Noaman *et al. Hum. Cent. Comput. Inf. Sci.* (2018) 8:12
https://doi.org/10.1186/s13673-018-0133-x

Human-centric Computing and Information Sciences

RESEARCH Open Access

# Enhancing recurrent neural network-based language models by word tokenization

Hatem M. Noaman[1,2*†], Shahenda S. Sarhan[1†] and Mohsen. A. A. Rashwan[3†]

*Correspondence: hnoaman@bsu.edu.eg
†Hatem M. Noaman, Shahenda S. Sarhan and Mohsen. A. A. Rashwan contributed equally to this work
[2] Computer Science Department, Beni-Suef University, Beni-Suef, Egypt
Full list of author information is available at the end of the article

**Abstract**

Different approaches have been used to estimate language models from a given corpus. Recently, researchers have used different neural network architectures to estimate the language models from a given corpus using unsupervised learning neural networks capabilities. Generally, neural networks have demonstrated success compared to conventional n-gram language models. With languages that have a rich morphological system and a huge number of vocabulary words, the major trade-off with neural network language models is the size of the network. This paper presents a recurrent neural network language model based on the tokenization of words into three parts: the prefix, the stem, and the suffix. The proposed model is tested with the English AMI speech recognition dataset and outperforms the baseline n-gram model, the basic recurrent neural network language models (RNNLM) and the GPU-based recurrent neural network language models (CUED-RNNLM) in perplexity and word error rate. The automatic spelling correction accuracy was enhanced by approximately 3.5% for Arabic language misspelling mistakes dataset.

**Keywords:** Recurrent neural networks, Statistical language modeling, Automatic speech recognition

## Introduction

Statistical language models estimate the probability for a given sequence of words. Given a sentence s with n words such as $s = (w_1, w_2 \dots w_n)$, the language model assigns P(s). Statistical language models assess good word sequence estimations based on the sequence probability estimation. Building robust, fast and accurate language models is one of the main factors in the success of building systems such as machine translation systems and automatic speech recognition systems. Statistical language models can be estimated based on various approaches. Classical language models are estimated based on n-gram word sequences such as $P(s) = P(w_1, w_2 \dots w_n) = \prod_{i=1}^{n} P(w_i|w_{i-1})$ and can be approximated based on the Markov concept for shorter contexts (for example, bigram if $n = 2$ or trigram if $n = 3$ and so on). Recent researchers have applied neural networks different architectures to build and estimate language models. The classical feed-forward neural network-based language models have been continuously reporting good results among the traditional n-gram language modeling techniques [1]. It

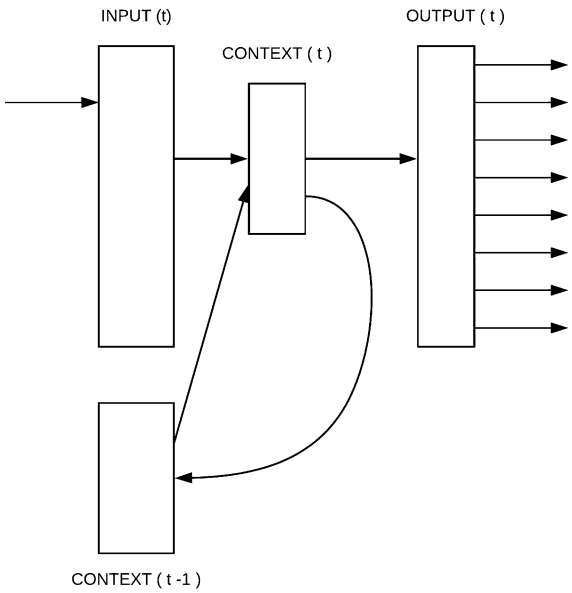


**Fig. 1** Basic recurrent neural network language model. Basic recurrent neural network language model consists of three layers: input layer, hidden layer and output layer; input word is presented to the network input layer using a 1-of-n encoding. The feedback between hidden and input layer allows the hidden neurons to remember the history of the previously processed word. The hidden layer output is computed using Tanh function [2]. The final network output is computed using the Softmax activation function [3]

computes the probability of a word given its context. Inputs into the network are the previous n-words according to the language models order. A word feature vector is projected using the word index in the projection layer. The hidden layer output is computed using Tanh function [2]. The final network output is computed using the Softmax activation function [3] to ensure that network output is a valid probability between 0 and 1. Network weights are updated using the back-propagation training algorithm [4]. Neural network-based language models offer several advantages. Smoothing is applied implicitly, while in n-gram language models, smoothing must be handled explicitly for an unseen n-gram. Semantically, similar words are clustered due to the projection of the entire vocabulary into a small hidden layer [5]. Recurrent neural network-based language models [6] are the other proposed models. In them, the feedback between hidden and input layer allows the hidden neurons to remember the history of the previously processed words, as shown in Fig. 1. The results that are reported using recurrent neural network-based language models are always better than those of basic neural network models and other traditional model results. The results reported by this work show that recurrent neural network-based language models results outperform traditional models results for two different tasks: English automatic speech recognition and Arabic automatic spelling error correction. The neural network input vector is presented using a binary representation where the current word is set to one in the vector according to its index in the vocabulary, and the other vector values are set to zero. The main problem with this model is the size of both the input and output layer where it needs to be at least as large as the number of words in the language vocabulary. The vocabulary size varies from languages such as English that have simple morphological systems to languages that have richer morphological systems such as the Arabic language [7] (where every

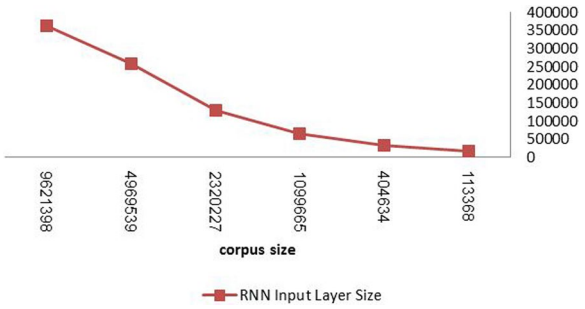


**Fig. 2** Effect of corpus size on the network input layer neurons number. Recurrent neural network-based language models input layer size depends on the vocabulary words number, training time and memory cost increased remarkably as the number of neurons in input and output layers becomes big number. This figure illustrates the relationship between corpus size and the recurrent neural network input layer size, as shown as we add more words to the corpus it will cause vocabulary expansion and as a result recurrent neural network input layer size will increase which will downgrade the network performance.

single root can be converted to several other valid forms using morphological derivation rules). Rich morphological system makes this architecture inefficient if it is used in rich morphological language applications since it needs high computational costs to train and use these models within real-time applications, such as speech recognition and automatic machine translation.

In this work, we try to build recurrent neural network-based language models that can handle the network training speed problem with languages that have rich morphological systems based on word tokenization. The proposed model input layer size is shown in Fig. 2 for the different training sets extracted from an open source Arabic language corpus [8]. As shown in the figure, the network size remarkably increases as the number of words in the training corpus increases. In real applications, to build a robust language model, a large corpus with several millions of words is needed. Networks with a very large number of neurons have two main problems. The first one is that training time is very long to achieve network learning convergence. It is reported that the network learning process takes several weeks with a corpus size of approximately 6.4 M words [6].

The second problem is the memory size. Recurrent neural networks-based language models with large numbers of neurons are expected to need larger memory sizes than that of other traditional language models. Researchers tried to find solutions to these problems through merging all words that occur less than a given threshold into a special rare token or by adding classes of neurons in the output layer and factorizing the output layer into classes [9]. The main contribution of this work is building a recurrent neural network language modeling model that outperforms the basic RNNLM [6]. It is faster and consumes less memory than the RNNLM and its enhanced versions (the factored recurrent neural network language model (fRNNLM) [10] and the CUED-RNNLM [11]). It also adds word features implicitly with no need to add a different vector for each word, as proposed by the related work in the fRNNLM [10] or the FNLM [12]. These features make the proposed model suitable for highly inflected languages and in building models with dynamic vocabulary expansion. It also decreases the number of vocabulary words since unseen words can be inferred from other seen words if the words have the same stem. This paper is organized as follows. "Related work" section includes an

overview of the related works. In "Proposed model" section , the researchers discuss the word tokenization process for English and Arabic then the proposed model is presented. The experimental results are discussed in "Experiments and results" section . Finally, the conclusion is presented in "Conclusion" section .

## Related work

Neural networks different architectures have been investigated and applied to language model estimations by many researchers. Feed forward neural networks [1] have been adapted in language modeling estimation [1]; feed forward neural network language models simultaneously learn the probability functions for word sequences and build the distributed representation for individual words, but this model has a drawback in that a fixed number of words can be considered as a context window for the current or target word. To enhance the conventional feed forward neural network language models training time, researchers proposed continuous space language modeling (CSLM), which is a modular open-source toolkit of feed forward neural network language models [13]; this model introduces support for GPU cards that enable neural networks to build models with corpora that contain more than five billion words in less than 24 hours with about a 20% perplexity reduction [13]. Recurrent neural networks have been applied to estimate language models. However, with this model, there is no need to specify the context window size by using feedback from the hidden to the input layer as a kind of network memory for the word context. Experiment results have proved that recurrent neural networks in language models outperform n-gram language models [5, 6, 9, 14, 15]. An RNNLM toolkit was designed to estimate the class-based language model using recurrent neural networks [5, 6]. It can also provide functions such as an internist model evaluation using perplexity, N-best rescoring and model-based text generation. The training speed is the main RNNLM drawback, especially with large vocabulary sizes and large hidden layers. The RWTHLM [16] is another recurrent neural network-based toolkit with long short-term memory (LSTM) implementation, and the RWTHLM toolkits BLAS library was used to support reduced training time and efficient network training. The CUED-RNNLM [11] provides an implementation for the recurrent neural network-based model, and it has GPU support to achieve a more efficient training speed. Both the basic feed forward network and the recurrent neural network-based language models do not include any type of word level morphological features, but some researchers tried to add this type of word feature explicitly by input layer factorization. Factored neural language models (FNLM) [12] add word features explicitly in the neural network input layer in the feed-forward based neural network language model and the factored recurrent neural network language model (fRNNLM) [10]. They also add word features to the recurrent neural network input layer to model the results better than the basic model. Their complexity is higher than that of the original models since they add word features explicitly to the input layer. While adding these features improves network performance, it adds more complexity to the models estimation and the application performance, especially when applying it to large size vocabulary applications or language with rich morphological features. Researches tries to build RNNLM personalization models [17] using dataset collected from social media networks, model-based RNNLM personalization aims to captures patterns posted by used and his/her related friends while another approach is

feature-based where RNNLM parameters are static throw users. Recently neural-based language modeling models added as an extension to Kaldi automatic speech recognition (Kaldi-RNNLM) [18] software, this architecture combines the use of subword features and one-hot encoding of words with high frequency to handle large vocabularies containing infrequent words. Also Kaldi-RNNLM architecture improves cross-entropy objective function to train unnormalized probabilities. In addition to feed forward network and the recurrent neural network-based language models architectures convolution neural network (CNN) [19] was applied to estimate language models with inputs to the network in the form of character and output predictions is at the word-level.

## Proposed model

The proposed model is a modified version of the basic recurrent neural network language model [6]. Instead of presenting the full word to the network input layer, we split the word into three parts: the prefix, the stem and the suffix. Both the prefix and the suffix may or may not exist. "Word tokenization" section presents a full description about word tokenization and how we implement it with English and Arabic text using modified versions of two free open source stemmers in "English word tokenization" and "Arabic word tokenization" sections. Next, the proposed models architecture is discussed in "The proposed model" section with full details about the model's components, inputs and outputs.

### Word tokenization

The first step to build the word vector is using a stemmer. Generally, in this step, the input to the stemmer is a complete surface word, and the output is the stemmed word vector consisting of a prefix ID, a stem ID and a suffix ID. The general framework of this process is shown in Fig. 3. As illustrated in the figure, the stemmer has 3 lookup tables. The first one is for the prefixes to get the prefix ID using this table. The second one is for the stems. The last one has the suffix IDs. After splitting the word into its composing parts, the stemmer assigns each part a unique ID. If a word does not have a prefix or a suffix, it has the value $-1$ to indicate that this part is not present for the given word. "English word tokenization" section gives a detailed description for the English stemming process. "Arabic word tokenization" section shows that the Arabic language has a more different and richer morphological system than English to prove that this proposed model can effectively handle languages with highly inflected systems.

#### *English word tokenization*

Table 1 presents different examples of the stemmer input and its corresponding output processed by a modified version of PorterStemmer . The original version of this stemmer generates only the word stem. The proposed work needs to tokenize any word into its three components. Therefore, the researchers constructed a list of common English prefixes (ante, anti, co, de, dis, em, en, epi, ex and un) and check if the word starts with any of them. Then, the researchers separate it from the original word. If the remaining part of the word is still a valid English word, the stemmer tries to get the stem from it. Otherwise, it returns back to the original word and tries to stem it directly. After getting the word stem, if it is different from the original word, the stemmer assumes that the word has a suffix and splits the word into the stem and suffix. The final modified

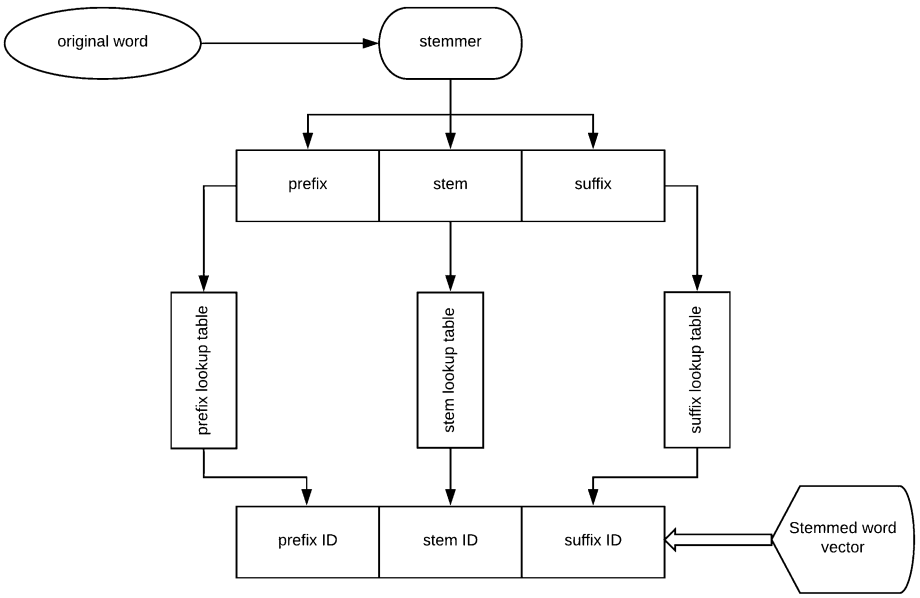


**Fig. 3** Word tokenization process flowchart. The proposed approach uses stemmer to split the word into 3 parts word prefix, word stem and word suffix, the input to the stemmer is a complete surface word, and the output is the stemmed word vector consisting of a prefix ID, a stem ID and a suffix ID. After splitting the word into its composing parts, the stemmer assigns each part a unique ID. If a word does not have a prefix or a suffix, it has the value − 1 to indicate that this part is not present for the given word.

**Table 1 English stemmer original input and output after converting it into prefix, stem and suffix form**

| Input word | Prefix | Stem | Suffix |
|---|---|---|---|
| Years | – | Year | s |
| Government | – | Govern | ment |
| Unless | Un | Less | – |
| Hope | – | Hope | – |

stemmer output always provides the word in the form of (prefix + stem + suffix). At the end, every part has a unique ID so that it can be used in the neural networks internal representation.

#### *Arabic word tokenization*

The Arabic morphological system applies different generation rules on a given root to generate a list of words from the same root. Many approaches have been applied to solve tokenization problem in Arabic. This paper investigates applying the Khoja stemmer [20] to handle the Arabic word tokenization problem. Khoja stemmer is a java open source tool that finds the roots for Arabic words. To find the root for any given Arabic word, first, the Khoja stemmer attempts to remove definite articles, prefixes, and suffixes. It then tries to find the word root using a set of Arabic morphological patterns. If the found root is not applicable, it returns the word as is. We have tried to modify the

**Table 2 Arabic stemmer original input and output after scanning words and convert it into prefix + stem + suffix form**

| **Input text** | اقتصاد وأعمال ـ الهاشمي يقاطع مناقصة عقود الطاقة العراقية... |
|---|---|
| Buckwalter transliteration | AqtSAd w?'EmAl-AlhA$my yqATE mnAqSp Eqwd AlTAqp AlErAqyp |
| English translation | Economics and business-Hashemi boycott Iraqi energy contracts tender |
| Stemmer output | اقتصاد و+ أعمال ال+ هاشم +ي ي+ قاطع مناقص +ة عقود ال+ طاق +ة ال+ عراق +ية |

stemmers functionality to fit our needs. The original stemmer produces only one output for any word, the word root. To find the word root, Khoja's algorithm removes any prefix or suffix that is not considered as a part of a word root and compares the rest of the word to the Arabic morphological patterns. If it fits with any given pattern, the word root is then extracted using this pattern. The work in this paper modifies the original Khoja stemmer to generate 3 parts [the words prefix, suffix (if any), and the word stem] by keeping any removed characters from the word if the word roots extracted word is divided into the prefix, stem and suffix. The original Khoja stemmer includes a list of 168 stop words that are removed while preprocessing Arabic words since these words cannot be rooted. In the modified version, the stop words are not removed. Additionally, the word may be normalized by the Khoja stemmer to find its root. In the modified version, the word stem is not normalized since we try to tokenize the word into parts and not find the valid root for this word. Figure 3 depicts the Arabic word tokenization process using the modified Khoga stemmer tool. The input is an Arabic word, and the final output is a stemmed word vector that consists of three parts (a prefix ID, a stem ID and a suffix ID). To construct such a vector, the stemmer tries to get an input word root by removing any possible prefixes or suffixes attached to the original root. Finally, the stemmer tries to extract the root based on different morphological patterns. If the input word cannot be rooted (or there is no prefix or suffix attached to the stem), the prefix ID and suffix ID values are set to $-1$, which indicates that there is no prefix or suffix attached to the stem. The stemmer output may be the same as the input word in another case when the input word cannot be stemmed for some words such as stop words and non-Arabic words (i.e., words from other languages that are written in the Arabic alphabet without having an Arabic root). The proposed stemming algorithm can be used to tokenize a single word or to tokenize a complete paragraph. Table 2 shows the modified stemmer input, its corresponding Buckwalter transliteration [21], the English translation and its final output for a selected paragraph from a policy section in the open corpus [8]. As shown, the prefixes and suffixes don't appear in all words and the stemmer. The stemmer output is the word divided into the prefix that is followed by the (+) Markup, then the stem followed by the (+) Markup, and finally the word suffix.

### The proposed model

The proposed network architecture (as shown in Fig. 4) splits the input layer into three parts: the word prefix part, the word stem part and the word suffix part. Each part corresponds to the word input part where each part of the word is presented to the network using a 1-of-n encoding. The input layer size equals the sum of the hidden layer size,

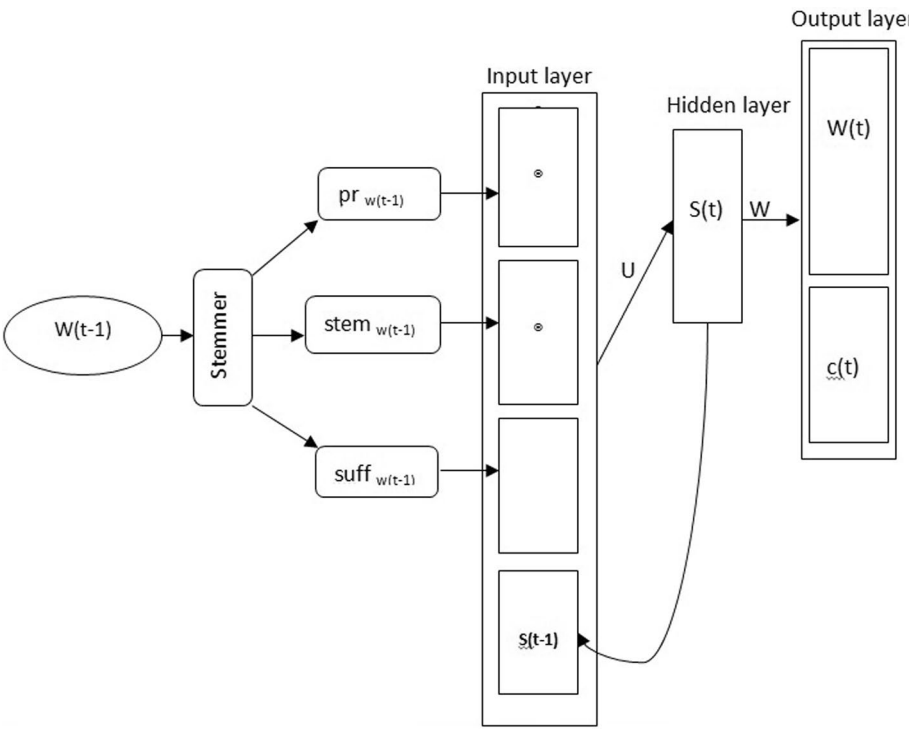


**Fig. 4** The proposed recurrent neural network-based language model architecture with input layer segmented into three components: the prefix, the stem and the suffix. Work in this paper suggests splitting the input layer of the recurrent neural network-based language models to contains three parts of the word; prefix part, the word stem part and the word suffix part, where each part of the word is presented to the network using a 1-of-n encoding.

prefix, stem and suffix counts. The training input vector x(t) is formed by concatenating vector w that is representing the current word and the output from the neurons in context layer s at time $t-1$. The current word is split into three parts. The word vector w is concatenated as expressed in Eq. 1. The prefix, the stem and the suffix vectors are encoded with a 1-of-n coding by setting the ith element to 1 and any other values to 0.

$$w(t) = pr(t) + stem(t) + suf(t) \tag{1}$$

After representing the concatenated word vector to the input neurons, the previous time steps hidden layer is copied to its corresponding history neurons in the input layer, Eq. 2. The hidden and output layers are then computed as shown in Eqs. 3 and 4:

$$x(t) = w(t) + s(t-1) \tag{2}$$

$$s_j(t) = f\left(\sum_i x_i(t) u_{ij}\right) \tag{3}$$

$$y_k(t) = g\left(\sum_j s_j(t) v_{kj}\right) \tag{4}$$

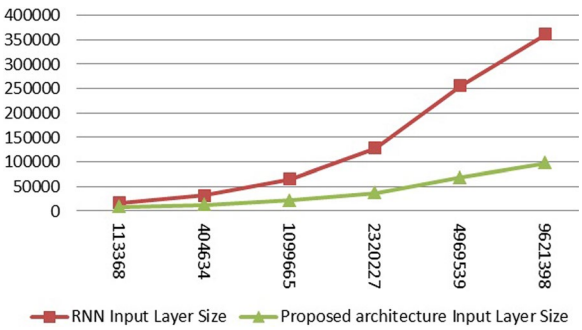


**Fig. 5** Neural network input layer size in the proposed architecture against the basic RNN architecture with increasing size of training corpus. In this figure we compare the number of neurons in the input layer in the proposed architecture with the basic RNN architecture. The figure shows that the input layer size is decreased from 360K to only 98K neurons with nearly 9.6 M words corpus size which is a very efficient architecture for memory usage and processing performance.

where f is the sigmoid function [22] and g are Softmax functions [3]. It is computed as follows:

$$f(z) = \frac{1}{1 + e^{-z}} \tag{5}$$

$$g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}} \tag{6}$$

where u is the weight matrix between the input and hidden layer, and v is the weight matrix between the hidden and output layers.

## Experiments and results

The proposed model is compared with other language modeling approaches based on three different evaluation approaches. The first one is the model computation complexity presented in "Model complexity" section. The second one in "Models perplexity results" section is the proposed model perplexity enhancement and word error rate. The results are shown for English automatic speech recognition using the AMI meeting corpus and the model perplexity enhancement plus the entropy reduction result for the Online Open Source Arabic language corpuss experimental results. Finally, in "Arabic automatic spelling correction application" section, the Arabic automatic spelling error correction results are presented using the Qatar Arabic Language Bank (QALP) corpus.

### Model complexity

The proposed system shows a very efficient memory and processing performance. To prove this, we have compared the neural network input layer size (i.e.: the number of neurons in the input layer) in the proposed architecture with the basic RNN architecture. The results are shown in Fig. 5. It is clear that the number of neurons in the input layer generated by the proposed architecture is always smaller than that in the basic RNN, even with a very limited number of corpus words. This outcome confirms that the tokenization of Arabic words before presenting it to the neural network decreases the

**Table 3 English AMI meeting corpus perplexity and WER results using RNNLM and CUED-RNNLM against our proposed model**

| | PPL | | WER | |
|---|---|---|---|---|
| | *Dev* | *Eval* | *Dev* | *Eval* |
| 3g [25] | 93.6 | 82.8 | 25.2 | 25.4 |
| +CRNN (CE) [25] | 83.3 | 75.2 | 23.9 | 24.1 |
| +FRNN(CE) [25] | 81.0 | 71.7 | 23.9 | 24.0 |
| +FRNN(VR) [25] | 80.4 | 71.6 | 23.9 | 23.9 |
| +FRNN(NCE) [25] | 81.1 | 72.8 | 24.0 | 24.1 |
| +Proposed model | 75.3 | 69.7 | 22.3 | 22.5 |

memory use and computation time. The values presented in Fig. 5 show that the input layer size is decreased from 360K to only 98K neurons.

Complexity of the RNNLM and factored RNNLM models are $(|V| + H) * H + H * (C + |V|)$ and $(|f1| + ... + |fK| + H) * H + H * (C + |V|)$, respectively. While proposed model complexity is $(|pr| + |stem| + |suff| + H) * H + H * (C + |V|)$, where V, H, C fi are the vocabulary count, the hidden layer size, the classes count and the ith feature vector respectively. The researchers observe that the sum of the prefix number and the stems count and the suffixes count $(|pr| + |stem| + |suff|)$ will be much less than vocabulary words count $(|V|)$ especially for highly inflected language and language with rich morphological system. Also proposed model does not need extra GPU processing capabilities as needed with CUED-RNNLM system.

### Models perplexity results

#### *English AMI meeting corpus experiments*

Our first experiment evaluates the proposed model against the n-gram, he RNNLM and the CUED-RNNLM using the English AMI meeting corpus [23]; the corpus consists of 100 annotated hours of meeting records. This dataset has various numbers of features that make it good to be used as a benchmarking dataset for language model evaluation. It is a free of charge dataset that can be used for training and testing. Other good features of this dataset are its annotations that present time and how the annotations are related to the transcription and to each other. This paper divided the English AMI meeting corpus [23] into a training set that consists of 78 h, and the remaining part is used as the development and test sets. Table 3 presents the performances of different language modeling techniques using the AMI meeting corpus. To build an n-gram model, we have used the popular SRILM toolkit [24]; the CRNN and FRNN with the AMI dataset are reported to outperform the basic 3-gram model models that were built using the RNNLM, and the toolkit was used to build the RNNLM [6] and CUED-RNNLM [11]. The proposed language model results are measured by their perplexity and word error rate (WER), as shown in Table 4.

The results show that the proposed token-based recurrent neural network language model has outperformed the n-gram LM by approximately 3% and enhances the basic RNNLM and its GPU version CUED-RNNLM by approximately 1.5% when using the

**Table 4 Perplexity on 70K word as test from Arabic Open Corpus using different smoothing techniques against proposed algorithm**

| Model | Perplexity | Entropy reduction (%) |
|---|---|---|
| GT5 | 113.473 | – |
| KN3 | 99.1785 | 2.85 |
| KN5 | *98.9021* | *2.9* |
| Basic RNN | 70.58 | 10.04 |
| Proposed model | *68.42* | *10.69* |

Italic values represent proposed model perplexity and entropy reduction against different smoothing techniques

English AMI meeting corpus dataset. While the proposed approach is relatively close to the CUED-RNNLMs reported results with the same dataset, the proposed systems training and decoding times are much improved compared to those of the RNNLM and CUED-RNNLM. Moreover, memory consumption is much lower with our proposed model, which make it much more applicable to be used for rescoring tasks rather than for the models generated by the RNNLM (that have no GPU support and have high memory needs) and the CUED-RNNLM (that relies on the GPU architecture and needs more computational resources).

#### *Online Open Source Arabic language corpus experiments*

To evaluate the proposed model, we have chosen the Online Open Source Arabic Language corpus, which is available online. It has an ample amount of words to test and evaluate our proposed model. In this work, the Arabic Open corpus is used as the source of the training, validation and test data. Approximately 1.9 M words are used as training data, and 70K words are used as the test set. To train the models using this dataset, the vocabulary of the 10K most common Arabic words is constructed in the online open source Arabic language corpus. All words outside this vocabulary are counted as out-of-vocabulary (OOV) words and are treated as unknown words (UNK-token). Table 4 shows the results of various language models. To build the N-gram models, the SRILM toolkit [24] and the RNNLM tool [6] are used to build the basic RNN language models. The results have shown that the modified Kneser-Ney smoothing with order 5 (KN5) performs the best among traditional n-gram models. Thus, it was used as the benchmark for our test set. As shown in Table 4, our proposed models perplexity outperforms the baseline n-gram model by up to 30% with about a 2% enhancement compared to basic RNN models.

### Arabic automatic spelling correction application

The automatic spelling correction problem involves two main parts. The first part detects the spelling mistake in the written text. The second corrects the spelling errors [26, 27]. Spelling errors can be identified by using a lexicon of Arabic words. If any given word in the text is outside the lexicon, it is considered as a spelling error. We have built an Arabic automatic spelling correction hybrid system based on the confusion matrix and the noisy channel spelling correction model to detect and automatically correct Arabic spelling errors. It searches for the correct word that can generate this misspelled word (typo) using Eq. (7) [20].

**Table 5 Automatic Arabic spelling correction application**

| Model | Correction accuracy | | | Average (%) |
|---|---|---|---|---|
| | Test set 1 (%) | Test set 2 (%) | Test set 3 (%) | |
| 3-gram+ GT3 | 72.65 | 70.85 | 69.03 | 70.84 |
| RNNLM | 74. 5 | 73.85 | 71.07 | 73.14 |
| Proposed model | 76.03 | 75.36 | 71.50 | 74.30 |

$$\dot{w} = argmax_{word} = P(word|typo) = argmax_{word} P(typo|word) * P(word) \quad (7)$$

where P(typoword) is computed based on a confusion matrix and P(word) using the language model probability of a given word. In this paper, the proposed algorithm was compared with traditional n-gram language models. The proposed algorithm was used to estimate the probability of a given correction candidate for typos. Table 5 shows the overall correction accuracy using the 3-gram language model with Good–Turing smoothing and the proposed algorithm. The results show that the correction accuracy was enhanced by nearly 3.5% using the 4-gram token-based language model where the test set consists of 1500 test cases extracted from the Qatar Arabic Language Bank (QALP) corpus [28]. It was divided into 3 test sets where each one consists of 500 test cases.

## Conclusion

In this paper, we have introduced a modified recurrent neural network-based language model for language modeling. The modification was to segment the network input into three parts. It is observed that the computational complexity is much lower than that in the basic recurrent neural network model. This outcome makes it possible to build language models for highly inflective languages (such as Arabic) from large corpora with smaller training times and memory costs. Using the intrinsic evaluation (perplexity), it is observed that our proposed model outperforms the baseline n-gram model by up to 30% based on the Arabic Open Corpus experimental results shown in Table 4. The results obtained from applying the proposed model to the Arabic automatic spelling correction problem show about a 3.5% total accuracy enhancement. This finding indicates that more complex and advanced Arabic language applications (such as speech recognition and automatic machine translation) can make use of the model described in this paper.

**Authors' contributions**
HN is the corresponding author, and SS and MR are the co-authors. HN has made substantial contributions in the design and implementation of proposed algorithm. SH was involved in drafting the manuscript or critically revising it. All authors read and approved the final manuscript.

**Author details**
[1] Computer Science Department, Mansoura University, Mansoura, Egypt. [2] Computer Science Department, Beni-Suef University, Beni-Suef, Egypt. [3] Electronics and Communications Department, Cairo University, Cairo, Egypt.

**Competing interests**
The authors declare that they have no competing interests.

**Ethics approval and consent to participate**
We confirm that this manuscript has not been published elsewhere and is not under consideration by another journal. All authors have approved the manuscript and agree with its submission.

**Funding**
Not applicable.

## Publisher's Note



Received: 26 January 2018 Accepted: 9 April 2018
Published online: 27 April 2018

**References**

1. Bengio Y, Ducharme R, Vincent P, Jauvin C (2003) A neural probabilistic language model. J Mach Learn Res 3:1137–1155
2. Abramowitz M, Stegun IA (1964) Handbook of mathematical functions: with formulas. Graphs Math Tables 55:83
3. Sutton RS, Barto AG (1998) Reinforcement learning: an introduction. MIT Press Cambridge, Cambridge, p 30
4. Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. Nature 323(6088):533
5. Kombrink S, Mikolov T, Karafiát M, Burget L (2011) Recurrent neural network based language modeling in meeting recognition. In: Twelfth annual conference of the international speech communication association
6. Mikolov T, Karafiát M, Burget L, Černockỳ J, Khudanpur S (2010) Recurrent neural network based language model. In: Eleventh annual conference of the international speech communication association
7. Bousmaha KZ, Rahmouni MK, Kouninef B, Hadrich LB (2016) A hybrid approach for the morpho-lexical disambiguation of arabic. J Inf Proces Syst 12(3):358–380
8. Saad MK, Ashour W (2010) Osac: open source arabic corpora. In: 6th ArchEng international symposiums, EEECS, vol. 10
9. Mikolov T, Kopecky J, Burget L, Glembek O et al (2009) Neural network based language models for highly inflective languages. In: IEEE international conference on acoustics, speech and signal processing. ICASSP 2009, pp 4725–4728
10. Wu Y, Yamamoto H, Lu X, Matsuda S, Hori, C, Kashioka H (2012) Factored recurrent neural network language model in ted lecture transcription. In: International workshop on spoken language translation (IWSLT)
11. Chen X, Liu X, Qian Y, Gales M, Woodland PC (2016) Cued-rnnlman open-source toolkit for efficient training and evaluation of recurrent neural network language models. In: 2016 IEEE international conference on acoustics, speech and signal processing (ICASSP), pp 6000–6004
12. Alexandrescu A, Kirchhoff K (2006) Factored neural language models. In: Proceedings of the human language technology conference of the NAACL, companion volume: short papers. Association for computational linguistics, pp 1–4
13. Schwenk H (2013) Cslm-a modular open-source continuous space language modeling toolkit. In: INTERSPEECH, pp 1198–1202
14. Devlin J, Zbib R, Huang Z, Lamar T, Schwartz R, Makhoul J (2014) Fast and robust neural network joint models for statistical machine translation. In: Proceedings of the 52nd annual meeting of the association for computational linguistics (Vol. 1: long papers), vol. 1, pp 1370–1380
15. De Mulder W, Bethard S, Moens M-F (2015) A survey on the application of recurrent neural networks to statistical language modeling. Comput Speech Lang 30(1):61–98
16. Sundermeyer M, Schlüter R, Ney H (2014) rwthlmthe RWTH Aachen University neural network language modeling toolkit. In: Fifteenth annual conference of the international speech communication association
17. Tseng B-H, Wen T-H (2017) Personalizing recurrent-neural-network-based language model by social network. IEEE/ACM Trans Audio Speech Lang Proces (TASLP) 25(3):519–530
18. Xu H, Li K, Wang Y, Wang J, Kang S, Chen X, Povey D, Khudanpur S (2018) Neural network language modeling with letter-based features and importance sampling. In: 2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)
19. Kim Y, Jernite Y, Sontag D, Rush AM (2016) Character-aware neural language models. In: AAAI, pp 2741–2749
20. Kernighan MD, Church KW, Gale WA (1990) A spelling correction program based on a noisy channel model. In: Proceedings of the 13th conference on computational linguistics. Association for computational linguistics, vol. 2, pp 205–210
21. Buckwalter T (2004) Buckwalter arabic morphological analyzer version 2.0. linguistic data consortium, University of Pennsylvania, 2002. ldc cat alog no.: Ldc2004l02. Technical report, ISBN 1-58563-324-0
22. Han J, Moraga C (1995) The influence of the sigmoid function parameters on the speed of backpropagation learning. In: International workshop on artificial neural networks. Springer, Berlin, pp 195–201
23. Carletta J, Ashby S, Bourban S, Flynn M, Guillemot M, Hain T, Kadlec J, Karaiskos V, Kraaij W, Kronenthal M et al (2005) The ami meeting corpus: a pre-announcement. In: International workshop on machine learning for multimodal interaction. Springer, Berlin, pp 28–39
24. Alumäe T, Kurimo M (2010) Efficient estimation of maximum entropy language models with n-gram features: An srilm extension. In: Eleventh annual conference of the international speech communication association
25. Chen X (2015) Cued rnnlm toolkit
26. Noaman HM, Sarhan SS, Rashwan M (2016) Automatic arabic spelling errors detection and correction based on confusion matrix-noisy channel hybrid system. Egypt Comput Sci J 40(2):2016
27. Attia M, Al-Badrashiny M, Diab M (2014) Gwu-hasp: hybrid arabic spelling and punctuation corrector. In: Proceedings of the EMNLP 2014 workshop on Arabic natural language processing (ANLP), pp 148–154
28. Zaghouani W, Mohit B, Habash N, Obeid O, Tomeh N, Rozovskaya A, Farra N, Alkuhlani S, Oflazer K (2014) Large scale Arabic error annotation: guidelines and framework. In: LREC, pp 2362–2369