions, the headings which contained the inputs formed the corpus upon which the Word2Vec model was based. Since the input columns CSV file only contain a single word per row, only the vector for each word is needed to be learned as opposed to a full sentence. This will be used as the state which is passed into the neural network. It is worth noting that the functionality is built in this system to represent a sentence, where the sum of the vector representing each word can be summed and then averaged (Narkede, 2017).

Note that inputting a sentence such as “Can you recommend an Italian restaurant?” will only be represented by the word vector of “Italian”, since it is the only word in the corpus.

The current input by the user and the previous input by the user (0 if no previous input) are combined to form the state, therefore the states were a fixed-sized float array of size 2. This means that the context of the conversation can be held. For example, if the user inputs “Italian” and then “Expensive” then the system will search for expensive Italian places based on the 2 inputs.

An example of this vector representation is shown in *Figure 3*, where the state is [0.18898918 0.] for “Hello” where “0.18898918” represents the current input “Hello”, and “0.” represents the previous input (none). Then the state changes when the user inputs “Italian” to [0.24221805 0.18898918] where “0.24221805” represents the current input “Italian” and “0.18898918” represents the previous input (“Hello”). So, the state = [Current Input, Previous Input].

2.2.2 Outputs

As a response to the user question, the system will output an answer. The answer will be one of the actions, a, that the agent can take for any given question input.

The system interprets these outputs just as numbers from 0-21, however these correspond to an answer in the answers vector which will be displayed to the user in text form e.g. 0 corresponds to “Hello, how can I help?”

For this system, there are 22 possible actions the dialogue agent can take for every input. This consists of:

- 1 greeting (“Hello, how can I help?”).
- 8 place responses (e.g. “What type of Italian place (cost)?”).
- 14 final answers to present to the user (e.g. “<Expensive Italian Place Name>”).

2.3 Deep Reinforcement Learning, Q-Learning Algorithm and DQN (Deep Q-Learning)

Reinforcement learning is a learning model which tries to return the maximum cumulative future reward (Szepesvári, 2009). *Section 3.1.2* describes the architecture of the deep reinforcement model proposed for this system which consists of a full-connected multilayer neural network. *Figure 5* shows the data flow for the system for this paper.

Q-Learning (Watkins and Dayan, 1989) is a simple way for agents to learn how to act optimally in controlled Markovian domains. The basic idea is to have the agent estimate Q-values whenever it receives an input in text form from the user (see *Section 3.1*). The Q-values tell the agent which action to take is most likely to give the highest reward or highest cumulative reward in the future as described below.

As mentioned previously, a reinforcement learning agent is an agent which learns from interactions with an environment. A reinforcement agent is typically characterised by:

- A finite or infinite set of states, $S = \{s_i\}$, where i is the number of states.
- A finite or infinite set of actions, $A = \{a_j\}$, where j is the number of actions.
- A state transition function $F(s_t, a_t, s_{t+1})$ that specifies the text state at time $(t+1)$ given the current state s and the action a, where t is time.
- A reward function $R(s_t, a_t, s_{t+1})$ which is the mapping of each-state action pair to the reward.
- A policy $\pi: S \rightarrow A$ that defines a mapping from state space to action space.

The state, action and reward are the training data within a reinforcement learning system. The reinforcement learning agent will select its actions by approximating the optimal value function. Q-values are computed by applying Bellman updates. This is the formula for this created by DeepMind (Mnih et al., 2015):

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \middle| s_t = s,\ a_t = a,\ \pi\right]$$

Formula 1: Q-Learning Formula

Q* is the maximum sum of rewards r_t discounted by γ, the discount factor, at each timestep t, achievable by a behaviour policy $\pi = Pr(a|s)$, after making an observation (s) and taking an action (a), where Pr is the probability of taking an action. The aim is to develop an optimal policy for the learner to maximise the long-term reward. The rewards for this system are based on if the correct final answer is selected for the full conversation.

While there are other stable methods of training neural networks in the reinforcement learning setting, such as neural fitted Q-iteration (Diuk, Cohen and Littman, 2008). However, these methods can be quite inefficient, particularly if they are scaled into larger neural networks (Mnih et al., 2015).

As mentioned above, Q-learning with experience replay will be implemented. Experience replay is where the experiences are stored. The experience replay in this case is $e_t = \{s_t, a_t, r_t, ee_t\}$ in a dataset $D = \{e_1,\ldots e_t\}$ collected over many episodes into a replay memory. Note ee_t is an end of episode Boolean which signifies if the state is the last state in the episode. This is stored for each iteration of the system (where each episode contains several iterations – and the end of which is denoted by an end of episode Boolean).

The multilayer neural network serving as the function approximator with weight θ is regarded as a Q-network. The weights are being optimized to improve the ability of the neural network to estimate Q-values. Q-learning updates are applied on samples of experience = *{(s,a,r,ee) ~ U(D)}*, drawn uniformly at random from the pool of stores samples i.e. a random batch will be sampled from the replay memory which will be used to update the function approximator. This approach is beneficial because random sampling breaks correlations and reduces the variance of the updating compared to updating with consecutive indices; and using a random mini-batch approach helps to avoid a divergence in parameters. The Q-learning update at iteration *i* will uses the following loss function in the paper by Mnih et al. (2015):

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim \mathrm{U}(D)}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)^2\right]$$

Formula 2: Loss Calculation

Where *r* is the reward, γ is the discount factor and θ_i are the parameters of the Q-network at iteration *i* and θ are the target parameters which are updated with the Q-network parameters (θ^-_i), over *C* steps. The loss function for this system will calculate the mean-squared error between the Q-values in the replay memory and the current Q-values estimated by the neural network. This produced the following equation for $L_i(\theta_i) = [(r + \gamma(Q_{t+1})) - Q(s, a, \theta_i)]^2$. The network parameters here are updated every time the replay memory is full.

A loss function is what must be optimized by the neural network. This is the mean-squared error between the Q-values that are output by the neural network and the target Q-values. This means that when a reward is issued, there will be updated Q-values different to the Q-values that the neural network initially expects, so it must learn to increase the Q-values for action *a* given this state *s*, that has achieved a positive reward. The neural network also learns how the system achieved this positive reward as the reward is distributed back to previous states at a discounted rate, meaning that the neural network will learn how it managed to achieve this reward.

The selected optimizer for this system was RMSProp (Hinton, Srivastava and Swersky, 2014), like many related systems. This operates on the loss function.

The replay memory discussed above updates the Q-values and performs one optimization run when the replay memory is completely full (Pederson, 2017). This is a different method than used by DeepMind (Mnih et al., 2013) (Mnih et al. 2015), where optimization was completed every step of training. All the data from the replay memory is thrown away after learning.

2.4 Reward Function

The reward function for this system is quite simple; it is based on whether the system was able to select the correct final answer that the environment was expecting, and the agent deduced this by selecting correct responses for the entire conversation. For example, if the environment was expecting an expensive Chinese restaurant to be returned as the eventual answer then the agent must answer this correctly and deduce that the environment asked for both an expensive restaurant and a Chinese restaurant along the way.

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 (this helps learn the individual steps). When the agent gets the answer correctly as well as all the previous steps, then a reward of 1 is issued. This reward is discounted for each step backwards, so the agent begins to learn this entire correct sequence (*Section 2.3*).

Unlike SimpleDS (Cuayáhuitl, 2016), this system does not use a reduced action set roughly 90% of the time (*Section 2.5*), so a large number of episodes is used to ensure that the system eventually selects the correct answer and begins to learn how did it achieve the correct result (higher reward).

2.5 Action Selection

The agent selects actions according to an Ɛ-greedy policy (Mnih et al., 2013). The Ɛ-value indicates the chance to select a random action a_t. Otherwise the action selected is the maximum Q-value.

Like SimpleDS (Cuayáhuitl, 2016), this system uses a reduced action set. This means that for a given state, the system has access to a fewer number of actions (actions more relevant to that input). For this system, the reduced action set is decided by the step in the conversation; for example, when the conversation has just started the action set is reduced to just the actions which are greetings.

However, unlike SimpleDS, this reduced action set is only when the epsilon selects a random action. This means that when the system is selecting the max Q-value, it has access to the full action set.

2.6 Algorithm

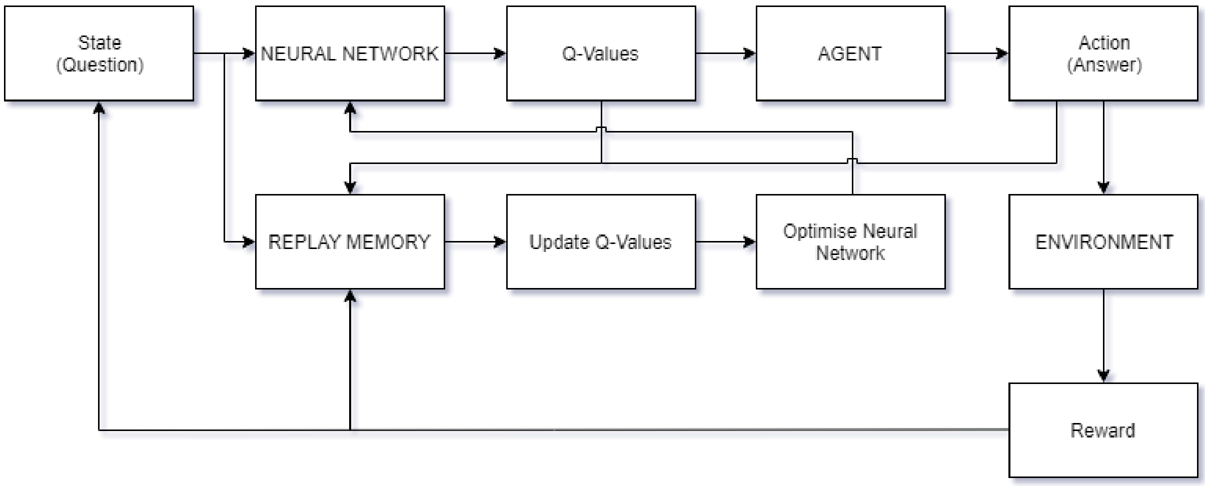


Figure 5: System Data Flow

The previous sections come together to produce the following algorithm:

```
Initialize the replay memory with size $N$
Initialize the Q-values with a standard deviation of 0.1
Between episode 1 and the total number of episodes, do:
    With probability $\varepsilon$ select a random action $a$
    Or select the argmax action, $argmax([a_0...a_{21}])$
    Reply to the system with the selected action, $a$
    Observe reward, $r$
    Store state ($s$), Q-values, action ($a$), reward ($r$), end of episode Boolean in the replay memory
    When the replay memory is full (size = $N$):
            Update Q-values in replay memory
            Sample random minibatches from the replay memory
            Calculate loss and optimize with RMSProp
            Reset replay memory
```

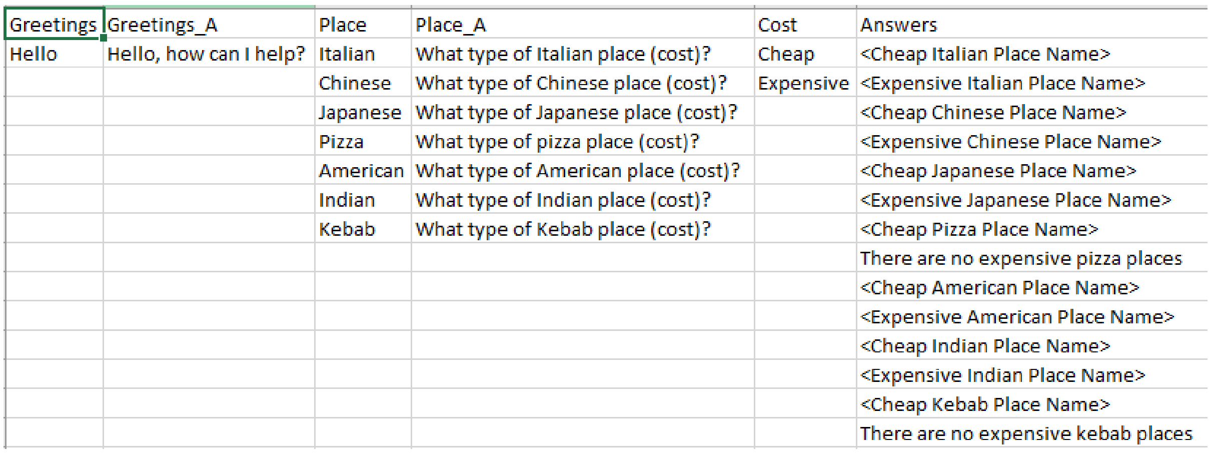
3. RESULTS AND ANALYSIS

3.1 Experimental Inputs

Section 3.1 entails the dataset, the machine and software versions used for the implementation and testing of the system, as well as the measurements that will be taken along with the factors that affect these measurements.

3.1.1 Dataset

As mentioned briefly in *Section 2.2.*1, the dataset was a custom-built excel CSV spreadsheet (*Figure 6)* consisting of 10 inputs and 22 answers. The 10 inputs will be transformed into vector arrays of size 2 (1 representing the current input, 1 representing the previous input – *Section 2.2.1*) to be input into the neural network. The 22 answers represent the actions the agent can take at any time as a response to a given state (input) – [A_0…A_{21}].



Greetings	Greetings_A	Place	Place_A	Cost	Answers
Hello	Hello, how can I help?	Italian	What type of Italian place (cost)?	Cheap	<Cheap Italian Place Name>
		Chinese	What type of Chinese place (cost)?	Expensive	<Expensive Italian Place Name>
		Japanese	What type of Japanese place (cost)?		<Cheap Chinese Place Name>
		Pizza	What type of pizza place (cost)?		<Expensive Chinese Place Name>
		American	What type of American place (cost)?		<Cheap Japanese Place Name>
		Indian	What type of Indian place (cost)?		<Expensive Japanese Place Name>
		Kebab	What type of Kebab place (cost)?		<Cheap Pizza Place Name>
					There are no expensive pizza places
					<Cheap American Place Name>
					<Expensive American Place Name>
					<Cheap Indian Place Name>
					<Expensive Indian Place Name>
					<Cheap Kebab Place Name>
					There are no expensive kebab places

Figure 6: Dataset

3.1.2 Architecture, Hyper-parameters and Set up

The architecture proposed for this system is a fully-connected multilayer neural network. It consists of an input layer, 2 hidden layers and an output layer. The input layer has an input size of 2, which is the size of a state (previous question and current question). The hidden layers have 20 nodes each and the output layer is of size 22 (the number of actions). The hidden layers use rectified linear units (ReLU) to normalise the weights (Nair and Hinton, 2010).

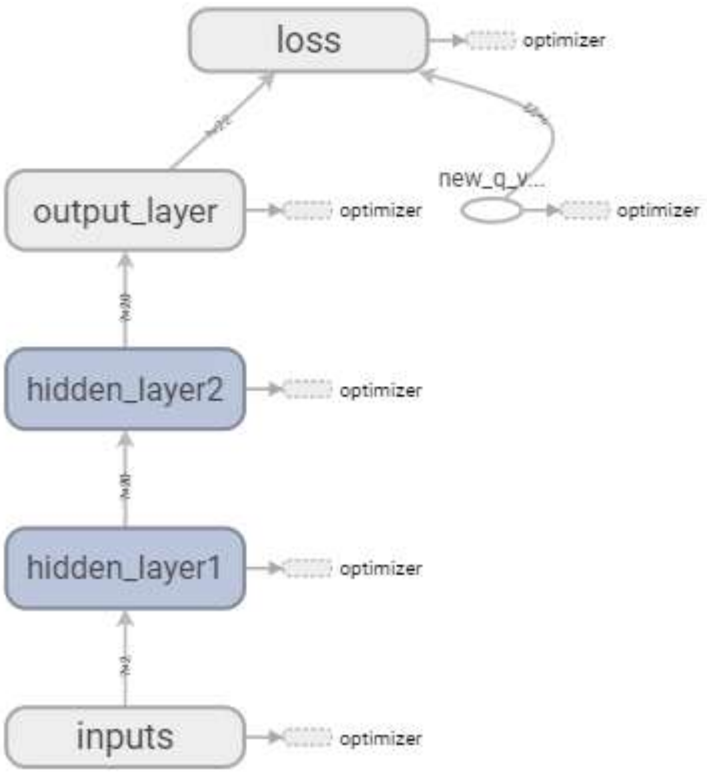


Figure 7: Data Flow of the Neural Network

The replay memory experience size was set to 1000 with a mini-batch size of 50. The replay memory was much smaller than that of the replay memory used by DeepMind while conducting their experiment on Atari games, due to the large amount of data they would need to store as a game would produce a high number of states, as discussed previously.

The number of simulated dialogs was 1,000,000 i.e. the system was run for 1,000,000 episodes; this corresponds to 3,000,000 states being generated by the system as 1 episode generates 3 states of the conversation. This number of simulated dialogs was much greater than that of SimpleDS; partly since SimpleDS uses a reduced action set (between 4-5 actions) for a given state. As discussed later in this section, the proposed system only uses a reduced action set 10% of the time and this reduced action set can still consist of up to 14 actions.

The Q-values were initialised using a standard deviation of 0.1. This randomly initialises the Q-values to around 0 – but the Q-values cannot be too low otherwise it made it more difficult to train the system, hence why this standard deviation was selected.

The discount factor was set to 0.95. This is higher than the discount factor selected by the SimpleDS paper. Experimentation with different values for the discount is left for future work, which is discussed in *Section 4.2*.

The epsilon during training was set to 0.1. This remained the same for the entire duration of training. This means there is a 10% chance to take a random action, otherwise the action with the highest Q-value is selected. This means the system may initially keep selecting the wrong answers, but this will be balanced out by the system estimating lower Q-values for these actions.

The selected learning rate was 0.01. Learning rates of 0.1 and 0.001 were experimented with also, 0.1 produced very high and distorted Q-values which eventually led to the same Q-value being estimated for every state; whereas the learning rate of 0.001 led to the Q-values being learned too slowly, meaning that even more training episodes would be required.

For optimizing the neural network and reducing the loss, RMSProp was selected.

3.2 Experimental Conditions

This section details the exact conditions under which the experiment was conducted.

3.2.1 Machine

The neural network was trained 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. This machine did not have the GPU version of TensorFlow installed.

3.2.2 Implementation

The system was implemented and tested using Windows 10 with the following software and library versions:

- Python 3.6
- Tensorflow 1.4.0
- Gensim 3.1.0
- NLTK 3.2.5
- Pandas 0.22.0
- NumPy 1.14.1

- argparse 1.1
- Matplotlib 2.1.2

3.3 Measurements

This section details the measurements that will be taken along with the factors which will influence these measurements.

These measurements are taken as they are used to evaluate the system's performance, as discussed within the individual sections. The Q-values, loss and reward will help to indicate some flaws within the system which may potentially lead to better solutions being found.

3.3.1 Q-Values

The Q-values are values that the agent estimates when it receives a state (question input) as input. The values tell the agent which action is likely to lead to the highest reward value i.e. the higher the Q-value the higher the estimated reward. The calculation for a Q-value for a given state is presented in more detail in *Section 2.3*.

The Q-values for this system will be the actions the system can take for a state. As shown by the dataset in *Section 3.1.1* the number of actions for an input is 22, therefore there is an array of 22 Q-values when a state is input representing each possible answer.

These 22 Q-values are initialised to around zero at the beginning of training and will be learned over the duration of training. At the end of training, the highest Q-value for a given state (question) should be the correct response to that question. Therefore, over the duration of training it is expected that the Q-values will initially lie close to 0, then over time the Q-values will spread out further away from 0 as the system finds correct answers (which gives a reward and therefore the system should estimate a higher Q-value). A graph of the Q-values will show how quickly these Q-values were learned.

3.3.2 Loss

The loss should be minimised during training and to do this the loss-function must be optimized. This is a measure of the error between the Q-values the neural network outputs and the target Q-values, which are the Q-values generated by sampling the replay memory and calculating the corresponding Q-values. This is further described in *Section 2.3*.

During training, the loss is measured to check if it is decreasing over time; as that would imply the neural network is estimating more correct Q-values i.e. there is less difference between estimated Q-values and the actual Q-values.

3.3.3 Reward

One of the most essential measurements to be taken is the reward value for each episode (full conversation). The reward is increased by one when a correct reply is given by the learning agent (*Section 2.4*). Therefore, a higher reward indicates a more successful conversation. As the agent does not know any of the correct answers at the beginning, the episode rewards should be low at the beginning and should gradually increase over time.

Towards the end of the training when and the system has explored the environment and figured out the correct answer to take for a given state the reward values should be high and, in many cases, reaching maximum value. To achieve a maximum value, the system will select the correct response for every state of the conversation that has been supplied to it for the duration of one conversation (3 states).

To more accurately view this, the mean reward for each 1000 episodes is plotted at the end of training so that this can be observed to how the system learns over the duration. At the beginning of training, the values should be around 0. At the end of training, the reward values per episode should often be 1, however the graph should not reach 1 as the average per 1000 episodes due to the epsilon factor. This will show the learning curve of the system.

3.3.4 Correctness of Learned Policy

The system will be tested after training is complete to ensure that the system selects the correct response for every state that it was tested with. This means it will be tested with every conversation sequence that the environment could generate. *Figure 1* is an example of one of the dialogues it will be tested against – it will be repeated for every other combination.

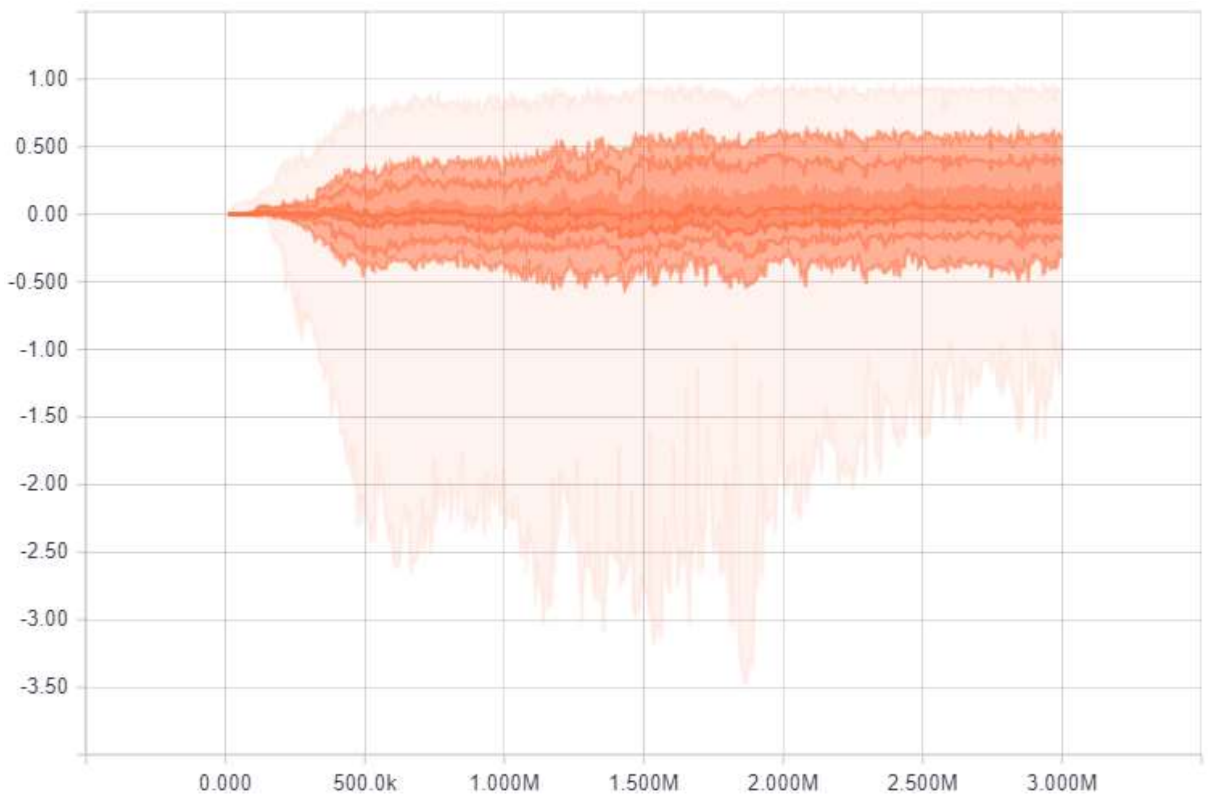
3.4 Results

Presented in the section below are the results for the measurements. The graphs are presented in this section and the significance of these results are analysed in *Section 3.5*.

The training sequence completed in 41.59 minutes.

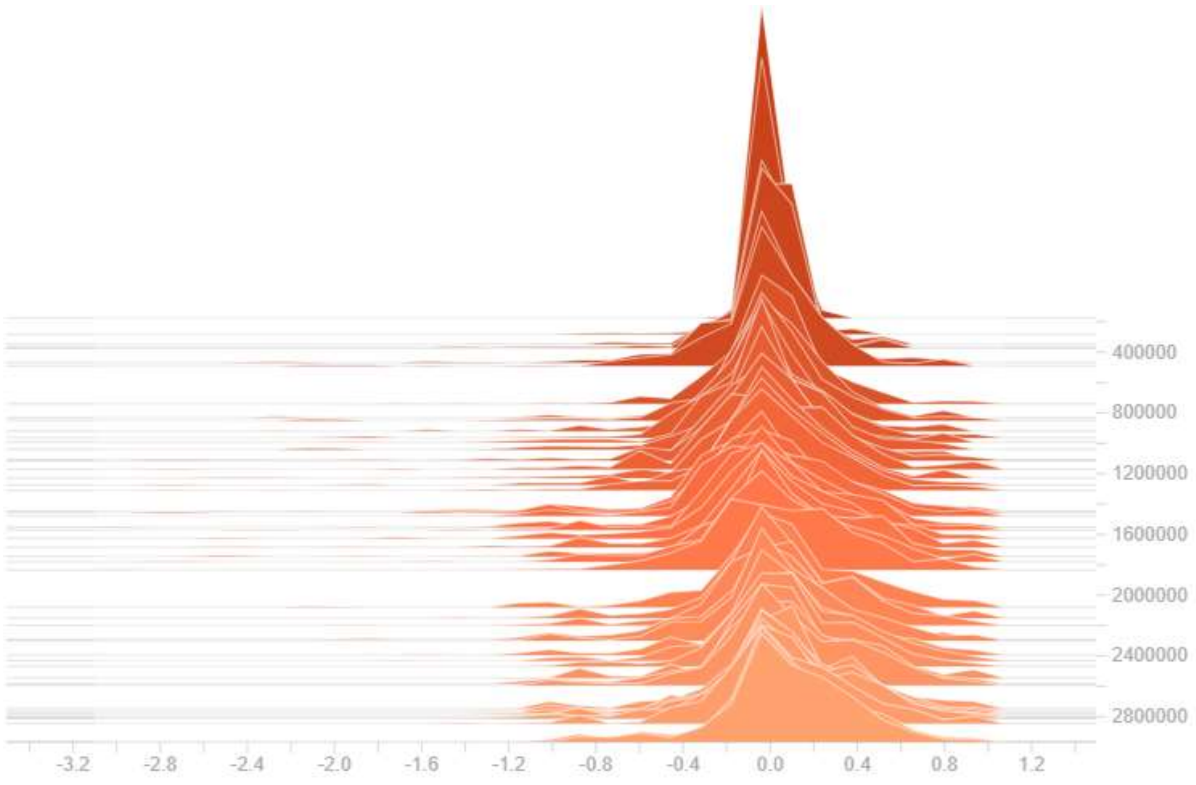
3.4.1 Q-Values

Using TensorBoard, the distribution of Q-Values was plotted:



Graph 1: Distribution of Q-Values over Training Iterations

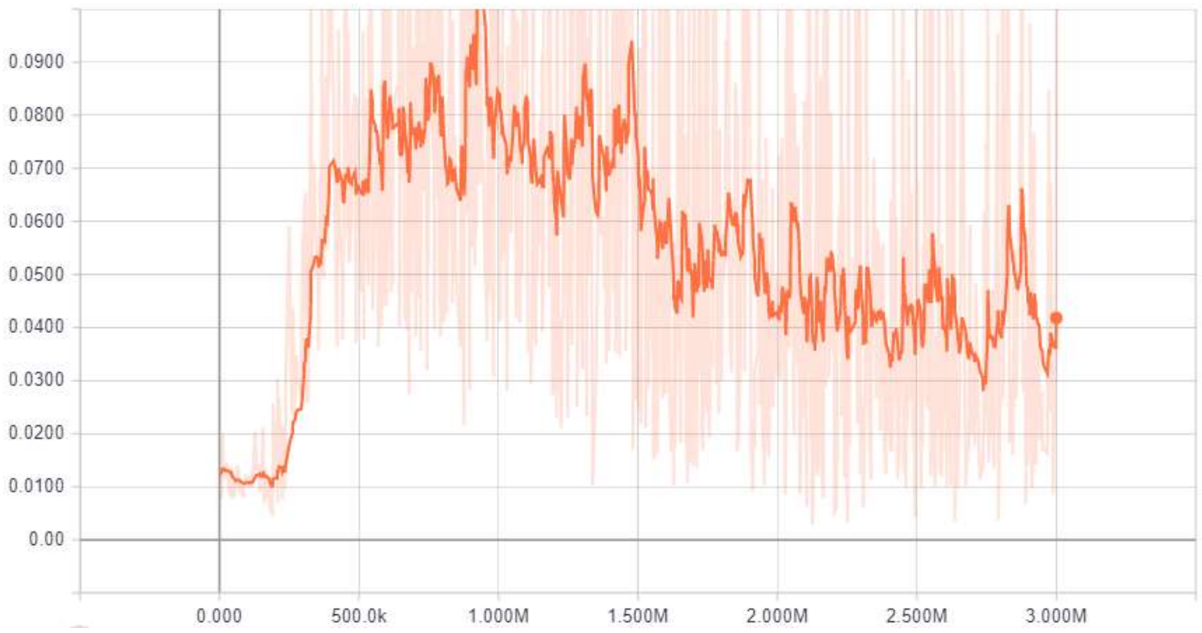
The following histograms were also generated. The darker elements at the back are the older training iterations, where the Q-value are initialised around 0; whereas the elements closer to the foreground are the episodes towards the end of training. The histogram function in TensorBoard shows how a tensor value has changed over time. Each slice represents one histogram, from an index chosen at random.



Graph 2: Histograms Showing Q-Value Distribution over Time

3.4.2 Loss

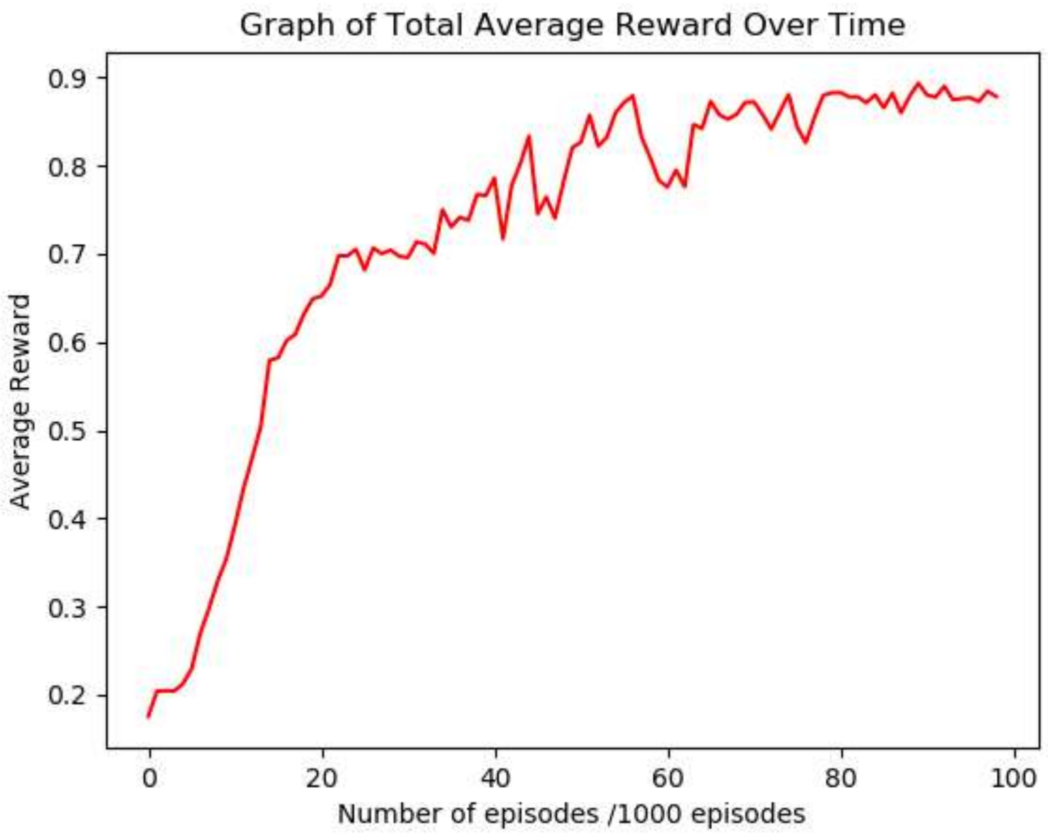
Using TensorBoard, a graph of loss over training steps was produced:



Graph 3: Loss During Training

3.4.3 Average Reward and Reward Statistics

Using Matplotlib, a graph of average reward over time was drawn. Average reward was the mean reward from the previous 10,000 episodes:



Graph 4: Average Reward per Episode

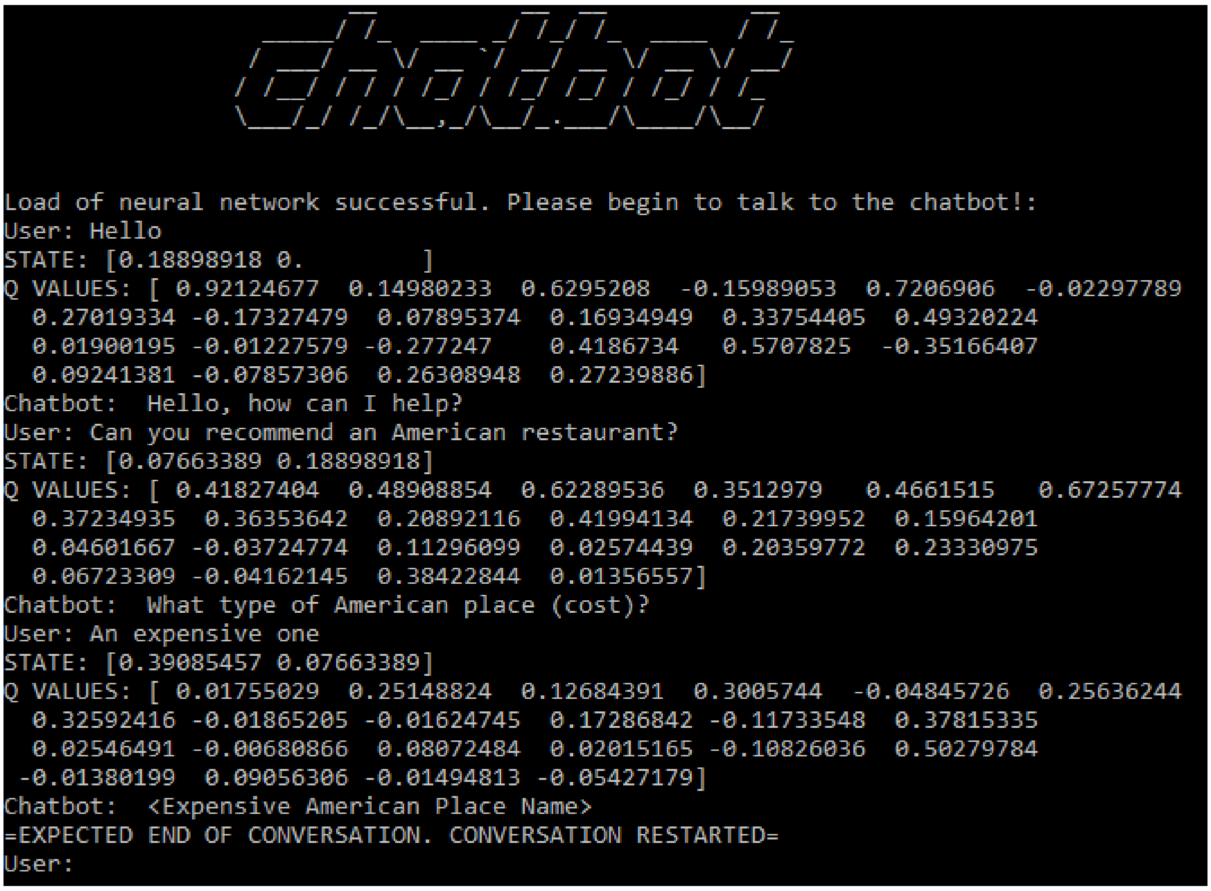
Additional reward statistics:

- The number of fully correct episodes was 614,278 (614,278/1,000,000).
- The first occurrence of a fully correct episode was episode 3966.
- The mean reward across all episodes was 0.734.

3.4.4 Correctness of Learned Policy

After training had completed, the system was tested to see if it selected the correct response for every input, as it had learned. The chatbot returned correct for 100% of conversations; tested on the possible input sequences that it could receive during training.

Here is an example dialogue from the policy learned from the training sequence above with the state and estimated Q-values printed.



```
                          __          __  __          __
                   _____/ /_  ____ _/ /_/ /_  ____  / /_
                  / ___/ __ \/ __ `/ __/ __ \/ __ \/ __/
                 / /__/ / / / /_/ / /_/ /_/ / /_/ / /_
                 \___/_/ /_/\__,_/\__/_.___/\____/\__/


Load of neural network successful. Please begin to talk to the chatbot!:
User: Hello
STATE: [0.18898918 0.        ]
Q VALUES: [ 0.92124677  0.14980233  0.6295208  -0.15989053  0.7206906  -0.02297789
  0.27019334 -0.17327479  0.07895374  0.16934949  0.33754405  0.49320224
  0.01900195 -0.01227579 -0.277247    0.4186734   0.5707825  -0.35166407
  0.09241381 -0.07857306  0.26308948  0.27239886]
Chatbot:  Hello, how can I help?
User: Can you recommend an American restaurant?
STATE: [0.07663389 0.18898918]
Q VALUES: [ 0.41827404  0.48908854  0.62289536  0.3512979   0.4661515   0.67257774
  0.37234935  0.36353642  0.20892116  0.41994134  0.21739952  0.15964201
  0.04601667 -0.03724774  0.11296099  0.02574439  0.20359772  0.23330975
  0.06723309 -0.04162145  0.38422844  0.01356557]
Chatbot:  What type of American place (cost)?
User: An expensive one
STATE: [0.39085457 0.07663389]
Q VALUES: [ 0.01755029  0.25148824  0.12684391  0.3005744  -0.04845726  0.25636244
  0.32592416 -0.01865205 -0.01624745  0.17286842 -0.11733548  0.37815335
  0.02546491 -0.00680866  0.08072484  0.02015165 -0.10826036  0.50279784
 -0.01380199  0.09056306 -0.01494813 -0.05427179]
Chatbot:  <Expensive American Place Name>
=EXPECTED END OF CONVERSATION. CONVERSATION RESTARTED=
User:
```

Figure 8: Example Dialogue from Learned Policy

3.5 Analysis of Results

Observing the graphs for Q-values, *Graph 1* shows the gradual spread of the Q-values around the mean of 0, where negative Q-values indicate the system not gaining a reward, whereas the positive values indicate the neural network learning correct responses and gaining a reward. However, this graph generated by TensorBoard is plotted from Q-values ranging from -3.5 to 1.0, showing that the neural network estimated very large negative Q-values even though the system does not issue a negative reward at any point. Observing the feinter shade of orange, over the duration of training, particularly in the middle, the system calculated very large negative Q-values, which began to smooth out towards the end; whereas the positive Q-values remained quite smooth throughout.

A better way to present the spread of Q-values learned over time is presented by the histograms in *Graph 2*, which displays histograms at random steps. The darker histograms further back show the Q-values from the earliest training iterations, where the Q-values were initialised around 0. Over time, the Q-values begin to spread out and towards the end particularly, many of the Q-values lie between 0 and 0.8.

Interestingly, the loss value increases over time then decreases; and it appears to form a graph vaguely similar to a log-normal distribution. This seems to be in line with the high negative Q-values which were previously discussed. This means that when the loss values are calculated, the minimum reward observed will be 0 and therefore the system is trying to calculate the loss between these large negative Q-values and zero, resulting in a high loss value.

Other reasons for these high, erratic error values might be: The learning rate may be too high for the system. A lower learning rate was not selected as it took too long for the system to learn the correct actions to take. Experimenting with a different optimizer may also help to improve the loss. Another reason may be the use of a reduced action set only when selecting a random action.

Graph 4 shows the reward over time for the system i.e. the learning curve for the system. A reward value of 1 for an episode implies that the agent replied correctly to the entire sequence in the conversation array generated by the environment. On the other hand, a reward of 0 indicates that the agent selected the incorrect reply every time. The reward function is further described in *Section 2.5*.

Another indication by *Graph 4* is that around the 600,000 episode mark the system may actually be adequately trained. However, after around 800,000 episodes the line for measuring reward is smoother.

Finally, and most importantly, the system selects the correct action for each input during testing corresponding to the inputs in the conversations CSV file. An example of this is presented in *Section 3.4.4*. This means the system successfully learned the dataset.

Based on the findings by *Graph 4*, the system was run again for only 600,000 episodes and the system successfully responded to all sequences (*Section 3.3.4*). The statistics for this system were as follows:

- The time taken was 24.92 minutes
- The number of fully correct episodes was 292,679 (292,679/600,000).
- The first occurrence of a fully correct episode was episode 23901.
- The mean reward across all episodes was 0.634.

The time has been significantly reduced to under 25 minutes for training, yet the policy was fully learned. This set-up was run again, and the policy was not fully learned (a few responses were incorrect); proving that 600,000 is not a stable enough number of episodes – perhaps around 750,000 would be suffice.

3.6 Comparisons to Existing Work

The best paper to compare work would be the SimpleDS paper (Cuayáhuitl, 2016), as the environment domain and the number of possible actions were relatively similar; the system in that paper along with the system in this paper both have relatively small neural network architectures too.

Comparing *Graph 4* (*Section 3.4.3*) with Figure 2 in the SimpleDS paper, the learning curve for both systems look relatively similar. Using Figure 2 in SimpleDS, it appears the system was trained for approximately 35 minutes, similar the time taken to train this system. Even though the number of actions and the vocabulary is larger in SimpleDS, SimpleDS uses a reduced action set (between 4 and 5 per state), whereas this system only uses a reduced action set roughly 10% of the time (*Section 2.5*), resulting in requiring lengthier training time.

4. CONCLUSIONS

4.1 Summary of Project Work and Findings

In this paper, a deep reinforcement learning dialogue agent which implements a Q-learning algorithm was used to learn how to chat to a user in the domain of recommending a restaurant. This dialogue agent was motivated by the idea that chatbots in the future should be created with little assistance from the developer i.e. less hand-coded learning elements. The only input to the model was a word in numerical vector format.

This system learned to select the correct responses to the user's input using trial-and-error learning by selecting actions when given a conversation generated by the environment. The system was trained for approximately 40 minutes to ensure that the agent selects the correct responses for conversation that the environment issues. It was also demonstrated that it is possible to learn this dataset in under 30 minutes. In the end the system was able to select a cheap or expensive restaurant for the user for 7 different types of restaurant, as well as greet the user.

However, when chatting to this, a user may feel that the dialogue system is giving an automated scripted response – and does not appear very human-like, particularly because the responses and inputs for this system are very fixed. It remains to be demonstrated whether reinforcement learning algorithms can resemble true human-like intelligence.

4.2 Future Work

Further work for this system and approach includes the following:

1. Experimentation with different hyper-parameters. This includes modifying the replay memory size, the discount rate, the epsilon greedy etc. This could lead to the system learning more quickly if better hyper-parametes are selected.

2. Comparing the results from this paper to agents with different model architectures. This includes the use of greater or fewer hidden layers or varying the size of the nodes within the hidden layers.

3. Using other forms of learning (Pietquin and Lopes, 2014) to implement the system and compare results.

4. Experimenting with different reward functions. This system does not actually penalize incorrect conversation. An example of a different reward function to implement would be to use a larger conversation size and use a maximum number of response steps that the agent can select an answer to a user's problem without getting it correct. If the number of steps exceeds this, issue a negative reward.

5. The use of another optimizer for the neural network might assist with training speed or stability; only RMSProp was experimented with as it is commonly chosen among previous research.

6. The use of a dataset with noise. This system used a dataset with no noise which will result in overfitting to the individual inputs.

7. The dataset used for this system was small. Further work should be done to train an agent in larger scale domains. A larger dataset will also produce a richer corpus for learning word vectors. An even better solution would be to implement an LSTM (Schmidhuber, 2017), as

LSTM is a desirable choice when it comes to sequences which have long term dependencies in it. LSTM is also a more beneficial choice because it may be possible for the system to assign the same word vectors to 2 different inputs, as the current architecture gets a vector sum of all the words in the sentence.

8. When optimizing the neural network, the batch values selected may not contain any sequence where the system achieved correct responses due to the random nature of selecting batch values. A proposed solution to this is estimating the error between the newly calculated Q-values and the Q-values estimated by the neural network. At the beginning of training, when a correct sequence occurs, it will notice that the Q-value has significantly increased, therefore replay memory indices with high-estimation errors could be added to the batch. This should significantly decrease the number of episodes required and therefore vastly increase the required training time. This could also be balanced by inputting some values with low-estimation error (Q-values it is good at estimating).

9. The time could be significantly reduced by using a true reduced action set per state. For example, when the user enters a type of food (e.g. Italian), the chatbot can only select the actions which reply, “What cost of X place?” where “X” corresponds to all the types of restaurants. This system only used a reduced action set when selecting a random action, which was a 10% chance. The implementation of this will require re-tuning of the hyper-parameters.

10. Some papers (Mnih et al., 2015) use a high epsilon value and decrease it over time whereas this paper uses a fixed epsilon value (0.1). The benefit of a high initial epsilon value is because it is better for exploring rather than the system becoming fixated on a wrong response for a long time. This was experimented with, but it did not prove fruitful; but this may have been due to the other hyper-parameters of the system.

11. This system did one optimization run every time the replay memory was full, whereas DeepMind did an optimization run and updated Q-values every step of the environment. The system may require less episodes and may learn quicker if the optimization was run every step in the game environment.

5. REFERENCES

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstre, D. and Riedmiller, M. (2013). *Playing Atari with Deep Reinforcement Learning*, [Online]. Available at: https://arxiv.org/pdf/1312.5602.pdf (Accessed: 19th April 2018).

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. and Hassabis, D. (2015). *Human-Level Control Through Deep Reinforcement Learning*, [Online]. Available at: https://www.nature.com/articles/nature14236 (Accessed: 19th April 2018).

Cuayáhuitl, H. (2016). *SimpleDS: A Simple Deep Reinforcement Learning Dialogue System*, [Online]. Available at: https://arxiv.org/pdf/1601.04574.pdf (Accessed: 19th April 2018).

Silver D. (2016). *Deep Reinforcement Learning*. [Online] DeepMind. Available at: https://deepmind.com/blog/deep-reinforcement-learning/ (Accessed 13th December 2017).

Su, M., Huang, K., Yang, T., Lai, K. and Wu, C. (2016). Dialog State Tracking and Action Selection Using Deep Learning Mechanism for Interview Coaching. In: *2016 International Conference on Asian Language Processing (IALP)*. [Online] Tainan: IEEE. Available at: https://ieeexplore.ieee.org/document/7875922/ (Accessed 19th April 2018).

Watkins, C. and Dayan, P. (1992). Machine Learning. *Q-Learning*, [Online] Volume 8(3-4), pp. 279-292. Available at: https://doi.org/10.1007/BF00992698 (Accessed 19th April 2018).

Britz, D. (2016). *Deep Learning for Chatbots, Part 1 – Introduction*. [Online] WildML. Available at: http://www.wildml.com/2016/04/deep-learning-for-chatbots-part-1-introduction (Accessed 31st January 2018).

Nair, V. and Hinton, E. G. (2010). *Rectified Linear Units Improve Restricted Boltzmann Machines*, [Online]. Available at: https://www.cs.toronto.edu/~hinton/absps/reluICML.pdf (Accessed 31st January 2018).

Ruizendal, R. (2017). *Deep Learning #4: Why You Need to Start Using Embedding Layers*. [Online] Available at: https://towardsdatascience.com/deep-learning-4-embedding-layers-f9a02d55ac12 (Accessed 26th March 2018).

Brownlee, J. (2017). *How to Use Word Embedding Layers for Deep Learning with Keras*. [Online] Available at: https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/ (Accessed 26th March 2018).

Mikolov, T., Chen, K., Corrado, G. and Dean, J. (2013). *Efficient Estimation of Word Representations in Vector Space*. [Online] Available at: https://arxiv.org/pdf/1301.3781.pdf (Accessed 15th March 2018).

Narkhede, P. (2017). *Sentence2Vec: Evaluation of popular theories — Part I (Simple average of word vectors)*. [Online] Medium. Available at: https://medium.com/@premrajnarkhede/sentence2vec-evaluation-of-popular-theories-part-i-simple-average-of-word-vectors-3399f1183afe (Accessed 4th April 2018).

Diuk, C., Cohen, A. and Littman, M. L. (2008). *An Object-Oriented Representation for Efficient Reinforcement Learning* [Online]. Available at: http://carlosdiuk.github.io/papers/OORL.pdf (Accessed 31st January 2018).

Serban, I. V., Sankar, C., Germain, M., Zhang, S., Lin, Z., Subraman, S., Kim, T., Pieper, M., Chandar, S., Ke, N. R., Rajeshwar, S., de Brebisson, A., Sotelo, J. M. R., Suhubdy, D., Michalski, V., Nguyen, A., Pineau, J. and Bengio, Y. (2017). *A Deep Reinforcement Learning Chatbot* [Online]. Available at: https://arxiv.org/pdf/1709.02349.pdf (Accessed 21st April 2018).

Nusca, A. (2017). The Most Popular Google Android and Apple iOS Apps of 2017. *Fortune*, [online]. Available at: http://fortune.com/2017/12/29/best-most-popular-apps/ (Accessed 15th February 2018).

Szepesvári, C. (2009). *Algorithms for Reinforcement Learning*, [Online] Volume 4(1), p. 2. Available at: https://sites.ualberta.ca/~szepesva/papers/RLAlgsInMDPs.pdf (Accessed 15th April 2018).

Schmidhuber, J. (2017). *Recurrent Neural Networks*. [Online] IDSIA. Available at: http://people.idsia.ch/~juergen/rnn.html (Accessed 15th April 2018).

Pietquin, O., Lopes, M. (2014). *Machine Learning for Interactive Systems: Challenges and Future Trends*, [Online]. Available at: http://www.cristal.univ-lille.fr/~pietquin/pdf/WACAI_2014_OPML.pdf (Accessed 29th April 2018).

Pederson, M. (2017). *Reinforcement Learning*. [Online] GitHub. Available at: https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/16_Reinforcement_Learning.ipynb (Accessed 2nd May 2018).

Sutton, R. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press.

Hinton, G., Srivastava, N. and Swersky, K. (2014). *Neural Networks for Machine Learning*. [Online] University of Toronto. Available at: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf (Accessed 2nd May 2018).

6. APPENDICES

6.1 Meeting Minutes

The following pages will contain the minutes from my meetings with my research project advisor, Vien Ngo.

QUEEN'S UNIVERSITY OF BELFAST
COMPUTER SCIENCE

Minute of Project Supervision Meeting

Student Name:	Keelan Graham-Sands		
Project Module Code:	CSC4006		
Project Supervisor:	Vien Ngo		
Meeting Number:	1	**Date of Meeting:**	24/10

Progress since last meeting, and decisions arrived at during meeting:

- Arranged meeting schedule, Tuesday bi-weekly and discussed project background

Action Points:

- Finish the courses and read the background information
- Look into Python and Tensorflow
- Make a start on my "Research Topic Report"

Date of next meeting:

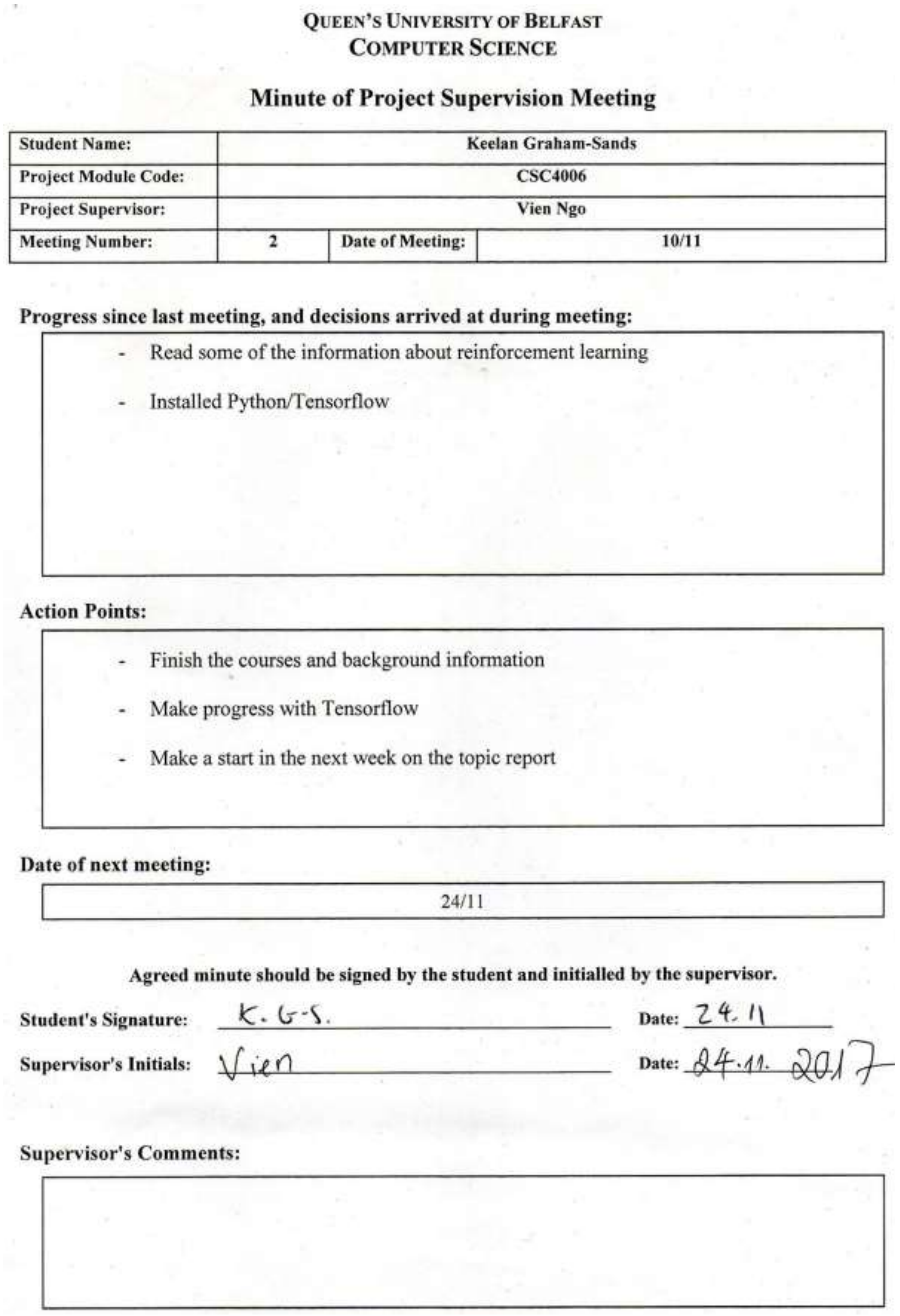
7/11

Agreed minute should be signed by the student and initialled by the supervisor.

Student's Signature: K. G. S. **Date:** 10/11

Supervisor's Initials: Vien Ngo **Date:** 10/11/2017

Supervisor's Comments:



QUEEN'S UNIVERSITY OF BELFAST
COMPUTER SCIENCE

Minute of Project Supervision Meeting

Student Name:	Keelan Graham-Sands		
Project Module Code:	CSC4006		
Project Supervisor:	Vien Ngo		
Meeting Number:	2	**Date of Meeting:**	10/11

Progress since last meeting, and decisions arrived at during meeting:

- Read some of the information about reinforcement learning
- Installed Python/Tensorflow

Action Points:

- Finish the courses and background information
- Make progress with Tensorflow
- Make a start in the next week on the topic report

Date of next meeting:

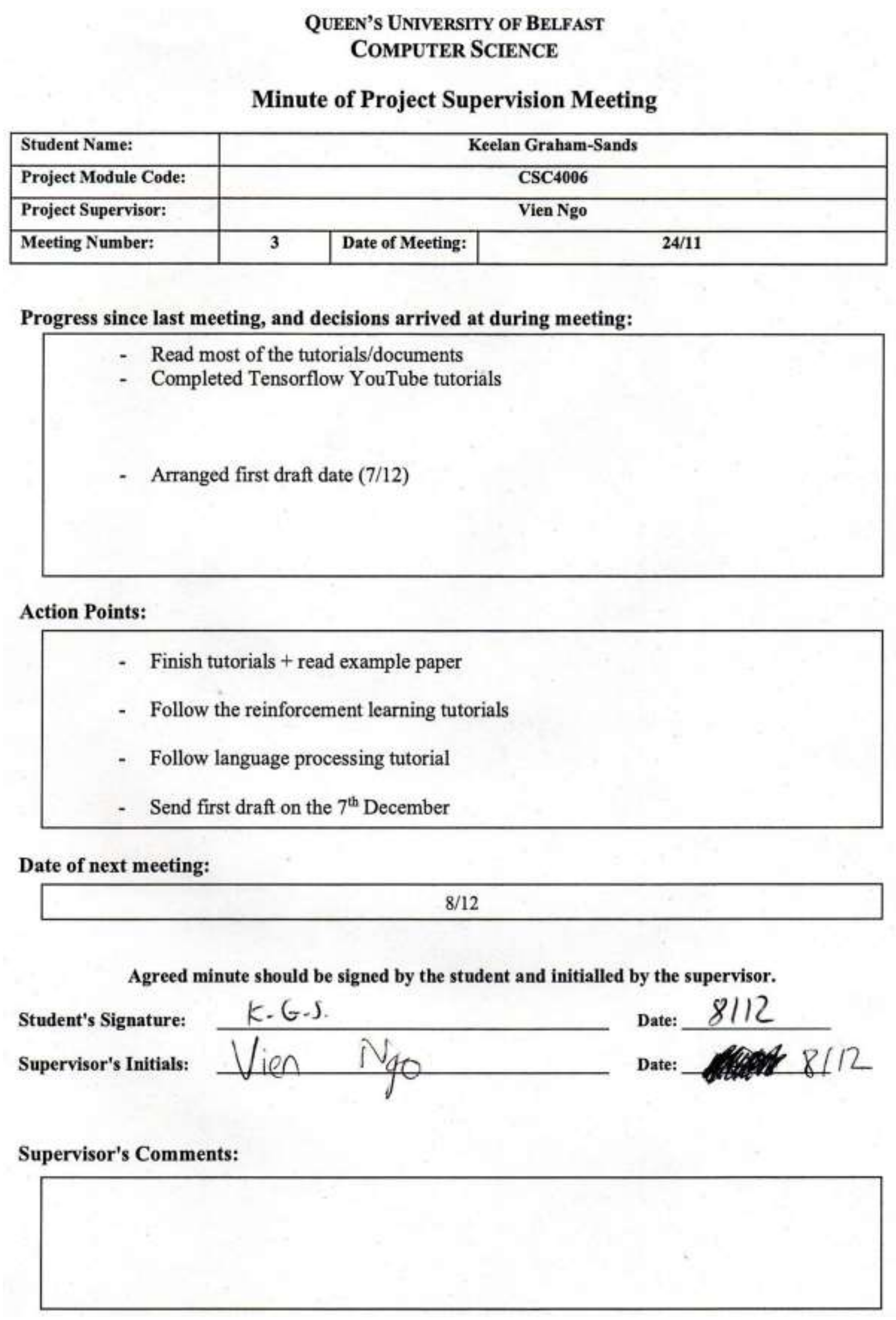
24/11

Agreed minute should be signed by the student and initialled by the supervisor.

Student's Signature: K. G-S. **Date:** 24.11

Supervisor's Initials: Vien **Date:** 24.11. 2017

Supervisor's Comments:



QUEEN'S UNIVERSITY OF BELFAST
COMPUTER SCIENCE

Minute of Project Supervision Meeting

Student Name:	Keelan Graham-Sands		
Project Module Code:	CSC4006		
Project Supervisor:	Vien Ngo		
Meeting Number:	3	**Date of Meeting:**	24/11

Progress since last meeting, and decisions arrived at during meeting:

- Read most of the tutorials/documents
- Completed Tensorflow YouTube tutorials

- Arranged first draft date (7/12)

Action Points:

- Finish tutorials + read example paper
- Follow the reinforcement learning tutorials
- Follow language processing tutorial
- Send first draft on the 7th December

Date of next meeting:

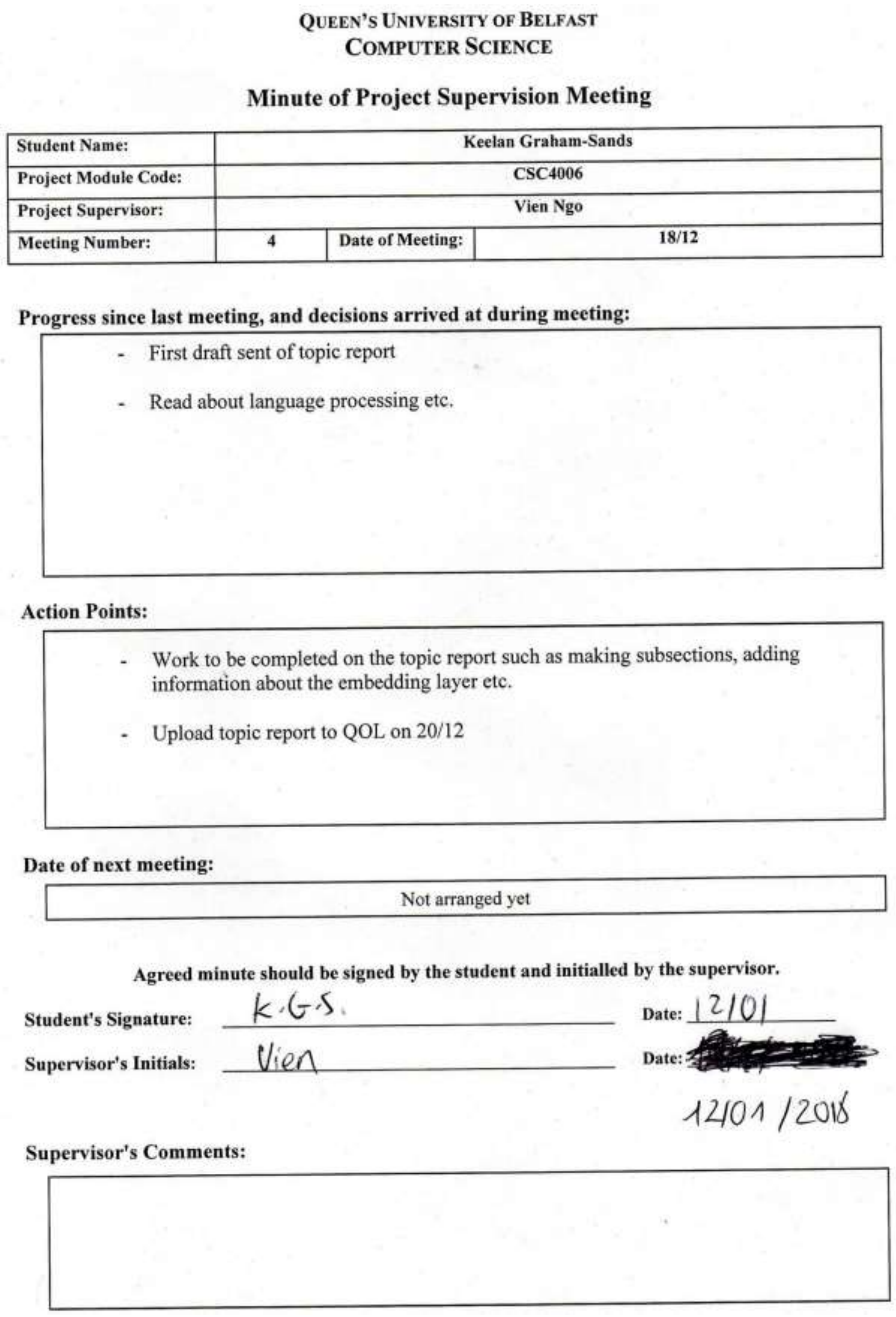
8/12

Agreed minute should be signed by the student and initialled by the supervisor.

Student's Signature: K.G.S. **Date:** 8/12

Supervisor's Initials: Vien Ngo **Date:** [illegible] 8/12

Supervisor's Comments:



QUEEN'S UNIVERSITY OF BELFAST
COMPUTER SCIENCE

Minute of Project Supervision Meeting

Student Name:	Keelan Graham-Sands		
Project Module Code:	CSC4006		
Project Supervisor:	Vien Ngo		
Meeting Number:	4	**Date of Meeting:**	18/12

Progress since last meeting, and decisions arrived at during meeting:

- First draft sent of topic report
- Read about language processing etc.

Action Points:

- Work to be completed on the topic report such as making subsections, adding information about the embedding layer etc.
- Upload topic report to QOL on 20/12

Date of next meeting:

Not arranged yet

Agreed minute should be signed by the student and initialled by the supervisor.

Student's Signature: K.G.S. **Date:** 12/01

Supervisor's Initials: Vien **Date:** [illegible]

12/01/2018

Supervisor's Comments:

Queen's University of Belfast
Computer Science

Minute of Project Supervision Meeting

Student Name:	Keelan Graham-Sands		
Project Module Code:	CSC4006		
Project Supervisor:	Vien Ngo		
Meeting Number:	5	**Date of Meeting:**	12/01

Progress since last meeting, and decisions arrived at during meeting:

- Topic report submitted

Action Points:

- Discussed where to progress first with the system, and what date the initial system should be created

Date of next meeting:

Not arranged yet

Agreed minute should be signed by the student and initialled by the supervisor.

Student's Signature: K.G.S. **Date:** 01/2

Supervisor's Initials: Vien **Date:** 01/02/2018

Supervisor's Comments:

QUEEN'S UNIVERSITY OF BELFAST
COMPUTER SCIENCE

Minute of Project Supervision Meeting

Student Name:	Keelan Graham-Sands		
Project Module Code:	CSC4006		
Project Supervisor:	Vien Ngo		
Meeting Number:	6	**Date of Meeting:**	01/02

Progress since last meeting, and decisions arrived at during meeting:

- Received feedback on topic report
- Started looking at environment and asked for advice

Action Points:

- Continue to work on the project

Date of next meeting:

Not arranged yet

Agreed minute should be signed by the student and initialled by the supervisor.

Student's Signature: K-G-S. **Date:** 15/2

Supervisor's Initials: Vien **Date:** 15/2/2018

Supervisor's Comments:

QUEEN'S UNIVERSITY OF BELFAST
COMPUTER SCIENCE

Minute of Project Supervision Meeting

Student Name:	**Keelan Graham-Sands**		
Project Module Code:	**CSC4006**		
Project Supervisor:	**Vien Ngo**		
Meeting Number:	7	**Date of Meeting:**	**15/02**

Progress since last meeting, and decisions arrived at during meeting:

- Made some progress, however ran into some complications and discussed them in the meeting

Action Points:

- Talked in detail about the structure such as the environment and how to allocate reward
- Continue to work on the project

Date of next meeting:

Not arranged yet (proposed 1/03)

Agreed minute should be signed by the student and initialled by the supervisor.

Student's Signature: K.G.S. **Date:** 8/3

Supervisor's Initials: Vien Ngo **Date:** 8/3/2018

Supervisor's Comments:

Queen's University of Belfast
Computer Science

Minute of Project Supervision Meeting

Student Name:	Keelan Graham-Sands		
Project Module Code:	CSC4006		
Project Supervisor:	Vien Ngo		
Meeting Number:	8	**Date of Meeting:**	8/03

Progress since last meeting, and decisions arrived at during meeting:

- Initial system almost completed however just need to complete the agent talking with the user simulator – discussed how to implement

Action Points:

- Finish the project and then discuss future work

Date of next meeting:

15/03

Agreed minute should be signed by the student and initialled by the supervisor.

Student's Signature: K.G.S **Date:** 8/3

Supervisor's Initials: V **Date:** 8/3

Supervisor's Comments:

QUEEN'S UNIVERSITY OF BELFAST
COMPUTER SCIENCE

Minute of Project Supervision Meeting

Student Name:	Keelan Graham-Sands		
Project Module Code:	CSC4006		
Project Supervisor:	Vien Ngo		
Meeting Number:	9	**Date of Meeting:**	15/03

Progress since last meeting, and decisions arrived at during meeting:

- System implemented, however it actually gets worse over time as opposed to learning the correct action to take – discussed solutions to this problem. It appears the same Q-Values are estimated for every state

Action Points:

- Fix issue with project

Date of next meeting:

22/03

Agreed minute should be signed by the student and initialled by the supervisor.

Student's Signature: K. G. S. **Date:** 28/3

Supervisor's Initials: V **Date:** 28/3

Supervisor's Comments:

QUEEN'S UNIVERSITY OF BELFAST
COMPUTER SCIENCE

Minute of Project Supervision Meeting

Student Name:	**Keelan Graham-Sands**		
Project Module Code:	**CSC4006**		
Project Supervisor:	**Vien Ngo**		
Meeting Number:	**10**	**Date of Meeting:**	**28/3**

Progress since last meeting, and decisions arrived at during meeting:

- Issue with code not fixed yet – issue is that the neural network estimates the same Q-values for every state
- Made progress with report
- Discussed some parts of the report and detail required
- Asked about some citing examples

Action Points:

- Fix issue with project
- Continue to work on research dissertation

Date of next meeting:

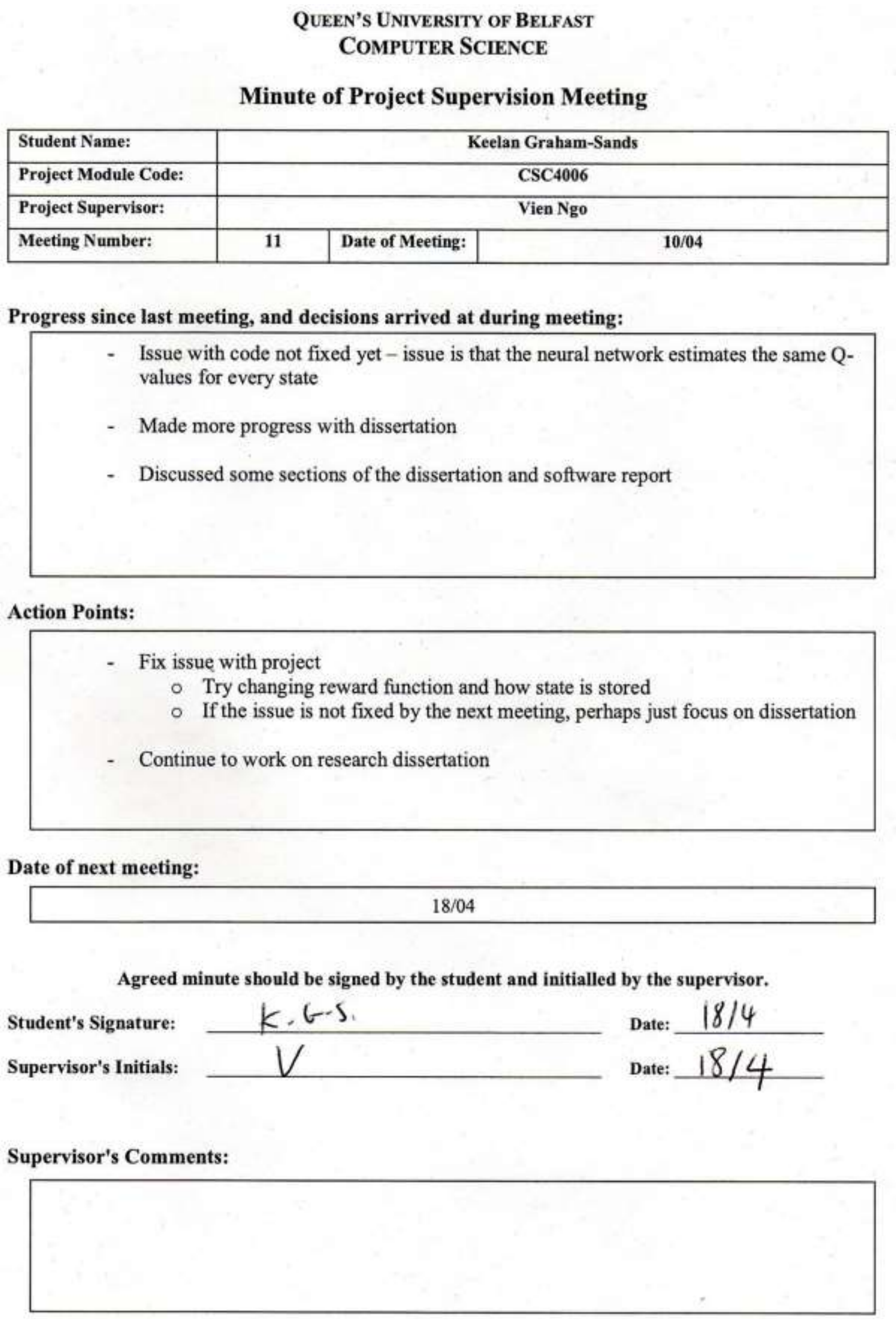
Not arranged yet – after Easter break

Agreed minute should be signed by the student and initialled by the supervisor.

Student's Signature: K.G.S. **Date:** 10/4

Supervisor's Initials: V **Date:** 10/4/2018

Supervisor's Comments:



QUEEN'S UNIVERSITY OF BELFAST
COMPUTER SCIENCE

Minute of Project Supervision Meeting

Student Name:	Keelan Graham-Sands		
Project Module Code:	CSC4006		
Project Supervisor:	Vien Ngo		
Meeting Number:	11	**Date of Meeting:**	10/04

Progress since last meeting, and decisions arrived at during meeting:

- Issue with code not fixed yet – issue is that the neural network estimates the same Q-values for every state
- Made more progress with dissertation
- Discussed some sections of the dissertation and software report

Action Points:

- Fix issue with project
  - Try changing reward function and how state is stored
  - If the issue is not fixed by the next meeting, perhaps just focus on dissertation
- Continue to work on research dissertation

Date of next meeting:

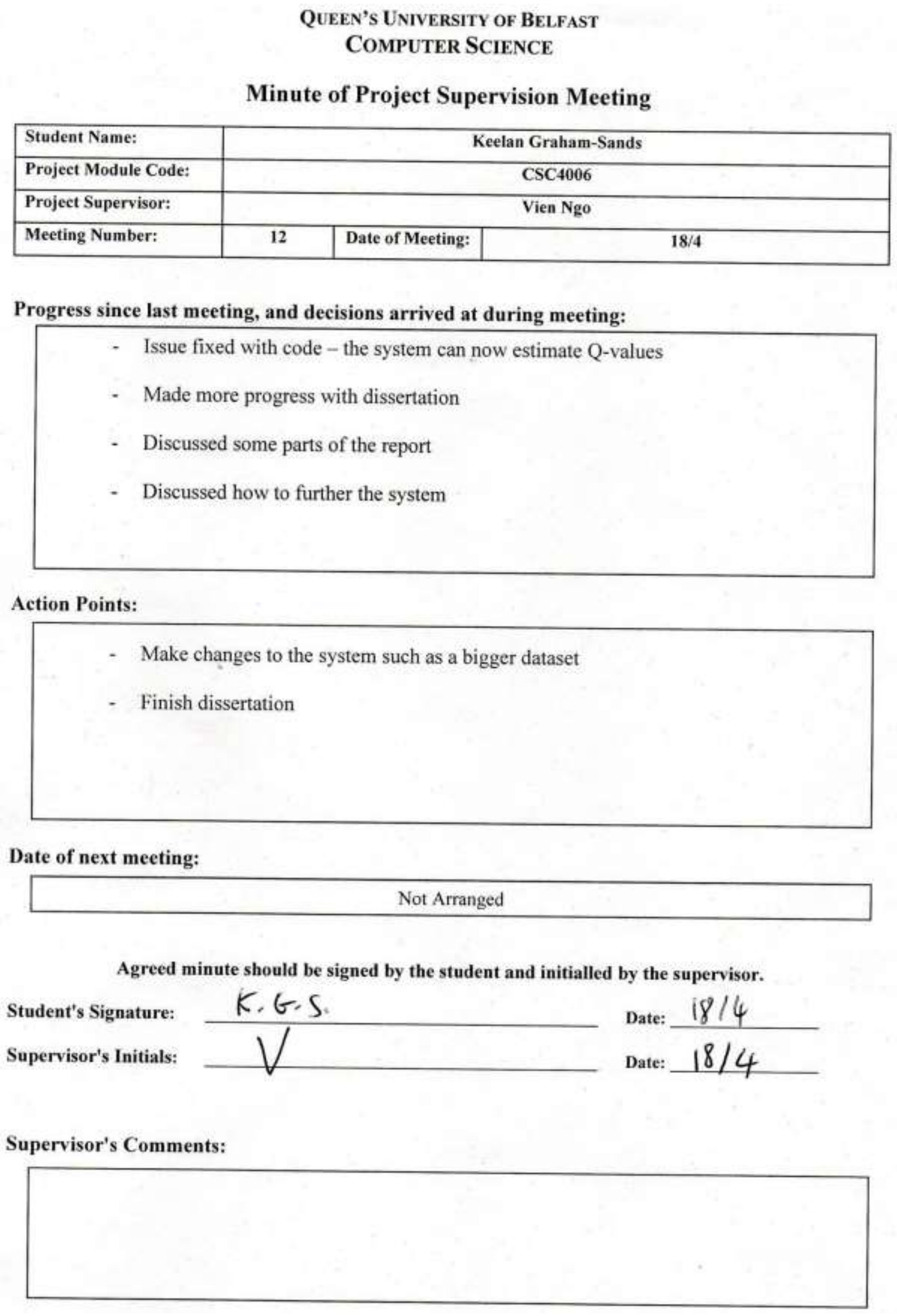
18/04

Agreed minute should be signed by the student and initialled by the supervisor.

Student's Signature: K. G-S. **Date:** 18/4

Supervisor's Initials: V **Date:** 18/4

Supervisor's Comments:



Queen's University of Belfast
Computer Science

Minute of Project Supervision Meeting

Student Name:	Keelan Graham-Sands		
Project Module Code:	CSC4006		
Project Supervisor:	Vien Ngo		
Meeting Number:	12	**Date of Meeting:**	18/4

Progress since last meeting, and decisions arrived at during meeting:

- Issue fixed with code – the system can now estimate Q-values
- Made more progress with dissertation
- Discussed some parts of the report
- Discussed how to further the system

Action Points:

- Make changes to the system such as a bigger dataset
- Finish dissertation

Date of next meeting:

Not Arranged

Agreed minute should be signed by the student and initialled by the supervisor.

Student's Signature: K.G.S. **Date:** 18/4

Supervisor's Initials: V **Date:** 18/4

Supervisor's Comments: