Received: 16 December 2018 | Revised: 15 February 2019 | Accepted: 7 April 2019

DOI: 10.4218/etrij.2018-0553

ORIGINAL ARTICLE

ETRI Journal WILEY

Deep recurrent neural networks with word embeddings for Urdu named entity recognition

Wahab Khan[1] | **Ali Daud**[1] | **Fahd Alotaibi**[2] | **Naif Aljohani**[2] | **Sachi Arafat**[2]

[1]Department of Computer Science and Software Engineering, International Islamic University, Islamabad, Pakistan

[2]Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia

Correspondence
Wahab Khan, Department of Computer Science and Software Engineering, International Islamic University, Islamabad, Pakistan.
Email: wahab.phdcs72@iiu.edu.pk

Abstract

Named entity recognition (NER) continues to be an important task in natural language processing because it is featured as a subtask and/or subproblem in information extraction and machine translation. In Urdu language processing, it is a very difficult task. This paper proposes various deep recurrent neural network (DRNN) learning models with word embedding. Experimental results demonstrate that they improve upon current state-of-the-art NER approaches for Urdu. The DRRN models evaluated include forward and bidirectional extensions of the long short-term memory and back propagation through time approaches. The proposed models consider both language-dependent features, such as part-of-speech tags, and language-independent features, such as the "context windows" of words. The effectiveness of the DRNN models with word embedding for NER in Urdu is demonstrated using three datasets. The results reveal that the proposed approach significantly outperforms previous conditional random field and artificial neural network approaches. The best f-measure values achieved on the three benchmark datasets using the proposed deep learning approaches are 81.1%, 79.94%, and 63.21%, respectively.

KEYWORDS

conditional random fields, deep recurrent neural network, machine learning, named entity recognition, Urdu

1 | INTRODUCTION

Named entity recognition (NER), sometimes referred to as "entity identification," "entity chunking," or "entity extraction," is one of the most basic natural language processing (NLP) tasks. It is generally considered to be a sequential labeling task corresponding to the identification and classification of proper nouns into predefined categories, such as persons, locations, organizations, expressions of time, quantities, and monetary values. It is considered to be an essential preliminary step in many NLP tasks, such as question answering, conversation, and voice search. Therefore, NER plays a vital role in the management and extraction of intelligent information from text.

The general approach in an NER task is to generate and assign class labels for each part of a sentence. Initially, this seems to be a relatively simple task when using a predefined database to hold various types of named entities. However, this task poses a significant challenge for languages such as Urdu that are disadvantaged by a lack of supporting resources (eg., grammatically annotated corpora, specialized dictionaries, and gazetteers) and capitalization features, both of which are helpful for computational linguistic analysis. The morphological richness of Urdu adds to the challenge of the NER

task because a single word can take multiple forms and acquire multiple NE tags in different contexts.

NER techniques were initially studied in Message Understanding Conferences that were initiated by the Defense Advanced Research Projects Agency to aid in the development of information extraction techniques. The study of such techniques expanded rapidly as they were adopted in a wide range of systems [1]. Early systems mainly supported European languages [2], including English. Such systems have achieved a mature status and are able to provide effective results. Although NER frameworks have been proposed for non-European languages, such as Arabic, Persian, and South Asian languages [3], Urdu systems are still in the early stages of development [1,4]. Therefore, NER for Urdu is a difficult task requiring greater sophistication in linguistic analysis and the development of techniques for effective task performance.

Currently, many people are using deep-learning– and artificial-intelligence–based approaches for NLP tasks. Such approaches are considered to represent the state of the art for various task in NLP.

In this work, we developed a deep recurrent neural network (DRNN)-based system for Urdu NER and compared it to conventional linear-chain conditional random field (CRF)- and artificial neural network (ANN)-based NER schemes.

In our literature review, we discovered that only the authors of [5] have reported the application of a traditional ANN to three Urdu NE classes. To date, no studies has been reported by the unnatural language processing (ULP) research community in which researchers have examined the effects of deep learning models and word embeddings on Urdu NER. This research gap, as well as the limitations of rule-based approaches and advantages of deep learning architectures in diverse areas, motivated us to develop a DRNN model with word embeddings and part of speech (POS) information as features for Urdu NER.

The main contributions of our work can be summarized as follows:

- This study is the first to apply a deep learning architecture for Urdu NER to benchmark datasets.
- This paper presents a systematic evaluation of DRNN models with CRF and ANN baselines based on three datasets.
- The proposed model includes a novel feature set for Urdu NER that includes templates, word embeddings, context, and run time.
- The proposed model is the first to employ POS information as features for Urdu NER.
- The proposed model uses seven entity classes for NER, while most previous approaches have been limited to three entity classes: person, location, and organization. The proposed model adds four classes of designation, number, date, and time.

2 | RELATED WORK

A long tradition of research into NER for English starting in the early 1990s has seen the development various techniques, such as rule-based, purely supervised, and hybrid approaches. NER schemes constructed for particular domains do not typically transfer effectively to other domains, meaning techniques developed for European languages are not immediately applicable in the relatively new ULP context. However, techniques for Urdu NER can be broadly classified as rule-based, machine learning (ML), and hybrid approaches.

2.1 | Rule-based approaches

Rule-based approaches [3,6] use a collection of tailor-made rules, such as grammar, affix lookups, and lexicon lookups, which are designed for each class of NE. These rules are then applied to a given text to extract NEs. A rule-based algorithm first searches for an NE and compares it to a set of predefined rules. Once a rule is matched, it assigns the NE to the corresponding classification, which is the output of the algorithm. A notable aspect of systems formulated by applying a linguistic-rule-supported approach is that they generally lack potency and manageability based on the following reasons:

- They must be continuously updated with new rules as the corresponding domain changes.
- Adding rules for a specific task requires both knowledge of the corresponding language and expertise in rule creation.
- Rules developed for one language cannot generally be successfully exported to another language.
- They suffer from much longer development times compared to other techniques.

Riaz [6] presented the first instance of a rule-based algorithm for Urdu NER. This algorithm considers six NE classes in offline plain Urdu text: proper (human) names, territory or state names, organizations, numbers, dates, and designations. It was evaluated using the Becker-Riaz dataset and achieved precision, recall, and f-measure values of 91.5%, 90.7%, and 91%, respectively.

Singh et al [3] presented a rule-based approach that recognizes 13 entities in offline plain Urdu text by extending the 12 tags from IJCNLP-2008. The rules were evaluated on test data compiled from news sources that were divided into two sets of 12 032 and 150 243 tokens. An accuracy of 74.09% was achieved across all thirteen tags.

2.2 | Machine learning approaches

Currently, the leading contemporary approaches to NER for most languages are based on ML approaches. Recent trends

in the analysis of large datasets using supervised learning approaches [7] are indicative of the overall inclination toward ML approaches. Supervised models operate by deriving rules from pre-labeled data, known as training data. These rules correspond to parametric, nonparametric, or kernel-based learning algorithms, or to algorithms that employ logic [7].

ML-based NER frameworks are generally more flexible and robust than rule-based methods, meaning they can be easily applied to different languages and areas. If training data is readily available, an ML model can adapt to changing domains with very little effort.

The lack of prior work on Urdu NER, especially prior ML approaches, is partly caused by the low level of interest from the NLP community and partly caused by the scarcity of resources for computational analysis. An initial significant study on NER for digital Urdu text was [8], which led to the foundation of a popular Urdu dataset called the Becker-Riaz Urdu corpus.

In 2008, additional significant studies and techniques for Urdu NER were presented at the Third International Joint Conference on Natural Language Processing (IJCNLP-08),[1] which aimed to investigate NER for five south Asian languages, including Urdu, using ML techniques. Since that pioneering conference, several additional studies have explored ML approaches to Urdu NER.

Malik and Sarwar [9] adopted a CRF-based approach and evaluated it on the benchmark IJCNLP-2008 NER dataset with three NE types: person, organization, and location. They limited language-independent features, such as context words, and reported precision, recall, and f-measure values of 63.7%, 62.3%, and 63%, respectively.

The study by Ekbal et al [10] is considered to be the first ever to investigate NER for five widely spoken Indian languages (Urdu, Hindi, Oriya, Bengali, and Telugu) using CRF models. Evaluations using the IJCNLP-2008 dataset resulted in f-measure values of 35.52% for Urdu and 59.3%, 28.71%, and 4.74% for Bengali, Oriya, and Telugu, respectively.

Mukund et al [4] also applied a CRF approach, but they used both language-independent and language-dependent features, such as POS tags. They trained and tested their algorithm on two datasets: one for POS learning and one for NE learning. Evaluations were performed for three main NE classes: person, location, and organization. The results of 10-fold cross-validation were reported as 68.89% for f-measure with a maximum value of 69.21%. Additionally, there is experimental evidence for improved results when using heuristics, with the f-measure value for the worst test set increasing from 68.89% to 74.67% and that for the best test set increasing from 69.21% to 71.3%.

Jahangir et al [11] developed unigram and bigram models that use a gazetteer list and employed smoothing algorithms in their bigram approach. Five NE classes were considered: person, location, organization, date, and time. A CRL dataset was used for training and testing. The precision, recall, and f-measure values for the unigram approach using gazetteer lists were reported as 65.21%, 88.63%, and 75.14%, respectively. Respective values of 66.2%, 88.18%, and 75.83% were achieved by the bigram approach using gazetteer lists and backoff smoothing.

Malik [5] proposed a traditional ANN- and hidden Markov model (HMM)-based Urdu NER system. He evaluated the performance of an ANN and HMM for three NEs: person, organization, and location. For training and testing, he used a custom corpus called KPU-NE and reported values of 55.98%, 83.11%, and 66.9% for precision, recall, and f-measure, respectively, for the HMM model, with values of 81.05%, 87.54%, and 84.17%, respectively, for the traditional ANN model.

2.3 | Hybrid approaches

Hybrid methods typically combine both rule-based and ML methods. Kumar Saha et al [12] developed a hybrid NER system for NE identification for five languages: Hindi, Bengali, Telugu, Oriya, and Urdu. Their proposed approach makes use of linguistic rules combined with a maximum entropy model. They also used gazetteer lists for performance improvement. The f-measure values reported for the five languages listed above are 65.13%, 65.96%, 44.65%, 18.74%, and 35.47%, respectively. The hybrid system proposed by Gali et al [13] makes use of tailor-made rules combined with CRFs. Based on insufficient training data, their system yielded an f-measure of only 43.46%.

Kumar and Kiran [14] created a hybrid NER system based on the combination of CRFs and HMMs with tailor-made heuristics. They evaluated the performance of their approach on the IJCNLP-2008 workshop dataset. This hybrid approach based on rules and HMMs achieved an f-measure of 44.73% for Urdu, with a value of only 38.25% when using the hybrid CRFs alone.

3 | CHALLENGES IN URDU NER

Generally, NER is considered to be a challenging task [6]. The general approach to an NER task is to generate and assign a class label for each part of a sentence. Initially, this may seem like a relatively simple problem when using a predefined database to hold various types of NEs. However, the task poses significant challenges for morphologically rich languages, such as Urdu and Arabic [6,15]. The morphological richness of Urdu [16–18], which is similar to that of English, Arabic, and other South Asian languages, such as Hindi, introduces significant challenge to the NER

[1] http://ltrc.iiit.ac.in/nlp-lpl-08/

task [15] because a single word can take multiple forms and acquire multiple NE tags in different contexts. For example, the word (Jan) can be used as a common noun, proper noun, or adjective. This polymorphemic behavior of many Urdu words makes the NER task very challenging. Therefore, Urdu NER requires detailed analysis and adaptation.

4 | RECURRENT NEURAL NETWORKS

ANNs are the parent class from which RNNs have evolved. The simplest form of an RNN is the Elman framework. This framework is considered to be a conventional RNN in the literature. The core difference between RNNs and other conventional neural networks lies in their input and output sizes. Conventional networks, such as autoencoders and convolutional neural networks (CNNs), support fixed input and output sizes, whereas RNNs can handle inputs and outputs with arbitrary lengths.

The CNN is a type of conventional neural network that is very useful for solving problems based on processing two-dimensional data, such as image and video recognition tasks, whereas RNNs are considered to be a good choice for processing multidimensional sequential data, such as text and speech data. In an RNN memory unit/black box, both the current input and the output of the previous unit are used, but a conventional neural network lacks this property.

Figures 1 and 2 depict the basic structures of a traditional feed-forward neuron and recurrent neuron, respectively. In these diagrams, the traditional neuron is shown with a single input/output, but it can accept N inputs and can yield M outputs.

The distinctive feature of an RNN is its hidden layers, which are made up of recursive edges that enable RNNs to consider past information from input data. Because a hidden layer stores previous information, hidden layers are also referred to as memory units. Therefore, based on the storage characteristics of hidden layers, as time increases, hidden-layer neurons acquire long-term dependencies.

A conventional RNN has three layers. The input layer collects inputs for the network. The hidden layer consists of recursive edges and operates based on stored historical data. The output layer outputs predicted class labels. Figure 3 illustrates a conventional forward RNN that is extended into a complete network.

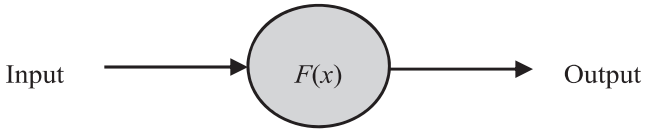

FIGURE 1 Basic Structure of a feed-forward neuron

In Figure 3, U, W, and V represent the parameters for the input, hidden, and output layers, respectively. These parameters are calculated during the training phase and are used to connect input, hidden, and output layers.

The mathematical expressions that govern the computations in an RNN are presented below.

$$h_t = \sigma\left(U \times x(t) + W \times h_{t-1}\right), \quad t = 1, \ldots, T, \tag{1}$$

$$y(t) = g(V \times h(t)), \tag{2}$$

where $x(t)$ is an input state at a time t that consists of a single vector and $h(t)$ is a memory unit in the network that corresponds to the hidden layer at a time t. Its calculation is based on the output of the previous hidden state, as well as the current input state.

$g(Vh(t))$ calculates $y(t)$ using the output of the hidden layers and the weights between the hidden layer and output layer y. $y(t)$ corresponds to the output at a time t;

σ(Tanh) represents a function that is responsible for initializing the first hidden state.

4.1 | Deep recurrent neural network

Pascanu et al [19] indicated that the conception of depth associated with an RNN is not as clear as that associated with feed-forward neural networks. In a feed-forward neural network, the concept of depth is refers to the number of nonlinear layers between input and output layers. By this definition, RNNs are all deep because any RNN can be visualized as a stack of multiple nonlinear layers when unfolded with respect to time. However,

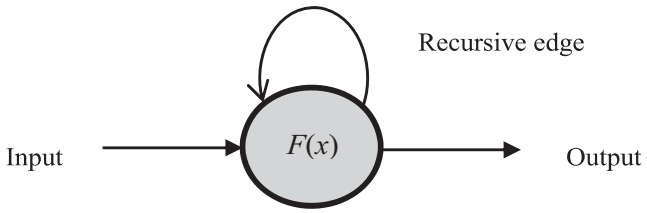

FIGURE 2 Basic structure of a recurrent neuron

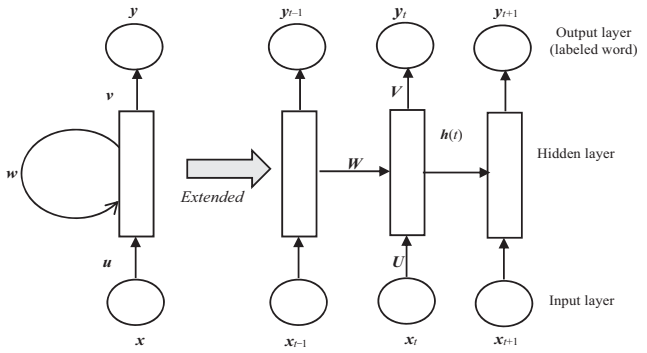

FIGURE 3 Schematic representation of a conventional forward RNN being extended to create a complete network

in the case of RNNs, depth takes on a different meaning based on the temporal structure of an RNN, where outputs are largely based on preceding contextual computations.

The most naive, conceivable, and widely used architecture for a deep RNN was proposed in [20]. This architecture stacks multiple recurrent layers to frame a deep RNN with the goal of reducing the high computational costs associated with previous versions of neural networks. This stacked architecture is able to consider the hidden states at each level to operate at varied timescales, providing enhanced learning capability.

The authors of [19] were the first to explore numerous deep extensions of conventional RNNs. Pascanu et al [19] carefully analyzed RNN architectures and determined that a conventional RNN can be extended to create a deep RNN through three main types of modifications. The first way in which a conventional RNN can be extended to create a deep RNN is by introducing one or more intermediate layers between the input and hidden state ($x(t)$, $h(t)$). In this manner, the transitions between hidden and input states will become deeper. In the second modification, the transitions between two successive hidden states ($h(t)$, $h(t-1)$) are made deeper. Finally, the same approach can be adapted to make the transitions between hidden states and the output state deeper ($h(t)$, $y(t)$). For additional details, we encourage readers consult the original work by Pascanu et al [19].

Both regular and deep RNNs models can be executed in two modes: forward and bidirectional. In a bidirectional RNN, the final result at any time t is established not only based on the results of element $N-1$, but also based on those of element $N+1$. Bidirectional RNNs are a simple form of RNN. They can be visualized as two simple RNNs stacked on top of each other. A schematic of a regular bidirectional RNN is presented in Figure 4. A bidirectional deep RNN architecture has multiple hidden states with bidirectional recursive edges (see Figure 5).

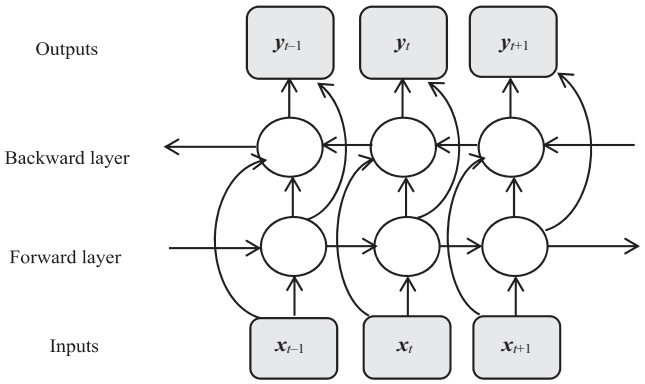

FIGURE 4 Schematic representation of a regular bidirectional RNN

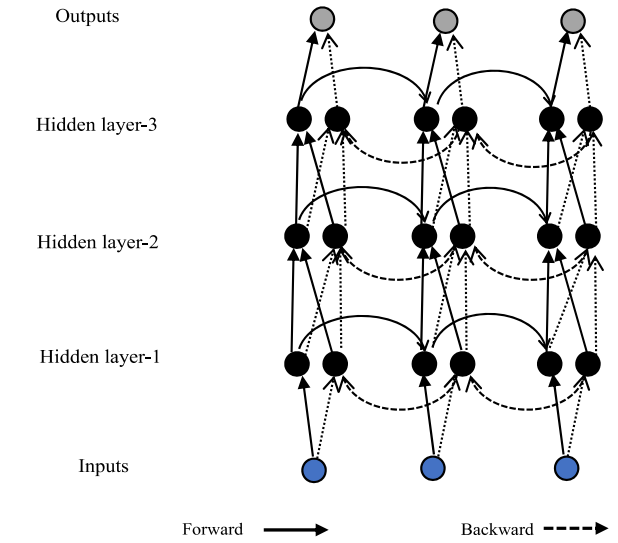

FIGURE 5 Schematic representation of a deep bidirectional RNN

4.2 | RNN training

The training of RNNs can be accomplished using either long short-term memory (LSTM) or back-propagation through time (BPTT) algorithms.

4.2.1 | Back-propagation through time

Almost all neural networks are trained using back-propagation algorithms. Because all RNN parameters (U, W, V) are shared across all timestamps, BPTT is referred to as a gradient-based approach. The gradient (or vector-valued function) at each output depends on the calculations at the current time, as well as the calculations at time $N-1$. For example, if we were interested in finding the gradient at $t=3$, we would be required to back-propagate $N-2$ steps and add the gradients. This process is defined as BPTT.

However, when conventional RNNs are trained using BPTT, it can lead to the vanishing gradient problem, which was first discovered in 1991, restricting RNN ability to capture long-run dependencies in sequences.

4.2.2 | Long short-term memory network

As mentioned above, conventional RNNs encounter the vanishing gradient problem when they are trained using BPTT. Therefore, to address this problem, a robust framework called LSTM was introduced in 1997. Currently, LSTMs are a commonly used class of RNN that have been shown to be more effective at attaining long-run information persistence compared to conventional RNNs.

Compared to conventional RNNs, LSTM-RNNs engage a distinct computing mechanism in their hidden layers.

The recursive edges of the hidden layers, which are responsible for information storage in LSTM, are referred to as memory cells or black boxes. The output of the previous layer and the input for the current layer constitute its parameters.

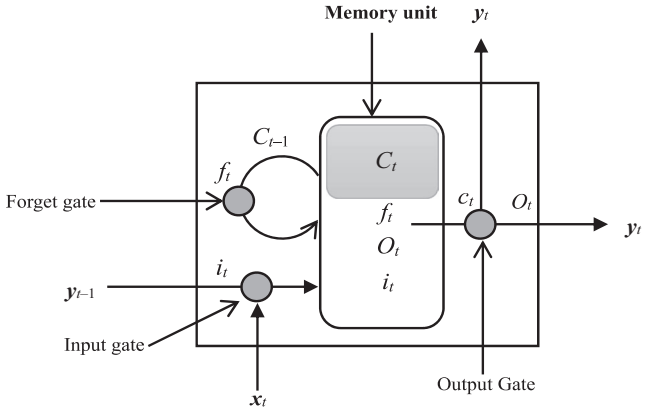

FIGURE 6 Illustration of an LSTM cell

The memory units of LSTM networks are intelligent units that are responsible for choosing which information to remember and which information to forget. These LSTM units combine the information from layer $N-1$ with the incumbent memory and a new input. An LSTM memory cell is a combination of three multiplicative gates that adaptively ensure information flow within a memory unit. The input gate is responsible for determining the proportion of input information that is shifted to a memory cell. The forget gate is responsible for determining the proportion of historical information to erase from the previous state. Finally, the output gate is responsible for determining the proportion of output information to shift to adjacent units [21]. Figure 6 illustrates the basic structure of an LSTM unit, where x_t, y_{t-1}, and c_{t-1} represent inputs relative to a time t, and $y(t)$ and $c(t)$ represent the outputs of the unit. The equations that regulate calculations inside the unit are listed below.

$$i_t = \sigma(\mathbf{W}_{xi}x_t + \mathbf{W}_{yi}y_{t-1} + \mathbf{W}_{ci}\mathbf{c}_{t-1} + \mathbf{b}_i, \quad (3)$$

$$f_t = \sigma(\mathbf{W}_{xf}x_t + \mathbf{W}_{yf}y_{t-1} + \mathbf{W}_{cf}\mathbf{c}_{t-1} + \mathbf{b}_f, \quad (4)$$

$$\mathbf{c}_t = f_t \odot \mathbf{c}_{t-1} + i_t \odot \tanh\left(\mathbf{W}_{xc}x_t + \mathbf{W}_{yc}y_{t-1} + \mathbf{b}_c\right), \quad (5)$$

$$o_t = \sigma(\mathbf{W}_{xo}x_t + \mathbf{W}_{yo}y_{t-1} + \mathbf{W}_{co}\mathbf{c}_t + \mathbf{b}_o), \quad (6)$$

$$h_t = o_t \odot \tanh\left(c_t\right), \quad (7)$$

where i_t, f_t, and o_t represent the input, forget, and output gates, respectively. σ denotes the element-wise product of the gates and $\mathbf{c}_t$ is the cell vector. $\mathbf{W}_i$, $\mathbf{W}_f$, and $\mathbf{W}_o$ are the weight matrices, and $\mathbf{b}_i$, $\mathbf{b}_f$, and $\mathbf{b}_o$ are the bias vectors. The vanishing gradient problem associated with the traditional RNN approach not only led to the development of the LSTM architecture, but also spawned two popular variants of LSTM in the literature, namely the bidirectional LSTM RNNs proposed in [22] and the gated recurrent units proposed in [23]. Figure 7 depicts a bidirectional LSTM with the Urdu sentence "بینظیر بھٹو مسحورکن شخصیت کی مالک تھیں," which means "Benazir Bhutto has an alluring personality."

5 | EXPERIMENTAL EVALUATION

We performed numerous experiments with different combinations of RNN model architectures and a novel feature set to analyze the influence of different models on Urdu NER. In this study, the effectiveness of DRNN models for URDU NER was evaluated on three datasets, including a bouffant set of tagged NEs. We employed the schematic SBME[2] annotation scheme for chunk tags. Additionally, we examined the effects of using word templates, such as the context windows of words and language-dependent features (POS tags and pretrained word embeddings).

The micro- and macroaverage f-measures for precision and recall identified in the CoNLL-2003 shared task dataset were adopted as a system of measurement. Because this study only focused on interpreting the influence of statistical models on Urdu NER, we selected baseline ML approaches (eg, CRF and ANN) instead of state-of-the-art rule-based approaches. The results demonstrate that the proposed approach significantly outperforms previous CRF- and ANN-based approaches.

5.1 | Datasets

The effectiveness of NLP tasks depends on the particular approaches adopted by different algorithms (ie, statistical or rule-based approaches) and the availability of gold-standard data.

Our DRNN models were evaluated using the IJCNLP-2008 dataset and a portion of the dataset from [11], which we refer to as the Jahangir et al dataset. Additionally the proposed DRNN models were evaluated on the UNER news dataset [24]. This dataset contains text from three subdomains of the BBC Urdu website: sports, national news, and international news. The dataset was annotated manually with NE tags (person, location, organization, title, number, date, and time) and contains 48 673 tokens. Annotation of the UNER dataset was performed with help from two native Urdu speakers who are experts in the field of NE annotation mechanisms. Both annotators had extensive experience with the text annotation process and a deep knowledge of the Urdu language.

5.2 | Features

In almost all approaches, achieving the best performance relies on constructing an optimal set of features. In fact, an optimal feature set positively affects performance more than the choice of a learning model. Many progressive NER systems use a variety of linguistic features, ranging from semantic information on words to information on the morphological and syntactic structures of words. In most contemporary approaches to NLP tasks, it is common to learn word representations from labeled or unlabeled data initially, and then use these representations

[2]https://github.com/zhongkaifu/CRFSharp

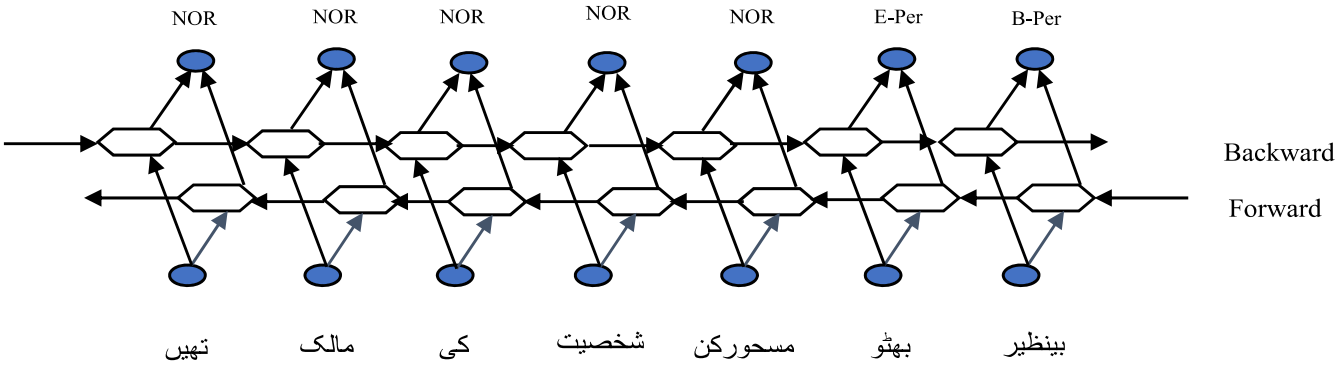

FIGURE 7 Illustration of a bidirectional LSTM network with inputs and NE outputs

as features in supervised algorithms. The following subsections detail the features that we used for training.

5.2.1 | Template features

Template features are also known as sparse features. In this type of feature, if the current token contains some features, then values of only those features become non-zero, but the other feature values remain at zero. The goal of template/sparse features is to develop feature sets from training and testing datasets using defined templates. In this study, the template features we used consisted of both linguistic features, such as POS information, and windows of context words. Specifically, the template features were: previous lexical word, current lexical word, next lexical word, POS of the current word, bigram of current word and its POS, bigram of $(N + 1)$ word + POS tag of $(N + 1)$ word, bigram of $(N - 1)$ word + POS tag of $(N - 1)$ word, bigram of $(N - 2)$ word + POS tag of $(N - 2)$ word, bigram of $(N + 2)$ word + POS tag of $(N + 2)$ word, length of current token, and suffix of current word in the length three.

Our template files are organized in rows such that each row represents a single template and is stored in a separate text file. All templates have three components, namely a prefix, ID, and rule-string, where the prefix is related to template typecasting. The two supported prefix types are "*U*" for unigram and "B" for bigram. The ID is used to discriminate templates from each other and the rule-string portion of the template guides the DRNN in producing features. The DRNN stores features in a separate model file generated from training records according to the given templates.

5.2.2 | Context template features

Context template features are based on features generated by templates for a particular token and are identified by referencing the context setup. In our case, the context setup for such features is "−2, −1, 0, 1," where "0" refers to the current token, "1" refers to the $(N + 1)$ token, and "−1" and "−2" refer to the $(N - 1)$ and $(N - 2)$ tokens, respectively. This context setup combines the features of the current token with the neighboring $(N + 1)$, $(N - 1)$, and $(N - 2)$ tokens. Consider the following Urdu sentence: صفائی نصف ایمان ہے (Safai nisf emaan hey), which means "Cleanliness is half of faith."

Suppose our current token is ایمان (Emaan) in the given sentence. Then, the generated feature set will be

{Template feature of ("نصف, Nisf"), Template feature of ("صفائی, Safai"), Template feature of ("ایمان, Emaan"), Template feature of ("ہے, hey")}.

5.2.3 | Word embedding features

Recently, the use of word embeddings as features for the NER of low-resource languages has attracted significant interest in the NLP research community. The rationale for the use of word vector embeddings is that words occurring close to each other should acquire congruent NE labels. Word embedding features, known as dense features, play a vital role when an unlabeled corpus is much larger than a labeled corpus. These features are used to describe the characteristics of a given token. In word embedding features, each token can point to a vector and the value of that vector will be used as the embedding feature. The vectors of corresponding words are produced in an unsupervised manner. Word embeddings correspond to the grouping of similar words to capture the diverse aspects of their meanings and the phrase-based information within vector representation features. This improves NLP task performance because it resolves the issue of data scarcity for low-resource languages. In this study, the Txt2Vec[3] project, which is an open-source C# package, was adopted to obtain word embeddings from unlabeled data using a continuous bag of words representation model. This implementation of word embeddings aims to extract vectors from related words and phrases such that each vector can potentially be employed as a feature. Various characteristics,

[3]https://github.com/zhongkaifu/Txt2Vec

such as incremental training capability and model vector quantization, are additional features of this package. Vector generation is based on the semantics of words, which are measured based on cosine similarity.

5.2.4 | Runtime features

In RNNs, the outputs of the current state are largely dependent on previous memory unit computations. Additionally, the parameters for each recurrent step are shared. Runtime features extend the feature pool by employing previous output tokens as features for the current token. Such features are only available for forward RNNs. Bidirectional RNNs do not support these features. In the configuration file of the RNNSharp library, runtime features are tagged with numbers such as "−1," "−2," or "−3." A tag of −1 means that the output of the previous token will be used in the current token calculation, −2 means that the outputs of the two previous tokens will be used in the current token calculation, and −3 indicates that outputs of the three previous tokens will be used in the current token calculation.

5.3 | Training and testing

We propose a DRNN approach that was tested against ANN and CRF baselines using C#-based open source libraries, namely RNNSharp,[4] NeuralNetwork,[5] and CRFSharp.[6] CRFSharp and RNNSharp are relatively new CRF and DRNN packages that should be easily generalizable for use with other open-source libraries. The main reason for using CRFSharp is that its model parameters include limited-memory Broyden-Fletcher-Goldfarb-Shanno encoding capability. Furthermore, it is more advanced than CRF++[7] in that it employs complete parallel encoding and computer memory utilization in an optimized manner to provide faster computation. During the training phase, when using a large volume of data with numerous tags, CRFSharp has superior capability to manage the memory of multicore CPUs efficiently at full usage compared to CRF++. Therefore, in a homophonic environment, CRFSharp is capable of encrypting more complex models at a lower cost than CRF++. Similarly, RNNSharp is an advanced form of the DRNN model that is widely used for many automatic linguistic and sequence labeling tasks, including NER. The core feature of RNNSharp is its support of various RNN model types and structures. For example, RNNSharp supports both BPTT and LSTM [25]. In terms of memory and output layers, its RNN structure supports RNN-CRF for both forward and bidirectional RNNs. Regarding BPTT and LSTM, a BPTT-RNN is typically called a "simple RNN" because the structure of its hidden layer nodes is very simple. In our experiments, for all model architectures, we used a hidden layer size of 200 with an alpha value of 0.1. We conducted experiments using the IJCNLP-2008 dataset, Jehangir et al dataset, and UNER news dataset. Because CRFSharp and RNNSharp require input data with the SBME[8] tagging format, we converted the data from all three datasets into this format. Consider the Urdu sentence کمانڈر تھے بریگیڈیئر ایڈ بٹلر سنہ دو ہزار چھ میں ہلمند کے فوجی, which means "In 2006, Brigadier Ed Butler was Helmand's military commander" (structure given in Table 1).

S indicates that an entity is single, while B, M, and E represent the beginning, middle, and end portions of an entity, respectively, and NOR indicates that a token is a normal term, rather than an entity.

Our proposed deep-learning-based system for Urdu NER consists of two stages. The first stage makes use of a restricted amount of labeled data and the second stage makes use of a large amount of unlabeled data. This type of learning is called semisupervised learning because it makes use of both labeled and unlabeled data (see Figure 8).

6 | RESULTS

Our proposed DRNN method was tested against the CRF model proposed in [9] and the ANN model proposed in [5] for Urdu NER. Our model outperformed the other models by a significant margin.

TABLE 1 Training file format

Token	POS Tag	Class type
بریگیڈیئر	PNN	S_TITLE
ایڈ	PNN	B_PERSON
بٹلر	PNN	E_PERSON
سنہ	PNN	B_DATE
دو ہزار چھ	PNN	E_DATE
میں	PSP	NOR
ہلمند	PNN	S_LOCATION
کے	PSP	NOR
فوجی	JJ	NOR
کمانڈر	NN	NOR
تھے	VBF	NOR
۔	PU	NOR

[4]https://github.com/zhongkaifu/RNNSharp

[5]https://github.com/trentsartain/Neural-Network

[6]https://github.com/zhongkaifu/CRFSharp

[7]http://crfpp.googlecode.com/svn/trunk/doc/in-dex.html

[8]https://github.com/zhongkaifu/CRFSharp

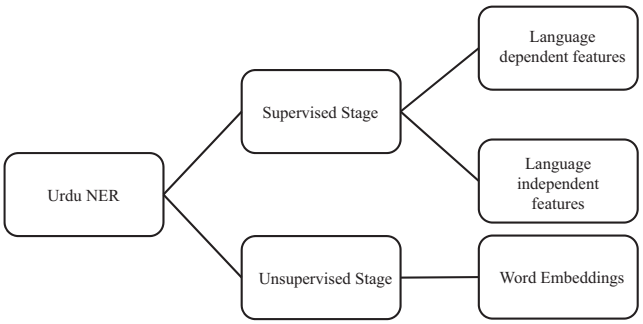

FIGURE 8 Schematic representation of the proposed deep-learning-based Urdu NER system

We performed 10-fold cross-validation experiments for each DRNN model, as well as for the ANN and CRF models. For the DRNNs, we used a data ratio of 70% for training, 10% for validation, and 20% for testing. We used various DRNN structures to generate results, namely LSTM-forward, LSTM-bidirectional, BPTT-forward, and BPTT-bidirectional. The results demonstrate that our proposed DRNN models outperform the baseline approaches on benchmark datasets.

Table 2 lists the global average precision, recall, and f-measure values of the proposed and baseline approaches on the INCNLP-2008 dataset. Table 3 lists the values for the Jehangir et al dataset. Table 4 lists the values for the UNER news dataset.

7 | DISCUSSION AND ERROR ANALYSIS

We hypothesized that by employing numerous DRNN model frameworks with combined template features and word embeddings, we could improve on traditional linear-chain CRF-based Urdu NER systems. The baseline methods examined in this study, namely linear-chain CRFs and traditional ANNs, lack comprehensiveness. They do not make use of language-dependent features, such as POS tags, meaning the tagging accuracy of these models may suffer. Additionally, while

TABLE 2 Global average results for the proposed and baseline approaches on the IJCNLP-2008 dataset

Model	Precision (%)	Recall (%)	f-Measure (%)
Baseline approach			
Linear-chain CRF	56.49	72.36	59.97
Artificial neural network	55.69	70.16	58.32
Proposed approach			
LSTM-forward RNN	57.37	72.62	61.20
LSTM-bidirectional RNN	61.20	70.01	63.21
BPTT-forward RNN	59.29	68.61	60.69
BPTT-bidirectional RNN	54.00	70.59	58.37

TABLE 3 Global average results for the proposed and baseline approaches on the Jehangir et al. dataset

Model	Precision (%)	Recall (%)	f-Measure (%)
Baseline approach			
Linear-chain CRF	81.12	77.26	77.25
Artificial neural network	79.01	77.59	76.46
Proposed approach			
LSTM-forward RNN	79.71	81.56	79.94
LSTM-bidirectional RNN	78.22	80.24	78.92
BPTT-forward RNN	77.05	79.88	78.10
BPTT-bidirectional RNN	7.06	79.37	77.85

TABLE 4 Global average results for the proposed and baseline approaches on the UNER dataset

Model	Precision (%)	Recall (%)	f-Measure (%)
National domain			
Baseline approach			
Linear-chain CRF	78.38	71.08	73.41
Artificial neural network	77.26	72.00	73.42
Proposed approach			
LSTM-forward RNN	73.24	76.67	74.09
LSTM-bidirectional RNN	74.20	77.16	74.90
BPTT-forward RNN	66.12	69.57	66.77
BPTT-bidirectional RNN	73.35	75.97	73.48
International domain			
Baseline approach			
Linear–chain CRF	67.75	72.67	67.29
Artificial neural network	68.02	73.27	67.78
Proposed approach			
LSTM-forward RNN	69.56	71.28	68.59
LSTM-bidirectional RNN	69.25	71.06	69.06
BPTT-forward RNN	66.96	70.53	67.50
BPTT-bidirectional RNN	69.24	72.24	69.80
Sports domain			
Baseline approach			
Linear-chain CRF	82.79	79.44	79.53
Artificial neural network	82.36	79.20	79.11
Proposed approach			
LSTM-forward RNN	84.75	79.35	81.10
LSTM-bidirectional RNN	84.13	79.13	80.90
BPTT-forward RNN	83.54	77.61	79.31
BPTT-bidirectional RNN	84.06	76.86	79.12

TABLE 5 Matrix of conflation for the BPTT-Bidirectional RNN on the international news domain of the UNER dataset

	Per:	Loc:	Org:	Title	Number	Date	Time
Person	119	16	24	8	1	3	0
Location	8	174	7	1	11	0	0
Organization	48	25	161	5	6	0	0
Title	13	3	5	28	0	0	0
Number	9	8	2	2	53	1	4
Date	10	20	6	0	3	60	2
Time	4	1	8	1	5	2	9

CRF models assign tags after reading an entire sequence, the proposed DRNN models attach labels to words directly. Additionally, input and output sequences are directly associated with each other in CRF models, whereas the recurrent cells in RNNs can handle interconnected input and output sequences. Furthermore, RNNs make use of word embeddings that correspond to the grouping of similar words to capture diverse meanings, as well as phrase-based information within vector representation features, meaning the probability that similar entities will be close to each other increases.

After testing, we observed that the proposed model tended to conflate person NEs with location and organization NEs. Similarly, organization NEs were conflated with person and location NEs. Finally, organization NEs were conflated with person and location NEs. The majority of these conflated tag errors are caused by a phenomenon in Urdu in which a single word can have different meanings in different contexts. For example, the Urdu word دو (du) can be part of a number entity (دوہزار, two thousand), part of a date entity پانچ ہزار دو سنہ, 2005 AD), or part of a time entity (دو بجے, 2 o'clock), and can also be used as a normal term. In our original test data, the word دو (du) appears 20 times. It acquired a date tag seven times, a number tag seven times, a time tag twice, and a NOR tag four times. After testing, the same word acquired a person tag once, an organization tag once, a number tag eight times, a date tag five times, a time tag once, and a NOR tag four times.

The results in this article demonstrate the superiority of DRNN models for Urdu NER tasks. Additionally, these results are the best results reported to date for Urdu NER when using ML and traditional neural network techniques. Details regarding frequently conflated tags when using the RNN-BPTT-bidirectional model are listed Table 5.

8 | CONCLUSION

The development of accurate and robust NER tools is essential in the modern era because the tasks of machine translation and information extraction are highly dependent on accurate NER systems. Further development of standard NER methodologies is vital for enriching Urdu language technology. The most striking feature of a DRNN that led us to adopt it for Urdu NER is its flexibility of operation over a sequence of vectors, which can be input or output sequences, or both. In this paper, we presented DRNN models with word embeddings as features for Urdu NER and carefully compared various DRNN architectures to CRF and ANN models. Our DRNN approach outperformed the CRF- and ANN-based approaches by a considerable margin. We found that the different DRNN models perform competitively. We used word embeddings as features alongside both language-dependent and language-independent features. We believe that this choice of features accounts for the improved performance compared to linear-chain CRF and traditional ANN models. In this study, we tested deep learning models on the Urdu NER task. These models are solely based on word level representations and word embeddings. However, in the future, testing such models with character-level representations and character embeddings is one option to improve performance further.

ORCID

Wahab Khan https://orcid.org/0000-0002-5694-0419

REFERENCES

1. A. Daud, W. Khan, and D. Che, *Urdu language processing: a survey*, Artif. Intell. Rev. **47** (2017), no. 3, 1–33.
2. E.F.T. Kim Sang and F. de Meulder, *Introduction to the CoNLL-2003 shared task: language-independent named entity recognition*, in Proc. Conf. Nat. Lang. Lear., HLT-NAACL, Edmonton, Canada, 2003, pp. 142–147.
3. U. Singh, V. Goyal, and G.S. Lehal, *Named entity recognition system for Urdu*, in Proc. COLING, Mumbai, India, 2012, pp. 2507–2518.
4. S. Mukund, R. Srihari, and E. Peterson, *An information-extraction system for Urdu–-a resource-poor language*, ACM Trans. Asian Language Inf. Process. **9** (2010), no. 4, 15:1–43.
5. M.K. Malik, *Urdu named entity recognition and classification system using artificial neural network*, ACM Trans. Asian Language Inf. Process. **17** (2017), no. 1, 2:1–13.
6. K. Riaz, *Rule-based named entity recognition in Urdu*, in Proc. Named Entities Workshop, Uppsala, Sweden, July 2010, pp. 126–135.
7. W. Khan et al., *A survey on the state-of-the-art machine learning models in the context of NLP*, Kuwait J. Sci. **43** (2016), 66–84.
8. D. Becker and K. Riaz, *A study in Urdu corpus construction*, in Proc. Workshop Asian Language Resoures. Int. Standardization, 2002, pp. 1–5.

9. M.K. Malik and S.M. Sarwar, *Urdu named entity recognition and classification system using conditional random field*, Sci. Int. **5** (2015), 4473–4477.
10. A. Ekbal et al., *Language independent named entity recognition in Indian languages*, in Proc. IJCNLP NER South South East Asian Languages, Hyderabad, India, Jan. 2008, pp. 33–40.
11. F. Jahangir et al., *N-gram and gazetteer list based named entity recognition for Urdu: A scarce resourced language*, in Proc. Workshop Asian Language Resources, Mumbai, India, Dec. 2012, pp. 95–104.
12. S.K. Saha et al., *Named entity recognition in Hindi using maximum entropy and transliteration*, Polibits **38** (2008), 33–41.
13. K. Gali et al., *Aggregating machine learning and rule based heuristics for named entity recognition*, in Proc. IJCNLP NER South South East Asian Languages, Hyderabad, India, Jan. 2008, pp. 25–32.
14. P. Kumar and V.R. Kiran, *A hybrid named entity recognition system for South Asian languages*, in Proc. IJCNLP NER South South East Asian Languages, Hyderabad, India, Jan. 2008, pp. 83–88.
15. S. Naz et al., *Challenges of Urdu named entity recognition: a scarce resourced language*, Res. J. Appl. Sci. Eng. Tech. **8** (2014), 1272–1278.
16. Q. Abbas, *Morphologically rich Urdu grammar parsing using Earley algorithm*, Nat. Lang. Eng. **22** (2016), 775–810.
17. A.Z. Syed, *Redefining Urdu morphology and grammar for the development of an integrated sentiment analysis framework*, PhD dissertation,University of Engineering & Technology, Lahore, 2013.
18. M. Humayoun, H. Hammarström, and A. Ranta, *Urdu morphology, orthography, and lexicon extraction*, in Workshop Computat. Approaches Arabic Script-Based Language, July 2007, pp. 1–8.
19. R. Pascanu et al., *How to construct deep recurrent neural networks*, arXiv preprint arXiv: 1312.6026, 2013.
20. J. Schmidhuber, *Learning complex, extended sequences using the principle of history compression*, Neural Computat. **4** (1992), no. 2, 234–242.
21. V.M. Janakiraman, *Explaining aviation safety incidents using deep learned precursors*, arXiv preprint arXiv: 1710.04749, 2017.
22. A. Graves and J. Schmidhuber, *Framewise phoneme classification with bidirectional LSTM and other neural network architectures*, Neural. Netw. **18** (2005), no. 5–6, 602–610.
23. C. Gulcehre et al., *Learned-norm pooling for deep feedforward and recurrent neural networks*, in Proc. Joit Euro. Conf. Mach. Learning. Knowledge Discovery Databases., Nancy, France, Sept. 2014, pp. 530–546.
24. W. Khan et al., *Urdu named entity dataset for urdu named entity recognition task*, in Proc. Sixth Int. Conf. Lang. Tech., 2016, pp. 51–56.
25. S. Hochreiter and J. Schmidhuber, *Long short-term memory*, Neural. Comput. **9** (1997), 1735–1780.

AUTHOR BIOGRAPHIES

Wahab Khan has earned his MS degree in Computer Science from the University of Science and Technology, Bannu, Khyber Pakhtunkhwa, Pakistan in 2009. He is pursuing his PhD in Computer Science at the International Islamic University, Islamabad, Pakistan. His research interests include natural language processing, machine learning, deep learning, and data mining.

Ali Daud obtained his PhD from Tsinghua University, China in 2010. He is currently working as an Associate Professor at the Department of Computer Science and Software Engineering of the International Islamic University, Islamabad, Pakistan. He is head of the Data Mining and Information Retrieval Group of the International Islamic University, Islamabad, Pakistan. He has published 70 papers in reputable international impact factor journals and conferences. He has supervised five PhD, 22 MS, and 18 BS dissertations/theses. He has taken part in many research projects and was also the principal investigator of two projects. His research interests include data mining, social network analysis and mining, probabilistic models, scientometrics, and natural language processing.

Fahd Alotaibi earned his PhD in Computer Science from the University of Birmingham, United Kingdom in 2015. He is currently working as an Assistant Professor in the Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia. His research interests include natural language processing, social networking, and data mining.

Naif Aljohani earned his PhD in Computer Science from the University of Southampton, United Kingdom in 2014. He is currently working as an Assistant Professor in the Faculty of Computing and Information Technology of King Abdulaziz University, Jeddah, Saudi Arabia. His research interests include linked open data and social network data mining.

Sachi Arafat earned his PhD in Computer Science from the University of Glasgow, United Kingdom in 2008. He is currently working as an Assistant Professor in the Faculty of Computing and Information Technology of King Abdulaziz University, Jeddah, Saudi Arabia. His research interests include the foundations of information science, philosophy of technology, and new media applications of quantum theory outside of physics, data science, and information retrieval.