Natural Language Processing and Deep Learning for Games.

IMAT-5234 Applied Computational Intelligence

Dinis Marques Firmino
MSc Intelligent Systems
Centre of Computational Intelligence
De Montfort University, The Gateway, Leicester, LE1 9BH

***Abstract*—** Natural language is the most intuitive form of communication between human beings. However, due to the complex and abstract nature of language, the challenge to enable machines to exhibit the same reasoning capabilities as a human when handling language, proposes large obstacles which separate the capabilities of AI today, and those of a artificial general intelligence (AGI). This project is aimed at exploring the possibilities of creating game simulations which have intelligent inhabitants that can be interacted with using natural language. The idea of one day being able to experience such interactive worlds with the combined effect of VR, motion controllers for hand movement tracking and the ability to use voice as a means of interaction, would lead to incredibly immersive experiences, akin to real life.

Keywords— Deep Learning, Recurrent Neural Network, LSTM, Seq2Seq, NLP, Conversation Systems, Sentiment Analysis,

I. Introduction

My mini project is focused on researching the possibilities of integrating NLP and Deep learning techniques into interactive environments such as video games, in order to enable a more natural form interaction between players and game agents.
This experience will be incredibly immersive and, in a more direct sense, could potentially greatly benefit those who suffer from motor movement disabilities and can't use traditional input methods.
Recent advances in the field of Deep learning and natural language processing (NLP) have made it possible for machines to achieve superhuman level performance in particular domains. A good example being IBM's Watson in the game Jeopardy!, a complex trivia question based game in which it beat the world champion. The use cases of such techniques span many applications such as voice assistants, machine translation and text mining.
By no means is the project aimed at creating a full demonstration of such system, let alone see it fully integrated into a game engine, as that would require many hours of development time which fall outside the scope of this project.
The focus will be on researching and creating small demos of individual techniques such as conversational agents and sentiment analysis using NLP and various machine learning techniques. However, there is some work demonstrating NLP text processing techniques as a means of giving simple voice commands to a game agent, however this relies on more traditional methods, and was mainly for practicing core NLP techniques used in advanced systems.

II. Research

This section will cover all the topics researched during this project, and aims to explain in detail, the functionality of the techniques demonstrated in the experimental work covered in the next section. The research undertaken is on 3 main topics, natural language text processing, sentiment analysis and conversational agents (chatbots, dialog systems). These topics together are responsible for a great portion of the research being done in the field of natural language processing.
In addition, research into the technical challenges faced when integrating such techniques into a game environment will be included to further support current state of the experimental work.

A. Natural Language Processing

As a primer to the main research and development areas, initial research was focused on the various pre-processing techniques used throughout this project to ensure data was in the right format to be fed into the machine learning algorithms. Other core NLP techniques such as part-of-speech tagging, chunking and named entity recognition were also explored and used in the experimental work.

Sentence and word tokenization are the key first step in text processing. Text, in most scenarios, is received as paragraphs. However, most NLP techniques handle only individual words as features. So the need to pre-process incoming text into individual sentences and then further into words/tokens for

each sentence is mandatory.
Using the NLTK framework, we use the word_tokenize() function to split sentences into individual words.
Part of speech tagging (POS tagging) techniques were also experimented with using the NLTK framework. POS tagging is the grammatical tagging of words (nouns, verbs, adjectives) in a sentence based on their definition and context, i.e the relationship with other adjacent words. The end result is usually a tuple array of the word and its POS tag. This a preliminary technique for further text parsing techniques.
Chunking can only be performed after POS tagging and essentially is a higher level representation of text and a means of information extraction by grouping together various words by their POS tags, which together may represent a single entity. This is usually performed using regular expressions to parse the sentences and find related terms.
Finally, named entity recognition is another technique which could be performed using chunking, but many other techniques exist such as those in Stanford NER tagger. The point is to correctly identify entities in a given text, i,e people, organizations, locations… This information could be used to identify who and what the text is referring to.

B. Sentiment Analysis

Sentiment analysis (SA) also known as Opinion Mining (OM), is an area of research within natural language processing that focuses on the understanding of emotion expressed through text, whether positive, neutral or negative.
The vast growth of communication on the internet enabling large numbers of users to have conversations and express feelings towards various different entities such as organisations, sports, movies, people as opened much potential for many machine learning techniques, such as sentiment analysis. The value in this data is what makes sentiment analysis an important field in NLP and machine learning, as it helps obtain an overall picture of what people like and dislike which could be used by companies for market analysis, which can be for the purpose of knowing which products to sell or improve customer experience.
Capturing humans emotions such as when someone is "Angry", "Sad", "Happy", etc, can be exploited not just in the sense of text mining, but to trigger particular behaviours in a game agent akin to human emotional reactions.
I wanted to investigate whether it would be feasible to incorporate such techniques into a game environment to add another layer of realism to complement the voice interactions even further.
Sentence level sentiment classification, explored in [1], would be the most suited for the kinds of interactions in a game, as in most cases, a single sentence is enough to communicate enough information to the game agent.
The techniques used to classify sentiment correctly range from more basic bag of words models, to deep learning method utilizing advanced techniques such as word embeddings and LSTM neural networks.

C. Conversation Agents

The main area of research in this project was to investigate intelligent conversational agents and their potential use in game environments. We will cover the current state of the art and the most popular machine learning models for language modelling.

Games today, and for some time, will rely on hand-crafted dialog systems as a means of interaction between the player and game agents, environments or user interfaces. While progress is being made towards a more intuitive experience using traditional dialog systems, some major challenges that we will discuss later, hold state of art NLP and Deep learning techniques seen in real-world applications of conversational agents, from being implemented in video games.

A conversational agent is a application which can intelligently understand and reply to a received query, usually within a given domain.
Early systems relied on rule-based systems and hand-crafted responses to queries and basic pattern matching algorithms would be used to select the appropriate response. However, creating rules was extremely time consuming and these systems were unfit for handling complex or "unscripted" queries.
Machine learning lies at heart of most conversational agents today, which enable greater degrees of flexibility and the ability to be trained on large corpuses of words such as real conversations between humans. A good example is the Ubuntu Dialog Corpus[2]. Many large companies are making big bets on such systems, good examples of systems with narrower domains are Slack's chatbot which can assist users with system related queries, or Facebook's messenger bot that can handle domain specific queries.
More advanced commercial examples are Google's Assistant, Apple's Siri, Amazon's Alexa, which differ from the previous systems by being flexible in the responses they give and generally can respond quite accurately within the context of many domains and not restricted to a single domain. These are usually trained with various vast datasets of organised data.

Retrieval-based and Generative

Conversation bots fall under two categories, retrieval-based and generative. Retrieval-based models use a corpus of predefined responses which get selected based on an input i.e sentence and context. The responses can be selected by using a simple rule-based system like traditional systems, or machine learning techniques such as deep learning. The key difference

is that these models don't generate any new text that wasn't in the training corpus.
Generative models such as the system proposed in[3], are the polar opposite and don't rely on predefined responses. These models tend to be based around machine translation techniques, but instead of translating from one language to another, the translation is between the input text and an output response. There are pros and cons to both approaches. Retrieval-Based models don't make grammatical mistakes due to being restricted to responses in the training data, but for the same reasons are unable to handle unseen data i.e text for which there is no response to.
Generative models are smarter by generating an output response word by word based on the input. In contrast to retrieval-based, they are prone to making grammatical mistakes and are more difficult to train. Once trained, they outperform retrieval-based models both in terms of accuracy at responding to unseen data, and producing more natural conversations. Much of the research is being made towards improvement of generative models and further details of the sort of problems are covered in the next section.

Challenges

There are many challenges associated with achieving a full NLP system implementation into a game. This is down to a simple fact that when you incorporate a natural means of communication such as language into a game, it is second nature for humans to expect the same level of realism as in the real world. If the experience isn't on par, then it may make the whole experience less enjoyable, and a more traditional form of input preferred.
The difficulty of integration is also heavily influenced by the genre of game, for example, open world story based games would be a tough challenge as there will be many agents to interact with and the game needs to stick to a particular narrative/s, which means the outcomes of conversations with other game agents will have to meet certain constraints. This is unless of course we reach a point where systems are so intelligent that given a world theme, personalities and context awareness capabilities, simulations at this point will be akin to real life (we are far from achieving this).

Context awareness

One obstacle faced in games would be to incorporating the context of the game. Conversations in real life are largely influenced by both linguistic and physical context, the same can be said for a game. Physical context would be how the agent's surroundings would affect its responses, and linguistic context is having some sort of persistent knowledge during the exchange of information in a conversation.
If this isn't considered, then generating sensible responses is close to impossible. Work in [3], is moving in the direction of context aware systems by demonstrating a state of the art generative model that uses recurrent neural networks as a means of storing the context in the hidden layers of the network.

Personalities

Another huge factor to consider is embedding personalities into dialog systems. This is important because it has a profound impact on the interaction, and will ultimately make such systems far more realistic, by ensuring each game agent is "different" in how it handles more personal questions. Recent work in [4], presents a persona-based model that can incorporate a personality and affect the response generation, enabling it to respond to correctly to questions such as "Where do you live?", "Where were you born?", "How old are you?" etc…

Recurrent Neural Networks

This section aims to cover the technical aspect of conversation models by describing the underlying machine learning techniques that are being researched to tackle problems in computational linguistics and NLP.
Recurrent Neural Networks (RNNs) are a form of deep learning, and a variation of general feed-forward neural networks capable of dealing with sequential data such as, but not limited to, text sequences. The term recurrent is because the same task is repeated on each element of a sequence and each prediction is usually impacted by a previous element in the sequence, i.e a word.
Unlike Multi-Layered Perceptron or Convolutional Neural Networks, RNNs have an internal state which essentially represents the internal memory of the network. In the context of processing a sentence, the network will analyse a single word at each timestep and update its internal state with new information in order to determine the meaning of the complete sentence.

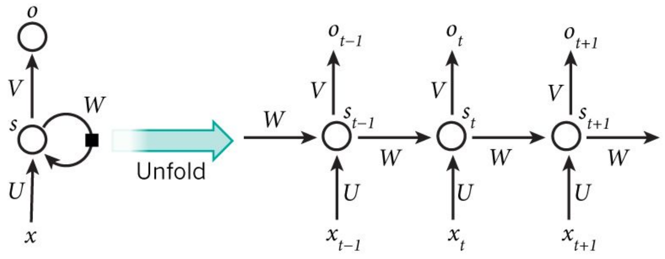


Image from [5]

The above diagram shows the layers of an RNN. Typically, there is a layer for each element in a sequence, i.e for a sequence of words, each word will represent a layer. Below is a brief description of what the notation on the diagram represents.

- x_t = Represents the input at timestep t. The input is typically a numerical representation of a word.

- s_t = Represents the hidden state of the neuron at time step t. This is essentially the memory of the network and is calculated using the previous hidden state s_t, and input x_t at the current timestep, and an activation function such as tanh or ReLU.
- o_t = Represents the output at timestep t. For predicting the next word, this would typically be in the form of a vector of probabilities across the word vocabulary.

There are many types of RNNs, but the most common one being used is LSTM (Long Short-Term Memory) networks. They are the same as barebones RNN's, however due to the changes in how the hidden state is computed, are better at capturing long term information. They are very popular for tasks such as language modelling, machine translation and speech recognition.

Seq2Seq

There are many different RNN architectures in use today, however for the purpose of this report, a model called Seq2Seq (Sequence to Sequence) introduced in [6], will be the focus as it is very popular in tackling problems in machine translation and dialog systems, and is also used in the experimental work.

It is made up of two stacked RNN's, an encoder and a decoder. The encoder is responsible for processing a sequence of words in an input sentence. It processes a single word at each timestep until all words are processed. A fixed length feature/thought vector is produced containing only the important information in the sequence such as context.
The image below depicts the flow of information at each timestep through the Seq2Seq model of LSTMs.

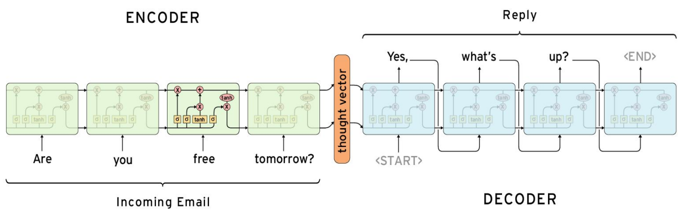


Image from [7]

The thought vector generated from the input sequence containing the intention/context of the previous sequence, is used by the decoder to generate a new output sequence of words. It contrasts from the encoder in that it is influenced by the thought vector generated by the previous words.

Like every machine learning model, there are limitations which need to be addressed during implementation. The biggest issue with Seq2Seq and stacked RNN networks is that they are poor at dealing with sequences of different sizes i.e size mismatch between input and output sequences. This is a big problem because in most cases, sequences will vary in lengths.
Vocabulary size can also have a large impact during decoding as the network will have to process large amount of words to choose the right one. This can have a massive impact on training times, even on capable hardware. To solve these issues data needs to undergo pre-processing. These will be covered in the development section.

III. Tools & Frameworks

A. *Rendering application*

To develop a simple demonstration of NLP being used in an interactive environment, a simple OpenGL rendering engine was used, built using C++. Due to most machine learning frameworks used in this project being available for high level programming languages such as Python, an interface between the C++ application and Python scripts needed to be considered. While not the most important for a demonstration system such as the one aimed for at the completion of this mini-project, understanding how such system come together is important.

B. *Boost.Python*

Boost.Python is the most robust and active projects around for embedding Python within C++ code. The wide availability of information made it the framework of choice for this problem.

C. *NLTK*

In any natural language processing task, the need to pre-process text before applying more advanced techniques such as machine learning is important. NLTK provides a wide gamut of tools for tokenization, part-of-speech tagging, chunking, named entity recognition and much more.

D. *Scikit-Learn*

A high level machine learning library that provides many tools for all stages of the machine learning process.

E. *Tensorflow*

Tensorflow is an open source machine learning library developed by Google and is widely used in many large projects. It is much lower level compared to SciKit-Learn or Keras, but enables a finer grain of control over the construction of models.
One of the main reasons for choosing it was GPU support which radically sped up training times during experimental work.

F. *Pickle*

Pickle is a serialisation and deserialisation library for python, and is commonly used in machine learning to serialise trained machine learning models. This saves the hassle of having to retrain a system every time and encourages a decoupled python workflow.

IV. Experimental work

A. Voice Interaction with Game Environments

The initial plan for the project was to build a simple game environment consisting of a simple scene of objects in which the player could interact using voice commands.

The sort of interactions in a real game context would depend on the genre, but in the context of an open world game, some of the more basic kinds of interactions would be to instruct a friendly game agent to scavenge for a particular item (x) in an environment, attack or avoid enemy (x), move to a location (x).

I set out to try and create a system that could handle these sorts of interactions, with the goal of discovering the technical challenges associated with integrating NLP systems into games, and also as a means of learning by performing some of the more basic NLP techniques such as tokenization, chunking, named entity recognition, etc…

The core of this system was a simple graphics engine made in C++ which uses OpenGL for rendering and XML for loading the game scene. The system had to be designed to support embedded Python scripting in order to use NLTK which was only available for Python and Java. Boost.Python was used for the integration work between the C++ engine and NLP Python module.

Some time was also spent researching various speech recognition frameworks (CMUSphinx,) and implementing a basic system. To avoid spending more time on development, the framework chosen was Microsoft Speech Api 5.4 (SAPI) as opposed to the Google Speech API. This was because it comes pre-bundled with Windows 10 making it easy to integrate into the project, whereas for Google's, the application would need to support microphone to wav. File storage and client-server communication to send the audio file to be processed by the servers. In C++ this is a lot more work compared to Python or Java because a new protocol called GRPC++ needs to be used. The speech recognition was nowhere near as accurate out of the box as Google's and would struggle in noisy environments, however, it allowed further voice training (which was still tedious) and manual corrections when the system misunderstood what was said.

The python NLP module was solely for the purpose of processing a received sentence from the main app and returning a response objects. The python scripts runs when it receives a sentence and performs tokenization, part-of-speech tagging, chunking and named entity recognition. The key information retrieved was stored in the response object which was passed down as a reference from the app, which could then process it and trigger specific states. By knowing which entity the command is referring to, the characteristics of that entity (green, blue, tall, short) and the particular action to be executed, the mapping of the command to a particular game agent finite state behavior can be accomplished. Currently the system can handle commands such as "Move Bob to England" to "Move short Bob to France" and with very little work could be expanded to variations of these commands.

However, there are many limitations, such as, the commands have to be related to the game world, i.e referring to available agents and supported actions. Also the system isn't good at recognising entity or location names which aren't real which could be resolved by training the speech recognition system and nlp techniques such as named entity recognition, with a corpus of in-game items, names, etc… The finite state system could be overhauled to support goal driven states which would make more complex commands possible such as "Bob go to France while searching for gold" or "Bob go to England while avoiding enemies". Other AI techniques such as Pathfinding could also be incorporated into the states.

This experimental work showed from a game engine integration perspective how difficult it can be to integrate NLP, even a basic system such as this one which is more alike a sentence parsing system and doesn't feature any real learning.

The next sections will show experimental work on current machine learning approaches to a few key topics within NLP, but will not include any kind of integration with a game, the purpose is more to demonstrate the potential of one day seeing all these techniques bundled together.

B. Sentiment Analysis

The first hurdle was to obtain labelled training data that contained 2 classes of text correctly classified as being either positive or negative sentiment. The example dataset chosen to demonstrate sentiment analysis techniques is of IMDB movie reviews, which while different in context to phrases that a player would say in a game, with some modifications such as using a different training corpus for the system, would make it possible to be used in a game.

The dataset comes with a labelled training set containing 3 columns, id, review and sentiment. The review columns contain the review text and the sentiment is the sentiment of that particular class. It can be negative(0) or positive(1).

An unlabeled test set doesn't contains an empty sentiment column and is used solely for predicting and the sentiment labels. Unfortunately, a labelled test set was not found, therefore no precise prediction accuracy metrics can be reproduced. However, some the output dataset is outputted

This experimental work is a single python script called "random_forest_bag_of_words.py" contained in the "sentiment_analysis" folder. The script starts off by reading both training and test sets into memory.

Due to the reviews being retrieved from HTML web pages, HTML markup tags made it into the training and test data, along with other undesirable characters. Beautiful soup was used to remove HTML markup from the original movies reviews. NLTK was used to remove stopwords i.e words that don't carry any significant meaning for sentiment analysis such a "the", "and", etc… Regular expressions were used to exclude all characters but letters from A-Z.

Creating features using bag of words model

The most common and less computationally expensive approach to sentiment analysis is using a bag of words model. The bag of words model is a common technique using in NLP and is when text is represented by a multiset of words disregarding grammar and word order but containing only word frequencies. These frequencies can be used as features for training a machine learning model.
We used the "CountVectorizer" from the feature_extraction module in Scikit-Learn to create the feature vectors where each element represents a word in the text vocabulary and its frequency/number of times occurred.

Training the model

For the sentiment analysis, a more simplistic machine learning model was preferred. A popular and effective model is the random forest classifier. This model consists of many variations of decision tree classifiers that are trained and assessed together and the best performing one is chosen as the main classifier.
Scikit learn was used for providing random forest algorithm in the machine learning module.
To train the model, simply run the "random_forest_bag_of_words.py" script.

Testing the model

Due to not not finding the appropriate labelled test set, obtaining accurate metrics on the prediction accuracy of the model was impossible. Instead, the results were outputted onto a .csv file called "model_test_results.csv" which contains two columns, one for the review text, and the other for the predicted sentiment label.

C. Conversation Agents

This section is solely for the purpose of demonstrating experimental work on conversation bots to show a more hands on approach of the research work undertaken.
The goal for this experimentation was to create a retrieval-based conversation bot that could be trained on a specific dataset containing conversations between real humans, with each question/query containing a specific response.

Data

There were two datasets chosen to demonstrate how the bot's responses would be affected by each dataset. The dataset used was the twitter chat log dataset from [https://github.com/Marsan-Ma/chat_corpus/blob/master/twitter_en.txt.g]. This dataset contains 267,518 short dialogs between humans.

Project Structure

The work is split into a series of python scripts, each with their own individual functionalities. As a simple overview, there is a data pre-processing script for the dataset, a general data retrieval utility script, a Seq2Seq wrapper class for Tensorflow's Seq2Seq core functionality and training-test scripts.

The general data utility script "data_utils" was borrowed from Tensorflow's Seq2Seq English-French machine translation tutorial [8], and provides utility functions for splitting datasets into train, validation and test sets given a ratio, and generating batches of sequences to train the model.
The Seq2Seq wrapper was borrowed from [9], and provides higher level control over Tensorflow's Seq2Seq module. Using wrappers was preferable due to Tensorflow being a lower-level machine learning library, Seq2Seq being an advanced neural network architecture and the project time constraints. This lead to a bigger focus on researching and obtaining preliminary results from small scale experiments and not necessarily on the developing everything from scratch.
The wrapper provides functionality for constructing a tensor flow graph for the Se2Seq neural architecture model by initialising the correct tensors for the 2 RNNs, encoder and decoder.
A training and predicting functions are also provided with extra functionality for pickling/serializing the model progress into the checkpoint files on the disk. This way after a set amount of epochs, the model training progress would be saved, and could be resumed later on. In this particular case, the model can be tested using the pre-trained model without having to retrain it from scratch.

Pre-Processing

Data pre-processing is an important part of any machine learning model, the same applies to this. As stated in the research section, the primary factors for pre-processing for NLP systems is large vocabulary and sequences of different lengths.
To resolve these issues, the "twitter_data_process" script performs a series of pre-processing on the twitter dataset.
Firstly, each line in the dataset is read into an array and filtered to remove characters other than those in the whitelist.

Next, the array of filtered lines is put through further filtering to remove sequences that are above or below a set word limit, in this case sequences have to be between 0-20 words. As the dataset is structured so that there is a question and the following line is the answer to that question, the array of individual lines is split into two individual arrays, one for questions and another for answers.
Tokenization then takes place to convert each sequence into a list of tokens/words which results in two 2D arrays for each question and answer sequence along with their tokens.
To put a cap on the amount of words in our vocabulary, NLTK's "FreqDist" function is used to obtain the frequency of each word. This is then used to pick the first 6000 most common/frequently occurring words. We are left with the vocabulary of available words for training.
As machine learning models deal only with numerical data, a form of mapping had to be established between indexes and words. A dictionary of indexes to words was created, followed by another for words to indexes.

The 2D arrays of each tokenized sequence from the dataset are further processed to ensure that question and answer sequences are stored in arrays of matching sizes. This process is called padding, and for our particular use case it is sufficient as the dataset contains "tweets" which are usually short sentences.
We start the padding process by initialising two 2D arrays with an "i" size of the total number of sequences (267518) and "j" of max number of word indices per sequence (20). The indices start value is 0.
By iterating over the tokenized word sequences and by using the word to index dictionary, we could retrieve the correct index for each word and append it to the correct sequence in the previously initialised index arrays. If the word is not found in the vocabulary, the index for the "UNK" token/word is used which represents an unknown word in the vocabulary.
The result of this are the same 2D arrays, but instead of containing words for each sequence, they contain the indexes which can be fed into the neural network model.

These numpy arrays and dictionaries are then saved to the hard disk, ready to be used during training.

Training

The training of the model is handled by the "train_twitter_bot" script. Firstly, using the "load_data()" function from the "twitter_data_process" script, we load the metadata (dictionaries) and the zero padded word index arrays for the question and answer sequences into variables.
As with any other machine learning model, train, validation and test sets needed to be created by splitting the full dataset. The splits were 70% train and 30% between the validation and test sets.

For initialising the Seq2Seq model we use the wrapper class and pass the length of the sequences (20), the vocabulary size (6000), model's pickle checkpoint path, embedding dimension(1024) and number of layers (3).

After the model has been initialised, training and validation batches of 32 sequences each were yielded from the respective sets to prevent loading all sequences into memory at once.

The model's train function could then be called from the Seq2Seq wrapper class and parameters for the path to the empty checkpoint passed, and the training and validation batches passed one by one into the function for training.
The pre-trained model included in the project took approximately 5 hours to train on a NVIDIA GTX 970.

Testing

The testing procedure was performed by investigating the output responses from the model only, and no formal testing procedure for evaluating the accuracy of NLP systems was carried out.

V. Preliminary Results

Refer to appendix (A) for a snippet of the sentiment analysis results. For full results inspect the output file called "model_test_results.csv" in the sentiment analysis experiments folder.

Refer to appendix (B) for a snippet of the responses generated by the twitter conversation bot. For full results, inspect the output file called "twitter_test_output.txt" in the conversational bots experiment folder.

VI. Future work

There are mandatory project expansions needed to achieve the end goal system. First of all, using a more robust game engine as the core of the project. The Source engine would make a strong potential candidate, as it is open-source, well documented and would meet performance requirements for integrating with computationally expensive systems such as those proposed in the experimental work.

Determining a suitable game context was out of the scope of the mini-project, however, for the final system, it is a key requirement in order to know exactly the goal in which we are training the machine learning systems to accomplish. In the case of conversation agents, the generated responses would need to be related to the game context by influencing the agent's personality. A suitable corpus of training data would need to be created/obtained in order to ensure the context of the game world would be captured by the machine learning models.

A major revamp would have to be made to the initial experimental work on the system for giving voice commands and translating them into particular game agent behaviour. A further machine learning model could be developed to replace the traditional finite state machine and be used to depict the correct behaviour the agent should perform. This would lead to an overall easier system to work with, as the task of mapping text to game actions proved to be tedious.

Time constraints took away the possibility to unify all the experiments. This was a shortcoming of this mini-project. as no game demo was able to be produced. Further work could be done towards creating a singular system.
Thinking in terms of expandability, if the system was completed, the ease of relocating it from one game to another would be a very important factor to consider if such a system were to go commercial.

Finally, due to the vast expanse of topics involved with natural language processing, there was simply no time to do more research to further expand my knowledge in the topics covered and ones that weren't in the scope of the mini-project. Given more time, this would be a priority before resuming work on the project.

VII. Conclusion

The goal for this mini-project was to establish the feasibility of the idea of harnessing natural language processing techniques for enabling natural interactions between players and intelligent game agents. This was important to shed light on the possibility of progressing with this project further.
The research question could have been better defined from the start, as it turned out to be quite vague which lead to a variance in topics covered. The research undertaken in each of the topics was still successful, in that it helped uncover all the tough tasks associated with the end goal.
Initially, an underserved chunk of project time was spent trying to resolve practical problems in relation to the integration of an NLP system into a C++ engine. This experiment could have been largely sped up by choosing a well known open source game engine, nonetheless, while irrelevant to the module, the experience gained from building a hybrid C++-Python system is invaluable for developing systems of similar nature in the future.
The results obtained from the sentiment analysis and conversation agents experiments were positive, and indicated that with further work towards merging each of these techniques into a single system, would make it possible to create a demo game implementing this system.
Overall, the experience of doing this project has been extremely valuable in developing my knowledge of hot research topics such as deep learning and the various state of the art machine learning architectures suited to natural language processing and which are being used in many large commercial applications today.

References

[1] "Analysis of different approaches to Sentence-Level Sentiment Classification,"

[2] The Ubuntu Dialogue Corpus: A Large Dataset for Research in Unstructured Multi-Turn Dialogue Systems

[3] **Building End-To-End Dialogue Systems Using Generative Hierarchical Neural Network Models**

[4] A Persona-Based Neural Conversation Model,

[5] *RECURRENT NEURAL NETWORKS TUTORIAL, PART 1 – INTRODUCTION TO RNNS*

[6] Learning Phrase Representations using RNN Encoder–Decoder for StatisticalMachineTranslation\

[7] Deep Learning for Chatbots : Part 1

[8] https://www.tensorflow.org/tutorials/seq2seq

[9] http://suriyadeepan.github.io/2016-12-31-practical-seq2seq/

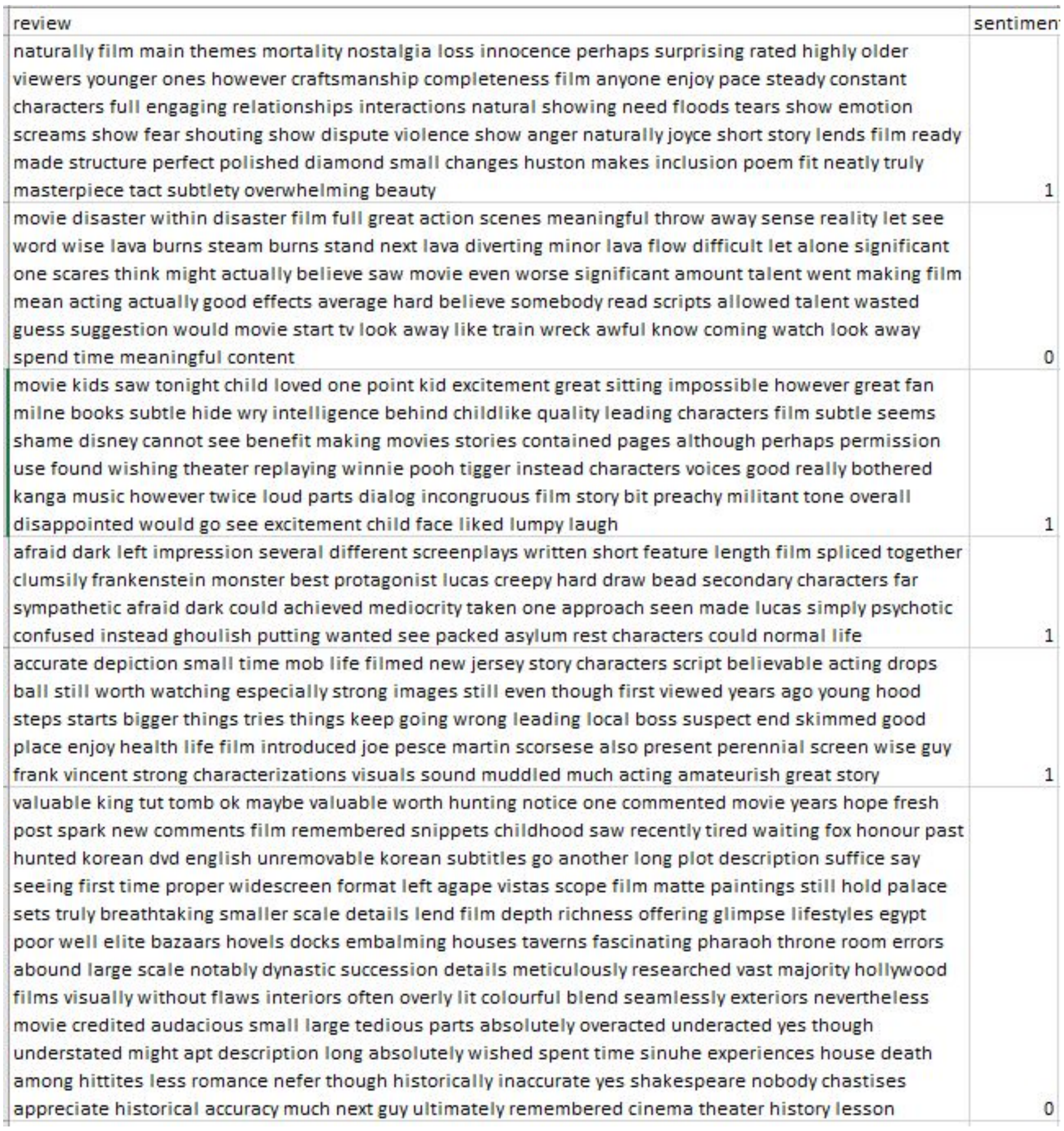
[10] Sequence to Sequence Learning with Neural Networks

[11] http://www.wildml.com/2016/04/deep-learning-for-chatbots-part-1-introduction/

[12] http://www.wildml.com/2016/07/deep-learning-for-chatbots-2-retrieval-based-model-tensorflow/

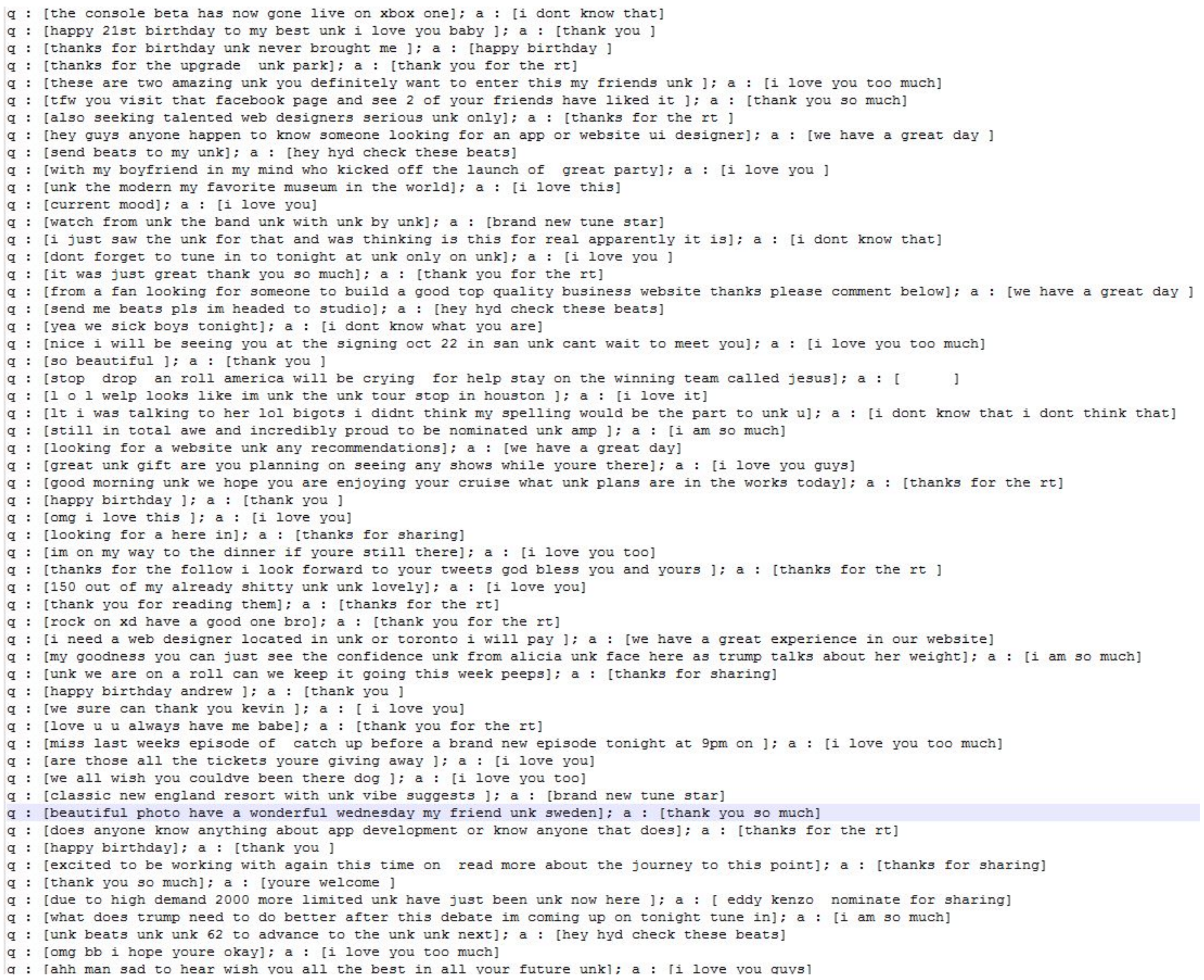
Appendices

Appendix (A) - Sentiment Analysis Results Output Snippet



review	sentimen
naturally film main themes mortality nostalgia loss innocence perhaps surprising rated highly older viewers younger ones however craftsmanship completeness film anyone enjoy pace steady constant characters full engaging relationships interactions natural showing need floods tears show emotion screams show fear shouting show dispute violence show anger naturally joyce short story lends film ready made structure perfect polished diamond small changes huston makes inclusion poem fit neatly truly masterpiece tact subtlety overwhelming beauty	1
movie disaster within disaster film full great action scenes meaningful throw away sense reality let see word wise lava burns steam burns stand next lava diverting minor lava flow difficult let alone significant one scares think might actually believe saw movie even worse significant amount talent went making film mean acting actually good effects average hard believe somebody read scripts allowed talent wasted guess suggestion would movie start tv look away like train wreck awful know coming watch look away spend time meaningful content	0
movie kids saw tonight child loved one point kid excitement great sitting impossible however great fan milne books subtle hide wry intelligence behind childlike quality leading characters film subtle seems shame disney cannot see benefit making movies stories contained pages although perhaps permission use found wishing theater replaying winnie pooh tigger instead characters voices good really bothered kanga music however twice loud parts dialog incongruous film story bit preachy militant tone overall disappointed would go see excitement child face liked lumpy laugh	1
afraid dark left impression several different screenplays written short feature length film spliced together clumsily frankenstein monster best protagonist lucas creepy hard draw bead secondary characters far sympathetic afraid dark could achieved mediocrity taken one approach seen made lucas simply psychotic confused instead ghoulish putting wanted see packed asylum rest characters could normal life	1
accurate depiction small time mob life filmed new jersey story characters script believable acting drops ball still worth watching especially strong images still even though first viewed years ago young hood steps starts bigger things tries things keep going wrong leading local boss suspect end skimmed good place enjoy health life film introduced joe pesce martin scorsese also present perennial screen wise guy frank vincent strong characterizations visuals sound muddled much acting amateurish great story	1
valuable king tut tomb ok maybe valuable worth hunting notice one commented movie years hope fresh post spark new comments film remembered snippets childhood saw recently tired waiting fox honour past hunted korean dvd english unremovable korean subtitles go another long plot description suffice say seeing first time proper widescreen format left agape vistas scope film matte paintings still hold palace sets truly breathtaking smaller scale details lend film depth richness offering glimpse lifestyles egypt poor well elite bazaars hovels docks embalming houses taverns fascinating pharaoh throne room errors abound large scale notably dynastic succession details meticulously researched vast majority hollywood films visually without flaws interiors often overly lit colourful blend seamlessly exteriors nevertheless movie credited audacious small large tedious parts absolutely overacted underacted yes though understated might apt description long absolutely wished spent time sinuhe experiences house death among hittites less romance nefer though historically inaccurate yes shakespeare nobody chastises appreciate historical accuracy much next guy ultimately remembered cinema theater history lesson	0

Appendix (B) - Twitter bot conversation output snippet


```
q : [the console beta has now gone live on xbox one]; a : [i dont know that]
q : [happy 21st birthday to my best unk i love you baby ]; a : [thank you ]
q : [thanks for birthday unk never brought me ]; a : [happy birthday ]
q : [thanks for the upgrade  unk park]; a : [thank you for the rt]
q : [these are two amazing unk you definitely want to enter this my friends unk ]; a : [i love you too much]
q : [tfw you visit that facebook page and see 2 of your friends have liked it ]; a : [thank you so much]
q : [also seeking talented web designers serious unk only]; a : [thanks for the rt ]
q : [hey guys anyone happen to know someone looking for an app or website ui designer]; a : [we have a great day ]
q : [send beats to my unk]; a : [hey hyd check these beats]
q : [with my boyfriend in my mind who kicked off the launch of  great party]; a : [i love you ]
q : [unk the modern my favorite museum in the world]; a : [i love this]
q : [current mood]; a : [i love you]
q : [watch from unk the band unk with unk by unk]; a : [brand new tune star]
q : [i just saw the unk for that and was thinking is this for real apparently it is]; a : [i dont know that]
q : [dont forget to tune in to tonight at unk only on unk]; a : [i love you ]
q : [it was just great thank you so much]; a : [thank you for the rt]
q : [from a fan looking for someone to build a good top quality business website thanks please comment below]; a : [we have a great day ]
q : [send me beats pls im headed to studio]; a : [hey hyd check these beats]
q : [yea we sick boys tonight]; a : [i dont know what you are]
q : [nice i will be seeing you at the signing oct 22 in san unk cant wait to meet you]; a : [i love you too much]
q : [so beautiful ]; a : [thank you ]
q : [stop  drop  an roll america will be crying  for help stay on the winning team called jesus]; a : [      ]
q : [l o l welp looks like im unk the unk tour stop in houston ]; a : [i love it]
q : [lt i was talking to her lol bigots i didnt think my spelling would be the part to unk u]; a : [i dont know that i dont think that]
q : [still in total awe and incredibly proud to be nominated unk amp ]; a : [i am so much]
q : [looking for a website unk any recommendations]; a : [we have a great day]
q : [great unk gift are you planning on seeing any shows while youre there]; a : [i love you guys]
q : [good morning unk we hope you are enjoying your cruise what unk plans are in the works today]; a : [thanks for the rt]
q : [happy birthday ]; a : [thank you ]
q : [omg i love this ]; a : [i love you]
q : [looking for a here in]; a : [thanks for sharing]
q : [im on my way to the dinner if youre still there]; a : [i love you too]
q : [thanks for the follow i look forward to your tweets god bless you and yours ]; a : [thanks for the rt ]
q : [150 out of my already shitty unk unk lovely]; a : [i love you]
q : [thank you for reading them]; a : [thanks for the rt]
q : [rock on xd have a good one bro]; a : [thank you for the rt]
q : [i need a web designer located in unk or toronto i will pay ]; a : [we have a great experience in our website]
q : [my goodness you can just see the confidence unk from alicia unk face here as trump talks about her weight]; a : [i am so much]
q : [unk we are on a roll can we keep it going this week peeps]; a : [thanks for sharing]
q : [happy birthday andrew ]; a : [thank you ]
q : [we sure can thank you kevin ]; a : [ i love you]
q : [love u u always have me babe]; a : [thank you for the rt]
q : [miss last weeks episode of  catch up before a brand new episode tonight at 9pm on ]; a : [i love you too much]
q : [are those all the tickets youre giving away ]; a : [i love you]
q : [we all wish you couldve been there dog ]; a : [i love you too]
q : [classic new england resort with unk vibe suggests ]; a : [brand new tune star]
q : [beautiful photo have a wonderful wednesday my friend unk sweden]; a : [thank you so much]
q : [does anyone know anything about app development or know anyone that does]; a : [thanks for the rt]
q : [happy birthday]; a : [thank you ]
q : [excited to be working with again this time on  read more about the journey to this point]; a : [thanks for sharing]
q : [thank you so much]; a : [youre welcome ]
q : [due to high demand 2000 more limited unk have just been unk now here ]; a : [ eddy kenzo  nominate for sharing]
q : [what does trump need to do better after this debate im coming up on tonight tune in]; a : [i am so much]
q : [unk beats unk unk 62 to advance to the unk unk next]; a : [hey hyd check these beats]
q : [omg bb i hope youre okay]; a : [i love you too much]
q : [ahh man sad to hear wish you all the best in all your future unk]; a : [i love you guys]
```