Modelling Sentence Generation from Sum of Word Embedding Vectors as a Mixed Integer Programming Problem

Lyndon White, Roberto Togneri, Wei Liu Mohammed Bennamoun
The University of Western Australia
35 Stirling Highway, Crawley, Western Australia
lyndon.white@research.uwa.edu.au
{roberto.togneri, wei.liu, mohammed.bennamoun}@uwa.edu.au

Abstract—Converting a sentence to a meaningful vector representation has uses in many NLP tasks, however very few methods allow that representation to be restored to a human readable sentence. Being able to generate sentences from the vector representations demonstrates the level of information maintained by the embedding representation – in this case a simple sum of word embeddings. We introduce such a method for moving from this vector representation back to the original sentences. This is done using a two stage process; first a greedy algorithm is utilised to convert the vector to a bag of words, and second a simple probabilistic language model is used to order the words to get back the sentence. To the best of our knowledge this is the first work to demonstrate quantitatively the ability to reproduce text from a large corpus based directly on its sentence embeddings.

1 Introduction

Generally sentence generation is the main task of the more broad natural language generation field; here we use the term only in the context of sentence generation from sentence vector representation. For our purposes, a sentence generation method has as its input a sentence embedding, and outputs the sentence which it corresponds to. The input is a vector, for example $tilde07Es = [0.11, 0.57, -0.21, ..., 1.29]$, and the output is a sentence, for example "The boy was happy.".

Dinu2014CompositionalGeneration motivates this work from a theoretical perspective given that a sentence encodes its meaning, and the vector encodes the same meaning, then it must be possible to translate in both directions between the natural language and the vector representation. In this paper, we present an implementation that indicates to some extent the equivalence between the natural language space and the sum of word embeddings (SOWE) vector representation space. This equivalence is shown by demonstrating a lower bound on the capacity of the vector representation to be used for sentence generation.

The current state of the art methods for sentence generation produce human readable sentences which are rough approximations of the intended sentence. These existing works are those of iyyer2014generating and Bowman2015SmoothGeneration. Both these have been demonstrated to produce full sentences. These sentences are qualitatively shown to be loosely similar in meaning to the original sentences. Neither work has produced quantitative evaluations, making it hard to compare their performance. Both are detailed further in relwork. Both these methods use encoder/decoder models trained through machine learning; we present here a more deterministic algorithmic approach, but restrict the input sentence vector to be the non-compositional sum of word embeddings representation.

RitterPosition and White2015SentVecMeaning found that when classifying sentences into categories according to meaning, simple SOWE outperformed more complex sentence vector models. Both works used sentence embeddings as the input to classifiers. RitterPosition classified challenging artificial sentences into categories based on the positional relationship described using Naïve Bayes. White2015SentVecMeaning classified real-world sentences into groups of semantically equivalent paraphrases. In the case of RitterPosition this outperformed the next best representation by over 5%. In the case of White2015SentVecMeaning it was within a margin of 1% from the very best performing method. These results suggest that there is high consistency in the relationship between a point in the SOWE space, and the meaning of the sentence.

wieting2015towards presented a sentence embedding based on the related average of word-embedding, showing excellent performance across several competitive tasks. They compared their method's performance against several models, including recurrent neural networks, and long short term memory (LSTM) architectures. It was found that their averaging method outperformed the more complex LSTM system, on most sentence similarity and entailment task. Thus these simple

standalone
tikz positioning
[every node/.style= text width=4em, align=center, font=, inner sep=1pt , proc/.style= draw, font=, inner sep = 2pt] (input) [inner sep=-4pt] SOWE Sentence Vector; (selection) [proc, right = 0.7em of input]Word Selection; (ordering) [proc, right = 3.4em of selection]Word Ordering; (vocab) [above = 1em of selection]Vocabulary of Word Vectors; (lm) [above = 1em of ordering] Stochastic Language Model; (output) [inner sep=-4pt, right=0.7em of ordering] Natural Language Sentence; [-¿] (input) – (selection); [-¿] (vocab) – (selection); [-¿] (selection) – (ordering) node[midway] Bag of Words; [-¿] (lm) – (ordering); [-¿] (ordering) – (output); pgfexternal@did@a@shipout

Fig. 1. The process for the regenerating sentences from SOWE-type sentence vectors.

methods are worth further consideration. SOWE is the basis of the work presented in this paper.

Our method performs the sentence generation in two steps, as shown in block$_d$iagram.Itcombinestheworkof White2015BOWgen on generating phrases from bags of words (BOW) from unordered embeddings (SOW sentence representations). These input and output stages are performed using long short-term memory recurrent neural networks hochreiter1997long. They demonstrate a number of uses of this technique, one of which is sentence generation, in the sense of this paper. While Bowman et al. do define a generative model, they do not seek to recreate a sentence purely from its vector input, but rather to produce a series of probability distributions on the words in the sentence. These distributions can be evaluated greedily, which the authors used to give three short examples of resynthesis. They found the sentence embeddings created captured largely syntactic and loose topical information.

The rest of the paper is organized into the following sections. relwork discusses the prior work on sentence generation. framework explains the problem in detail and how our method is used to solve it. evalsettings describes the settings used for evaluation. results presents the results of this evaluation. The paper concludes with conclusion and a discussion of future work on this problem.

2 Related Works

To the best of our knowledge only three prior works exist in the area of sentence generation from embeddings. The first two (Dinu2014CompositionalGeneration, iyyer2014generating) are based on the recursive structures in language, while Bowman2015SmoothGeneration, uses the sequential structure.

Dinu2014CompositionalGeneration extends the models described by zanzotto2010estimating and Guevara2010 for generation. The composition is described as a linear transformation of the input word embeddings to get an output vector, and another linear transformation to reverse the composition reconstructing the input. The linear transformation matrices are solved using least squares regression. This method of composing, can be applied recursively from words to phrases to clauses and so forth. It theoretically generalises to whole sentences, by recursively applying the composition or decomposition functions. However, Dinu and Baroni's work is quantitatively assessed only on direct reconstruction for decomposing Preposition-Noun and Adjective-Noun word phrases. In these cases where the decomposition function was trained directly on vectors generated using the dual composition function they were able to get perfect reconstruction on the word embedding based inputs.

iyyer2014generating extends the work of SocherEtAl2011:PoolRAE defining an unfolding recursive dependency-tree recursive autoencoder (DT-RAE). Recursive neural networks are jointly trained for both composing the sentence's words into a vector, and for decomposing that vector into words. This composition and decomposition is done by reusing a composition neural network at each vertex of the dependency tree structure, with different weight matrices for each dependency relation. The total network is trained based on the accuracy of reproducing its input word embeddings. It can be used to generate sentences, if a dependency tree structure for the output is provided. This method was demonstrated quantitatively on five examples; the generated sentences were shown to be loosely semantically similar to the originals.

Bowman2015SmoothGeneration uses a a modification of the variational autoencoder (VAE) kingma2013auto with natural language inputs and outputs, to learn the sentence representations. These input and output stages

We note that none of the aforementioned works present any quantitative evaluations on a corpus of full sentences. We suggest that that is due to difficulties in evaluation. As noted in iyyer2014generating and Bowman2015SmoothGeneration, they tend to output lose paraphrases, or roughly similar sentences. This itself is a separately useful achievement to pure exact sentence generation; but it is not one that allows ready interpretation of how much information is maintained by the embeddings. Demonstration of our method at generating the example sentences used in those work is available as supplementary material[1]. As our method often can exactly recreate the original sentence from its vector representation evaluation is simpler.

Unlike current sentence generation methods, the non-compositional BOW generation method of White2015BOWgen generally outputs a BOW very close to the reference for that sentence – albeit at the

1. http://white.ucc.asn.au/publications/White2016SOWE2Sent/

cost of losing all word order information. It is because of this accuracy that we base our proposed sentence generation method on it (as detailed in selection). The word selection step we used is directly based on their greedy BOW generation method. We improve it for sentence generation by composing with a word ordering step to create the sentence generation process.

3 General Framework

As discussed in intro, and shown in $block_diagram, the approach taken to generate the sentences from the vector sc omes in two steps. First selecting the words used - - this is done deterministically, based on a search of the embedding space. Second is to order them, which was solved by finding the most likel Hard computer science problems; thus variations on known techniques can be used to solve them$

3.1 Word Selection

White2015BOWgen approaches the BOW generation problem, as task of selecting the vectors that sum to be closest to a given vector. This is related to the knapsack and subset sum problems. They formally define the vector selection problem as:

$$(tilde07Es, \mathcal{V}, d) \mapsto_{\left\{\forall tilde07Ec \in N_0^{|\mathcal{V}|}\right\}} d(tilde07Es, \sum_{tilde07Ex_j \in tilde07Ec} tilde07Ex_j, p)$$

to find the bag of vectors selected from the vocabulary set $\mathcal{V}$ which when summed is closest to the target vector $tilde07Es$. Closeness is assessed with distance metric d. $tilde07Ec$ is the indicator function for that multi-set of vectors. As there is a one to one correspondence between word embeddings and their words, finding the vectors results in finding the words. White2015BOWgen propose a greedy solution to the problem[2].

The key algorithm proposed by White2015BOWgen is greedy addition. The idea is to greedily add vectors to a partial solution building towards a complete bag. This starts with an empty bag of word embeddings, and at each step the embedding space is searched for the vector which when added to the current partial solution results in the minimal distance to the target – when compared to other vectors from the vocabulary. This step is repeated until there are no vectors in the vocabulary that can be added without moving away from the solution. Then a fine-tuning step, n-substitution, is used to remove some simpler greedy mistakes.

The n-substitution step examines partial solutions (bags of vectors) and evaluates if it is possible to find a better solution by removing n elements and replacing them with up-to n different elements. The replacement search is exhaustive over the n-ary Cartesian product of the vocabulary. Only for $n = 1$ is it currently feasible for practical implementation outside of highly restricted vocabularies. Never-the-less even 1-substitution can be seen as lessening the greed of the algorithm, through allowing early decisions to be reconsidered in the full context of the partial solution. The algorithm does remain greedy, but many simple mistakes are avoided by n-substitution. The greedy addition and n-substitution processes are repeated until the solution converges.

3.2 The Ordering Problem

After the bag of words has been generated by the previous step, it must be ordered (sometimes called linearized). For example "are how , today hello ? you", is to be ordered into the sentence: "hello , how are you today ?". This problem cannot always be solved to a single correct solution. Mitchell2008 gives the example of "It was not the sales manager who hit the bottle that day, but the office worker with the serious drinking problem." which has the same word content (though not punctuation) as "That day the office manager, who was drinking, hit the problem sales worker with a bottle, but it was not serious.". However, while a unique ordering cannot be guaranteed, finding the most likely word ordering is possible. There are several current methods for word ordering

To order the words we use a method based on the work of Horvat2014, which uses simple trigrams. More recent works, such as beam-search and LSTM language model and proposed by 2016arXiv160408633S; or a syntactic rules based method such as presented in Zhang-WordOrderSyntax, could be used. These more powerful ordering methods internalise significant information about the language. The classical trigram language model we present is a clearer baseline for the capacity to regenerate the sentences; which then be improved by using such systems.

Horvat2014 formulated the word ordering problem as a generalised asymmetrical travelling salesman problem (GA-TSP). fig:ordergraph shows an example of the connected graph for ordering five words. We extend beyond the approach of Horvat2014 by reformulating the problem as a linear mixed integer programming problem (MIP). This allows us to take advantage of existing efficient solvers for this problem. Beyond the GA-TSP approach, a direct MIP formulation allows for increased descriptive flexibility and opens the way for further enhancement. Some of the constraints of a GA-TSP can be removed, or simplified in the direct MIP formulation for word ordering. For example, word ordering does have distinct and known start and end nodes (as shall be detailed in the next section). To formulate it as a GA-TSP it must be a tour without beginning or end. Horvat2014 solve this by simply connecting the start to the end with a zero cost link. This is not needed if formulating this as a MIP problem, the start and end nodes can be treated as special cases. Being able to special case them as nodes known always to occur allows some simplification in

2. We also investigated beam search as a possible improvement over the greedy addition and n-substitution used by White2015BOWgen, but did not find significant improvement. The additional points considered by the beam tended to be words that would be chosen by the greedy addition in the later steps – thus few alternatives where found.

standalone arrayjobx xifthen tikz,textcomp amssymb
shorten ¡¿/.style= shorten ¿=0.2cm, shorten ¡=0.2cm
Words$w_\triangleright w_1 w_2 w_3 w_4 w_\triangleleft$ =5
in 1,...,6 MyColorblack1=MyColorgray =1MyColorgray
[MyColor]
(w$_{1\text{J}}$)$at(0cm, 0cm - 0.5 * \text{J}cm)(1)(\text{J}); (0cm, 0cm - 3.5 * 0.5cm)ellipse(0.7cmand0.5 * 4cm); at(0cm, 0cm + 1 * 0.5cm)$S((1));
in 1,...,6 MyColorblack2=MyColorgray =1MyColorgray
[MyColor] (w$_{2\text{J}}$)$at(2.5cm, 2.5cm - 0.5 * \text{J}cm)(2)(\text{J}); (2.5cm, 2.5cm - 3.5 * 0.5cm)ellipse(0.7cmand0.5 * 4cm); at(2.5cm, 2.5cm + 1 * 0.5cm)$S((2));
in 1,...,6 MyColorblack3=MyColorgray =1MyColorgray
[MyColor] (w$_{3\text{J}}$)$at(5cm, -1.5cm - 0.5 * \text{J}cm)(3)(\text{J}); (5cm, -1.5cm - 3.5 * 0.5cm)ellipse(0.7cmand0.5 * 4cm); at(5cm, -1.5cm - 8 * 0.5cm)$S((3)); $\text{J}in1, ..., 6 MyColorblack4 = \text{J}MyColorgray\text{J} = 1MyColorgray$
[MyColor] (w$_{3\text{J}}$)$at(5cm, -1.5cm - 0.5 * \text{J}cm)(3)(\text{J}); (5cm, -1.5cm - 3.5 * 0.5cm)ellipse(0.7cmand0.5 * 4cm); at(5cm, -1.5cm + 1 * 0.5cm)$S((3));
in 1,...,6 MyColorblack5=MyColorgray =1MyColorgray
[MyColor]
(w$_{5\text{J}}$)$at(10cm, 0cm - 0.5 * \text{J}cm)(5)(\text{J}); (10cm, 0cm - 3.5 * 0.5cm)ellipse(0.7cmand0.5 * 4cm); at(10cm, 0cm + 1 * 0.5cm)$S((5));
5in 1,...,5 in 2,...,5 MyColorblack=2MyColorcyan!50!black
=3MyColorblue!50!black=4MyColoryellow!50!black=5MyColorgreen!50!black
5= in 2,...,6 =5= [-¿,MyColor] (w$_{5\text{J}}$) $- -(w_{\text{J}});$
(w$_S$)$at(-2cm, -1.5)$w(1); (-2cm,-1.5) ellipse (0.7cm and 1 cm); at (-2cm,-1.5cm +1.3 cm) $S(w)$;
in 2,...,5 [-¿, red!70!black] (w$_S$) $- -(w_{1\text{J}})$;
(w$_E$)$at(12cm, -1.5)(6)$w; (12cm,-1.5) ellipse (0.7cm and 1 cm); at (12cm,-1.5cm +1.3 cm) $S((6))$; in 1,...,5 [-¿, red!70!black] (w$_{\text{J}6) - -(w_E);}$
pgfexternal@did@a@shipout

Fig. 2. A graph showing the legal transitions between states, when the word-ordering problem is expressed similar to a GA-TSP. Each edge $\langle w_a, w_b\rangle \to \langle w_c, w_d\rangle$ has cost $-\log(P(w_c \mid w_a w_b)$. The nodes are grouped into districts (words). Nodes for invalid states are greyed out.

the subtour elimination step. The formulation to mixed integer programming is otherwise reasonably standard.

standalone verbatim amsthm amsmath amssymb mathtools multirow resizegather

3.2.1 Notation

We will write w_i to represent a word from the bag $\mathcal{W}$ ($w_i \in \mathcal{W}$), with arbitrarily assigned unique subscripts. Where a word occurs with multiplicity greater than 1, it is assigned multiple subscripts, and is henceforth treated as a distinct word.

Each vertex is a sequence of two words, $\langle w_i, w_j\rangle \in \mathcal{W}^2$. This is a Markov state, consisting of a word w_j and its predecessor word w_i – a bigram.

Each edge between two vertices represents a transition from one state to another which forms a trigram. The start vertex is given by $\langle w, w_\triangleright\rangle$, and the end by $\langle w_\triangleleft, w\rangle$. The pseudowords $w, w_\triangleright, w_\triangleleft, w$ are added during the trigram models' training allowing knowledge about the beginning and ending of sentences to be incorporated.

The GA-TSP districts are given by the sets of all states that have a given word in the first position. The district for word w_i is given by $S(w_i) \subseteq \mathcal{W}^2$, defined as $S(w_i) = \{\langle w_i, w_j\rangle \mid \forall w_j \in \mathcal{W}\}$. It is required to visit every district, thus it is required to use every word. With this description, the problem can be formulated as a MIP optimisation problem.

3.2.2 Optimization Model

Every MIP problem has a set of variables to optimise, and a cost function that assesses how optimal a given choice of values for that variable is. The cost function for the word ordering problem must represent how unlikely a particular order is. The variables must represent the order taken. The variables are considered as a table (τ) which indicates if a particular transition between states is taken. Note that for any pair of Markov states $\langle w_a, w_b\rangle, \langle w_c, w_d\rangle$ is legal if and only if $b = c$, so we denote legal transitions as $\langle w_i, w_j\rangle \to \langle w_j, w_k\rangle$. Such a transition has cost: C$[\langle w_i, w_j\rangle, \langle w_j, w_k\rangle] = -\log\left(P(w_k|w_i, w_j\rangle\right)$ The table of transitions to be optimized is: $\tau[\langle w_i, w_j\rangle, \langle w_j, w_k\rangle] = \{$ 2∗1if transition from $\langle w_i, w_j\rangle \to \langle w_j, w_k\rangle$ occurs
0otherwise The total cost to be minimized, is given by $\text{C}_{total}(\tau) = \sum_{w_i, w_j, w_k \in \mathcal{W}^3} \tau[\langle w_i, w_j\rangle, \langle w_j, w_k\rangle] \cdot C[\langle w_i, w_j\rangle, \langle w_j, w_k\rangle]$

The probability of a particular path (i.e. of a particular ordering) is thus given by $P(\tau) = e^{-C_{total}(\tau)}$

The word order can be found by following the links. The function $f_\tau(n)$ gives the word

that, according to τ occurs in the nth position.

$$\mathrm{f}_\tau(1) = \{w_a \mid \tau[\langle w, w_\triangleright\rangle, \langle w_\triangleright, w_a\rangle] = 1\}_1$$
$$f_\tau(2) = \{w_b \mid \tau[\langle w_\triangleright, f_\tau(1)\rangle, \langle f_\tau(1), w_b\rangle] = 1\}_1$$
$$f_\tau(n) = \text{when } n \geq 3\{w_c \mid \tau[\langle f_\tau(n-2), f_\tau(n-1)\rangle, \langle f_\tau(n-1), w_c\rangle] = 1\}_1$$

+ 0.5cm+ 0.5cm when $n \geq 3\{w_c \mid w_c \in \mathcal{W} \wedge \tau[\langle f_\tau(n-2), f_\tau(n-1)\rangle, \langle f_\tau(n-1), w_c\rangle] = 1\}_1$

The notation $\{\}_1$ indicates taking a singleton set's only element. The constraints on τ ensure that each set is a singleton.

3.2.3 Constraints

The requirements of the problem, place various constraints on to τ: The Markov state must be maintained: $\forall\langle w_a, w_b\rangle, \langle w_c, w_d\rangle \in \mathcal{W}^2$: $\mathrm{w}_b \neq w_c \tau[\langle w_a, w_b\rangle, \langle w_c, w_d\rangle] = 0$ Every node entered must also be exited – except those at the beginning and end. $\forall\langle w_i, w_j\rangle \in \mathcal{W}^2 \backslash \{\langle w, w_\triangleright\rangle, \langle w_\triangleleft, w\rangle\}$: $\sum_{\forall\langle w_a, w_b\rangle \in \mathcal{W}^2} \tau[\langle w_a, w_b\rangle, \langle w_i, w_j\rangle] = \sum_{\forall\langle w_c, w_d\rangle \in \mathcal{W}^2} \tau[\langle w_i, w_j\rangle, \langle w_c, w_d\rangle]$ Every district must be entered exactly once. i.e. every word must be placed in a single position in the sequence. $\forall w_i \in \mathcal{W} \backslash \{w, w\}$: $\sum^{\forall\langle w_i, w_j\rangle \in S(w_i)} \sum_{\forall\langle w_a, w_b\rangle \in \mathcal{W}^2} \tau[\langle w_a, w_b\rangle, \langle w_i, w_j\rangle] = 1$

To allow the feasibility checker to detect if ordering the words is impossible, transitions of zero probability are also forbidden. i.e. if $P(w_n | w_{n-2}, w_{n-1}) = 0$ then $\tau[\langle w_{n-2}, w_{n-1}\rangle, \langle w_{n-1}, w_n\rangle] = 0$. These transitions, if not expressly forbidden, would never occur in an optimal solution in any case, as they have infinitely high cost.

3.2.3.1 Lazy Subtour Elimination Constraints: The problem as formulated above can be input into a MIPS solver. However, like similar formulations of the travelling salesman problem, some solutions will have subtours. As is usual callbacks are used to impose lazy constraints to forbid such solutions at run-time. However, the actual formulation of those constraints are different from a typical GA-TSP.

Given a potential solution τ meeting all other constraints, we proceed as follows.

The core path – which starts at $\langle w, w_\triangleright\rangle$ and ends at $\langle w_\triangleleft, w\rangle$ can be found. This is done by practically following the links from the start node, and accumulating them into a set $T \subseteq \mathcal{W}^2$

From the core path, the set of words covered is given by $\mathcal{W}_T = \{w_i \mid \forall\langle w_i, w_j\rangle \in T\} \cup \{w\}$. If $\mathcal{W}_\mathcal{T} = \mathcal{W}$ then there are no subtours and the core path is the complete path. Otherwise, there is a subtour to be eliminated.

If there is a subtour, then a constraint must be added to eliminate it. The constraint we define is that there must be a connection from at least one of the nodes in the district covered by the core path to one of the nodes in the districts not covered.

The districts covered by the tour are given by $S_T = \bigcup_{w_t \in \mathcal{W}_T} S(w_t)$. The subtour elimination constraint is given by $\sum^{\forall\langle w_{t1}, w_{t2}\rangle \in S_T} \sum_{\forall\langle w_a, w_b\rangle \in \mathcal{W}^2 \backslash S_T} \tau[\langle w_{t1}, w_{t2}\rangle, \langle w_a, w_b\rangle] \geq 1$ i.e. there must be a transition from one of the states featuring a word that is in the core path, to one of the states featuring a word not covered by the core path.

This formulation around the notion of a core path that makes this different from typical subtour elimination in a GA-TSP. GA-TSP problems are not generally guaranteed to have any nodes which must occur. However, every word ordering problem is guaranteed to have such a node – the start and end nodes. Being able to identify the core path allows for reasonably simple subtour elimination constraint definition. Other subtour elimination constraints, however, also do exist.

pgfexternal@did@a@shipout