A Proposed Algorithmic approach for Natural Language Generation using Genetic Evolutionary Algorithm

By
VENALI SONONE AND MOHIT GURNANI
VEERMATA JIJABAI TECHNOLOGICAL INSTITUTE

Abstract

The advancement in the field of artificial intelligence and machine learning leading to the human interaction with technology has formed the need for the machines to be able to generate language enabling the experience feel more natural rather work only on understanding natural language that human have uttered. To address this demand for natural looking machine generated text, the discipline of Natural Language Generation (NLG) was born. The ability to generate a text similar to particular writing skill of an author is interesting aspect to generate literature. This paper proposes algorithm using genetic algorithm. A genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA) which makes it persuasive to be used in Natural Language Generation. Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as mutation, crossover and selection.

Keywords: Natural Language Generation, Evolutionary algorithms, genetic algorithms, genetic operators, Word2vec word embedding.

I. Introduction

Natural Language Generation (NLG) is defined as the systematic approach for producing human understandable natural language text based on non-textual data or from meaning representations. This is a significant area which empowers human-computer interaction. Content generation systems assist human writers and makes writing process more efficient and effective. An approach can be considered as a high-level structure of human-authored text is used to automatically build a template for a new topic of a new literature content which is similar to a particular author's content.

Linguistic style is an integral aspect of natural language communication. It conveys the social context in which communication occurs and defines particular ways of using language to engage with the audiences to which the text is accessible. As an approach to generate a natural text we use genetic model to generate natural text and evaluate the generated content to meet the required linguistic style using NLG evaluation and word2vec similarity.

II. Literature and background.

In this section of paper lets provide literature to each algorithmic techniques which shall give exhaustive background to build up on the proposed algorithm for natural language generation using genetic evolutionary algorithm.

A. Natural Language Generation

Natural language generation (NLG) is the natural language processing task of generating natural language from a machine representation system such as a knowledge base or a logical form. Psycholinguists prefer the term language production when such formal representations are interpreted as models for mental representations. NLG may be viewed as the opposite of natural language understanding: whereas in natural language understanding the system needs to disambiguate the input sentence to produce the machine representation language, in NLG the system needs to make decisions about how to put a concept into words.

Natural language generation, or NLG, is the sub-field of artificial intelligence and computational linguistics that focuses on the development of computer systems that can produce understandable texts in a human language, starting from some nonlinguistic representation of a communicative goal as input (Reiter and Dale,

2000). To achieve this, NLG systems employ knowledge resources about both the specified human language and the application domain.

Evans et al. (2002) remark that standard definitions of what an NLG system should specifically do are notoriously asymmetrical. For instance, although there seems to be a consensus as to the output of an NLG system, namely human language texts, the form of input it should receive and the different types of processes to be performed on it, vary from system to system. Indeed, comparative studies of existing NLG systems such as Paiva (1998) and Cahill and Reape (1999) reveal many differences as to what, how, and when operations are performed, if at all.

INPUT SPECIFICATION

The input to an NLG system is some nonlinguistic representation of a communicative goal coupled with various knowledge resources.

As formally defined by Reiter and Dale (2000), the input to an NLG system is a four tuple k, c, u, d , where k is the knowledge source to be used, c is the communicative goal to be achieved, u is the user model of the intended audience, and d is the discourse model.

The knowledge source, k, is essentially all the background information about the task domain. Its representation varies from system to system, e.g. first order predicate logic propositions, relational database tables. For instance, ILEX (ODonnell et al., 2001), an NLG system that generates short texts describing pieces of jewellery in a museum, uses a database of known facts and relations concerning the pieces, whereas FOG (Goldberg et al., 1994), a weather forecast report generator, receives as its input a table of sampled meteorological data.

The communicative goal, c, is the purpose of the text to be generated. This usually consists of an underspecification of some propositional information to be conveyed. Some systems further specify rhetorical relations within the propositional information that are to be conveyed by the text. Additionally, c can include non-propositional constraints that must be obeyed by the resulting text, e.g. a requirement that the resulting document fit on a certain number of pages.

The user model, u, is a characterisation of the intended audience for whom the text is gen- erated, which can affect the generated text according to prior knowledge of that audiences expectations. For instance, a stock market reporting system could have a report tailor-made for a broker, an executive analyst, or a public newsfeed. Factors which could be affected by u include content, grammar and stylistic preferences, e.g. a brokers report could be much more in-depth and detailed in terms of content, whereas a public newsfeed could include clear explanations for technical terms.

Finally, the discourse model, d, keeps track of the history of what has been conveyed by the system so far. This is useful during the syntactic and lexical realisation stages, for instance, for referring back to material that has already been conveyed for clarity or brevitys sake, i.e. anaphora.

These last two components, u and d, are not always present in NLG systems, or sometimes they are implicitly hardwired into the system.

Note that linguistic resources, i.e. the grammar and lexicon of the particular human language being targeted, is not specified in Reiter and Dale (2000)s four-tuple, implying that this re- source is an internal component of the NLG system. It is plausible, however, to envision an NLG system for which these linguistic resources are viewed as a separate input, e.g. a multi- lingual system that can target a document to several different languages.

OUTPUT SPECIFICATION

The output from a natural language generation system is, naturally, a text, i.e. a natural lan- guage artefact that can be understood by a human. Most research in NLG is not concerned with the formatting details of the output, and will simply produce strings of words. However, recent research has looked at augmenting NLG systems with the ability to produce texts that incorporate information on visual layout (Bouayad-Agha et al., 2000) and prosody for speech synthesis (Prevost, 1996).

Processes and tasks

A common broadstroke description of the NLG process comprises two levels, the strategic level and the tactical level (Thompson, 1977, McKeown, 1985). The first level can roughly be paraphrased as being the what-to-say level, and the latter the how-to-say-it level. They have also

been referred to as deep generation and surface generation (McKeown and Swartout, 1988), and the conceptualizer and formulator (De Smedt et al., 1996).

The Reiter model is a widely held model that refines the strategic/tactical distinction into three processes (Reiter, 1994):

1. Content determination

The first process is to construct a selection of information from k that realizes the communicative goal in c, or in terms of propositional semantics, is subsumed by the underspecification in c. In certain cases, c may already explicitly provide the exact information to be conveyed. This process of selecting information can be affected by the user model u, e.g. the level of a users technical expertise may determine whether certain domain-specific concepts are explicitly elaborated or simply glossed over.

2. Sentence planning

Once the propositional information has been determined, it must then be structured and ordered so as to produce a coherent and meaningful text. This includes chunking the information into smaller segments, determining the relationship between these segments, and constructing the global syntactic structures that will be used to realize these segments as spans of text, e.g. generating referring expressions, choosing content words, etc.

3. Surface realisation

When the global syntactic structure has been realized, the available linguistic resources are consulted to produce actual spans of text. The tasks here include determining the complete syntactic structure of the sentences, choosing the appropriate function words to be used, and fulfilling all necessary grammatical requirements such as inflectional morphology.

Mellish and Dale (1998) refine this further by identifying six main categories of problems that are addressed in most NLG research: content determination, document structuring, lexicalization, aggregation, referring expression generation, and surface realization.

B. Word2vec word embedding

Word2vec is a group of related models that are used to produce word embeddings. These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space. The word2vec tool takes a text corpus as input and produces the word vectors as output. It first constructs a vocabulary from the training text data and then learns vector representation of words. The resulting word vector file can be used as features in many natural language processing and machine learning applications. A simple way to investigate the learned representations is to find the closest words for a user-specified word. There are two main learning algorithms in word2vec : continuous bag-of-words and continuous skip-gram. The switch -cbow allows the user to pick one of these learning algorithms. Both algorithms learn the representation of a word that is useful for prediction of other words in the sentence. These algorithms are described in detail as follows

CONTINUOUS BAG-OF-WORDS MODEL

The first proposed architecture is similar to the feedforward NNLM, where the non-linear hidden layer is removed and the projection layer is shared for all words (not just the projection matrix); thus, all words get projected into the same position (their vectors are averaged). We call this architecture a bag-of-words model as the order of words in the history does not influence the projection. Furthermore, we also use words from the future; , where the training criterion is to correctly classify the current (middle) word. Training complexity is then

$$Q = N(D + Dlog_2(V)).$$

We denote this model further as CBOW, as unlike standard bag-of-words model, it uses continuous distributed representation of the context. Note that the weight matrix between the input and the projection layer is shared for all word positions in the same way as in the NNLM.

CONTINUOUS SKIP-GRAM MODEL

The second architecture is similar to CBOW, but instead of predicting the current word based on the context, it tries to maximize classification of a word based on another word in the same sentence. More precisely, we use each current word as an input to a log-linear classifier with continuous projection layer, and predict words within a certain range before and after the current word. We found that increasing the range improves quality of the resulting word vectors, but it also increases the computational complexity. Since the more distant words are usually less related to the current word than those close to it, we give less weight to the distant words by sampling less from those words in our training examples.

The training complexity of this architecture is proportional to

$$Q = C(D + Dlog_2(V)),$$

where C is the maximum distance of the words. Thus, if we choose C = 5, for each training word we will select randomly a number R in range ¡ 1; C ¿, and then use R words from history and R words from the future of the current word as correct labels. This will require us to do R 2 word classifications, with the current word as input, and each of the R + R words as output.

C. *Genetic and Evolutionary algorithms with explanation of genetic operators in field of interest.*

As defined in Michalewicz (1996), an evolutionary algorithm, or EA, is a multi-point stochastic search algorithm, which means that it is a form of heuristic search that simultaneously explores several points in a search space, and navigates the search space stochastically so as to prevent getting trapped in local maxima as straightforward hillclimbing algorithms do. All EAs maintain a population of individuals over time t as follows:

$$P(t) = x_i t, ..., x_n t$$

Each individual x represents a potential solution to the problem at hand, and is implemented as some possibly complex data structure. The main algorithm of an EA is as follows: Initialization: For t = 0, construct a new population

$$P(t) = x_i t, ..., x_n t$$

, which represents a set of starting points to explore the search space. Ideally, the distribution of points is evenly spread out across the space. Evaluation: Each solution xit is evaluated to give some measure of its fitness. Selection: A new population P(t +1) is formed by stochastically selecting individuals from P(t) , usually with a bias towards fitter individuals. Evolution: Some members of the new population undergo transformation by means of genetic operators to form new solutions. repeat steps 2 to 4 until termination. Termination is achieved either after (a) a given number of iterations has elapsed, or (b) a given fitness score has been achieved, or (c) the EA has converged to a near-optimal solution, i.e. it has not yielded any better solution for a given number of iterations.

Upon termination of this algorithm, the fittest individual is hoped to be an optimal, or near-optimal, solution.

We will now elaborate on each of the steps above, insofar as it is relevant to our purposes and interests.

INITIALIZATION

As mentioned above, during the initialization phase a population of individuals is created, where each individual represents a starting point to begin exploring the search space.

How initialization is performed varies across EA systems. Davis and Steenstrup (1987) suggest that purely random initialization is useful for research purposes, as evolving a well-adapted solution from a randomly created population is a good test of the EA. The fitness of the resulting solution will have been produced by the search, as opposed to being pre-programmed at the initialization. More practical applications, however, would benefit from a more directed approach.

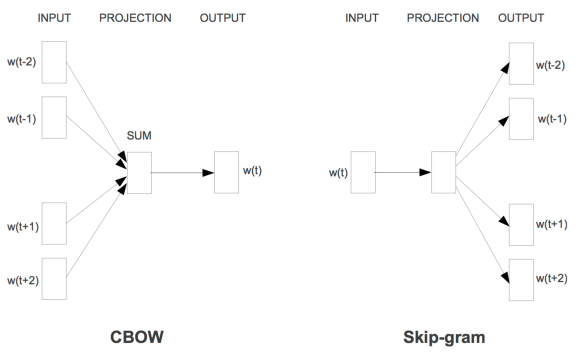

FIGURE 1. THE CBOW AND SKIP-GRAM.

If the random initialization approach is adopted, it is important to bear in mind the issues of implementing constraints, as it is possible that any ill-formed individuals that are created during initialization may propagate throughout the evolution. Hence, randomness here should be qualified as being randomness within the inviolable constraints that the problem domain specifies.

EVALUATION

During the evaluation phase, a fitness function is applied to each individual, yielding a numerical score that indicates its suitability as a solution to the problem at hand.

A fitness function can actually be implemented as an aggregation of a set of separate functions, which we will call evaluators, each of which is designed to measure a different property, or aspect, of an individual.

This multi-objective nature introduces a problem of how to integrate all the scores returned by each evaluator. One can aggregate them in a linear combination, but determining the optimal weight of the contribution of each evaluator is by no means straightforward.

Other methods such as non-Pareto population based and Pareto-optimality based multi-objective optimization have been investigated (Schaffer, 1985, Fonseca and Fleming, 1995).

SELECTION

During the selection phase, a number of individuals are chosen from the current population whose offspring will be present in the next generation. This is the phase in which the driving force in Darwins theory of evolution, natural selection, is simulated.

There are many variations of selection algorithms, such as proportionate, tournament, ranking, and Boltzmann selection (Goldberg, 1989, Ba ck et al., 1997). The main property shared between all of them is the fact that selection is a function of an individuals fitness (Mitchell, 1996). Where they mainly vary is in the stochastic bias towards good individuals, where one extreme, non-regressional evolution, possibly leads to premature convergence, and the other extreme towards inefficient searching through unpromising regions of the space.

John Hollands seminal work on genetic algorithms (Holland, 1975) introduces the notion of intrinsic parallelism, which states that individuals are sampled in such a way that schemata from this space are represented in succeeding generations in proportion to their observed fitness relative to the population average. Schaffer (1987) shows that this property holds for different variations on the selection algorithm. In essence, there exists a degree of freedom of choice for the selection algorithm, as whichever is chosen, the principles of evolution will still apply.

EVOLUTION

An EA has a set of genetic operators, which are functions to modify a solution, yielding a different solution, i.e. to cause a move in the search space. These operators are either unary functions mi (mutators), i.e.

$$m_i : S \to S$$

that randomly perturb the properties of a solution, or higher arity functions ci (crossovers), i.e.

$$c_i : S * .... * S \rightarrow S$$

, that attempt to combine the properties of two or more solutions. The domain S is the set of all possible individuals, or candidate solutions.

Spears (1992) notes that historically, proponents of the Holland (1975) style of genetic algorithm tend to believe that crossover is more powerful than mutation, whereas researchers using evolution strategies, another type of evolutionary algorithms, view mutation as the key genetic operator. The results in Spears (1992) show that both operators have their strengths and weaknesses, that are linked to the issues of exploitation vs. exploration: mutation is best used to create random diversity in a population, while crossover serves as an accelerator that promotes emergent behaviour from partial solutions. The balance between these two processes is dependent on the specific application, and even implementation, of the EA.

D. Overgeneration and ranking methods

In recent years, an alternative method of NLG has become increasingly popular. It is based on the generate and test principle, a well known approach to problem solving in AI. The underlying idea is to shift the burden of domain knowledge from a generation component, where knowledge about the solving of complex multiple constraints is limited, to an evaluation component that can identify a possible solution as being an optimal one. An implicit assumption behind this idea is that it is easier to state domain knowledge in terms of what a good text should be, i.e. a discriminative model, rather than how a good text should be written, i.e. a generative model.

(Knight and Hatzivassiloglou, 1995, Langkilde and Knight, 1998) first introduced this novel method of generation. The motivation for building NITROGEN, their implemented system, was to increase robustness for sentence generation in the face of incomplete and possibly ill- formed input. This is a problem that they had to deal with in machine translation, their application domain. They achieve this by exploiting the power of corpus-based statistical models in place of extensive linguistic knowledge. Such models, such as n-grams, have enjoyed significant success in other areas of computational linguistics (Manning and Schu tze, 1999).

NITROGEN consists of two main components: a symbolic generator and a statistical extractor. The symbolic generator works from underspecified input and simple lexical, morphological and grammatical knowledge bases. It is consciously designed to be simple and knowledge-poor, and hence pays no heed to linguistic decisions such as word choice, number, determinateness, agreement, tense, etc. Instead, it over generates many alternative utterances to convey the input, and relies heavily on the corpus knowledge of the extractor to provide the answers to these choices.

The output of the generator is a word lattice, a compact representation of multiple generation possibilities. A statistical extractor selects the most fluent path through the lattice using bigram and unigram statistics collected from two years of the Wall Street Journal.

Bangalore and Rambow (2000b,a) present a very similar approach in their system, FERGUS. The main difference is that their chosen formalism for the symbolic generation phase is TAG. This tree-based account is claimed to capture more complex linguistic features than NITROGENs simple surface-based rules. Accordingly, they present a statistical language model based on trees for the ranking phase.

Oberlander and Brew (2000) follow Ward (1994) in arguing that NLG systems must achieve fidelity and fluency goals, where fidelity is the faithful representation of the relevant knowledge contained within the communicative goal, and fluency is the ability to do it in a natural sounding way such that it engenders a positive evaluation of the system by the user.

It is this fluency goal that sets NLG research apart from NLU (Natural Language Understanding). An NLU system must recover meaning representations from strings of text, and whether or not a given string sounds natural, elegant, or forceful is considered far less important than identifying its propositional content.

In practice, applied NLG systems often sidestep the fluency goal by targeting a very restricted domain of output, with a limited style that may be just enough to serve the purpose of the application. Oberlander and Brew (2000) introduce two kinds of fluency goals that they

argue are worth achieving from a usability engineering perspective: maximizing syntactic expectedness and maximizing user satisfaction.

To exemplify this, they stipulate two simple scenarios where NLG systems must be able to control the sentence length and the vocabulary diversity of their generated text. The difficulty is that these features are emergent, macroscopic properties of a large number of decisions, few of which are based solely on stylistic considerations. Thus it is difficult to identify and control decisions in such a way as to satisfy the requirements imposed on the text.

They propose an architecture which consists of two modules: an author and a reviewer. The author faces the task of generating a text that conveys the correct propositional content, or in short, achieving the fidelity goal. On the other hand, the reviewer must ensure that the authors output satisfies whatever macroscopic properties have been imposed on it. In other words, achieving the fluency goal. In doing so, the two modules may collaborate to produce a mutually acceptable text.

They note that Langkilde and Knights NITROGEN system essentially embodies this architecture: the symbolic generator acts as the author, and the statistical language model is the reviewer. The difference here is that instead of collaborating, the author produces all possible drafts that it considers acceptable from its point of view, and the reviewer selects the version that best meets its needs.

They show how by slightly modifying the NITROGEN model, one can implement a system that achieves the example constraints of sentence length and vocabulary diversity. The modification is to implement both the author and reviewer as different stochastic models, unlike NITROGEN where only the latter model is a stochastic model. In particular, the reviewer language model characterizes the desired distributions of sentence length and vocabulary diversity through the use of techniques such as negative binomial distributions and maximum entropy. Additionally, the word lattice from the author module is augmented with weights attached to each path, which reflects the degree of acceptability that the author assigns to the utterance represented by that path. They propose that the two stochastic models can be combined through a linear combination, where the extreme parameters would stand for systems that exclusively satisfy fidelity or fluency goals.

Varges (2002) presents another overgeneration system that is inspired by NITROGEN. It is significantly different in two ways. Firstly, it adopts a lazy learning technique in that domain knowledge, in this case linguistic knowledge, is not compiled into a statistical model such as n-grams, but rather maintained as a set of exemplar instances. The candidate texts produced by the symbolic generator are evaluated by a distance metric comparing them to these instances. This technique is useful for applications where there is only a limited size corpus.

The second difference is that the phases of generation and ranking are interleaved. This is enabled by the use of a chart as the data structure shared between the generator and ranker, instead of a complete word lattice as in NITROGEN. As the generator incrementally adds edges to the chart, the ranker monitors the chart and, by employing the A search algorithm, an expectation-based heuristic search technique, it is able to prune the search space by removing edges that are known to be suboptimal.

E. TAG and Generation

Elementary trees are minimal linguistic structures that are enough to capture dependencies locally. Put another way, an elementary tree accounts for all and only the arguments of its head. This allows the local definability of all dependencies, the locality of feature checking, and the locality of argument structure (Joshi, 1987). These trees are said to have an extended domain of locality compared to, for example, the rules in CFG-based formalisms. Furthermore, the adjunction operation factors recursion in such a way that argument structure is preserved over arbitrarily long distances. Finally, within LTAG, each elementary tree has a lexical anchor which is the semantic head of the constituent represented by the tree.

These features make elementary trees the appropriate syntactic building blocks to work with while incrementally consuming semantic input to be generated. Recent tactical generators such as SPUD and PROTECTOR employ TAG for a flexible approach to generation. They first construct the skeletal frame of a sentence, i.e. the verb and its complements, via substitution.

Later on, they add modifiers and adjuncts via the adjoining in of adjectives, adverbials, etc.

III. Proposed algorithm

The scenario we are trying to tackle is to create literature which is unique and on the contrary yet follows similar linguistic style. The granularity to effectuate the proposed genetic algorithm of the natural language generation we have covered the theory in section 2. Given the power of vocabulary of the theory the algorithm can now be defined

A. *The algorithm*

The genetic algorithm of an EA for natural language text generation is as follows:

1) Initialization: For t = 0, construct a new population

$$P(t) = x_i t, .., x_n t,$$

which represents a set of starting points to explore the search space. Ideally, the distribution of points is evenly spread out across the space.The initialization is performed by Overgeneration and ranking methods as covered in section 2.3

2) Evaluation: Each solution

$$x_i t$$

is evaluated to give some measure of its fitness. The evaluation criteria is dual fold in our case. First the solution shall fit in the paradigm of natural language generation output specification as in section 2.1. Second the solution should be selected evaluated on fitness score achieved by Word2vec explained in section 2.2

3) Selection: A new population P(t +1) is formed by stochastically selecting individuals from P(t) , usually with a bias towards fitter individuals on the score of evaluation.

4) Evolution: Some members of the new population undergo transformation by means of genetic operators to form new solutions. The evolution is implemented by genetic operator cover in section 2.4 (TAG and Generation)

5) repeat steps 2 to 4 until termination. Termination is achieved either after
(a) a given number of iterations has elapsed, or
(b) a given fitness score has been achieved, or
(c) the EA has converged to a near-optimal solution, i.e. it has not yielded any better solution for a given number of iterations.

IV. Future work

The algorithm being proposed is to be implemented to evaluate the performance and improvise. The non trivial optimizing for this algorithm is indeed in the Initialization and Evaluation criteria as the search space is vast for natural language. We intend to create a concrete implementation to prove the algorithmic approached proposed via this document.

V. References

[1] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. In Proceedings of Workshop at ICLR, 2013.
[2] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. In Proceedings of NIPS, 2013.
[3] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic Regularities in Continuous Space Word Representations. In Proceedings of NAACL HLT, 2013.
[4] Natural language generation https://en.wikipedia.org/.
[5] Genetic algorithm https://en.wikipedia.org/.
[6] Stylistic Transfer in Natural Language Generation Systems Using Recurrent Neural Networks. Jad Kabbara and Jackie Chi Kit Cheung.
[7] Efficient Estimation of Word Representations in Vector Space. Tomas Mikolov,Greg Corrado, Kai Chen, Jeffrey Dean.
[8] An evolutionary algorithm approach to poetry generation. Hisar Maruli Manurung.