applied sciences

MDPI

Article

A Deep Belief Network Classification Approach for Automatic Diacritization of Arabic Text

Waref Almanaseer *, Mohammad Alshraideh and Omar Alkadi

King Abdullah II School for Information Technology, The University of Jordan, Amman 11942, Jordan; mshridah@ju.edu.jo (M.A.); o.alkadi@ju.edu.jo (O.A.)
* Correspondence: warefalmanaseer@gmail.com

Abstract: Deep learning has emerged as a new area of machine learning research. It is an approach that can learn features and hierarchical representation purely from data and has been successfully applied to several fields such as images, sounds, text and motion. The techniques developed from deep learning research have already been impacting the research on Natural Language Processing (NLP). Arabic diacritics are vital components of Arabic text that remove ambiguity from words and reinforce the meaning of the text. In this paper, a Deep Belief Network (DBN) is used as a diacritizer for Arabic text. DBN is an algorithm among deep learning that has recently proved to be very effective for a variety of machine learning problems. We evaluate the use of DBNs as classifiers in automatic Arabic text diacritization. The DBN was trained to individually classify each input letter with the corresponding diacritized version. Experiments were conducted using two benchmark datasets, the LDC ATB3 and Tashkeela. Our best settings achieve a DER and WER of 2.21% and 6.73%, receptively, on the ATB3 benchmark with an improvement of 26% over the best published results. On the Tashkeela benchmark, our system continues to achieve high accuracy with a DER of 1.79% and 14% improvement.

Keywords: deep learning; Deep Belief Network; Restricted Boltzmann Machine; natural language processing; arabic diacritization

Citation: Almanaseer, W.; Alshraideh, M.; Alkadi, O. A Deep Belief Network Classification Approach for Automatic Diacritization of Arabic Text. *Appl. Sci.* **2021**, *11*, 5228. https://doi.org/10.3390/app11115228

Academic Editor: Malik Yousef

Received: 20 April 2021
Accepted: 25 May 2021
Published: 4 June 2021

1. Introduction

Arabic is one of the six official languages of the United Nations (UN), which belongs to the Semitic languages used by Arabs and Muslims all over the world. The estimated Arabic speaking population in the world is around 400 million native speakers and 1 billion Muslims with 30 different dialects [1]. The Arabic language alphabet consists of 28 letters in addition to the Hamza. The orientation of writing in Arabic is from right to left, there is no capitalization in Arabic and Arabic letters change shape according to their position in the word [2].

In the Arabic language, diacritic marks are used to clarify how to pronounce a letter. These diacritics are written either above or below a letter within a word as shown in Table 1. Diacritization is important since it removes word ambiguity. For example, the word دَرَسَ it is pronounced "darasa", means studied and the word دَرْسٌ it is pronounced "darsun" and means a lesson. Both words have the same characters, however, adding the correct diacritic marks presents two different meanings. The meaning of a sentence is greatly influenced by the diacritization which is determined by the context of the sentence. Text diacritization helps children and non-native speakers to learn Arabic properly.

The automation of diacritizing Arabic text is involved with finding an efficient method to automatically diacritize Arabic text for different applications such as speech processing applications. In order to remove the many levels of word ambiguity arisen of incorrect diacritization or absence of diacritic marks in writing, lemmatizer tools have been used to to alleviate this phenomenon, however, it is considered a stressful task to obtain a high quality

lemmatizer [3]. Automatic diacritization enhances the performance of many applications in accordance with their accuracy and speed of processing and can be beneficial in other Arabic processing steps such as Part-of-Speech (POS) tagging. Nevertheless, diacritization suffers low accuracy as indicated in prior works [4]. This problem is considered challenging because many words in Arabic have different meanings subject to their diacritization [5] and correct diacritization requires analyzing the full sentence as it takes into account the context of the sentence to diacritize words.

Table 1. Primary Arabic diacritics.

Name	Shape	Sound	Unicode
Fatha	بَ	/a/	064E
Damma	بُ	/u/	064F
Kasra	بِ	/i/	0650
Fathatan	بً	/an/	064B
Dammatan	بٌ	/un/	064C
Kasratan	بٍ	/in/	064D
Sukoun	بْ	None	0652
Shadda	بّ	Doubling	0651

Natural Language Processing (NLP) is defined as "an area of research and application that explores how computers can be used to understand and manipulate natural language text or speech to do useful things" [6]. Applications of NLP include a number of interesting areas including sentiment analysis, machine translation, information retrieval, speech recognition and expert systems. Many machine learning algorithms have been used to provide an improved performance to NLP applications since the accuracy of some of the NLP applications is very crucial.

It is worth understanding some basic aspects of text processing because high-quality text processing is the key to creating robust NLP applications and therefore, improve the way it is indexed or linked to other resources. Arabic NLP is gaining a lot of attention by researchers and the particular reason for this is that the Arabic language is being widely used in different applications and services. Processing text in the Arabic language and Arabic NLP applications present researchers and developers with serious challenges due to the linguistic structure, as it is considered a highly structured language that depends on derivatives and morphology [2].

To make the language more usable, many NLP tasks were applied to the Arabic language. Over the last decade, Arabic have begun to gain ground in the area of research within NLP. Much work targeted different aspects related to how this language is processed, such as: morphological analysis [7], lemmatization [3], text categorization [8], documents summarizing [9], word segmentation [10], sentiment analysis [11] and automatic diacritization [12].

Automatic text diacritization is essential for a range of applications including Text-to-Speech (TTS) [13], Automatic Speech Recognition (ASR) [14], machine translation [15], information retrieval [12] and much more. These applications have been integrated with deep learning to provide robust performance [16].

Roughly speaking, deep learning is a machine learning method based on neural network architectures with multiple layers of processing units that provide a hierarchical representation of the data by means of various convolutions. In deep learning, the neural networks have various (deep) layers that enable learning. The more deep learning algorithms learn, the better they perform. Deep learning has been widely used in different fields and has made great progress in the areas of computer vision, speech recognition and NLP [17]. The most substantial breakthrough in deep learning was in 2006, when Hinton and Salakhutdino [18] introduced the Deep Belief Network (DBN) with multiple

layers of Restricted Boltzmann Machines (RBMs). In general, a DBN is a probabilistic generative model that is pretrained using a greedy layer-wise method. In this greedy layer-wise algorithm, the DBN learns one layer at a time in an unsupervised way, and then undergoes fine-tuning via supervised learning with backpropagation. DBNs were introduced as a solution for the dilemmas encountered when using traditional neural networks trained in deep architectures, such as slow-paced learning, becoming stuck in local minima, and demanding a lot of training data for ameliorate learning.

Deep learning is a rich family of methods, encompassing Convolution Neural Network (CNN), Auto-Encoders (AE), Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), and Generative Adversarial Nets (GAN). While deep learning is extensively used in NLP, only few studies have applied it to Arabic text diacritization. These studies mostly use a hybrid system of deep learning and statistical or rule-based approaches. For example, in [19], which is recently published research, a multi-components system for automatic Arabic diacritization was proposed, the system uses an LSTM with rule-based corrector.

This work provides the first attempt to implement and examine the performance of the DBN to automatically diacritize Arabic text. Unlike previous studies, this work does not employ any prior morphological or contextual analysis nor requires post-processing of data.

The remainder of this paper is organized as follows: Section 2 provides a review of related work from literature on Arabic text diacritization. Section 3 provides a detailed background on the main concepts addressed in this work. Section 4 describes the dataset used. The proposed methodology is detailed in Section 5. Section 6 outlines the experimental settings and the results. Finally, Section 7 summarize the findings of this work.

2. Literature Review

Many approaches were used to tackle the diacritization problem; however, it is still an open research problem that needs more work to improve its accuracy [20]. Diacritization techniques are classified into three categories: rule-based, statistical systems and hybrid systems. In literature, two standard evaluation metrics are used to measure the accuracy and performance of these systems [4,21]: Diacritization Error Rate (DER) and Word Error Rate (WER). DER is concerned with finding the proportion of letters which are incorrectly diacritized, while WER finds the percentage of incorrectly diacritized words. The smaller the error rate, the better the performance.

2.1. Rule-Based Systems

Rule-based systems formulate a set of rules that are used to derive a solution for a problem. Human knowledge is utilized in this kind of approaches in the form of morphological analyzers, dictionaries and grammar modules. Rule-based techniques were used to develop a morphological analyzer and a syntax analyzer for Arabic [22]. Furthermore, well-formed rules were used to detect diacritics in Modern Standard Arabic (MSA) scripts [23]. In this work, the relation between each word with its part of speech is taken into account as well as the relation between a word and its position in the sentence. The WER reported by this work was 9.36%.

Metwally et al. [24], proposed a multi-layered approach to diacritze Arabic text. The authors embed their knowledge of Arabic syntactic rules in order to decrease WER. Experimental results show that the system achieves a syntactic WER of 9.4%.

Alserag is a rule-based diacritizer proposed by Alansary [25]. The system is based on three components in order to provide fully diacritized Arabic words, these components are: morphological analysis, syntactic analysis and morph-phonological processing. Alserag depends on two resources; the Arabic diacritized dictionary and a set of linguistics rules which guide the diacritization process. The system was evaluated on the LDC Arabic Treebank dataset and has a WER and DER of 8.68% and 18.63%, respectively.

2.2. Statistical Systems

Statistical approaches assign probabilities to sequences of words and characters based on some statistics such as their frequency in the dataset [12]. A hidden markovian model and Viterbi search algorithm were used in [26] to automatically find the optimal diacritic marks of a sentence. Randomly selected Quranic verses were used to test the proposed approach. The testing resulted with a DER of 4.1%.

An n-gram statistical language model was used in [27]. The author uses n-gram as a smoothing technique that improves the accuracy of diacritization which is affected by the data sparse problem.

Nelken and Shieber [28] also used a statistical approach in their proposed system for Arabic diacritization. Specifically, they used a trigram language model for words and letters. The proposed system achieves a 7.33% word accuracy without case ending and 23.61% with case ending.

2.3. Hybrid Systems

On the other hand, hybrid techniques have gained a lot of interest by NLP researchers seeking to improve the diacritization problem for Arabic language. Hybrid techniques combine the rule-based techniques with statistical techniques [12]. For instance, Shaalan et al. [13], proposed a hybrid approach to build an Arabic diacritizer. The diacritizer uses a lexicon and a Support Vector Machine (SVM). Experimental results showed that statistically based methods are promising in addressing the ambiguity resolution problem in Arabic language diacritization.

Darwish et al. [29], used a Viterbi decoder and a support vector machine to guess the diacritics of words and their case endings. Their experimentation results in a 3.29% and 12.77% WER without and with case endings.

Mijlad et al. [30] proposed a hybrid system that uses a deep LSTM network and Alkhalil Morpho Sys 2 analyzer [31]. The analyzer associates each word with its morphological-syntactic features. The purpose of using the analyzer is to take the word out of its context and provide all possible diacritization forms for it along with its POS. At the same time, the LSTM is trained to diacritize input text. Next, undiacritized words by the LSTM are selected with their context and further associated with their corresponding diacritized forms and POS outputted by Alkhalil Morpho Sys 2.

Rashwan et al. [32] used a deep learning framework with the Confused Sub-set Resolution (CSR) method to improve the classification accuracy of diacritics and Arabic PoS tagging using deep neural networks. The authors report a syntactic WER of 9.9% on the LDC ATB3 corpus.

Abandah et al. [33] proposed a sequence transcription approach for the automatic diacritization of Arabic text using RNN. The authors used a bidirectional LSTM network that builds high-level linguistic abstractions of text and exploits long range context. This approach does not require any lexical, morphological, or syntactical analysis to be performed on the data before being processed by the RNN. For experimentation, the authors used LDC ATB3, the simple version of the holy Quran and ten books drawn from the Tashkeela dataset. Results achieved were as follows: for Tashkeela books the DER and WER were 2.09% and 5.82%, respectively. For LDC ATB3 DER and WER were 2.72% and 9.07%, respectively.

Vergyri and Kirchhof [34] studied the effect of combining acoustic information with morphological and contextual constraints to automatically diacritize letters with missing diacritics in a text for use in acoustic model training for ASR systems. Experimental results achieved a DER and WER of 11.5% and 27.3%, respectively.

Zitouni et al. [14] proposed a maximum entropy based approach that exploits the lexical, segment-based and part-of-speech features to build a diacritization model. The authors formulate the problem of diacritics restoration as a sequence classification problem such that every character in a sequence is assigned a diacritic mark. The model was trained and tested using LDC ATB3. The approach achieves DER and WER of 5.5% and 18%, respectively.

Habash and Rambow [35] extend the use of their morphological analyzer to generate all the possible analysis of a word. An SVM was used to help narrow down the list of generated words and a language model was constructed to select the best solution from the list. This approach was trained and tested using LDC ATB3 dataset. The experiments achieve DER and WER of 4.8% and 14.9%, respectively.

Rashwan et al. [21] used a hybrid system of language factorizing and un-factorizing textual features. In order to infer the most likely diacritization and phonetic transcription of the input text, different diacritization system rely on factorizing text using morphological analysis and case diacritics and then statistically recapitulate the most likely sequence using lattice search. In this work, the most likely sequence is selected using the n-gram probability estimation and A* lattice search. The reported DER and WER of this approach using LDC ATB3 are 3.8% and 12.5%, respectively.

Another hybrid approach was proposed in [4]. This hybrid approach combines both rule-based and data- driven techniques to diacritize MSA text and lattice of analyses, using automatic correction, morphological analysis, POS tagging and out of vocabulary diacritization components. The authors developed a lightweight statistical morphological analyzer that is trained and tested on LDC ATB3 corpus. The POS tagger is concerned with selecting the most likely sequence of analyses produced from the previous step. The POS tagger disambiguate this lattice and selects the most likely diacritized form for the word using Hidden Markov Models (HMM) and Viterbi algorithm. Their approach achieved DER and WER of 3.6% and 11.4%, respectively.

Fadel et al. [36] did a critical review for the currently existing systems, measures and resources for Arabic text diacritization. Moreover, the authors made available a clean benchmark dataset that can be used for this problem. The data is mined from the Tashkeela Corpus and it consists of 55K lines comprising about 2.3 M words. The results of this review showed that Shakkala outperformed the best performing rule-based approaches, primarily Mishkal and Harakat, with DER and WER of 3.73% and 11.19%, respectively.

Mubarak et al. [37] implemented a character level sequence-to-sequence model consisting of an encoder, decoder LSTM RNN and attention mechanism. The model was trained on a fixed length sliding window of n words. The diacritization of a word is selected using a voting mechanism to pick the most common diacritized form of a word in different contexts. It is worth mentioning that the authors used the freely available WikiNews corpus of 18,300 words to test their model however, they do not identify their training data or refer to its source. The DER is 1.21% and the WER is 4.49%.

Abandah and Abdel-karim [38] used four bidirectional LSTM layers to diacritize Arabic text. The authors proposed alternative design and data encoding options to achieve a fast and accurate solution for Arabic text diacritization. Experiments were done using ATB3 and Tashkeela datasets. The best achieved DER is 2.46% and 1.97% for ATB3 and Tashkeela, respectively.

Alqahtani et al. [39] used Temporal Convolutional Neural Networks (TCN) and a variant of TCN called Acausal TCN (A-TCN) for diacritization in three different languages: Arabic, Yoruba, and Vietnamese. A-TCN allows the model to learn from preceding and proceeding context. TCN was deployed in this work as a character-level sequence model because such models are able to better generalize to unseen data compared to word-based models [40]. The authors experimented with LDC's ATB3 corpus and reported a DER of 3% and WER of 10.2% for Arabic language.

In [19], Abbad and Xiong propose a multi-component architecture to address the problem of automatic Arabic text diacritization. The first component of their architecture is a multi-layered LSTM, the second is a rule-based corrector that exclude any impossible daicritization on the output according to linguistic rules. Finally, the last component is a word-level correcter that relies on the text and the distance information to fix diacritization errors. The experimental results of the proposed system on the Tashkeela dataset achieve a DER of 3.39%, WER of 9.94% and on the LDC ATB3 dataset a DER of 9.32%, WER of 28.51%.

3. Preliminaries

3.1. Artificial Neural Networks and Deep Architectures

Artificial Neural Networks (ANN) are a branch of machine learning that seek to learn tasks in order to solve problems and make decisions by simulating the behavior of the human brain. McCulloch and Pitts (1943) were the first to model the human neuron [41]. After then, ANN have emerged in many disciplines such as classification, clustering, pattern recognition and prediction due to its powerful self-learning and adaption

ANNs have an input layer and output layer. Between these two layers there are other hidden layers that perform the mathematical computations. The layers of the ANN are comprised of interconnected processing units that work together to learn and perform very simple tasks, called neurons [42]. The ANN gains its power from the numerous interconnections between these neurons. Each neuron in the network is able to receive input signals, to process them, and to send an output signal. Each connection link between neurons is associated with a weight that has information about the input signal [43].

The simplest form of an ANN has input vector $x = (x_1, x_2, ..., x_n)$, weight vector w, bias b, and $h_{w,b}(x)$ is the output of the neural network. As the neurons learn from the training data, each of w and b changes [44]. Each time the ANN runs, the weighted average values of the vector is recalculated. This result is then passed into an activation function. The role of the activation function is highly important as it allows the network to learn complex patterns in the data, and it is responsible for determining whether a neuron should be activated or not.

As mentioned earlier, each time the ANN runs the weights are recalculated or in other words fine-tuned. Fine-tuning the weights of a neural network based on the error rate obtained is known as Backpropagation. Backpropagation is the learning algorithm concerned with training multi-layer perceptrons. Such algorithm helps to adjust the weights of the neurons to get a more accurate output by computing the gradient of the loss function [45]. Applying gradient descent to the loss function helps find weights that achieve lower and lower error values, making the model gradually more accurate.

The simplest form of neural networks are the Feed-Forward Neural Networks (FFNNs). Its name indicates the flow of input data; the input travels in one direction (forward) passing through the input nodes, then through the hidden nodes (if present), and finally exiting through the output nodes [46]. This actively demonstrates that there is no "feedback" from the outputs of the neurons towards the inputs throughout the network. Depending on the number of the layers, FFNNs are categorized into two types, either "single layer" or "multi-layer" [47]. Each node at every layer computes the sum of the inputs weight and bias, and transfer this sum through an activation function such as sigmoid function to acquire the output [48]. The training process of a neural network focuses on minimizing some error value achieved via a cost function defined as a Mean Squared Error (MSE) or a Sum of Squared Error (SSE).

As mentioned earlier, the backpropagation algorithm is one of the learning algorithms accountable for adjusting the neural network weights with the objective of minimizing the network's error. Backpropagation is a popular supervised learning algorithm used for training multi-layer FFNNs. The main principle of backpropagation is to train a multi-layered neural network to model a function that reflects the internal representations of data by adjusting the model's parameters (weights and biases) to produce the expected output. Technically speaking, backpropagation computes the gradient of the loss function with respect to the weights of the network for a single input—output tuple [49].

The training algorithm consists of two phases [50,51]. In the first phase, known as the forward phase, where weights are randomly initialized, and the input data are propagated forward through the network, layer-wise. The forward phase terminates after computing the output error [52]. In the second phase, called the backward phase, the calculated error is propagated backward through the network. Then the network parameters are adjusted such that the error is minimized and the neural network learns the training data.

Despite its popularity, this gradient-based training algorithm is highly subject to getting stuck into local minima [49], and is susceptible to parameter settings and to the network initial weights, biases and architecture [53]. Therefore, evolutionary algorithms were proposed by researchers as alternatives to gradient-based methods for training FFNNs. Evolutionary algorithms are proved to be more efficient in escaping from local optima for optimization problems. Evolutionary algorithms are appealing for ANN training since they maintain a population of solutions during search in a search space, enabling extreme exploration and massive parallelization.

Evolutionary neural networks, or "neuroevolution", which utilize evolutionary algorithms to optimize neural networks have been used to train FFNNs [54]. One of the earliest methods for training FFNNs was using the Genetic Algorithm (GA) [55]. Results showed that GA excel the backpropagation algorithm when applied on a classification problem. This contribution of neuroevolution was followed by many other similar contributions that employ nature inspired algorithms. For example, in [56] Mirjalili et al. proposed a hybrid Particle Swarm Optimization (PSO) and Gravitational Search Algorithm (GSA) to train FFNNs. Ant Colony Optimization (ACO) [57], Multiverse Optimizer (MVO) [58], Cuckoo Search (CS) [59], Artificial Bee Colony (ABC) [60], Grey Wolf Optimizer (GWO) [61], and many more nature inspired algorithms were adopted for the sake of training FFNNs and overcoming the flaws of backpropagation.

Just as the human brain processes information in multiple stages of transformation that reflect a deep architecture of the organization of the brain cortex, neural networks have developed to act similarly. This is done via training deep multi-layer neural networks. The architecture of such multi-layered neural networks consists of neural nets with many hidden layers. In the case of a multi-layer neural network, depth corresponds to the number of (hidden and output) layers. A neural network with deep architecture is known as Deep Neural Networks (DNN), and training them is called as deep learning.

Deep learning is a branch of artificial intelligence that attempts to develop the techniques that will allow computers to handle complex tasks such as recognition and prediction at high performance [62]. Deep learning is able to learn more abstract features in the higher levels of the representation as well as generalizing expressive data representations [63]. As mentioned earlier, deep learning usually occurs in two phases: first, unsupervised layer-wise training and second, supervised training. In the unsupervised phase, each layer is added and trained greedy manner. Each layer uses the representation learned by the previous layer as input that it tries to model and transform to a new and better representation.

3.2. Deep Belief Network (DBN)

Although backpropagation provided a fairly efficient way of learning multiple layers of nonlinear features, it has difficulty optimizing the weights in deep networks that contain many layers of hidden units and it requires labeled training data, which is often expensive to obtain.

DBNs overcome the limitations of backpropagation by using unsupervised learning to create layers of feature detectors that model the statistical structure of the input data without using any information about the required output. High-level feature detectors that capture complicated higher-order statistical structure in the input data can then be used to predict the labels. DBNs is one of the most important tools for deep learning constructed from RBMs. RBMs have an efficient training procedure which makes them suitable as building blocks for DBNs.

3.2.1. Restricted Boltzmann Machine (RBM)

RBMs are probabilistic graphical models that can be interpreted as stochastic neural networks that can learn a probability distribution over its set of inputs. RBMs are a variant of Boltzmann machines, with the restriction that their neurons must form a bipartite graph. A bipartite graph is a graph whose vertices (V), can be divided into two independent

sets, V_1 (visible units) and V_2 (hidden units), and every edge of the graph connects one vertex in V_1 to one vertex in V_2. These two sets may have a symmetric connection between them, and there are no connections between nodes within the same group [64] as shown in Figure 1.

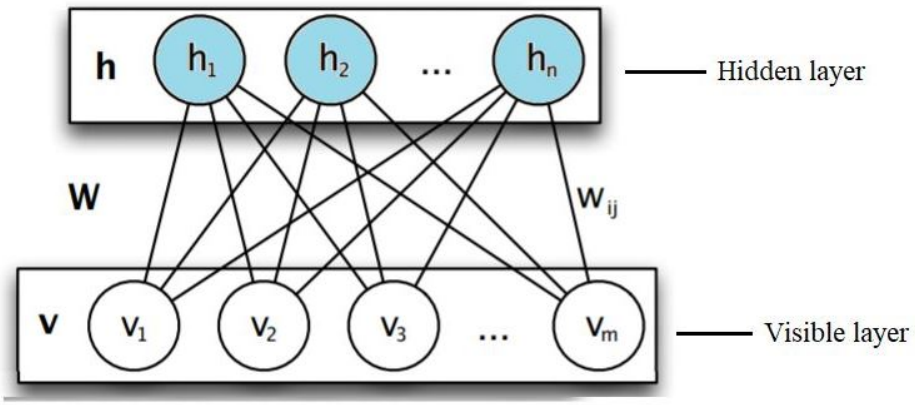


Figure 1. The structure of RBM.

An ordinary RBM accepts binary-values for the visible and hidden units. This type of RBM is known as Bernoulli-Bernoulli RBM. The Bernoulli distribution is a discrete distribution having two possible outcomes labelled by $n = 0$ and $n = 1$. If $n = 1$ it means that the true value occurs with probability p and when n = 0 it means the false case occurs with probability $q = 1 - p$, where $0 < p < 1$ [65].

RBM is an energy-based model, it consists of n visible units and m hidden units, vector v and h represent the state of visible and hidden units respectively. Given a set of state (v, h), the energy of an RBM system is defined as [66]:

$$E(v,h) = -\sum_{i=1}^{n} a_i v_i - \sum_{j=1}^{m} b_j h_j - \sum_{i=1}^{n}\sum_{j=1}^{m} v_i W_{ij} h_j, \tag{1}$$

where v_i represents the state of the ith visible unit, and h_j represents the state of the jth hidden unit. W_{ij} represents connection weights of visible and hidden units. There are also bias weights (offsets) a_i for the visible units and b_j for the hidden units.

When the parameters are defined, we can find the joint probability distribution of (v, h) in terms of the energy function as:

$$P(v,h) = \frac{1}{Z} e^{-E(v,h)} \tag{2}$$

$$Z = \sum_{v,h} e^{-E(v,h)}, \tag{3}$$

where Z is a normalization constant. When the state of a visible unit is given, the activation states of each of the hidden units are conditionally independent. Thereupon, the probability of activation of the jth hidden unit is:

$$P(h_j = 1|v) = \sigma\left(b_j + \sum_{iv_i} W_{i_j}\right), \tag{4}$$

where $\sigma(x) = 1/(1 + e^{(-x)})$ is a logistics sigmoid activation function. Similarly, given a specific hidden state, h, the activation states of each visible unit are also conditionally independent and the probability of the ith visible units of v given h is obtained by:

$$P(v_i = 1|h) = \sigma\left(a_i + \sum_{jh_j} W_{i_j}\right). \quad (5)$$

Differentiating a log-likelihood of training data with respect to W is computed as follows:

$$\frac{\partial log p(v)}{\partial W_{i_j}} = \left\langle v_{ih_j} \right\rangle_{data} - \left\langle v_{ih_j} \right\rangle_{model}, \quad (6)$$

where $\langle . \rangle_{data}$ and $\langle . \rangle_{model}$ indicate expected values in the data or model distribution. The learning rules for weights of the network in the log-likelihood-based training data can be obtained as:

$$\Delta W_{i_j} = \epsilon\left(\left\langle v_{ih_j} \right\rangle_{data} - \left\langle v_{ih_j} \right\rangle_{model}\right), \quad (7)$$

where ϵ is the learning rate. Since there are no direct connections in the hidden layer of an RBM, we can get an unbiased sample of $\langle \mathrm{v}_i h_j \rangle_{data}$ easily. Unfortunately, it is difficult to compute an unbiased sample of $\langle \mathrm{v}_i h_j \rangle_{model}$ since it requires exponential time. To avoid this problem, a fast learning algorithm, called Contrastive Divergence (CD) [67], is proposed by Hinton [68]. CD sets visible variables as training data. Then the binary states of hidden units are all computed in parallel using Equation (4). Once the states have been chosen for the hidden units, a "reconstruction" is produced by setting each v_i to 1 with a probability given by Equation (5). In addition, weights are also adjusted in each training pass as follows:

$$\Delta W_{i_j} = \epsilon\left(\left\langle v_{ih_j} \right\rangle_{data} - \left\langle v_i h_j \right\rangle_{recon}\right). \quad (8)$$

$\langle v_i h_j \rangle_{data}$ is the average value over all input data for each update and $\langle v_i h_j \rangle_{recon}$ is the average value over reconstruction; it is considered as a good approximation to $\langle v_i h_j \rangle_{model}$.

3.2.2. DBN Structure

The DBN is a neural network constructed from many layers of RBM that form a stack of RBMs as shown in Figure 2. By stacking RBMs in this way, we can learn a higher level representation of input data. DBNs were recently proposed by Hinton et al. [18], along with an unsupervised greedy learning algorithm for constructing the network one layer at a time. DBN, is presented as the state of the art of ANN in their traditional forms with network topologies built from layers of neuron models but with more advanced learning mechanics and deeper architecture, without modeling the detailed biological phenomena constituting human intelligence.

In practice, the DBN training often consists of two steps: (1) greedy layer-wise pre-training and (2) fine-tuning. Layer-wise pretraining involves training the model parameters layer by layer via unsupervised training and CD algorithm [69]. In this initial step the training starts by the lower-level RBM that receives the DBN inputs, and gradually moves up in the hierarchy, until finally the RBM in top layer, containing the DBN outputs, is trained. Therefore, learned features or output of the previous layer is used as the input of the subsequent RBM layer [70].

In fine-tuning, as a final step after the training of each RBM, the network can be trained in a supervised way using backpropagation algorithm in order to "fine-tune" the weights. This greedy learning problem-solving approach of DBN is quick and efficient. It involves making the optimal choice at each layer in the stack of RBMs, eventually finding a global optimum as each layer consistently trained to get the optimum value [71].

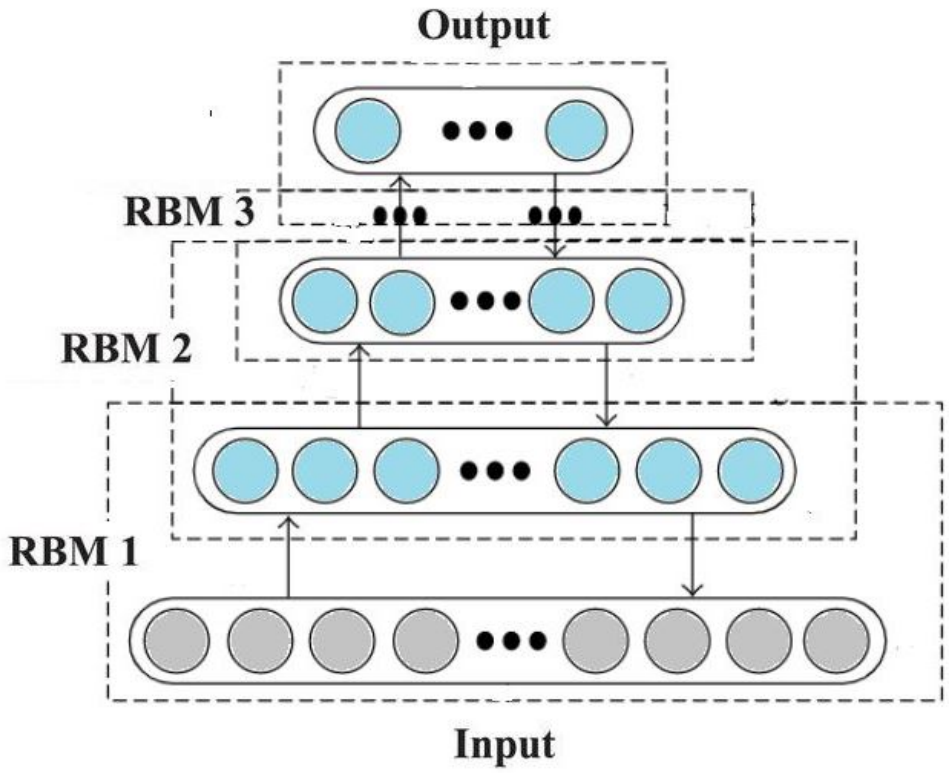


Figure 2. The DBN structure.

4. Dataset

The training, validation and testing data are drawn from the Tashkeela and LDC Arabic Tree Bank part 3 (ATB3) corpuses. The Tashkeela corpus consists of 97 books written in Classical Arabic (CA) style and 293 books written in Modern Standard Arabic (MSA) style. The files are compiled from books, articles, news, speeches and school lessons. The original Tashkeela corpus has over 75.6 million words, where about 67.2 million are diacritized Arabic words. For the purposes of this study and to facilitate comparisons with previous systems, we used ten books of the Tashkeela corpus. These books are summarized in Table 2.

Table 2. Dataset statistics.

Book ID	Book Name	Size (K Words)	Used (K Words)	Letters Per Word	Words Per Sentence	Zero Diacritics (%)	One Diacritic (%)	Two Diacritics (%)
1	Alahaad Walmathany	241	24	3.78	6.01	43.1	52.6	4.3
2	Majma Aldamanat	218	114	4.04	14.25	21.1	74.6	4.3
3	Sahyh Muslim	494	188	3.81	21.01	26.4	67.8	5.8
4	Alqawaed Labn Rajab	169	127	4.12	16.20	20.9	74.2	4.9
5	Alzawajer An Aqtiraf Alkabaer	284	261	3.94	9.57	21.6	72.3	6.1
6	Ghidaa Alalbab	316	281	3.99	9.28	21.9	72.2	5.9
7	Aljawharah Alnayyrah	385	201	3.99	22.77	20.7	74.1	5.2
8	Almadkhal Lilabdary	361	293	4.05	13.68	21.1	73.1	5.8
9	Durar Alhokam	646	375	3.83	24.22	21.5	73.2	5.3
10	Moghny Almohtaj	1306	838	3.93	9.63	20.5	73.9	5.6
11	LDC ATB3	305	225	4.64	11.31	39.8	54.8	5.4
12	Children stories	26	26	3.2	5.0	27.5	56.2	16.3
	Average	396 K	246 K	3.94	13.58	25.5	68.3	6.2

LDC's Arabic Tree Bank (ATB) consists of 599 distinct newswire stories collected from different news agencies and newspapers, including the Agency France-Press (AFP), Al-Hayat, and An-Nahar newspapers. The dataset, comprising about 305,000 words, is written in MSA style. LDC's ATB3 data statistics are provided in Table 2. These books vary in their sizes, letters per word, and words per sentence, as well as the number of non-diacritized letters which may have none, or one diacritic or even two diacritics. we choose not to select random portions of sentences from the books, rather we experimented

with all fully diacritized words in sentences. Except for book 1 since it has a high percentage of non-diacritized letters, therefore we took 10% of its content. As can be noticed from Table 2, the total number of words collected from all the datasets experimented with in this work are about 5 million words. After preprocessing and removing non-diacritized words this number is reduced to about 3 million words. The fourth column in Table 2 shows the total number of words used after preprocessing.

5. Methodology

The only requirement to train in this architecture is a human-readable, fully diacritized Arabic text without any additional morphological or syntactic information.

The proposed approach undergoes several steps to automatically diacritize Arabic text using DBN. These steps are presented in Figure 3. The dataset is initially partitioned into training, validation and testing. The training dataset is a sample of data used for learning, that is to fit the model. The validation set is used to validate the trained model; it provides an unbiased assessment of a model fit and tune the hyperparameters of the model. Finally, the testing dataset represents the sample of data used to evaluate the performance of the optimal model. As can be seen from Figure 3, the first step is to clean and preprocess the dataset to ensure that the dataset is consistent, not corrupted and free of errors. The second step is data encoding where in this step, an encoding scheme is used to transform the data into a form that is appropriate for the algorithm to process. In the third step, Borderline-SMOTE algorithm is applied to over-sample the input data in order to solve the class imbalance found in the training data. The data is now ready to be input to the DBN to start the training and validation process. The result of training yields the best validated model where the testing dataset will be input to. Finally, the output which is fully diacritized sentences is evaluated against some measures to indicate its performance. The details of these steps are described in the following subsections.

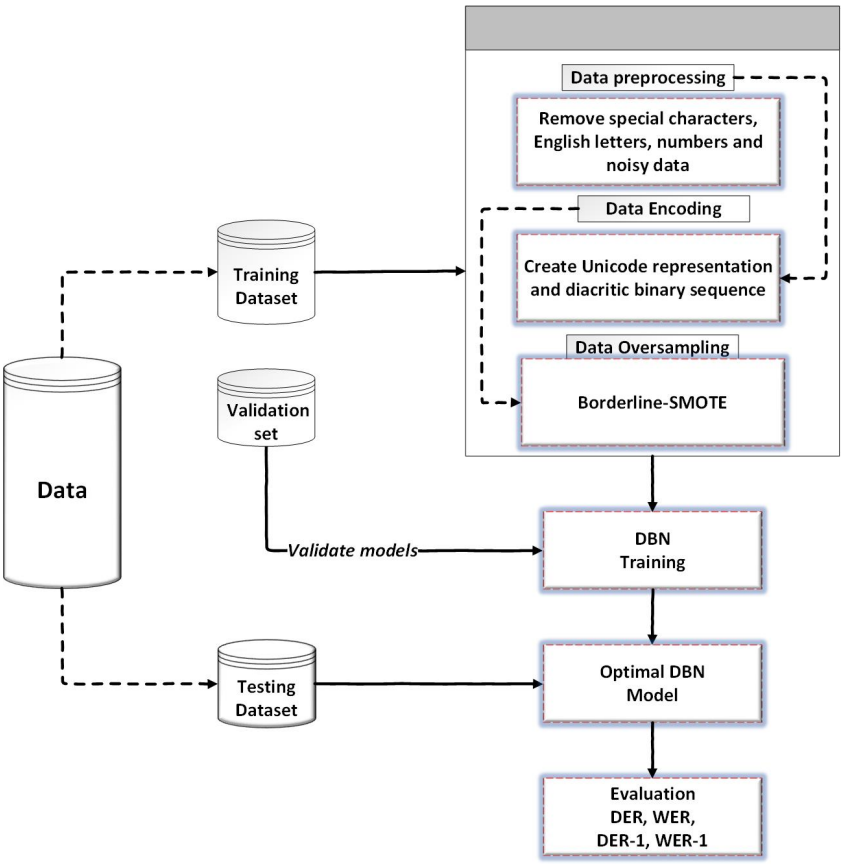


Figure 3. The main steps followed in the proposed methodology.

5.1. Data Cleaning and Preprocessing

One of the key components of the success of training a deep learning model is the preparation of data. Most of the time and effort is usually put into this stage as it critically affects the following stages of learning. Feeding a deep learning model clean and normalized data grant proper learning. This section describes the steps followed to prepare our data for training.

Data preprocessing in machine learning refers to the technique of preparing (cleaning and organizing) the raw data to make it suitable for building better predictive machine learning models. Typically, real-world data is incomplete, inconsistent, inaccurate (contains errors or outliers), and often lacks specific attribute values/trends.

There are different ways to preprocess text data and it is most of the time language dependent. In Arabic language, preprocessing tasks may include stop-words removal, tokenization, normalization and morphological analysis as well as eliminating punctuation marks and foreign symbols from the text. To unify the structures of our data because they came from different sources, we performed a few preprocessing operations. Firstly, any special characters, English letters and numbers are removed from text. Noisy data are also removed such as incomplete words that have missing letters and scattered letters in the text. Extra spaces were removed from each sentence as well.

In every text file, paragraphs are split into sentences having one sentence per line. A paragraph is split into more than one sentence at the following ending punctuation marks: : ‘.’, ‘:’, ‘،’, ‘«’, ‘»’, ‘؟’, ‘؛’, ‘[’, ‘]’, ‘{’, ‘}’. After that, the lines with length more than 10 words are split into lines of length no more than 10. This way we can limit memory usage, and with this length we preserve the semantic structure of the text. The positions of the diacritics were unified, where each diacritic was inserted directly after the corresponding letter. When Shaddah and other diacritics were used, the sequence was unified, where the Shaddah came first, followed by the other diacritics.

Normalization of Arabic letters is a common preprocessing step when dealing with Arabic text. Normalization aims to normalize certain letters that have different forms in the same word to one form. In this step, letters ‘أ’, ‘إ’ and ‘آ’ are replaced with ‘ا’ while the letter ‘ة’ is replaced with ‘ه’ and the letter ‘ى’ is replaced with ‘ي’. Moreover, we remove extra white spaces and Tatweel symbols. For example, the word كبيـــــر (kabīr ‘big’) has Tatweel, whereas (كبير) has no Tatweel.

Most of the preprocessing operations in this work were carried out using regular expressions in Python. A Regular Expression (RegEx) is a highly specialized programming language embedded inside Python [72]. RegEx uses a sequence of characters that defines a search pattern to check if a particular string matches a given regular expression. Regexes are commonly used for parsing text and searching text files for a precise pattern [73]. Likewise, regular expressions were written in this work to remove special characters and noisy data as well as normalize specified letters into a single form.

5.2. Data Encoding

As mentioned earlier, a DBN is a stack of RBMs that process the input data at each layer to discover hidden features of the input. Arabic input text (characters and diacritics) is encoded in the Unicode code block 0600-06FF. In order to be processed by RBMs that constitute the DBN model trained in this work, the Unicode of each character is mapped to a bit code of 11 bits; 4 bits mask (1100) and 7 bits for the character. For example, the word الكون which means (the universe) is represented as follows:

$$\underbrace{\mathbf{1100}1000110}_{\text{ن}}\ \underbrace{\mathbf{1100}1001000}_{\text{و}}\ \underbrace{\mathbf{1100}1000011}_{\text{ك}}\ \underbrace{\mathbf{1100}1000100}_{\text{ل}}\ \underbrace{\mathbf{1100}0100111}_{\text{ا}}$$

After having the data prepared, we take the clean diacritized text and create a sequence that represents the diacritics class of every letter in every word. That is, each word

has its own diacritic binary sequence of length 13 because we have 13 diacritic class consisting of the primary 8 diacritics listed in Table 1, in addition to the Shadda mark associated with the other diacritic as shown in Table 3. Then, we remove all diacritics from text and create a Unicode representation for the undiacritized text. Eventually, we have Unicode representation and diacritic binary sequence that correspond to our input. Figure 4 provides an example that illustrates this operation using the word لَوْ which means "if" in the English language.

Table 3. Diacritics marks class sequences.

Diacritic Mark	Binary Class Sequence	Class Number
بَ	1000000000000	0
بُ	0100000000000	1
بِ	0010000000000	2
بْ	0001000000000	3
بً	0000100000000	4
بٌ	0000010000000	5
بٍ	0000001000000	6
بّ	0000000100000	7
بَّ	0000000010000	8
بُّ	0000000001000	9
بِّ	0000000000100	10
بّْ	0000000000010	11
بًّ	0000000000001	12

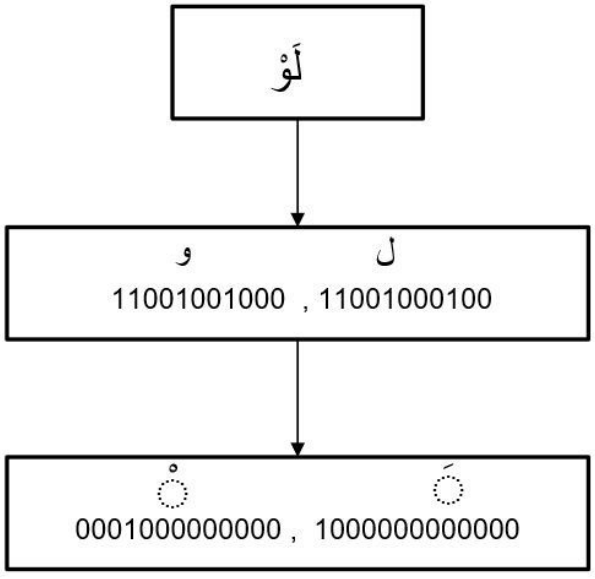


Figure 4. mapping sequence.

5.3. Data Oversampling

The problem of class imbalance is an essential difficulty in the construction of learning systems. A dataset is called class imbalanced when the numbers of examples representing each class are not equal [74]. The class with more examples is typically denoted as the majority class, whereas the underrepresented class is called the minority class. The issue of

class imbalance may seriously cause a negative impact on the accuracy of the learner [75]. To tackle this problem, oversampling method is used. Oversampling generates synthetic examples to balance the distribution of data. SMOTE, Borderline-SMOTE, ADASYN and random oversampling are examples of popular oversampling techniques.

In this work, class imbalance exists among diacritic marks that only appear at the end of the word. These diacritics are "Fathatan, Dammatan, Kasratan" and "Shadda" with the former mentioned diacritics. Therefore the distribution of there appearance differs from the other classes of diacritics that may appear at any position on the character in a word. For example, "Fatha" diacritc mark may appear at the beginning, middle or end of the word whereby "Shadda with Dammatan" may only appear at the end of the word.

To solve this problem, we used Borderline-SMOTE. It is a popular extension to SMOTE which involves selecting those instances of the minority class that are mis-classified (near the borderline), such as with a k-nearest neighbor classification model [76].

5.4. Training with DBN

This section describes the deep learning model designed to solve the automatic diacritization problem and how the DBN training occurs to generate diacritic marks. The task is formulated as a sequence classification such that we predict a diacritic for each character in the input.

Because DBN has strong feature detection and extraction capabilities, DBN is often used for predictive classification problems. The classification DBN contains the top-level associative memory that is actually a classification RBM. This allows for training the top layer to generate class labels corresponding to the input data vector.

Classification DBNs require the observation labels to be available during training of the top layer, so a training session involves first training the bottom layer, propagating the data through the learned RBM, and then using that new transformed data as the training data for the next RBM. This continues until the data has been propagated through the second last trained RBM, where the labels are concatenated with the transformed data and used to train the top layer associative memory. As previously explained, the learning process of DBN is divided into two phases: unsupervised learning and supervised learning. The propagation of data from the input layer to the top layer is the unsupervised learning process, while for the contrary, it is supervised learning process. An initial goal for a DBN is to propagate a different representation of the data to each stacked RBM. In theory, this will allow each successive RBM to learn more abstract features in the data. When DBN training is completed, we use labeled data to fine-tune all the parameters. The process of fine-tuning is to achieve a global minimum for the loss function of the whole model.

5.4.1. Automatic Diacritization Using DBN

The function of any classification algorithm is to transfer the input data X_t to a certain output Y_t. For our DBN, the input layer v is the text samples, and the output of h_n is the feature learning results produced by DBN. Given that automatic diacritization of Arabic text is a sequence-to-sequence problem, the classification RBM at the top layer holds the diacritic marks (i.e., labels) thus it generates a sequence holding the corresponding label for each input data X_t.

As demonstrated in Figure 5, the binary representation of input text is received by the first RBM in the bottom layer (i.e., RBM1). RBM1 works on the received input text and learns hidden features. Next, RBM1 propagates what it has learned to the subsequent RBM on the next hidden layer, that is RBM2. RBM2 receives the output of RBM1 and treats it as its input. Now RBM2 is ready to learn hidden features and propagates data to the next layer. At each RBM layer, learned features of the previous layer is used as the input of the subsequent RBM layer with each higher layer representing higher-level abstraction of the input data. This greedy layer-wise training is conducted until reaching the highest hidden layer which is RBM3.

Next, the second training stage begins, fine-tuning, which consists of fine-tuning the weights and supervised learning at the top layer. In this stage, a new layer is added on top of the DBN, specifically, after RBM3 and removes the links in the top to down direction. Labeled data are used to train the new layer, which acts as a classifier. As shown in Figure 5, the labels are provided on top and include the diacritic marks listed in Table 3. The backpropogation algorithm is used to learn the network weights and biases based on labeled training examples. The trained weights and biases that are fine-tuned in each layer correspond to features at different layers. Here, the learning goal is to minimize the classification error given the labeled examples. Since the features obtained in the unsupervised learning stage captures the properties of the data, they can be saved for future classifications. After the supervised learning and backpropagation to train a DBN for our classification problem, it is used to predict the diacritic mark for each character.

Many sequence transcription problems, including diacritization, require the exploitation of future context. Future context can be exploited by maintaining long-term dependencies, however, not all deep learning models are capable to keep long-term dependencies and memorize them. Taking into account the potentials of DBNs to reconstruct input through layering of RBMs, each RBM transfers to the next layer what it has learned from the complete input and taking it to the next level. At each level, the RBM will dig deeper once it receives its input from the previous layer and reprocess it again to generate its own, consequently, full context will be covered once it reaches the top.

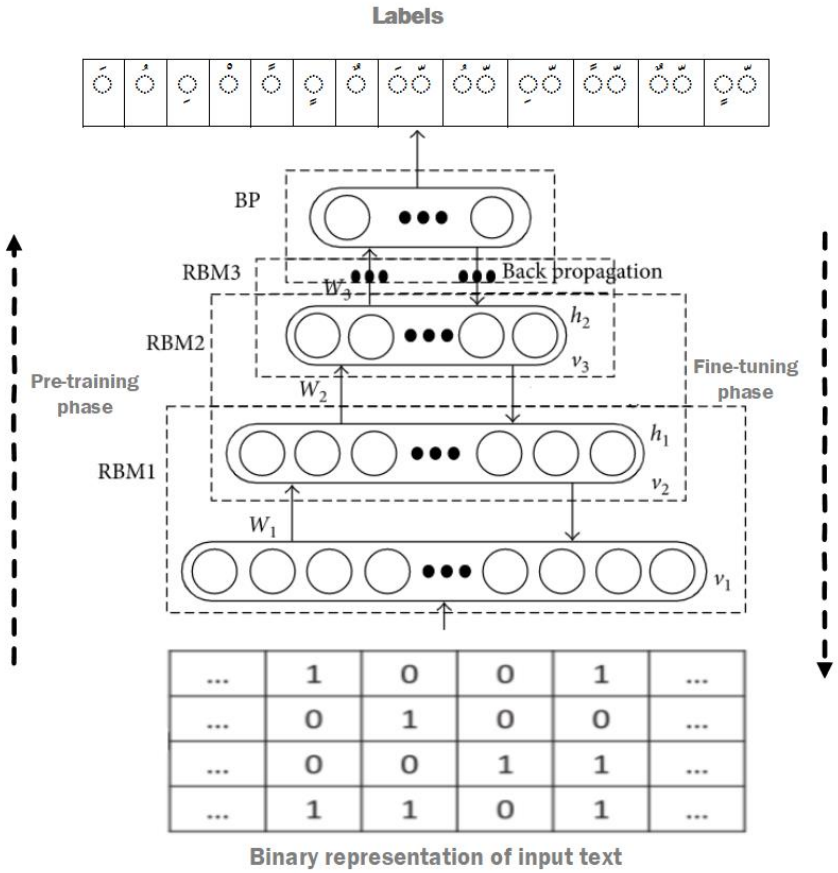


Figure 5. A schematic representation of the designed DBN model for the automatic diacritization of Arabic text.

5.4.2. Rectified Linear Unit (ReLU) Activation Function

Activation functions are the core of deep learning. These functions usually facilitate deep neural networks by introducing non-linearity to the learning process. The non-linearity feature gives the neural network the ability to succeed in training and learning complex patterns [77]. Training a deep network with saturated activation functions has been experimentally proved to be hard. Nowadays, non-saturated activation functions are one of the key factors contributing to the success of the modern deep learning models.

Formally, ReLU function is defined as in Equation (9). Where x_i is the input and y_i is the output from the activation function. The function returns 0 if it receives a negative input, and for a positive value x_i it returns that value back.

$$y_i = \begin{cases} x_i & \text{if } x_i \geq 0 \\ 0 & \text{if } x_i < 0. \end{cases} \tag{9}$$

ReLU was first used in RBMs to improve their performance [78]. In [79] it was shown that adding ReLU to the hidden layers of the network could improve the learning speed of the deep network. Using ReLU, the vanishing gradient problem was avoided, this is due to its ability to make the gradients at the positive values become constants and do not vanish any more.

ReLU is known for its simplicity and light computation as there is no complicated math. The advantage of these characteristics allow the model to train and run in a relatively short time. Another advantageous characteristic of ReLU is its sparsity. Since ReLU gives output zero for all negative inputs, it is likely for any given unit to not activate at all which causes the network to be sparse. In a sparse network it is more likely that neurons are actually processing valuable parts of a problem [80]. For example, a model that is processing images of the universe may contain a neuron that is specialized to identity stars. That specific neuron would not be activated if the model was processing images of satellites instead. It is worth mentioning that sparsity results in concise models that often have better predictive power and less noise [81].

5.5. Evaluation Metrics

Throughout our experiments, we evaluate the accuracy of the DBN using the Diacritization Error Rate (DER). DER is the percentage of characters with incorrectly predicted diacritics. We also report the Word Error Rate (WER) of our best performing model. WER is the percentage of incorrectly diacritized words. A word is considered incorrectly diacritized if it has at least one incorrectly diacritized letter.

- Diacritization Error Rate (DER): which is the ration of characters with incorrectly restored diacritics. DER can be calculated as follows:

$$DER = (1 - T_S/T_G) \times 100\% \tag{10}$$

  where T_S is the number of letters assigned correctly by the system, and T_G is the number of diacritized letters in the gold standard text.
- Word Error Rate (WER): the percentage of incorrectly diacritized white-space delimited words. At least one letter in the word must have a diacritization error so that it can be counted as incorrect.

$$WER = (1 - T_W/T_G) \times 100\% \tag{11}$$

  where T_W is the number of words fully and correctly diacritized by the system, and T_G is the number of diacritized words in the gold standard text.

6. Experiments and Results

6.1. Experimental Settings

The challenge of training deep learning neural networks involves carefully selecting and configuring the hyperparameters such as the learning rate, the number of hidden layers, dropout, batch size and the number of epochs. These hyperparameters are pivotal since they directly control the training and learning process, hence having important impact on the performance of the model being trained. Table 4 summarizes the selected values for these parameters in this work. The results of conducted experiments demonstrated that the model performance is poor with a small batch size. With regard to the run-time of the model with different batch sizes, the model with the small batch size requires a relatively longer run-time. Therefore, selecting the appropriate batch size can improve the performance of the DBN model and shorten the run-time. The selected batch size in this study is 256.

Table 4. Parameters configuration of RBM layers and DBN.

Parameter	RBM Layers	DBN
Epochs	30	200
Batch	256	-
Learning rate	0.05	0.1
Dropout	-	0.2
Number of nodes	250	-

For training and testing purposes, the LDC ATB3 benchmark dataset was used as a single set for training and testing as previously recommended by the founders of the benchmark [14], such that 70% of the data was allocated for training and 30% for testing. We also follow our own approach of data split to avoid having an optimized DBN towards the test set by having a separate development and test set. The data was randomly shuffled and divided into a training set containing 80% of the sentences, and 20% for validation. Afterwards, the DBN is tested with an unseen nor trained with test set. On the other hand, using the K-fold cross-validation approach for partitioning data and validating the DBN model not only significantly increased the training time but also resulted in poor performance over the folds. Therefore, we adopt the previously discussed split.

The development platform used in this work was Google's Colaboratory (Colab). Colab is a free Jupyter notebook environment that runs entirely in the cloud and grants users with powerful hardware such as GPUs and TPUs [82]. It also supports many popular machine learning libraries and executes python code. The processor granted was an Nvidia Tesla T4 GPU CUDA Version:11.2., and the deep learning framework was developed on the Tensorflow back-end.

6.2. RBM Structure

The DBN is constructed from a stack of RBMs. In this work, and after a number of experiments, we found the best architecture of the DBN with three hidden layers of RBMs. We have also considered 30 epochs for the RBM learning procedure with mini-batches of size 256.

As shown in Figure 6, the reconstruction error decreases with the increase in the number of RBM epochs. The first RBM begins with the lowest reconstruction errors, however, while training the other two layers outperform it. The reason for this behaviour is because the number of nodes in the visible layer of the first RBM is large and the input data is sparse. The two other RBM layers have only a slight increase in error at the first 5 epochs and then decreases gradually in the following epochs. It is clear from Figure 6 that as the number of RBM layers increase the reconstruction error tends to decrease and stabilizes. Thus, it is concluded that the deeper the number of RBM layers is, the smaller

the reconstruction error will be. Therefore, the optimal number of hidden layers for this model is 3.

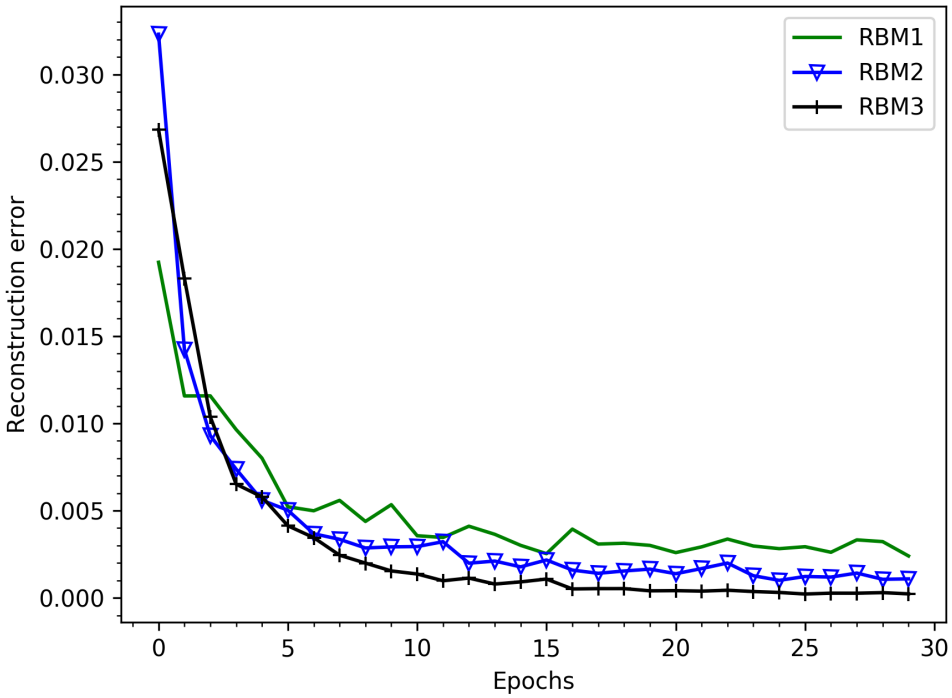


Figure 6. RBM reconstruction error.

6.3. Weight Noise Regularization

Deep learning neural networks are likely to quickly over-fit a training dataset with minimal examples. To solve this problem to get improved generalization performance, weight noise regularization is used. Specifically, dropout regularization method is used. Dropout temporarily removes visible and hidden units and their connections in a neural network. It has been known that dropout is a simple architectural way used to prevent neural networks from over-fitting. In the case of RBM, it was proven in literature that dropout RBMs are better than standard RBMs [83].

A number of experiments were conducted to show the performance of the DBN with and without dropout. In all the experiments we have conducted, dropout method exhibit outstanding performance. Therefore, we adopt this option in all the experiments. The best dropout probability was 0.2.

Figure 7 shows the effect of training the DBN with dropout and without dropout. The performance was evaluated against the DER metric for four selected books from the Tashkeela books. This adaptation reduces over-fitting and improves the generalization of deep neural networks.

6.4. Datasets Training

The dataset size is an important factor that affects the performance of the DBN. Acquiring more training data allows the model to learn and perform better. Therefore, we have conducted several experiments to study the effect of changing the training data size on the performance. The experiments are done on the basis of individually training a DBN for each book separately and on the training of a single DBN for all books combined into a single dataset.

It can be seen from Figure 8 that the DER decreases as the training set size increases. The average DER when training a DBN using all training sets is 1.79, which is lower than the average DER for the separate DBNs training with an average DER of 2.98. The reason for such improvement between the two approaches is that each RBM layer in a DBN learns

the entire input once received. In other kinds of models like convolutional nets early layers only detect simple patterns and later layers recombines them.This lets RBMs with more feature detector units to have a high chance to learn information from the data distribution, which is consequently affected by the amount of data received.

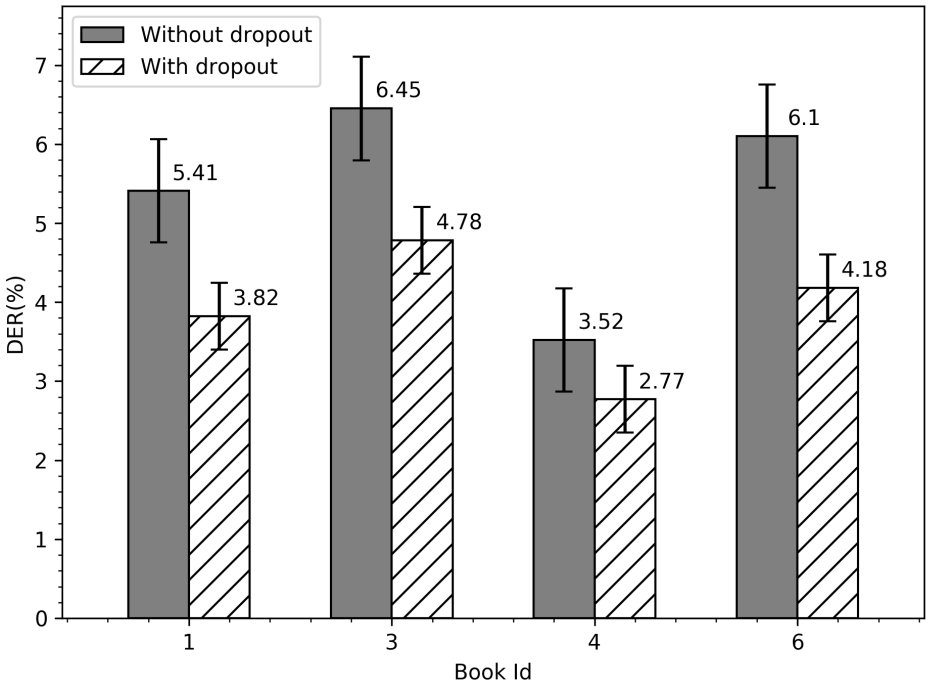


Figure 7. The reflex of dropout method on the training data.

According to conclusions found in the literature, the ATB3 dataset was better if trained alone since the differences are large between CA and MSA. Further, it was recommended that different networks should be used with each type. Therefore, in this work the ATB3 dataset was trained independently.

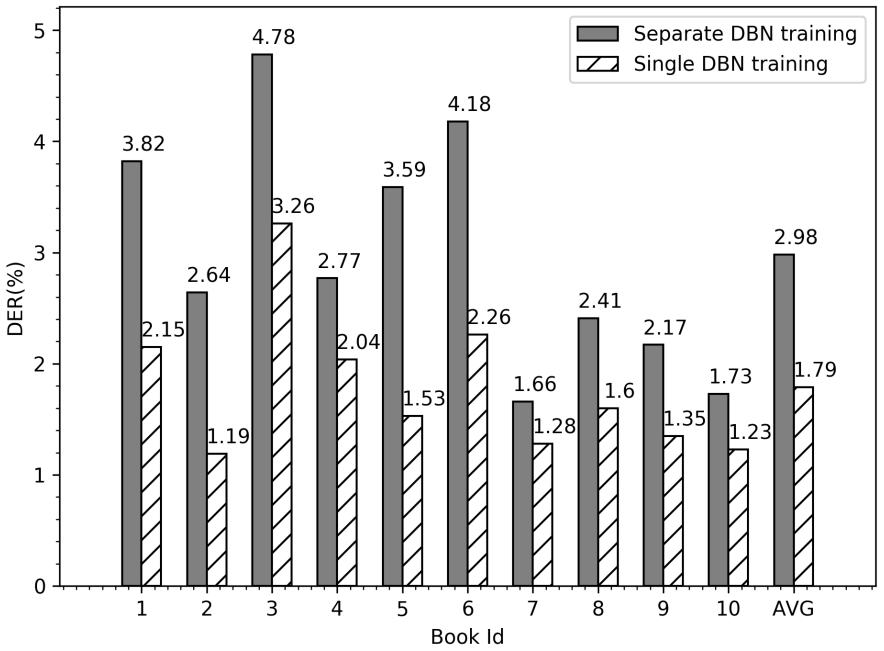


Figure 8. DER results for separate DBN and single DBN training of datasets.

6.5. Comparisons with Literature

In this subsection, we present our results on ATB3 and Tashkeela datasets by comparing them to the best published systems. To the best of our knowledge, Abandah and Abdel-Karim [38], is the system that has the best reported accuracy on the same datasets used in our work. Table 5 provides a summary of our results along with the best published systems in terms of DER, WER, DER-1 and WER-1.

Table 5. Diacritization results of previous systems and our system.

System	Dataset	All Diacritics		Ignore Last	
		DER	WER	DER-1	WER-1
Nelken & Shieber (2005) [28]	ATB3	12.8	23.6	6.5	7.3
Zitouni et al. (2006) [14]	ATB3	5.5	18.0	2.5	7.9
Habash & Rambow (2007) [84]	ATB3	4.8	14.9	2.2	5.5
Schlippe et al. (2008) [85]	ATB3	4.3	19.9	1.7	6.8
Alghamdi et al.(2010) [86]	ATB3	13.8	46.8	9.3	26.0
Rashwan et al. (2011) [21]	ATB3	3.8	12.5	1.2	3.1
Said et al. (2013) [4]	ATB3	3.6	11.4	1.6	4.4
Abandah et al. (2015) [33]	ATB3	2.72	9.07	1.38	4.34
Alqahtani et al. (2019) [39]	ATB3	2.8	8.2	-	-
Abandah & Abdel-Karim (2019) [38]	ATB3	2.46	8.12	1.24	3.81
Abbad & Xiong (2020) [19]	ATB3	9.32	28.51	6.37	12.85
This work	ATB3	**2.21**	**6.73**	**1.2**	**2.89**
Abandah (2015) [33]	Tashkeela	2.09	5.82	1.28	3.54
Barqawi (2017) [87]	Tashkeela	3.73	11.19	2.88	6.53
Abandah & Abdel-Karim (2019) [38]	Tashkeela	1.97	5.13	1.22	3.13
Fadel et al. (2019b) [36]	Tashkeela	2.60	7.69	2.11	4.57
Abbad & Xiong (2020) [19]	Tashkeela	3.39	9.94	2.61	5.83
This work	Tashkeela	**1.79**	**4.63**	**1.15**	**2.13**

To the best of our knowledge, Abandah et al. bidirectional LSTM based systems in [33,38] has the best reported accuracy's on the ATB3 and Taskeela datasets. In terms of the ATB3 dataset our DBN model provides 19% DER improvement and 26% WER improvement compared to Abandah's et al. system in [33]. And a 10% DER improvement and 17% WER compared to the system proposed in [38].

Looking at Table 5, it can be observed that the DER is also decreasing in this work compared to the best published systems presented in the table. For the Tashkeela dataset, our system provides 14% DER improvement and 20% WER improvement compared to Abandah's system in [33].

From Table 5, it is recognized that, over the past several years, diacritization performance has been improved overall and our system has greatly improved the performance of its predecessor while continuing its trend.

In 2015, Abandah et al. (2015) [33,38] reported the best accuracy on ATB3 and Tashkeela datasets. For these two datasets, we report a higher improvement on error rates as mentioned earlier. Recently, over the past 2 years, Fadel et al. (2019) [36], Alqahtani et al. (2019), and Abbad and Xiong (2020) [19] have reported interesting results on the ATB3 and Tashkeela datasets. Still, our results outperform their reported results in terms of DER, WER, DER-1 and WER-1. Our system provides 19% DER improvement compared to Abandah el al. (2015), 18% WER improvement compared to Alqahtani et al. (2019), and 76% DER improvement compared to Abbad and Xiong (2020) regarding the ATB3 dataset. Figures 9 and 10 present the results of using our DBN model and the Bi-LSTM proposed in [33] on the Tashkeela corpus. Apparently, the DER and the WER of the DBN model outperform the competing model of Bi-LSTM. The reason DBN works so well is related to the fact that the pre-training phase in an unsupervised fashion of DBN improves

model performance by avoiding over-fitting and enhancing the model generalization. Furthermore, a DBN exploits hidden features and their context layer by layer.

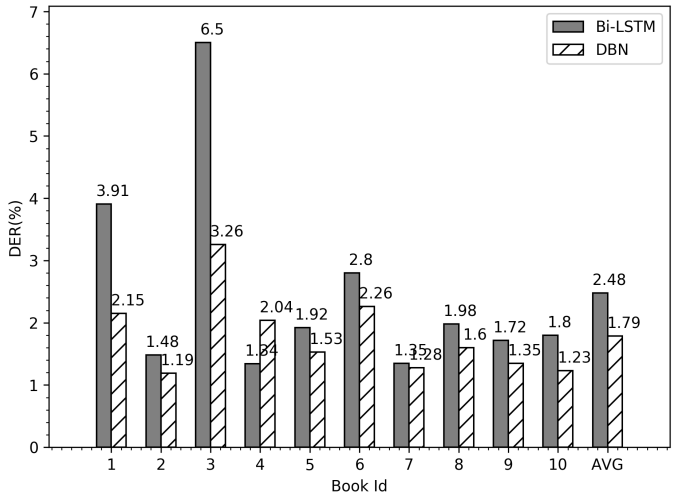


Figure 9. DER results using DBN vs. Bi-LSTM on the Tashkeela books.

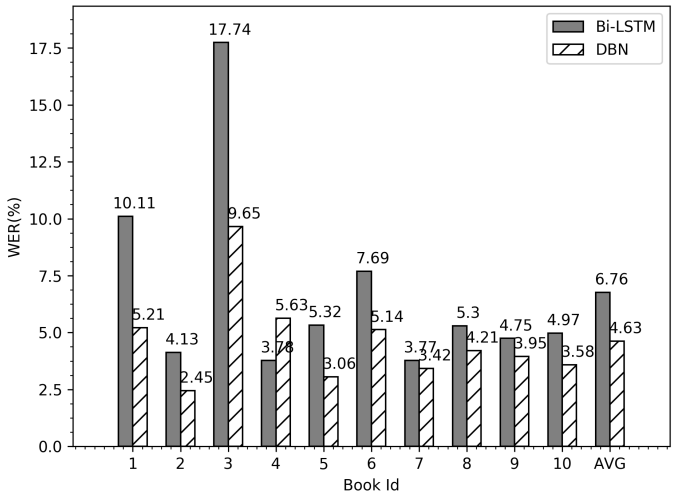


Figure 10. WER results using DBN vs. Bi-LSTM on the Tashkeela books.

The diacritic of the last character of the word usually depends on the POS tag making it harder to diacritize and harder to get right because it is determined by the syntax. Therefore, ignoring the diacritization of the last letter of every word results in lower error rates and this is the approach followed in almost every research in this domain. It is thus usual to report a variant of the above two mentioned metrics that ignore the last letter (assumed to have no syntactic diacritics), denoted as DER-1 and WER-1. Figures 11 and 12 present the error rates for the Tashkeela books.

According to the results provided in Figures 11 and 12 we can note that the general behavior of the error rates is as expected, which is decreasing. Once the diacritic mark of the last character is ignored the error rate improves. The average DER drops from 1.79% to 1.15% and WER from 4.63% to 2.13%. An improvement of DER and WER of 36% and 54%, respectively.

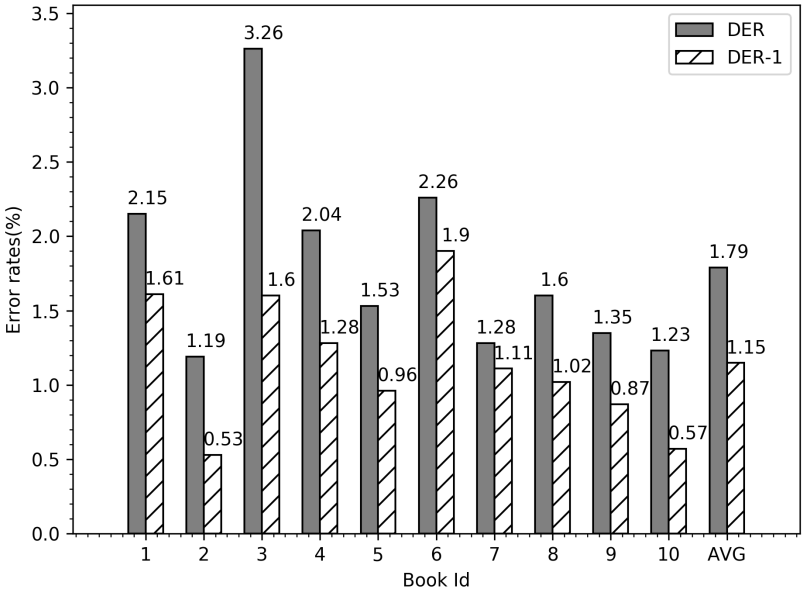


Figure 11. DER and DER-1.

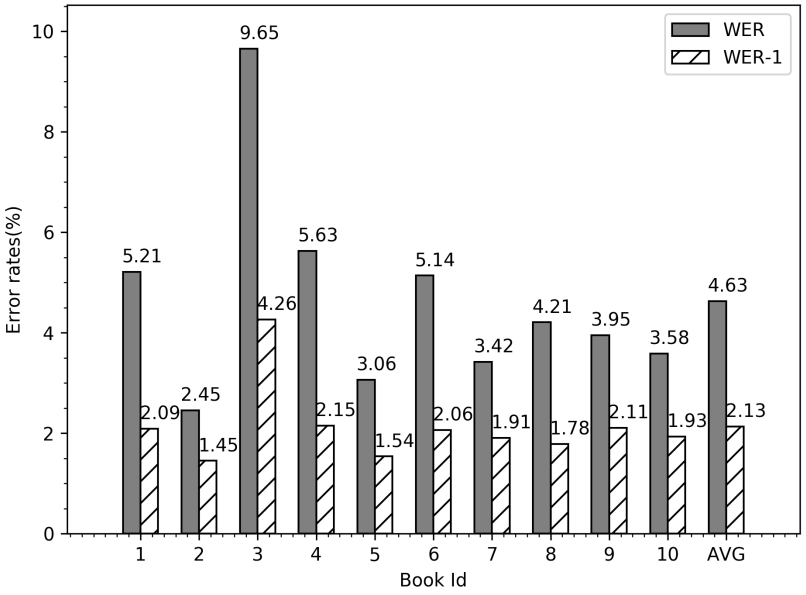


Figure 12. WER and WER-1.

6.6. Probability Distribution of Diacritics

As explained previously, the building blocks of a DBN are RBMs and each RBM hidden layer learns a probability distribution over an input dataset presented to the visible layer. So, a DBN can learn to probabilistically reconstruct its inputs whereby its layers act as feature detectors. After this learning step, a DBN is trained with supervision to perform classification. The objective of the DBN used here is to extract in depth features and patterns in the original data in order to diacritize Arabic text. At the end of that, the DBN should label each input with the correct class label.

Figure 13 provides an example of a correctly diacritized word from the children stories corpus. The word is "أُحَلِّقُ" which is equivalent to the word "fly" in English. The word is 4 characters long, and each character has a diacritization mark. As can be seen from the figure, there are probabilistic (P) values setting between the input and the class label for that input.

In simple words, the DBN assigns a probability of its certainty of the class label associated with that input. The higher the probability, the more certain the DBN is and the higher the accuracy. For example, The DBN model assigns an activation probability of (p = 0.77) for class 1 on the first input character, meaning that it has identified the diacritization mark "Damma" with a certainty level of 77% for the first input character which is 'أ'. The second input character of the provided word 'ح' is classified by the DBN as class 0 with activation probability of (p = 0.94). Which implies that the model is 94% certain that the diacritization mark for this input in this context is "Fatha". The same logic applies for the rest of the characters in the input word provided in this example as well as it applies for longer words and sentences.

Figure 14 shows the probability distribution of each diacritization mark for each character in the input sequence in our example. RBMs that shape a DBN provide a closed-form representation of the distribution underlying the training data. These RBMs are trained to model the joint probability distribution of inputs and the corresponding labels, both represented by the visible units of the RBM. Afterwards, a new input pattern can be linked to the corresponding visible variables and the label can be predicted by sampling. Therefore, we have a generative model that allows sampling from the learned distribution.

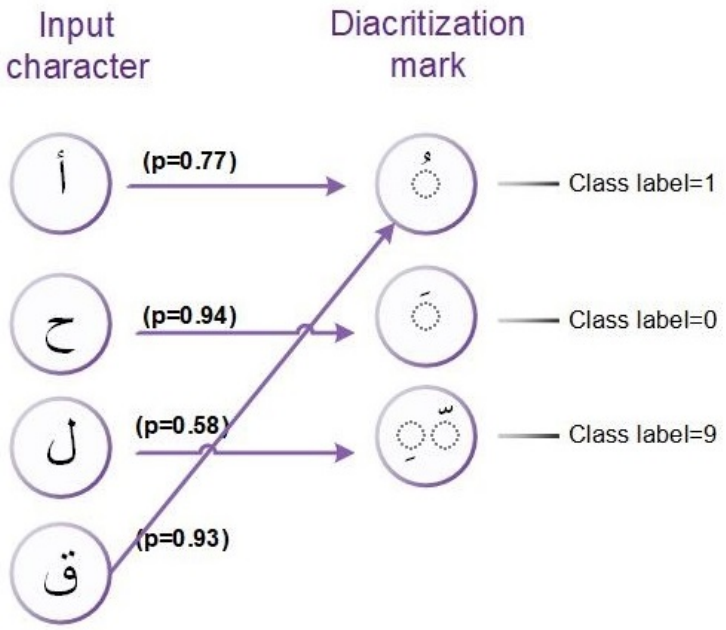
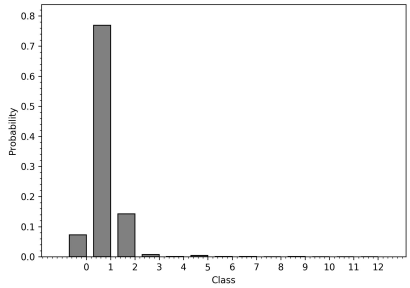
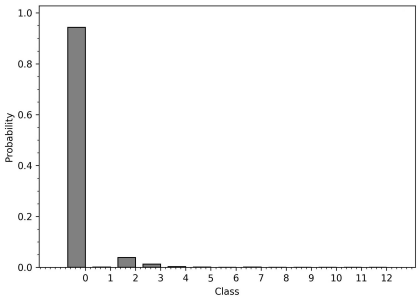


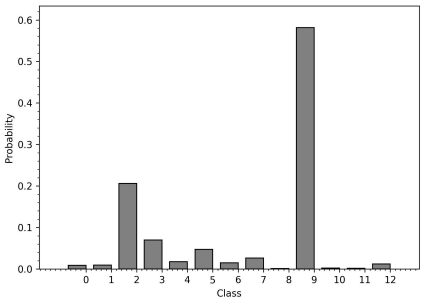
Figure 13. Example of how DBN probabilistically classify input.



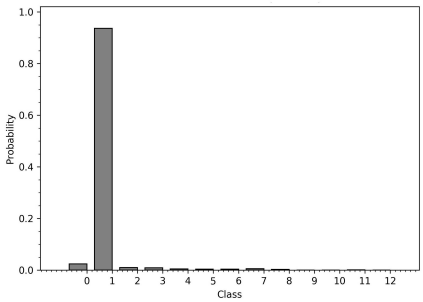
(**a**) First character class (Damma)



(**b**) Second character class (Fatha)



(**c**) Third character class (Shadda-Kasra)



(**d**) Fourth character class (Damma)

Figure 14. Probability distributions of class labels for the example word from the children's stories corpus.

6.7. Children Stories: A Novel Corpus of Arabic Diacritized Text

Corpora are built for a wide range of applications such as modeling language use for linguistics research, material for education, or training data for NLP applications. Research in NLP for Arabic language is drawing attention as it is becoming one of the most common languages used on the internet. Today, there are more than 168 million Arabic internet users worldwide and more than 2.1 billion Arabic indexed Web pages [88] and the availability of resources and tools is being developed [89].

The availability of a diacritized Arabic corpus has been a challenge for researchers in order to progress in new directions to solve the problem of automatic Arabic text diacritization. The matter of diacritized text is crucial for new learners to read Arabic and to the applications of text-to-speech systems.

There is only a small number of benchmark corpora for this purpose that limit the scope of dealing with it. In this paper, we present a corpus called children's stories. The corpus is freely available, it contains 26 thousands MSA words collected from children's stories books. The size of this corpus may be considered moderate or in some cases small compared to the size of the benchmarks used for Arabic NLP applications, however, it is unique as it has a different linguistic structure due to the type of books it is collected from and the audience these books were written to. Thus, more features may be extracted using machine learning algorithms. This resource can be leveraged for educational purposes and for researchers to use in developing NLP applications.

Our DBN model was tested using the children's stories dataset and the performance of the model was evaluated in terms of DER, WER, DER-1 and WER-1. Table 6 illustrates the results of this evaluation. As observed from the results, we get an improvement of 44% DER and 57% WER when ignoring the diacritic mark of the last character. Clearly, we have an improvement that we cannot underrate and is directly related to the type of

text we are dealing with and the linguistic structure it has. For example, most of the text written in children books and stories has a lot of rhymes, and the words and sentences are ordinarily shorter.

Table 6. Diacritization results of our DBN model on the children's stories dataset.

Dataset	All Diacritics		Ignore Last	
	DER	WER	DER-1	WER-1
Children stories	2.4	6.57	1.33	2.83

7. Conclusions and Future Work

In this study, we present a novel diacritization approach based on a DBN. DBN uses a network structure composed of multiple RBMs, which is more effective for data modeling and feature extraction. The key idea of DBN is to use a greedy layer-wise training to extract deep hierarchical representation of input data, followed by fine-tuning to achieve competitive classification performance.

This approach adds diacritic marks to undiacritized Arabic text using DBN. We have used two well-known benchmarks, which are Tashkeela and LDC ATB3. From the Tashkeela benchmark, we used ten books that are used in previous research by other researchers so that we can compare our results to theirs. Our results were evaluated in terms of DER and WER.

Experimentation on the ATB3 yielded a DER of 2.21%, which made an improvement of 19% over state-of-the-art systems. In addition, the WER scored by our approach was 6.73% outperforming competitive systems with an improvement of 26%.

On the Tashkeela benchmark, our system continues to achieve high accuracy. Our DBN-based approach scored a DER of 1.79% and 14% improvement over the best published systems. On the other hand, the WER recorded for our system was 4.63% and achieved an improvement of 20% over the best published system. With respect to the DER-1 and WER-1 metrics which imply ignoring the last letter diacritization mark, our system makes an improvement of DER and WER of 36% and 54%, respectively.

This approach outperforms state-of-the-art approaches. It does not require the use of rule-based techniques nor morphological, syntactic or diacritic rules. More importantly, it does not require any post-processing. When compared to state-of-the-art approaches, the DBN model significantly improves error rates for both the ATB3 and Tashkeela datasets.

The limitation of freely available resources including the availability of diacritized Arabic corpus creates a challenge and sometimes a barrier for researchers that work in this problem domain. Therefore, we presented in this work a new corpus of diacritized Arabic text collected from children's stories that consists of about 26K MSA words.

For future work, we plan to incorporate a hybrid approach of RBMs, DBNs and LSTMs as a preprocessing approach to examine to what extent it may make any considerable improvement to the performance of the DBN. It is expected that when using a DBN for preprocessing and then employing it with an LSTM for training a deep learning model the accuracy will improve considerably. We will further use the principal component analysis technique for feature extraction as it may be useful for a better understanding of the performance of the implemented approach. Moreover, to gain a better understanding of the effectiveness of the implemented approach we will use the K-fold cross validation approach with evolutionary techniques to optimize the training time and performance. In addition to that, a comparative work with different types of neural networks could be researched. Furthermore, we seek to extend the range of the children's stories corpus to a couple more thousands or millions of words.

Author Contributions: Conceptualization, W.A., M.A. and O.A.; Methodology, W.A.; Validation, W.A.; Data curation, W.A.; Writing—original draft preparation, W.A.; Writing—review and editing, W.A., M.A and O.A.; Supervision M.A. and O.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data available in a publicly accessible repository at http://tashkeela.sourceforge.net (accessed on 9 November 2020).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Albirini, A. *Modern Arabic Sociolinguistics: Diglossia, Variation, Codeswitching, Attitudes and Identity*; Routledge: London, UK, 2016 .
2. Farghaly, A.; Shaalan, K. Arabic natural language processing: Challenges and solutions. *ACM Trans. Asian Lang. Inf. Process. (TALIP)* **2009**, *8*, 1–22. [CrossRef]
3. Freihat, A.A.; Abbas, M.; Bella, G.; Giunchiglia, F. Towards an Optimal Solution to Lemmatization in Arabic. *Procedia Comput. Sci.* **2018**, *142*, 132–140. [CrossRef]
4. Said, A.; El-Sharqwi, M.; Chalabi, A.; Kamal, E. A hybrid approach for Arabic diacritization. In *International Conference on Application of Natural Language to Information Systems*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 53–64.
5. Hamed, O.; Zesch, T. A Survey and Comparative Study of Arabic Diacritization Tools. *J. Lang. Technol. Comput. Linguist.* **2017**, *32*, 27–47.
6. Chowdhury, G.G. Natural language processing. *Annu. Rev. Inf. Sci. Technol.* **2003**, *37*, 51–89. [CrossRef]
7. Saad, M.K.; Ashour, W.M. Arabic morphological tools for text mining. In Proceedings of the Corpora, 6th ArchEng International Symposiums, EEECS'10 the 6th International Symposium on Electrical and Electronics Engineering and Computer Science, Lefke, Cyprus, 25–26 November 2010; Volume 18.
8. Alsaleem, S. Automated Arabic Text Categorization Using SVM and NB. *Int. Arab. J. e Technol.* **2011**, *2*, 124–128.
9. Belkebir, R.; Guessoum, A. A supervised approach to arabic text summarization using adaboost. In *New Contributions in Information Systems and Technologies*; Springer: Berlin/Heidelberg, Germany, 2015; pp. 227–236.
10. Almuhareb, A.; Alsanie, W.; Al-Thubaity, A. Arabic word segmentation with long short-term memory neural networks and word embedding. *IEEE Access* **2019**, *7*, 12879–12887. [CrossRef]
11. Duwairi, R.M.; Qarqaz, I. Arabic sentiment analysis using supervised classification. In Proceedings of the 2014 International Conference on Future Internet of Things and Cloud, Barcelona, Spain, 27–29 August 2014; pp. 579–583.
12. Azmi, A.M.; Almajed, R.S. A survey of automatic Arabic diacritization techniques. *Nat. Lang. Eng.* **2015**, *21*, 477. [CrossRef]
13. Shaalan, K.; Bakr, H.M.A.; Ziedan, I. A hybrid approach for building Arabic diacritizer. In Proceedings of the EACL 2009 Workshop on Computational Approaches to Semitic Languages, Athens, Greece, 31 March 2009; pp. 27–35.
14. Zitouni, I.; Sorensen, J.; Sarikaya, R. Maximum entropy based restoration of Arabic diacritics. In Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Sydney, Australia, 17–21 July 2006; pp. 577–584.
15. Diab, M.; Ghoneim, M.; Habash, N. Arabic diacritization in the context of statistical machine translation. In Proceedings of the MT-Summit, Copenhagen , Denmark, 10–14 September 2007.
16. Rubi, C.R. A review: Speech recognition with deep learning methods. *Int. J. Comput. Sci. Mob. Comput.* **2015**, *4*, 1017–1024.
17. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef]
18. Hinton, G.E.; Osindero, S.; Teh, Y.W. A fast learning algorithm for deep belief nets. *Neural Comput.* **2006**, *18*, 1527–1554. [CrossRef]
19. Abbad, H.; Xiong, S. Multi-components System for Automatic Arabic Diacritization. In *European Conference on Information Retrieval*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 341–355.
20. Maamouri, M.; Bies, A.; Kulick, S. Diacritization: A challenge to Arabic treebank annotation and parsing. In Proceedings of the Conference of the Machine Translation SIG of the British Computer Society, London, UK , 23 October 2006 ; pp. 35–47.
21. Rashwan, M.A.; Al-Badrashiny, M.A.; Attia, M.; Abdou, S.M.; Rafea, A. A stochastic Arabic diacritizer based on a hybrid of factorized and unfactorized textual features. *IEEE Trans. Audio Speech Lang. Process.* **2010**, *19*, 166–175. [CrossRef]
22. Shaalan, K. Rule-based approach in Arabic natural language processing. *Int. J. Inf. Commun. Technol. (IJICT)* **2010**, *3*, 11–19.
23. Fashwan, A.; Alansary, S. A Rule Based Method for Adding Case Ending Diacritics for Modern Standard Arabic Texts. Available online: https://www.researchgate.net/publication/322223483_A_Rule_Based_System_for_Detecting_the_Syntactic_Diacritics_in_Modern_Standard_Arabic_Texts (accessed on 10 April 2021).
24. Metwally, A.S.; Rashwan, M.A.; Atiya, A.F. A multi-layered approach for Arabic text diacritization. In Proceedings of the 2016 IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), Chengdu, China, 5–7 July 2016; pp. 389–393.

25. Alansary, S. Alserag: An automatic diacritization system for arabic. In *Intelligent Natural Language Processing: Trends and Applications*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 523–543.
26. Elshafei, M.; Al-Muhtaseb, H.; Alghamdi, M. Statistical Methods for Automatic Diacritization of Arabic Text. Available online: https://www.researchgate.net/publication/236115959_Statistical_Methods_for_Automatic_diacritization_of_Arabic_text (accessed on 10 April 2021).
27. Hifny, Y. Smoothing techniques for Arabic diacritics restoration. *Proc. Conf. Lang. Eng.* **2012**, *1*, 6–12.
28. Nelken, R.; Shieber, S.M. Arabic diacritization using weighted finite-state transducers. In Proceedings of the ACL Workshop on Computational Approaches to Semitic Languages, Ann Arbor, MI, USA, 29 June 2005; pp. 79–86.
29. Darwish, K.; Mubarak, H.; Abdelali, A. Arabic diacritization: Stats, rules, and hacks. In Proceedings of the Third Arabic Natural Language Processing Workshop, Valencia, Spain, 3 April 2017; pp. 9–17.
30. Mijlad, A.; Younoussi, Y.E. Arabic Text Diacritization: Overview and Solution. pp. 1–7. Available online: https://dl.acm.org/doi/10.1145/3368756.3369088 (accessed on 10 April 2021).
31. Boudchiche, M.; Mazroui, A.; Bebah, M.O.A.O.; Lakhouaja, A.; Boudlal, A. AlKhalil Morpho Sys 2: A robust Arabic morpho-syntactic analyzer. *J. King Saud Univ. Comput. Inf. Sci.* **2017**, *29*, 141–146. [CrossRef]
32. Rashwan, M.A.; Al Sallab, A.A.; Raafat, H.M.; Rafea, A. Deep learning framework with confused sub-set resolution architecture for automatic Arabic diacritization. *IEEE/ACM Trans. Audio Speech Lang. Process.* **2015**, *23*, 505–516. [CrossRef]
33. Abandah, G.A.; Graves, A.; Al-Shagoor, B.; Arabiyat, A.; Jamour, F.; Al-Taee, M. Automatic diacritization of Arabic text using recurrent neural networks. *Int. J. Doc. Anal. Recognit. (IJDAR)* **2015**, *18*, 183–197. [CrossRef]
34. Vergyri, D.; Kirchhoff, K. *Automatic Diacritization of Arabic for Acoustic Modeling in Speech Recognition*; Technical Report; Washington University Seattle Department of Electrical Engineering: Seattle, WA, USA, 2004.
35. Habash, N.; Rambow, O. Arabic tokenization, part-of-speech tagging and morphological disambiguation in one fell swoop. In Proceedings of the 43rd Annual Meeting of the Association for computational Linguistics (ACL'05), Ann Arbor, MI, USA, 25–30 June 2005; pp. 573–580.
36. Fadel, A.; Tuffaha, I.; Al-Ayyoub, M. Arabic text diacritization using deep neural networks. In Proceedings of the 2019 2nd International Conference on Computer Applications & Information Security (ICCAIS), Riyadh, Saudi Arabia, 1–3 May 2019; pp. 1–7.
37. Mubarak, H.; Abdelali, A.; Sajjad, H.; Samih, Y.; Darwish, K. Highly effective arabic diacritization using sequence to sequence modeling. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Minneapolis, MN, USA, 2–7 June 2019; pp. 2390–2395.
38. Abandah, G.; Abdel-Karim, A. Accurate and fast recurrent neural network solution for the automatic diacritization of Arabic text. *Jordan. J. Comp. Inform. Technol.* **2020**, *6*, 103–121. [CrossRef]
39. Alqahtani, S.; Mishra, A.; Diab, M. Efficient Convolutional Neural Networks for Diacritic Restoration. *arXiv* **2019**, arXiv:1912.06900.
40. Bai, S.; Kolter, J.Z.; Koltun, V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv* **2018**, arXiv:1803.01271.
41. McCulloch, W.S.; Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.* **1943**, *5*, 115–133. [CrossRef]
42. Hassoun, M.H. *Fundamentals of Artificial Neural Networks*; MIT Press: Cambridge, MA, USA, 1995 .
43. Wang, S.C. Artificial neural network. In *Interdisciplinary Computing in JAVA Programming*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 81–100.
44. Hua, Y.; Guo, J.; Zhao, H. Deep belief networks and deep learning. In Proceedings of the 2015 International Conference on Intelligent Computing and Internet of Things, Harbin, China, 17–18 January 2015; pp. 1–4.
45. Werbos, P.J. Backpropagation through time: What it does and how to do it. *Proc. IEEE* **1990**, *78*, 1550–1560. [CrossRef]
46. Hornik, K.; Stinchcombe, M.; White, H. Multilayer feedforward networks are universal approximators. *Neural Netw.* **1989**, *2*, 359–366. [CrossRef]
47. Sazlı, M.H. A brief review of feed-forward neural networks. *Commun. Fac. Sci. Univ. Ank.* **2006**, *50*, 11–17. [CrossRef]
48. Wu, H.; Zhou, Y.; Luo, Q.; Basset, M.A. Training feedforward neural networks using symbiotic organisms search algorithm. *Comput. Intell. Neurosci.* **2016**, *9063065* . [CrossRef]
49. Nawi, N.M.; Ransing, R.S.; Salleh, M.N.M.; Ghazali, R.; Hamid, N.A. An improved back propagation neural network algorithm on classification problems. In *Database Theory and Application, Bio-Science and Bio-Technology*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 177–188.
50. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning representations by back-propagating errors. *Nature* **1986**, *323*, 533–536. [CrossRef]
51. Werbos, P.J. *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*; John Wiley & Sons: Hoboken, NJ, USA, 1994; Volume 1.
52. Simon, H. *Neural Networks: A Comprehensive Foundation*; Prentice Hall: Hoboken, NJ, USA, 1999 .
53. Rocha, M.; Cortez, P.; Neves, J. Evolutionary neural network learning. In *Portuguese Conference on Artificial Intelligence*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 24–28.

54. Stanley, K.O.; Clune, J.; Lehman, J.; Miikkulainen, R. Designing neural networks through neuroevolution. *Nat. Mach. Intell.* **2019**, *1*, 24–35. [CrossRef]
55. Montana, D.J.; Davis, L. Training Feedforward Neural Networks Using Genetic Algorithms. Available online: https://www.ijcai.org/Proceedings/89-1/Papers/122.pdf (accessed on 10 April 2021).
56. Mirjalili, S.; Hashim, S.Z.M.; Sardroudi, H.M. Training feedforward neural networks using hybrid particle swarm optimization and gravitational search algorithm. *Appl. Math. Comput.* **2012**, *218*, 11125–11137. [CrossRef]
57. Blum, C.; Socha, K. Training feed-forward neural networks with ant colony optimization: An application to pattern classification. In Proceedings of the Fifth International Conference on Hybrid Intelligent Systems (HIS'05), Rio de Janerio, Brazil, 6–9 November 2005.
58. Faris, H.; Aljarah, I.; Mirjalili, S. Training feedforward neural networks using multi-verse optimizer for binary classification problems. *Appl. Intell.* **2016**, *45*, 322–332. [CrossRef]
59. Valian, E.; Mohanna, S.; Tavakoli, S. Improved cuckoo search algorithm for feedforward neural network training. *Int. J. Artif. Intell. Appl.* **2011**, *2*, 36–43.
60. Karaboga, D.; Akay, B.; Ozturk, C. Artificial bee colony (ABC) optimization algorithm for training feed-forward neural networks. In *International Conference on Modeling Decisions for Artificial Intelligence*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 318–329.
61. Mirjalili, S. How effective is the Grey Wolf optimizer in training multi-layer perceptrons. *Appl. Intell.* **2015**, *43*, 150–161. [CrossRef]
62. Bengio, Y. *Learning Deep Architectures for AI*; Now Publishers Inc.: Delft, The Netherlands, 2009 .
63. Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.; Dean, J. Distributed representations of words and phrases and their compositionality. *arXiv* **2013**, arXiv:1310.4546.
64. Zepeda-Mendoza, M.L.; Resendis-Antonio, O., Bipartite Graph. In *Encyclopedia of Systems Biology*; Dubitzky, W., Wolkenhauer, O., Cho, K.H., Yokota, H., Eds.; Springer: New York, NY, USA, 2013; pp. 147–148. [CrossRef]
65. Weisstein, E.W. Bernoulli Distribution. 2002. Available online: https://mathworld.wolfram.com/ (accessed on 10 April 2021).
66. Fischer, A.; Igel, C. An introduction to restricted Boltzmann machines. In *Iberoamerican Congress on Pattern Recognition*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 14–36.
67. Hinton, G.E. Training products of experts by minimizing contrastive divergence. *Neural Comput.* **2002**, *14*, 1771–1800. [CrossRef] [PubMed]
68. Hinton, G.E. A practical guide to training restricted Boltzmann machines. In *Neural Networks: Tricks of the Trade*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 599–619.
69. Morabito, F.C.; Campolo, M.; Ieracitano, C.; Mammone, N. Deep Learning Approaches to Electrophysiological Multivariate Time-Series Analysis. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*; Elsevier: Amsterdam, The Netherlands, 2019; pp. 219–243.
70. Yepes, A.J.; MacKinlay, A.; Bedo, J.; Garvani, R.; Chen, Q. Deep belief networks and biomedical text categorisation. In Proceedings of the Australasian Language Technology Association Workshop 2014, Melbourne, Australia, 26–28 November 2014; pp. 123–127.
71. Lu, P.; Guo, S.; Zhang, H.; Li, Q.; Wang, Y.; Wang, Y.; Qi, L. Research on improved depth belief network-based prediction of cardiovascular diseases. *J. Healthc. Eng.* **2018**, 8954878. [CrossRef]
72. Goyvaerts, J.; Levithan, S. *Regular Expressions Cookbook*; O'reilly: Newton, MA, USA, 2012.
73. Chapman, C.; Stolee, K.T. Exploring regular expression usage and context in Python. In Proceedings of the 25th International Symposium on Software Testing and Analysis, Saarbrucken, Germany, 18–20 July 2016; pp. 282–293.
74. He, H.; Garcia, E.A. Learning from imbalanced data. *IEEE Trans. Knowl. Data Eng.* **2009**, *21*, 1263–1284.
75. Faris, H.; Abukhurma, R.; Almanaseer, W.; Saadeh, M.; Mora, A.M.; Castillo, P.A.; Aljarah, I. Improving financial bankruptcy prediction in a highly imbalanced class distribution using oversampling and ensemble learning: A case from the Spanish market. *Prog. Artif. Intell.* **2020**, *9*, 31–53. [CrossRef]
76. Han, H.; Wang, W.Y.; Mao, B.H. Borderline-SMOTE: A new over-sampling method in imbalanced data sets learning. In *International Conference on Intelligent Computing*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 878–887.
77. Yu, Y.; Adu, K.; Tashi, N.; Anokye, P.; Wang, X.; Ayidzoe, M.A. Rmaf: Relu-memristor-like activation function for deep learning. *IEEE Access* **2020**, *8*, 72727–72741. [CrossRef]
78. Nair, V.; Hinton, G.E. Rectified Linear Units Improve Restricted Boltzmann Machines. Available online: https://www.cs.toronto.edu/~fritz/absps/reluICML.pdf (accessed on 10 April 2021).
79. Glorot, X.; Bordes, A.; Bengio, Y. Deep sparse rectifier neural networks. In Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, JMLR Workshop and Conference Proceedings, Fort Lauderdale, FL, USA, 11–13 April 2011; pp. 315–323.
80. Ide, H.; Kurita, T. Improvement of learning for CNN with ReLU activation by sparse regularization. In Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN), Anchorage, AL, USA, 14–19 May 2017; pp. 2684–2691.
81. Agarap, A.F. Deep learning using rectified linear units (relu). *arXiv* **2018**, arXiv:1803.08375.
82. Bisong, E. Google colaboratory. In *Building Machine Learning and Deep Learning Models on Google Cloud Platform*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 59–64.
83. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.

84. Habash, N.; Rambow, O. Arabic Diacritization through Full Morphological Tagging. Available online: https://www.aclweb.org/anthology/N07-2014/ (accessed on 10 April 2021).
85. Schlippe, T.; Nguyen, T.; Vogel, S. Diacritization as a machine translation problem and as a sequence labeling problem. AMTA-2008. MT at work. In Proceedings of the Eighth Conference of the Association for Machine Translation in the Americas, Waikiki, HI, USA, 21–25 October 2008; pp. 270–278.
86. Alghamdi, M.; Muzaffar, Z.; Alhakami, H. Automatic restoration of arabic diacritics: A simple, purely statistical approach. *Arab. J. Sci. Eng.* **2010**, *35*, 125.
87. Barqawi, A.; Zerrouki, T. Shakkala, Arabic Text Vocalization. Available online: https://github.com/Barqawiz/Shakkala2017 (accessed on 10 April 2021).
88. Alarifi, A.; Alghamdi, M.; Zarour, M.; Aloqail, B.; Alraqibah, H.; Alsadhan, K.; Alkwai, L. Estimating the size of Arabic indexed web content. *Sci. Res. Essays* **2012**, *7*, 2472–2483.
89. Habash, N.Y. Introduction to Arabic natural language processing. *Synth. Lect. Hum. Lang. Technol.* **2010**, *3*, 1–187. [CrossRef]