Interpolating Humour – Can Lines Be Funny?

Authors

Oscar De Leon (MacEwan University; deleono@mymacewan.ca)
Isaac McCracken (MacEwan University; mcrackeni@mymacewan.ca)
Kevin Ulliac (MacEwan University; ulliack@mymacewan.ca)
Calin Anton (MacEwan University; antonc@macewan.ca)
DOI: 10.1145/3774399.3774405

Introduction

The use of AI to play games has a long history. Some of the earliest examples were deterministic, fully observable games with clearly defined rules and possible game states. AI has proven best in games of this nature. Games such as checkers are considered solvable (Schaeffer et al., 2007) because there is always an optimal move to be made at any given point in the game, which means an AI agent can mathematically learn the best move.

On the opposite side of the spectrum, there are games like Apples to Apples, where AI is not so easily applied since there are elements that cannot be straightforwardly quantifiable and thus are not amenable to mathematical approaches. The goal of Apples to Apples is to satisfy the preferences of the judging player by playing the ”best” card of their seven red cards - nouns, which should match the judge’s green card - an adjective. The game involves dealing seven red cards from a randomly shuffled deck to each player. Each round, a new judge is chosen to draw a green card, whereby all other players will play one red card to match the judge’s green card best. The judge chooses a winning red card based on their own criteria of what is the “best” red card in play. The judge awards one point to the chosen winner, and this cycle continues until one person wins the total number of points needed to win the game. Apples to Apples is an interesting case because it involves three factors that make it far different from games like chess: randomness, use of natural language, and subjectivity. Games with any of these elements are challenging because simulating subjective preferences and determining the “best” choice to make is complex and difficult to quantify.

Research Question

Our goal is to create an Apples to Apples AI agent that can quickly and effectively learn the preferences of its opponent judges to win more games. For this, the AI agent must be able to model and appeal to various kinds of criteria, such as subjective measures like hu mour. We also want our AI agent to have its own unique preferences and to make decisions that are logically sound and mimic human creativity and humour in some way. Human creativity and humour are exceptionally difficult to model but are core attributes of these types of multiplayer word games. This leads us to some very difficult technical questions. Are there ways to quantify human elements like creativity and humour? Are there existing models that we can leverage and/or modify? A review of existing models did not appear to make full use of the vector dimensionality that encodes the semantic meaning of words. Therefore, to address the questions above, we hypothesize that using Linear Interpolation on word embedding vectors on an element-by-element basis provides an effective estimate of each opposing player’s preferences and, therefore, is an effective model for AI learning in Apples to Apples.

Problems to Solve

To address the research questions, we need to solve several challenging problems. The first major challenge in creating an AI agent for Apples to Apples is representing words as numerical values - word embeddings. By encoding words into multidimensional vectors, an agent can calculate the “best” red card to choose as both a judge and a player. The agent must model the preferences of all op ponents and select cards based on what is in their hand and in play. We also need a way for players to choose a winning red card as the judge based on their modelled pref erences. Additionally, we must recreate the game logic, pre-process all cards, store and update the game state, and record the agent’s performance for analysis. Lastly, we need to optimize our agents to win as many games as possible.

Background

Encoding and Processing Natural Language

Turney & Pantel (Turney & Pantel, 2010) discuss the Word-Context Matrix, where words are represented as vectors, and each dimension represents a type of context in which a given word could be found. The value of each dimension is proportional to the frequency that the given word appears in that context; the frequency also determines the word’s meaning. Using differing clustering techniques, they extracted some meaning from the target word, suggesting that the relationship between verbs and nouns can play a role in determining both a verb and a noun’s meaning.

Mikolov et al. (Mikolov, Chen, Corrado, & Dean, 2013) proposed two novel word em bedding models: Continuous Bag-of-Words (CBOW), which averages the word vectors in a context and predicts the current word based on the context, and Continuous Skip-gram, which predicts surrounding words based on the current word. They tested their model’s effectiveness through a word similarity task, demonstrating significant improvements over previous techniques. CBOW and Skip-gram are the primary algorithms for creating word embeddings in Word2Vec.

Aside from Word2Vec, other popular word embedding models include GloVe and Doc2Vec. Pickard (Pickard, 2020) describes how Word2Vec outperformed GloVe at multiword expressions, which are word combinations that exhibit one or more idiosyncrasies. Pickard also describes how Bidirectional Encoder Representations from Transformers (BERT) are used for similar Natural Language Processing (NLP) tasks, but it considers the sentence in which the words appear. The BERT model goes beyond what GloVe and Word2Vec use, so he omitted BERT from the comparison. Pickard found that Word2Vec outperformed GloVe by a substantial degree.

Related Work

Representing, Measuring, and Predicting humour

While most existing humour detection models use feature extraction to identify humorous words or phrases, Guo et al.(Guo et al., 2022) introduced an alternative approach called Fedhumour. This model uses BERT and fine-tunes pre-trained weights to update its understanding of humour based on individual preferences. Guo et al. demonstrated that Fedhumour outperformed other humour recognition models, including Doc2Vec (with the bag of words approach), Word2Vec (with a Random Forest classifier), and two BERT model variations.

Gultchin et al. (Gultchin, Patterson, Baym, Swinger, & Kalai, 2019) examined how word embeddings can represent word based hu mour. They used embeddings like GNEWS, WebSubword, Webfast, and WebGlove, all of which trained on different token sizes, to predict humour ratings from three datasets. Using a least-squares LR model, they could predict the mean humour ratings from the datasets for all four-word embeddings. GNEWS, with the smallest dataset, performed the worst. Despite this, all embeddings showed significant correlations

with humour ratings, proving their effectiveness. Cluster analysis also revealed demographic differences in humour, highlighting the utility of word embeddings in humour detection.

Weller & Seppi (Weller & Seppi, 2019) extended the AI's ability to determine whether a joke is humourous or not. Their agent learned to identify funny jokes based on ratings from Reddit's r/Jokes thread. They also refer to Convolutional Neural Networks (CNN) and another method which uses a transformer.

In their analysis of humour in the card game Cards Against Humanity, Ofer et al (Ofer & Shahaf, 2022) utilized the online version of the game to train several models to predict winning jokes, which focused primarily on the punchline cards. Their findings suggested that the context surrounding the punchline had minimal influence on the perceived humour compared to the punchline itself. Further more, short punchlines of 5 words or less were referred, with a 9% higher win rate.

Cosine Similarity

A common NLP method is Cosine Similarity, which calculates the cosine of the angle between vectors in multidimensional vector space. Cosine Similarity can be used to determine the relative similarity between words. For example, an output of 1 means the two vectors are identical, an output of 0 means they are orthogonal to one another, and an output of -1 means they are diametrically opposed to one another. This method seems plausible for generating two possible AI archetypes for the game of Apples to Apples: a literalist and a contrarian. The literalist would have a strong preference for similar word pairings. The contrarian would have a strong preference for dis similar word pairings. However, it is not clear how one could use Cosine Similarity to design effective preference types beyond those two, such as a preference for humourous card pairrings.

Methods

Word2Vec and Dataset Selection

During our research, we learned about Word2Vec, which vectorizes words into n dimensional vectors. Each dimension theoretically represents a feature of a given word, though the features are more abstract in nature. Word embeddings generated from models like Word2Vec have the useful property of placing semantically similar words close in vector space, allowing measures like Cosine Similarity to gauge word resemblance.

We chose Word2Vec for several reasons. First, it outperforms GloVe for multiword expressions, as noted in our background section. Second, BERT and Doc2Vec appear to be better suited for broader context language processing, which is not necessary for the short text on Apples to Apples cards. Lastly, we had access to Google's pre-trained Word2Vec model, trained on 100 billion words from Google News, providing 300-dimensional vectors for about 3 million words and phrases. This pre-trained model is ready to use and expected to cover all words in the Apples to Apples sets of cards.

However, using the Google News pre-trained Word2Vec model has drawbacks: the binary file is large (3.5 GB uncompressed) and is not customizable in its training. Apples to Apples has a more limited vocabulary than what is available in the Google News dataset, so most of the data is unused. A smaller, customized dataset would reduce file size and runtime computation. Furthermore, with a custom pre-trained dataset we could have more control over how the AI agents select word pairings. Despite these tradeoffs, we decided that the advantages of the pre-trained model outweighed the downsides.

Implementation

We hypothesized that instead of boxing every judge into a preference type and modelling those types only using Cosine Similarity or other similar methods, the agent should directly access the float values and leverage known machine-learning techniques or other mathematical or statistical tools. Thus, the agent effectively maps the player's preferences directly to the abstract encoding of meaning in the word

embeddings. We concluded we could combine the green and red card vectors by multiplying them, feature by feature, and that the most highly correlated features between the two would result in the highest values, while low correlated features would get much smaller values. Furthermore, we could use a simple linear equation y = mx + b where x represents whether the green and red card pairing was a winning one, y represents whether the green and red card pairing was a winning one (y = 1) or a losing one (y = -1). Therefore, the opponent's preferences could be modelled using the slope (m) and intercept (b). In summary, our agent interpolates its opponent's preferences as a linear function and uses linear regression to determine the slope and intercept of the decision boundary. We call this design a Linear Regression Agent (LRA).

Initially, the LRA represents the preferences of opposing players by assuming that they have the same preferences as itself. Therefore, the opponent models begin with the same slope and intercept vector values and are then replaced by the slope and intercept vector values produced by LR on that opponent player's continuously updated history of green and red card pairings. The starting preferences are static.

Linear Regression Model

At runtime, we calculate the score using the LRA's model of the current judge. For each card, we calculate: score = compsum(mrg + b), where m is the judge's slope vector, r is the red card vector, g is the green card vector, and b is the judge's bias vector. The compsum function adds every component of the vector, thus producing a scalar. The LRA chooses the red card with the highest score.

We keep a history of all green and red card pairings and their respective word embed dings for each of the opponent judges. We then use LR on each combination of the green card, the winning red card, our initial/previous representation of the judge's modifying vector (m) and the judge's bias (b); LR is performed on each vector dimension individually, and the resulting linear model allows us to calculate how far off our representation of the judge's preferences was. We then update that judge's slope vector and bias vector accordingly based on the winning and losing cards' vector values.

LRA Archetypes

For testing purposes, we devised 3 AI archetypes: A Literalist (Ltrls) - who likes highly associated words that are logical to pair together; a Contrarian (Cntrn) - who likes the least associated words, and a Comedian (Cmdn) - who likes humourous combinations. We created each model archetype in different ways. The Literalist's slope vector values were set to positive 1s, and the bias vector values were set to 0s. With this model, the more closely related a pair of green and red cards is, the more similar each component in word embedding will be. For example, when the same component is similar between two words, they will both be positive or both be negative, so multiplying them together will result in a positive number, and more of these similar components result in a higher score value. In this way, we use the embedding directly to achieve the highest association value. Conversely, the Contrarian's slope vector values were set to negative 1s, and their bias vector values were set to 0s. This results in the model choosing the lowest association values.

For the Comedian archetype, slopes and biases were calculated using LR on preselected green and red card pairings. For each green card, we manually selected a "winning" red card that was humourous as a pairing and a "losing" red card that was not and then serialized them. We did Linear Interpolation on all the components upon start-up, producing a static linear model that should emulate the preferences of the card pairings chosen in the file. We used 30 pairs for testing. This was the minimum number of pairings needed to reasonably emulate a humour preference. We chose winning pairings based on what we considered to be "funny" pairings, such as the green card "disgusting" with the red card "my high school prom." We chose losing pairings that

seemed unlikely to appear humourous to most people, such as the green card "disgusting," with the red card "nosebleeds."

Performance Improvements

Following our implementation, we investigated improvements that could be made to enhance the LRA's performance. The LRA solely considered the green and red cards and the judge's preferences. There are many potential factors that could have improved the LRA's performance. We identified one such major factor: we were resetting the models in between every game. With this setup, the LRA performed no better than chance and many times performed even worse, getting an aggregate LRA round win rate of less than 50% and sometimes a game win rate of less than 40%.

When multiple games of Apples to Apples are played, and no new players are introduced between games, humans carry over their knowledge of another player's preferences between games. We implemented a similar setting for the LRA, allowing them to preserve learned opposing model vectors between games. This improved the LRA's performance by around a 10-15% increase in aggregate round win rate. We added a feature to the LRAs to store the red words that were revealed in each round but not chosen by the judge; we refer to these as losing red cards. The observation was that the losing red cards have just as much information about the opponent judge's preferences as the winning red cards. We modified the agent so that after each round, the green card, the accompanying winning red card and all the losing red cards were stored in the model as an aggregate set of data points. We take the component-wise product of the green card vector with all combinations of the losing red card vectors, just as we do with the winning red card vector, but the corresponding y values are set to negative 1s instead of positive 1s. To match this new implementation, we added 30 losing red card pairings in addition to the 30 existing Comedian preselected green and winning red card pairings.

After fully implementing this change, there were some noticeable differences. The LRA's aggregate win rate increased significantly, but the program's runtime grew exponentially with each round, taking longer to process the losing red cards. However, we concluded that the increase in win rate outweighed the impact on performance, so we opted to keep this change.

Experiments

In all test cases, we ran games with the same number of LRAs and random agents (Rand). This ensures a balance of LRAs to random agents, which avoids any unfair advantages for the LRAs. If the LRAs are able to win more rounds than the random agents with statistical significance, it would be plausible to conclude that they exhibit partial learning the preferences of opponent players. We ran 22 sets of 100 games, with varying points to win and different numbers of players in the game, to determine whether the LRA could perform consistently well across many different scenarios. We played all possible combinations of games with either 5, 7, or 10 points to win and 4, 6, 8, or 10 players, which makes up 12 different game scenarios. We played at least one of each of the 12 different game scenarios; the remaining 10 of the 22 games were used to experiment with different LRA archetypes. Some games had a single LRA archetype, while others had differing archetypes. The total possible combinations between game scenarios and combinations of LRA archetypes were much too large to test, so some variations were not included due to their similarity with other game scenarios or how long they would take to run. These dynamic elements in our games are our manipulated variables.

We used the same game settings across all variations of the game sets: the same player types are used throughout all 100 games in each set, the judge for each round is cycled, which follows the game rules, and the starting judge for each game is also cycled. As mentioned previously, we did not reset the models representing the LRA's opponents within any set of 100 games. This means the models are continuously updated for 100 games until the program ends, and

only when the program is run again will these values be reset back to the default initialized values. The losing red cards feature was used for all games. We used all the base sets and expansion sets for both the green cards and red cards rather than limiting the deck sizes to that of a normal Apples to Apples deck size. All these elements of the game are our control variables and were consistent throughout all sets of 100 games.

Results

Win Rates

The results show that our agent is very capable of interpolating a static preference type. For all Round Summary tables, we list the total round win counts and total round win rates for the entire 100 game set. We also provide mean wins per game, and the standard deviation, and the confidence intervals and p values for the round win rates. For all Game Summary tables, we list the total game win counts, total game win rates for the entire 100 game set, and the confidence intervals and p values for the game win rates.

We calculated 95% confidence intervals for each agent's win rate using a normal approximation to the binomial distribution. The confidence interval quantifies uncertainty in the win rates observed over the total number of games. A z-score of 1.96 was used to compute the intervals. These intervals were used to assess the reliability of the win rate estimates before performing hypothesis tests.

The p-values were calculated using a one tailed binomial test. The alternative hypothesis assumed that the win rates for our LRA were larger than the random agent's win rate. A significance level of 0.05 was used for the tests. In the tables, a p-value of 0 signifies $p < 0.00001$. For each agent, the number of wins was compared to the total number of rounds/games played, and the corresponding values were determined to evaluate whether the observed win rates were larger. All game sample sizes were n=100 while round sample sizes varied depending on the total number of points to win, the number of players in the game, and how fast a player was able to win the game. The round sample sizes varied between n=1137 and n=4905. We observed a trend where the total number of players in the game is positively related to aggregate LRA agent round wins. This trend holds true for all points-to-win variations. Table 1 shows that for 100 games, 10 points to win, 5 LRAs and 5 random agents, the LRA aggregate round win rate is 68.14%, while Table 2, with 100 games, 10 points to win, 2 LRAs and 2 random agents, shows an LRA aggregate round win rate of 61.12%.

Player	Win Cnt	Win %	Mean Wins per Game	Std Dev	Conf Int	p-Value
LRA Model Cmdn 1	664	13.54	6.64	2.156	12.58 to 14.49	0
LRA Model Cntrn 1	692	14.11	6.92	2.378	13.13 to 15.08	0
LRA Model Cntrn 2	666	13.58	6.66	2.732	12.62 to 14.54	0
LRA Model Ltrls 1	663	13.52	6.63	2.560	12.56 to 14.47	0
LRA Model Ltrls 2	657	13.39	6.57	2.495	12.44 to 14.35	0
Rand Ag 1	302	6.15	3.02	1.833	-	-
Rand Ag 2	300	6.12	3.00	1.892	-	-
Rand Ag 3	339	6.91	3.39	1.984	-	-
Rand Ag 4	306	6.24	3.06	1.719	-	-
Rand Ag 5	316	6.44	3.16	2.023	-	-

Table 1: Round Summary for 100 games, 10 points to win, 5 LRAs and 5 random agents

Player	Win Cnt	Win %	Mean Wins per Game	Std Dev	Conf Int	p-Value
LRA Model Cntrn 1	350	30.17	3.50	1.520	27.53 to 32.81	4e-5
LRA Model Ltrls 1	359	30.95	3.59	1.357	28.29 to 33.61	0
Rand Ag 1	233	20.09	2.33	1.607	-	-
Rand Ag 2	218	18.79	2.18	1.627	-	-

Table 2: Round Summary for 100 games, 10 points to win, 2 LRAs and 2 random agents

We used statistical analysis on the round

win rates and game win rates to determine the confidence intervals and p-values. For the round wins, we found that with 100 games, most of the p-values for the LRAs were well below 0.05, which was especially true for a higher number of players and for a higher number of points to win. The highest p-value obtained was 0.05877.

The p-values for game win rates exhibit a different behaviour. In a game set with 10 players per game, the highest p-value of one LRA was 0.21824; for another LRA was 0.07257, but most of the p-values across all game sets were below 0.05, many of which were much closer to 0. As discussed earlier, all our 22 test sets comprised 100 games; increasing the total number of games played in each set would have decreased the LRA's p-values, but that would have required longer processing time.

As shown in Table 3 in the case of only four players, our LRAs still had good results, but less so compared to when there were more players and points to win. This demonstrates that even at one of their worst performances, the LRAs still performed high above the probability of random chance. Other tests also portrayed similar outcomes in both the round and game win rates. The game set for Table 3 has the fourth lowest round and game win rate out of all 22 variations we did during testing, further proving how consistent and well the LRAs play. The lowest score for both game and round win rates during testing came from a set of 100 games, with 5 points to win, 2 Comedian LRAs, and 2 random agents, where the aggregate LRA game win rate was 75%, and the round win rate was 59.40%.

Player	Win Cnt	Win %	Mean Wins per Game	Std Dev	Conf Int	p-Value
LRA Model Cmdn 1	346	29.67	3.46	1.513	27.05 to 32.30	1.7 e-4
LRA Model Cntrn 1	355	30.45	3.55	1.519	27.80 to 33.09	1e-5
Rand Ag 1	227	19.47	2.27	1.475	-	-
Rand Ag 2	238	20.41	2.38	1.580	-	-

Table 3: Round Summary for 100 games, 5 points to win, 2 LRAs and 2 random agents

Table 4 provides further evidence that as the player count and/or number of points to win goes up, so does the aggregate LRA performance. The game set for Table 4 was a particularly interesting set because there was only one of each LRA archetype in the game, which meant that the LRAs could potentially deviate from their initial understandings of their opponents' preferences since all archetypes start every set of games under the assumption that all other players have the same preferences as their own. The fact that this game set has such high win percentages demonstrates that the LRAs could identify opposing players as having completely different preferences from their own at some point throughout the set of games.

Player	Win Cnt	Win %	Confidence Interval	p-Value
LRA Model Cmdn 1	33	33.00	23.78% to 42.22%	5e-5
LRA Model Cntrn 1	28	28.00	19.20% to 36.80%	3.1e-3
LRA Model Ltrls 1	34	34.00	24.72% to 43.28%	2e-5
Rand Ag 1	2	2.00	-	-
Rand Ag 2	2	2.00	-	-
Rand Ag 3	1	1.00	-	-

Table 4: Round Summary for 100 games, 10 points to win, 3 LRAs and 3 random agents

Judge Choices

Throughout our development process, we

monitored each judge agent's preferences to declare other agents as the winner, and we represented those choices as a heatmap. In Figure 1, we provide an example of such a heatmap; this is the heatmap for the same 100-game set displayed in Table 1 (10 points to win, 5 LRAs and 5 random agents). On the y-axis, we have all players in the game as judges; on the x-axis, we have all players in the game as regular players. The value in the matrix portion denotes the percentage of rounds for which the judge (y-axis) chose each player (x-axis) as the winner of that round. The color scaling (white to black) ranges from 0 to 28.8%, and are the lower and upper bounds, respectively, of the total win selections. Of the 22 game sets that were simulated, every heatmap had a similar distribution to Figure 1.

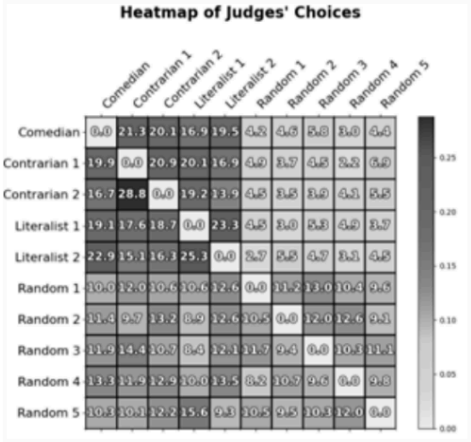

Figure 1: Heatmap of Judge Selections for 100 games, 10 players, 10 points to win, 5 LRAs, 5 random agents. Row labels represent each individual judge. Column labels represent the opponent players chosen as the winner. The value intersecting rows and columns represent the percentage that the judge chose that given player as a winner.

Discussion

Overall, our results indicate that the agents we implemented were successful in playing Apples to Apples. Our Linear Interpolation model showed promising potential for a significantly high round and game win rate, which appears to indicate that our LRAs were able to learn opposing player preferences. Although we wanted the Comedian to make creative choices to achieve humour, we cannot assert that we succeeded in this regard since humour is subjective. The LRA's ability to be humourous could have partially been due to the limited number of cards at their disposal in each given round. A more thorough measurement of how humourous the LRAs are would be to test them against many human players with differing preferences, but this is beyond the scope of our project. Further research could yield more significant findings in this regard.

Further Considerations

Continuing our research into this topic would entail experimenting with BERT for word embeddings, as it excels in context-dependent material, which could be helpful for humour analysis. This could improve our agent's ability to interpret meaning from context. Additionally, training our own Word2Vec model using custom datasets will allow us to reduce the number of vector dimensions, thus lowering computations during training or increasing them to represent more features of a word.

We would also like to explore more ways to determine the most important features for accurately representing a judge's preferences. More research might lead to an effective solution, possibly choosing features dynamically throughout the game and tracking the most important ones for each judge separately.

Another possible improvement involves tracking all red cards played in each round, even when the LRA is a judge. This could help us understand if opponent judges used the same criteria for choosing a red card as they do for selecting from their hand. Collecting these extra data points to regress linearly could potentially help our agents converge faster on each opponent's preferences.

Conclusion

We gathered the results and investigated a novel approach for AI agents to play Apples to Apples using Linear Interpolation to estimate which red card a given judge would choose as the "best." Additionally, we aimed

for the LRAs to learn their opponents' preferences and perform well in Apples to Apples and tried many variations to optimize the aggregate LRA win rate. Our analysis showed that the LRAs were able to achieve high round win rates and very high game win rates, all of which were statistically significant. Although we could not extrapolate how well the LRAs would perform against human players, our results demonstrated potential for a new, unique approach to word association games that provides insight into the area of automated playing of natural language card games.

References

Gultchin, L., Patterson, G., Baym, N., Swinger, N., & Kalai, A. (2019, 09–15 Jun). Humor in word embeddings: Cockamamie gobbledegook for nincompoops. In K. Chaudhuri & R. Salakhutdinov (Eds.), *Proceedings of the 36th international conference on machine learning* (Vol. 97, pp. 2474–2483). PMLR. Retrieved from https://proceedings.mlr.press/v97/gultchin19a.html

Guo, X., Yu, H., Li, B., Wang, H., Xing, P., Feng, S., . . . Miao, C. (2022, May). Federated learning for personalized humor recognition. *ACM Trans. Intell. Syst. Technol.*, *13*(4). Retrieved from https://doi.org/10.1145/3511710 doi: 10.1145/3511710

Mikolov, T., Chen, K., Corrado, G. S., & Dean, J. (2013). Efficient estimation of word representations in vector space. In the International *conference on learning representations.* Retrieved from https://api.semanticscholar.org/CorpusID:5959482

Ofer, D., & Shahaf, D. (2022, December). Cards against AI: Predicting humor in a fill-in-the-blank party game. In Y. Goldberg, Z. Kozareva, & Y. Zhang (Eds.), *Findings of the association for computational linguistics: Emnlp 2022* (pp. 5397–5403). Abu Dhabi, United Arab Emirates: Association for Computational Linguistics. Retrieved from https://aclanthology.org/2022.findings-emnlp.394/ doi: 10.18653/v1/2022.findings-emnlp.394

Pickard, T. (2020, December). Comparing word2vec and GloVe for automatic measurement of MWE compositionality. In S. Markantonatou et al. (Eds.), *Proceedings of the joint workshop on multiword expressions and electronic lexicons* (pp. 95–100). online: Association for Computational Linguistics. Retrieved from https://aclanthology.org/2020.mwe-1.12/

Schaeffer, J., Burch, N., Bjornsson, Y., Kishimoto, A., Muller, M., Lake, R., Sut- ¨ phen, S. (2007). Checkers is solved. *Science*, *317*, 1518 - 1522. Retrieved from https://api.semanticscholar.org/CorpusID:10274228

Turney, P., & Pantel, P. (2010, 03). From frequency to meaning: Vector space models of semantics. *Journal of Artificial Intelligence Research*, *37*. doi: 10.1613/ jair.2934

Weller, O., & Seppi, K. (2019, November). Humor detection: A transformer gets the last laugh. In K. Inui, J. Jiang, V. Ng, & X. Wan (Eds.), *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (emnlp-ijcnlp)* (pp. 3621–3625). Hong Kong, China: Association for Computational Linguistics. Retrieved from https://aclanthology.org/D19-1372/ doi: 10.18653/v1/ D19-1372

Oscar De Leon is currently studying for his Bachelor of Science at MacEwan University. He is majoring in Computer

Science and minoring in Psychology. His fields of interest include database management systems and artificial intelligence. Additionally, he wants to apply his interests in behavioural and social psychology to computer science in order to help humans better understand each other.

Isaac McCrackenis an undergraduate student in Computer Science at MacEwan University. His academic interests include programming languages, compiler design, mathematics, and artificial intelligence. He is particularly focused on exploring the theoretical and practical aspects of software development.

Kevin Ulliac graduated from MacEwan University with a Bachelor of Arts in Philosophy and a minor in Psychology in Spring 2017, and more recently with a Bachelor of Computer Science with a minor in Mathematics in April 2025, also from MacEwan. His major areas of interest are in machine learning and artificial intelligence, data science/analysis, music-related software like plugins and DAWs, and standalone/desktop applications including custom debugging tools. He also has a passion for linear algebra and calculus and their application in software and engineering.

Calin Anton holds an MSc from the University of Bucharest, Romania and a Ph.D. from the University of Alberta, Canada, both in Computer Science. He has been teaching in different roles at postsecondary institutions for more than 25 years. For the last 15 years, he has taught at MacEwan University in Edmonton, Canada, where he is currently an associate professor of Computer Science. His current interests reside in Computer Security and Artificial Intelligence.