Word-Embedding Data-Driven Evaluation Measures

Matan Solomon$_1$
322853334

Rotem Ben Zion$_2$
211324975

$\{^1$matan-so, ^{2}rotm$\}$@campus.technion.ac.il

Abstract

The development of deep learning in recent years has led to an extensive practical use of word-embedding across many domains, and along with it, the need to correctly assess and evaluate the performance of embeddings methods. In this project, we suggest a family of evaluation measure which are based on human data in the framework of the famous game Codenames. We test our new evaluation measures and use them to compare the performances of well-known word embeddings.

1 Introduction

Deep learning has been the focus of many research in the recent years, motivated by the need to capture complicated functions that model complex relations from the real world. One of the most investigated topics is the field of natural language processing (NLP), which deals with developing models to understand and interact with natural language. New method and models arise rapidly, and it has become hard to differentiate between human and machine. One thing that most of the models have in common is the usage of word-embedding. In general, word-embeddings are representations of words in highly dimensional vector spaces, trying to capture the complex structure of words and the inner relations between them. The exhaustive use of word-embedding in most of the recent models has led to the need to reliably assess their performance. Can a word embedding method really capture the natural language? If so, how good? These questions are the focus of nowadays research.

In this project, we would like to suggest a family of new evaluation measures for word embedding. Those measures use human data to evaluate the extent to wich an embedding-based agent can cooperate with human agents. The measures we suggest are motivated by the fact that word embeddings try to capture the natural languages used by humans, so the measure should assess its performance by examining the way it interact with humans.

The framework in which we examining the interaction between a word embedding to humans, is a version of the famous card game Codenames. The game Codenames, which we will explain in details in the next section, requires understanding of the communication strategy of other players, and the ability to interact, using languages, with the other players. These two characteristics makes it a suitable framework for testing an embedding method. By replacing some (or all) of the players with an embedding based agent, we can test its ability to succeed in the game.

1.1 Codenames

Codenames is a cooperative card game between two agents, where one (called ”Sender”) has information the other (”Receiver”) doesn’t. We used the following version of Codenames. On the board, seen by both Sender and Receiver, are $\mathcal{T}$ words from a vocabulary. $\mathcal{G}$ of these words are blue (”good”), and the other $\mathcal{T} - \mathcal{G}$ are red (”bad”). The Sender knows the colors and the Receiver doesn’t. The sender must send a single word from the vocabulary as a clue with the goal of passing as much information as possible.

- **Sender** chooses a word c in the vocabulary which isn’t on the board, and a natural number $\tau \in \{0, 1, \ldots, \mathcal{T}\}$. We denote to the τ words the clue refers to as the ”targets”.
- **Receiver** receives c, τ and chooses τ words from the board. These are called the guess.

The objective of both players is to maximize the amount of blue words and minimize the amount of red words in Receiver’s guess. For the original rules of the game, see instruction on Wikipedia (wik).

The project is structured as follows: In section 2 we preset related work and usages of Codenames as an evaluation measure. Section 3 is devoted to the different agents and strategies we suggest, where in section 4 we present the evaluation methods and their results on those agents. In section 5 we describe the different experiments we conduct with the suggested measures, and we finally conclude in section 6.

2 Related work

In the last years, there were several attempts to develop word-embedding evaluation measures. Kim et al. (2019) defines a Receiver equivalent to our EmbeddingNearestReceiver we describe later, and a Sender that chooses clues based on their distance to red words, constrained to some threshold. They run experiment simulations between the agents by a round-robin tournament and find that GloVe and word2vec are both superior to WordNet, though they do not test their models in comparison to humans as we do in this project. Bard et al. (2020) used reinforcement learning to train playing bots. They found that bots that had a paired partner were able to play nearly perfectly, but when paired with a partner that did not have the same communication strategy the bots performed horribly. This result is in line with our test that is detailed section 3.3. Koyyalagunta et al. (2021) defines language-graph-based agents and develops the DETECT metric, designed to re-weight word similarities in embedding space and in knowledge graphs anywhere that word representations are used. Koyyalagunta et al. (2021) also provided us with their collected dataset, as detailed in section 4.1.

3 Our Agent and Strategies

To evaluate the embeddings, we must implement agents that use the embeddings in the framework of Codenames. For that, we had to create an API version of the game, in which the virtual agent could play with each other and with human, and in which we could test their performances. In this section, we describe the different agent and their strategies, followed by a proof of concept to the idea of using Codenames. From here on, we use the distance between words, to denote the distance (euclidean / cosine) between the embedding vectors representing the words in the vector space.

3.1 Agents

We chose to implement two families of Sender agents and a single, intuitive Receiver agent.

3.1.1 ClusterSender

The cluster sender operates as follows. Given blue words $B \subset V$ and red words $R \subset V$, so the board words are $W = B \uplus C$, it creates many clusters using k-means with random initialization points in the embedding space. Out of all clusters $C \subset W$ which only contain blue word, i.e. $C \subseteq B$, take the largest cluster. The returned clue is the $w \in V$ closest to the centroid of that largest cluster (by cosine similarity or Euclidean norm), and the clue size is the size of that cluster. The pseudo code is provided in algorithm 1.

3.1.2 ExhaustiveSearchSender

Given blue words red words, for each word in the vocabulary, count the number of blue words closer to that word than any red word. The clue should be the word in the vocabulary (not including the board words) that maximizes that count. If multiple words achieve the maximal count, pick the clue according to the tiebreak rule which is a hyperparameter. The pseudo code is provided in algorithm 2.

The tie-break parameter has 4 different options:

- ”avg blue dist” – minimize the average distance to the targets
- ”max blue dist” – minimize the maximal distance to the targets
- ”max radius” – maximize the distance to the closest red word
- ”red blue diff” – maximize the margin between targets and red words

3.1.3 EmbeddingNearestReceiver

This is the most intuitive receiver, which chooses the closest words to the clue. The pseudo code is provided in algorithm 3.

3.2 ranked forward

For the evaluation of the agents, we require a slightly different output than a single clue and clue-size from Sender and clue-size words from Receiver. Rather, we want Sender to output the entire vocabulary, sorted such that the first word is the best clue for the given input and the last word is the worst clue. Similarly, we want Receiver to output

Algorithm 1 ClusterSender

```
1: Generate cluster $C_1, C_2, \cdots \subset W$
2: $size = 0$
3: $cluster = None$
4: for $C = C_1, C_2, \ldots$ do
5:     if $|C| > size$ and $C \subseteq B$ then
6:         $size = |C|$
7:         $cluster = C$
8:     end if
9: end for
10: $cent = centroid(cluster)$
11: Return $\arg\min_{w \in V} (w, cent),\ size$
```

Algorithm 2 ExhaustiveSearchSender

```
1: $counts = \{\}$
2: for $w \in V$ do
3:     $counts[w] = |\{b \in B : dist(b, w) < dist(r, w) \forall r \in R\}|$
4: end for
5: $optimals = \{w \in V : count[w] = \min counts\} \equiv \arg\min counts$
6: $tiebreaks = \{\}$
7: for $w \in optimals$ do
8:     $tiebreaks[w] =$ tiebreak$(w)$
9: end for
10: $result = \arg\min_{w \in optimals} tiebreaks[w]$
11: Return $result,\ counts[result]$
```

Algorithm 3 EmbeddingNearestReceiver

```
1: $dists = \{\}$
2: for $w \in V$ do
3:     $dists[w] = dist(w, clue)$
4: end for
5: Return the $clue - size$ words which minimize $dists$
```

the entire board as a list, sorted such that the first word best matches the given clue, and the last word least matches the clue. We define that operation as ”ranked forward”. Note that in the receiver’s case, this output structure requires a hidden assumption – that the receiver function treats words independently. This is not a viable assumption for human agents, but EmbeddingNearestReceiver does hold that assumption. These are the implementations of ”ranked forward” in our defined agents:

- ClusterSender chooses the centroid as explained, and ranks the vocabulary according to the distance from that centroid.
- ExhaustiveSearchSender ranks the vocabulary according to the number of blue words closer than the closest red word. Out of the optimal words, the arrangement is defined by ”tiebreak” .
- EmbeddingNearestReceiver ranks the board words according to their distance from the clue.

In all cases above, Taking the top word is equivalent to our earlier definition.

3.3 Motivation

After implementing the above API, we could firstly test the hypothesis that Codenames is the right framework to test the embeddings. We have implemented the function ”game_metric”, which takes a sender model Sender and a receiver model Receiver and simulates the Codenames game by iteratively taking a clue from Sender and giving it to Receiver, until all blue words disappear from the board, or a maximum number of iterations is reached. We used this metric on different combinations of word embeddings, with an ExhaustiveSeachSender with tie_break taking the first optimal clue (see section 3.1 for details) and the usual EmbeddingNearestReceiver. We generated a random board with 100 total words and 50 good words, and averaged the metric over 30 calculations.

The results, given in table 1, match the hypothesis that agents use the same embedding will perform better then agents use different embeddings. Without coordination the agents can’t complete the game after as many iterations as blue words. This effect of miscoordination between the agents lead us to the question: which embedding-based agents will cooperate well with humans?

Sender	Receiver	metric
Glove	Glove	9.3
Glove	Word2Vec	51
Word2Vec	Glove	51
Word2Vec	Word2Vec	8.7

Table 1: Different embedding-based agents combinations in a Codenames game. Low values of the metric indicates better performance.

4 Our Evaluation Measure

In this section we present two datasets: the Human Receiver Dataset, collected by Koyyalagunta et al. (2021) who where willing to share their data with us, and the Human Sender Dataset, which we collected ourselves using Prolific (Pro). We discuss the motivation behind the Human Sender Dataset and each dataset’s advantages and disadvantages. For each of the datasets, we suggest two evaluation measures, one for each agent (Sender/Receiver).

4.1 Human Receiver Dataset

4.1.1 Dataset

The data collected by Koyyalagunta et al. (2021) was generated as follows. After generating 60 random Codenames boards, each of their predefined sender agents (”initial models”) gave a clue to each board. Human annotators at Amazon Mechanical Turk (AMT) picked which board words are best associated with that clue, without knowing the true word colors. We call this dataset ”Human Receiver Dataset” because the human annotators acted as the receiver agent. After some preprocessing, the data structure given at table 2.

4.1.2 Sender measure

Our goal regarding the sender model, as stated in the introduction, is to evaluate how well it gives human like clues. For this dataset, we propose to do so heuristically by retrospectively selecting the colors of the board words according to the AMT worker’s response. This can be done as follows: Using the sender’s function `ranked_backwards` (defined in senction 3.2), we calculate the average rank of the clue words generated by the initial models. However, the input to the sender isn’t the same as the input to the initial model: the board words picked by the AMT worker are defined as the blue words, and all the other words are defined to be red. This means that every blue word targeted by the initial model that isn’t picked by the AMT worker is

Initial model	Clue word	Word1	$\cdots$	Word20	Rank1	$\cdots$	Rank3
bertHKimFX	animal	dog	$\cdots$	airplane	dog	$\cdots$	mouse

Table 2: Koyyalagunta et al. (2021) Human receivers dataset

turned into a red word and every red word "wrongfully" chosen by the AMT worker is turned into a blue word, thus creating a board in which the initial model's clue successfully delivered all blue words. The pseudo code is provided in algorithm 4.

The idea behind this heuristic approach is that since the AMT worker chose all the blue words given the clue, it should be ranked well in the sender model we are testing. Notes:

- This measure is very sensitive to the size of the agent's vocabulary. Since word2vec and GloVe have different vocabularies in our code, we redefined all agents' vocabulary to be the intersection between GloVe and word2vec.
- Koyyalagunta et al. (2021) also included the clue targets in the dataset, which we didn't use, as we do not want to rely on a non-human model (more than necessary) in our evaluation model.

4.1.3 Receiver measure

Our goal regarding the receiver model, as stated in the introduction, is to evaluate how well it receives clues generated by humans. To do so with the Human Receiver Dataset, we chose a metric commonly usen in information retrieval: the Mean Average Precision (MAP). In information retrieval terms, we have a search engine S, a set of queries Q and a set of documents D. For each query $q \in Q$, $rel^q : D \to \{0,1\}$ indicates whether a document is relevant for that query. We define $R^q \equiv \sum_{d \in D} rel^q(d)$ the number of relevant documents to q. $S(q)$ is a list of documents the search engine believes are relevant to q, sorted by confidence. Then the MAP evaluation of S is defined as:

$$MAP = \frac{\sum_{q \in Q} AP(q)}{|Q|}$$

$$AP(q) = \frac{1}{R^q} \sum_i rel^q \left(S(q)^i \right) \cdot P@i$$

Put to words, the Average Precision sums the precision values at every relevant document's position in $S(q)$, normalized by the amount of relevant documents. Applied to out problem, the Receiver model serves as the search engine, the clue generated by the initial model is the query, the board words are the set of documents, and the board words chosen by the AMT worker as relevant to the clue – are relevant documents. The pseudo code is provided in algorithm 5.

4.1.4 Disadvantages of the Human Sender Dataset

As its name suggests, the main feature of the Human Receiver Dataset is that the receiver in the generated data is human. Therefore, this data is a perfect match if we wish to evaluate the initial models used in its generation process. However, when we wish to evaluate whether a new sender model creates human-like clues, or whether a receiver model perform well when paired with a human receiver, this dataset has the two following major issues:

1. The quality of our evaluation measure is greatly affected by the quality of the initial model used to generate the dataset. If the initial models created clues without any similarity to the words on the board, even after the re-coloring in our evaluation measure, we won't have reliable word-clue matchings to work with. This type of problem also exists in the areas of passive reinforcement learning (Ostrovski et al., 2021) and logged bandit feedback (Swaminathan and Joachims, 2015).
2. With the goal in mind that the evaluation of the receiver tests whether it understand human clues, evaluating the receiver model based on information from human receivers will only work if humans are perfectly good at understanding human clues. That obviously isn't a feasible assumption.

These two issues motivated us to create a new dataset, Human Sender Dataset, in which the humans play the sender role by picking the clue.

4.2 Human senders dataset

4.2.1 Dataset

Motivated by the issues of the first dataset presented above, we decided to construct our own

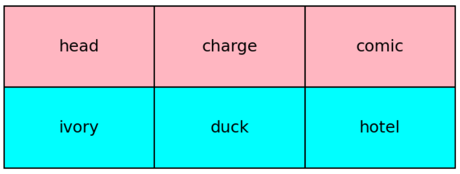
Algorithm 4 SenderEvaluation-HumanReceiverDataset

```
1: input: Sender
2: loss=0
3: for clue, board_words, chosen_words $\in$ HumanReceiverDataset do
4:     B $\equiv$ chosen_words
5:     R $\equiv$ board_word - B
6:     sender_ranks = Sender.ranked_forward(B, R)
7:     loss = loss + position(clue $\in$ sender_ranks)
8: end for
9: loss = $\frac{loss}{size(HumanReceiverDataset)}$
10: Return loss
```

Algorithm 5 ReceiverEvaluation-HumanReceiverDataset

```
input: Receiver
MAP_score=0
for clue, board_words, chosen_words $\in$ HumanReceiverDataset do
    B $\equiv$ chosen_words
    receiver_ranks = Receiver.ranked_forward(board_words, clue)
    ranks_of_B = {position(b $\in$ receiver_ranks): b $\in$ B}
    AP_score=0
    for index, rank$in$enumerate(ranks_of_B) do
        AP_score = AP_score + $\frac{index}{rank}$
    end for
    AP_score = $\frac{AP\_score}{|B|}$
    MAP_score = MAP_score + AP_score
end for
AP_score = $\frac{AP\_score}{size(HumanReceiverDataset)}$
Return AP_score
```

dataset using Prolific (Pro). The main idea is to create a dataset of human senders. We generated 45 different Codenames board with 6 words each, sampled from the original Codenames words, among them 3 randomly selected words were chosen to be the blue words. Example board is given in figure 1. We partitioned the generated boards into 5 different online forms and added an attention check - a board with 3 blue words which are all animals, and other 3 completely unrelated red words. We used the attention check to reject participants whose clue wasn't animal related. For each board, we asked the participants for a clue, and also for the blue words the clue tries to describe ("targets"). In total, there were 35 participants, each giving clues to 10 boards. An illustration of the final dataset after preprocessing is given in table 3.



head	charge	comic
ivory	duck	hotel

Figure 1: Example of a generated board

4.2.2 Sender measure

The idea of the sender evaluation measure for this dataset is very similar to the one from the previous dataset, except here the dependency on the human answer is more strict then before. Given the human senders dataset, we have the generated boards, and the clue a human annotator gave for this board. We would like to give the exact same board the human saw, to the sender agent, and see what is the rank of the clue the human gave in the ranked list the agent outputs (Using its `ranked_backward`). The average rank across all of the entries is the final evaluation measure. The pseudo code is provided in algorithm 6.

The idea of this measure is that a good agent sender will think like a human, and so give the same clues the human gave, given the same information.

4.2.3 Receiver measure

Recall we wish to evaluate how well the receiver model can understand clues given by humans. Unlike the Human Receiver Dataset, in this dataset the clues are given by humans, so the measure can be much simpler.

- We use the normal `forward` operation – the receiver chooses τ words from the board given a clue word and the number τ. Unlike the `ranked_forward` operation, `forward` doesn't assume the receiver treats the board words independently.
- We set penalty/reward values as hyperparameters.
- For each clue & board in the dataset, for each word the receiver chooses:
  - We penalize if the word is a red word.
  - We reward if the word is a target (the annotator mentions his targets).
  - We add some middle score if the word is a blue word but isn't a target (the annotator meant neither to hint to that word nor to avoid it).

The pseudo code is provided in algorithm 6.

5 Experiments

In order to experiment with our new evaluation measures, we conduct a set of experiment. In this section, we first describe the experimental setup, follows by the results from the suggested measures.

5.1 Setup

The function `calculate_results` in the appended code calculates the loss of each combination of `sender_type`, `sender_embedding` and `initial_model` on the HumanReceiverDataset. We ran this function on 7 Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz cpu cores in about 3 hours. The function `calculate_results_prolific` does the same for the HumanSenderDataset. The experiments we performed are as follows:

1. Agent type comparisons: we fixated all agents' embedding method to GloVe, and evaluated the different agent types (ExhaustiveSearchSender with different hyperparameters, ClusterSender). We have also implemented a random sender which chooses a clue randomly from the vocabulary and a random receiver which chooses a random guess from the board, as a bad baseline to sanity-check our measures.

2. Agent embedding comparisons: we fixated the sender type to be ClusterSender and the receiver type to be EmbeddingNearestReceiver,

Algorithm 6 SenderEvaluation-HumanSenderDataset

```
1: input: Sender
2: loss=0
3: for B, R, chosen_clue $\in$ HumanSenderDataset do
4:     sender_ranks = Sender.ranked_forward(B, R)
5:     loss = loss + position(chosen_clue $\in$ sender_ranks)
6: end for
7: loss = $\frac{\text{loss}}{\text{size(HumanSenderDataset)}}$
8: Return loss
```

Algorithm 7 ReceiverEvaluation-HumanSenderDataset

```
input: Receiver
Hyper-parameters:    TARGET_SCORE = 1    BLUE_SCORE = 0    RED_SCORE = -1
score=0
for B, R, chosen_clue, targets $\in$ HumanSenderDataset do
    words = B $\cup$ R
    $\tau$ = size(targets)
    receiver_choise = receiver.forward(words, chosen_clue, $\tau$)
    row_score=0
    for word $\in$ receiver_choise do:

        if word $\in$ targets then:
            row_score = row_score + TARGET_SCORE
        end if

        if word $\in$ B - targets then:
            row_score = row_score + BLUE_SCORE
        end if

        if word $\in$ R then:
            row_score = row_score + RED_SCORE
        end if
        row_score = $\frac{\text{row\_score}}{\tau}$
    end for
    score = score + row_score
end for
score = $\frac{\text{score}}{size(HumanSenderDataset)}$
Return score
```

BWord1	···	BWord3	RWord1	···	RWord3	Clue word	T1	···	T3
ivory	···	hotel	head	···	comic	expensive	ivory	···	NoMoreTargets

Table 3: Our Human senders dataset. BWord, RWord, TWord stands for Blue, Red amd Target Words, respectively.

and compared GloVe and word2vec. In all experiments on HumanReceiverDataset, we calculated scores with specific initial models and denoted 'all' to be the measure using the entire dataset. When using a specific initial model, we calculate the evaluation on the sub-dataset that contains only that initial model.

5.2 Results

5.2.1 Human Receiver Dataset

In figure 2, we compare the different sender classes that we have defined using our evaluation measures. The x-axis is some of the initial models provided in the HumanReceiverDataset, and "all" refers to the score calculated over the entire dataset. The loss is calculated according to 4.1.2. We can see that the random sender receives the highest loss by a large margin, and ClusterSender has the best evaluation. The fact ExhaustiveSearchSender receives higher loss can be accounted for by it's aggressiveness – if the human-like clue has less than the highest possible amount of blue words closer to it than the closest red word, it will instantly be pushed in the returned list of "ranked_forward" below any vocabulary words which do. Next, in figure 3, we fixate the ClusterSender type and compare GloVe to word2vec with the different initial models. A clear preference to word2vec seems to hold! This can be caused by the higher dimensionality (dimension 300 in word2vec versus dimension 100 in GloVe), or the embedding structure difference. Interestingly, when using other sender types (ExhaustiveSearchSender), the opposite result tends to appear. The next figure, 4, presents the scores for our defined Receiver agent and for the random receiver agent, showing clearly that EmbeddingNearestReceiver is preferred by the evaluation measure. The evaluation measure is the MAP measure defined in 4.1.3. Figure 5 compares GloVe and word2vec when using EmbeddingNearestReceiver. The plot doesn't show any clear difference between the model evaluations.

5.2.2 Human Sender Dataset

In figure 6 the different sender types are evaluated with the corresponding measure calculated on HumanSenderDataset. The same conclusion from the respective graph of the previous section holds. Figure 7 shows the same data as figure 6 in a different way, so that we can compare the embedding methods when using different agent types. ClusterSender shows a clear preference toward word2vec in this dataset as well. These results are in line with figure 3. Lastly, in figure 8 we can sanity-check that the random receiver model performs poorly in comparison to EmbeddingNearestReceiver, when using the evaluation measure defined in 4.2.3

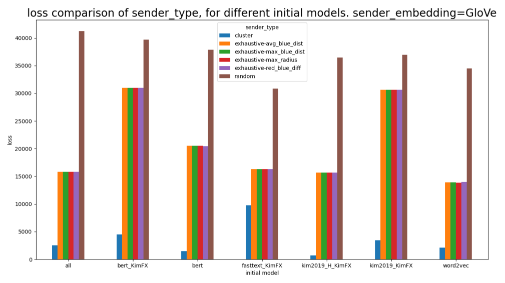


Figure 2: The loss of the different sender types, as a function of the measure's initial model, for constant GloVe embedding.

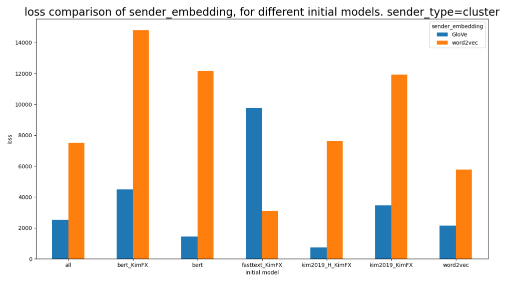


Figure 3: The loss of the different embeddings, as a function of the measure's initial model, for constant sender type cluster.

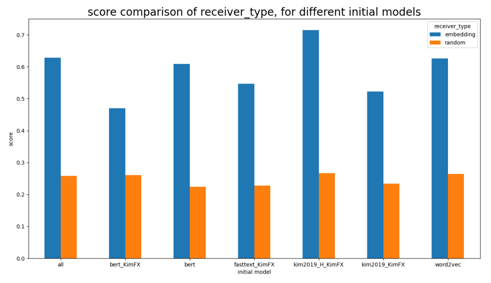


Figure 4: The scores of the different receiver types, as a function of the measure's initial model, for constant GloVe embedding.

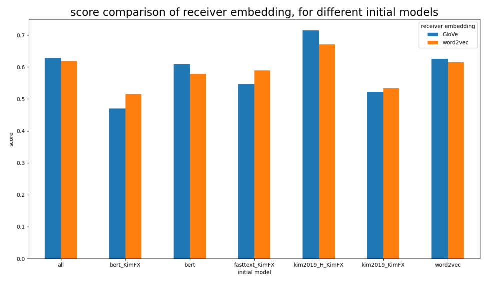


Figure 5: The scores of the different embeddings, as a function of the measure's initial model.

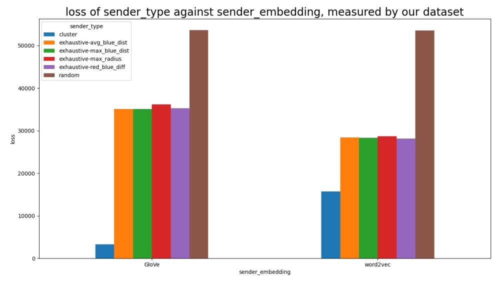


Figure 6: The loss of the different sender types, as a function of the embedding.

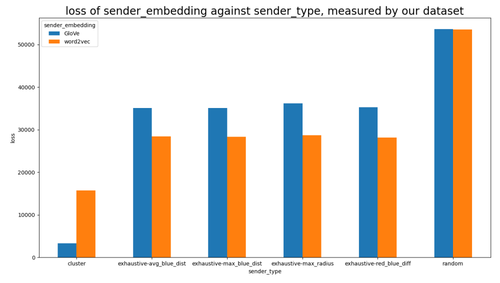


Figure 7: The loss of the different embeddings, as a function of the sender type.

6 Conclusion

The increasing use of word embedding has led to the demand for reliable embedding method evaluation measures. In this project, we suggested a family of novel, data-driven evaluation measures, that aim to provide an assessment of the performance of embeddings-based agents, relative to their ability to function in a human-based environment. The ability of the agent to cooperate with humans is measured in the framework of the famous game Codenames, which is suitable for the task since it requires multimodal understanding of natural language, and the ability to express it to other humans. We collect a new dataset and conduct experiments in which we apply the measures to our own defined embeddings-based agents, including a random baseline. The results show the measures reject the random agent, suggesting they are adequate for their task. In the future, we may expand this project and test more word embedding, while looking for re-weighting methods to further decrease the influence of the biased initial model on the measure's results.

Acknowledgments We would like to thank the faculty of industrial engineering at the Technion – Israel Institute of Technology for sponsoring the Prolific crowdsourcing experiment, as the final project in the course Language, Computation and Cognition (096222), instructed by Yevgeni Barzak. We would also like to thank Divya, and the authors of (Koyyalagunta et al., 2021), for kindly sharing the AMT data with us.

References

Prolific.

Wikipedia - codenames.

Nolan Bard, Jakob N. Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H. Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, Iain Dunning, Shibl Mourad, Hugo Larochelle, Marc G. Bellemare, and Michael Bowling. 2020. The hanabi challenge: A new frontier for ai research. *Artificial Intelligence*, 280:103216.

Andrew Kim, Maxim Ruzmaykin, Aaron Truong, and Adam Summerville. 2019. Cooperation and codenames: Understanding natural language processing via codenames. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 15(1):160–166.

Divya Koyyalagunta, Anna Sun, Rachel Lea Draelos, and Cynthia Rudin. 2021. Playing codenames with

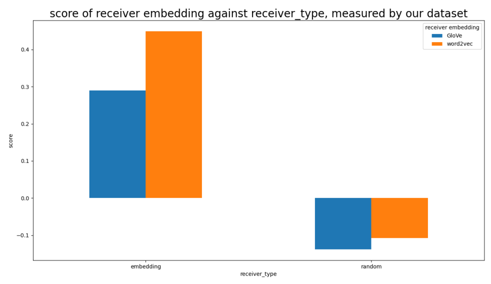


Figure 8: The score of the different embeddings as a function of the receiver type.

language graphs and word embeddings. *Journal of Artificial Intelligence Research*, 71:319–346.

Georg Ostrovski, Pablo Samuel Castro, and Will Dabney. 2021. The difficulty of passive learning in deep reinforcement learning.

Adith Swaminathan and Thorsten Joachims. 2015. Counterfactual risk minimization: Learning from logged bandit feedback. In *International Conference on Machine Learning*, pages 814–823. PMLR.