International Journal of Electrical and Computer Engineering (IJECE)
Vol. 9, No. 3, June 2019, pp. 2025~2032
ISSN: 2088-8708, DOI: 10.11591/ijece.v9i3.pp2025-2032

Arabic named entity recognition using deep learning approach

Ismail El Bazi, Nabil Laachfoubi
IR2M Laboratory, FST, Univ Hassan 1st, Settat, Morocco

Article Info

Article history:

Received Apr 27, 2018
Revised Nov 20, 2018
Accepted Dec 10, 2018

Keywords:

Arabic
Deep learning
Named entity recognition
NLP
Word embeddings

ABSTRACT

Most of the Arabic Named Entity Recognition (NER) systems depend massively on external resources and handmade feature engineering to achieve state-of-the-art results. To overcome such limitations, we proposed, in this paper, to use deep learning approach to tackle the Arabic NER task. We introduced a neural network architecture based on bidirectional Long Short-Term Memory (LSTM) and Conditional Random Fields (CRF) and experimented with various commonly used hyperparameters to assess their effect on the overall performance of our system. Our model gets two sources of information about words as input: pre-trained word embeddings and character-based representations and eliminated the need for any task-specific knowledge or feature engineering. We obtained state-of-the-art result on the standard ANERcorp corpus with an F1 score of 90.6%.

Corresponding Author:

Ismail El Bazi,
IR2M Laboratory, FST,
Hassan 1st University,
Casablanca Street, Box 577, Settat, 26000, Morocco.
Email: ismailelbazi@gmail.com

1. INTRODUCTION

The Named Entity Recognition (NER) task aims to identify and categorize proper nouns and important nouns in a text into a set of predefined categories of interest such as persons, organizations, locations, etc. [1] NER is a mandatory preprocessing module in several natural language processing (NLP) applications such as syntactic parsing [2], question answering [3] and entity coreference resolution [4]. Achieving the best performance on NER task requires large amounts of external resources such as gazetteers, plenty of hand-crafted feature engineering and extensive data pre-processing. However, developing such task-specific resources and features is costly and needs a lot of time. For morphologically rich languages like Arabic, this task becomes even more challenging due to its unique characteristics. The highly agglutinative nature of Arabic allows for the same word to have different morphological forms which generate a lot of data sparseness. Also, the absence of diacritics in most modern standard Arabic texts creates a lot of ambiguity since many words can share the same surface form without diacritics but have different named entity (NE) tags. Furthermore, unlike most European languages, there is no capitalization in Arabic. Therefore, it is not possible to use capitalization as feature indicator to detect named entities. Finally, there are a very limited number of linguistic resources such as gazetteers and NE annotated corpora, freely available for researchers to build decent Arabic NER systems.

Mainly, the researchers interested in NER for the Arabic language follow three approaches: rule-based [5],[6], machine learning(ML)-based [7], [8] and hybrid approaches [9]-[11]. These three approaches suffer from the same issues since it needs a lot of language-specific knowledge and an extensive feature engineering to obtain useful results. This is even more accentuated by the lack of linguistic resources and the complex morphology of the language.

Recently, the Deep Learning (DL) [12] paradigm has emerged and made impressive achievements in fields such as speech processing [13] and image recognition [14]. For NLP, the application of deep

learning has proven to be very effective yielding state-of-the-art in various common NLP tasks as sequence labeling [15], sentiment analysis [16],[17] and machine translation [18] for the English language. Unlike traditional approaches, DL is an end-to-end model that did not rely on data preprocessing, manual feature engineering or large amounts task-specific resources and can be adapted to various languages and domains. This makes it a very attractive solution for complex and low resource language like Arabic.

Motivated by the success of deep learning in several NLP applications, we introduce an Arabic NER system based on deep neural networks. In the DL literature two neural network architectures are widely used: convolutional neural networks (CNN) [19] and long-short-term memory (LSTM) [20]. Thus the neural network architecture that we introduce on this paper embraces both models. We employ CNN to induce character-level representations of words and we feed it in conjunction with word embeddings to a bidirectional LSTM network (BiLSTM) that perform the training. Finally, we use a conditional random fields (CRF) [21] layer to do the decoding of the input sequence.

Since the careful selection of optimal parameters can often make a huge difference in the performance of neural network architecture, we thoroughly investigated the impact of diverse hyperparameters on the overall performance of the chosen neural architecture and selected the best ones for our final model.

Our main contributions of this paper are as follows:

a. Proposing a deep learning approach to address the Arabic NER task.
b. Evaluating and selecting the optimal hyperparameters for the proposed neural network architecture.
c. Confirming the advantage of integrating character-based representations for morphologically rich languages like Arabic.
d. Achieving state-of-the-art results on the standard ANERCorp corpus without the need of any feature engineering or domain-specific knowledge.

2. PROPOSED APPROACH

In this section, we outline the deep learning approach that we adopted to tackle the NER task for the Arabic language. We propose neural network architecture composed of a BiLSTM layer and a CRF layer. First, we compute the character representation for each word using either CNN or BiLSTM (see Section 2.5 for details), then we concatenate it with the word embeddings before feeding into the BiLSTM layer. This layer is composed of two LSTM networks. The forward LSTM reads the word sequence from the beginning when the backward LSTM reads it in opposite order. Finally, the output vectors of both LSTM networks are concatenated and sent as input to the CRF layer to generate the tags prediction for the input sequence. The architecture of our neural network is illustrated in detail in Figure 1. We briefly describe the layers of our model in the following sections.

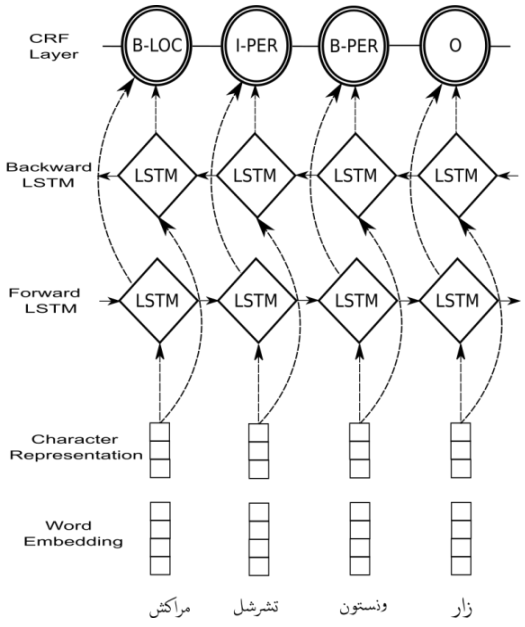

Figure 1. The main architecture of our neural BiLSTM-CRF network

2.1. LSTM

Long-short term memory (LSTM) networks are variants of recurrent neural networks (RNN) specially designed to address some well-known issues related to exploding and vanishing gradient by appending an extra memory-cell. LSTMs are very effective to capture long-distance dependencies. They take as input a sequence of vectors (x_1,x_2,…,x_n) of length n and return an output sequence of vectors (h_1,h_2,…,h_n) called hidden states. The LSTM implementation used is represented by the following formulas at time t:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1}) + b_i \quad (1)$$

$$c_t = (1 - i_t) \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{ci}h_{t-1} + b_c) \quad (2)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \quad (3)$$

$$h_t = o_t \odot \tanh(c_t) \quad (4)$$

where σ denotes the element-wise sigmoid function and Θ the element-wise product. i_t is the input gate vector, c_t the cell state vector and o_t the output gate vector. All W and b are trainable parameters.

2.2. BiLSTM

Despite their capability to capture long-distance dependencies, standard LSTMs are not very effective on sequence tagging tasks like NER. In fact, an LSTM unit can take information only from past context, but for sequence tagging is very useful to retrieve both past and future information. To overcome this constraint we use bidirectional LSTM. The basic idea is that we will use two separate LSTM units. The first one is a forward LSTM that reads the sequence of words and induces a representation of the past context. The second one is a backward LSTM that takes the same sequence but in reverse and induces a representation of the future context. The final representation of a word is the combination of its past and future context representations.

2.3. CRF layer

To predict the final tag sequence for the input sentence, we feed the output of the BiLSTM layer to a classifier. A very simple example of classifier layer is softmax. It is suitable for simple tasks where the output tags are independent. For more complex sequence tagging tasks like NER, where we have strong dependencies between output tags, the independence assumptions are not valid. Actually, in NER with IOB2 format I-LOC cannot follow B-PER. Hence, instead of decoding each tag independently, we jointly decode the tag predictions utilizing a conditional random field component which maximizes the tags probabilities of the whole sentence.

2.4. Word embeddings

Word embeddings are dense low-dimensional real-valued vectors learned over unlabeled data using unsupervised approaches. Each word in an input sentence can be mapped to a pre-trained word embedding. For unseen words, word embedding has a very good generalization since it potentially captures useful semantic and syntactic properties between words. These interesting characteristics, allow it to significantly boost the performance of various NLP tasks [15], [22]. For our neural network architecture, we use pretrained word embeddings as input to efficiently initialize the lookup table of our model.

2.5. Character representations

The use of word embeddings is usually sufficient to get the best performance for the English language. For morphologically rich languages like Arabic, the richness of the morphological forms make the vocabulary sizes larger and the out-of-vocabulary (OOV) rate relatively higher. Hence the needs of another representation of word based on its characters to effectively capture the orthographic and morphological information such as pre- and suffixes of words and encode it into neural representations that can be used by our model. Mainly, there are two ways to learn character representations. We can use convolutional neural networks [15] to encode a character-based representation of a word. Figure 2 shows the CNN architecture used. On the other hand, we can also use bidirectional LSTMs [22] to generate a character-based representation of a word from its characters. Figure 3 describes the BiLSTM architecture.

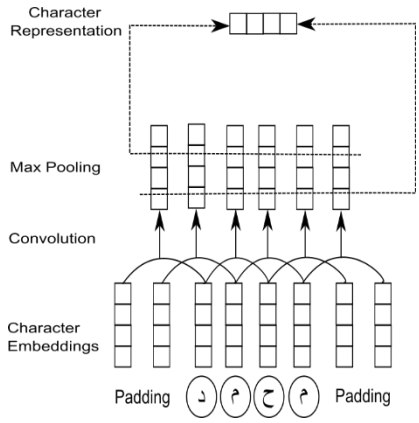

Figure 2. Character-based representation using CNN

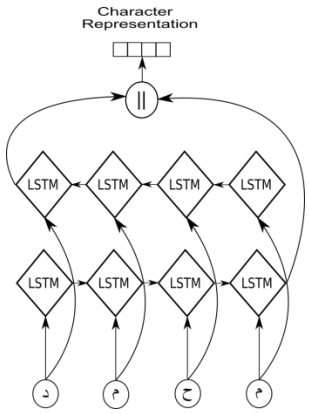

Figure 3. Character-based representation using BiLSTM

3. EXPERIMENTS

This section provides details about the training of our neural network. Since the achievement of state-of-the-art results using neural networks requires the selection and optimization of many hyperparameters, we will also study the impact of the hyperparameters and the parameter initialization on the overall performance of our models. We will precisely evaluate the impact of the following hyperparameters: pre-trained word embeddings, character representation, dropout, and optimizers.

3.1. Network training

Our neural model is implemented using Keras API with the Theano library as a backend [23]. The training is done using the back-propagation algorithm with the Adam optimizer. We use gradient normalization of 1 to deal with "gradient exploding". For all our experiments, we run the training with the mini-batch size of 8 for 50 epochs and apply early stopping of 5 based on the performance on the validation set. The remaining default settings of the hyperparameters are summarized in Table 1.

Table 1. The Default Hyperparameters of the Network

Layer	Hyperparameter	Value
CNN	window size	3
	number of filters	30
LSTM	state size	50
	number of layers	2
Dropout	Dropout type	Naive
	dropout rate	0.5

3.2. Pre-trained word embeddings

We employ pretrained word representations to initialize our lookup table. We learned our own word embeddings using the Arabic Wikipedia dump of December 2016 with a dimension of 50. To assess if the choice of the learning algorithm is relevant, we experiment with 5 models namely, SkipGram [24], CBOW [25], GloVe [26], FastText [27] and Hellinger PCA (H-PCA) [28]. We also assess the impact of the vector size by varying it for the best performing algorithm between 50 and 500.

3.3. Character representations

In this experiment, we check if the use of character representation is helpful and can really have a tangible impact on the performance of the network. Additionally, we compare the CNN and BiLSTM approaches of learning character-based representations and analyze which one to be preferred in regard to performance.

3.4. Dropout

Dropout is a key method to regularize the neural model and mitigate overfitting. In this experiment, we evaluate three setups: No dropout, naive dropout, and variational dropout [29]. The dropout rate is selected from the set {0.25, 0.5, 0.75}.

3.5. Optimizer

The optimizer is an algorithm that helps us to minimize the objective function of the neural network. The choice of an optimizer can influence both the performance and the training time of our model. We experiment with 6 popular optimizers. Namely, Stochastic Gradient Descent (SGD), Adagrad, Adadelta, Adam, Nadam, and RMSProp.

3.6. Data sets

To evaluate the impact of hyperparameters we use the Arabic Wikipedia named entity corpus (AQMAR) [30]. It is a small annotated corpus of 74K token that we choose it for convenience due to the limited computation power that we have to run our experiments. The corpus statistics are depicted in Table 2.

Table 2. AQMAR Corpus Statistics

Dataset	Sentences	Words	Entities
Train	1976	52650	3781
Dev	336	10640	1099
Test	376	10564	974

For the comparison with previously state-of-the-art Arabic NER systems, we use the ANERCorp corpus. It is a publicly available dataset and considered the standard benchmark for the Arabic NER task. The corpus statistics are summarized in Table 3.

The training of neural networks is a very non-deterministic process as it typically depends o the random number generator to initialize the weights of the network [30]. To mitigate the impact of this observed randomness in the evaluation of our neural network, we execute all the experiments 5 times and use the average of F1 scores as the comparison metric.

Table 3. ANERCorp Corpus Statistics

Dataset	Sentences	Words	Entities
Train	4756	120011	12833
Dev	585	14976	1726
Test	546	15019	1605

4. RESULTS AND DISCUSSION

Table 4 shows the impact of various pretrained word embeddings on the Arabic NER task. Despite that we run all the five learning algorithms using the default setting on the same unlabeled data, we can see that FastText has consistently outperformed the other models with an average F1 score of 70.86%. The second best model is SkipGram with a 61.91% in F1 score. In fact, FastText is an extension of SkipGram, but instead of using words directly, it learns word embedding using character n-grams. This simple trick allows it to take the morphology of words into account and helps to deal with rare and out of vocabulary words which is always the case for morphology rich language like Arabic. Hence, our empirical results show that the FastText is more suitable for these types of languages in comparison with other learning algorithms.

Table 4. Results with Different Choices of Word Embeddings

Model	F1					
	Run 1	Run 2	Run 3	Run 4	Run 5	Average
FastText	70,82	70,79	68,84	70,89	72,94	**70,86**
SkipGram	66,04	56,33	59,79	61,53	65,9	61,91
CBOW	55,94	59,44	61,78	51,17	52,38	56,14
Glove	55,74	63,98	64,05	58,25	61,63	60,73
HPCA	36,38	41,18	39,38	38,93	40,98	39,37

In Table 5, we vary the size of the FastText word embeddings to see if it influences the performance of our system. Surprisingly, increasing the vector size did not further enhance the performance even with bigger values as 500, rather it decreases it. So vector dimension 50 was optimal in our case. Actually, while intrinsic tasks like word similarity have usually clear tendency to prefer higher vector dimensionality to effectively capture semantic relationships between words, the extrinsic tasks like NER usually require more

careful tuning to find the optimal dimensionality and tend to favor lower vector size [31]. Therefore, the choice of vector size between 50 and 100 should usually be enough for similar tasks as NER to get the best results.

Table 5. The Performance Comparison when Varying the Size of FastText Word Embeddings

Size	F1					
	Run 1	Run 2	Run 3	Run 4	Run 5	Average
50	70,82	70,79	68,84	70,89	72,94	70,86
100	69,65	69,25	70,39	70,52	69,32	69,82
200	67,45	68,25	67,95	66,89	68,67	67,84
300	69,66	68,87	69,37	66,58	67,89	68,47
400	68,61	68,57	69,61	68,54	67,97	68,66
500	67,57	69,26	69,75	69,95	69,5	69,2

Concerning character representations, Table 6 shows that using it yields to significantly better performance on the Arabic NER task. Precisely, the CNN approach was superior to the BiLSTM one in all the 5 runs of our setup. Thus, we adopt it as the default setting for all upcoming experiments due to its superiority and its higher computational efficiency. Interestingly, recent studies [31] suggest that there is no statistical difference of using character representations when applied to the English NER task. Indeed, for languages like English which did not exhibit morphology richness, the use of character representations is no mandatory to get the best results, but for morphology rich languages like Arabic it is crucial to use it to deal with the complexity and the higher number of rare and out-of-vocabulary words observed.

Table 6. Comparison of not using Character Representations and using CNN or BiLSTM to Induce Character-based Representations

	F1					
	Run 1	Run 2	Run 3	Run 4	Run 5	Average
None	65,9	67,15	65,31	65,68	65,79	65,96
CNN	70,82	70,79	68,84	70,89	72,94	70,86
BiLSTM	68,8	67,92	67,65	68,16	67,66	68,04

In Table 7, we study the impact of dropout. We evaluate three options: naive dropout, variational dropout, and no dropout and select the dropout rates from the set {0.25, 0.5, 0.75}. We observe the best performance with a dropout rate of 0.25. The naive dropout produces the best results with an average F1 of 71.08%. The variational dropout yields a competitive result of 70.52%.

Table 7. Results with and without Dropout using different Rates

Dropout	Dropout rate	F1					
		Run 1	Run 2	Run 3	Run 4	Run 5	Average
None	n/a	70,31	69,83	71,49	71,08	69,58	70,46
Naive	0.25	70,53	71,5	71,55	72,61	69,19	**71,08**
Naive	0.5	70,82	70,79	68,84	70,88	72,94	70,85
Naive	0.75	69,19	71,62	69,86	66,6	62,21	67,9
Variational	0.25	70,12	71,65	70,15	70,19	70,48	70,52
Variational	0.5	69,07	67,66	65,75	60,17	58,46	64,22
Variational	0.75	55,5	58,27	57,84	56,83	55,17	56,72

Table 8 depicts the results for the different optimizers applied to our neural network. We used the settings recommended by the authors of each optimizer. Adam shows the best performance, yielding the highest score for 70.86%. Nadam which is a variant of Adam (Adam with Nesterov momentum) achieves a very competitive performance of 70.57%. Remarkably, SGD produces the worst score of 33.82%. Actually, SGD is very sensitive to the choice of the learning rate and since we did not fine tune it manually, it failed to converge to a minimum. Furthermore, applying early stopping did not help as SGD needs usually more epochs to find the global minimum of the objective function.

In order to compare our neural network with the best performing Arabic NER systems, we apply our BiLSTM-CRF model to the standard ANERcorp dataset using the best hyperparameters evaluated and selected during our previous experiments. Since the performance of both naive and variational dropout was

quite close and it is also the case for the Adam and Nadam optimizers, we decided to experiment with different settings of these hyperparameters in combination with the other best hyperparameters to be sure that we have the optimal setup for our model. Table 9 summarizes the results. The best performance of our BiLSTM-CRF model is achieved using Nadam as an optimizer and variational dropout with an average F1 score of 90.60%.

Table 8. Performance Comparison for Various Optimizers

Optimizer	F1					
	Run 1	Run 2	Run 3	Run 4	Run 5	Average
Adam	70,82	70,79	68,84	70,89	72,94	70,86
Nadam	71,1	71,57	71,28	67,93	70,96	70,57
Rmsprop	69,19	68,08	69,53	68,9	69,42	69,02
Adadelta	55,97	58,91	65,2	58,82	53,51	58,48
Adagrad	57,34	57,77	52,54	53,62	60,74	56,4
SGD	36,35	24,24	37,27	37,47	33,78	33,82

Table 9. Results on the ANERcorp Dataset using the best Hyperparameters

Settings	F1					
	Run 1	Run 2	Run 3	Run 4	Run 5	Average
Nadam + Variational Dropout	90,56	90,38	90,36	90,78	90,91	**90,60**
Adam + Variational Dropout	89,51	90,22	89,97	90,07	90,12	89,98
Adam + Naive Dropout	88,05	88,75	88,27	88,11	87,71	88,18
Nadam +Naive Dropout	89,62	88,25	89,16	88,99	89,38	89,08

In Table 10, we present the results of our system in comparison with three previous top performing systems for Arabic NER. Our system achieves significant improvements over [7] and [9] on the standard ANERcorp dataset with an F1 score of 90.6%. We obtain state-of-the-art result in comparison with [10]. Our model is slightly lower with 0.06%.

In fact, the system introduced by Shaalan and Oudah [10] is a hybrid model that combines machine learning-based component and rule-based component. It relies heavily on the task-specific and language dependent knowledge provided by the rule-based component and uses a lot of handcrafted engineered features including morphological features, POS tags, capitalization features and gazetteers to achieve state-of-the-art performance. On the other hand, Our BiLSTM-CRF model has the advantage of being a true end-to-end system that does not require any feature engineering, data pre-processing or external resources and therefore can be easily extended to other domains with minimal tweaking.

Table 10. Comparison with Previous Top Performance Arabic NER Systems on ANERcorp Dataset

Model	F1
CRF-based system [7]	75.66
Abdallah et al. [9]	88.33
Shaalan and Oudah [10]	**90.66**
Our system	***90.60***

5. CONCLUSION

This paper proposes neural network architecture for Arabic NER based on bidirectional LSTMs. We evaluated different commonly used hyperparameters for our BiLSTM-CRF architecture to assess their impact on the overall performance. Our best model obtains state-of-the-art results with an F1 of 90.6% using FastText pre-trained word embeddings, CNN Character Representations, a variational dropout and Nadam optimizer.

In comparison with previously state-of-the-art Arabic NER systems, our neural model is truly end-to-end and does not depend on any data preprocessing, external task-specific resources or handcrafted feature engineering. It is also very flexible by allowing the effortless addition of another type of named entities as numeral and temporal NEs, facilities and geo-political NEs, etc.

Our ongoing work is to explore multi-task learning approaches and see if it can further improve our model. Also, we hope that we can extend this work to other domains like noisy user-generated text which is more challenging.

REFERENCES

[1] I. El bazi and N. Laachfoubi, "Exploring the Effects of Stemming on Arabic Named Entity Recognition," *International Journal of Artificial Intelligence and Applications*, vol/issue: 7(1), pp. 33–43, 2016.

[2] A. Shahrour, *et al.*, "Camelparser: A system for Arabic syntactic analysis and morphological disambiguation," *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: System Demonstrations*, pp. 228–232, 2016.

[3] L. Abouenour, *et al.*, "IDRAAQ: New Arabic question answering system based on query expansion and passage retrieval," 2012.

[4] M. Beseiso and A. Al-Alwani, "A Coreference Resolution Approach using Morphological Features in Arabic," *International Journal of Advanced Computer Science and Applications*, vol/issue: 7(10), pp. 107–113, 2016.

[5] S. Mesfar, "Named Entity Recognition for Arabic Using Syntactic Grammars," *Proceedings of the 12th International Conference on Applications of Natural Language to Information Systems*, pp. 305–316, 2007.

[6] W. Zaghouani, "RENAR: A Rule-Based Arabic Named Entity Recognition System," vol/issue: 11(1), pp. 2:1–2:13, 2012.

[7] Y. Benajiba and P. Rosso, "Arabic named entity recognition using conditional random fields," *Proc. of Workshop on HLT & NLP within the Arabic World, LREC*, vol. 8, pp. 143–153, 2008.

[8] I. El bazi and N. Laachfoubi, "Arabic Named Entity Recognition Using Topic Modeling," *International Journal of Intelligent Engineering and Systems*, vol/issue: 11(1), pp. 229–238, 2018.

[9] S. Abdallah, *et al.*, "Integrating Rule-Based System with Classification for Arabic Named Entity Recognition," *Computational Linguistics and Intelligent Text Processing*, pp. 311–322, 2012.

[10] K. Shaalan and M. Oudah, "A hybrid approach to Arabic named entity recognition," *Journal of Information Science*, vol/issue: 40(1), pp. 67–87, 2014.

[11] M. Oudah and K. Shaalan, "NERA 2.0: Improving coverage and performance of rule-based named entity recognition for Arabic," *Natural Language Engineering*, vol/issue: 23(3), pp. 441–472, 2017.

[12] C. Mishra and D. Gupta, "Deep Machine Learning and Neural Networks: An Overview," *IAES International Journal of Artificial Intelligence*, vol/issue: 6(2), pp. 66–73, 2017.

[13] A. van den Oord, *et al.*, "Wavenet: A generative model for raw audio," arXiv preprint arXiv: 1609.03499, 2016.

[14] J. Hu, *et al.*, "Squeeze-and-Excitation Networks," arXiv preprint arXiv: 1709.01507, 2017.

[15] X. Ma and E. Hovy, "End-to-end sequence labeling via bi-directional lstm-cnns-crf," arXiv preprint arXiv: 1603.01354, 2016.

[16] C. dos Santos and M. Gatti, "Deep convolutional neural networks for sentiment analysis of short texts," *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pp. 69–78, 2014.

[17] S. Shah, *et al.*, "Sentimental Analysis of Twitter Data Using Classifier Algorithms," *International Journal of Electrical and Computer Engineering (IJECE)*, vol/issue: 6(1), pp. 357–366, 2016.

[18] A. Vaswani, *et al.*, "Attention is All You Need," 2017.

[19] Y. LeCun, *et al.*, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Comput.*, vol/issue: 1(4), pp. 541–551, 1989.

[20] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol/issue: 9(8), pp. 1735–1780, 1997.

[21] J. D. Lafferty, *et al.*, "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data," *Proceedings of the Eighteenth International Conference on Machine Learning*, pp. 282–289, 2001.

[22] G. Lample, *et al.*, "Neural architectures for named entity recognition," arXiv preprint arXiv: 1603.01360, 2016.

[23] https://github.com/UKPLab/emnlp2017-bilstm-cnn-crf

[24] T. Mikolov, *et al.*, "Efficient estimation of word representations in vector space," arXiv preprint arXiv: 1301.3781, 2013.

[25] T. Mikolov, *et al.*, "Distributed Representations of Words and Phrases and their Compositionality," *Advances in Neural Information Processing Systems 26, Lake Tahoe, Nevada: Curran Associates, Inc*., pp. 3111–3119, 2013.

[26] J. Pennington, *et al.*, "GloVe: Global Vectors for Word Representation," *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014.

[27] P. Bojanowski, *et al.*, "Enriching Word Vectors with Subword Information," arXiv preprint arXiv: 1607.04606, 2016.

[28] R. Lebret and R. Collobert, "Word Embeddings through Hellinger PCA," *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 482–490, 2014.

[29] Y. Gal and Z. Ghahramani, "A Theoretically Grounded Application of Dropout in Recurrent Neural Networks," *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pp. 1027–1035, 2016.

[30] B. Mohit, *et al.*, "Recall-oriented Learning of Named Entities in Arabic Wikipedia," *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 162–173, 2012.

[31] N. Reimers and I. Gurevych, "Reporting Score Distributions Makes a Difference: Performance Study of LSTM-networks for Sequence Tagging," *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 338–348, 2017.