Accepted for publication in Journal of King Saud University - Computer and Information Sciences, Dec 2020

Classifying and diacritizing Arabic poems using deep recurrent neural networks

Gheith A. Abandah[a,*], Mohammed Z. Khedher[a], Mohammad R. Abdel-Majeed[a], Hamdi M. Mansour[b], Salma F. Hulliel[a], Lara M. Bisharat[a]

[a]*School of Engineering, The University of Jordan, Amman 11942, Jordan*
[b]*School of Arts, The University of Jordan, Amman 11942, Jordan*

Abstract

Poetry has a prominent history in Arabic literature. The classical Arabic poetry has 16 meters that vary in rhythm and target purpose. Chanting a poem eloquently requires knowing the poem's meter and obtaining a diacritized version of its verses (letters inscribed with their short vowels); diacritics are often not inscribed in Arabic texts. This work proposes solutions to classify input Arabic text into the 16 poetry meters and prose. It also investigates the automatic diacritization of Arabic poetry. We adopt machine learning approach using a large dataset of 1,657 k verses of poems and prose to develop neural networks to classify and diacritize Arabic poetry. We propose deep and narrow recurrent neural networks with bidirectional long short-term memory cells for solving these problems. The proposed model classifies the input text with an average accuracy of 97.27%, which is significantly higher than previous work. We also propose a solution that achieves an accuracy that approaches 100% when multiple verses of the same poem are available through predicting the class from the aggregate probabilities of the multiple verses. Diacritizing poetry is much harder than diacritizing prose due to the poet's meticulous selection of phrases and relaxation of some diacritization rules.

Keywords: Arabic poetry; Poem-meter classification; Automatic diacritization; Bidirectional recurrent neural networks; Long short-term memory; Deep learning

1. Introduction

Arabic poetry is a thriving, traditional literature that has roots dating to prior the 6th century. Arabs continue to give great attention to this art and to celebrate gifted poets. Arabic poets compose poems to express emotions, record events, explain ideas, give wisdom, motivate, flirt, praise and defame, and pride and ridicule (Zwettler, 1978). *Arabic Prosody*, the science of poetry, has been developing since the 8th century to study the poetic patterns and meters and to identify sound and broken poetry verses. The classical Arabic poetry has 16 main meters that we introduce in the next section. These meters have differing popularities and are used in composing poems for the various purposes.

Hazem al-Carthajini (d. 684 AH) in the 13th century linked the purpose to the meter and proposed that each meter has a distinct rhythm that fits some purposes (Al-Carthajini, 1966). As the purposes of poetry are various, including seriousness and sobriety, humor and gracefulness, splendor and glorification, and belittling and contempt, the purpose must be matched with the appropriate meter. If the poet intends to be proud, this purpose is matched with a luxurious, shining and sober meter. If the poet intends to be sarcastic or cynical or to insult, such purpose is matched with what suits it from the reckless meters of little splendor. More recently, Al-Tayyib (1989) also linked the poetic purpose with meter. For example, he described the C12-Madīd meter by "It has toughness, brutality and violence suitable for war" and the C13-Hazaj meter by "It has a sweetness and tone that requires a flowing saying that is dominated by a single thought that the poet sings without scrutiny and investigation".

Arabic prosody is regarded a difficult science that has many sophisticated rules and techniques. Although gifted poets compose poetry naturally, others need to apply these rules and techniques to analyze poetry. The task becomes harder when the poem is written without diacritics, which are accents for the Arabic letters and indicate various short vowel sounds. A verse written without diacritics has ambiguous pronunciation. However, a proficient reader can often infer the proper pronunciation given the poem meter and context. Accurate solutions for analyzing poems, identifying meters, and

* Corresponding author. Tel.: +962-79-815-9900; fax: +962-6-53-0813.
E-mail address: abandah@ju.edu.jo.

automatically diacritizing verses are currently unsatisfactory. Such solutions would be very valuable to novice and seasoned poets and to whomever interested to facilitate composing, chanting and enjoying fine poetry.

The main objective of this work is to develop solutions that support analyzing and reading Arabic poetry. In particular, this work tackles the problem of how to accurately distinguish poetry from prose and how to accurately find the meter of a poem from the sequences of characters that constitute its verses. The previous work provides unsatisfactory accuracy and does not distinguish poetry from prose. This work also investigates the accuracy of our previous work in diacritizing poetry. Automatic poetry diacritization is the process of predicting the diacritics of a verse from the sequence of verse letters without diacritics. This investigation is needed because previous work in using machine learning to automatically diacritize Arabic text did not investigate it on Arabic poetry.

The contemporary success of deep learning approaches has recently included solutions for Arabic language processing such as speech recognition and automatic diacritization (Al-Ayyoub *et al.*, 2018). We use in this work deep recurrent neural networks (RNN) trained on a large dataset of Arabic poetry and prose. We carefully select the network architecture and tune these networks to achieve high classification accuracy. We also investigate improving the accuracy through classifying a poem based on the predictions of its comprising verses and predicting the diacritics of undiacritized verses. For diacritizing poetry, we use deep RNN in the sequence transcription configuration with undiacritized verses as the input sequence and the predicted diacritics as the output sequence.

This work has three main contributions: (*i*) We adopt, clean and analyze a large Arabic poetry dataset, and complement it with prose samples and clear split to train and test subsets, thus, proposing a benchmark for this research area. (*ii*) We extend the poetry classification problem to distinguish prose from poetry in addition to predicting the entire 16 poetry meters. We achieve accuracy significantly better than previous work and we propose a solution that provides 100% accuracy in some cases. (*iii*) To the best of our knowledge, this is the first work that uses deep learning to diacritize Arabic poetry. We found that the poetry diacritization accuracy is lower than that of the prose, suggesting the need for further research.

This paper has nine sections. Section 2 gives essential background on Arabic poetry, Section 3 surveys related work, Section 4 describes the neural networks used in this work, Section 5 describes and analyzes the dataset used, Section 6, describes the experimental part of this work, Section 7 presents the detailed results, Section 8 discusses the main results and compares with related work, and Section 9 provides the conclusions and outlines future work.

2. Arabic poetry

With the rich vocabulary of the Arabic language, the famous poets of the pre-Islamic era were eager to compete and to earn the fame of having their poems hanged on the walls of the Holly Mosque (al-Ka‘bah) and be one of the pendants (al-Mu‘alaqāt) poets (Margoliouth, 1925). People at that time and in the first centuries of the post-Islamic era, were able intuitively to recognize the quality of the poem, its rhythm as accepted poem or not, as well as its deep meaning. Later, a large number of people whose mother tongue is not Arabic started to learn Arabic and speak it, but with lower language mastery. This caused the language spoken by the common person to deteriorate in quality. One of the great linguists, al-Khalīl bin Aḥmad al-Farāhīdī (AD 718–786) passed one day by the market where copper pots were shaped by knocking. The collective sound of knocking attracted his attention to the similarity with the poetry rhythm. After deep thinking, he found that the Arabic poetry has rhythmic patterns and can be classified into 15 classes. Later on, one of his students, al-Akhfash discovered and added the 16th class. Each class has a certain meter (baḥr).

The passage of poetry called qaṣīdah consists of a number of verses of the same pattern in most cases. Each verse consists of two couplets (shaṭir) of approximately same lengths. The first couplet is called ṣadir and the second is ‘ajuz. The end of ‘ajuz usually has the same rhyme (qāfīyah), mostly same letter or sometimes letters, in the entire qaṣīdah (Atiq, 1987).

Arabic letters usually have diacritics. The diacritics are often not written but pronounced. The diacritics are called harakāt, whose absence is called *sukun*, which is marked in some cases by the diacritic (ْ). Harakah may be either *fatha* (َ), that is equivalent to short “a”, *damma* (ُ) that is equivalent to short “o” or “u”, or *kasra* (ِ) that is equivalent to short “i” or “e”. There are other diacritics, namely the *shadda* (ّ) that indicates double letters and the tanwīn variants are signs for adding the sound “n” at the end of a word, whether it is *fathatan* (ً), *dammatan* (ٌ), or *kasratan* (ٍ) (Alen *et al.*, 2012).

Word forms in Arabic morphology are usually based on the use of the verb pattern fa’ala (فَعَلَ). The three letters of this pattern are combined with extra letters and diacritics to change its tense or form or even to convert it to a noun form. In metering poetry verses, al-Farāhīdī used a similar approach. The rhythm in Arabic poetry comes from the succession of letters with and without harakāt. Al-Farāhīdī called the basic repeated sequences in a meter taf‘īlāt (feet). Table 1 lists the 16 meters (in Roman letters and Arabic) and their couplet patterns as sequences of two to four feet. The two couplets usually have the same pattern. The table also lists the meter circle, which is a group of similar meters. However, the meters in this

table are ordered according to their frequency, not circle (see Section 5). Each meter comes complete (tām) with all its feet in some poems or shortened (majzw') without some feet in other poems (Atiq, 1987). The table gives our estimations of the verse lengths (two couplets) in number of letters for the complete and the shortest variants. We use this estimation in the dataset preparation (see Section 5).

Table 1. The 16 classic Arabic poetry meters

No.	Meter	Baḥr	Circle	Couplet pattern	Complete length	Shortest variant (majzw') length
1	Ṭawīl	طَوِيل	1	فَعُولُنْ مَفَاعِيلُنْ فَعُولُنْ مَفَاعِيلُنْ	48	44
2	Kāmil	كَامِل	2	مُتَفَاعِلُنْ مُتَفاعِلُنْ مُتَفَاعِلُنْ	42	28
3	Basīṭ	بَسِيط	1	مُسْتَفْعِلُنْ فَاعِلُنْ مُسْتَفْعِلُنْ فَاعِلُنْ	48	34
4	Khafīf	خَفِيف	4	فَاعِلَاتُنْ مُسْتَفْعِلُنْ فَاعِلَاتُنْ	42	28
5	Wāfir	وَافِر	2	مُفَاعَلَتُنْ مُفاعَلَتُنْ فَعولُنْ	38	38
6	Rajaz	رَجَز	3	مُسْتَفْعِلُنْ مُسْتَفْعِلُنْ مُسْتَفْعِلُنْ	42	14
7	Ramal	رَمَل	3	فَاعِلَاتُنْ فَاعِلَاتُنْ فَاعِلَاتُنْ	42	28
8	Mutaqārib	مُتَقَارِب	5	فَعُولُنْ فَعُولُنْ فَعُولُنْ فَعُولُنْ	40	26
9	Sarīʿ	سَرِيع	4	مُسْتَفْعِلُنْ مُسْتَفْعِلُنْ فَاعِلُنْ	38	36
10	Munsariḥ	مُنْسَرِح	4	مُسْتَفْعِلُن مَفْعُولَاتُ مُسْتَفْعِلُن	42	28
11	Mujtathth	مُجْتَثّ	4	مُسْتَفْعِلُنْ فَاعِلَاتُنْ	28	28
12	Madīd	مَدِيد	1	فَاعِلَاتُنْ فَاعِلُنْ فَاعِلَاتُنْ	38	32
13	Hazaj	هَزَج	3	مَفَاعِيلُنْ مَفَاعِيلُنْ	28	28
14	Mutadārik	مُتَدَارِك	5	فَعِلُنْ فَعِلُنْ فَعِلُنْ فَعِلُنْ	40	30
15	Muqtaḍab	مُقْتَضَبّ	4	مَفْعُولَاتُ مُسْتَفْعِلُن	28	26
16	Muḍāriʿ	مُضَارِع	4	مَفَاعِيلُنْ فَاعَلَاتُنْ	28	28

During the seventies of the last century, with the rising use of binary numbers in digital computers, El-Katib (1971) used binary numbers to analyze Arabic poetry. The series of letters with sukuns and harakāt is represented as binary digits "1" and "0", respectively. For example the first foot in Ṭawīl meter has the mnemonic taf'īlah fa'ūlun (فَعُولُنْ), which has two sukun letters and three letters with harakāt. In binary, this foot is coded as 10100, where the least significant digit maps to the rightmost letter in فَعُولُنْ. Note that the *sukun* diacritic on a long vowel such as the *waw* (و) is usually omitted.

Arabic poets have some freedom in making changes on the basic pattern of a meter, where the allowed variations do not adversely affect the poem rhythm. Some of these variations can be in some poem verses and other variations, when used, must be applied to all the poem verses. For example, the listener ear is tolerant to omitting some sukun sounds. Omitting one sukun letter from the foot is called holding (qabḍ). In Ṭawīl meter, the fifth sukun in the taf'īlah mafā'īlun (مَفَاعِيلُنْ), coded as 101[1]0100, can be held (omitted) to the taf'īlah variant mafā'ilun (مَفَاعِلُنْ), and coded as 100100.

Metering verses is usually performed in few steps based on the way verses are pronounced and not how they are written. Figure 1 shows an example of metering a verse by Hātim al-Ṭā'ī, proverbial in generosity, addressing his wife Māwīyah. The verse is shown in the first row and means: *Māwīyah, money comes and goes, and what remains of money is good reputation and remembrance*. The second row shows the verse broken to two couplets and the third row shows the two couplets written as pronounced, which is called 'arūḍī writing. There are many rules for this writing; we list here some of the rules that are used in Row 3 of this example.

1. The letter with *shadda* is converted to two characters: the first one is with *sukun* and the second is with harakah; as in the wife name أَمَاوِيَّ that is converted to أَمَاوِيْ يَ.
2. The definite article al (ال) is either converted to (لْ) or removed with doubling the next letter, depending on the letter after this article; as in المَالَ that is converted to لْمَالَ and الذّكْرُ that is converted to ذْذِكْرُ.
3. Tanwīn diacritic is converted to the related harakah and the letter "n" (نْ); as in غَادٍ that is converted to غَادِنْ.
4. When the verse (and sometimes the first couplet) ends with harakah diacritic, this harakah is usually pronounced as a long vowel, so a vowel letter of same sound is added after this harakah; as in الذّكْرُ that is converted to ذْذِكْرُوْ.

Then the series of letters of the 'arūdī form are transcribed to the corresponding scansion code. Row 4 shows the verse scansion using the binary coding described above. Note that there are other used scansion codes. The verse scansion code is mapped to the corresponding taf'īlāt, as shown in Row 5. For example, the scansion 10100 is mapped to (فَعُولُنْ) because it stands for three letters with harakāt and two letters with *sukun* in the same order as in (فَعُولُنْ). Note that Row 6 shows

comments of the type of taf‘īlāt identified. Here five taf‘īlāt are intact and one is held. Finally, the identified taf‘īlāt are looked up in Table 1 to find the verse meter, which is Ṭawīl in this case (shown in Row 7).

1.	**Verse**	أَمَاوِيَّ إِنَّ المَالَ غَادٍ وَرَائِحٌ * وَيَبْقَى مِنَ المَالِ الأَحَادِيْثُ وَالذِّكْرُ							
2.	**Couplets**	وَيَبْقَى مِنَ المَالِ الأَحَادِيْثُ وَالذِّكْرُ				أَمَاوِيَّ إِنَّ المَالَ غَادٍ وَرَائِحٌ			
3.	**‘arūdī form**	ـثُ وَذِكْرُوْ	أَحَادِيـْ	مِنَ لَمَالِ كْ	وَيَبْقَى	وَرَائِحُنْ	لَ غَادِنْ	يَ إِنَّنَ لَمَا	أَمَاوِيْ
4.	**Scansion**	1010100	10100	1010100	10100	100100	10100	1010100	10100
5.	**Taf‘īlāt**	مَفَاعِيْلُنْ	فَعُوْلُنْ	مَفَاعِيْلُنْ	فَعُوْلُنْ	مَفَاعِلُنْ	فَعُوْلُنْ	مَفَاعِيْلُنْ	فَعُوْلُنْ
6.	**Taf‘īlāt type**	سَالِمَة Intact	سَالِمَة Intact	سَالِمَة Intact	سَالِمَة Intact	مَقْبُوضَة Held	سَالِمَة Intact	سَالِمَة Intact	سَالِمَة Intact
7.	**Meter**	Ṭawīl							

Figure 1. An example for metering a verse from a poem composed by Hātim al-Ṭā’ī

We should mention here that there are other Arabic poetry types. In the 20th century, a new kind of poetry appeared and is called the *free poetry*. This kind of poetry differs from the classical Arabic poetry described above, which is often referred to as the *vertical poetry*. The free poetry has the rhythm of poetry, but does not follow the structure of couplets or fixed number of feet and does not have consistent rhyme (qāfīyah). The free poem has borrowed its shape from other languages and sometimes can hardly be distinguished from prose (Badawi, 1975).

3. Literature review

Many research efforts have been carried out in the field of processing Arabic language. In the following paragraphs, we review related work in classifying Arabic poetry and diacritizing Arabic text.

3.1. Arabic poetry meter classification

Several approaches have been used to recognize the meter of an Arabic poem. These approaches can be roughly divided into rule-based and machine-learning approaches.

Ismail *et al.* (2010) developed a rule-based prototype called expert system harmony test (ESHT) for testing the harmony and identifying the meters of Arabic poetry. The authors used expert knowledge to design this rule-based system and tested it on only 20 poems. Alnagdawi *et al.* (2013) develop a three-phase program for finding the poem meter. First, regular expressions and context-free grammar are applied to convert the poem into its ‘arūdī form. Second, segmentation is used to divide the ‘arūdī form into short and long sounds. Third, the generated sound string is compared against the patterns of the poetry meters to determine the best match. The system was tested on 128 verses from different Arabic poems and with modest classification accuracy of 75%. Abuata and Al-Omari (2018) followed a similar procedure to classify poetry but they only considered the ṣadir couplet of the verse. The algorithm was tested on 417 verses from different Arabic poems and achieved 82.2% accuracy.

In all these works, the authors relied on rule based techniques. Such techniques require deep understanding of the poetry meters and ‘arūd science to develop a comprehensive set of rules to do the classification accurately. The performance of such technique relies heavily on the accuracy of the developed and selected rules. Based on the reported accuracies, the effectiveness of the rule-based approaches in this domain is unsatisfactory.

Yousef *et al.* (2019) developed machine learning models using deep recurrent neural networks (RNN) to classify Arabic and English poetry verses. The RNN proved to be powerful in extracting features for each class taking into consideration the variations between the samples that belong to the same class. Hence, there is no need to manually handcraft feature extraction. This approach was tested on huge datasets crawled from specialized web sites. For Arabic poetry, the approach achieved 96.38% classification accuracy on a trimmed dataset of 11 classes and 94.11% on the entire 16 classes.

Al-Talabani (2020) developed a voice-based model instead of a text-based model to solve the classification task. The author generated from the input voice a time series consisting of the linear prediction cepstrum coefficient (LPCC) and Mel

frequency cepstral coefficient (MFCC). Then the time series is fed to a long short-term memory (LSTM) classifier to determine the poem meter. Also, they converted the time series features vector into a non-time series vector by calculating the mean and the standard deviation of the features in each frame. Then the generated vector is fed into an SVM classifier. The dataset used in the evaluation (230 verses) is relatively small and includes only three meters out of the sixteen. For the speaker independent case, the best classification accuracy of 88.89% is achieved using the LSTM classifier fed with MFCC features only.

Al-shaibani *et al.* (2020) used text-based deep RNN to classify Arabic poetry. They propose using five wide bidirectional recurrent layers and the GRU memory cell in lieu of the LSTM cell. They tested their network using a dataset collected for this purpose consisting of 55,440 verses and including samples for 14 out of the 16 Arabic poetry meters. Although they used a dataset smaller than the one used by Yousef *et al.* (2019), they report a slightly better accuracy of 94.32%.

In this work, we extend these machine learning approaches to reach higher classification accuracy by proposing efficient deep RNN architecture and exploiting classification over multiple poem verses. We also investigate the diacritics presence on the classification accuracy and we use a machine learning approach to automatically add diacritics to Arabic poetry to facilitate poetry pronunciation and classification.

3.2. Diacritization

Diacritizing Arabic text is important for the proper reading and pronunciation of printed or written text. This importance is even higher for poetry as the poetry language is more refined and innovative. Previous work used rule-base, statistical and hybrid approaches (Azmi and Almajed, 2015). We present here the state of the art approaches that have been proposed to automatically add diacritics to Arabic word sequences.

Elshafei *et al.* (2006) and Hifny (2012) used statistical models followed by search algorithms to find the best probable sequence of diacritized words for a given undiacritized word sequence. The former researchers used the hidden Markov statistical models (HMM) for modeling and the Viterbi algorithm to find the best probable result. On the other hand, Hifny, used statistical n-gram language modeling approach to assign scores for the possible diacritized sequence and dynamic programming to search for the best probable result. These proposed approaches achieve 4.1% and 3.4% diacritization error rates, respectively. Azim *et al.* (2012) used speech based diacritizer to complement the text based diacritizer. The text based modeling is done using conditional random fields and the speech based models are done using HMM. This hybrid approach reduced the diacritization error rate to 1.5%.

With the rise of deep machine learning approaches, the most recent works are based on deep recurrent neural network (RNN) models to solve the diacritization problem as a sequence transcription problem (Abandah *et al.*, 2015; Rashwan *et al.*, 2015; Mubarak *et al.*, 2019; Abandah and Abdel-Karim, 2020). Abandah et al (2015) used a bidirectional LSTM network to diacritize sequences without the need to perform syntactical or morphological preprocessing steps. Mubarak *et al.* (2019) used the encoder decoder RNN models with attention mechanism to improve the accuracy. The most recent work proposed by Abandah and Abdel-Karim (2020) performed intensive evaluation for different encoding strategies and tuned the hyper-parameters and configuration of the bidirectional LSTM RNN and evaluated the proposed systems in terms of speed and performance. The recommended system achieved significant improvement over the best published results.

For general sequence transcription, RNN were often used in encode-decoder configuration (Cho *et al.*, 2014). These networks have problems with long sequences. The state of the art approaches use attention mechanisms and transformers to overcome these problems (Vaswani *et al.*, 2017; Devlin *et al.*, 2018). However, these techniques are not needed here because we have one-to-one relationship between the input and output sequences. Currently, the bidirectional RNN networks perform best in diacritizing Arabic text.

4. Sequence classification and transcription

Sequence classification is the process of finding the class (type) of a sequence be it a time series or a sequence of characters or words. Sequence transcription is the process of translating an input sequence to the corresponding target sequence of a different type. These processes include finding the sentiment of a paragraph, finding the meter of a poetry verse, language translation, voice recognition, and diacritizing Arabic texts.

4.1. Recurrent neural networks

Recurrent neural networks (RNN) are often successfully used to solve sequence related problems (Rumelhart *et al.*, 1986; Sutskever *et al.*, 2014). RNN advantage comes from their internal state (memory cells) that is kept during processing the sequence one step at a time and using this state in the next step. Given a sequence of inputs $(\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_T)$, an RNN

computes a sequence of outputs $(\boldsymbol{y}_1, \boldsymbol{y}_2, \dots, \boldsymbol{y}_T)$ based on the computation of a sequence of hidden vectors $\boldsymbol{h}_t$ by iterating the following equations from step $t = 1$ to T.

$$\boldsymbol{h}_t = f_h(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) \tag{1}$$

$$\boldsymbol{y}_t = f_y(\boldsymbol{h}_t) \tag{2}$$

In classification, the final output $\boldsymbol{y}_T$ is used to find the class of the input sequence (many-to-one) whereas the entire output sequence $(\boldsymbol{y}_1, \boldsymbol{y}_2, \dots, \boldsymbol{y}_T)$ is used in sequence transcription (many-to-many). The basic RNN described here are inefficient in handling sequences with long dependencies. The memory cells tend to forget the first inputs of the sequence.

4.2. Long short-term memory cells

Hochreiter and Schmidhuber (1997) proposed the long short-term memory cell (LSTM) for its advantage in faster convergence and detecting and remembering long-term dependencies. In addition to the short-term state $\boldsymbol{h}_t$, the LSTM cell has a long-term state $\boldsymbol{c}_t$. Both states are functions of the current input and the previous states as summarized in the following functions. The cell output is simply the short-term state.

$$\boldsymbol{h}_t = f_h(\boldsymbol{h}_{t-1}, \boldsymbol{c}_{t-1}, \boldsymbol{x}_t) \tag{3}$$

$$\boldsymbol{c}_t = f_c(\boldsymbol{h}_{t-1}, \boldsymbol{c}_{t-1}, \boldsymbol{x}_t) \tag{4}$$

$$\boldsymbol{y}_t = \boldsymbol{h}_t \tag{5}$$

When transcribing a sequence, the RNN output at Step t depends on the "seen" input subsequence $(\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_t)$. For problems such as text diacritization where the output depends on the entire input including the "unseen" subsequence $(\boldsymbol{x}_{t+1}, \boldsymbol{x}_{t+2}, \dots, \boldsymbol{x}_T)$, the conventional unidirectional RNN fail to give satisfactory output.

4.3. Bidirectional RNNs

Schuster and Paliwal (1997) proposed bidirectional RNNs to solve problems that require exploiting the future context in addition to the past context. A bidirectional RNN layer has two adjacent unidirectional networks in each layer. The forward network is trained by presenting the input sequence in the forward direction $(\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_T)$ and the backward network is trained by presenting it in the backward direction $(\boldsymbol{x}_T, \boldsymbol{x}_{T-1}, \dots, \boldsymbol{x}_1)$. The output is a function of both layers and exploits past and future contexts. The output of a bidirectional layer is often a concatenation of the outputs of its forward and backward networks.

4.4. Deep RNNs

For complex problems such as language translation or diacritizing text, multiple RNN layers are needed to achieve efficient solution. Multiple RNN layers are stacked on top of each other forming a deep network where the output sequence of one layer is the input sequence for the next higher layer (Graves *et al.*, 2013). The input sequence is presented to the lowest RNN layer and the final output is derived from the highest RNN layer often through one or more dense layers. With N layers in the stack, the hidden vectors $\mathbf{h}^n$ are computed by iterating from layer $n = 1$ to N and from step $t = 1$ to T, as shown in Equation 6, where $\mathbf{h}^0 = \mathbf{x}$. The network final output $\boldsymbol{y}_t$ is computed according to Equation 7.

$$\boldsymbol{h}_t^n = f_h^n(\boldsymbol{h}_{t-1}^n, \boldsymbol{h}_t^{n-1}) \tag{6}$$

$$\boldsymbol{y}_t = f_y(\boldsymbol{h}_t^N) \tag{7}$$

Deep RNN's efficiency comes from breaking the problem into stages where lower layers extract basic features that are used by higher layers to extract more sophisticated features that are used, in turn, to predict the final output. Also using bidirectional layers provides better context for detecting features. In this work we use deep bidirectional RNNs with LSTM cells (BiLSTM).

5. Dataset

Yousef *et al.* (2018) collected the Arabic Poem Comprehensive Dataset (APCD). They collected this dataset from two specialized web sites: The Collection (2020) and The Poetry Encyclopedia (2020), which aim to collect, preserve, and publish Arabic poetry comprehensively and hold millions of Arabic poetry verses. We downloaded the APCD dataset that has 1,831,770 poem verse records: 1,691,671 records of the 16 classical meters and the remaining 140,099 records either are of seven other non-classical meters or unlabeled. Each record has eight fields: era, poet, collection, rhyme, meter, left couplet, right couplet, and the entire verse of both couplets, as shown in Figure 2.

	A	B	C	D	E	F	G	H
1	العصر	الشاعر	الديوان	القافية	البحر	الشطر الايسر	الشطر الايمن	البيت
2	قبل الإسلام	عمرو بن قُمَيئة	الديوان الرئيسي	ل	السريع	مَن كان مِن كِندَة أو وائِلِ	يا راكِباً بَلِّغ ذَوي حِلفِنا	يا راكِباً بَلِّغ ذَوي حِلفِنا مَن كان مِن كِندَة أو وائِلِ
3	قبل الإسلام	عمرو بن قُمَيئة	الديوان الرئيسي	ل	السريع	مَن سَعفِ البَحرَينِ وَالساحِلِ	وَالحَيُّ عِندَ القَيسِ حَيثُ اِنتَووا	وَالحَيُّ عِندَ القَيسِ حَيثُ اِنتَووا مَن سَعفِ البَحرَينِ وَالساحِلِ
4	قبل الإسلام	عمرو بن قُمَيئة	الديوان الرئيسي	ل	السريع	كَمَوضِعِ الزَورِ مِن الكاهِلِ	إِنّا وَإِيّاهُمُ وَما بَينَنا	إِنّا وَإِيّاهُمُ وَما بَينَنا كَمَوضِعِ الزَورِ مِن الكاهِلِ

Figure 2. Example three records of the Sarīʿ meter from the APCD dataset

5.1. Dataset preparation

We noticed that there are some errors in collecting this dataset. To reduce the effect of these errors, we excluded records that have any of the following three problems:

1. Missing left or right halves
2. Too long verse: Length in number of letters is larger than 120% of the complete meter length (see Table 1)
3. Too short verse: The length is smaller than 80% of the shortest meter variant length (see Table 1)

The last two criteria exclude records with verse lengths inconsistent with the meter label. Using these three criteria, we excluded 63,303 records. We call this version APCD2 that has 1,628,368 records and is summarized in Table 2. This table shows the number of sample verses per meter in descending order and our split to test and train sets. The following subsection elaborates on this split. Note that this dataset is skewed; some meters have much more verses than others.

Table 2. APCD2 dataset number of verses per meter and the test/train split

No.	Meter	Kept verses	Test set	Test ratio	Train set	Train ratio
1	Ṭawīl	395,638	38,249	9.7%	357,389	90.3%
2	Kāmil	358,462	35,048	9.8%	323,414	90.2%
3	Basīṭ	235,606	23,939	10.2%	211,667	89.8%
4	Khafīf	151,784	13,691	9.0%	138,093	91.0%
5	Wāfir	130,918	12,866	9.8%	118,052	90.2%
6	Rajaz	103,059	12,196	11.8%	90,863	88.2%
7	Ramal	71,527	7,017	9.8%	64,510	90.2%
8	Mutaqārib	62,350	6,322	10.1%	56,028	89.9%
9	Sarīʿ	56,249	5,344	9.5%	50,905	90.5%
10	Munsariḥ	27,708	2,815	10.2%	24,893	89.8%
11	Mujtathth	15,718	1,728	11.0%	13,990	89.0%
12	Madīd	7,418	687	9.3%	6,731	90.7%
13	Hazaj	6,916	915	13.2%	6,001	86.8%
14	Mutadārik	4,204	294	7.0%	3,910	93.0%
15	Muqtaḍab	702	119	17.0%	583	83.0%
16	Muḍāriʿ	109	19	17.4%	90	82.6%
	Total	1,628,368	161,249	9.9%	1,467,119	90.1%

5.2. Test/train splitting

Verses of same poem are listed consecutively in this dataset. Assuming that a poem's verses are consecutive and a change in any of the attributes era, poet, collection, rhyme, or meter implies a new poem, we estimated that APCD2 has verses from 115,478 poems. The median poem length is five verses and the range is one to 2,367 verses. We randomly split these poems into two sets: 10% test set and 90% train test. Table 2 shows that this split is generally well stratified over the 16 meters. On the verse level, the table shows that the split is not perfect 10-90 split (especially for meters with low numbers of samples) because the number of verses per poem is not constant. We publish this version with this clear test/train split hoping that this dataset will become a benchmark in the related research (Abandah, 2020a).

5.3. Dataset characteristics

We present here some statistics about APCD2 to illustrate its diversity. Table 3 presents three aspects about this dataset: rhyme, era and poet. The rhyme letters and eras are ordered in descending order according to their respective counts. The table shows that all 29 Arabic letters are used as rhymes. However, the top six letters (*Reh*, *Lam*, *Meem*, *Dal*, *Beh*, and *Noon*) are the most popular rhymes and are used in about 70% of the sample. The dataset classifies the poems into 12 eras. About 40% of the sample verses are of the modern era and the rest are older dating back to pre-Islam in the sixth and early seventh centuries. The sample has poems for 3,360 poets; 50% of these poets are from the Modern, pre-Islamic, and Fatimid eras.

Table 3. APCD2 rhyme, era and poet distributions

Rhyme letter		Verses	Era	Verses	Poets
Reh	ر	256,197	Modern	645,621	691
Lam	ل	190,412	Abbasid	228,288	386
Meem	م	184,513	Mamluk	150,467	132
Dal	د	183,843	Ottoman	141,999	174
Beh	ب	159,828	Fatimid	121,705	477
Noon	ن	151,043	Al Ayoubi	108,553	101
Ain	ع	65,935	Morocco and Andalusia	99,064	285
Qaf	ق	64,092	Umayyad	61,518	236
Hamza	ء	48,301	Seasoned	29,635	167
Teh	ت	45,405	Pre-Islam	21,156	523
Feh	ف	41,921	Between the two countries	18,238	42
Hah	ح	39,574	Islamic	2,124	146
Heh	ه	35,257			
Seen	س	33,150			
Yeh	ي	28,044			
Kaf	ك	25,013			
Jeem	ج	15,209			
Dad	ض	11,994			
Alef	ا	8,088			
Tah	ط	7,614			
Zain	ز	5,162			
Sad	ص	4,721			
Sheen	ش	4,676			
Waw	و	4,475			
Theh	ث	3,945			
Thal	ذ	2,691			
Khah	خ	2,218			
Ghain	غ	2,056			

Zah	ظ	1,546			
Alef Maksura	ى	1,259			
Total		1,628,182	1,628,368	3,360	

Assuming that APCD2 is representative sample of the Arabic poetry, we present the change in meter popularity with time using Table 4. The table shows distributions of the 16 meters in the 12 eras ordered from the oldest to the modern era. For example, C1-Ṭawīl meter popularity declined with time with a peak of 50.7% in the Early Islamic era and 19.8% in the Modern era. On the other hand, the C2-Kāmil meter's popularity increased over time with only 11.0% in the Early Islamic era and 24.2% in the Modern era.

Table 4. Distribution of the 16 meters in the 12 eras ordered from the oldest to the modern era

Meter	Pre-Islamic	Seasoned	Islamic	Umayyad	Between the two countries	Abbasid	Morocco and Andalusia	Fatimid	Al Ayoubi	Mamluk	Ottoman	Modern
1. Ṭawīl	37.3%	42.2%	50.7%	47.5%	23.6%	21.2%	32.5%	24.8%	26.3%	24.7%	25.5%	19.8%
2. Kāmil	15.1%	14.0%	11.0%	13.2%	13.0%	18.5%	25.1%	22.2%	23.7%	21.7%	22.2%	24.2%
3. Basīṭ	11.8%	12.7%	9.6%	12.8%	11.1%	11.7%	14.5%	13.7%	15.6%	16.8%	16.3%	14.9%
4. Khafīf	4.4%	3.2%	2.6%	4.5%	7.6%	10.2%	4.3%	7.5%	5.8%	7.3%	10.6%	11.9%
5. Wāfir	16.4%	13.8%	13.4%	13.0%	7.5%	8.8%	5.8%	8.5%	6.5%	6.8%	6.1%	8.0%
6. Rajaz	1.5%	1.2%	3.0%	1.1%	19.2%	7.6%	2.1%	6.8%	5.4%	9.9%	5.7%	6.4%
7. Ramal	3.4%	1.9%	5.6%	1.3%	3.5%	3.4%	3.6%	1.8%	3.7%	2.8%	4.9%	6.2%
8. Mutaqārib	4.2%	7.4%	2.8%	3.0%	3.4%	5.5%	4.6%	6.3%	3.5%	2.4%	2.0%	3.3%
9. Sarīʿ	2.7%	1.9%	0.1%	1.0%	4.8%	5.1%	3.8%	4.7%	5.0%	3.9%	3.1%	2.6%
10. Munsariḥ	1.9%	1.3%	0.8%	1.8%	3.7%	5.0%	1.3%	2.3%	2.2%	1.6%	1.3%	0.5%
11. Mujtathth	0.0%	0.0%	0.2%	0.0%	0.2%	0.9%	0.8%	0.6%	0.9%	0.9%	1.0%	1.3%
12. Madīd	0.7%	0.2%	0.2%	0.5%	0.1%	0.6%	0.7%	0.3%	0.6%	0.5%	0.8%	0.3%
13. Hazaj	0.6%	0.2%	0.0%	0.2%	2.2%	1.3%	0.1%	0.5%	0.5%	0.4%	0.1%	0.2%
14. Mutadārik	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.6%	0.1%	0.2%	0.3%	0.2%	0.4%
15. Muqtaḍab	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.2%	0.0%	0.0%	0.0%	0.0%	0.1%
16. Muḍāriʿ	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%

Anees (1952) noted that some poetry meters were popular in some eras and declined in others. He noted that the C2-Kāmil and C3-Basīṭ meters popularity increased since the Andalusian and Abbasid eras, perhaps due to many reasons and factors, including the new environments that the Arabs opened up to in the Levant, Iraq and Andalusia, which are different from the environment of the desert in the Arabian Peninsula, as well as the cultural cross-fertilization between the Arab and other non-Arab elements, such as the Persians in the Levant, the Goths, the Saqlabis and the Berbers in the West Arab region. He also noted that in our modern era, when poets increased, cultures converged, and technologies spread, some meters receded, such as the C1-Ṭawīl meter, and others advanced, such as the C7-Ramal, which remained dormant until the modern era came and brought about a great renaissance for this meter.

5.4. Use of diacritics

The sample has wide range of diacritics usage. Figure 3 shows the cumulative distribution function of the diacritics to letters ratio of APCD2 verses. About 18.5% of the sample verses have zero diacritics, the average ratio for the entire sample is 0.27 diacritics per letter, and there are some verses with heavy diacritics reaching the ratio 1.2.

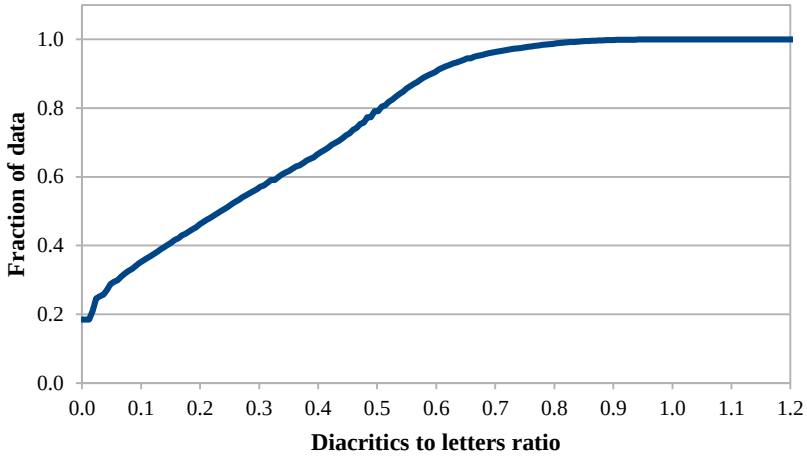

Figure 3. Cumulative distribution of the verse diacritics to verse letters ratios

Figure 4 shows three sample verses with zero, average, and large diacritics to letters ratios. Samples without diacritics are hard to pronounce and only fluent Arabic speakers can pronounce them correctly. When some crucial diacritics are present, pronunciation becomes much easier, and when all diacritics are present, pronunciation becomes straightforward. In Arabic, the number of diacritics can exceed the number of letters when the *shadda* diacritic (ّ) is present because its letter can have another diacritic. *Shadda* diacritic, in fact, indicates that the original spelling has double letter collapsed into one. For example, the origin of the word sayyidu "master" (سَيِّدُ) is (سَيْيِدُ).

Diacritics to letters ratio	Sample
0.0	وكيف تكاونوا من غير شيء وكيف تناولوا الغرض البعيدا
0.27	وَلَو شِئنا حَمَيناها البَوادي كَما تَحمي أُسودُ الغابِ غابا
1.2	حُبُّ عَلِيَّ عُلُوُّ هِمَّهْ لِأَنَّهُ سَيِّدُ الأَئِمَّهْ

Figure 4. Three sample verses of none, average, and heavy diacritics

5.5. Prose samples

In order to build machine-learning models that can distinguish poetry from prose, we added to APCD2 some prose samples. We selected these samples to represent the classical and the modern standard Arabic. The classical samples come from the Tashkeela dataset (Fadel *et al.*, 2019) and the modern samples come from the LDC ATB3 dataset (Maamouri *et al.*, 2004). These two datasets are frequently used as benchmarks in Arabic text diacritization research. We added 28,635 prose sequences available in the two datasets that have sequence lengths compatible with the poetry verse lengths. The selected sequences have lengths between 12 and 57 letters. As these two datasets are diacritized, we removed the diacritics from 30% of the selected sequences to reduce the effect of diacritics presence on the model's ability to distinguish poetry from prose. Table 5 shows the numbers of the selected prose sequences for the test and train subsets.

Table 5. Prose samples in APCD2 (test/train numbers)

Source	Original dataset sequences	Selected sequences	Total selected
Tashkeela	2,500/52,500	695/14,748	15,443
LDC ATB3	3,857/22,170	1,973/11,219	13,192
Total		2,668/25,967	28,635

Figure 5 summarizes the number of sample verses in APCD2 comprising the 16 poetry classes and the prose; a total of 1,657,003 poetry and prose verses. Notice that the figure has a log scale and the dataset is highly skewed. The most frequent

class is more than three orders of magnitude larger than the least frequent class. This dataset skew reflects the fact that Arabic poets prefer some poetry meters over others.

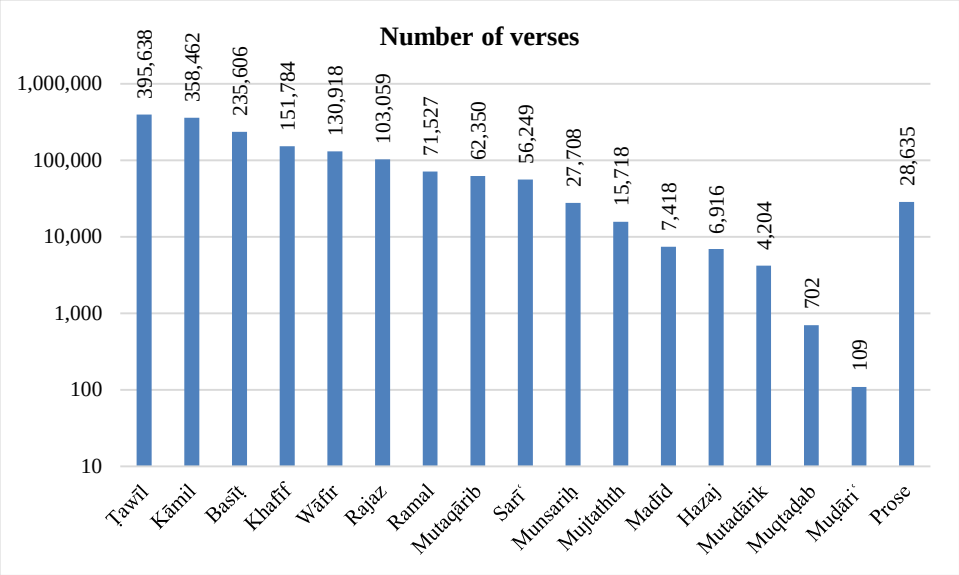

Figure 5. Number of verses for each class in APCD2

6. Experiments

In this section, we describe our experimental setup and basic machine learning experiments conducted to select and tune a suitable classification model and an automatic diacritization model.

6.1. Experimental setup

The specifications of the platform on which our experiments are performed are shown in Table 6. Although the computer has a powerful GPU, we conducted most of the experiments on the CPU because the GPU does not give better speedup. Only the diacritization experiments get better performance on the GPU.

Table 6. Specifications of the experimental platform

Aspect	Specification
CPU	Intel Core i7-9700KF @ 3.6 GHz, 8 cores, 12 MB cache
GPU	Nvidia GeForce RTX 2080 @ 2.1 GHz, 2944 CUDA cores, 8 GB memory
Memory	32 GB DDR4-SDRAM @ 2666MHz
OS	Ubuntu 20.04 LTS, 64-bit
Libraries	Python 3.8.2, TensorFlow 2.2.0, Keras 2.3.0-tf

The dataset and source code used in this work are posted on GitHub (Abandah, 2020a). You can recreate the main results of this work from this repository. Moreover, the trained model is available on a web page for interested users to find poetry meters (Abandah, 2020b).

6.2. Classification base model

We developed our models using Python programming language, Keras high-level API, and TensorFlow deep learning library (Google TensorFlow, 2020). Figure 6 shows the network of our baseline model. The input verse is presented one character at a time to the embedding layer that translates the input characters to 32-long vectors. The model has two BiLSTM layers; each layer has 2×128 cells. The output layer is a fully-connected layer that uses the softmax activation

function with one output for each of the 17 classes. This model is compiled to use Adam optimizer in the backpropagation through time (BPTT) training scheme and categorical cross entropy as loss function. The model trains using mini batches of size = 64 sequences (Géron, 2019). The skeleton code of this model is in Appendix A.

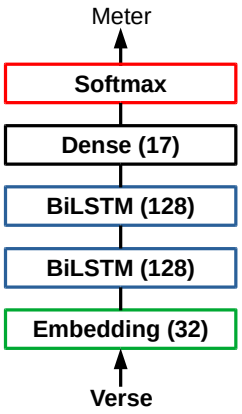

Figure 6. The base model with two BiLSTM layers

We use 85% of the training set for training the model for a maximum of 100 epochs. At the end of each epoch, the model predicts the classes of the rest 15%-validation set and the accuracy on predicting this set is used to early stop training when this accuracy does not improve for five consecutive epochs. Finally, the weights of the epoch that has best validation accuracy are used in the prediction phase.

6.3. Model tuning

We performed many experiments to tune the model's hyper-parameters. Table 7 summarizes these experiments with respect to the base model described above that has two BiLSTM layers, 128 cells per layer, 64-batch size, and Adam optimizer. The table shows the number of model parameters to reflect the model's complexity and speed, accuracy in predicting the classes of the test subset, and training length in total epochs and training time. As APCD2 is huge and requires long training times, we selected only 100,000 verses randomly of the APCD2 training set to perform many tuning experiments. The accuracy reported in this table is on 15%-test subset of these selected verses.

Table 7. Summary of the tuning experiments with respect to the base model using a subset of APCD2

Model hyper parameters	Model parameters	Accuracy	Training epochs	Training time
BiLSTM layers = 1	171 k	94.0%	42	1.5 hrs.
BiLSTM layers = 2 (Base model)	565 k	94.8%	30	3.1 hrs.
BiLSTM layers = 3	959 k	94.5%	23	3.9 hrs.
Cells/layer = 64	152 k	94.3%	37	1.8 hrs.
Cells/layer = 128 (Base model)	565 k	94.8%	30	3.1 hrs.
Cells/layer = 256	2,177 k	94.4%	14	4.7 hrs.
Batch size = 32	565 k	94.9%	24	3.6 hrs.
Batch size = 64 (Base model)	565 k	94.8%	30	3.1 hrs.
Batch size = 128	565 k	94.6%	29	2.3 hrs.
Optimizer = NAG	565 k	93.4%	59	6.1 hrs.
Optimizer = Adam (Base model)	565 k	94.8%	30	3.1 hrs.
Optimizer = RMSProp	565 k	93.8%	20	2.1 hrs.

The table shows sensitivity analysis for using 1, 2 and 3 BiLSTM layers. Although the shallow network has the fewest parameters (171 k) and is the fastest (1.5 hrs.), it is less accurate (94.0%). The deeper networks are slower and more accurate and the two-layer network has the best accuracy (94.8%). The table also shows that the network complexity in total

number of parameters is roughly proportional to the square number of cells per LSTM layer. The network with 256 cells per layer is roughly 4^2 times more complex than the one of 64 cells per layer (2,177 k vs. 152 k). The most complex model is the slowest (4.7 hrs.) but is less accurate (94.4%) than the base model of 128 cells per layer.

Increasing the mini batch size improves the training time as it benefits from the available hardware parallelism and slightly affects the model accuracy. The base batch size of 64 is a good balance between speed and accuracy. Finally, the table shows the performance of the following three optimizers:

1. The Nesterov accelerated gradient descent (NAG) optimizer with η=0.01 and momentum β=0.9 (Nesterov, 1983).
2. The adaptive moment estimation (Adam) optimizer (Kingma and Ba, 2014) with default learning rate η=0.001 and default hyperparameters β_1=0.9, β_2=0.999 and ε=10^{-7}.
3. RMSprop optimizer with η=0.001 and decay rate β=0.9 (Tieleman and Hinton, 2012).

The adaptive optimizers Adam and RMSprop converge faster than NAG (59 epochs) and are more accurate. Adam optimizer converges slower (30 epochs) than RMSprop (20 epochs) but gives more accurate model.

6.4. Network architecture effect

We evaluated the performance of the base model on the entire APCD2 dataset (see the first entry in Table 8). The model's accuracy improves when it trains on a larger training set (97.15% here versus 94.8% in Table 7). Table 8 also shows the error rate and macro accuracy. The error rate is (1 - accuracy) and the macro accuracy comes from computing the accuracy independently for each class and then taking the average. As we have dataset class skew, the macro accuracy is lower than the accuracy over all instances because classes with fewer instances have lower accuracy. We elaborate on this issue in Section 7.

Table 8. Comparison among wide, deep and unidirectional LSTM networks on the entire APCD2 dataset

Experiment	Model parameters	Accuracy	Error rate	Macro Accuracy
Wide, 2 bidirectional LSTM layers, 128 cells/layer	565 k	97.15%	2.85%	94.81%
Deep, 4 bidirectional LSTM layers, 64 cells/layer	**350 k**	**97.27%**	**2.73%**	**94.88%**
Deep, 4 unidirectional LSTM layers, 110 cells/layer	358 k	97.01%	2.99%	93.97%

As deep networks often have higher parameter efficiency than shallow ones, we tested a network with twice the LSTM layers and half the number of cells per layer (Srivastava *et al.*, 2015). The table shows that the deep network has slightly better accuracy at 97.27% with fewer parameters (350 k). This deep bidirectional network is shown in Figure 7. Similar to the wide network, it has an embedding layer at the input and its output layer is a dense layer with the softmax activation function. However, the deep network has four bidirectional LSTM layers instead of two layers.

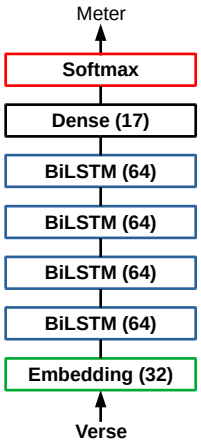

Figure 7. The deep network with four BiLSTM layers

To demonstrate that the BiLSTM architecture is beneficial for classifying poetry, we tested a unidirectional network. The tested unidirectional network also has four LSTM layers, but we increased the number of cells per layer to 110 so that it has

model parameters (358 k) similar to the bidirectional network. Table 8 shows that the unidirectional network is less accurate (97.01%), implying that the bidirectional architecture is more efficient in extracting features from the input verses that are useful to classify poetry.

The confidence in these accuracy numbers is pretty high because the support test set is very large (163,917 verses). For example, with 95% confidence level, the deep model accuracy of 97.27% has a confidence interval (97.19%, 97.35%). Moreover, the confidence intervals of the accuracies of the above three models are not overlapping.

6.5. Training convergence

We propose adopting the deep bidirectional model because it efficiently achieves the best accuracy. Figure 8 shows the training curves of this model. With early stopping and 5-epoch patience, training stopped after 57 epochs. The best accuracy on the validation set is 97.93% at Epoch 52. As expected, the accuracy on the training set steadily improves with time; it reaches 98.07% at Epoch 57. However, the accuracy on the test set is lower at 97.27%. This difference is mainly due to our train/test split on poem basis, not verse basis.

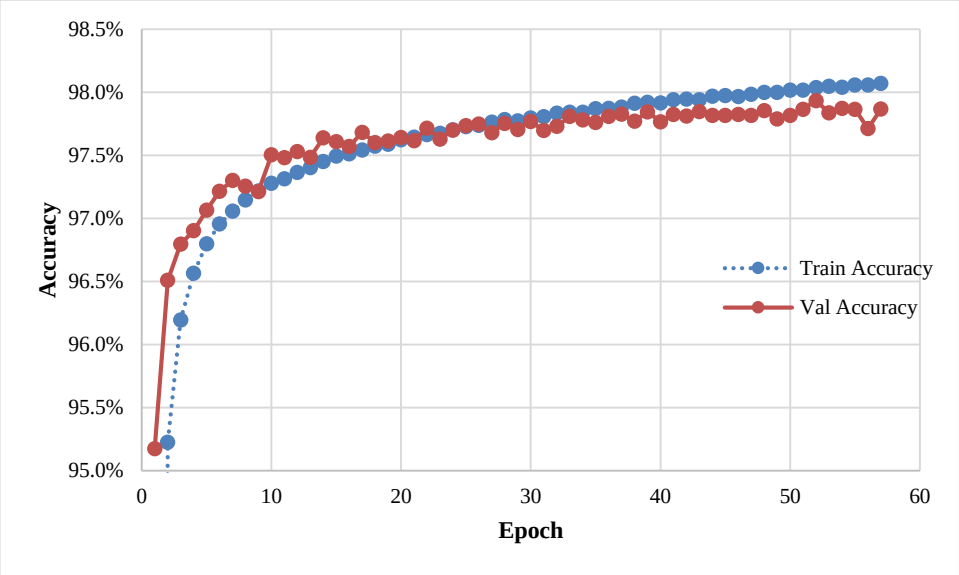

Figure 8. Training curves of the proposed deep bidirectional model

6.6. Automatic diacritization experiments

We experimented with automatically adding diacritics to Arabic poetry by using the machine learning approach suggested by Abandah and Abdel-Karim (2020). To develop a model for diacritizing Arabic poetry verses, we selected all the verses in the training set that have diacritics to letters ratio of 0.50 or higher (refer to Figure 3). This selection rule provided 368,617 diacritized verses consisting of 3,475 k words and were split into 85% training set and 15% validation set.

It turned out that poetry is harder to diacritize compared with prose. The diacritization model performance on the validation set is 6.08%-diacritization error rate (DER) and 20.40%-word error rate (WER). Table 9 compares this performance with the performance of two models trained to diacritize prose samples (Abandah and Abdel-Karim, 2020). Tashkeela is a dataset representative of the classical Arabic and LDC ATB3 is a dataset of modern standard Arabic.

Table 9. The diacritization and word error rates for poetry and prose

Dataset	Size in words	DER	WER
Tashkeela	2,312 k	1.97%	5.13%
LDC ATB3	305 k	2.46%	8.12%
Arabic Poetry	**3,475 k**	**6.08%**	**20.40%**

Although the dataset used in diacritizing poems is larger than the two prose datasets combined, the error rate in diacritizing poems is about three times worse than diacritizing prose. There are two main reasons for this difficulty. First, the poetry language is usually carefully and innovatively selected resulting in verses that often have uncommon structures and terms making them harder to diacritize. Second, the poet has some freedom in diacritizing the ends of some words to meet the selected rhyme and pattern of the poem.

7. Detailed Results

We present here the performance details of the proposed deep bidirectional model and investigate the diacritics presence on the model's accuracy.

7.1. Detailed accuracy results

Table 10 shows the model's prediction precision, recall and F1 score for the 17 classes. The support column shows the number of verses in the test set. These three metrics generally decrease with smaller training samples. The most frequent class (C1-Ṭawīl) has the best accuracy with F1 score of 99.0%. C14-Mutadārik class has the worst precision (87.6%) because the model wrongly classifies many other classes such as C6-Rajaz and C8-Mutaqārib verses as C14-Mutadārik (see the confusion matrix in Figure 9). C12-Madīd has the lowest recall (88.4%) because the model wrongly classifies many of its verses as C4-Khafīf and C7-Ramal. For these reasons, the F1 scores of C14-Mutadārik and C12-Madīd are the lowest (90.6% and 89.9%), even lower than classes with fewer samples such as C15-Muqtaḍab and C16-Muḍāriʿ.

Table 10. Precision, recall, and F1 score for each class

No.	Class	Precision	Recall	F1 score	Support
1	Ṭawīl	99.1%	99.0%	99.0%	38,249
2	Kāmil	96.8%	97.9%	97.4%	35,048
3	Basīṭ	98.1%	98.7%	98.4%	23,939
4	Khafīf	97.8%	96.5%	97.1%	13,691
5	Wāfir	98.6%	98.8%	98.7%	12,866
6	Rajaz	94.9%	91.6%	93.2%	12,196
7	Ramal	93.9%	95.9%	94.9%	7,017
8	Mutaqārib	97.9%	96.9%	97.4%	6,322
9	Sarīʿ	93.0%	94.1%	93.6%	5,344
10	Munsariḥ	94.2%	92.1%	93.2%	2,815
11	Mujtathth	92.0%	95.3%	93.6%	1,728
12	Madīd	91.4%	88.4%	89.9%	687
13	Hazaj	97.0%	94.8%	95.9%	915
14	Mutadārik	87.6%	93.9%	90.6%	294
15	Muqtaḍab	93.1%	90.8%	91.9%	119
16	Muḍāriʿ	94.4%	89.5%	91.9%	19
17	Prose	97.0%	95.8%	96.4%	2,668
	Average	95.1%	94.7%	94.9%	
	Weighted Average			97.3%	Total: 163,917

The model has relatively excellent precision in predicting C17-Prose (97.0%). This precision is higher than classes with much more support verses such as C2-Kāmil. Moreover, C17-Prose has relatively high recall (95.8%). These high precision and recall indicate that the model successfully classifies and detects prose from poetry.

To solve the low accuracy of meters of low support, we experimented with modifying the weight loss using techniques such as the *balanced* heuristic, which give more weight to classes of fewer samples (King and Zeng, 2001). This technique reduces the accuracy of the best classes and does not improve the macro average nor the weighted average of the F1 scores (91.4% and 96.3%, respectively).

7.2. Confusion matrix

Figure 9 shows the confusion matrix for predicting the test set. Each row in this matrix shows the model's prediction of the corresponding class. Taking the first row as an example, the model correctly classifies 99.0% of C1-Ṭawīl test verses and incorrectly classifies 0.4% of the C1-Ṭawīl test verses as C2-Kāmil, and so on. As described above, Column 14 of this matrix explains the low precision of C14-Mutadārik and Row 12 explains the low recall of C12-Madīd.

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	Ṭawīl	99	0.4	0.3	0.2	0.1	0	0	0	0	0	0	0	0	0	0	0	0
2	Kāmil	0.2	97.9	0.3	0	0.1	**0.7**	0.3	0	**0.2**	0.1	0.2	0	0	0	0	0	0
3	Basīṭ	0.5	0.4	98.7	0	0.1	0.1	0	0	0	0.1	0	0	0	0	0	0	0
4	Khafīf	0.5	0.4	0.1	96.5	0	0.3	0.8	0.4	0.5	0.1	0	0.3	0	**0**	0	0	0.1
5	Wāfir	0	0.4	0.2	0	98.8	0.2	0.2	0.1	0.1	0	0	0	**0**	0	0	0	0
6	Rajaz	0.1	**4.1**	0.9	0.2	0.1	91.6	0.5	0.1	**1.4**	0.4	0.3	0	0	0.1	0	0	0.2
7	Ramal	0.1	0.5	0.1	1.5	0.4	0.4	95.9	0	0	0.1	0.4	0.1	0.1	0.1	0	0	0.2
8	Mutaqārib	0.8	0	0	0	0.5	0.1	1	96.9	0.3	0.1	0	0	0	0.2	0	0	0
9	Sarīʿ	0.1	**2.4**	0.3	0.1	0.1	**1.9**	0.1	0.4	94.1	0.4	0	0	0	0.1	0	0	0
10	Munsariḥ	0	2	2.6	0.6	0.1	1.2	0	0	1.1	92.1	0.2	0	0	0	0	0	0
11	Mujtathth	0	1.7	0	0.1	0	2	0.3	0.2	0	0	95.3	0	0.1	0	0	**0.1**	0.2
12	Madīd	0	0.3	0.1	4.1	0.4	0	5.5	0.1	0.6	0	0	88.4	0	0.3	0	0	0.1
13	Hazaj	0	0.5	0	0	**3.2**	1	0.2	0	0	0	0.2	0	94.8	0	0	0	0.1
14	Mutadārik	0	0.7	0.3	**2**	0.3	0	0.7	1	0.7	0	0	0.3	0	93.9	0	0	0
15	Muqtaḍab	0	0.8	0	0	0	5	1.7	1.7	0	0	0	0	0	0	90.8	0	0
16	Muḍārīʿ	0	0	0	5.3	0	0	0	0	0	0	**5.3**	0	0	0	0	89.5	0
17	Prose	0.4	0.5	0.1	0.1	0.2	1.4	0.4	0.3	0.1	0.1	0.3	0	0.1	0.1	0	0	95.8

Figure 9. Confusion matrix of predicting the 17 classes

Some of the incorrect classifications shown in this matrix are explainable by the characteristics of Arabic poetry. Arabic poets often make allowed variations to the basic patterns described in Section 2 (Atiq, 1987). These variations are called ʻillaah (عِلّة) and ziḥāf (زِحاف). The main difference between the two types is ʻillaah must be applied to all poem verses whereas ziḥāf can be applied to some of the poem verses. Some of these variations make the verses of one meter indistinguishable from some other meter. The meter pairs that are affected by these variations are marked in bold and underline in Figure 9. For example, such variations make the pattern of C6-Rajaz similar to the pattern of C2-Kāmil, which may explain the 4.1% of C6-Rajaz verses incorrectly classified as C2-Kāmil. The pairs of meters that become indistinguishable with such variations are: C2-Kāmil and C6-Rajaz, C2-Kāmil and C9-Sarīʿ, C4-Khafīf and C14-Mutadārik, C5-Wāfir and C13-Hazaj, C9-Sarīʿ and C6-Rajaz, and C16-Muḍārīʿ and C11-Mujtathth (El-Katib, 1971).

7.3. Diacritics effect

As described in Section 5.4, the sample has wide range of diacritics usage. We investigate here the effect of diacritics presence on the classification accuracy. The results presented above are for the deep model trained on verses that include available diacritics and tested using verses that also include the available diacritics. The summary of these results are repeated in the first row of Table 11 for convenience. The second row illustrates that this model is adversely affected when the diacritics are removed from the input test verses. Removing diacritics from the input verses raises the error rate from 2.73% to 3.49%. This implies that the model partially relies on the presence of diacritics to extract features and accurately classify the input verse.

Table 11. Effect of verse diacritization on performance

Experiment	Model parameters	Accuracy	Error rate	Macro Accuracy
Diacritized training/diacritized input	350 k	97.27%	2.73%	94.88%
Diacritized training/undiacritized input	350 k	96.51%	3.49%	90.96%
Undiacritized training/undiacritized input	350 k	97.00%	3.00%	93.75%

We also trained the deep model on the train set verses after removing available diacritics. This model can only be tested with input verses without diacritics. The performance of this model is shown in the last row of Table 11. Although its error rate is higher than the first model with diacritized input (3.00% vs. 2.73%), it is better than the first model when the diacritics are not present at the input (3.00% vs. 3.49%). These results demonstrate that diacritics are useful for accurate classification. However, the model trained without diacritics is more robust in classifying verses when diacritics are not available.

8. Discussion

In this section, we investigate improving the classification accuracy using multiple verses from the same poem and automatically adding missing diacritics. We also compare the main results with previous work.

8.1. Classifying multiple verses

In Section 7, we showed that some errors can be attributed to the freedom that Arabic poets have in making some variations to the basic poetry patterns resulting in verses that become identical to the patterns of other meters. This insight motivated us to improve the classification accuracy by considering multiple verses of the same poem instead of one verse at a time. Remember that the results reported above are based on classifying one verse at a time even for poems that have multiple verses. Specifically, the classifier predicts the class $\hat{y}$ of the highest probability of the estimated probabilities $\hat{p}_k$ of all classes at the model output using the following equation.

$$\hat{y} = \underset{k}{\operatorname{argmax}} \ \hat{p}_k \tag{8}$$

In Section 5.2, we described that the dataset is split on poem bases to train and test sets and has 115,478 poems of varying verse lengths. Figure 10 shows the CDF of the test set according to the poem length in verses. 4.5% of these poems have one verse only, 90% have 36 verses or shorter, and 98.8% have 100 verses or shorter.

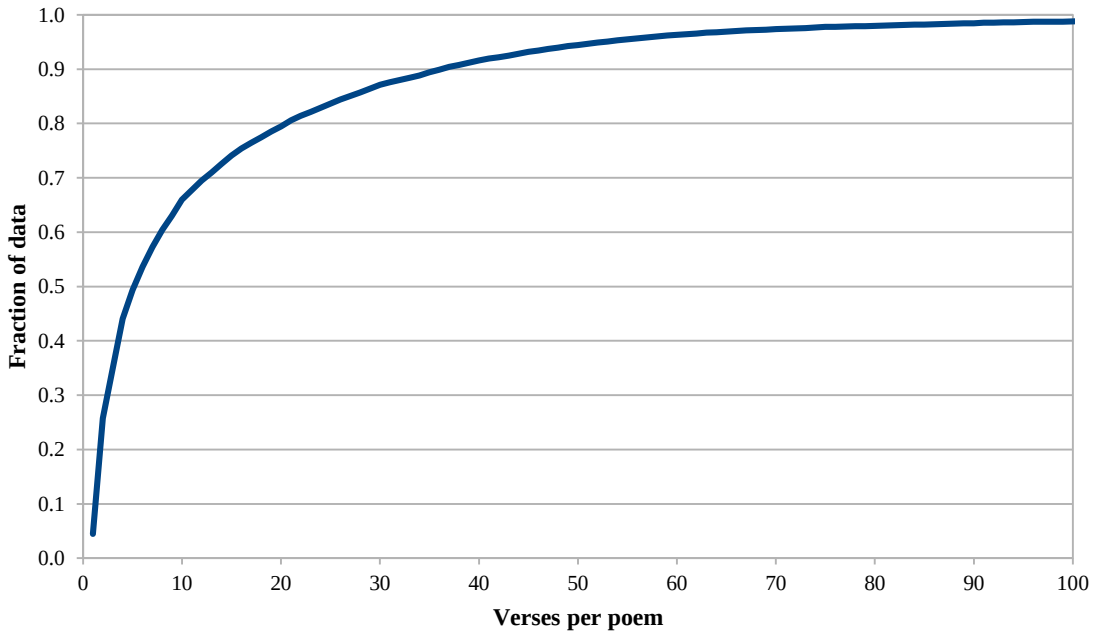

Figure 10. CDF of the test poems according to the length of the poem in verses

To exploit the above insight, we use the following equations when classifying a poem of multiple verses. We first aggregate the estimated probabilities $\hat{p}_{i,k}$ over all poem verses i, then we predict the class $\hat{y}$ of the highest aggregated probability $\hat{s}_k$.

$$\hat{s}_k = \sum_i \hat{p}_{i,k}$$
$$\hat{y} = \underset{k}{\operatorname{argmax}}\ \hat{s}_k \tag{9}$$

Figure 11 shows the accuracy as function of poem length in verses for the two deep models trained with and without diacritics. The accuracy increases rapidly when the poem length increases from one to two and from two to three. The accuracy keeps increasing slowly beyond three verses and approaches 100% for very long poems. By considering multiple verses from a poem, the average accuracy of the first model rises to 98.2%, 98.5%, 99.1%, and approaches 100% for 2, 3, 36, and larger than 110 verses, respectively. Although the second model trained without diacritics has lower accuracy for one verse (96.44% vs. 97.07%), it is generally more robust and has slightly higher average accuracy for verse lengths 1 through 36 (98.72% vs. 98.68%).

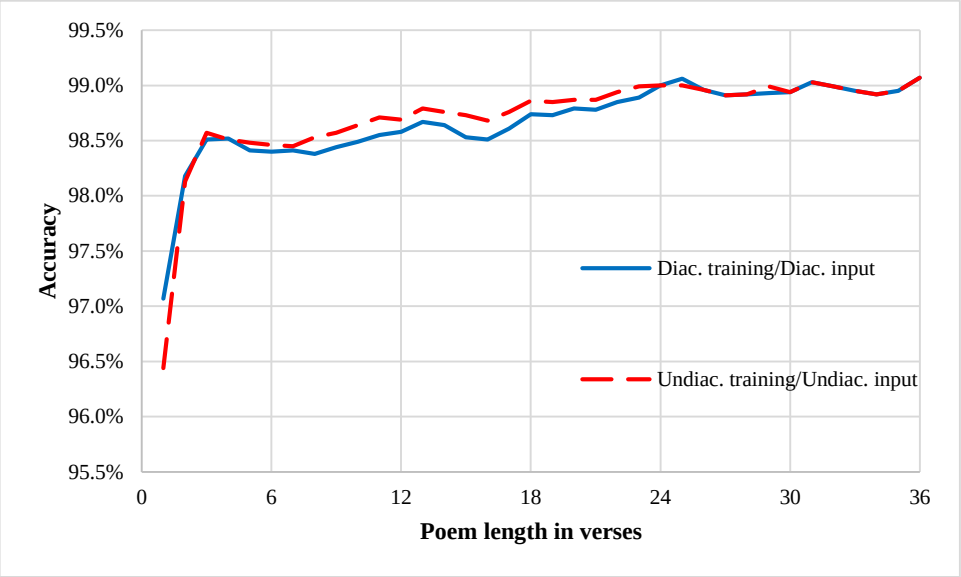

Figure 11. Accuracy as function of the poem length in verses

8.2. Classifying automatically diacritized verses

In order to improve the classification accuracy on verses without diacritics, we tried adding diacritics before classifying. We used the model described in Subsection 6.6 to automatically add diacritics before predicting the class using the deep model. Unfortunately, when diacritizing all test verses (after removing the original diacritics, when present), the classification accuracy dropped from 97.27% to 95.63%. Moreover, when diacritizing only the test verses of zero diacritics, the accuracy was not better at 96.97%. Recalling that the error rate of automatic poetry diacritization is high, we conclude that adding diacritics inaccurately does not help in classifying Arabic poetry.

8.3. Comparison with previous work

To the best of our knowledge, the classification accuracies reported in this work are significantly higher than previous work. Table 12 compares the classification results of this work with previous related work. Our proposal achieves higher accuracy than other machine learning solutions with smaller model and classifies input verses to the entire 16 poetry meters and the prose class.

Table 12. Classification results of related work and our system

System	Model parameters	Dataset size	Classes	Accuracy
Expert system (Ismail *et al.*, 2010)	Not applicable	20 poems	8 poetry	100%
Context free grammar (Alnagdawi *et al.*, 2013)	Not applicable	128 verses	16 poetry	75 %
Rule-based algorithm (Abuata and Al-Omari, 2018)	Not applicable	417 verses	16 poetry	82.2%
7-BiLSTM (Yousef *et al.*, 2019)	401 k	1,722,321 verses	16 poetry	94.11%
5-BiGRU (Al-shaibani *et al.*, 2020)	5,600 k	55,440 verses	14 poetry	94.32%
This work (4-BiLSTM)	**350 k**	**1,657,003 verses**	**16 poetry + prose**	**97.27%**
				100% for long poems

Notice that the systems listed in this table use six different datasets of various sizes because no suitable benchmark dataset was available. Therefore, flawlessly fair comparison with previous work is not feasible. However, we hope that the discussion here sheds some light about the advantages of the proposed solution. Moreover, we are proposing the APCD2 as a benchmark dataset to facilitate future comparisons in this area.

Our model uses bidirectional LSTM layers similar to the model proposed by Yousef *et al.* (2019). However, the proposed model achieves higher accuracy (97.27% vs. 94.11%). They experimented with three input data encoding techniques and considered specific network depths and widths. Instead, we present the raw input character codes to an embedding layer relying on the machine learning to find efficient data encoding. We also use carefully selected hyper-parameters, wider network (128 vs. 50 cells) and fewer LSTM layers (4 vs. 7 BiLSTM layers).

Al-shaibani *et al.* (2020) used similar architecture with the GRU cell, which is a simplified version of the LSTM, and they used a much smaller dataset. However, we think that their model is too large and is an "over kill". Our proposal uses a model 16 times smaller and is more accurate, even when trained without diacritics (97.00% vs. 94.32%).

This work achieves even higher accuracies by addressing the inherent problem of allowed pattern variation in Arabic poetry. By considering multiple verses from a poem, the accuracy increases with more verses and approaches 100% for long poems. Also, and to the best of our knowledge, this is the first work that tackles the problem of automatically adding diacritics to Arabic poetry.

9. Conclusions

In this work, we use machine learning approach to classify and diacritize Arabic poetry. We have refined the Arabic Poetry Comprehensive Dataset (APCD) and extended it to include prose samples. APCD2 includes 1,657 k verses of the 16 classical Arabic poetry meters and prose samples representing classical and modern Arabic. This dataset has a clear split to train and test subsets to facilitate comparing results of related research.

To classify input verses into the 16 poetry meters and prose, we propose a deep and narrow recurrent neural network that has an embedding layer at the input, four hidden bidirectional LSTM layers, and softmax output layer. The suggested network is well tuned and achieves an average accuracy much higher than previous work (97.27%); it also detects the entire 17 classes.

Motivated by an insight about Arabic poetry, we propose a solution that improves the classification accuracy above 97.27% and approaches 100%. Poets are allowed to have some verse variations in a poem of certain meter that may make such verses indistinguishable from another meter. When multiple verses are available from the same poem, predicting the class as the one of the highest aggregate probability overcomes these allowed variations and also some model errors.

Although diacritics are required for the correct pronunciation of Arabic text, these diacritics are often absent relying on the context and the reader's language proficiency to properly read and pronounce the text. The APCD2 dataset has a wide range of diacritics usage. The model achieves higher accuracy on diacritized verses compared with the accuracy on undiacritized verses. However, when the model is trained on the sample with diacritics removed, the model's accuracy on undiacritized verses is higher. Therefore, we recommend using the model trained on undiacritized sample for classifying undiacritized verses.

In order to improve readability and classification accuracy of undiacritized poetry verses, we investigated adding the verse missing diacritics using another recurrent neural network. It turned out that diacritizing poetry is harder than diacritizing prose. The diacritization error rate of poetry is about three times larger than that of the prose. This difficulty is expected as Arabic poets carefully and innovatively select their verses and have some freedom in diacritizing some word endings. Due to this high error rate, the accuracy of predicting the class of automatically diacritized verses is lower than that when leaving them undiacritized.

As a future work, we plan to continue providing solutions for Arabic poetry. We intend to extend the poetry classification to identify good verses and defective verses (verses with flawed deviation from the meter pattern). We also plan to develop a model that transcribes poetry verses into their respective scansions. Such solutions would help scholars and poets to analyze and improve poetry verses. We think that the automatic diacritization of poetry is important for people to properly read and enjoy poetry. We also think that the diacritization accuracy can be improved by exploiting some poem features such as meter, couplet symmetry, and the poem rhythm.

Acknowledgements

The authors would like to thank the University of Jordan and its Deanship of Scientific Research for supporting this work. We also would like to thank Waleed Yousef and his team for providing the APCD dataset and for their valuable cooperation.

References

[dataset] Abandah, G., 2020a. Classify Arabic poetry. GitHub, https://github.com/Gheith-Abandah/classify-arabic-poetry.git.

Abandah, G., 2020b. Arabic poem meters. http://www.abandah.com/gheith/Poetry/ (accessed 24 Aug 2020).

Abandah, G., Abdel-Karim, A., 2020. Accurate and fast recurrent neural network solution for the automatic diacritization of Arabic text. The Jordanian Journal of Computers and Information Technology. 6(2), 103–121.

Abandah, G., Graves, A., Al-Shagoor, B., Arabiyat, A., Jamour, F., Al-Taee, M., 2015. Automatic diacritization of Arabic text using recurrent neural networks. International Journal on Document Analysis and Recognition. 18(2), 183–197.

Abuata, B., Al-Omari, A., 2018. A rule-based algorithm for the detection of Arud meter in classical Arabic poetry. Int'l Arab J. Information Technology. 15(4), 1–5.

Al-Carthagini, H., 1966. The curriculum of the eloquent and the lamp of the Literati (in Arabic). Presentation and investigation by Muhammad Al-Habib bin Al-Khawaja. Eastern Books House, Tunisia.

Allen, J.D., Anderson, D., Becker, J., Cook, R., Davis, M., Edberg, P., Everson, M., Freytag, A., Iancu, L., Ishida, R., Jenkins, J.H., 2012. The Unicode Standard. Mountain view, CA.

Alnagdawi, M., Rashideh, H., Aburumman, F., 2013. Finding Arabic poem meter using context free grammar. J. Communication and Computer Engineering. 3(1), 52–59.

Al-Ayyoub, M., Nuseir, A., Alsmearat, K., Jararweh, Y., Gupta, B., 2018. Deep learning for Arabic NLP: A survey. Journal of Computational Science. 26, 522–531.

Al-shaibani, M.S., Alyafeai, Z., Ahmad, I., 2020. Meter Classification of Arabic poems using deep bidirectional recurrent neural networks. Pattern Recognition Letters. 136, 1–7.

Al-Talabani, A.K., 2020. Automatic recognition of Arabic poetry meter from speech signal using long short-term memory and support vector machine. ARO-The Scientific Journal of Koya University. 8(1), 50–54.

Al-Tayyib, A., 1989. The guide to understanding the poetry of the Arabs and its industry (in Arabic). Government of Kuwait Press, Kuwait.

Anees, I., 1952. Music of Poetry (in Arabic). Anglo-Egyptian Library, Egypt.

Atiq, A., 1987. Elm Al-Arud wal Qafiah (in Arabic). Dar Alnahda, Beirut, Lebanon.

Azim, A.S., Wang, X., Sim, K.C., 2012. A weighted combination of speech with text-based models for Arabic diacritization. In: 13th Annual Conference of International Speech Communication Association, pp. 2334–2337.

Azmi, A.M., Almajed, R.S., 2015. A survey of automatic Arabic diacritization techniques. Natural Language Engineering. 21(3), 477–495.

Badawi, M.M., 1975. A critical introduction to modern Arabic poetry. Cambridge University Press.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y., 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078.

Devlin, J., Chang, M.W., Lee, K., Toutanova, K., 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

El-Katib, T., 1971. The metrics of Arabic poetry using binary numbers (in Arabic). Iraqi Ports Press, Basra.

Elshafei, E., Al-Muhtaseb, H., Alghamdi, M., 2006. Statistical methods for automatic diacritization of Arabic text. In: Saudi 18th National Computer Conference, pp. 301–306.

Fadel, A., Tuffaha, I., Al-Ayyoub, M., 2019. Arabic text diacritization using deep neural networks. In: 2019 2nd International Conference on Computer Applications & Information Security, IEEE, pp. 1–7.

Géron, A., 2019. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. O'Reilly Media.

Google TensorFlow, 2020. TensorFlow. https://www.tensorflow.org/ (accessed 22 Jul 2020).

Graves, A., Mohamed, A.R., Hinton, G., 2013. Speech recognition with deep recurrent neural networks. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 6645–6649.

Hifny, Y., 2012. Smoothing techniques for Arabic diacritics restoration. In: 12th Conference on Language Engineering (ESOLEC '012), pp. 6-12.

Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural Computation. 9(8), 1735–1780.

Ismail, A., Eladawy, M., Keshk, H., Saleh, S., 2010. Expert system for testing the harmony of Arabic poetry. J. Engineering Sciences, 1, 401–411.

Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Maamouri, M., Bies, A., Buckwalter, T., Mekki, W., 2004. The Penn Arabic treebank: Building a large-scale annotated Arabic corpus. In: NEMLAR conference on Arabic language resources and tools, 27, pp. 102–109.

Margoliouth, D.S., 1925. The origins of Arabic poetry. Journal of the Royal Asiatic Society. 57(3), 417–449.

Mubarak, H., Abdelali, A., Sajjad, H., Samih, Y., Darwish, K., 2019. Highly effective Arabic diacritization using sequence-to-sequence modeling. In: 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 1, pp. 2390–2395.

Nesterov, Y.E., 1983. A method for solving the convex programming problem with convergence rate O(1/k^2). Doklady AN USSR. 269, 543–547.
Rashwan, M., Sallab, A., Raafat, H., Rafea, A., 2015. Deep learning framework with confused sub-set resolution architecture for automatic Arabic diacritization. IEEE/ACM Transactions on Audio, Speech and Language Processing. 23(3), 505–516.
Rumelhart, D.E., Hinton, G.E., Williams, R.J., 1986. Learning representations by back-propagating errors. Nature. 323(6088), 533–536.
Schuster, M., Paliwal, K.K., 1997. Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing. 45(11), 2673–2681.
Srivastava, R.K., Greff, K., Schmidhuber, J., 2015. Training very deep networks. In: Advances in Neural Information Processing Systems, pp. 2377–2385.
Sutskever, I., Vinyals, O., Le, Q.V., 2014. Sequence to sequence learning with neural networks. In: Advances in Neural Information Processing Systems, pp. 3104–3112.
The Collection: The Encyclopedia of Arabic Poetry (الديوان: موسوعة الشعر العربي), 2020. https://www.aldiwan.net (accessed 24 August 2020).
The Poetry Encyclopedia (الموسوعة الشعرية), 2020. Abu Dhabi Culture and Tourism Department. https://poetry.dctabudhabi.ae (accessed 24 August 2020).
Tieleman, T., Hinton, G., 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 4(2), 26–31.
Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I., 2017. Attention is all you need. In Advances in neural information processing systems, pp. 5998–6008.
Yousef, W.A., Ibrahime, O.M., Madbouly, T.M., Mahmoud, M.A., El-Kassas, A.H., Hassan, A.O., Albohy, A.R., 2018. Poem Comprehensive Dataset (PCD). https://hci-lab.github.io/ArabicPoetry-1-Private/#PCD (accessed 24 August 2020)
Yousef, W.A., Ibrahime, O.M., Madbouly, T.M., Mahmoud, M.A., 2019. Learning meters of Arabic and English poems with recurrent neural networks: A step forward for language understanding and synthesis. arXiv preprint arXiv:1905.05700.
Zwettler, M., 1978. Oral tradition of classical Arabic poetry: Its character and implications. The Ohio State University Press.

Appendix A. Source code

Figure 12 shows the skeleton code of our baseline model coded in Python and Keras high-level API. This is a sequential model that starts with an embedding layer. This layer embeds each of the 45 input tokens (plus the null character) into a 32-long vector. The longest expected sequence is 128 characters with null padding and this embedding layer masks out the null characters for the higher layers. The model has two BiLSTM layers; each layer has 2×128 cells and uses the dropout regularization technique to solve the overfitting problem. The output layer is a fully-connected layer that uses the softmax activation function with one output for each of the 17 classes. This model is compiled to use Adam optimizer in the backpropagation through time (BPTT) training and categorical cross entropy as loss function. The model trains using mini batches of size = 64 sequences using 15% of the training set for validation. The full source code of the final model is posted on GitHub (Abandah, 2020a).

```
model = Sequential()
model.add(Embedding(46, 32, input_length=128, mask_zero=True))
model.add(Bidirectional(LSTM(128, return_sequences=True, dropout=0.1, recurrent_dropout=0.3),
                        merge_mode='concat'))
model.add(Bidirectional(LSTM(128, dropout=0.1, recurrent_dropout=0.3), merge_mode='concat'))
model.add(Dense(17, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(train_verses, train_meters, batch_size=64, epochs=100, validation_split=0.15)
```

Figure 12. Base model coded in Python Keras