Arabic Named Entity Recognition Using Variant Deep Neural Network Architectures and Combinatorial Feature Embedding Based on CNN, LSTM and BERT

Manel Affi
RIADI / ENSI
University of Manouba / Tunis
Tunisia
manel.affi@ensi-uma.tn

Chiraz Latiri
LIPAH / FST
University of Tunis El Manar/ Tunis
Tunisia
chiraz.latiri@gnet.tn

Abstract

One of the most important factors that significantly affects the quality of sequence labeling models is the selection and encoding of input features. The complexity and morphological richness of Arabic language is the main reason why most existing Arabic Named Entity Recognition (NER) systems rely heavily on hand-crafted feature engineering. To overcome such a limitation, we propose a novel neural network architecture to tackle the Arabic NER task. The proposed approach takes advantage of the recent success of deep neural networks in various Natural Language Processing (NLP) applications. We use a variant deep neural network architecture combined with a combinatorial feature embedding based on Convolutional Neural Network (CNN), Long Short-Term Memory networks(LSTM) and BERT to generate rich semantic and syntactic word representation vectors. Without using any external knowledge or hand-crafted feature engineering, the proposed models outperform the state-of-the-art systems on the ANERCorp dataset by yielding an F1-score of 93.34% and 93.68 % using bidirectional LSTM-CRF (BLC) and bidirectional GRU-CRF (BGC) architectures, respectively.

1 Introduction

Named Entity Recognition (NER) is defined as an information extraction task which aims to identify, extract and automatically categorize named entities into a set of predefined categories such as persons, organizations, locations, etc (Nadeau and Sekine, 2007). It is also considered as a mandatory preprocessing module in many wider NLP applications, such as information retrieval, speech recognition, syntactic parsing, machine translation, text classification, question answering, entity coreference resolution and entity linking and disambiguation.

There is a fair amount of literature on NER research in English, Chinese and other widely spoken languages. The massive growth of Arabic content on the web has led to increased demand for developing accurate and robust Arabic NLP tools. In recent years, Arabic NER has become a challenging task and has received increasing attention from current researchers due to its characteristics and peculiarities (Farghaly and Shaalan, 2009), and given the limited availability of annotated datasets (Zirikly and Diab, 2015).

Researchers interested in Arabic NER have mainly pursued three approaches: rule-based (Mesfar, 2007; Zaghouani, 2012), machine learning-based (Benajiba and Rosso, 2008) and hybrid methods (Abdallah et al., 2012; Oudah and Shaalan, 2017). All these three methods suffer from the same problem, as it requires a lot of language-specific knowledge and feature engineering to obtain useful results. This is even more highlighted by the deficiency of linguistic resources and the complex morphology and syntax of the language.

Recently, the deep learning paradigm (Mishra and Gupta, 2017) has emerged and led to impressive advances in fields such as speech processing (Oord et al., 2016) and image recognition (Hu et al., 2018). For NLP, the application of deep learning has proven to be very effective as it yields the state-of-the-art in

various common NLP tasks such as sequence labeling (Ma and Hovy, 2016) and named entity recognition (Affi and Latiri, 2021) for the English language. Unlike traditional methods, deep learning is an end-to-end model that does not rely on data pre-processing, manual feature extraction or large amounts of task-specific resources, and it can be applicable to different languages and domains. This makes it an attractive solution for complex and low resource languages like Arabic.

Motivated by the success of deep learning in many NLP tasks, we propose a novel Arabic NER approach based on deep neural networks. The proposed model takes benefits from CNN and LSTM to induce character-level representations of words. These representations are fed in conjunction with BERT word embeddings to a deep learning model for further sequence modeling. Two RNN models are investigated in this work: the Bi-LSTM and Bi-GRU models. Finally, we use a Conditional random fields (CRF) layer to get the probability distribution over the tags.

The main contributions of the proposed Arabic NER model are summarized as follows:

- We propose a novel neural network architecture based on variant deep neural network architectures combined with word embeddings based on the CNN, LSTM and BERT to tackle the Arabic NER task.
- We study the impact of the combinatorial feature embedding based on the CNN, Bi-LSTM and BERT on Arabic NER.
- We investigate the use of two types of character-level representations to slove the Arabic NER task. We also show that using pre-trained word embeddings enhances the system performance.
- We give an empirical evaluation of this system on the ANERCorp dataset.
- We evaluate our model against different baselines to demonstrate the empirical strength of our work.

The remainder of this paper is structured as follows. In the next section, we review the related work briefly. Section 3 presents the proposed model architecture in detail. Section 4 describes the training mechanism. Section 5 presents the experimental results and discussions. Finally, we conclude and discuss possible improvements for future work in section 6.

2 Related work

Recent NER research studies have used deep learning, which proves its powerful ability for feature abstraction. As far as Arabic NER is concerned, some consideration has been specified to neural models. (Mohammed and Omar, 2012) proposed one of the first artificial neural network for Arabic NER. The authors introduced a simple feed forward ANN to tackle the task. The suggested model consists of three steps: data pre-processing, transforming Arabic letters to Roman letters, after that the collected data is categorised using a neural network. This approach was tested on the ANERCorp dataset. The obtained results showed that the proposed method gave better performance than the decision trees, and the model accuracy improved with the size of the data. A tagging model for Arabic NER was proposed by (Awad et al., 2018). To solve the out-of-vocabulary (OOV) problem, a CNN character-based embedding layer was concatenated with Word2Vec word embeddings to initialize the word representations. Then, the vector representations were fed to a Bi-LSTM-CRF architecture. The system was evaluated on the ANERCorp and AQMAR datasets. Many hyper-parameter setting were checked, and the model achieved an F1-score of 75.68% . Recently, the attention mechanism has demonstrated its effectiveness in different NLP applications such as machine translation. It can handle long input sequences and learn to focus only on the most important words. In (Ali et al., 2018), the authors added a self-attention layer to enhance their Arabic NER model. The proposed model learnt to give low or high attention to words based on their context in the input sentence. Also, the writers in (Ali et al., 2019) introduced a multi-attention based model for Arabic NER. This proposed system showed the impact of using word-level embeddings and character-level representations followed by a Bi-LSTM and self-attention layers. Tested on the ANERCorp dataset,

it achieved an F1-score of 91.31% .
A recent work (Al-Smadi et al., 2020) examined the impact of transfer learning on a Pooled-GRU architecure for Arabic NER. Their model outperformed the Bi-LSTM-CRF model suggested by (Sa'a et al., 2018).

3 Proposed neural network architecture

In this section, we describe the architecture of our neural network model based on BERT word embeddings and CNN and LSTM-based character level vectors for the Arabic NER task. The proposed system comprises three main layers, namely an word embedding layer, a context layer, and a tag decoder layer.
Definition: Assuming an input sentence S coming from a series of tokens of size $\| V \|$ with a sequence of labels $Y = (y_1, ..., y_n)$, its word-level input is defined as $X = (x_1, ..., x_n)$, where x_i is the i^{th} token in a series of words of size $\| n \|$, and n is the number of the tokens in the input sentence. With a neural sequence labeling system, the objective is to find the adequate entity labels for all tokens in X, and then assign a sequence of annotations $\tilde{y} \in Y$ to it, where Y is the list of all possible tag categories. The highest probable symbol sequence $\tilde{y}$ is outputted by maximizing the sequence posterior probability of an optimal resulting sequence $\tilde{y} = (\tilde{y_1}, \tilde{y_2}, ..., \tilde{y_n})$, which closely matches the gold annotations sequence $y = (y_1, y_2, ..., y_n)$ that indicates the right label for each given token.
As shown in Figure 1, our proposed model is composed of three main blocks which are explained below.

3.1 Embedding layer

The representation layer is designed to provide the main features that will be used as a key component for our NER system. The quality of features has a significant impact on the model performance. Traditionally, features are hand-crafted keeping some interesting rules that may not be relevant to other areas. Thus, many state-of-the-art NLP approaches tend to use various deep neural networks architecture for outputting distributed word representations in order to catch both syntactic and semantic patterns of words. In distributed embeddings,

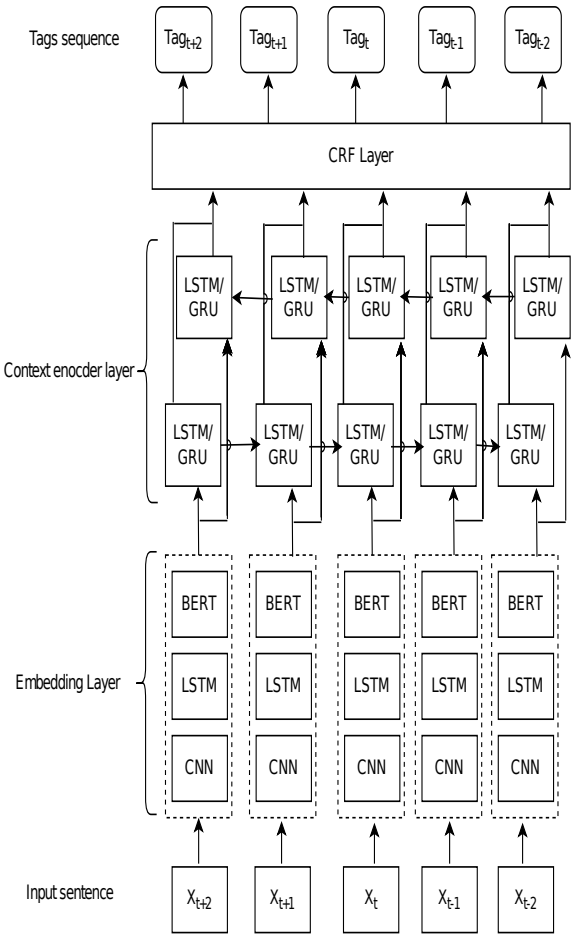

Figure 1: Main model architecture

the model is generalizable because each word refers to dense low-dimensional real-valued vectors in the space, so that tokens with approximated semantic and syntactic properties will have similar vector representations. However, learning high-quality word vectors can be challenging. Ideally, they should represent successfully both complex characteristics of word uses and how these uses change according to their linguistic context. Hence, word embeddings have attracted a lot of consideration from many researchers (Mikolov et al., 2013; Lai et al., 2016).
Over the last years, different tools, such as word2vec (Mikolov et al., 2013) and GloVe (Pennington et al., 2014) have been widely used in the NLP field. In order to reach high-quality of word embeddings, researchers have recently introduced different techniques to generate various embeddings for the same word depending on its context (Devlin et al., 2018;

Peters et al., 2018).
However, using a word embedding alone as the smallest feature representation unit may result in some loss of important information. For morphologically rich languages, such as Arabic, we need to catch all morphological and orthographic patterns. As the word embedding encodes semantic and syntactic word relationships, character-level representations model important morphological and shape information. Inspired by this integration, we acquire the word representations from BERT word embeddings and two different character-level representations extracted from a CNN and LSTM.

3.1.1 Word embedding layer

In distributed embeddings, words refer to a vector in a continuous space to capture syntactic and semantic relations among them. In this work, we use BERT (Devlin et al., 2018) as distributed word embeddings. BERT was published by researchers at Google AI Language in late 2018. Its full name is Bidirectional Encoder Representations from Transformers. It represents a language model representation based on self-attention blocks. The main innovation of this model is the pre-training approach, which determines word and sentence-level representations based on masked-language modeling and next sentence prediction training. BERT is pre-trained in various languages using existing unlabeled data. The pre-trained deep bidirectional model with one output layer has become a state-of-the-art in many NLP applications such as multi-genre natural language inference, named entity recognition and question answering. The idea is to have a common architecture adequate for many NLP applications and a pre-trained model that decreases the need for labeled data and boost the performance for different downstream NLP tasks.
To obtain the pre-trained contextual word embedding for Arabic from BERT model, we make use of a publicly available pre-trained resource for research purposes: AraBERT[1] (Antoun et al., 2020), with a dimension of 768 and trained on 70M sentences with 3B words of Arabic text. AraBERT uses the same Google's BERT architechture and BERT-Base[2] configuration with 768 hidden layers, 12 layers of transformer encoders and 12 attention heads. For each token, a contextual word embedding is generated by averaging the corresponding word, subword and position embeddings in the last four layers.

3.1.2 Character embedding layer

In addition to word embeddings, character-level embeddings are also used to represent input words. It is used to handle OOV words that are not present in the trained word vectors. Especially for the Arabic language which is rich in morphology and syntax, character-level embeddings help to efficiently extract morphological patterns of each token. We use two different neural networks, the CNN and LSTM, to extract the character-level patterns.

Character-level CNN model
Among deep learning techniques, the CNN is a well-known architecture that is widely used to capture local information. CNNs, which were originally used for image processing (Krizhevsky et al., 2012), are now also used to capture character-level information for various NLP tasks. The authors in (Labeau et al., 2015) and (Santos and Guimaraes, 2015) successfully studied the use of CNNs to model character-level features in POS-tagging and NER tasks, respectively. The writer in (Collobert et al., 2011) also used CNNs to improve the semantic role labeling task. The authors in (Kim et al., 2016) proposed an interesting CharCNN model, which is a character-aware neural language model that learns character-based word representations using CNNs.
In this approach, we also investigate the use of the CNN architecture to represent each character in the input token by a character vector. First, each token is mapped to a suitable representation using its character embeddings. This character embedding vectors are then passed through a convolutional layer using multiple filters to detect various features. Formally, for an input token W with T characters $\{ w_1, w_2, ..., w_T \}$, matrix $\Omega \in R^{d_{ch} * T}$ is learned to map the token into character embeddings, where d_{ch} is the dimension of the character representations.

[1] https://huggingface.co/aubmindlab/bert-base-arabertv01

[2] More details about the transformer architecture can be found in (Vaswani et al., 2017)

The convolution layer has as input the sequence of character embeddings $\{ \omega_1, \omega_2, ..., \omega_T \}$. Then, a convolution operation is used between Ω and filter $H \in R^{d_{ch} * w}$ of width h. After that, we add a bias and apply the $tanh$ function to add non-linearity to acquire the feature map $f \in R^T$:

$$f_i = tanh(< \Omega_{i:i+h-1}, H > +b) \quad (1)$$

where $< \Omega_{i:i+h-1}, H >$ denotes the Frobenius inner product and b represents the bias term.
Finally , a max-pooling operation (Eq 2) is applied to learn a single feature for all feature maps.

$$Q_H^W = max_i f_i \quad (2)$$

where Q_H^w represents the character-level embedding of a token W produced by filter H to capture the local information.

Character-level LSTM model
LSTM models are also used to extract character-level patterns. LSTM is a well-recognized scheme of learning long-term dependencies. The typical LSTM unit employs a memory cell controlled by a forget gate that determines which previous memory should be scaled into the next time-step. Similarly, the new input to memory cells is passed through an input gate.
Formally, the implementation of the LSTM memory unit is described as follows:

$$f_t = \sigma(W_f.[x_t, h_{t-1}] + b_f) \quad (3)$$

$$i_t = \sigma(W_i.[x_t, h_{t-1}] + b_i) \quad (4)$$

$$\tilde{c}_t = tanh(W_c.[x_t, h_{t-1}] + b_c) \quad (5)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \quad (6)$$

$$o_t = \sigma(W_o.[x_t, h_{t-1}] + b_o) \quad (7)$$

$$h_t = o_t \odot tanh(C_t) \quad (8)$$

where σ is the logistic sigmoid function, $\odot$ represents the element-wise multiplication, f, i, o and c are respectively the forget gate, the input gate, the output gate and the cell vector, x_t represents the input vector at the t^{th} time-step, h_t is the hidden layer state vector that saves all the important information at the t^{th} time-increment, W_f, W_i, W_c and W_o denote the weight matrices of the various gates, and b_f, b_i, b_c and b_o represent the bias vector.

3.2 Context encoder layer: Bi-LSTM/Bi-GRU

The context layer aims to get the local dependencies using neighboring words for every word. The local context is crucial for effectively predicting labels, since there could exist strong relationships among neighboring tokens in a sentence. Therefore, it is important to model the local context information for each word. In the NER task, it is common to use the RNN as a context encoder model. It treats the input sequence in order, where shuffling or reversing the time-steps influences the extracted representations from the input sequence. The longer the input sequence, the less precise the RNN becomes because it is difficult for the network to remember the output of the time steps far away from the previous (Goyal et al., 2018). This issue is named the vanishing gradient problem. LSTM and GRU are variations of the RNN that help in dealing with the vanishing gradient issue and can learn long dependency input. In the NER task, both previous and future information is helpful for prediction. For that, we use the Bi-LSTM and Bi-GRU networks as a context encoder layer after obtaining word embeddings from the concatenation (Eq.9) BERT model, CNN as well as the LSTM-based character embeddings B^{emb}, CN^{emb} and CL^{emb}, respectively.

$$X_t^{emb} = B_t^{emb} \oplus CN_t^{emb} \oplus CL_t^{emb} \quad (9)$$

where B_t^{emb} is is the embedding learned by BERT, CN_t^{emb} is the CNN-based character embedding, CL_t^{emb} represents the LSTM-based character embedding, and X_t^{emb} is the vector representation for the t^{th} word of the input sentence.

3.2.1 Bi-LSTM

In addition to learning character embedding, we use LSTM, which is formally explained in section 3.1.2 also to learn the contextual features of a sequence of words. To better represent the current word information, it is beneficial to obtain both prior and posterior contexts. In our model, we use Bi-LSTM to effectively predict the current label information. As input, Bi-LSTM takes the vector representation of each word X_t^{emb}. The basic idea is to present the sequence from left to right using a forward layer (Eq.10) and to compute a representation of the same sequence in reverse via a backward layer

(Eq.11). These two different networks use various parameters to learn privious and future patterns. The left and right context representations are then concatenated to form the final output (Eq.12)

$$\overrightarrow{h_t} = f(\overrightarrow{h_{t-1}}, [X_t^{emb}]) \quad (10)$$

$$\overleftarrow{h_t} = f(\overleftarrow{h_{t+1}}, [X_t^{emb}]) \quad (11)$$

$$h_t = \overrightarrow{h_t} \oplus \overleftarrow{h_t} \quad (12)$$

where $f(.)$ represents the unidirectional LSTM cell, $\overleftarrow{h_t}$ and $\overrightarrow{h_t}$ represent respectively the hidden states of the backward LSTM cell and the forward cell, and t is the index of the encoding steps.

3.2.2 Bi-GRU

The GRU was proposed by (Cho et al., 2014). Similarly to the LSTM cells, the GRU was modeled to adaptively update or reset its memory content by using reset and update gates which are similar to the forget and input gates of the LSTM unit. The update gate decides what information to use as input at the next time step, whereas the reset gate uses the previous time step output to decide which information should be removed and which information is useful for the current time step input. The update gate, the reset gate, and the hidden state h_t at time step t are updated utilizing the equations described below.

$$u_t = \sigma(Z_u.[x_t, h_{t-1}] + b_u) \quad (13)$$

$$r_t = \sigma(Z_r.[x_t, h_{t-1}] + b_r) \quad (14)$$

$$\tilde{h}_t = tanh(Z.[x_t, h_{t-1}.r_t] + b) \quad (15)$$

$$h_t = (1 - u_t).h_{t-1} + u_t.\tilde{h}_t \quad (16)$$

where σ denotes the logistic sigmoid function, u, r and $\tilde{h}$ respectively represent the update gate, the reset gate and the cell state, x_t is the input vector at time t, h_t represents the hidden layer state vector at time t, Z_u, Z_r and Z denote the weight matrices of the different gates and the cell state, respectively, and b_u, b_r and b represent the bias vector. In this work, we use a Bi-GRU to process the input sentence by a forward layer from left to right (Eq.17) and from the other side by a backward layer (Eq.18). Then, the forward layer hidden state and the backward layer hidden state are combined to represent the final hidden state (Eq.19).

$$\overrightarrow{h_t} = g(\overrightarrow{h_{t-1}}, [X_t^{emb}]) \quad (17)$$

$$\overleftarrow{h_t} = g(\overleftarrow{h_{t+1}}, [X_t^{emb}]) \quad (18)$$

$$h_t = \overrightarrow{h_t} \oplus \overleftarrow{h_t} \quad (19)$$

where $g(.)$ represents unidirectional GRU unit, $\overleftarrow{h_t}$ and $\overrightarrow{h_t}$ respectively denote the hidden states of the backward GRU unit and forward unit, and t is the index of the encoding steps.

3.3 Tag encoder layer: CRF layer

For sequence tagging tasks, it is beneficial to take into consideration the relationship between tags in neighborhoods; and for a given input sentence, it is important to jointly decode the label chain that yields the best resulting label sequence. In NER task with the IOB annotation, there is a strong dependency between labels; e.g., I-PER cannot follow I-ORG. Therefore, modeling the label correlations is important for the NER task (Ma and Hovy, 2016; Liu et al., 2018). Following (Ma and Hovy, 2016) and (Huang et al., 2015), we incorporate a CRF (Lafferty et al., 2001) layer upon the context layer to jointly capture the label correlations.

Formally, for a given input word sequence $X = (x_1, x_2, ...x_n)$ with a sequence of annotations $Y = (y_1, y_2, ...y_n)$, we denote Ω as the scoring matrix resulting from the context layer. The size of matrix Ω is n*m, where n and m denote the length of the sentence and the number of different labels, respectively. $\Omega_{i,j}$ is the score of the j^{th} label of the i^{th} word in the input sentence. We also use $Y_{(X)}$ to define the set of possible label sequences for X.

The score of a given sentence is determined through Eq.20:

$$S(X, y) = \sum_{i=0}^{n} \psi_{y_i, y_{i+1}} + \sum_{i=1}^{n} \Omega_{i, y_i} \quad (20)$$

where ψ denotes the matrix of transition scores. The transition score matrix values are learned during training. After that, a softmax function is applied to regularize the conditional probability of the output path y:

$$P(y|X) = \frac{\exp^{S(X,y)}}{\sum_{\tilde{y} \in Y_{(X)}} \exp^{S(X,\tilde{y})}} \quad (21)$$

During the training step, the aim is to maximize the following log-probability of the right label sequence:

$$\log(P(y|X)) = S(X, y) - \log(\sum_{\tilde{y}} \exp^{S\,(X,\tilde{y})}) \quad (22)$$

For the evaluation step, we aim to retrieve the most probable output label sequence y^* by maximizing the score function:

$$y^* = \arg\max_{\tilde{y} \in Y_{(X)}} S(X, \tilde{y}) \quad (23)$$

Finally, we employ the Viterbi (Forney, 1973) algorithm to train the CRF layer and decode the best sequence y^*.

4 Training mechanism

4.1 Dataset description

To prove the effectiveness of our Arabic NER system, we conduct extensive experiments using the ANERCorp[3] which was developed by Benajiba (Benajiba et al., 2007) from diverse online resources and which is freely available for research purposes. It has 4,901 sentences with 150,286 tokens of articles from Modern Standard Arabic. The ANERCorp dataset is manually annotated where each word was tagged with one of the following tags: person, location, company, and others. ANERCorp is corpus that has two corpora, namely training and testing corpora.

4.2 Evaluation metrics

In order to evaluate the efficiency of our models, the standard measures for NER precision (P), recall (R), and F1-Score (F) are computed.

The precision measure represents the percentage of name entities found by the system and which are correct. It can be computed as:

$$P = \frac{NumberOfCorrectNamedEntities}{NumberOfNamedEntitiesExtracted} \quad (24)$$

On the other hand, the recall measure represents the percentage of name entities that the system extracted correctly out of the overall number of named entities existing in the corpus, and it can be expressed as:

$$R = \frac{NumberOfCorrectNamedEntities}{TotalNumberOfNamedEntities} \quad (25)$$

Finally, the F1-score is the most commonly used and is computed based on the precision and recall. This is defined as:

$$F1 - score = 2X\frac{P * R}{P + R} \quad (26)$$

[3] https://camel.abudhabi.nyu.edu/anercorp/

4.3 Hyper-parameter settings

Our neural network model is implemented using Keras environment and TensorFLOW API. A word vector embedding represents the input of the system. We obtain the word vector using the concatenation of the pre-trained BERT word vector of dimension 768, CNN and LSTM-based character level representations where each of them is of dimension 20. The training is performed using the backpropagation algorithm to update the parameters of all models during the training process. We choose the Adam algorithm (Kingma and Ba, 2014) and we set a learning rate of value 1e-5. Our model uses two layers of the Bi-LSTM and Bi-GRU networks with hidden layer nodes of dimension 512. For the over-fitting problem, a 20% dropout is regarded as an additional measure to control the input of the neural model and alleviate the over-fitting issue very well. In each iteration, we split the entire training data into batches and pass one batch at a time. In our experiments, we train our model with a batch size of 64 for only three epochs.

5 Experimental results and discussions

In this section, we present the experimental results obtained by using different architectures based on two RNN variants, namely the Bi-LSTM and the Bi-GRU applied on ANERCorp dataset. We run three parts of experiments: The first part focuses on the selection of the best architecture giving the highest performance. In the second part, we discuss the obtained results, while the third part of the experiments presents a comparison between the best-selected architecture and the best previous state-of-the-art methods.

5.1 Results

To explore the performance of our proposed models and show the effect of different architectures. We have conducted a series of comparative experiments. We have examined five different word embedding choices: Randomly initialized Word Embeddings(RWE), BERT word embeddings, two different types of character embeddings based on the

CNN and LSTM and CNN-LSTM-BERT (CLB) combinatorial feature embeddings. All these word embeddings are tested in combination first with BLC and then with BGC models. Tables 1 and 2 show the obtained results.'Exp', 'P', 'R', and 'F1' denote experiment, precision, recall and F1-score, respectively.

Exp	Model	R	P	F1
1	RWE-BLC	85.40	84.91	85.15
2	CNN-BLC	90.62	90.43	90.52
3	LSTM-BLC	90.50	90.30	90.39
4	BERT-BLC	92.33	92.90	92.61
5	CLB-BLC	**93.20**	**93.50**	**93.34**

Table 1: Results obtained based on Bi-LSTM.

Exp	Model	R	P	F1
1	RWE-BGC	85.69	85.09	85.38
2	CNN-BGC	90.97	90.75	90.85
3	LSTM-BGC	90.80	90.57	90.68
4	BERT-BGC	92.58	93.10	92.83
5	CLB-BGC	**93.60**	**93.77**	**93.68**

Table 2: Results obtained based on Bi-GRU.

Related to the experimental results presented in Tables 1 and 2, we can observe that the combination of BGC and CLB combinatorial word embeddings exhibits the highest performance and outperforms the model based on BLC which reaches 93.20% precision, 93.50% recall and a 93.34 F1-score% by achieving 93.77% precision, 93.60% recall and a 93.68% F1-score. In table 1 and 2, In five different experiments, the obtained results show that the Bi-GRU performs better than Bi-LSTM by about 0.34% in the F1-score with the CLB combinatorial feature embeddings.

5.2 Effects of different word embeddings

To analyze the effects of different types of word embeddings, we conduct experiments using the BLC and the BGC with five different combinations of embeddings. The results are presented in Tables 1 and 2. In experiment 1, the model applies only the RWE. In experiments 2 and 3, the models use embeddings that combine one type of character-level embedding, the CNN and LSTM, respectively. In experiment 4, the models use pre-trained BERT word embeddings whereas in experiments 5, the models use a combinatorial feature based on the CNN, LSTM and BERT. Experiment 1 shows that the RWEs exhibit the lowest performance. The results of experiments 2 and 3 significantly outperform those of experiment 1, indicating that character-level embedding is useful for handling OOV words in Arabic NER tasks. Additionally, in experiments 2 and 3, the CNN model outperforms the LSTM model in extracting character-level features by 0.12% and 0.17% in the F1-score using BLC and BGC models, respectively. In experiment 4, we investigat the effect of using pre-trained BERT word embeddings. It demonstrates that pre-trained BERT embeddings have the most positive impact on the performance by achieving 92.33% and 92.58% F1-score using the BLC and BGC, respectively. In experiment 5, the proposed models utilizing all three types of embedding (CNN,LSTM and BERT) for word representation exhibit the highest performance achieving an F1-score of 93.34% 93.68% F1-score using BLC and BGC architectures, respectively. This indicates that in Arabic NER, the use of character-level based word representations and Arabic pre-trained word embeddings are effective for handling OOV words and effciently capturing semantic and syntactic word relationships.

5.3 Performance comparison with state-of-the-art methods

In this part of these experiments, we compare the performance of our proposed models with the models used in previous studies on ANERCorp. Table 3 presents the results of three competitive Arabic NER models from the studies of (Alsaaran and Alrabiah, 2021)(BERT-GRU), (Ali et al., 2019) (Multi-Attention) and (El Bazi and Laachfoubi, 2019) (CW-Bi-LSTM-CRF) evaluated on entity-level matching.

The authors in (Alsaaran and Alrabiah, 2021) introduced a novel Arabic NER model for Arabic based on the BERT and the Bi-GRU. This model achieves a 92.28 % F1-score on ANERCorp. In addition to word-level embeddings, the writers in (Ali et al., 2019) adopted character-level embeddings and combined them via an embedding-level attention mechanism followed by Bi-LSTM and a self-attention layer. This model yielded an F1-

Model	R	P	F1
BERT-BGRU	92.40	92.20	92.28
Multi-Attention	90.62	90.43	90.52
CW-Bi-LSTM-CRF	-	-	90,60
CLB-BLC (**our**)	**93.20**	**93.50**	**93.34**
CLB-BGC (**our**)	**93.60**	**93.77**	**93.68**

Table 3: Comparison between our best models and three Arabic NER systems on ANERCorp dataset.

score of 91.31% on ANERCorp. A Bi-LSTM-CRF based neural network model was introduced by (El Bazi and Laachfoubi, 2019). The proposed system got two sources of information about words as input: pre-trained word embeddings namely Skip-Gram, CBOW, GLOVE, FastText and Hellinger PCA and character-based representations. This approach yielded an F1-score of 90.60%.

Table 3 presents the comparison between our models and some state-of-the-art approaches. We outperform the best Arabic NER model (Alsaaran and Alrabiah, 2021) on ANERCorp by about 1.4 points. As far as we know, we are the first to use two types of character-level embedding based on the CNN and LSTM combined with BERT word embeddings as the main feature block for the Arabic NER task. Using these three types of word embeddings together allows the model to capture both important morphological and orthographic patterns as well as the semantic and syntactic word relationships.

6 Conclusion

In this paper, we have presented a technically simple neural network architecture using variant deep neural network architectures and a combinatorial feature embedding based on the CNN, LSTM and BERT. Several deep learning architectures have been investigated and evaluated on ANERCorp dataset. The experimental results have shown that our CLB-BLC and CLB-BGC models achieve superior outcomes over the existing deep learning models in Arabic NER with a high F-score of 93.34% and 93.68% respectively. Also, the evaluation has clearly demonstrated that:

- The CNN model outperforms the LSTM model in extracting character- level features.
- The Bi-GRU model performs better in modeling long-term dependencies for Arabic NER than the Bi-LSTM model. In our experiments, the Bi-GRU architecture outperforms the Bi-LSTM architecture by about 0.34% in the F1-score.
- The proposed models utilizing the combinatorial feature embedding based on the CNN, LSTM and BERT exhibit the highest performance achieving an F1-score of 93.34% and 93.68% F1-score using the BLC and BGC architectures, respectively.

One of the potential directions for future work will be to apply an embedding attention layers on top of the word and character level representations to dynamically determine the information that must be used to best represents the given token and to extend this work to solve the NER task for other languages.

References

Sherief Abdallah, Khaled Shaalan, and Muhammad Shoaib. 2012. Integrating rule-based system with classification for arabic named entity recognition. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 311–322. Springer.

Manel Affi and Chiraz Latiri. 2021. Be-blc: Bert-elmo-based deep neural network architecture for english named entity recognition task. *Procedia Computer Science*, 192:168–181.

Mohammad Al-Smadi, Saad Al-Zboon, Yaser Jararweh, and Patrick Juola. 2020. Transfer learning for arabic named entity recognition with deep neural networks. *Ieee Access*, 8:37736–37745.

Mohammed NA Ali, Guanzheng Tan, and Aamir Hussain. 2018. Bidirectional recurrent neural network approach for arabic named entity recognition. *Future Internet*, 10(12):123.

Mohammed Nadher Abdo Ali, Guanzheng Tan, and Aamir Hussain. 2019. Boosting arabic named-entity recognition with multi-attention layer. *IEEE Access*, 7:46575–46582.

Norah Alsaaran and Maha Alrabiah. 2021. Arabic named entity recognition: A bert-bgru approach. *CMC-COMPUTERS MATERIALS & CONTINUA*, 68(1):471–485.

Wissam Antoun, Fady Baly, and Hazem Hajj. 2020. Arabert: Transformer-based model for arabic language understanding. *arXiv preprint arXiv:2003.00104*.

David Awad, Caroline Sabty, Mohamed Elmahdy, and Slim Abdennadher. 2018. Arabic name entity recognition using deep learning. In *International Conference on Statistical Language and Speech Processing*, pages 105–116. Springer.

Yassine Benajiba and Paolo Rosso. 2008. Arabic named entity recognition using conditional random fields. In *Proc. of Workshop on HLT & NLP within the Arabic World, LREC*, volume 8, pages 143–153. Citeseer.

Yassine Benajiba, Paolo Rosso, and José Miguel Benedíruiz. 2007. Anersys: An arabic named entity recognition system based on maximum entropy. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 143–153. Springer.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(ARTICLE):2493–2537.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Ismail El Bazi and Nabil Laachfoubi. 2019. Arabic named entity recognition using deep learning approach. *International Journal of Electrical & Computer Engineering (2088-8708)*, 9(3).

Ali Farghaly and Khaled Shaalan. 2009. Arabic natural language processing: Challenges and solutions. *ACM Transactions on Asian Language Information Processing (TALIP)*, 8(4):1–22.

G David Forney. 1973. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278.

Palash Goyal, Sumit Pandey, and Karan Jain. 2018. Deep learning for natural language processing. *New York: Apress*.

Jie Hu, Li Shen, and Gang Sun. 2018. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2016. Character-aware neural language models. In *Thirtieth AAAI conference on artificial intelligence*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25.

Matthieu Labeau, Kevin Löser, and Alexandre Allauzen. 2015. Non-lexical neural architecture for fine-grained pos tagging. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 232–237.

John Lafferty, Andrew McCallum, and Fernando CN Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

Siwei Lai, Kang Liu, Shizhu He, and Jun Zhao. 2016. How to generate a good word embedding. *IEEE Intelligent Systems*, 31(6):5–14.

Liyuan Liu, Jingbo Shang, Xiang Ren, Frank Xu, Huan Gui, Jian Peng, and Jiawei Han. 2018. Empower sequence labeling with task-aware neural language model. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *arXiv preprint arXiv:1603.01354*.

Slim Mesfar. 2007. Named entity recognition for arabic using syntactic grammars. In *International Conference on Application of Natural Language to Information Systems*, pages 305–316. Springer.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26.

Chandrahas Mishra and DL Gupta. 2017. Deep machine learning and neural networks: An overview. *IAES International Journal of Artificial Intelligence*, 6(2):66.

Naji F Mohammed and Nazlia Omar. 2012. Arabic named entity recognition using artificial neural network. *Journal of Computer Science*, 8(8):1285.

David Nadeau and Satoshi Sekine. 2007. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

Mai Oudah and Khaled Shaalan. 2017. Nera 2.0: Improving coverage and performance of rule-based named entity recognition for arabic. *Natural Language Engineering*, 23(3):441–472.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

D A Alzboun Sa'a, Saia Khaled Tawalbeh, Mohammad Al-Smadi, and Yaser Jararweh. 2018. Using bidirectional long short-term memory and conditional random fields for labeling arabic named entities: A comparative study. In *2018 Fifth International Conference on Social Networks Analysis, Management and Security (SNAMS)*, pages 135–140. IEEE.

Cicero Nogueira dos Santos and Victor Guimaraes. 2015. Boosting named entity recognition with neural character embeddings. *arXiv preprint arXiv:1505.05008*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Wajdi Zaghouani. 2012. Renar: A rule-based arabic named entity recognition system. *ACM Transactions on Asian Language Information Processing (TALIP)*, 11(1):1–13.

Ayah Zirikly and Mona Diab. 2015. Named entity recognition for arabic social media. In *Proceedings of the 1st workshop on vector space modeling for natural language processing*, pages 176–185.