MapChat: Technical Paper

CSCI 599: Applied Machine Learning for Games

Professor Mike Zyda

Team Members

Bailin Chen (bailinch@usc.edu)
Supreeth Kabbin (skabbin@usc.edu)
Peiyuan Liu (peiyuanl@usc.edu)
Sriya Mahankali (smahanka@usc.edu)
Spencer Ortega (sportega@usc.edu)

Motivation and Background

MapChat is a quest-based role playing game (RPG) that revolves around practicing english through "chats" about everyday topics. The target demographic for this game will be any international student or tourist looking to assess their english conversational skills for any scenario that may arise. The main purpose of this game is to create a fun and interactive environment to develop your english in an environment that resembles the real-world, but for the scope of our project we focused more on how we could use machine learning to engage and assess MapChat users throughout their journey.

There is a marked difference between the english you would encounter in an academic setting as opposed to your everyday life. As an english practicing game, MapChat would like to tap into the language learning experience a person receives from candid conversations, but without the discouraging obstacles that come with the real-world. We will mainly accomplish these feats by using artificial intelligence to facilitate the conversations, creating a safe and non-judgemental interaction, while pace and difficulty of the game is totally dependent upon the user's expertise.

Prior Research

Prior research for MapChat included common ways machine learning is used in Game development and how applicable these methodologies would be towards the goal. Reinforcement learning and Artificial Intelligence bots, while commonly used in Game development, would not be suitable for this project. In order to achieve the goal of making more realistic interaction to gauge the users English speaking abilities and improve them, the team looked into Google’s

Language Proficiency Detection in Social Applications [1] and Knowledge Level Assessment in e-Learning Systems Using Machine Learning and User Activity Analysis [2].

This research led to Natural Language Processing particularly in the subdomains of Automated Speech Recognition, Speech to Text Conversion, Natural Language Understanding and Interactive Chatbots. Existing work in this domain existed as callable interfaces from Google Cloud Recognition, IBM Watson, DialogFlow, Kaldi and Natural Language Toolkit.

Final Product

Overview

In order to simulate a real-world environment, the player will control a character navigating through a map which contains various "chat spots". These locations in the game represent places that people go to on a daily basis, such as: coffee shops, grocery stores, and gyms. The player can go to any of these spots and have a conversation with some non-player character (NPC), where the topic of the conversation is influenced by the location. At each level, the player will be given a list of words and a round score of 0. To beat the level the player must use every word in the list at least once throughout their conversations and must have accumulated a high enough score to advance. The words in the list will become more sophisticated and score requirements will increase as the player advances through the levels, making the levels inherently more difficult. Points are given for every reply that is on topic and grammatically correct, regardless if a word from the list is present. All points will be directly weighted by how similar the context of the user's reply is to the current dialog. Appropriate feedback will be given to the player in the case of any irrelevant input or grammatical mistakes for all replies.

Unity

We decided to use Unity as the game engine for MapChat mainly due to it having a relatively easy learning curve, especially since none of our team members have any significant game development experience, but also because of the numerous free Unity Assets and Software Development Kits (SDK) that are available. Both of these aspects of Unity significantly helped us in the long run, especially how seamless it was to import assets and SDKs into our project.

The map from our 2D version was mainly created by using Internal Unity tools and 3rd party Assets. The background was all created brick by brick using Unity's Tilemap grid component, while the other UI components, such as characters, chat boxes, and buildings were all sourced from Unity Playground [3]. For our 3D version of the map we wanted to focus more on simulating reality and providing the user with a real-world experience. Our solution to accomplishing this was to use some Unity SDK that would allow us to generate a 3D map of the real-world, where the user's character can interact with objects inside the map. We initially were going to use the Google maps SDK [4] but it can only be accessed by commercial use. We ended up using an open source

alternative, MapBox [5], which was capable of doing all tasks that were required. We no longer needed to manually create a background because Mapbox was able to generate the streets, buildings, etc as game objects. To make it even more realistic and personal for our fellow USC students, we set up MapBox to only focus on the USC village area.

Machine Learning

IBM Watson

The machine learning technology in our project is primarily implemented via IBM Watson AI, which is a cloud native solution that uses deep learning AI algorithms. We make use of a couple of features offered by Watson, namely Speech to Text Recognition [6] and Natural Language Understanding [7]. The presence of a Watson Unity SDK makes it ideal for our application.

Speech to Text:

Watson Speech to Text is a Deep learning neural technology which can be used to enable voice conversation, transcription and analytics in an application. In order to give players the option to provide their vocal input, speech recognition is employed to record this data. This gives the player the ease of replying with speech instead of typing, while at the same time practicing their oral articulation skills. The pre-trained model provided by Watson is not accurate and is challenged by environmental factors such as noise and accent of a user. To tackle this, we have trained the model with datasets of voice inputs having a noisy background and different accents.

Natural Language Understanding:

Watson Natural Language Understanding is a deep learning tool that analyzes text and extracts metadata such as keywords, categories, entities, sentiment, emotions and syntax. It enables us to break down a sentence, identify the context, and categorize the various entities present in it. By comparing the categorical breakdown of the NPC's dialog and the user's reply statement, we can verify whether the user is speaking in context of the current "chat spot" or if they are off topic. We make use of the category labels, along with their confidence scores, to assess how similar two statements are. A weighted average of the tags is used with the intersection over union measure to calculate context similarity.

Content Generation

A very important aspect of our game is how we engage our users with dialog. We initially had intention to train some 3rd party AI chatbot, such as IBM Watson Assistant [8], to learn how to hold a conversation about a specific topic. After further investigation of the capabilities of these chatbots it seemed like these services were more geared towards a personal assistant, rather than a bot that tries to conversate. We came to the conclusion that training a chatbot to have a continuous conversation was a much more lofty goal than we anticipated. Our next best option was to try and generate artificial text resembling a human via some machine learning model. We looked into multiple different open-source text generators, ultimately deciding to use the "textgenrnn" [9] python package.

textgenrnn

The textgenrnn package allows you to easily train your own text-generating Recurrent Neural Network (RNN), with a Tensorflow backend, on any generic input text file. Our idea was to train a single RNN for every chat spot topic so that we can generate text and use it as dialog to engage the user. Before we could even begin testing textgenrnn we had to decide on an appropriate natural language processing (NLP) data set to parse and train it on. We ended up choosing Yelp's NLP dataset [10] because its context aligned exactly with the type of dialog we wanted: real people talking about real businesses.

Yelp's NLP Dataset

Yelp's dataset consists of over 5GB of real user reviews, all provided complimentary of Yelp for the purpose of advancing NLP research. With there being so many reviews to choose from, we had to figure out an intelligent way to parse training data for each desired topic. Our solution to this dilemma was two python scripts that parsed and formatted Yelp data accordingly. For the parsing script the user would first have to provide a list of business category tags, which is then used to filter the raw reviews. After the business category reviews are parsed, we can then pass it through our format script, which takes in a list of keywords that it will filter the reviews a second time with. This methodology allowed us to quickly test different training sets by simply changing some input parameters. Once we created a training set for all of our models we were now ready to move our data to the cloud to start training our RNNs on GPUs to accelerate the process.

Training Resources

For our first trial of training we followed textgenrnn's suggestion to train models on Google Colab [11], which is Google's free jupyter notebook service that has access to GPU resources. However, when trying to use this free service we would always abruptly time-out of the system due to inactivity on the browser, which made training multiple models unfeasible. Our next best option was to use a paid version of Colab or some other cloud service. Coincedently one of our team members had access to USC's Center for High Performance Computing (HPC) [12], allowing us to do all of our training in house at USC. The installation and training of textgenrnn was seamless with USC HPC having pre-installed CUDA libraries and tons of GPUs available to use. We were able to complete all of our training within a few days, but we knew immediately when we saw the results that it would not be suitable enough for our requirements. With the deadline approaching, we decided to just use the actual yelp reviews as our dialog content, as opposed to further tweaking the training data and model parameters to get better results.

Grammar Bot

In addition to the speech to text tool, we use a tool (GrammarBot) to check for grammatical errors made by the user and explicitly (displayed to the user) correct them on input. GrammarBot [13] is an API that checks the grammar mistakes of English sentences, which returns a JSON file with the information such as error location and correction. The API uses language models, sequence labeling and natural language parsing to perform its task. We call the API by HTTP requests as

GrammarBot can check the English sentences that users speak during the game, so that users can realize the mistakes that they made.

Results & Analysis

Overall we were able to make significant strides with our progress developing MapChat.
The game is clearly not ready to be released to the public, but it does serve as a proof of concept of what we first envisioned. We were able to successfully create this quest-based english practicing game by connecting multiple crucial components together.

The rule set of mapchat is the foundation for the english-practicing our users get from playing. By increasing the difficulty of the words in the list as the user advances through the levels, we are able to use it as an adjustable parameter for all levels of expertise. In order to change the difficulty of the game according to the user's conversational skills and encourage healthy competition, we developed a scoring system that intelligently evaluates the user's performance with the help of NLU.

We were able to implement all these rules using multiple different Machine learning services. Using NLU and GrammarBot, we score the user's input by checking its grammar correctness and its relevance to the NPC's dialog. We were able to utilize Yelp's NLP dataset as dialog to further facilitate a more realistic experience by providing content that was created by real people. Although we didn't get to use our training results, we did accomplish a reusable workflow to set up our RNN training environment on USC HPC and other cloud services alike. Furthermore, voice input is chosen as the main input method of the game to give the user a more realistic 'chat' experience. This is achieved by incorporating Watson Speech-To-Text service.

Other than the 2D Unity Playground virtual map that we created for the user to embark on their adventure, we implemented MapBox to provide a 3D map that is from the real-world map allowing the user to have a realistic practice experience.

Limitations, Conclusions & Future Work

As mentioned earlier in this document, the project is not ready for public as there are limitations and challenges that need to be addressed. They are categorized into topics and are listed below:

Speech to text

The free version of Watson Speech-to-Text service is highly inaccurate when the user has an accent. The accuracy did improve after the model was trained by us with some accented voice datasets. However, the size of data is limited by the free service. One simple way to address this is to use the paid service.

Natural Language Understanding

Since the free version of the Watson NLU service is used in this project, some of the keywords are not picked up by the NLU service. This greatly affects the accuracy of the relevance calculating mechanism of the game, which sometimes gives low similarity value even if the sentences are relevant. Although it could be improved by using the paid service, the nature of the existing NLU services (finding keywords and returning labels of them) are not intelligent enough for real-world implementation.

Content Generation

We had initial intentions of using the generated text from our trained models, but after further investigation we found a good majority of it was gibberish. It was probably best that we committed to using the Yelp data as our dialog, as opposed to gambling with training and generating more text, especially since it is such a crucial component to our english practice. At the end of the day your trained model is only as good as the training data you feed it. We would need to make some drastic improvements in the way we intelligently filter out reviews given a topic to implement this initial method. One possible idea is to run our NLU service on all of the reviews so we could have a better understanding of its context before filtering.

Setting up our software and computing resources was fairly a simple process on USC HPC. The only issue we encountered was attempting to connect GPUs from distributed nodes together, which in the end limited us to only being able to use one node with two GPUs for each model we needed to train. If we could get a model to train on four GPUs instead of two, there should be noticeable speed up in the training, giving us more time to train more models.

The last improvement that would really solidify our content generation solution would be to generate dialog in game. After we finish training our models using accelerated computing resources, we could use the newly created weight file to infer/generate text using the resources of a laptop. So after we successfully train our models to our liking and standard, we could then import them into our Unity project. While the game is running, we can call the textgenrnn python package to infer text on the fly. This would tie the conversation aspect of our game together by dynamically creating new dialog instead of using a static list, giving our users a fresh experience every time.

Unity

Due to the time limit and the fact that none of the team members has experience in game graphic design, the game utilizes the most basic graphical implementation of Unity and MapBox. This also limits us to achieve one of our ambitious goals: dynamically assigning ‘chat spots’ on the 3D map by reading the labels of the business on the map. Reading the labels itself is not a challenge since all the data is publicly accessible via MapBox or Google Map. But time is needed to seek a solution to layout the collider GameObjects dynamically in Unity.

Gameplay Mechanism

If the game is to be developed into a public-released game, some of the gameplay mechanisms need to be refined. For example, the scoring system should also record the user's performance for different topics separately, so that the user can be given more words related to the topics that they are weak at. Furthermore, a new words list creating system needs to be developed in order to create the words list dynamically, with the words that are related to the topics as well as the difficulty matches the level.

References

[1] Google: Language Proficiency Detection in Social Applications. Retrieved from https://patentimages.storage.googleapis.com/1c/82/c8/355135e6b8324e/US20140335483A1.pdf

[2] Knowledge Level Assessment in e-Learning Systems Using Machine Learning and User Activity Analysis. Retrieved from https://pdfs.semanticscholar.org/fb8f/2646a05ac4066b40444ab481a04157301a89.pdf

[3] Unity Assets Store. Retrieved from https://assetstore.unity.com/

[4] Google Maps SDK Overview. Retrieved March 4, 2020 from https://developers.google.com/maps/documentation/gaming/overview_musk

[5] MapBox. Retrieved from https://www.mapbox.com/unity/

[6] IBM Watson Speech to Text. Retrieved from https://www.ibm.com/cloud/watson-speech-to-text

[7] IBM Watson Natural Language Understanding. Retrieved from https://www.ibm.com/cloud/watson-natural-language-understanding

[8] IBM Watson Assistant. Retrieved from https://www.ibm.com/cloud/watson-assistant/

[9] textgenrnn
https://github.com/minimaxir/textgenrnn
[10] Yelp's NLP dataset
https://www.yelp.com/dataset
[11] Google Colab. Retrieved from
https://colab.research.google.com/
[12] Center of High-Performance Computing, University of Southern California. Retrieved from
https://hpcc.usc.edu/
[13] GrammarBot. Retrieved from
https://www.grammarbot.io/about