APPLYING RECURRENT NEURAL NETWORK TO ARABIC NAMED ENTITY RECOGNITION

Naassih Gopee

Advisors:

Kemal Oflazer, Houda Bouamor, Bhiksha Raj & William Cohen

Acknowledgement

While writing this thesis, I have been through hard times whether from personal to academic levels. However, I have had generous support from a large number of people who must receive my deepest gratitude.

First and foremost, I would like to extend my heartfelt gratitude to all my advisors.

Firstly, Professor Kemal Oflazer, who agreed to advise me for my thesis and letting me use his experience in doing this research. Without his patience, motivation, and his insight in the field, this thesis would not have been possible.

Professor Houda Bouamor, without her ongoing guidance and support, it would have been impossible for me to complete this thesis on time. Her positive attitude and humility has profoundly motivated me to stay on board and keep the momentum till the end of this research activity.

Professor Bhiksha Raj, for his patience and readiness to help. His ability to take complex machine learning concepts and simplifying it to help me clear my misconceptions allowed me to have a better appreciation for the field of machine learning.

Professor William Cohen, for introducing me to the field of machine learning and agreeing to advise me for this thesis as part of my machine learning minor.

I would also like to thank Professor Majd Sakr who has been an incredible mentor from day one at CMU and who always provided me with prompt support throughout my CMU journey.

My parents and sisters have provided me with tremendous support throughout my life and this has continued during the course of my thesis. I wish to thank them for such unwavering love and guidance.

Finally, I would like to express my gratitude to all my friends, especially Dilsher Ahmed and John Naguib. I thank them for such immense support and motivation throughout.

Abstract

Named Entity Recognition (NER) (also known as entity identification) is a subtask of information extraction that seeks to locate and classify elements in text into predefined categories such as the names of persons, organizations, locations, etc. NER plays an important role in many NLP problems, such as Machine Translation as it helps improve the performance of algorithms that solve these problems.

In this work, we plan to tackle Arabic NE recognition and classification with an approach using Long Short Term Memory (LSTM) neural networks. We use LSTMs' ability to memorize long term dependencies to train a model for Arabic NE recognition, on a training dataset. The model is then used to predict the NEs for a sample of Arabic sentences in our test set. We tested our system on a pilot dataset. In its current version, it achieves a word level accuracy of 85%. More recently we trained our model on the more standard ACE 2007 dataset and achieved an F1 score of 57.54 for detecting boundaries and 53.31 for categorizing the named entity. However, adding part-of-speech as a feature reduced our performance. Overall, LSTM seems to be a promising model for Arabic NER. We plan to compare it with different existing baselines trained on other dataset. We also plan to identify an optimal feature set in order to study its impact on the accuracy of our predictor.

Table of Contents

1 Introduction

Name Entity Recognition (NER) is the problem of identifying sequences of words that refer to named entities (NEs) such as persons, locations, or organizations. NER plays an important role in many natural language processing applications such as information extraction, machine translation, and question answering (Benajiba et al., 2008). Evidence on how impactful NER can be to information extraction and machine translation can be seen in many research works (Babych and Hartley, 2003; Ferrandez et al., 2004; Toda and Kataoka, 2005).

Similar to English, being able to effectively identify NE for Arabic is crucial as it is one of the most important factors for many natural language processing applications. NER has been rigorously studied for English and discussed for many languages, including Arabic. However, much work still remains to be done for Arabic.

Arabic is a Semitic language and this gives rise to morphological and orthographic challenges such as the facts that proper names are often common language words, capitalization is absent and conjunctions, prepositions, possessive pronouns, and determiners are attached to words as prefixes or suffixes (Abdul-Hamid et al., 2010). Therefore, the key challenges are to:

1. Identify a set of features that works well for Arabic NER.
2. Devise new ways for Arabic text pre-processing (dealing with morphology, etc.).
3. Determine a good approach for NE identification and categorization.

In this work, we tackled the Arabic NE recognition and classification task with a different machine learning technique. Following Hammerton, 2003, we used Long Short Term Memory (LSTM) neural networks (Hochreiter and Schmidhuber, 1997). In his work, LSTM was used to detect English and German NEs.

LSTMs are a form of Recurrent Neural Networks (RNNs) that were designed to solve the problem of rapid-decay of back propagated error in neural networks – error being back propagated decreases exponentially. With the ability to memorize relevant events over time, LSTM neural networks were shown to work well when prediction depends on long term dependencies. NER is one of many tasks in which modelling long term dependencies helps in developing accurate systems.

As a test bed, we used the Automatic Content Extraction (ACE) 2007 NE dataset for Arabic. ACE has facilitated evaluation for Arabic by creating standardized test sets and evaluation metrics and hence the ACE 2007 test set will be used to test our framework against other methods performing Arabic NER.

2 Machine Learning Background

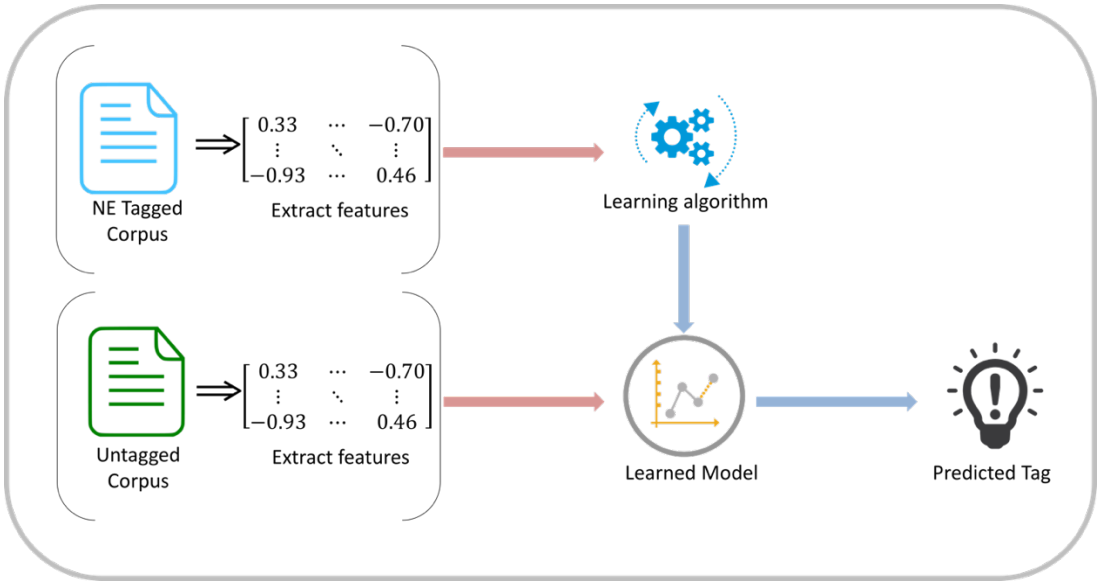

Figure 1: Overview of machine learning classification algorithm

To better understand research conducted for named entity recognition, we must first understand the current techniques that are being used to solve this task. Most named entity recognition makes use of supervised learning techniques where labelled data – the NE Tagged corpus in Figure 1 – are fed to a learning algorithm. Each word in the sentences in our NE tagged corpus is assigned a label depending on whether it is a named entity – in which case the label explains what kind of entity e.g. Location, person's name etc. – or not a named entity. Each word in the corpus is labelled by a human annotator – usually a linguist. More recently however, many projects have turned to crowdsourcing, which seems to be a promising solution to obtain high-quality aggregate human judgments for supervised and semi-supervised machine learning approaches to NER.

After getting the corpus, it is processed by mapping each word to a word embedding and extracting all the respective NE for each sentence. This in turn is fed to the machine learning classification algorithm that tries to learn from this feature set and find relations between the features. A myriad of classifiers have been used to perform machine-learned NER. The most commonly used classification algorithms for this task are Support Vector Machine (SVM) and Conditional Random Field (CRF). Numerous classification algorithms have been studied for NER; however, it has been shown that SVM and CRF achieve state-of-the-art for such tasks with CRF usually outperforming SVM. Additionally state-of-the-art NER systems for English produce near-human performance[1]. Due to this track record, researchers have applied such techniques for Arabic NER. The features encoding the tags relationships are captured by these algorithms and stored in a learned model. When a new instance is provided to the model, it uses those previously learned relationships from the model to predict the NE tags.

[1] MUC-7 Proceeding: Evaluation of IE Technology: Overview of Results

3 Literature Review

3.1 Named Entity Recognition: English and Arabic

Named Entity Recognition was first introduced in the 1990s, specifically at the Message Understanding Conferences, as an information extraction task and was deemed important by the research community. The majority of NER research has been devoted to English because of its dominance as an international language. This has limited the diversity of text genres and domain factors from other languages that are usually considered when developing NER for these fields (Shalaan, 2014) – especially for Arabic. Moreover, there are other linguistic issues and challenges when dealing with Arabic as it is a highly inflected language, with a rich morphology and complex syntax (Al-Sughaiyer and Al-Kharashi 2004; Ryding 2005). However, due to the massive growth of Arabic data, there is an increasing need for accurate and robust processing tools (Abdul-Mageed, Diab, and Korayem 2011) and NER, being a significant building block for NLP, is crucial to advance Arabic NLP.

All Arabic NER systems that have been developed use primarily two approaches: the rule-based (linguistic-based) approach (Shaalan and Raza 2009); and the machine learning (ML)-based approach, notably ANERsys 2.0 (Benajiba, Rosso, and Benedí Ruiz 2007). Rule-based NER systems rely on handcrafted local grammatical rules written by linguists – which is labor intensive and requires highly skilled labor. Grammar rules make use of gazetteers and lexical triggers in the context in which NEs appear. ML-based systems on the other hand utilize learning algorithms that require large tagged data sets for training and testing (Hewavitharana and Vogel 2011). The dataset for ML-based systems also has to be manually tagged. However, recently this tagging task is being crowd-sourced thereby reducing the cost of labor. One major advantage of using ML-based techniques is that that they are easily adaptable and determine features to predict NEs on their own. Recently, the two approaches have been merged in a hybrid system. This has resulted in a significant improvement by exploiting the rule-based decisions of NEs as features used by the ML classifier (Abdallah, Shaalan, and Shoaib 2012; Oudah and Shaalan 2012). In most Arabic NER literature, the ML-based technique of choice was one from an ensemble of the following: Support Vector Machines (SVM), Conditional Random Fields (CRF), Maximum Entropy, Hidden Markov models, and Decision Trees (Benajiba et al.,2009; Benajiba et al.,2008; Shalaan, 2012). Together with applying the ML-based algorithm, various feature sets have also been explored.

Benajiba, Rosso, and Benedí Ruiz (2007) have developed an Arabic Maximum Entropy-based NER system called ANERsys 1.0. They built their own linguistic resources, ANERcorp and ANERgazet[2] to evaluate their system which has an F1 score of 55.23%. The main issue with the system was boundary detection – the task of determining where an NE begins and ends. The Part-of-Speech (POS) feature was then added to improve the boundary detection which improved the F1 score of the overall system to 65.91%.

[2] ANERcorp and ANERgazet, see http://www1.ccls.columbia.edu/~ybenajiba/.

Benajiba and Rosso (2008) changed the ANERSys 1.0 and applied CRF instead of ME and named it ANERsys 2.0. The set of features used was language-independent and non-Arabic specific features were used: including POS tags, based-phrase chunk (BPC), gazetteers, and nationality. The system achieved an F1 score of 79.21.

Benajiba, Diab, and Rosso (2008a) explored the morphological, lexical, contextual, gazetteer of the ACE 2003, 2004 and 2005 data sets and applied an SVM classifier. The impact of the different features was independently measured for different datasets. The overall system achieves an F1 score of 82.71% for ACE 2003, 76.43% for ACE 2004, and 81.47% for ACE 2005.

In summary, a lot work has been done in trying to achieve state-of-the-art Arabic NER. Despite all the systems built for Arabic NER, to the best of our knowledge, no research has explored the possibility of applying neural networks for Arabic NER. Neural networks have been shown to boost gains on many other NLP tasks. Huang, Xu and Yu (2015), applied a Bi-LSTM (Long Short Term Memory) for CoNLL NER task. They further coupled the LSTM with a CRF layer, boosting the F1 score to 90.10 for the CoNLL English NER task. Therefore, following Hammerton (2003) – LSTM was used to detect English and German NEs – and Huang *et al.* (2015), we will use LSTM neural networks.

3.2 Word Embedding Generation: Word2vec

Word2Vec is a language modelling tool released by Mikolov *et al.* back in 2013. The tool converts words into vectors by computing word co-occurrence statistics. In doing so, the tools try to capture word semantics by learning from all possible contexts a particular word appears in. At its core, Word2vec is a two-layer neural network that processes text and takes as input one-hot encodings. Its input is a text corpus and its output is a set of vectors. These vectors are actually feature vectors that try to capture the meaning of words in that corpus.

3.3 MADAMIRA

MADA (Morphological Analysis and Disambiguation for Arabic) (Habash and Rambow, 2005; Habash et al., 2009; Habash et al., 2013) is the state-of-the-art manually-built morphological analysis system of the Arabic language. Along with word segmentation, MADA is an excellent word-in-context analyzer, and therefore provides accurate segmentation of a word in its context in a sentence. MADA has a high accuracy of usually over 94%.

AMIRA (Diab et al., 2009) is a suite of tools for the processing of Modern Standard Arabic text. It processes raw Arabic text and produces segmented output labeled with part of speech tag (POS) information and also chunk level information. AMIRA allows a user to choose different tokenization schemes. For part of speech tagging, the user can specify different levels of granularity for the POS tag set such as number, gender and person. It accepts Arabic script input as well as the Buckwalter transliteration input encoding formats (Buckwalter, 2002). The output can be produced in the user's choice of encoding, by default, it will produce the output in the same encoding as the input data.

Similar to MADA, MADAMIRA is also a tool for morphological analysis and disambiguation of Arabic. However, MADAMIRA combines some of the best aspects of two commonly used systems for Arabic processing, MADA and AMIRA. MADAMIRA has a better system with a more streamlined Java implementation that is more robust, portable, extensible, and is faster than its ancestors by more than an order of magnitude (Pasha et al, 2014). Moreover, MADAMIRA achieves an accuracy of 96% for POS tagging.

4 Methodology

Our Arabic named entity recognition system has been developed using machine learning based techniques. The mechanics of how ML-based algorithms work is described in the background section (see Section 2).

4.1 ML-based Technique: Neural Networks

Due to the nature of the algorithm being highly problem-oriented, choosing the appropriate technique to solve our task at hand is very important. Research has shown that ML-based techniques such as CRF and SVM can achieve state-of-the-art for English NER[1]. Given the good performance of CRF and SVM on English NER, researchers applied these techniques for Arabic NER but did not quite achieve the same result as for English (Benajiba et al.,2008, Benajiba et al.,2008a). One for the main reasons for this is because Arabic is a morphologically rich language and it has its challenges. Therefore, we concluded that there could potentially be other machine learning technique would be better suited to the task at hand.

Recently, there has been a boom in applying Artificial Neural Networks as a classification technique. Neural networks have been shown to provide enormous gain in performance on problems that were previously thought to have saturated. Many research activities conducted in neural classification have established that neural networks are a promising alternative to various conventional classification methods (Zhang, 2000).

ANN are an information processing paradigm that was inspired by the architecture of the human brain. The human brain consists of a network of neurons where information is stored in the strength of connections between these neurons. Using this analogy, similarly in an ANN (illustrated in Figure 2), we have a network of units where information is stored in weights of the connections between different units. Given input-output training pairs, the ANN learns the weights of the connections. This is done using an algorithm called back-propagation – that is used to adjust the weights. Back-propagation iterates over the training data several times (epochs) updating the weights each time, until the network's performance saturates. At each layer – input layer, hidden layer and output layer – computation is done according to the formula in Figure 2 where W_i is the weight being optimized and f_i is the activation function – typically a sigmoid function.

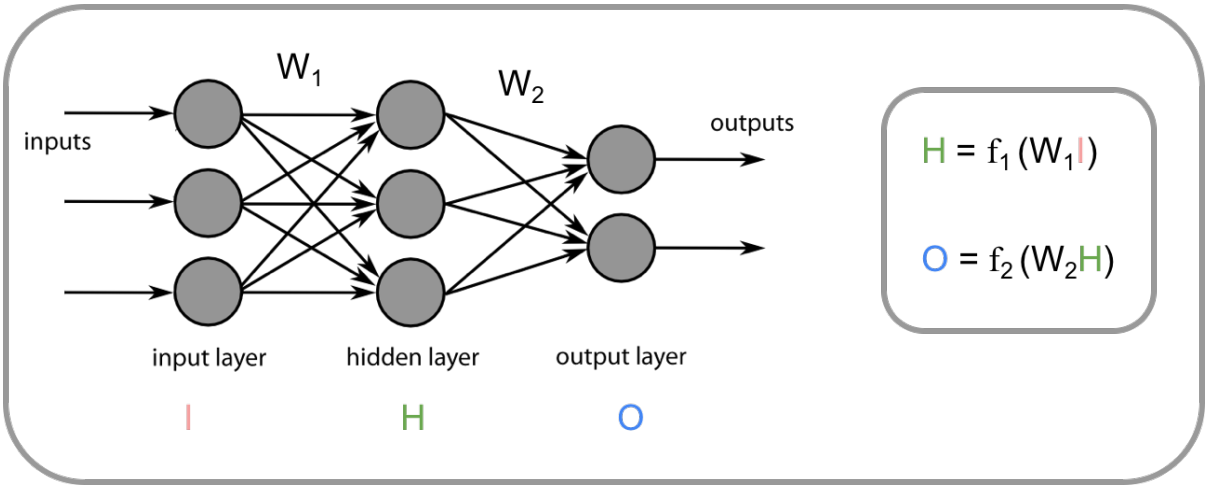

Figure 2: ANN with 2 layers

However, there are some issues when dealing with traditional Neural Networks particularly that they are unable to deal with time-series problems. Time-series problems are those problems where the output at any time depends on the past inputs (and possibly past outputs). To solve this problem, researchers came up with the idea of a *Recurrent Neural Network* (RNN). RNN (illustrated in Figure 3) is a class of artificial neural network where there is at least one feed-back connection. This allows the activations to flow in a loop. This feedback connection allows the network to do temporal processing and learn based on sequences - for instance a sentence.

RNN suffers from two widely known issues when properly training: the vanishing and the exploding gradient problems detailed in Bengio et al. (1994). These problems detail how over time the gradient being calculated in the network either becomes zero (vanishes) or becomes infinity (explodes). This prevents traditional RNN from capturing long term information. In 1997, Hochreiter and Schmidhuber proposed LSTM as a solution to the vanishing and exploding gradient problems. LSTMs are a form of Recurrent Neural Networks that store *long-term* memory through internal "memory" units. With the ability to memorize relevant events over time, LSTM neural networks were shown to work well when prediction depends on long term dependencies. NER is a problem that needs these long term dependencies in order to capture context in a sentence. Hence we concluded that LSTM would be a very appropriate neural network to be used for Arabic NER.

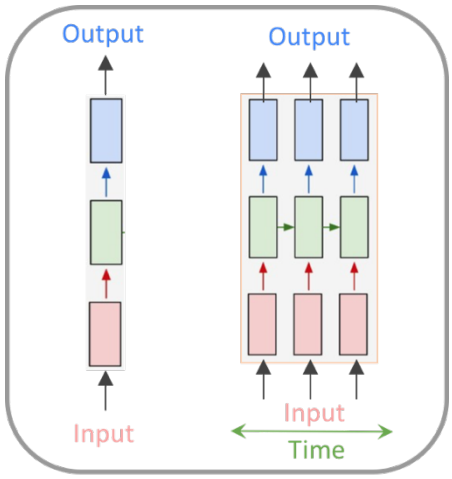

Figure 3: ANN on the left vs. RNN on the right

Simplified diagram where red box depicts input layer, green box depicts the hidden layer and blue box depicts the output layer.

4.2 Labeled Data

We use the Automatic Content Extraction (ACE) 2007 Arabic dataset by the Linguistic Data Consortium (LDC) that was annotated for named entities. The Arabic data is composed of newswire (60%) and weblogs (40%). Out of the total corpus — merging newswire and weblogs — a total of 2779 sentences were extracted. The class distribution

of the dataset is depicted in Figure 4. It was to be expected that the data was skewed towards non-NEs (NNE) with 28% of the corpus being NEs. The majority of the NEs are either a person's name (PER) and organization (ORG) or a geo-political entity (GPE) with very few being facilities, weapons, vehicles and locations.

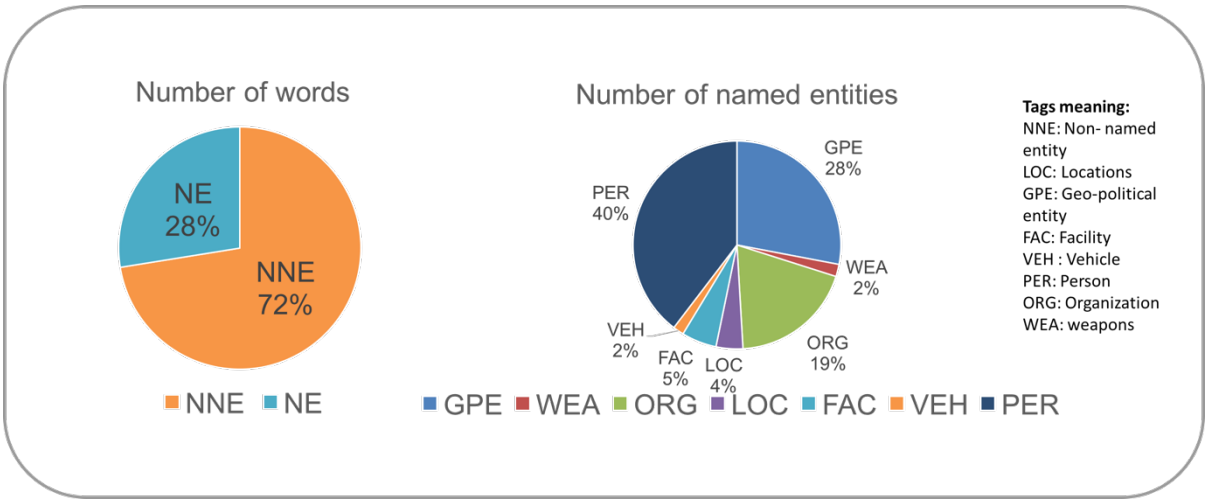

Figure 4: Named entity distribution for ACE 2007 dataset

4.3 Preprocessing

Before being able to feed our training data to an LSTM neural network, it has to go through a step of preprocessing so that that we can convert our raw text corpus into elements that can be processed by the neural network. Since our original data is set according to the LDC standard, we first needed to extract the data we needed from this raw data. For some of these steps, third party tools were used to convert the data. Those tools are described in the literature review (See Section 3).

Step 1: Tokenization

While extracting the data from the ACE dataset, we have to perform tokenization which is the process of segmenting text into smaller elements called tokens. In the context of NER, these elements are words or punctuation. However, because Arabic is a morphologically rich language, some research goes a step further and tokenizes the Arabic text into base phrase chunks — this has not yet been implemented in our current system but could be a possible addition in the future. In our context, most tokens are separated by spaces but there are some special cases that need to be considered. Punctuation like a full-stop or exclamation marks are not separated by a space and this had to be taken care of by our tokenization algorithm.

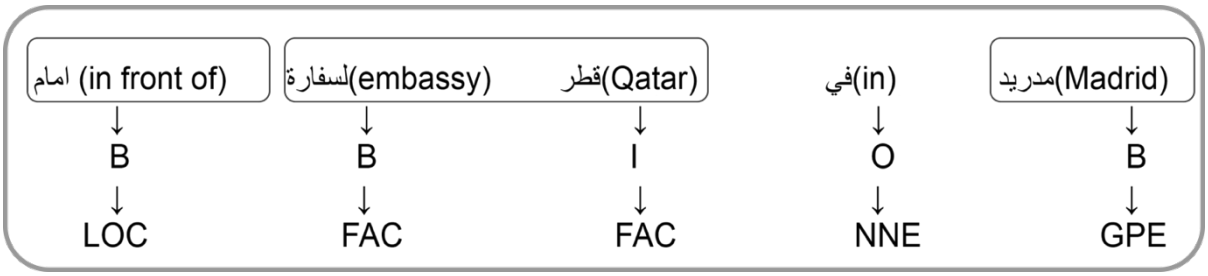

Figure 5: Example illustrating boundary detection and type recognition

Moreover, there are more NER specific problems that had to be factored in when performing tokenization. NER is actually a two-part problem: NE boundary detection and NE type recognition. Because some NE are composed of more than one word, we first have to identify the boundaries of a named entity — i.e. where a NE starts and ends. This is illustrated in the example (See Figure 5) where *'Qatar Embassy'* is actually one NE. The *'B'* marks the beginning of a named entity, the *'I'* marks words associated with a named entity and *'O'* marks words which are not a named entities. Therefore, when extracting the words from the raw corpus, we have to make sure that each word has its proper boundary tag.

Step 2: Embedding Generation

There is a major challenge when dealing with neural networks. Neural networks do not understand what a word means as they require numbers as inputs. Therefore, we need a numerical representation for words in a sentence. To do this, we make use of a tool called Word2vec (See Section 3.2). Word2Vec is a tool that converts words from a corpus to vectors that capture the semantic of the words. In order to get word embeddings, we needed a big corpus of Arabic text to ensure the Word2vec algorithm would capture of meaning of words in a vast number of contexts. Therefore, we used the Arabic Gigaword corpus[3] from the LDC which is a collection of Arabic newswire articles from various sources with 1,591,983 K-words (number of space-separated tokens in the text). The overall flow of this process is described in Figure 6.

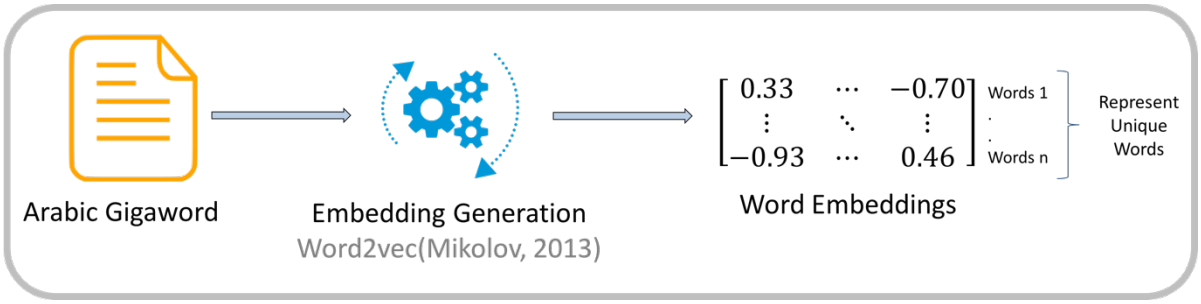

Figure 6: Generation of Embeddings from the Arabic Gigaword corpus

[3] For more on the Arabic Gigaword Corpus, see: https://catalog.ldc.upenn.edu/LDC2006T02

Step 3: Adding Part-of-Speech Tags

When processing text through a machine learning based technique that learns features, it is usually a good practice to add additional features to our dataset to improve performance. Parts of Speech (POS) tags are the grammatical characteristics of a word in a sentence marking words as nouns, verbs, adjectives etc. (See Figure 7). This feature is important for NER as more than 95% of NEs are nouns. Therefore, knowing the POS tag of a word can help us determine whether that word is a NE. Moreover, some words have different POS in different context. POS tags can help in disambiguating these cases.

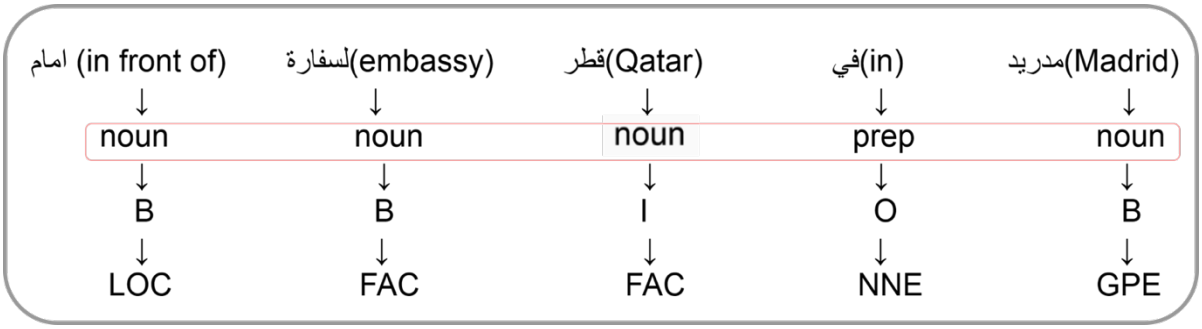

Figure 7: Example illustrating POS tags for a sentence

Our ACE 2007 Arabic corpus does not come with POS tags. Therefore, we used MADAMIRA (See section 3.3) to tag our corpus. MADAMIRA is known to have an accuracy of 96% for POS tagging. The process for POS tagging is illustrated in Figure 8.

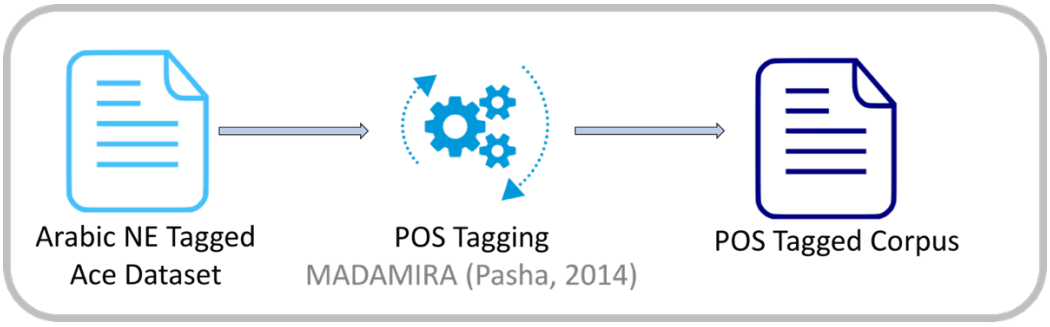

Figure 8: Addition of POS to ACE Corpus

4.4 Training POS Embeddings

A major challenge arises when adding extra features to a neural network. As previously described, an input to a neural network can only be a numeric format, hence the need for word embeddings. In order to add the POS feature to our neural network, we also need to find a numeric encoding for our POS tags. The first and simplest solution that can be used is to turn the POS into a categorical format — i.e. a binarized form. However, MADAMIRA gives us very granular POS tags. For example, our corpus contains 56 possible POS tags after being processed. Therefore, adding the POS in a binary format would make the input to the neural network sparse.

A better idea is to train an LSTM on our current POS tag corpus. The output of an LSTM is actually an embedding that is supposed to encode characteristics of what defines a POS

in our input. Our POS embedding generation process is illustrated in Figure 9. Each word in our POS tagged corpus is mapped to its appropriate word embeddings that were generated using Word2vec. Then we get input-output training pairs where the input is the word embeddings for a sentence and the output is the gold-standard POS tag. This is then passed through an LSTM and trained for 20 epochs. After training, the predicted POS tags form embeddings at the output layer which are saved for each sentence.

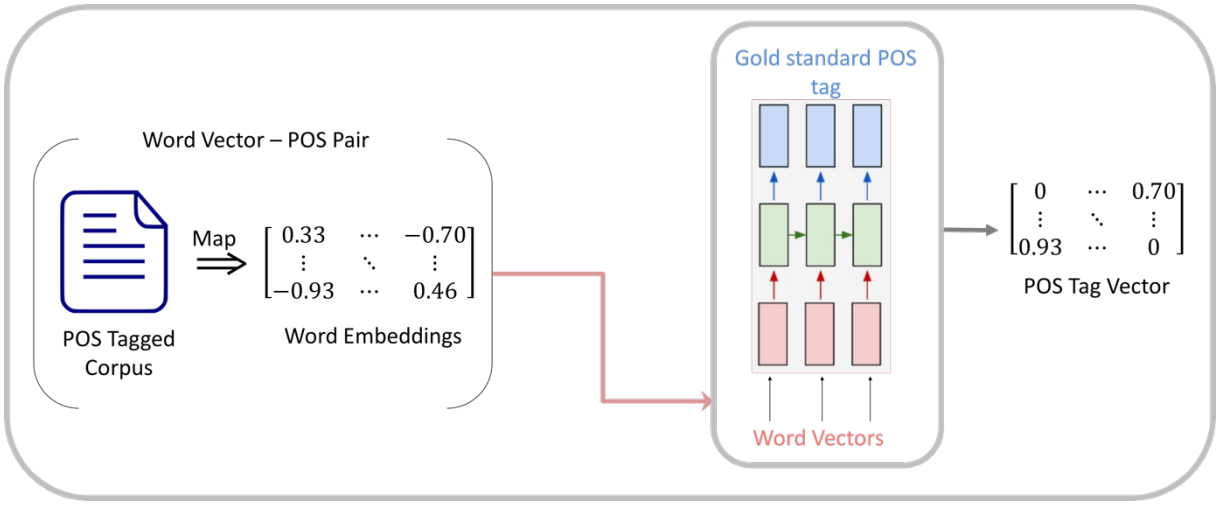

Figure 9: POS embedding generation

4.5 Training LSTM for NER

After preprocessing our corpus and generating the embeddings, we can now move on to the classifier (LSTM neural networks) for which the overall process is explained in Section 2. Figure 10 illustrates a typical way of training an LSTM for NE type recognition. The initial input is the word embeddings representing words in a sentence and the output is the golden NE tag. Back-propagation is then used to learn the weights at the hidden layers.

However, as depicted in Figure 5, NER is a two-stage problem: NE boundary detection and NE type recognition. Conventional LSTMs cannot perform both parts combined. To solve this, we use a two-state recurrent neural network as illustrated in Figure 10. We first train an LSTM to perform boundary detection. In order to do so, the B-I-O tags are stripped off the golden NE tags and we train the LSTM by feeding in the word embeddings of a respective sentence and the corresponding B-I-O tag. Similarly, we then train another LSTM but this time to predict the NE type. The two LSTM's outputs are then synced making it possible to predict both NE boundaries and NE types.

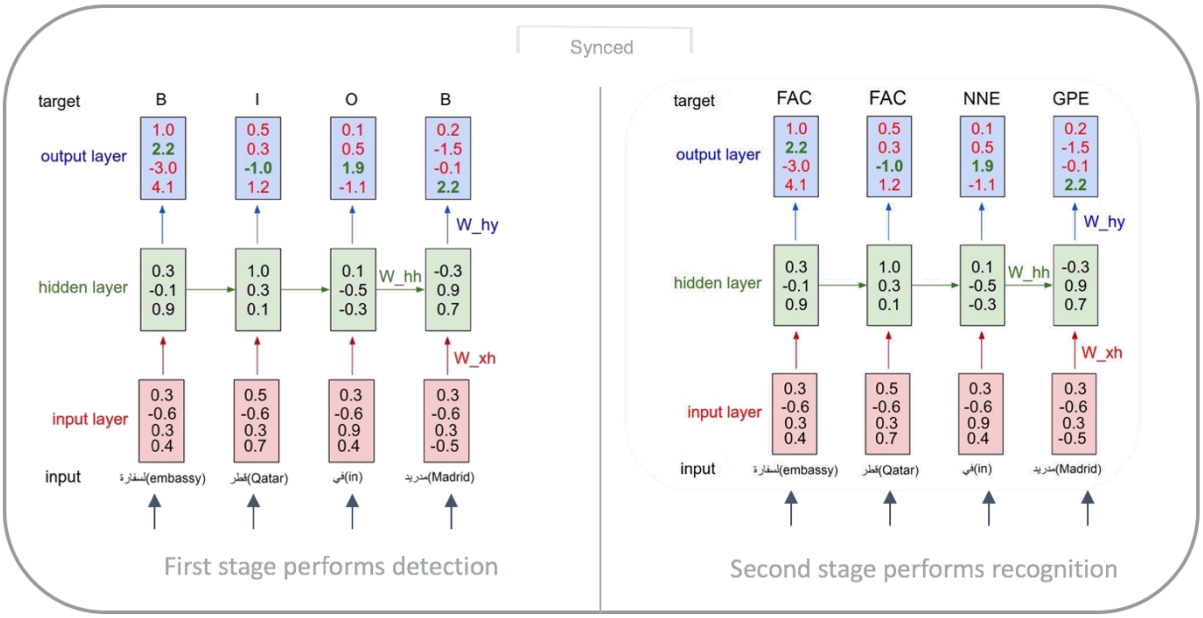

Figure 10: Two-stage RNN for NE boundary detection and NE type recognition

The overall process for training without POS is depicted in Figure 11. Then, each word in the preprocessed training dataset is mapped to its corresponding word embeddings. Then, every sentence is fed to the two-staged RNN and a model is trained on the gold B-I-O and NE type tags. The output models are then saved for later prediction.

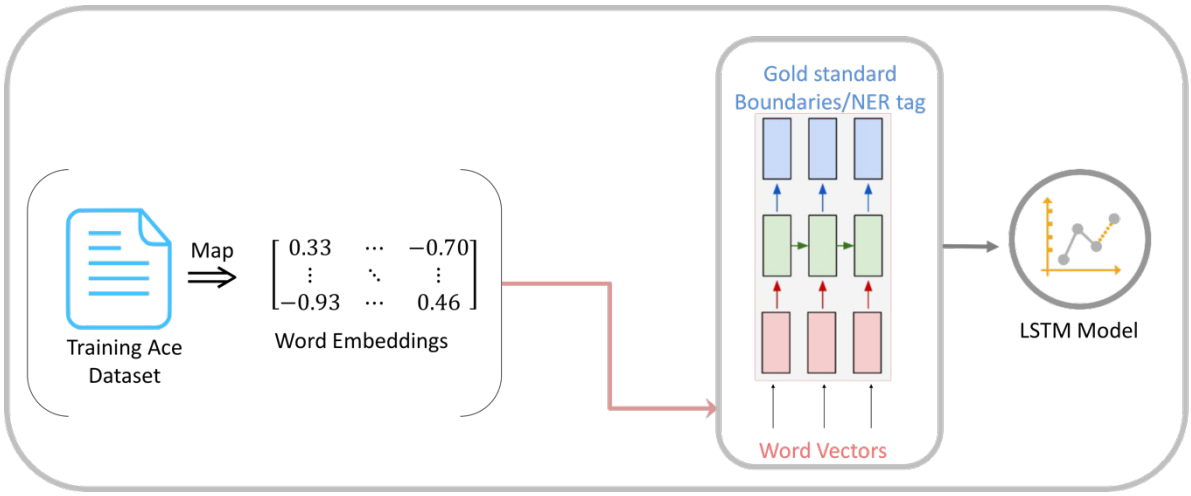

Figure 11: Overall solution to training an LSTM (without POS)

There is an additional step that needs to be factored in when training the classifier with the POS features. We need a way to add the POS features to our embeddings. The way this is done in our system is that each POS for each respective word is computed (See Section 4.4) and concatenated with the respective word embedding. When training, this ensures that some part of our input embeddings to the LSTM has the POS feature encoded in it which is hopefully learned in the training process. The overall process of adding POS to our system is illustrated in Figure 12. Now instead of the mapping going from words in the ACE corpus to word embeddings, it goes from words in the ACE corpus to word embeddings concatenated with POS embeddings. The input is then fed normally to the LSTM for training.

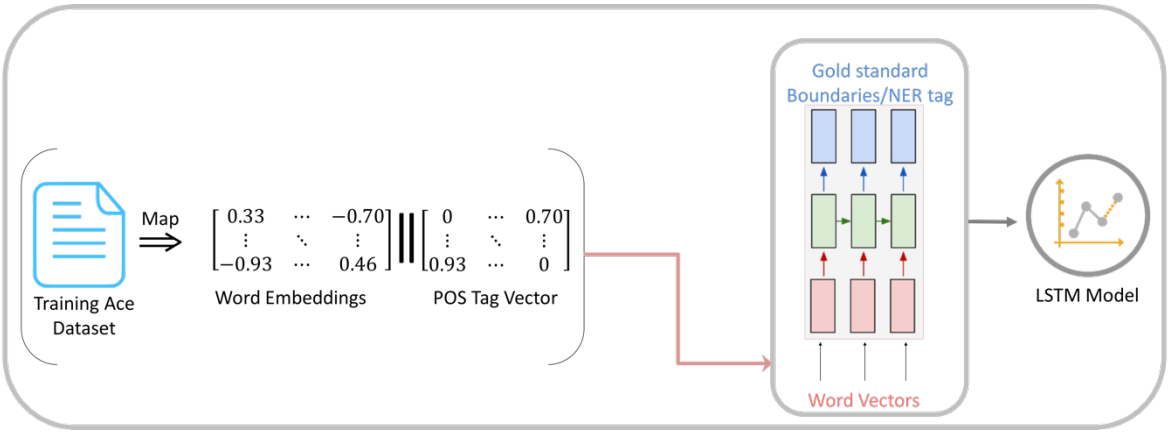

Figure 12: Overall solution to training an LSTM (with POS)

4.6 System Implementation & Parameter Tuning

4.6.1 System Implementation

Our system is implemented using *Keras*[4] and *Theano*[5] as a backend. *Theano* is a Python library that allows one to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. *Theano* is extensively used for machine learning algorithms due to its ability to handle large-scale intensive computation efficiently. *Keras* is a neural network library written in Python and running on top of *Theano*. Because of *Keras*' highly modular and minimalistic nature, it allows for easy and fast prototyping for RNN.

Our system uses a sequential many-to-many LSTM architecture with a hard sigmoid inner activation and a softmax activation on the outer layer. To prevent overfitting, we added a dropout rate. Dropout (Srivastata *et al,* 2014) is a technique that addresses the issue of overfitting. It prevents overfitting and provides a way of approximately combining exponentially many different neural network architectures efficiently. It does so by temporarily removing a unit out of the network - i.e. removing all its ingoing and outgoing connections. The rate at which the dropout happens can be adjusted. Our LSTM is then trained with an Adam optimizer (Kingma *et al,* 2014) with a categorical cross entropy loss function.

4.6.2 Parameter Tuning

When training a neural network, finding the optimal parameter can provide enormous gains in performance. However, due to time constraints, our LSTM has not been optimized at all. Most parameters are at their default value. The parameter settings are listed in Table 1 below.

[4] For more on Keras, see *http://keras.io*
[5] For more on Theano, see *http://deeplearning.net/software/theano/*

Parameter	Setting
Word embedding size	200
POS embedding size	56
Number of hidden nodes (without POS)	200
Number of hidden nodes (with POS)	256
Learning rate	0.001
Dropout rate	0.2
Number of epochs	50

Table 1: Parameter Settings

5 Experiments

To assess the effectiveness of the proposed systems, we conducted experiments on the ACE dataset. 80% of the ACE dataset was used for training and 20% was used as a test set on a 5-fold basis. After training our LSTM, the learned model was provided Arabic sentences from the test set without the NE tags. This process is illustrated in Figure 13. The model then predicted tags for these instances which were then compared with the gold NE tag. If the predicted NE was equal to the gold NE with the appropriate boundaries, it was marked as a correctly classified instance.

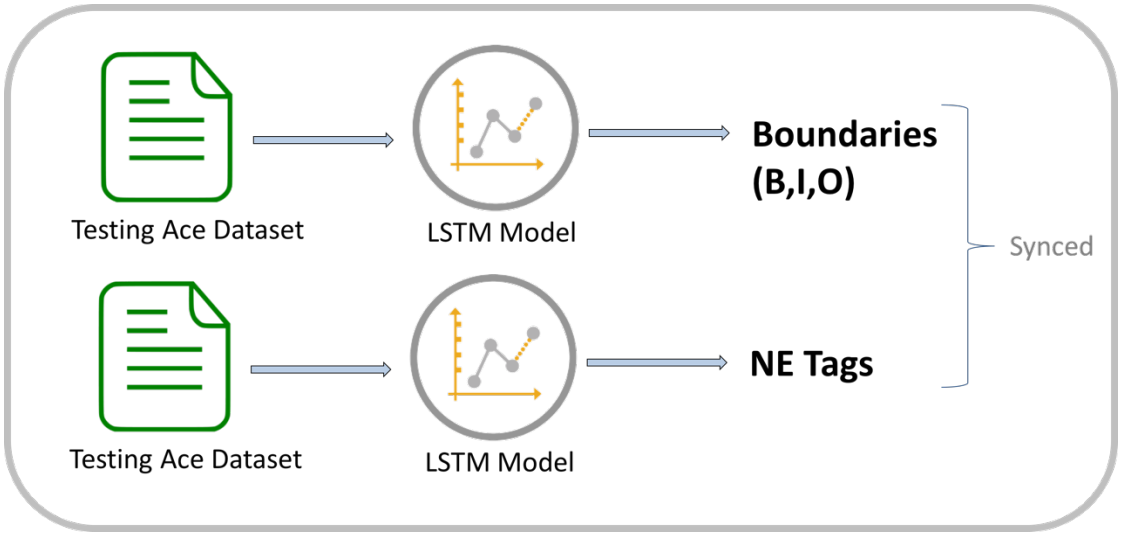

Figure 13: Predictions with the test dataset

The experiment was conducted on the mentioned dataset with training parameters as described in section 4.5.2.

5.1 Evaluation Metrics

In order to evaluate how well our system performed, we use the NER standard metric for precision and recall. Precision is defined as the percentage of NEs found by the system that are correct. Recall is defined as the percentage of NEs presents in the corpus that are found (remembered) by the system.

For our system, the true positives (tp) are the number of named entities (excluding non-named entities) that are actually predicted correctly. The false positive (fp) are the non-NE words that have been predicted as NEs. The false negatives (fn) are the words that are NEs but not predicted as such. Therefore, precision and recall are calculated as follows:

$$precision = \frac{tp}{tp + fp}$$

$$recall = \frac{tp}{tp + fn}$$

To gauge the overall performance of the system, we computed the F1-score. F1 is a measure that combines precision and recall. It is the geometric mean of precision and recall and is calculated as follows:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

5.2 Results

After defining our evaluation metrics, we computed our precision, recall and F1-score for boundary detection and NE type recognition. The results are tabulated in Table 2 below:

	Before POS		After POS	
	Boundaries	**Tags**	**Boundaries**	**Tags**
Precision(%)	62.14	58.11	24.24	22.77
Recall(%)	53.56	49.25	19.94	18.79
F1	57.54	53.31	21.88	20.59

Table 2: Results for system evaluation

5.3 Comparisons & Comments

There are a few systems that perform Arabic NER. However, all of these systems have been evaluated on different datasets that were not available to us at the time of the experiments. This made it hard to evaluate whether our system was state-of-the-art. The closest evaluation we could find was an evaluation of a system on the ACE 2005 Arabic dataset which is also based on newswires and weblogs. The system achieved an F1-score of 58.11 and claimed to be state-of-the-art on some datasets and close to state-of-the-art on others (Benajiba *et al.*, 2010). Extrapolating based on these results, we can conclude that we might not be too far from state-of-the-art.

Additionally, our system has not yet been optimized to find the optimal parameters setting that could potentially boost our performance as parameter exploration is very time consuming – our LSTM took ~18 hours to train – and there is no guideline on

performing parameter optimization. Moreover, we have not yet found an appropriate way to encode our POS tags. In most NER research, POS has been shown to give enormous gains in performance. Finally, other systems currently boast a vast range of features to boost their performance. Such features include n-grams, Gazetteers, base-phrase chunks, gender tagging etc. with some even adding additional data to the training dataset. This leaves much room for potential improvements to our system.

6 Conclusion

6.1 Findings

During this research we have showed that RNN, more specifically LSTM, is a promising classifier for Arabic named entity recognition. Despite not being able to determine our current standing with respect to other systems, comparing our system's F1-scores of 57.54 for boundaries detection and 53.31 for NE type recognition to systems that achieved an F1-score of 58.11 provides evidence that LSTM could be well-suited to the task at hand.

6.2 Limitations & Future Works

With further commitment to this research, many aspects can be improved.

1. Perform parameter exploration: Neural networks are very sensitive to their parameter settings. Finding the optimal settings can provide a big boost to the system performance particularly given that our LSTM has not been optimized at all.

2. Add POS encoding correctly: Finding proper ways to encode features is crucial when training a neural network. We have not yet found a proper way to encode the POS features. Finding the proper encoding can also improve performance – similar to other NER systems.

3. Explore optimal feature set: There are many features that can be used when doing Arabic NER. Such features include n-grams, Gazetteers, base-phrase chunks, gender tagging and many more. Most state-of-the-art system includes those features. Our system has not implemented such features yet. This can be explored as another potential solution to increase performance.

4. Perform error analysis: Little error analysis was done in this study. More analysis needs to be done to understand what features the LSTM is actually learning. This could give more insights on how to fine-tune the LSTM and add more features.

5. Refactor the problem: We are currently factoring the problem in terms of segmentation and then classification. However, there may be alternate approaches that could prove more efficient.

7 References

Abdul-Mageed, Muhammad, Mona Diab, and Mohammed Korayem. 2011. Subjectivity and sentiment analysis of modern standard Arabic. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (HLT 2011): short papers - Volume 2*, pages 587–591, Stroudsburg, PA.

Abdallah, Sherief, Khaled Shaalan, and Muhammad Shoaib. 2012. Integrating rule-based system with classification for Arabic named entity recognition. In Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing*, volume 7181 of *Lecture Notes in Computer Science*. Springer, Berlin Heidelberg, pages 311–322.

Ahmed Abdul-Hamid and Kareem Darwish. 2010. Simplified feature set for Arabic named entity recognition. In *Proceedings of the 2010 Named Entities Workshop*, pages 110–115. Association for Computational Linguistics.

Al-Sughaiyer, Imad and Ibrahim Al-Kharashi. 2004. Arabic morphological analysis techniques: A comprehensive survey. *Journal of the American Society for Information Science and Technology*, 55(3):189–213.

Babych, Bogdan, and Anthony Hartley. "Improving machine translation quality with automatic named entity recognition." *Proceedings of the 7th International EAMT workshop on MT and other Language Technology Tools, Improving MT through other Language Technology Tools: Resources and Tools for Building MT*. Association for Computational Linguistics, 2003.

Benajiba, Yassine, Paolo Rosso, and José Miguel Benedíruiz. "Anersys: An Arabic named entity recognition system based on maximum entropy." In *Computational Linguistics and Intelligent Text Processing*, pp. 143-153. Springer Berlin Heidelberg, 2007.

Benajiba, Yassine, and Paolo Rosso. "ANERsys 2.0: Conquering the NER Task for the Arabic Language by Combining the Maximum Entropy with POS-tag Information." In *IICAI*, pp. 1814-1823. 2007.

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

Benajiba, Yassine, Mona Diab, and Paolo Rosso. 2008a. Arabic named entity recognition: An SVM-based approach. In *Proceedings of Arab International Conference on Information Technology (ACIT 2008)*, pages 16–18, Hammamet.

Benajiba, Yassine, Mona Diab, and Paolo Rosso. 2009a. Arabic named entity recognition: A feature-driven study. *IEEE Transactions on Audio, Speech, and Language Processing*, 17(5):926–934.

Benajiba, Yassine, Mona Diab, and Paolo Rosso. 2009b. Using language independent and language specific features to enhance Arabic named entity recognition. *The International Arab Journal of Information Technology (IAJIT)*, 6(5):463–471.

Benajiba, Yassine, Imed Zitouni, Mona Diab, and Paolo Rosso. "Arabic named entity recognition: using features extracted from noisy data." In *Proceedings of the ACL 2010 conference short papers*, pp. 281-285. Association for Computational Linguistics, 2010.

Bengio, Yoshua, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult." *Neural Networks, IEEE Transactions on* 5.2 (1994): 157-166.

Diab, Mona. 2009. Second generation tools (AMIRA 2.0): Fast and robust tokenization, POS tagging, and Base Phrase Chunking. In *Proceedings of the Second International Conference on Arabic Language Resources and Tools*, pages 285–288, Cairo.

Guoqiang Peter Zhang, "Neural networks for classification: a survey", *IEEE Trans. Systems, Man and Cybernetics*, vol. 30, no. 4, pp. 451–462, 2000.

Habash, Nizar, and Owen Rambow. "Arabic tokenization, part-of-speech tagging and morphological disambiguation in one fell swoop." *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 2005.

Habash, Nizar, Owen Rambow, and Ryan Roth. "MADA+ TOKAN: A toolkit for Arabic tokenization, diacritization, morphological disambiguation, POS tagging, stemming and lemmatization." *Proceedings of the 2nd international conference on Arabic language resources and tools (MEDAR), Cairo, Egypt*. 2009.

Habash, Nizar, Ryan Roth, Owen Rambow, Ramy Eskander, and Nadi Tomeh. "Morphological Analysis and Disambiguation for Dialectal Arabic." In *HLT-NAACL*, pp. 426-432. 2013.

Hammerton, James. "Named entity recognition with long short-term memory." *Proceedings of the Seventh Conference on Natural language learning at HLT-NAACL 2003-Volume 4*. Association for Computational Linguistics, 2003.

Hewavitharana, Sanjika and Stephan Vogel. 2011. *Extracting parallel phrases from comparable data*. In Proceedings of the 4th Workshop on Building and Using Comparable Corpora, 49th Annual Meeting of the Association for Computational Linguistics (ACL), pages 61–68, Portland, OR.

Hiroyuki Toda and Ryoji Kataoka. 2005. *A Search Result Clustering Method using Informatively Named Entities*. In *Proc. of the 7th ACM International Workshop on Web Information and Data Management*.

Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9, no. 8 (1997): 1735-1780.

Khaled Shaalan. 2014. *A survey of Arabic named entity recognition and classification*. *Comput. Linguist.*, 40(2):469–510, June.

Kingma, Diederik P. and Ba, Jimmy. 2014. *Adam: A Method for Stochastic Optimization*. *arXiv:*1412.6980.

Oudah, Mai and Khaled Shaalan. 2012. A pipeline Arabic named entity recognition using a hybrid approach. In *Proceedings of the International Conference on Computational Linguistics*, pages 2,159–2,176, Mumbai.

Pasha, Arfath, Mohamed Al-Badrashiny, Mona T. Diab, Ahmed El Kholy, Ramy Eskander, Nizar Habash, Manoj Pooleery, Owen Rambow, and Ryan Roth. "MADAMIRA: A Fast, Comprehensive Tool for Morphological Analysis and Disambiguation of Arabic." In *LREC*, pp. 1094-1101. 2014.

Ryding, Karin. 2005. *A Reference Grammar of Modern Standard Arabic*. Cambridge University Press, New York.

Sergio Ferràndez, Òscar Ferràndez, Antonio Ferràndez and Rafael Muñoz. 2007. *The Importance of Named Entities in Cross-Lingual Question Answering* In *Proc. of RANLP'07*.

Shaalan, Khaled and Hafsa Raza. 2009. NERA: Named entity recognition for Arabic. *Journal of the American Society for Information Science and Technology*, 60(8):1,652–1,663.

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A simple way to prevent neural networks from overfitting." *The Journal of Machine Learning Research* 15, no. 1 (2014): 1929-1958.

Tim Buckwalter. 2002. *Buckwalter Arabic Morphological Analyzer*. In *Linguistic Data Consortium. (LDC2002L49)*.

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient estimation of word representations in vector space." *arXiv preprint arXiv:1301.3781* (2013).

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional LSTM-CRF models for sequence tagging. *CoRR*, abs/1508.01991.