Keywords-To-Text Synthesis Using Recurrent Neural Network

Nikolaos Kolokas(✉), Anastasios Drosou, and Dimitrios Tzovaras

Center for Research and Technology Hellas, Thermi, Thessaloniki, Greece
{nikolokas,drosou,dimitrios.tzovaras}@iti.gr
pitygonos@gmail.com

Abstract. This paper concerns an application of Recurrent Neural Networks to text synthesis in the word level, with the help of keywords. First, a Parts Of Speech tagging library is employed to extract verbs and nouns from the texts used in our work, a part of which are then considered, after automatic eliminations, as the aforementioned keywords. Our ultimate aim is to train a Recurrent Neural Network to map the keyword sequence of a text to the entire text. Successive reformulations of the keyword and full text word sequences are performed, so that they can serve as the input and target of the network as efficiently as possible. The predicted texts are understandable enough, and their performance depends on the problem difficulty, determined by the percentage of full text words that are considered as keywords (ranging from 1/3 to 1/2), and the training memory cost, mainly affected by the network architecture.

Keywords: Deep machine learning · Sequence modeling
Natural language processing · Text mining

1 Introduction

Keywords-to-text synthesis appertains to the general field of the well known Natural Language Processing (NLP). NLP regards the understanding of a human language by the computer, and also, conversely, the ability of the computer to synthesize text or speech. Examples of applications include speech recognition, machine translation, text-to-speech synthesis, Parts Of Speech (POS) tagging and text summarization [1].

NLP research generally started in the 1950's, but machine learning techniques for NLP were firstly employed in the 1980's, when the tasks started to be solved by statistical inference instead of the former disadvantageous handwritten rules. Even more especially, deep learning was involved very recently [1, 2]. Most approaches are supervised, but recently semi-supervised and unsupervised approaches are investigated as well. In this paper the supervised option is preferred, because it demands less data to achieve desirable performance.

In this work the problem of keywords-to-text synthesis is addressed. Especially, our main goal is to provide a tool which, taking a keyword sequence as input, is able to produce a full text containing the keywords, or synonyms of them, in the same order. This synthesis facilitates the text composition, since it demands less typing by the user. The Recurrent Neural Network (RNN) is chosen as such a tool, since RNNs are particularly appropriate for sequence modeling problems and they have many applications related to NLP, knowledge representation, reasoning and question answering [1, 3]. This paper concerns a supervised learning method and some full texts are used as a data set. As for the keywords, here they are defined as the least necessary words from which a unique full text can be inferred. Since such words are almost always verbs and nouns, the verbs and nouns are extracted from the texts using an available POS tagging algorithm. Finally, for the automatic synonym matching, a library containing a dictionary which includes synonyms for each of its words is used.

In this paper a relatively simple family (among those presented in [1, 2]) of an RNN is eventually employed. It is empirically confirmed that our text synthesis goal does not demand gated RNNs [e.g. Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU)], which deal with long-term dependencies, something rather irrelevant to this problem, where for a word production the present and a few neighboring input keywords and full text predicted words obviously suffice, as also shown by the results.

The remainder of the paper is organized as follows. In Sect. 2 related work is cited and the uniqueness of ours is briefly described. In Sect. 3 the preprocessing of our data set before training is presented, and in Sect. 4 our experimental results from several training approaches are shown and discussed. Finally, Sect. 5 summarizes our work and proposes possible next steps.

2 Previous Work and Motivation

In this section previous work is presented, related to the two coarse phases of ours: the keywords extraction from texts and the reproduction of the texts (text synthesis) from these keywords.

Considerable work on keywords extraction and text summarization is found in the bibliography, e.g. in [4–8]. The criteria of defining/selecting keywords, or even the goal, differ among references, so the methodologies are not directly linked and comparable with each other and with ours. The work of [4] is somehow relevant to ours, because in that paper the keywords are considered as the nouns (as characterized by a POS tagging algorithm) that imply many other nouns in the same sentence. However, the final aim in that work is not the text synthesis from the extracted keywords, which constitute a percentage of the total word number that is too small to serve such a goal. Thus, despite our inspiration by the cited keywords extraction methodology, it needed to be modified in our work. In [5] the purpose of keyword extraction is the classification (annotation) of texts, so words are evaluated by the number of families in which they appear and their mean frequency in them. In [6] keywords are defined according to their frequency and extracted from abstracts and titles. Finally, in [7, 8] keywords [7] or whole key-sentences [8] are extracted from texts according to several criteria, with the purpose of summarization. Particularly, in [7] a supervised and an unsupervised approach (both graph-based) are introduced for the extraction.

References about text synthesis regard mainly text-to-speech, speech-to-text and text-to-image synthesis. Regarding keywords-to-text synthesis, there is some work found about text generators based on keywords [9–11], where the meaning of the keywords or a categorization of them is necessary for their appropriate mapping to the full texts. Also, in those works the input of the text generator is either too complex for manual assignment [9, 10] or domain-specific [11]. In this paper the mapping model has been defined with the help of neural network (especially RNN) training, so that the manual specification of the meaning, category or part of speech of the input keywords is not necessary. In our methodology the input used for text synthesis is very simple; just a sequence of keywords, from the domain used in training, which may be any desired. So, a user may benefit from the provided tool even by assigning the input manually. Also, it is experimentally observed that our goal cannot be addressed by a standard encoder-decoder sequence-to-sequence architecture [1, 2, 12], which is too complex for our problem and not well-suited enough to it. (This is discussed more extensively in Sect. 3.5.) Other pieces of work on RNNs (some of which regard NLP) appear in [1].

3 Preprocessing Methodology

Apparently, the first step of the experiment is the definition of the data set. It should consist of texts of similar content, for the sake of sufficient training and appropriate evaluation. A family of steak recipes is constructed according to [13] and considered as such a data set, but any other content could have been selected. The used texts are 15, with a total length of 761 words. The first 9 of them are treated as the training set and the rest 6 as the test set. Two examples are shown below. Observe similarities and dissimilarities.

- “Mix garlic and oil in a bowl. Pour marinade into a resealable plastic bag over the steaks. Later squeeze excess air and seal bag. Afterwards marinate beef in the refrigerator for 4 h. Preheat grill for medium-high heat and bribe grate. Then remove meat from the marinade and shake off excess. Later discard marinade and afterwards cook steaks on preheated oven to desired degree for about 8 min.”
- “In a bowl combine garlic, oil, sauce and sugar. Pour marinade into a resealable plastic bag over the beef. Later in the fridge marinate meat for 8 h. Preheat grill for high heat and then bake steaks to desired degree for about 60 min.”

In the following, the dot and the comma are considered as separate words and all letters are treated as capitals.

In the rest of this section all preprocessing steps, included also in Fig. 1, are described in detail, as executed for our recipes data set. As mentioned above, first the verbs and nouns from all texts are extracted with POS tagging [14]. Afterwards, synonyms are detected and unified using an appropriate library containing a dictionary [15], and then some of the remaining verbs/nouns are automatically selected as keywords. Later, according to the extracted keywords, the texts are (also automatically) separated into chunks. Finally, the data are reformulated in a form acceptable by the network, which is taught using the training set. Our trained model is evaluated using mainly the test set.

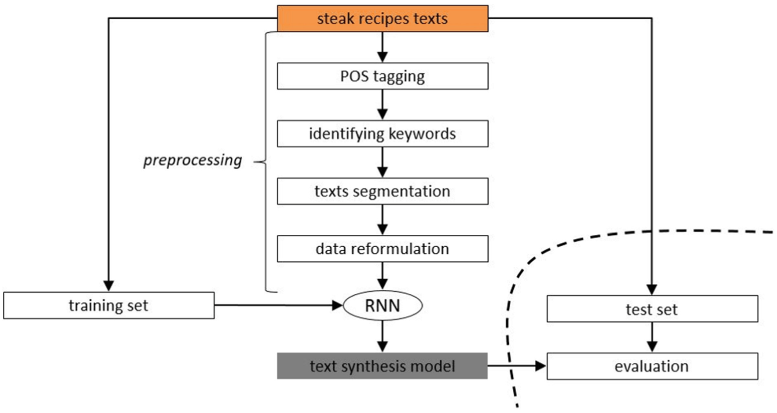


Fig. 1. Architecture of our work

3.1 Detecting Verbs and Nouns with Parts of Speech Tagging

In any of the employed texts the predicted nouns and verbs by the POS tagging algorithm (which was already available in [14]) plus the adjectives "high", "medium-high" and the numerical adjectives are initially treated as keywords. The participles with suffix "-ing" or "-ed" and the indefinite article "a" are prohibited from being considered as keywords, because it has been observed that they do not play a key role in a sentence. Therefore, no stemming by unifying words with the same prefix has been performed.

3.2 Unifying Synonyms

The elimination of the vocabulary to be used in the RNN not only may reduce the number of its parameters (weights), but it also decreases the size of the training set that is required for satisfactory results. Furthermore, by reducing the percentage of full text words that are considered as keywords, the mapping tool is rendered smarter, since less words suffice for the full text synthesis. Thus, only one word is attempted to be used for every group of synonyms, both in input and output/target. (The output is the prediction of the target.) The automatic identification of synonyms for the sake of training is subject to a library [15], which consists of a language dictionary providing meanings, translations, synonyms and antonyms of words. In the training scope two words are initially defined as synonyms when at least one of them is proposed by the library as one of the (up to 5) synonyms of the other one. However, this procedure leads to several wrong matches, so then the user is proposed to confirm which of the suggested pairs of words (s)he desires to be considered as synonyms (something subjective and dependent on the context, which is not taken into account by the library). This manual verification is practical only for a small training text corpus. Also, there are pairs of synonyms that are not identified, but this is not faced manually for training. For each finally considered group of synonyms, a representative is automatically selected to be used in training.

3.3 Other Keyword Elimination Measures

The preprocessing steps of the current and the following sections are programmed mainly from scratch, i.e. without employing some package.

For the further decrease of the number of keywords with respect to the sizes of the full texts, further elimination measures are taken for keywords. Particularly, it has been accepted that when two or more verbs or nouns always coexist in the same sentence, then only one of them implies the existence of the others. (An example is obvious in Fig. 2.) So only the first one is considered as keyword. Also, when the presence of a verb/noun always implies the presence of another verb/noun, even without the reverse holding, and the pair of these two words appears at least 5 times (in order to avoid cases of random coexistences), then optionally the second word is not considered as keyword (see also the term "degree of inclusion" in [4]). After this last optional measure, the non-distinct keywords constitute 31.8% of the total number of non-distinct words of the full texts. In case it is not applied, the keyword rate is 47.7%. Of course, the above implications depend on the data set. The remaining keywords are inserted into a single list (which will generate the RNN input during the data reformulation), and the extra keyword "." is introduced to separate adjacent recipes.

3.4 Segmentation of Texts

In the scope of preprocessing and training all words are replaced by their synonym group representative, and they appear in this way in the next figures.

Definition 1. A"chunk" is defined in this paper as a sequence of words (text segment) ending at a keyword and starting at the word after the previous keyword (or, if not any, at the first word).

According to this definition, a chunk may be seen as a part of a full text which is generated by a single keyword.

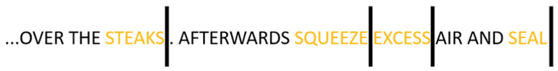


Fig. 2. Segmentation of a part of the first recipe shown in the beginning of the section into chunks according to the extracted keywords (colored). The noun "air" is not considered as keyword, because it always coexists with the verb "squeeze" in the same sentence. The word "excess" is an adjective in this context, but it has been mistakenly recognized as noun.

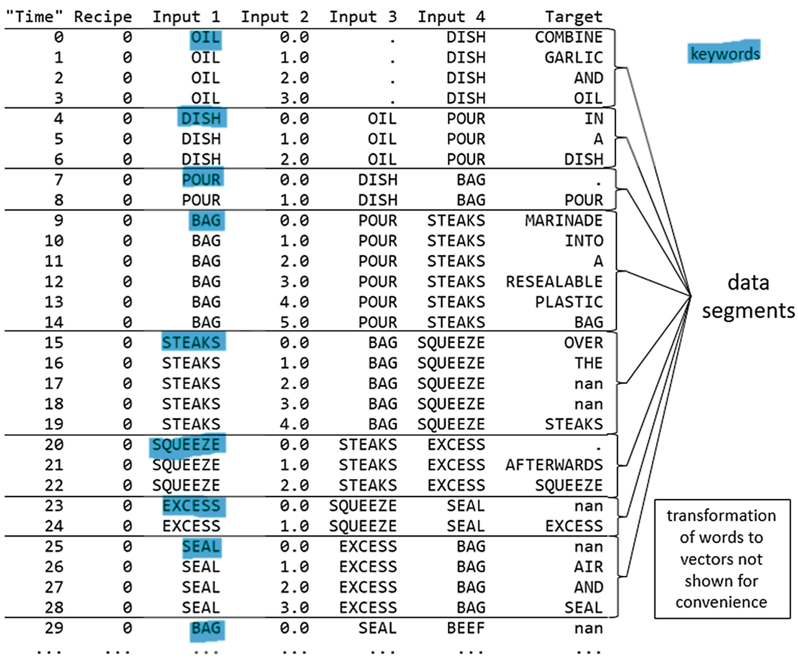
The full texts are divided into chunks. An example is depicted in Fig. 2. Our goal is to teach the RNN to map each input keyword to its chunk (with the additional help of the neighboring keywords and the predicted context), so that the full texts are automatically produced by concatenating the learned chunks.

3.5 Data Reformulation

Another interesting phase is to reformulate the data so that they suit an RNN.

The keywords and the corresponding full texts have to serve as input and output respectively in this work. Sequence-to-sequence models with and without attention [1, 12] were initially employed, but the result was fully unsatisfactory (worse predicted texts than the empty text, according to the measures of Sect. 4.2). A major mistake of such a model was that it was predicting many words multiple times, although these words were related to the respective chunks. This fact indicates that the sequence-to-sequence model is improper for word-to-word mappings, maybe due to the intervention of the context variable between the input and output. Therefore, a more appropriate, tailored idea has been implemented to resolve our problem. The model family used in this paper demands the input and output data sequences to have the same size. However, the keywords of each text are apparently less than all of its words, and also the chunks do not have fixed number of words. The consideration of a chunk as an undivided entity would not be a solution, because then the fact that same words appear in different chunks would not have been taken into account. That is, the output needs to be predicted in the word and not in the chunk level. The above issues are overcome by reformulating the keyword and text sequences, according to the first of the following steps. The second step regards the appending of the previous and the next keyword to each keyword, and the third one converts the words to vectors, so that they can be inserted in the network. All these steps are also executed automatically. The effect of the first two is clear in Fig. 3.

1. All training chunks are stored in groups according to their keyword, and empty words are inserted where needed such that all chunks of the same keyword have the same length, and same words are at the same position of the text segment (so that their alignment with the input is more appropriate). Then, the reformulated chunks compose the extended full texts (with the empty words inserted), which are concatenated, forming an undivided sequence, that will be called "verbal target", since it is the verbal form of the network target sequence. In the RNN input the keywords are replicated in order to fit their chunk length. In our application the texts are artificial, and thus there was the possibility to construct a training set containing all the words of the test set. Auxiliary ordinal keywords are also created, indicating the serial numbers of the positions of each chunk, so that the prediction of every word of a chunk at the correct position is facilitated. Optionally, these numbers are bonded with the keywords, forming new words [e.g. the pair (oil, 2) becomes oil2].
2. As soon as the extended keyword list(s) resulting from the previous step has/have been constructed, two other auxiliary lists of the same length are created for the network's input, informing about the keyword of the previous and the next chunk. This step is based on our observations about the dependence between the current chunk and the neighboring keywords. Its advantage has also been inferred with trials.
3. The words are converted to vectors, so that they can be used by the network. In this step two approaches are examined and compared [1, 2] for the input and/or target words; a. a one-hot and b. a word embedding conversion.
   a. In the first one, the words are converted to one-hot vectors, (i.e. vectors with all their entries equal to 0 except one, which equals 1). This is applied for every



"Time"	Recipe	Input 1	Input 2	Input 3	Input 4	Target
0	0	OIL	0.0	.	DISH	COMBINE
1	0	OIL	1.0	.	DISH	GARLIC
2	0	OIL	2.0	.	DISH	AND
3	0	OIL	3.0	.	DISH	OIL
4	0	DISH	0.0	OIL	POUR	IN
5	0	DISH	1.0	OIL	POUR	A
6	0	DISH	2.0	OIL	POUR	DISH
7	0	POUR	0.0	DISH	BAG	.
8	0	POUR	1.0	DISH	BAG	POUR
9	0	BAG	0.0	POUR	STEAKS	MARINADE
10	0	BAG	1.0	POUR	STEAKS	INTO
11	0	BAG	2.0	POUR	STEAKS	A
12	0	BAG	3.0	POUR	STEAKS	RESEALABLE
13	0	BAG	4.0	POUR	STEAKS	PLASTIC
14	0	BAG	5.0	POUR	STEAKS	BAG
15	0	STEAKS	0.0	BAG	SQUEEZE	OVER
16	0	STEAKS	1.0	BAG	SQUEEZE	THE
17	0	STEAKS	2.0	BAG	SQUEEZE	nan
18	0	STEAKS	3.0	BAG	SQUEEZE	nan
19	0	STEAKS	4.0	BAG	SQUEEZE	STEAKS
20	0	SQUEEZE	0.0	STEAKS	EXCESS	.
21	0	SQUEEZE	1.0	STEAKS	EXCESS	AFTERWARDS
22	0	SQUEEZE	2.0	STEAKS	EXCESS	SQUEEZE
23	0	EXCESS	0.0	SQUEEZE	SEAL	nan
24	0	EXCESS	1.0	SQUEEZE	SEAL	EXCESS
25	0	SEAL	0.0	EXCESS	BAG	nan
26	0	SEAL	1.0	EXCESS	BAG	AIR
27	0	SEAL	2.0	EXCESS	BAG	AND
28	0	SEAL	3.0	EXCESS	BAG	SEAL
29	0	BAG	0.0	SEAL	BEEF	nan
...	...	...	...	...	...	...

keywords
data segments
transformation of words to vectors not shown for convenience

Fig. 3. Data reformulation corresponding to the first part of the first recipe shown in the beginning of the section. First auxiliary words are added to both the target and the first input column, so that they have the same length. (Empty words are represented as "nan".) Then, the serial numbers of the positions of the data segments defined by the chunks are appended as the second input column, and finally two other input columns are created (3, 4), giving information about the precedent and the upcoming keyword.

column (i.e. list) separately. Only the empty word is represented as zero vector instead of one-hot.

b. In the second approach, the words are represented as vectors of floats instead of one-hot vectors, as obtained by a word embedding (word-to-vector) algorithm [16]. In this way, not only the dimension of vectors may be reduced in comparison with the one-hot conversion, but also the distance of representations of all word pairs is determined by the degree of their context relevance. For the training of the word embedding model, the chunks of the training recipes have been used (since the training set contains all the words of the test set). Another option was to use a pre-trained word embedding model. This was examined with GloVe, which is considered as the most preferred by NLP practitioners [16]. As expected, the text synthesis results were remarkably worse, since the word-to-vector training was not based on the domain context. In case the keywords of the first input columns are not bonded with the serial numbers, every input word is also an output word, so word embedding may be applied both to the whole input and to the output/target.

With the above reformulation steps, the sequences of the RNN input ($\boldsymbol{x}(t)$) and target ($\boldsymbol{y}(t)$) are created. That is, $\boldsymbol{y}(t)$ is the vector representation of the (t + 1)-th word of the verbal target sequence (assuming that t starts from 0), and $\boldsymbol{x}(t)$ is the concatenation of the representations of the (t + 1)-th words of the input sequences. Although the argument t does not really stand for time in our case, it will be mentioned as such, because the data are sequential. The examined RNN predictive models belong to the family

$$\boldsymbol{y}_{\text{pred}}(t) = f\left(\{\boldsymbol{x}(t-i)\ :\ i \in\ I\}, \{y_{\text{pred}}(t-j)\ :\ j \in J\}\right), \tag{1}$$

where I, J are finite subsets of {0,1,…}, {1,2,…} respectively. Particular assumptions have been made for the multidimensional function f [17, 18].

4 Recurrent Neural Network Training and Results

After the preprocessing of the data so that they suit the network architecture, training is ready to start.

4.1 Training Details

The Mean Square Error is used as cost function. It is proportional to the Sum of Square Errors, i.e.

$$SSE = \sum\nolimits_{t \in Ttrain} \left[\boldsymbol{y}(t) - \boldsymbol{y}_{\text{pred}}(t)\right]^2 \tag{2}$$

where T_{train} denotes the set of "time points" of the training set.

A 2nd order iterative optimization algorithm (e.g. Levenberg-Marquardt, Broyden-Fletcher-Goldfarb-Shanno) is practical in our problem only in case of sufficient word embedding, due to its usually extreme memory cost (resulting from the Hessian approximations and the big input and output dimensions). Thus, a 1st order algorithm is employed otherwise.

Better results are achieved by setting the initial values of the model parameters to 0 than with random initialization. This is explained rather in the one-hot approach by the sparsity (big percentage of zeros) of the optimal model, which is anticipated due to the one-hot (or zero) encoding of the input and target.

4.2 Evaluation

As follows by the above, this work aims to resolve a classification problem using a regression model, so the continuous output has to be converted to a class (a word of the target vocabulary in our case). In the one-hot approach the output is converted to a one-hot vector by setting the maximum value of the output vector to 1 if the output vector's sum exceeds 0.5 (which indicates that possibly a non-empty word corresponds to that position), and to the zero vector otherwise. In the word embedding approach the output is converted to the closest vector representation of the target vocabulary.

The modified output is then compared to the target for the sake of evaluation and text production.

For vocabulary diversity in predictions, each predicted word is randomly mapped to one of its synonyms or itself. For the sake of evaluation, the pairs of synonyms that have not been identified by the library have been manually defined as synonyms.

For the evaluation of the model's performance, the following measures are used for each of the training and the test set. Both aim at quantifying the quality/correctness of the predicted texts, which are compared to the ground truth (i.e. actual) ones. However, these metrics do not consider the case of multiple correct syntaxes of a sentence, so they may admit a bit worse quality than real.

- Mean Word Error Rate (MWER): The widely known Word Error Rate (WER), common for NLP problems, is computed for each text (recipe) separately and the mean is taken as the evaluation measure. The WER equals the also well known Levenshtein distance between the ground truth and the predicted text divided by the number of ground truth non-empty text words.
- Mistake Rate (MR): This is a simple, heuristic measure similar to WER, but it never rewards the prediction of the correct word in the false position, even if the non-empty words are predicted in the correct order. It is the number of wrong predictions (in terms of time) for all texts divided by the number of ground truth non-empty words of all texts. A mistake is considered as single when an empty word is predicted as non-empty or vice versa, and as double when a non-empty word is predicted as another non-empty word. Apparently, the empty predicted text corresponds to an error of 1, as in the WER case.

4.3 Selecting Recurrent Neural Network Mapping Model

With intuitive trial and error and after search of several shallow and deep architectures (including also LSTM and GRU [17]), there is a general conclusion that one of the best models is the linear simple RNN with input delay 0 and output (direct) delays from 1 to 5, i.e. the shallow model

$$\boldsymbol{y}_{\text{pred}}(t) = \boldsymbol{U}\boldsymbol{x}(t) + \sum\nolimits_{j=1}^{5} \boldsymbol{V}_j \boldsymbol{y}_{\text{pred}}(t-j) + \boldsymbol{b}, \tag{3}$$

where $\boldsymbol{U}$, $\boldsymbol{V}_j$, $\boldsymbol{b}$ contain the weights. (The first 5 predictions are set as equal to the target values.) This intuition results from the fact that a word of a full text is dependent on the present and the neighboring keywords, as well as a few previous full text words. The use of even one hidden layer has proven to be completely helpless, since it leads to training and test errors comparable with those of the empty text. Consequently, the best architecture is found very easily. As for shallow gated RNNs, LSTM has comparable, but often slightly poorer performance, whereas GRU is even worse. Maybe LSTM would be the best if the average chunk length were higher (or, in other words, if the keyword rate were smaller).

4.4 Results and Discussion

In Table 1 the results of training the selected simple RNN of (3) [18] are summarized and compared to the best of LSTM [17] (shallow, stateful network with dropout and recurrent dropout equal to 0.01 and no feedforward activation), according to all possible approaches discussed above. In each case, one of the best numbers of iterations has been chosen. The optimization time of these executions ranges from a few seconds to a few minutes, and depends on the RNN type and dimensionality, the optimizer and the chosen number of iterations. It is worth mentioning that there are 66 distinct non-empty full text words after the unification of synonyms among the initial 73 distinct non-empty full text words. Word embedding, especially on the output/target, reduces the number of weights significantly. For example, all shallow simple RNNs have $O(n^2 + nm)$ parameters, where m, n are the input and the output/target dimensions respectively. This would be particularly important in case that more (and more complex) texts were used, where the need to save memory and computational time would emerge.

Table 1. Evaluation of best RNNs with test errors. IE/OE = Input/Output Embedding (+size), ICB = Input Columns 1 & 2 bonded, SRNN = Simple RNN with direct delays of orders from 1 to 5. The percentages in the headings denote keyword rate.

IE	OE	ICB	MWER 31.8% SRNN	MWER 31.8% LSTM	MWER 47.7% SRNN	MWER 47.7% LSTM	MR 31.8% SRNN	MR 31.8% LSTM	MR 47.7% SRNN	MR 47.7% LSTM
10	10	no	0.608	0.640	0.375	0.410	0.924	0.892	0.605	0.680
no	10	no	0.697	0.619	0.285	0.351	1.042	0.917	0.483	0.573
no	10	yes	0.319	0.311	0.111	0.168	0.378	0.373	0.203	0.220
20	20	no	0.354	0.489	0.262	0.342	0.479	0.627	0.407	0.515
no	20	no	0.378	0.501	0.233	0.325	0.484	0.651	0.400	0.494
no	20	yes	0.239	0.273	0.106	0.163	0.265	0.295	0.186	0.195
10	no	no	0.340	0.403	0.157	0.211	0.423	0.427	0.233	0.203
no	no	no	0.295	0.461	0.060	0.212	0.357	0.477	0.075	0.199
no	no	yes	0.229	0.335	0.017	0.069	0.232	0.357	0.017	0.071

A general remark is that the results worsen as the problem requirements (in terms of keyword rate -resulting from Sect. 3.3- and number of weights) strengthen. It can be subjectively inferred that when the keyword rate is 31.8% (which is the most useful and challenging scenario), the best choice is to apply embedding only on the output and not on the input, so that the first two input columns can be bonded (6th row of the table).

Word embedding training with the skip-gram instead of Continuous-Bag-Of-Words algorithm and use of hierarchical softmax yield the best results here.

The results with a Mean WER of about 0.3 can be characterized as quite satisfactory, because the meaning of the text is almost understandable. For an illustrative example, the prediction. “Preheated oven bake meat to desired extent for about 30 min.” of the text “Preheat oven and then bake beef to desired extent for about 30 min.” has WER = 0.29.

The variety of the keywords' order in some input sequence among recipes not only does not cause significant problem in training, but also results to an aesthetically appealing variety in the chunks order, which changes correspondingly.

5 Conclusion and Future Work

In this work a tailored and quite simple algorithm which, with a sequence of known keywords as input, is able to produce full text containing these keywords, or synonyms of them, in the same order, has been developed. This automatic mapping relies on an RNN model, based on some training texts, from which the keywords are automatically extracted with the help of a POS tagging algorithm and a dictionary library. The results, which depend on the problem difficulty, are rather satisfactory, because the meaning of the predicted text is almost clear.

So far it seems that the proposed methodology is applicable to any domain/vocabulary, although the used texts have to be quite similar (maybe after deliberate editing), especially if they are few. A basic limitation is that when the test set contains words neither included in the training set nor synonymous of a training word, then the word embedding may be based only on a pre-trained word-to-vector model, which is then necessary to determine the chunk length of a new keyword as that of a training neighboring one. The new words need to belong to the domain of the training texts for effective training as well.

The text synthesis work done for steak recipes is going to be repeated for a data set from another domain, containing much more distinct and non-distinct words. By increasing the amount and diversity of texts, it will be interesting to examine the necessary word embedding size to yield fairly good results. Furthermore, there is an ambition that our program will be useful for everyday applications, like news production.

Acknowledgements. This work has been partially supported by the European Commission through project Scan4Reco funded by the European Union Horizon 2020 programme under Grant Agreement No. 665091.

References

1. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016). http://www.deeplearningbook.org
2. Goldberg, Y.: A Primer on neural network models for natural language processing. J. Artif. Intell. Res. **57**, 345–420 (2016). https://www.jair.org/media/4992/live-4992-9623-jair.pdf
3. Sutskever, I., Martens, J., Hinton, J.: Generating text with recurrent neural networks. In: Proceedings of the 28th International Conference on Machine Learning, Bellevue, WA, USA (2011). http://machinelearning.wustl.edu/mlpapers/paper_files/ICML2011Sutskever_524.pdf
4. Shah, P., Perez-Iratxeta, C., Andrade, M.: Information extraction from full text scientific articles: where are the keywords? BMC Bioinform. BioMed Central **4**, 20 (2003). https://doi.org/10.1186/1471-2105-4-20

5. Andrade, M.A., Valencia, A.: Automatic extraction of keywords from scientific text: application to the knowledge domain of protein families. Bioinformatics **14**(7), 600–607 (1998). https://doi.org/10.1093/bioinformatics/14.7.600
6. HaCohen-Kerner, Y.: Automatic extraction of keywords from abstracts. In: Palade, V., Howlett, R.J., Jain, L. (eds.) KES 2003. LNCS (LNAI), vol. 2773, pp. 843–849. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45224-9_112
7. Litvak, M., Last, M.: Graph-based keyword extraction for single-document summarization (2008). https://dl.acm.org/citation.cfm?id=1613178
8. Al-Hashemi, R.: Text summarization extraction system (TSES) using extracted keywords. Int. Arab J. e-Technol. **1**(4), 164–168 (2010). https://core.ac.uk/download/pdf/25749874.pdf
9. Kasper, R.: A flexible interface for linking applications to Penman's sentence generator. In: Proceeding HLT 1989, Proceedings of the workshop on Speech and Natural Language, pp. 153–158 (1989) https://doi.org/10.3115/100964.100979
10. Feiner, S., McKeown, K.: Automating the generation of coordinated multimedia explanations. Computer, **24**(10), 33–41 (1991). https://doi.org/10.1109/2.97249. http://ieeexplore.ieee.org/abstract/document/97249/
11. Bernauer, J., Gumrich, K., Kutz, S., Lindner, P., Pretschner, D.P.: An interactive report generator for bone scan studies. In: Proceedings of the Annual Symposium on Computer Application in Medical Care, pp. 858–860 (1991). https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2247652/pdf/procascamc00004-0868.pdf
12. Brownlee, J.: How to develop an encoder-decoder model for sequenceto-sequence prediction in keras. https://machinelearningmastery.com/develop-encoder-decoder-model-sequence-sequence-prediction-keras/
13. allrecipes - Beef Steak Recipes. http://allrecipes.com/recipes/475/meat-and-poultry/beef/steaks/
14. Natural Language Toolkit (NLTK). http://www.nltk.org
15. PyDictionary 1.3.4. https://pypi.python.org/pypi/PyDictionary/1.3.4
16. Brownlee, J.: How to develop word embeddings in python with gensim. https://machinelearningmastery.com/develop-word-embeddings-python-gensim/
17. Keras Documentation - Models - Sequential. https://keras.io/models/sequential/
18. Atabay, D.: pyrenn: a recurrent neural network toolbox for python and matlab. Institute for energy economy and application technology, Technische Universität, München. http://pyrenn.readthedocs.io/en/latest/