Immersive Gameplay via Improved Natural Language Understanding

Berkeley Andrus
Brigham Young University
Provo, Utah
bandrus5@byu.edu

Nancy Fulda
Brigham Young University
Provo, Utah
nfulda@cs.byu.edu

ABSTRACT

Many first-person shooters feature non-player characters (NPCs) that work alongside the player. Interfacing with these NPCs can add unnecessary complication to a game and steepen the learning curve for new players. Recent improvements in automated voice recognition and language representation have set the stage for more immersive methods of interfacing with NPCs through player speech. In this paper, we present several promising methods of classifying user utterances to extract predefined commands from unstructured speech. This framework facilitates a more flexible interface than has been used in past speech-controlled games. We also show how our methods effectively leverage small sets of example data to outperform existing industrial utterance classification systems.

CCS CONCEPTS

• **Human-centered computing → Natural language interfaces**.

KEYWORDS

Speech Interfaces, Natural Language Interfaces, Intent Mapping, Command Extraction, Sentence Embeddings

ACM Reference Format:
Berkeley Andrus and Nancy Fulda. 2020. Immersive Gameplay via Improved Natural Language Understanding. In *International Conference on the Foundations of Digital Games (FDG '20), September 15–18, 2020, Bugibba, Malta.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3402942.3403024

1 INTRODUCTION

Many popular military-style video games feature a squad of AI-controlled soldiers who fight alongside the main character. These characters often have effective low-level behaviors but are unable to consistently align their low-level behaviors with the situational goals of the player. Even the best automated teammates can occasionally be found blocking the player's intended path, firing at unimportant targets, failing to enter or exit vehicles at appropriate times, or generally acting contrary to the player's motives. This is unsurprising, as there is no way for the game to know what the player is thinking without some sort of player intervention, such as interfacing through command and communication menus. If these communication interfaces have even a small amount of complexity, they usually require an interruption to core gameplay that is at best time-consuming and at worst a frustrating pain point.

A speech interface for controlling automated teammates seems like an ideal way to give players control over other characters without requiring them to pause gameplay and navigate through menus. But how do we interpret player speech accurately enough for a speech interface to be effective? And how do we do so without requiring the player to memorize a list of valid commands?

Word and sentence embeddings generated by neural embedding models introduce a promising tool well suited for this task. These embeddings are high-dimensional vectors that allow us to perform mathematical operations on unstructured language. For example, we can compare the embeddings of two sentences to find a real-value distance between them. More ambitiously, we can imagine that there is an appropriate mapping from the unbounded range of possible human inputs to the finite, predefined range of commands that the automated agent knows how to interpret. We could use this mapping to extract commands from player speech.

In this paper, we explore how we can leverage existing sentence embedding tools to extract meaningful commands from unstructured player speech. Section 2 discusses the prior research that makes our contributions possible. Sections 3 and 4 detail our methods and experimental results. Section 5 explains the implications of what we observed and next steps in our research.

2 RELATED WORK

Most researchers who work with language have seen the power of neural language models. These models include word2vec [15] and FastText [2], which are trained by having a neural network predict the contexts of words or sub-words in text, and GLoVE [17], which is trained on word-word co-occurrence statistics. The resulting word embeddings facilitate commonsense reasoning via mathematical operations. For example, it is possible to identify capital cities by adding a constant vector to the embedded representations of corresponding countries [16] or to determine algorithmically which behaviors are afforded by a specific object [7].

On the multi-word level, deep neural network structures such as LSTMs, convolutions, and attention mechanisms can process strings of words and produce a distributed representation of whole sentences. In this paper, we use Facebook's InferSent sentence encoder [3] as the basis for our linguistic comparisons.

Even before these more recent language modelling algorithms, researchers have found other ways to represent words as vectors and use those representations to detect semantic similarity [6, 8]. This idea has been applied to compare arbitrary words and groups of words [8] or to classify documents and utterances based on a

FDG '20, September 15–18, 2020, Bugibba, Malta

ACM ISBN 978-1-4503-8807-8/20/09…$15.00
https://doi.org/10.1145/3402942.3403024

fixed set of output classes [10–12, 18]. These latter classification problems are similar to the problem we address here, except that we make classifications based on only a handful (1-3) of example utterances per class as opposed to the hundreds or thousands of examples needed for even simple machine learning algorithms to work well. In that sense our work is similar to the intent-routing functionality of the Google Voice Assistant [9] and Amazon Alexa [1] platforms, whose implementation details are proprietary.

The idea of using human voices to control video games is not new. In 2011 Harada et al. presented the vocal joystick that used vowel intonations to interpret user commands [13]. Video games like *Mass Effect 3* and gaming systems like *Oculus Rift* use speech recognition to navigate menus or select predefined player actions, and exclusively voice-controlled games like *Mayday! Deep Space!* allow players to deliver a set of predefined commands via voice [22]. Our work takes these innovations a step farther by replacing concrete commands with unstructured, free-form player speech.

In that context, our work is perhaps most similar to the AIIDE Playable Experience *Traveler*, which uses free-form utterances to guide an interactive narrative [19]. However, *Traveler* uses only word-level representations, whereas our work leverages the representational power of pre-trained sentence embeddings. Another similar project is Upside Lab's StarCraft II voice interface [5], which uses the Amazon Alexa platform to allow players to execute common game commands via verbal instructions rather than a keyboard and mouse. Our goal is to create a model that can fill Amazon Alexa's role in similar voice interfaces and provide more flexibility for the player than is currently possible.

The task of classifying utterances according to predefined classes is related to but distinct from the concepts of query understanding, which uses a more open domain, and named-entity recognition, which assumes that relevant entities are referenced explicitly.

3 METHODOLOGY

Our general strategy for interpreting user commands is to compare each user utterance to a small domain of defined actions and determine which action was intended by the utterance. In order to make these comparisons, we use the InferSent sentence embedding tool [3] to represent all utterances as high-dimensional vectors.

We start by giving our model a list of possible actions that an automated agent can take. Each action includes a verb phrase (such as *Follow*, *Attack*, *Protect*, or *Go to*) and optionally an object or target (such as *vehicle*, *building*, *enemies*, or *allies*). Targets are represented by NULL in cases where an action needs no specific target, such as when the verb is *Flee*. This structure of organizing commands into verbs and objects does not cover every conceivable use case, but it has a high amount of flexibility while retaining simplicity.

In addition to the list of possible actions, we give the model a small list of example user utterances with labels of the verbs and objects each utterance should map to. We refer to these example cases as the model's *guidance data*. This guidance data is similar to the training data used in machine-learning models, except the set is too small to facilitate any actual 'training'.

Notably, we do not give our model any prior knowledge about the likelihood of specific actions. In a production environment a game engine would help our model make better decisions based on game context. Those decisions would need to be made explicitly on a game-by-game basis, which is why they are not included in our experiments.

3.1 Experiment Setup

We use four evaluative test environments to compare strategies for command extraction, each with its own domain of possible actions and its own test and guidance data. The Warcraft test environment (WC) was created from samples of NPC dialogue from the game Warcraft III, as found on *wowwiki.fandom.com* [4]. The Handwritten 1 and Handwritten 2 test environments (H1, H2) were written in our lab to represent common phrases that might be used while playing a sci-fi combat game like Halo. The Call of Duty test environment (CoD) was collected from human volunteers who were shown screenshots from the game Call of Duty and asked what they would say to get a teammate to accomplish a specified objective. The first three test sets were primarily used to tune hyper-parameters and develop methodologies while the fourth was primarily used for method validation. More detailed descriptions of the test environments as well as all test and guidance data can be found at https://github.com/bandrus5/immersive-gameplay/.

We evaluate our model by having it predict the verb phrase and object intended by novel utterances. We score its accuracy on verbs and objects separately, such that it earns 1 point by predicting either the verb or object correctly or 2 points for predicting both correctly. We calculate the accuracy score for each set of experiments as the percentage of possible points earned by the model.

We run each set of experiments twice, using two pretrained versions of InferSent [3] to generate embedded representations of sentences. Version 1 was trained using the GloVe word embedding space [17] and version 2 was trained using the FastText word embedding space [2]. We used both versions not to see which one was more effective for our task, but to help generalize our results and find patterns that will hold true for any embedding tools used.

4 EXPERIMENTS

Our primary experiment is to compare the accuracy achieved by four command extraction methods. Each method compares embedded representations of sentences to predict which predefined action is intended by a user's utterance. We use the following extraction methods, which are listed in increasing order of how directly they rely on the test environment's guidance data:

- **Simple Distance:** In this method, the embedded representation of the user's utterance is compared to the embedded representations of each possible action in the test environment's domain. The action whose representation is closest to the utterance's representation determines both the verb and the object predicted by our model.
- **Translation:** The *Translation* method selects verbs and objects independently by comparing the embedded representations of each utterance to the embedded representations of each verb and then to the representations of each object. It adds a guidance vector g_v to each utterance before comparing it to the set of possible verbs. g_v represents the distance in the embedding space between an arbitrary unstructured utterance and the domain of verbs and is defined by the

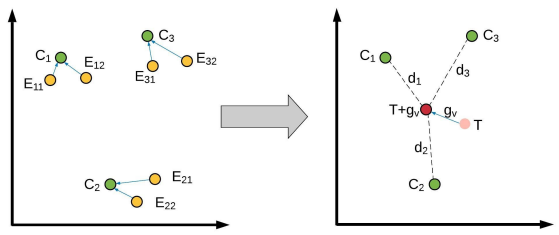

Figure 1: Translation method. On the left we calculate a guidance vector g_v as the average distance between each example utterance E_{ij} and its labelled verb C_i. On the right we add g_v to the embedded representation of the user's utterance (T) before comparing the utterance to each verb C_i. This process is repeated for objects.

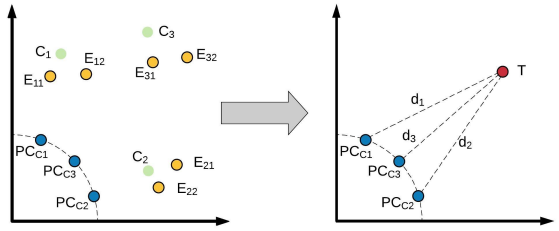

Figure 2: PC Distance method. On the left, unit length eigenvectors for the approximated first principle components (PC_{Ci}) are calculated based on example utterances (E_{ij}) corresponding to each verb (C_i). On the right, we compare the embedded representation of the user's utterance (T) to each eigenvector to determine the verb intended by the utterance. This process is repeated for objects.

equation $g_v = \frac{1}{n}\sum_0^n(v_i - u_i)$, where u_i comes from the set of representations of the labeled example utterances and v_i comes from the set of representations of the associated verbs. A similar guidance vector g_o is calculated and applied for objects. This process is visualized in Figure 1.

- **PC Distance:** This method starts with the embedded representations of the set of example utterances corresponding to each verb and each object. It approximates the first principle component of each set of representations and calculates a unit length eigenvector for that principle component. The eigenvector is used as a summary of the example cases corresponding to each verb and object. The embedded representation of the user's utterance is compared to each of the calculated eigenvectors and the shortest distances are used to select the intended verb and object. This process is visualized in Figure 2 and is related to a strategy first introduced for facial recognition in [20].
- **Nearest Neighbor:** In this method, the embedded representation of the user's utterance is compared to the representations of each hand-coded example utterance rather than against the representations of the predefined actions directly. The labelled verb and object of the closest example utterance become the model's predictions.

Table 1: The percentage of verbs and objects that were correctly identified using each command extraction method in each test environment. Accuracy scores are averaged over all distance metrics and both versions of InferSent.

	WC	H1	H2	CoD	Average
Random (Baseline)	20.8	12.0	14.4	16.7	16.0
Simple Distance	28.0	40.2	**46.0**	42.6	39.2
Translation	36.3	42.7	45.1	40.7	41.2
PC Distance	**58.8**	43.6	32.0	**53.5**	**46.9**
Nearest Neighbor	56.6	**47.6**	33.2	48.5	46.4

Table 2: The percentage of verbs and objects that were correctly identified using each distance metric in each test environment. Accuracy scores are averaged over all command extraction methods and both versions of InferSent.

	WC	H1	H2	CoD	Average
Cosine	47.6	50.8	42.1	49.5	47.5
Bray-Curtis	47.3	47.5	39.8	47.5	45.5
Canberra	40.3	35.5	36.6	43.7	39
Chebyshev	36.5	26.5	31.3	33.9	32.0
City Block	47.3	42.8	39.4	46.2	43.9
Correlation	47.6	**55.0**	**43.3**	**51.3**	**49.3**
Euclidean	**48.0**	46.5	40.9	46.3	45.4

The performance of each command extraction method on each test environment can be found in Table 1. Interestingly, there was little variation in relative performance of extraction methods between InferSent versions. Conversely, there was a high amount of variation in the performance of extraction methods between test environments. One possible reason for this could be the varying ratios between number of guidance data and number of possible actions in each test environment, as some methods rely more on the guidance data than others. *Nearest Neighbor*, *Translation*, and *PC Distance* (which used guidance data) all performed better overall than *Simple Distance* (which did not use guidance data). This suggests that even a minimal number of labeled examples can be leveraged to boost accuracy for speech classification tasks.

Each command extraction method involves finding the distance between two vector representations of text. Many distance metrics exist for comparing vectors [14], so as a secondary experiment we combine each extraction method with each of the following distance metrics as defined in SciPy's scipy.spatial.distance library [21] and compare their accuracy scores: [Cosine, Bray-Curtis, Canberra, Chebyshev, City Block (Manhattan), Correlation, Euclidean].

Cosine distance has been the commonly used distance metric for comparisons between representations of sentences [8, 12]. However, we found *Correlation* as a distance metric to yield more accurate results, as shown in Table 2. This was true for InferSent versions 1 and 2 and for three out of the four test environments.

The Google Voice Assistant [9] and Amazon Alexa [1] platforms both provide a service called 'intent routing', which classifies user

Table 3: Our model's top accuracy on each test environment compared to the accuracy of the Google Voice Assistant and Amazon Alexa platforms. In this experiment our model used the most accurate extraction method and distance metric for each test environment rather than averaging over many options as in Tables 1 and 2.

	WC	H1	H2	CoD
Google Voice Assistant	15.5	50.0	22.5	47.9
Amazon Alexa	36.7	28.0	28.4	42.8
Our model	**66.8**	**62.0**	**52.6**	**59.9**

commands by their intent and routes requests to applicable server endpoints. This intent routing task is analogous to our command extraction task, making their platforms a natural baseline for comparison. Their specific methodology is proprietary, which precludes any direct comparisons to the strategies they use. Instead, we performed an end-to-end test to compare classification accuracy. We reconstructed our four test environments as individual Google Actions and Alexa Skills, providing each action and skill with the same commands (framed as intents) and example utterances that were given to our model. We then ran each test utterance through both intent routing systems and scored their accuracy, again giving half credit in cases when either the verb or object were correct and full credit when both were. We provided our test cases as text rather than speech to provide a fair comparison to our model and avoid any speech recognition errors. We found that the best version of our model on each environment outperformed the Google Voice Assistant and Amazon Alexa platforms as shown in Table 3.

5 CONCLUSION

In this paper we have presented strategies for creating effective speech interfaces for games. Our largest contribution in this paper is the set of command extraction strategies we used to map from the unbounded range of possible human inputs to a finite domain of predefined game commands. We have shown that our methods outperformed the Amazon Alexa and Google Voice Assistant platforms on the classification task described. We have also demonstrated that *Correlation* as a distance metric outperformed the standard *Cosine distance* on our classification task. We believe that these findings are significant steps towards production-ready interfaces that will take advantage of unstructured speech to increase player immersion and engagement in a wide range of games.

We deliberately did not give our model any prior knowledge of the likelihood of each command. In a production environment, game developers would likely get even better performance by letting game context further inform the classification process.

Like all technologies, our command extraction methods should be tested and improved in a fully integrated gameplay environment. We are actively seeking collaborators to integrate our model into full-fledged games for play testing.

REFERENCES

[1] Amazon. 2015. Amazon Alexa Official Site. https://developer.amazon.com/en-US/alexa/ [Accessed January 2020].

[2] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics* 5 (2017), 135–146.

[3] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised Learning of Universal Sentence Representations from Natural Language Inference Data. *CoRR* abs/1705.02364 (2017). arXiv:1705.02364 http://arxiv.org/abs/1705.02364

[4] WoWWiki contributors. 2004. Quotes of Warcraft III. https://wowwiki.fandom.com/wiki/Quotes_of_Warcraft_III [Accessed September 2019].

[5] Rafal Cymerys. 2018. We Designed Voice User Interface for StarCraft II - Here's What We Learned. https://upsidelab.io/blog/design-voice-user-interface-starcraft/ [Accessed January 2020].

[6] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American society for information science* 41, 6 (1990), 391–407.

[7] Nancy Fulda, Daniel Ricks, Ben Murdoch, and David Wingate. 2017. What Can You Do with a Rock? Affordance Extraction via Word Embeddings. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. 1039–1045. https://doi.org/10.24963/ijcai.2017/144

[8] Evgeniy Gabrilovich, Shaul Markovitch, et al. 2007. Computing semantic relatedness using wikipedia-based explicit semantic analysis.. In *IJcAI*, Vol. 7. 1606–1611.

[9] Google. 2016. Google Assistant. https://developers.google.com/assistant [Accessed January 2020].

[10] Daniel Guo, Gokhan Tur, Wen-tau Yih, and Geoffrey Zweig. 2014. Joint semantic utterance classification and slot filling with recursive neural networks. In *2014 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 554–559.

[11] Patrick Haffner, Gokhan Tur, and Jerry H Wright. 2003. Optimizing SVMs for complex call classification. In *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03).*, Vol. 1. IEEE, I–I.

[12] Eui-Hong Sam Han and George Karypis. 2000. Centroid-based document classification: Analysis and experimental results. In *European conference on principles of data mining and knowledge discovery*. Springer, 424–431.

[13] Susumu Harada, Jacob O Wobbrock, and James A Landay. 2011. Voice games: investigation into the use of non-speech voice input for making computer games more accessible. In *IFIP Conference on Human-Computer Interaction*. Springer, 11–29.

[14] Anna Huang. 2008. Similarity measures for text document clustering. In *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, Vol. 4. 9–56.

[15] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *CoRR* abs/1301.3781 (2013).

[16] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. 2013. Linguistic Regularities in Continuous Space Word Representations. Association for Computational Linguistics.

[17] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. 1532–1543. http://aclweb.org/anthology/D/D14/D14-1162.pdf

[18] Robert E Schapire and Yoram Singer. 2000. BoosTexter: A boosting-based system for text categorization. *Machine learning* 39, 2-3 (2000), 135–168.

[19] Mike Treanor, Nicholas Warren, Mason Reed, Adam M. Smith, Pablo Ortiz, Laurel Carney, Loren Sherman, Elizabeth Carré, Nadya Vivatvisha, D. Fox Harrell, Paola Mardo, Andrew Gordon, Joris Dormans, Barrie Robison, Spencer Gomez, Samantha Heck, Landon Wright, and Terence Soule. 2017. Playable Experiences at AIIDE 2017. In *Proceedings of The Thirteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-17)*. Association for the Advancement of Artificial Intelligence, Snowbird, Utah. https://pdfs.semanticscholar.org/19f9/a76f6edcc6aa41bf19dba017da8c1c01e2b3.pdf

[20] Matthew Turk and Alex Pentland. 1991. Eigenfaces for recognition. *Journal of cognitive neuroscience* 3, 1 (1991), 71–86.

[21] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. 2019. SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python. *arXiv e-prints*, Article arXiv:1907.10121 (Jul 2019), arXiv:1907.10121 pages. arXiv:1907.10121 [cs.MS]

[22] Daniel Wilson. December 2015. Mayday! Deep Space! https://en.wikipedia.org/wiki/Mayday!_Deep_Space.