Towards an evolutionary-based approach for natural language processing

Luca Manzoni
Dipartimento di
Matematica e Geoscienze,
Università degli Studi di Trieste
Trieste, Italy

Domagoj Jakobovic
Faculty of Electrical Engineering and
Computing, University of Zagreb
Zagreb, Croatia

Luca Mariot
Cyber Security Research Group,
Delft University of Technology
Delft, The Netherlands

Stjepan Picek
Cyber Security Research Group,
Delft University of Technology
Delft, The Netherlands

Mauro Castelli
NOVA Information Management
School (NOVA IMS),
Universidade Nova de Lisboa
Lisbon, Portugal

This is the author accepted manuscript version of the article published by ACM as:

Manzoni, L., Jakobovic, D., Mariot, L., Picek, S., & Castelli, M. (2020). Towards an evolutionary-based approach for natural language processing. In GECCO 2020: Proceedings of the 2020 Genetic and Evolutionary Computation Conference (pp. 985–993). (GECCO 2020 – Proceedings of the 2020 Genetic and Evolutionary Computation Conference). Association for Computing Machinery. https://doi.org/10.1145/3377930.3390248

Towards an evolutionary-based approach for natural language processing

Luca Manzoni
Dipartimento di Matematica e Geoscienze, Università degli Studi di Trieste
Trieste, Italy

Domagoj Jakobovic
Faculty of Electrical Engineering and Computing, University of Zagreb
Zagreb, Croatia

Luca Mariot
Cyber Security Research Group, Delft University of Technology
Delft, The Netherlands

Stjepan Picek
Cyber Security Research Group, Delft University of Technology
Delft, The Netherlands

Mauro Castelli
NOVA Information Management School (NOVA IMS), Universidade Nova de Lisboa
Lisbon, Portugal

ABSTRACT

Tasks related to Natural Language Processing (NLP) have recently been the focus of a large research endeavor by the machine learning community. The increased interest in this area is mainly due to the success of deep learning methods. Genetic Programming (GP), however, was not under the spotlight with respect to NLP tasks. Here, we propose a first proof-of-concept that combines GP with the well established NLP tool word2vec for the next word prediction task. The main idea is that, once words have been moved into a vector space, traditional GP operators can successfully work on vectors, thus producing meaningful words as the output. To assess the suitability of this approach, we perform an experimental evaluation on a set of existing newspaper headlines. Individuals resulting from this (pre-)training phase can be employed as the initial population in other NLP tasks, like sentence generation, which will be the focus of future investigations, possibly employing adversarial co-evolutionary approaches.

CCS CONCEPTS

• **Computing methodologies** → **Natural language processing**; **Genetic programming**; *Bio-inspired approaches.*

KEYWORDS

Genetic Programming, Natural Language Processing, Next word prediction

1 INTRODUCTION

Natural language processing (NLP) is a branch of artificial intelligence that analyzes naturally occurring texts to achieve human-like language processing for different applications [10]. With the increasing popularity of deep learning, NLP is nowadays a hot research topic in the scientific community, and several contributions (mainly based on neural models) appeared in recent years [26]. Next word prediction is one of the most important tasks in this research field, and, given a sequence of words, it aims at predicting what word comes next. The importance of this task is motivated by the vast amount of applications in which it appears, including the composition of text and/or emails as well as ad-hoc applications designed to help people with physical disabilities to communicate [1]. The next word prediction problem can be addressed by considering different techniques [14]. The simplest approach is the one that takes into account words and their respective frequencies [13]. In this case, when a user entered the first characters of a word, the system suggests the most probable words beginning with the same character (or characters). While this method is simple to be implemented, it relies on standard word frequencies, thus ignoring the lexicon of different users. Thus, to obtain better suggestions, it is necessary to update the system once the user has entered a significant amount of text [25].

A different method to address the next word prediction problem takes into account the probability of appearance of each word after the one previously entered. In other words, instead of considering the simple information about words' frequencies, this approach uses a two-dimensional table containing the conditional probability of the word w_j appearing after the word w_i was entered. While this approach results in better prediction with respect to the one based only on words' frequencies, it has an important limitation associated with the size of the table that must be stored. Additionally, as pointed out in [14], it is difficult to adapt the system to the user's vocabulary when the dimensions of the table are fixed. Hence, this

approach is commonly used in combination with the previous one, and the table only stores the most probable word-pairs.

A different method considers the syntactic information inherent to natural languages to address the next word prediction task [24]. To implement this idea, we need two pieces of information: the first one is the probability of appearance of each word, while the second is the relative probability of appearance of every syntactic category after each syntactic category. By exploiting the syntactic information associated with the language, this approach results in better predictions with respect to the ones achieved by the aforementioned methods. The main limitation of this approach is the presence of words with ambiguous categories, that may significantly affect the quality of the prediction as well as the subsequent predictions [14]. A different approach is based on the analysis of sentences by the use of grammars and by applying NLP techniques to obtain the categories with the highest probability of appearance [15].

Nowadays, the most common and successful approaches to deal with NLP tasks are based on the use of deep learning techniques. This is mostly due to the ability of deep learning in modeling the recursivity of human language that is composed of words and sentences with a certain structure. Deep learning (especially recurrent neural models), can capture the sequence information more effectively compared to other existing techniques. The reader is referred to [26] and [2] for a complete review of the existing deep learning-based approaches in the field of NLP.

Despite the existence of different techniques dedicated to the next word prediction task, the use of evolutionary algorithms was not fully explored. In particular, Genetic Programming (GP) was applied to different tasks in the context of NLP [5], but no specific effort was dedicated to this prediction task.

Considering the ability of GP to address problems over different domains and its ability to explore search spaces characterized by a high-dimensionality, in this paper, we answer this call by presenting a GP-based system for the next word prediction problem. The choice of using GP is also motivated by the fact that it can provide information that cannot be extracted from a black-box model, thus providing a more interpretable model. The ability of GP to provide human-understandable solutions is particularly useful in the context of this application because it may provide interesting hints about the nature of the language and the learning process of GP. Additionally, this task can be addressed by building rules from the set of words in input, and GP was successfully used to generate rules-based models [6] for problems over different domains. Thus it is reasonable to investigate the potential of GP in addressing the next word prediction task.

This paper is organized as follows: related works are presented in Section 2, while the proposed method for next word prediction with GP is described in Section 3. The experimental settings and results are presented in Section 4 and discussed in Section 5. Section 6 concludes the paper and highlight some possible directions for future research.

2 RELATED WORKS

This section reviews the use of Genetic Programming for addressing different tasks in the field of natural language processing. While this research field is nowadays dominated by the use of deep learning models, we decided to restrict our attention to the GP-based techniques that represent the method explored in this paper.

The first applications of GP in the field of NLP were mostly related to the identification of the semantic structure of the language. The main reason relies on the fact that the syntax-tree representation used in GP is similar to the one commonly employed to describe the syntax of a language [5]. Thus, it comes naturally to use GP for the task of identifying the syntactic structures of the language. One of the first studies in this area was proposed in [22], where the author employed GP to address a parser failure problem in speech-to-speech machine translation. In the same research direction, Araujo [3] proposed a GP-based system for natural language parsing. The system was subsequently improved by considering a multi-objective formulation for simultaneous performance of language tagging and parsing [4].

Other works where GP was used for NLP tasks took into consideration the generation of regular expressions. Regular expressions are important in the field of NLP due to their ability to represent string patterns precisely. In [7], Bartoli and coauthors applied GP for extracting regular expressions for entity extraction applications. They designed a multi-objective GP-based system to automatically create a regular expression for a task that was represented through a set of examples. Experimental results demonstrated the suitability of the system they proposed, with the GP-based approach outperforming the existing techniques already tested on the same datasets. Subsequently, in [8], Bartoli and coauthors described an active learning approach where the user acts as an oracle. They assessed the performance of this approach on the same datasets used in [7], and they demonstrated that the active learning approach might reduce the computational cost by requiring a lower amount of user annotations.

Another task where GP was employed is the search and extraction of semantic relationships in texts, which is especially relevant in the medical domain. In this domain, GP was used to identify sentences that contain descriptions of interactions between genes and proteins [9]. More in detail, GP was employed to obtain a model of syntax patterns composed of part-of-speech tags. The model consisted of a set of automatically learned regular expressions, following the idea developed in [7].

Other works focused on the entity linking task, the identification of the different ways in which the same entity is mentioned in the texts. To address this task, Carvalho and coauthors [11] used GP to find effective functions that can identify, in a data repository, entries referring to the same entity despite typos. Experimental results demonstrated that GP was able to outperform the state-of-the-art SVM-based approach over real-world data. Subsequently, a combination of GP and active learning was proposed in [16] to address the same problem, and the results demonstrated the beneficial effect of combining active learning with GP.

Moving to the task of natural language generation, Manurung and coauthors [19] developed a GP system for generating poetry with a certain meter or patterns in the rhythm, while Kim and coauthors [17] developed a GP system for generating answers that an agent must provide to users' queries. These are the only GP-based contributions in the field of natural language generation, and

they cover very specific domains. Additionally, in these works, GP is used to represent the grammars that must be evolved.

For a complete review of the applications of GP in the NLP field, the reader is referred to [5].

3 NEXT WORD PREDICTION WITH GP

In this section, we describe how to adapt GP to work with words both at the input and the output levels and how to manipulate them. In particular, we tackle the problem of *next word prediction*, where, given an initial sequence of words, possibly of fixed length k, we want to predict the next word in the sequence. Since we need to produce a word, we have to perform text generation via GP. To accomplish that, we need to tackle multiple obstacles; specifically, there are the following points to be taken care of:

- **Input representation**. How can the input words be represented in a suitable way for GP?
- **Functional operators**. What operations can be performed on the representation of the words?
- **Output interpretation**. How can we decode the output of a GP individual and interpret it as a word?

These three questions must be addressed, and are all related to the same problem of deciding in which genotypic space GP has to work. In particular, the input representation will influence which kind of operations one can perform, and the operations will also dictate how the output will be decoded.

Behind our approach to all three problems lies the adoption of an *embedding*, where words are embedded into a vector space, usually $\mathbb{R}^d$ for some dimension d. Among the existing embedding methods, one that has risen to prominence since its inception in 2013 is *word2vec* [20]. The algorithm takes as input a large corpus of text and produces an embedding where each word of the corpus is represented as a real-valued vector, with the property that words sharing similar contexts will be near in their vector representation. Usually, the size of the space $\mathbb{R}^d$ in which the words are embedded is between 100 and 1 000, but we will see that, in the case of GP, a lower dimensionality appears to help. Notice that we assume that all necessary words, both for the input and the output, are present in the original corpus used to train the word2vec model. That is, the GP trees only receive inputs and produce the output only from a fixed vocabulary.

We are now going to detail our approach to solve all three previously highlighted obstacles. A graphical representation of the entire process is given in Figure 1.

Input representation. Given a sentence of fixed length k consisting of the words $w_1, \ldots, w_k$, we map each word to its corresponding vector via a word2vec embedding to interpret the sentence as an input to a GP individual. Therefore, we now have k vectors $\vec{w}_1, \ldots, \vec{w}_k \in \mathbb{R}^d$, where d is the dimension of the embedding. Without loss of generality, we consider all vectors to be of unitary length, that is $||\vec{w}_i||_2 = 1$ for all $1 \leq i \leq k$.

Functional operators. Once the input encoding as unitary vectors in $\mathbb{R}^d$ has been defined, it is necessary to define the set of functional symbols to employ. A first requirement is that the output of each operation should itself be a unitary vector in $\mathbb{R}^d$. If we consider some of the common binary GP operators as used in symbolic regression, i.e., $+$, $-$, $\times$, and protected division ($\div$), they can all be extended to work on vectors. Given two vectors $\vec{u}$ and $\vec{v}$ in $\mathbb{R}^d$, a binary operator $\circ\colon \mathbb{R}\times\mathbb{R} \to \mathbb{R}$ can be extended to a binary operation $\vec{\circ}\colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ by applying it component-wise as follows:

$$\left(\vec{u}\ \vec{\circ}\ \vec{v}\right)_i = \vec{u}_i \circ \vec{v}_i \qquad\qquad \forall 1 \leq i \leq n \ .$$

We can further restrict the output vector to be of unitary length by normalizing it, as long as its length is not null. This allows us to define the operator $\vec{\circ}_1\colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ as:

$$\left(\vec{u}\ \vec{\circ}_1\ \vec{v}\right) = \frac{\left(\vec{u}\ \vec{\circ}\ \vec{v}\right)_i}{||\vec{u}\ \vec{\circ}\ \vec{v}||_2} \qquad\qquad \forall 1 \leq i \leq n \ .$$

The same procedure can then be extended to unary operators, such as squaring, or even to operators of higher arity, by component-wise evaluation followed by normalization.

By performing this extension process on the classical GP operators, we are then able to produce unitary vectors in $\mathbb{R}^d$ as output whenever vectors in $\mathbb{R}^d$ are given as input.

Output interpretation. To obtain a word as the final output of our tree, one has to return from the embedding produced by word2vec to the actual set of words. Let us consider the vocabulary V consisting of the m distinct words $w_1, \ldots, w_m$ occurring in the corpus used to generate the embedding, and their respective vectors $\vec{V} = \{\vec{w}_1, \ldots, \vec{w}_m\}$. The output of a GP tree is a vector $\vec{v} \in \mathbb{R}^d$, which might not be an element of $\vec{V}$, and, in practice, it will almost never be. Therefore, we assign to $\vec{v}$ the word corresponding to the vector $\vec{u} \in \{\vec{w}_1 \ldots \vec{w}_m\}$ which is most similar to $\vec{v}$. In the case of word embeddings, a common measure of similarity is the *cosine similarity*, corresponding to the cosine of the angle formed between the two vectors $\vec{u}$ and $\vec{v}$, which is defined as:

$$\mathrm{sim}(\vec{u}, \vec{v}) = \frac{\sum_{i=1}^{n} \vec{u}_i \vec{v}_i}{||\vec{u}||_2 ||\vec{v}||_2} \ .$$

Since cosine is between -1 and 1 and maximal when the two vectors coincide, we will need to select, among the words in the vocabulary, the one whose embedding has the highest similarity to $\vec{v}$.

Computation of the fitness. Given a set of h fitness cases, each one of the form $((w_1, \ldots, w_k), w_{k+1})$, where $w_1, \ldots, w_k \in V$ are the input words and $w_{k+1} \in V$ is the target output, the fitness of a GP individual T is computed as the average of the cosine similarity between the output of T and $\vec{w}_{k+1}$. Since we are considering cosine similarity, the fitness is to be maximized.

There is, however, a possible ambiguity in this definition: one can consider the output of T either as the vector $\vec{v}$ obtained *before* re-interpreting the output as a word, or the word w_i from the vocabulary obtained *after* "decoding" the vector $\vec{v}$. The two approaches might produce different results, but the second one, while arguably more "precise" (i.e., by considering the output of the entire system and not only of T), is also more computationally intensive, since a naïve implementation requires the computation of the cosine similarity with all words in the vocabulary for *each* fitness case. Therefore, we have selected the first method to compute the fitness. That is, for each fitness case we compute the output $\vec{v}$ of T and $\mathrm{sim}(\vec{v}, \vec{w}_{k+1})$. The sum of these values across all fitness cases is then averaged to produce the final fitness value of T.

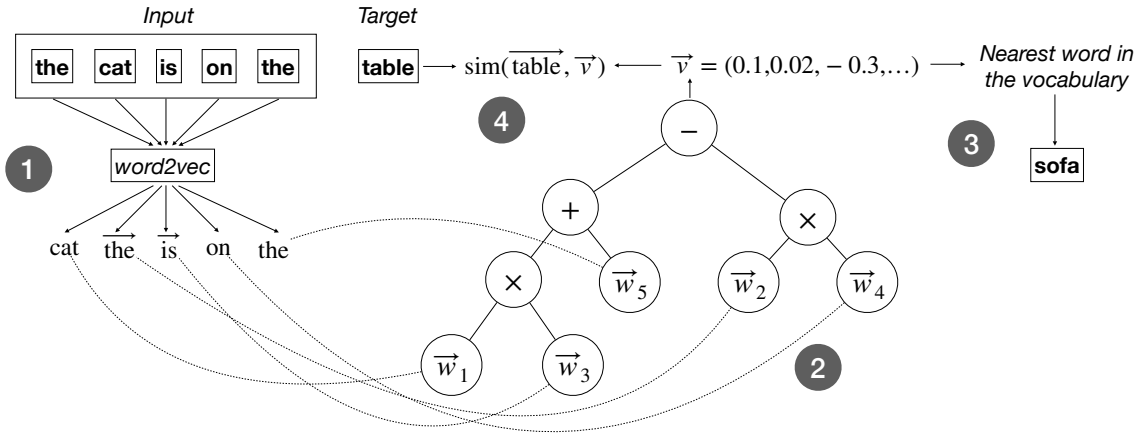

Figure 1: A graphical depiction of the process employed: conversion of the input word into vectors (1), evaluation of the GP tree (2), and conversion of the output vector into a word by finding the most similar word in the vocabulary (3). The fitness can be computed by computing the similarity between the target and output of the GP tree (4) directly.

4 EXPERIMENTS

In this section, we describe the experimental setting adopted to test our GP algorithm, along with a brief description of the text dataset used to train the GP trees for the next word prediction task. We then report and discuss the results obtained during the training and the test phase.

4.1 Experimental Setting

Recall that the optimization goal is to evolve a population of GP trees that take as input the first words of a sentence and predict the next word to complete the sentence. For our experiments, we considered the *Million News Headlines* (MNH) dataset [18], which contains headlines collected from the Australian Broadcasting Corporation over 17 years from 2003 to 2019. In particular, for our experiments, we fixed the problem instance to headlines of six words, which amounts to 267 292 training examples in the MNH dataset. Therefore, the prediction task of the trees evolved through GP was to generate the vector for the sixth word of these headlines by considering as input the first five words of the headline.

For the pre-training phase, we used word2vec to generate six different embeddings $E_d \subseteq \mathbb{R}^d$ of the whole MNH dataset, with dimension d ranging in the set $\{10, 15, 20, 25, 50, 100\}$. This was done to investigate whether the dimension of the embedding space influenced the training performance of GP. For the embedding, we employed the default parameters of word2vec, except for the size (which was a value in $\{0, 15, 20, 25, 50, 100\}$), and the `min_count` which was set to 1, in order to insert all words in the vocabulary. Notice that this means that very infrequent words will not have a semantically "good" representation as vectors. Finally, the number of iterations was increased to 20, and the number of threads to 16.

The input variables at the terminal nodes of the trees evolved by GP represented the vectors corresponding to the first words of a sentence under the considered embedding. The set of functionals for the internal nodes of the trees included sum, subtraction, multiplication, and protected division as binary operators, and squaring and square root as unary operators. Recall that all operators are pointwise evaluated, i.e., they are applied separately on each coordinate of the input vectors. We adopted a steady-state breeding strategy using a tournament selection operator of size $t = 3$, where the two candidates with the highest fitness are mated through crossover, and the resulting child tree replaces the worst individual in the tournament after being mutated. For the variation operators, we employed simple subtree crossover, uniform crossover, size fair, one-point, and context preserving crossover, randomly selected at each generation, while for mutation, we used simple subtree mutation [21]. To prevent bloat, we forced a maximum depth on our GP trees equal to 5, i.e., the number of input words. Further, we considered a population of $P = 500$ individuals and a mutation probability of $p_\mu = 0.3$, and we used a maximum budget of $fit = 100\,000$ fitness evaluations as a termination criterion. All these parameters were selected after a preliminary tuning phase, where we observed that perturbing the above values did not yield any significant difference in performances. In particular, we detected no substantial increase in the fitness value of the best solution after the 100 000 evaluations cap. Finally, each experiment was repeated for $R = 30$ independent runs, each time using a different training set. The training set for each experimental run was determined by sampling one-hundredth of a random shuffle of the 267 292 headlines with six words. Thus, each training set was composed of $T = 2\,672$ sentences.

4.2 Training Phase Results

The first question that we addressed in our training experiments is whether the trees evolved by GP actually learned a model of the MNH dataset, under the embeddings produced by word2vec. To this end, for each embedding dimension, we compared the following distributions approximated by our $R = 30$ experimental runs:

- The fitness value of the best GP individual in the population before the first generation takes place (i.e., after randomly initializing the population) and at the end of the experimental run (Figure 2).

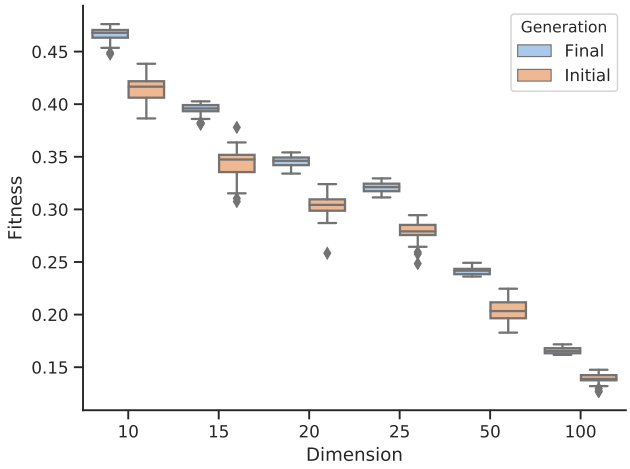

Figure 2: Best GP fitness at the first and last generation.

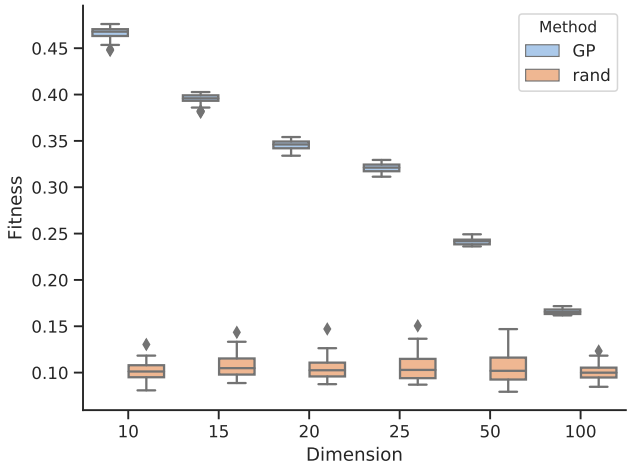

Figure 3: Best GP and random generation fitness values.

- The fitness value of the best GP individual at the end of the experimental run and the fitness value of the best random predictor (Figure 3).

In particular, the best random predictor was determined as follows. Given a training set of $T = 2\,672$ six-words headlines, for each embedding dimension $d \in \{10, 15, 20, 25, 50, 100\}$ we generated $P = 500$ sets of T random vectors, and used them to predict the sixth word of the headlines. Stated differently, we employed these vectors as random surrogates for the outputs of the GP individuals over the training set. Subsequently, we scored each of the P sets of vectors by applying the same fitness function used for GP, and the set achieving the highest fitness was selected as the best random predictor. We repeated this experiment for $R = 30$ times, using the same training sets employed for the GP experimental runs. Remark that the best random predictors just described are different from the best individuals obtained by randomly initializing the GP population: indeed, the latter are GP trees that, although having a random structure, read the first five words of a sentence to predict the next one, while the former are random vectors completely independent from the content of the sentences.

The boxplots of Figures 2 and 3 clearly confirm that the trees evolved by GP are learning a model of the training sets considered in the experimental runs. As a matter of fact, for all considered dimensions, the fitness of the final best individual is always higher than that of both the best individual after initialization and the best random predictor. More specifically, the difference between the final best individual and the random predictor is much more significant, as Figure 3 shows.

The second question that we investigated was whether a correlation exists between the dimension d of the embedding space produced by word2vec and the fitness of the final best individual bred by GP. In particular, one can see from Figure 3 that the lower the dimension of the embedding, the higher is the fitness attained by the best individual in the population. Although this finding is in stark contrast with the common NLP practice based on deep learning models, where the involved word2vec embeddings usually have hundreds of dimensions, this is a reasonable outcome for our GP algorithm. In fact, one can expect that a GP tree has an easier time in overfitting a training set of a couple of thousands of samples over a lower-dimensional space rather than on a higher-dimensional one since we enforce a maximum depth of the tree equal to the number of input words. Considering the relatively small size of our text corpus, it could be the case that the word2vec embeddings with lower dimensions produce vectors that are more closely packed, thus making overfitting easy for a compact GP tree. On the other hand, for the embeddings of higher dimension, the resulting vectors could be sparser, and thus harder to handle for small trees.

At first glance, this overfitting hypothesis seems to indicate that the GP trees evolved over lower dimensional embeddings fare worse on the test set. For this reason, we performed a preliminary validation test on a small random sample of 50 headlines of six words. Surprisingly, we remarked that the best GP trees evolved over the higher dimensional embeddings (i.e., $d = 50, 100$) resulted in a completely uninteresting behavior, since they learned to predict only one of the first five words seen in input to complete the sentence, almost always the first one. On the contrary, although the GP trees trained with the embeddings of dimension 10, 15, 20, and 25 failed to predict the original target words in the sample, nevertheless, they completed the sentences more creatively, producing plausible headlines in some cases. To further corroborate this observation, we compared the distribution of the best individuals evolved by GP with the trivial generators that always predict the first and the fifth word of a sentence, using the same training sets adopted for GP. For each considered dimension, Figures 4 and 5 report the boxplots of the fitness values attained by the best GP trees and the "first-word" and "last-word" predictors, respectively. One can notice from Figure 4, that the boxplots for the “first-word” predictor and the best GP individual obtained over dimension $d = 100$ are quite close to each other, thereby providing further evidence that this is the behavior learned by those GP trees over the higher

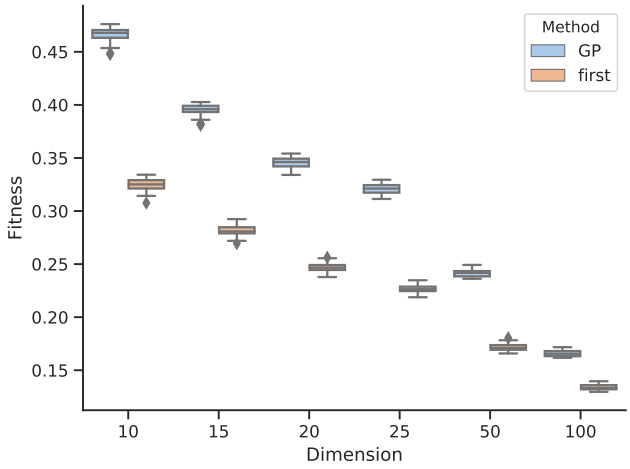

Figure 4: Best GP and "first-predictor" fitness values.

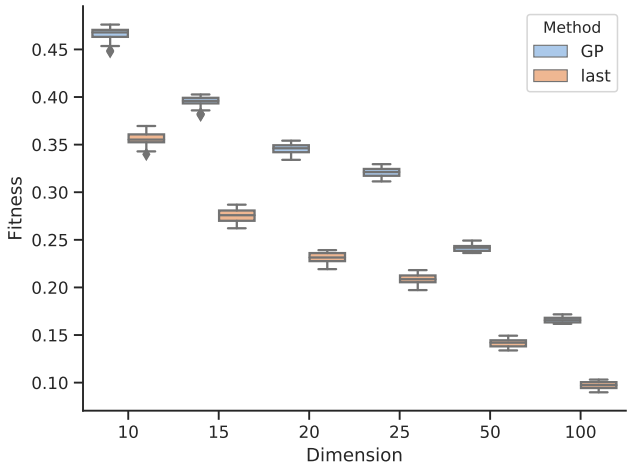

Figure 5: Best GP and "last-predictor" fitness values.

dimensional embeddings. On the contrary, there is a clear difference between the distribution of the best GP individuals and the first-word predictor for lower dimensions, showing that the fitness of the former is much higher than that scored by the latter. Hence, this seems to indicate that allowing to overfit the training data over a lower-dimensional embedding is beneficial to evolve GP trees that predict meaningful completions.

A final remark that can be drawn from the previous plots is that the distributions of the best fitness values obtained by GP at the final generation are not widely dispersed. Indeed, especially for high dimensions, the first and third quartiles are quite close to the median best fitness. The only remarkable difference is in the lower dimensional embeddings, where there is a wider difference between the minimum and the maximum best fitness. Nonetheless,

d	Size	Expression
10	27	$(((w_2 + (w_4 + w_0)) + ((w_4 + w_1) + w_2^2)) + ((w_3 + w_0^2) - ((\sqrt{w_2} - (w_1 \cdot w_4)) \cdot \sqrt{w_2})))$
15	38	$(((((w_4 + w_0) + (w_3 + w_2)) + (w_1 + (w_2 \cdot w_2))) + (w_3 + (w_4 + w_4))) + ((((w_0 + w_0) + w_2) - (w_4 \cdot (w_3 - w_4))) + (w_1 + (\sqrt{w_3} \cdot w_3))))$
20	39	$(((((w_3 + w_0) + w_0^2) + (w_1 + w_1)) + (w_1 + w_4)) + ((((w_4 + w_0) + (w_2 + w_0)) + (w_4 + w_2)) + (((w_3 + w_0) + w_2^2) + (w_4 + (w_4 \cdot w_4)))))$
25	48	$(((((w_4 \cdot w_4) + (w_4 + w_0)) + ((w_1 + w_3) + (w_4 + w_2))) + ((w_3^2 + w_1) + (w_0 + (w_0 + w_4)))) + ((((w_1 + w_3) + (w_4 + w_0)) + (w_2 + w_0)) + (((w_2 - w_1) \cdot (w_2 - w_1)) + w_2)))$
50	36	$((((w_4 + (w_3 + w_0)) + (w_1 - w_3^2)) + ((w_4 + (w_0 + w_1)) + w_2)) + ((w_2 + (w_4 + w_0)) + ((w_3 + w_0) + ((w_2 - w_3) \cdot (w_1 + w_3)))))$
100	27	$((((w_0 + (w_3 \cdot w_3)) + w_1^2) + (w_2 + w_1)) + ((((w_4 + w_0) + w_3) + w_1^2) - (w_4 \cdot (w_1 - w_4))))$

Table 1: Algebraic expressions and sizes of the best GP individuals tested for each embedding dimension.

this seems to indicate that the best GP trees evolved over a specific embedding learn the same prediction model. This hypothesis is also supported by our preliminary validation test, since for each dimension d the best individuals obtained over the $R = 30$ runs completed the 50 headlines approximately in the same way, almost always by predicting the same word for each sentence.

4.3 Testing Phase Results

To better investigate the prediction models learned by the trees evolved with GP over the training set, we performed a more systematic testing phase designed as follows. For each of the considered six embedding dimensions, we selected the best GP tree with the highest fitness among the 30 experimental runs. Table 1 reports the algebraic expressions of the best-selected individuals together with their sizes, while Figure 6 depicts the tree of the best-selected individual for dimension $d = 10$ as an example. Notice that in Table 1, we displayed the non-simplified expression of the trees since the output of each internal node undergoes normalization, which would have burdened the notation if we included it in the simplified formulae. Next, we tested each selected individual over a random test sample of 10 000 headlines of six words from the MNH dataset. Analogously to the preliminary validation step described in the previous section, we evaluated the GP tree over the embedding vectors corresponding to the first five words for each sentence in the test set, thus obtaining a sixth vector in output. Since this output vector does not correspond in general to a word of our corpus under the considered word2vec embedding, the predicted word is the one whose embedding vector has the highest cosine similarity with the output vector. Finally, for each sentence, we computed the cosine similarity between the predicted word and the original sixth word. Cosine similarity of 1 thus means that the GP tree predicts

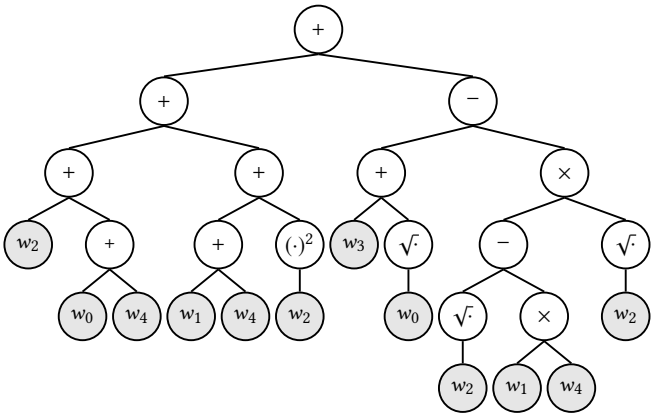

Figure 6: GP tree of the best individual evolved for dimension $d = 10$ during one of the runs.

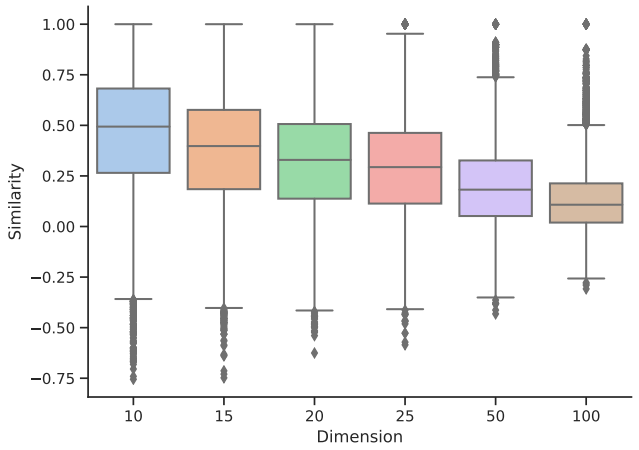

Figure 7: Distributions of similarity between predicted and original word over the test set.

exactly the original sixth word of a headline. However, remark that predicting the original word is not the main task of the models encoded by the GP trees since each headline can be completed in many ways that are both syntactically and semantically valid. In general, we observed that high values of cosine similarity usually correspond to meaningful completions of the headlines.

Figure 7 reports the boxplots of the distributions of similarity between predicted and original word over the test sample of 10 000 headlines. One can see from the plots that the initial findings suggested by the training phase results are confirmed. Indeed, the cosine similarity between predicted and original words obtained by the GP trees trained over higher-dimensional embeddings is lower than the similarity achieved by trees that were evolved over the lower-dimensional embeddings. In particular, it can be remarked that the outliers are arranged rather symmetrically: while for lower dimensions they are all focused below the bottom whisker, for dimension $d = 50$ and $d = 100$ they are almost all above the top one. On the other hand, the GP trees trained over dimension up to 20 consistently reach higher values of cosine similarity.

As we remarked in the previous section, the GP individuals predicting a final word having a high cosine similarity with the original word often resulted in meaningful headlines, when considering a small test set of 50 sentences, and we observed the same behavior also over the larger test set of 10 000 sentences. Table 2 reports a small selection of headlines completed by the best GP individual of dimension 10 on the test set, together with the original word and the associated cosine similarity. In particular, we chose among predictions that reached a cosine similarity of at least 0.8. It can be seen from the table that in most cases the meaning of the original sentence is completely changed by the prediction. Indeed, only for three sentences the predicted word is a synonym of the original one (see *houses–homes*, *robberies–heist* and *funds–funding*). However, all resulting headlines are plausible, and one can observe that the predicted word is always related to a similar context with respect to the original word. For example, in the last example of Table 2 the predicted word refers to a generic place (*towns*), while the original final word is the name of a region (*Victoria*). Moreover, the third and the eight examples suggest that the model learned by the GP tree is also able to discriminate between different contexts when the same word occurs in the input. In fact, in the third sentence, the word *tax* has been matched with *changes* instead of *review*, hence preserving the political overtone of the headline. Contrarily, in the eight sentence the GP tree paired *tax* with *bureaucracy* in place of *rates*, thus remaining on a more administrative/economic context.

5 DISCUSSION

The results presented in the previous two sections suggest that, to some extent, GP can learn a language model from a text dataset by leveraging the embedding produced by word2vec. We now summarize our main findings, framing them in a critical perspective to address the advantages and the shortcomings of the approach proposed in this paper.

Learning vs. Exact Prediction. We have observed that GP does not usually predict the exact missing word in a sentence, i.e., an *exact prediction*. However, we do not consider this a significant problem. In fact, there are multiple reasonable ways to complete sentences,

Predicted headline	Original
Regional education to fund youth *preschool*	allowance
Aerial footage of flooded Townsville *houses*	homes
Greens renew call for tax *changes*	review
Napthine to launch new Portland *rail*	marina
4 charged over 10000 jewellery *robberies*	heist
Vanstone defends land rights act *overhaul*	changes
Community urged to seek infrastructure *funds*	funding
Govt. pressured on company tax *bureaucracy*	rates
Petition urges probe into abattoir *maintenance*	closure
Rain does little for central *towns*	Victoria

Table 2: Examples of test headlines completed by the best GP individual of dimension 10.

and a dataset might itself contain multiple endings for the same sentence. What we are more interested in is the ability of GP to produce a *meaningful* completion of a sentence. This ability is, in some sense, more interesting. It shows that GP has learned to "navigate" the space given by the word2vec embedding and produce results that align with the semantics of the sentences.

Dimensionality and Fitness. One interesting observation from the experiments is that the dimensionality of the embedding has a large influence on the behavior of GP. First of all, the increase in the fitness values can be partially dictated purely by geometrical effects; in fact, the average fitness for the trivial predictors (that predict only the first or last word of a sentence) also increases. However, the increase in fitness obtained by GP is higher, showing that lower-dimensional spaces might be easier to manipulate by this kind of GP. Furthermore, the size of the embeddings used in this work is quite small compared to the ones usually employed, which can have up to hundreds of dimensions. The motivation for GP preferring lower-dimensional spaces is an interesting topic for future investigations.

6 CONCLUSIONS AND FUTURE WORKS

In this paper, we introduced a new approach to perform an NLP task, namely next word prediction, using GP combined with the word embeddings produced by word2vec. This combination required a way to manipulate vectors via GP (both as inputs and as outputs), to subsequently be able to "go back" from the embedding and generate words via GP. We have tested our approach on The Million News Headlines dataset. To check that the GP trees were not learning only trivial functions of the input, we have compared their fitness with random and trivial predictors, showing that GP is actually learning a model of the language represented by the dataset. By looking at how unseen sentences were completed, we found that, for dimension 10, a median cosine similarity around 0.5 was obtained, and, among the word generated, reasonable completions were usually obtained. While the results of this work do not top the current state of the art, our approach shows that GP is a promising technique for this kind of task, opening many research directions and improvements that, hopefully, will bring GP among the top techniques in NLP. In fact, we are only moving *toward* a possible application of GP to NLP tasks. Here, we mention some possible directions for future research; the list is by no means exhaustive, but it should provide an ample spectrum of possible improvements.

We have only employed operators that are pointwise adaptations of the classical GP operators. An immediate improvement would be to add operators that are *specific* for arrays of unitary length, such as *rotations* and other linear transformations.

The GP trees employed are deterministic; if we have two sentences that agree everywhere except for the word to be predicted, it is impossible to generate them in a stochastic way. Therefore, an immediate extension is to make the generation *probabilistic*. There are multiple ways to accomplish this; among them, one possibility is to use an entire set of trees for constructing the word and picking a single tree randomly each time a word needs to be generated.

We also tested our approach only on a single dataset of headlines, and we also restricted the training to a subset of it. It is essential to extend the approach to a larger number of datasets; this will require multiple improvements in terms of scalability of the GP algorithm, and possibly in terms of the management of low-frequency words, which could also be treated as "unknown" tokens.

While next word prediction has been the focus of this work, our approach can be extended to generate longer texts by using a "sliding window" methods: given the words $w_1, \ldots, w_k$, the word w_{k+1} is predicted as usual; then, the words $w_2, \ldots, w_{k+1}$ are used to generate w_{k+2}, and so on. Clearly, such an approach does not keep any internal state in the GP tree, and it leverages only on the last k words. Therefore, it would be interesting to add a "memory" feature to GP, similarly to LSTM networks [12, 23].

Finally, to evaluate the quality of the generated words or sentences, a competitive (or "adversarial") co-evolution could be employed, where there is a population of trees that act as the generators of sentences, while another population contains trees that act as discriminators between real and generated sentences.

REFERENCES

[1] Hisham Al-Mubaid and Ping Chen. 2008. Application of word prediction and disambiguation to improve text entry for people with physical disabilities (assistive technology). *International Journal of Social and Humanistic Computing* 1, 1 (2008), 10–27.

[2] Basemah Alshemali and Jugal Kalita. 2019. Improving the Reliability of Deep Neural Networks in NLP: A Review. *Knowledge-Based Systems* (2019), 105210.

[3] Lourdes Araujo. 2004. Genetic programming for natural language parsing. In *European Conference on Genetic Programming*. Springer, 230–239.

[4] L. Araujo. 2006. Multiobjective Genetic Programming for Natural Language Parsing and Tagging. In *Parallel Problem Solving from Nature - PPSN IX*, Thomas Philip Runarsson, Hans-Georg Beyer, Edmund Burke, Juan J. Merelo-Guervós, L. Darrell Whitley, and Xin Yao (Eds.). Springer, 433–442.

[5] Lourdes Araujo. 2019. Genetic programming for natural language processing. *Genetic Programming and Evolvable Machines* (2019), 1–22.

[6] Wolfgang Banzhaf, Randal S Olson, William Tozier, and Rick Riolo. 2019. *Genetic Programming Theory and Practice XVI*. Springer.

[7] Alberto Bartoli, Giorgio Davanzo, Andrea De Lorenzo, Eric Medvet, and Enrico Sorio. 2014. Automatic synthesis of regular expressions from examples. *Computer* 47, 12 (2014), 72–80.

[8] Alberto Bartoli, Andrea De Lorenzo, Eric Medvet, and Fabiano Tarlao. 2017. Active learning of regular expressions for entity extraction. *IEEE transactions on cybernetics* 48, 3 (2017), 1067–1080.

[9] Alberto Bartoli, Andrea De Lorenzo, Eric Medvet, Fabiano Tarlao, and Marco Virgolin. 2015. Evolutionary learning of syntax patterns for genic interaction extraction. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. 1183–1190.

[10] Robert Dale, Hermann Moisl, and Harold Somers. 2000. *Handbook of natural language processing*. CRC Press.

[11] Moises G De Carvalho, Alberto HF Laender, Marcos André Gonçalves, and Altigran S da Silva. 2010. A genetic programming approach to record deduplication. *IEEE Transactions on Knowledge and Data Engineering* 24, 3 (2010), 399–412.

[12] Marc Ebner. 2017. Distributed storage and recall of sentences. *Bio-Algorithms and Med-Systems* 13, 2 (2017), 89–101.

[13] Nestor Garay and J Abascal. 1994. Using statistical and syntactic information in word prediction for input speed enhancement. *Information Systems Design and Hypermedia* (1994), 223–230.

[14] Nestor Garay-Vitoria and Julio Abascal. 2006. Text prediction systems: a survey. *Universal Access in the Information Society* 4, 3 (2006), 188–203.

[15] Nestor Garay-Vitoria and Julio Gonzalez-Abascal. 1997. Intelligent word-prediction to enhance text input rate (a syntactic analysis-based word-prediction aid for people with severe motor and speech disability). In *Proceedings of the 2nd international conference on Intelligent user interfaces*. 241–244.

[16] Robert Isele and Christian Bizer. 2013. Active learning of expressive linkage rules using genetic programming. *Journal of web semantics* 23 (2013), 2–15.

[17] Kyoung-Min Kim, Sung-Soo Lim, and Sung-Bae Cho. 2004. User adaptive answers generation for conversational agent using genetic programming. In *International Conference on Intelligent Data Engineering and Automated Learning*. Springer, 813–819.

[18] Rohit Kulkarni. 2018. A Million News Headlines. https://doi.org/10.7910/DVN/SYBGZL Accessed on February 1st, 2020.

[19] Ruli Manurung, Graeme Ritchie, and Henry S Thompson. 2008. An implementation of a flexible author-reviewer model of generation using genetic algorithms. In *Proceedings of the 22nd Pacific Asia Conference on Language, Information and*

Computation. 272–281.

[20] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *ICLR (Workshop Poster)*.

[21] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. 2008. *A Field Guide to Genetic Programming*. lulu.com.

[22] Carolyn Penstein Rosé. 1999. A genetic programming approach for robust language interpretation. *Advances in genetic programming* 3 (1999), 67–88.

[23] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. 2012. LSTM neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*.

[24] Andy L Swiffin, John L Arnott, and Alan F Newell. 1987. The use of syntax in a predictive communication aid for the physically handicapped. In *10th Annual Conference on Rehabilitation Technology*. RESNA-Association for the Advancement of Rehabilitation Technology, 124–126.

[25] Horabail Venkatagiri. 1993. Efficiency of lexical prediction as a communication acceleration technique. *Augmentative and Alternative Communication* 9, 3 (1993), 161–167.

[26] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. 2018. Recent trends in deep learning based natural language processing. *IEEE Computational intelligence magazine* 13, 3 (2018), 55–75.