Rich Morphology Based N-gram Language Models for Arabic

Ahmad Emami, Imed Zitouni, Lidia Mangu

IBM T.J. Watson Research Center
1101 Kitchawan Rd, Yorktown Heights, NY 10598
{emami, izitouni, mangu}@us.ibm.com

Abstract

In this paper we investigate the use of rich morphology such as word segmentation, part-of-speech tagging and diacritic restoration to improve Arabic language modeling. We enrich the context by performing morphological analysis on the word history. We use neural network models to integrate this additional information, due to their ability to handle long and enriched dependencies. We experimented with models with increasing order of morphological features, starting with Arabic segmentation, and later adding part of speech labels as well as words with restored diacritics. Experiments on Arabic broadcast news and broadcast conversations data showed significant improvements in perplexity, reducing the baseline N-gram and the neural network N-gram model perplexities by 35% and 31% respectively.

Index Terms: Language Modeling, Arabic Morphology, Rich Language Modeling

1. Introduction

Arabic white-space delimited words are usually a composition of multiple segments: prefixes, a stem (i.e., root), and suffixes. We find attached pronouns that are often presented as prefixes or suffixes attached to a stem. As an example, the word تَعَلَّمَ (she meets them) has two parts, تَعَلَّمَ (she) and مِنْهُ (them), referring to two different pronouns. Also, short vowels and other diacritics are omitted in Arabic texts. Exceptions are made for important political and religious texts. Scripts without diacritics have considerable ambiguity because many words with different diacritics appear identical in a diacritic-less setting. Hence, to better handle the ambiguity of such a rich language when building a language model, it is appropriate to take into account these extra sources of information.

The commonly used language models (LMs) are N-gram models where the probability \( P(W = w_1w_2\ldots w_N) \) is computed as a product of the conditional probabilities of each word \( w_i \) given its history of the \( N - 1 \) most recent words. Estimating the probabilities \( P(w_i|w_{i-N+1}\ldots w_{i-1}) \) is inherently difficult even for small values of \( N \) since many word combinations \( w_{i-N+1}\ldots w_i \) are observed only infrequently or not at all. The problem becomes more acute as we deal with languages with rich morphology such as Arabic, where word inflections results in large vocabulary sizes and high out of vocabulary (OOV) rates. We propose in this paper a language modeling approach that uses the neural network language modeling framework to combine diverse and rich sources of morphological information that better estimate the likelihood of \( N \)-gram events in Arabic text. We extend the \( N \)-gram context in three stages: (1) break inflected words into morphological segments and use them to reduce data sparseness problem and to find a better generalization; (2) include part-of-speech (POS) tags in the context which better generalize the model and enable the use of shallow syntactical information; (3) add vowelized or diacritized form of the word history to reduce word ambiguities.

The idea of using rich morphology information in Arabic language modeling has been explored by several researchers. The most common idea has been to use segments, which are the result of breaking inflected word into parts, for better generalization when estimating the probabilities of \( N \)-gram events [1, 2]. Thanks to neural network language modeling framework, our approach benefits from richer set of information such as POS tags and diacritized form of words. Indeed, the use of this information (segments, POS tags and words with diacritics restored) creates richer context which make it very hard, if not impossible, to model with traditional back-off methods. Learning the dependencies in such a long context is daunting even with models such as the factored language models [1] simply due to the sheer number of links that need to be explored. On the other hand, the neural network language modeling framework showed to be a powerful smoothing method and has enabled the use of longer and richer probabilistic dependencies [3]. It uses a distributed representation of words, combined with a neural network for probability estimation. Since these models work in a distributed space, and since function estimation is better understood and solved in distributed (continuous) spaces, it can be assumed that the neural network models are better in generalizing to unseen data. The model size grows at most linearly with the \( N \)-gram order or the vocabulary size, compared to exponential and polynomial growth respectively for regular \( N \)-gram models. It has been shown that this language modeling framework improves performance over state-of-the-art smoothing methods when it is used with standard \( N \)-gram history as well as when it is used in the context of a syntactic based language model [4].

The next three sections of this paper describe how rich morphology information (segments, POS tags and words with diacritics restored) are extracted. We then briefly describe in section 5 how we integrate these resources in the neural network language modeling framework. Section 6 presents the experimental setup and the results, and section 7 concludes the paper.

2. Segmentation

A segmentation process consists of separating white-space delimited words into (hypothesized) prefixes, stems, and suffixes. In this paper we use a weighted finite state transducer (WFST) framework as described in [5]. A finite state machine (FSM) segmenter uses a language independent decoder that takes as input a WFST model and a document to be segmented. The decoder implements a Bellman dynamic programming search for
the most likely solution from the model’s lattice of segmentation hypotheses computed for the sequence of input characters.

We recast the segmentation strategy as the composition of three distinct finite state machines. The first machine is a finite state transducer that encodes the prefix and suffix expansion rules, producing a lattice of possible segmentations. The second machine is dictionary transducer that accepts characters and produces identifiers corresponding to dictionary entries. The final machine is a trigram language model, specifically a Kneser-Ney based back-off language model, that is trained upon the sequence of dictionary entries observed in the training corpus. Using 0.5 Million words from the combined Arabic Treebanks 1V2. 2V2 and 3V1, the dictionary based segmenter achieves an exact word match 98% correct segmentation. This is very good performance considering the kind of segmentation we perform: the segmentation process we use is more than a simple clitic tokenization, since we split the word into zero or more prefixes, followed by a stem (i.e., root) and zero or more suffixes. As an example, the white-space delimited word ṭaḥlibm (she met them) is segmented into three morphs: prefix ṭ (she), followed by stem qAbť (met) and suffix hN (them).

3. Part-of-Speech Tagging

The POS helps language models generalize better and enable the use of shallow syntactical information. For instance, in the sentence “مدرسة المكتب” (translation, “base of the house”), the token “المكتب” is an adjective as identified by its POS. It is also a person name when it appears in another context such as “قائماً” (translation, “Ined said”). In this paper, we run POS classifier on the segmented text (c.f. section 2). The POS classifier uses the Maximum Entropy (MaxEnt) framework to assign a POS label to each segment [6]. MaxEnt has the ability to integrate arbitrary types of information and make a classification decision by aggregating all information available for a given classification [7].

Let \( Y = \{y_1, \ldots, y_n\} \) be the set of POS labels to predict, \( X \) be the example space and \( F = \{0, 1\}^m \) be a feature space. Each segment example \( x \in X \) has associated a vector of binary features \( f(x) = (f_1(x), \ldots, f_m(x)) \). We also have access to a set of training examples together with their classifications: \( \{(x_1, y_1), \ldots, (x_n, y_n)\} \). The MaxEnt algorithm associates a set of weights \( (\beta_j)_{j=1}^{m} \) with the features, which are estimated during the training phase to maximize the likelihood of the data. The model computes the probability distribution over labels for a particular example \( x \) as follows:

\[
P(y_i|x) = \frac{1}{Z(x)} \prod_{j=1}^{m} \beta_{f_j}^{y_i}, \quad Z(x) = \sum_{i=1}^{n} \prod_{j=1}^{m} \beta_{f_j}^{y_i}
\]

where \( Z(X) \) is a normalization factor. To estimate the optimal \( \beta \) values we use the sequential conditional generalized iterative scaling (SCGIS) technique. After computing the class probability distribution, the chosen POS label is the one with the most posteriori probability. The decoding algorithm performs sequence classification, through dynamic programming [6]. Our POS Classifier is trained on the 500K words and 84 POS distinct labels of the Arabic Treebanks 1V2. As features, we limit ourselves to lexical information and POS tags of the four previous words. It achieves an accuracy of about 92%.

4. Diacritic Restoration

Arabic alphabet consists of 28 letters that represent the consonants and long vowels such as ِّ (both pronounced as /a:/), ِّ (pronounced as /i:/), and ِّ (pronounced as /u:/). The short vowels and certain other phonetic information such as consonant doubling (shadda) are not represented by letters, but by diacritics. A diacritic is a short stroke placed above or below the consonant. Those strokes are omitted in Arabic texts and restoring them will remove ambiguity, because many words with different diacritic patterns appear identical in a diacritic-less setting.

We split the Arabic diacritics into three sets: short vowels, doubled case endings, and syllabification marks. Short vowels are written as symbols either above or below the letter in text with diacritics, and dropped altogether in text without diacritics. We find three short vowels: fatha (/i/), damma (/u/) and kasra (/l/). The doubled case ending diacritics are vowels used at the end of the words; the term “tanween” is used to express this phenomenon. Similar to short vowels, there are three different diacritics for tanween: tanween al-fatha, tanween al-damma, and tanween al-kasra. They are placed on the last letter of the word and have the phonetic effect of placing an “N” at the end of the word. Text with diacritics contains also two syllabification marks: shadda to denote the doubling of the consonant and sukaam to indicate that the letter does not contain vowels.

Native speakers of Arabic are able, in most cases, to accurately vocalize words in text based on their context, the speaker’s knowledge of the grammar, and the lexicon of Arabic. Our goal is to convert knowledge used by native speakers into features and incorporate them into a MaxEnt framework. In this paper, we use the state-of-the-art diacritic restoration approach that we published in [7]. It is based on the MaxEnt framework because of its ability to integrate diverse types of information when making a classification decision [7]. We use features such as words, segments, individual characters, and POS tags of the history words. We also use the three previously restored diacritics as additional information. Using the publicly available LDC’s Arabic Treebank Part 3 corpus for training, we achieve a diacritic error rate of 5.1% and a word error rate of 17.3%.

5. Neural Network Model

In a neural network based language model words are represented by points in a continuous multi-dimensional feature space and the probability of a sequence of words is computed by means of a neural network. The feature vectors of the preceding words make up the input to the neural network, which then will produce a probability distribution over a given vocabulary [3, 4]. The main idea behind this model is to make the estimation task easier by mapping words from the original high-dimensional discrete space to a low-dimensional continuous space where probability distributions are smooth functions in their variables.

Suppose the goal is to compute the probability of a certain event \( Y = y \) given the values \( X = x_1, x_2, \ldots, x_m \) of \( m \) conditioning variables. The conditional probability function \( P(y|x_1, x_2, \ldots, x_m) \) is determined in two parts:

1. A mapping that associates a real vector of fixed dimension with each token in the input vocabulary \( V_i \): the set of all tokens that can be used for prediction.
2. A function which takes as the input the concatenation of the feature vectors of the input items \( x_1, x_2, \ldots, x_m \).
The function produces a conditional probability distribution (a vector) over the output vocabulary $V_o$: the set of all tokens to be predicted. The output vocabularies $V_i$ and $V_o$ are independent of each other and can be completely different. Usually used as an N-gram model, the model’s input variables $X_i$ are limited to the $N - 1$ previous words in the history. In this paper, we show the benefit of using additional variables such as the segments of the $N - 1$ previous words and their POS tags. We also explore the use of the $N - 1$ previous words with their diacritics restored. This will let us use richer morphology resulting in better LMs. One great advantage of this model is that context length (number of inputs) can be increased resulting in at most linear increase in model size, in contrast to exponential growth for regular N-gram models. This makes the neural network a very suitable model for capturing longer and richer probabilistic dependencies.

Training is achieved by searching for parameters $\Phi$ of the neural network and the values of feature vectors that maximize the penalized log-likelihood of the training corpus:

$$L = \frac{1}{T} \sum_{t} \log P(y_t|x_1^t, ..., x_m^t; \Phi) - R(\Phi) \quad (1)$$

where superscript $t$ denotes the $t^{th}$ event in the training data, $T$ is the training data size and $R(\Phi)$ is a regularization term, which in our case is a factor of the L2 norm squared of the hidden and output layer weights.

Figure 1: The neural network architecture

For every input $x_1^t, x_2^t, \cdots, x_m^t$, the model produces probabilities $P(y_t|x_1^t, ..., x_m^t)$ for all tokens $y$ in $V_o$. The neural network weights and biases, as well as the input feature vectors, are learned simultaneously using stochastic gradient descent training via back-propagation algorithm, with the objective function being the one given in Equation 1. The computational complexity of the neural network language model is very high since it requires normalization over all the words in the output vocabulary. This is addressed by limiting the output vocabulary to a subset of high frequency words and thus effectively reducing its size [8]. Other methods in speeding up the model were the bunch training [8] and parallelization of the output later over multiple machines [4].

6. Experiments

Experiments are performed on transcripts of Arabic audio data (Broadcast News and Broadcast Conversations) released by LDC. For LM purposes, the training set contains about 7 million words and 245K unique words. Test perplexity (PP) results are reported on a held-out set of 80K words from the acoustic transcripts released by LDC. The neural network used here is a 6-gram model with 100 hidden units and with an output vocabulary limited to the 20K most frequent words of the training data. This was the optimal configuration found in [9]. Using the 20K vocabulary, the out-of-vocabulary rate is about 10.8% for the training set and 13.8% for the test set.

After segmentation, a word is composed of zero or more prefixes, one stem, and zero or more suffixes. Since the context in the neural network needs to be of a fixed length, we process the text so that a word is the concatenation of a single prefix, a stem, and a single suffix: affixes are concatenated if more than one, and are replaced by the NULL token if empty. To evaluate the impact of morphology in improving LMs, we expand the $N - 1$ words in the history (5 previous words in our case) with their segments and POS labels: each word results in additional 3 segments and 3 POS labels. This expansion history can result in a very large effective ngram orders: for a 6-gram model that uses segments and POS labels, we have $5 \times 3 + 5 \times 3 = 30$ tokens in the history and so it becomes effectively a 31-gram model.

Table 1 shows effective ngram order (i.e., number of tokens in the history) and perplexity values for several LMs. The perplexities in this table are computed over the output vocabulary (20K most frequent words) and so any word outside this vocabulary is ignored. The perplexities computed in this manner are of course highly underestimated (lower values) due to the high frequency nature of the output vocab, but are still useful in comparing our different neural net LMs to each other. We denote by NN the baseline model using only words; NN-seg the model using only segments; NN-seg+word the model using words and segments; NN-pos+seg+word the model using words, segments and POS; NN-vowel+pos+seg+word the model using words, segments, POS and restored diacritics; intp-NN the linear interpolation (with uniform weights) of all previous models. As expected, results show that LM performance consistently and incrementally improves as additional context are used. When segments are added to the word based model perplexity is reduced by 12% (383 vs. 433). We notice 23% improvement in terms of perplexity (334 vs. 433) when segments, POS labels and restored diacritics are added. An additional 10% improvement (300 vs. 334) is obtained when we linearly interpolate the different neural network models. Also, it is important to note that unlike traditional N-gram LMs, the neural network model is effectively able to generalize and handle large context (i.e., $41 = 5(\text{words}) + 5 \times 3(\text{segments}) + 5 \times 3(\text{POS}) + 5(\text{diacritics on words}) + 1(\text{word to predict})$).

<table>
<thead>
<tr>
<th>Model</th>
<th>effective order</th>
<th>PP</th>
</tr>
</thead>
<tbody>
<tr>
<td>NN</td>
<td>6</td>
<td>433</td>
</tr>
<tr>
<td>NN-seg</td>
<td>16</td>
<td>406</td>
</tr>
<tr>
<td>NN-seg+word</td>
<td>21</td>
<td>383</td>
</tr>
<tr>
<td>NN-pos+seg+word</td>
<td>36</td>
<td>342</td>
</tr>
<tr>
<td>NN-vowel+pos+seg+word</td>
<td>41</td>
<td>334</td>
</tr>
<tr>
<td>intp-NN</td>
<td>-</td>
<td>300</td>
</tr>
</tbody>
</table>

Table 1: Perplexity of 6-gram neural network model with different morphological information.

Table 2 shows the true perplexities (over all vocabulary) of our models. Also included are the perplexities of two modified Kneser-Ney models: a small model trained on our training set of 7 million words and a big model trained on a much larger data (more than 700 million words) which includes the training set. In order to extend the output distribution of the neural net model to all the vocabulary (245K), it is normalized according to the following formula:

$$P(w_{i|W_{i-N+1}}) = \begin{cases} P_{NN}(w_{i|W_{i-N+1}}) & \text{if } w_i \in V_o; \\ P_{bo}(w_{i|W_{i-N+1}}) & \text{otherwise.} \end{cases}$$

where $P_{NN}()$ and $P_{bo}()$ denotes the neural network and the back-off (substitute) Kneser-Ney models respectively. The term $\alpha()$ is a normalization factor ensuring that the probabilities sum to 1 [8, 4]. The term $V_o$ denotes the 20K neural network output vocabulary. The PP-Small and PP-Big columns of Table 2 refer to perplexity values where the back-off LM ($P_{bo}$) is the small or the large Kneser-Ney model respectively. Table 2 uses the
same annotation in Table 1. In addition, we denote by Kneser-Ney (small) the N-gram model using Kneser-Ney smoothing technique trained on 7M words; Kneser-Ney (big) the N-gram model using Kneser-Ney smoothing technique trained on 700M words; and intp-NN+Kneser-Ney the linear interpolation (with equal weights) of intp-NN model and Kneser-Ney model. N-gram LMs using Kneser-Ney backoff technique achieves optimum performance with $N = 4$ ($P_{1011} = 1563$); no further improvement is obtained with larger value of $N$. When using the neural network framework, we obtain a test perplexity of 1456 with $N = 6$ on the same data set [9]. Results again show that performance consistently and incrementally improves as the number of used morphological resources increase. When the small model is used for $P_{1011}$, perplexity improves by 35% (1011 vs. 1563) when all resources are used. When the big model is used for $P_{1011}$, the use of available resources in neural net and Kneser-Ney improves perplexity by 19% (785 vs. 970). It is important to note that by using only 7 million words for training and additional morphological resources, we obtain a performance close to a model using 700 million words for training (1011 for intp-NN + Kneser-Ney (small) vs. 970 using Kneser-Ney on 700M words), and further improvements are achieved if the two models are interpolated.

<table>
<thead>
<tr>
<th>Model</th>
<th>PP-Small</th>
<th>PP-Big</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kneser-Ney (small)</td>
<td>1563</td>
<td>-</td>
</tr>
<tr>
<td>Kneser-Ney (big)</td>
<td>-</td>
<td>970</td>
</tr>
<tr>
<td>NN</td>
<td>1456</td>
<td>1240</td>
</tr>
<tr>
<td>NN-seg</td>
<td>1378</td>
<td>1173</td>
</tr>
<tr>
<td>NN-seg+word</td>
<td>1309</td>
<td>1115</td>
</tr>
<tr>
<td>NN-pos+seg+word</td>
<td>1189</td>
<td>1012</td>
</tr>
<tr>
<td>NN-vowel+pos+seg+word</td>
<td>1165</td>
<td>992</td>
</tr>
<tr>
<td>intp-NN</td>
<td>1062</td>
<td>904</td>
</tr>
<tr>
<td>intp-NN + Kneser-Ney</td>
<td>1011</td>
<td>785</td>
</tr>
</tbody>
</table>

Table 2: Impact of morphological resources on the perplexity of the neural network model normalized with N-gram model using Kneser-Ney smoothing technique.

We tried our models in re-scoring lattices output by a state-of-the-art automatic speech recognition (ASR) system. The experimental setup is the same as in [9]. Morphological analysis was performed on 6-grams extracted from the lattice. The use of these models did not lead to any noticeable improvement in word error rate (WER), when compared to the regular 6-gram neural network (NN) model (12.0% and 20.1% WERs on two different test sets [9]). One possible reason is that morphological analysis (segmentation, POS tagging and diacritic restoration) of ngrams, due to limited context, is more prone to errors than when performed on sentences. We performed morphological analysis on the training text as well as on 6-grams extracted from the same text. We measured mismatches by 5% and 15% for segmentation and POS tagging respectively. One idea to address this is to expand the lattice and extract longer ngrams in order to have an adequate context length for morphological analysis. Overall we found it surprising not to get any improvements in WER given the soundness of the idea and the significant improvements in perplexity; we are still investigating this as part of future work.

### 7. Conclusion

In this paper we experimented with the use of rich morphology for Arabic language modeling. We extended the word history of 6-gram model with its segmented text, POS tags, and restored diacritics, which for the most detailed model led to a history of 40 tokens. We used this rich set of information in the neural network language modeling framework which has the advantage of efficiently generalizing and learning long and rich dependencies. The use of rich morphology showed significant improvements in perplexity, achieving as much as 35% and 31% reduction in perplexity compared to the baseline Kneser-Ney and baseline N-gram neural network models. A noteworthy observation is that our rich morphological model trained on a small data set is able to improve on the perplexity of a Kneser-Ney LM trained on considerably larger amount of data. We tried using our models to re-score the lattices of a speech recognition system and did not observe a WER improvement compared to the N-gram neural net model. This might be due to the inherent errors in morphological analysis when working on ngrams since there is not enough context for the first items of the ngram. We plan to investigate this matter further and intend to either expand the lattice to generate longer context for morphological analysis or to integrate our models in the decoder directly.

### 8. References


