Language Independent Named Entity Recognition via Orthogonal Transformation of Word Vectors

Omar E. Rakha and Hazem M. Abbas

Dept. Computer and Systems Engineering
Faculty of Engineering
Ain Shams University, Cairo 11571, Egypt

Abstract

Word embeddings have been a key building block for deep learning NLP in which models relied heavily on word embeddings in many different tasks. In this paper, a model is proposed based on using Bidirectional LSTM/CRF with word embeddings to perform named entity recognition for any language. This is done by training a model on a source language (English) and transforming word embeddings from the target language into word embeddings of the source language by using an orthogonal linear transformation matrix. Evaluation of the model shows that by training a model on an English dataset the model was capable of detecting named entities in an Arabic dataset without neither training or fine tuning the model on an Arabic language dataset.

1. Introduction

Named Entity Recognition (NER) is an important component of information extraction that aims at extracting named entities from textual data. Named entities are words that represent a known person, location, organization or any other entity that can be identified by its name. Extraction of named entities can be used to further improve search engine queries [1] or for question answering [2]. Detection of named entities requires building hand annotated datasets for each target language which consumes significant human labor and time.

NER systems used to rely on hand coded features, part-of-speech (POS) tags, semantic lexicons and huge gazetteers [3]. However this restricted the model's performance and required huge work and domain knowledge to find suitable features.

Current state-of-the-art systems in NER are very accurate with performance exceeding 90% [4] [5] by using a neural network composed of a Bidirectional LSTM layer [6] and a Conditional Random Field (CRF) classifier [7]. However, these models are only trained for a specific language (English for example) and can not be used on any other language.

Due to language dependence, extending the knowledge obtained by these models to new languages is impossible. Thus trying to support new languages requires building new models that only work on the language they are trained on. The effect of this is that the number of languages that can be supported in an application is limited by the amount of labeled data that can be obtained in every language and the performance of each model.

In this work, a novel method is proposed for NER by training a model on an English corpus and evaluating the resulting model on an Arabic corpus by transforming

the Word embedding space from Arabic to English. The method can be extended to any language as long as it is possible to create the transformation matrix for its word embeddings. The proposed model does not explicitly rely on language specific features since the employed word embedding model implicitly captures language specific features and benefits from morphological features of words. This brings about a model that is both simple and efficient.

The paper organization is as follows. The problem is defined in details in Section 2. Previous NER models are reviewed in Section 3. The proposed language independent NER model is presented in Section 4. Experimental results produced when the model is applied to a new target language are analyzed in Section 5 and Section 6 concludes the paper.

2. Problem Statement

The language independent NER problem can be composed of two sub problems the first being the detection of named entities and the second being language independence. The sub problem of detecting named entities is a sequence labeling problem in which the goal is to find out words that are named entities (e.g.,: Persons, Organizations, Locations, ...).

A named entity is a word or a phrase that stands consistently for some referent. This includes names of people, places, organizations. This can also include temporal expressions (e.g., January fifth 2010, August, 20 November). Numerical expressions can also be considered named entities (e.g., \$50, 25.5%).

For example in Figure 1 each word is labeled in a text sequence as a named entity (In this case a person and a location). Outside words are labeled with "O"

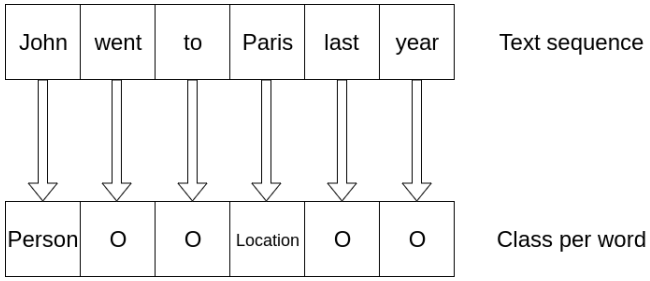


Figure 1: An NER Example

following the IOB (Inside-Outside-Beginning) format presented in [8]. The IOB format prefixes the class of the beginning word with a "B-" indicating it's the first word in this named entity. while the following words that belong to the same entity will be prefixed with "I-" which stands for inside. This way we can label named entities that are longer than 1 word. Refer to Figure 2 for illustration.

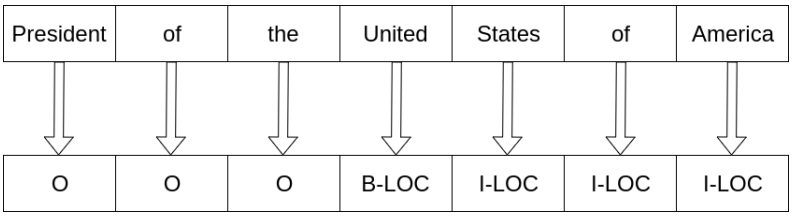


Figure 2: The IOB format

The second sub problem addressed here is language independence. The goal of building language independent models is to have solutions that can ultimately solve problems regardless of the language it receives. Ideally these models will learn the underlying structure of human language and thus be able to tackle the problems addressed at it without being concerned with the language it uses.

For example there exist many data sets for NER in English. However, Arabic data is very scarce. A model that is trained using English data and can be extended to solve Arabic input is said to be language independent (Figure 3).

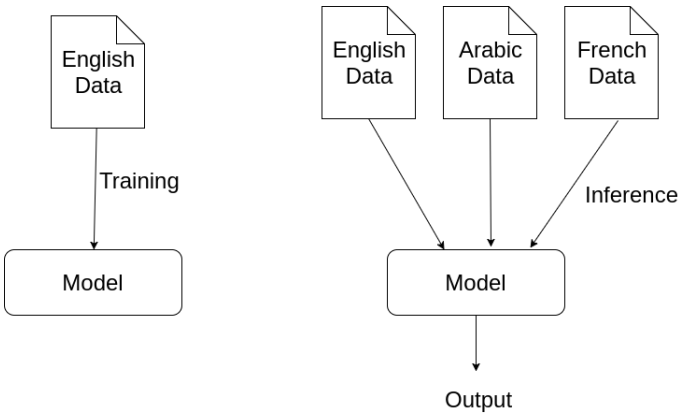


Figure 3: A language independent model

3. Related Work

Currently NER multilingual systems approach the problem by relying on data sets that are designed for building multilingual models like the UN parallel corpora [9]. In following subsections we will explore these approaches and their limitations before proposing the new approach.

3.1. Cross Lingual Resources

The cross lingual resources approach is centered around finding common features in languages or features that are language dependent. The goal of this approach is to unify the features of more than one language in order to build a classifier that is capable of finding the named entities based only on these features.

The system [10] incorporates Wikipedia and other knowledge bases like DBpedia [11]. It uses features like cross-lingual capitalization, transliteration and DBpedia based labeling.

Cross-lingual Capitalization: Since Arabic has no capitalization DBpedia was used to find capitalization weights for a word and be used as a feature.

An example: the phrase "Pacific Ocean" was capitalized 36.7% of the time. Thus, the arabic word for "Ocean" was assigned the feature "B-CAPS-0.4" and the arabic word for "Pacific" was assigned the feature "I-CAPS-0.4". The prefix of the feature determines if the word was a "Beginning" or "Inside" word. The body of the feature "CAPS" symbols capitalization and the suffix of the feature is the frequency of being "CAPS".

Transliteration: The intuition behind transliteration is that named entities are usually transliterated rather than translated. For example the word "Hassan" is an arabic name and also means "good". They use a transliteration weight as a feature similar to the capitalization weight.

DBpedia based labeling: DBpedia is a large collaboratively-built knowledge base in which structured information is extracted from Wikipedia. For example, NASA is assigned the following types: Agent, Organization, and Government Agency. These types were used as a feature for each word (if it exists in DBpedia).

The problem of this approach is the heavy reliance on the nature of a language, i.e., it has capitalization, transliteration is valid, and it is well represented in DBpedia.

3.2. Multi Task Cross Lingual Training

Due to the limitations of the previous approach in having to find features that are viable in more than one language, like Capitalization, a better approach was needed to overcome this limitation. A suitable approach was to rely on embedding to let the model find these features by itself.

In multi task cross lingual training [12], a model is trained on multiple tasks such as POS, chunking and NER for more than one language at the same time by using a character level embedding GRU [13] model and a word level embedding model. This model is trained with a shared embedding layer to capture the morphological similarity between languages and learn word vectors that are based on the two languages. This embedding space will capture the similarities between words from the two languages. The final layer is a Conditional Random Field (CRF) classifier [7] which has specific weights for each task.

The model in (Figure 4) is structured as pluggable components. The bottom component is an embedding layer. This layer is shared among different tasks and different languages, which means that the model will use the same embedding layer if it was trained for English NER, English POS, Arabic Chunking or Spanish POS. The preceding layer is a task dependent CRF [7] classifier that works on top of the shared embedding layer to produce task specific results (for example: named entities in case of NER).

However this model will require a huge amount of annotated data in every needed language. In addition, the model will require retraining whenever more languages need to be added. Also the parameters of the model will increase significantly with the number of languages and hence increase the complexity of the model. The shared embedding layer will not be able to capture any similarity between languages that are intrinsically different like English and Arabic.

These approaches show promising results on the multilingual NLP tasks, however the growing complexity of these models with the number of languages makes it near impossible to have models that can work on a huge number of languages. Building a model to work with a few languages would consume a considerable amount of resources(time, annotation effort, ...). Therefore an approach that would work independently of the number of languages is needed.

4. Proposed Approach

In this section, an approach to build a model that is both efficient and can scale reliably with the number of languages presented. The idea of this approach

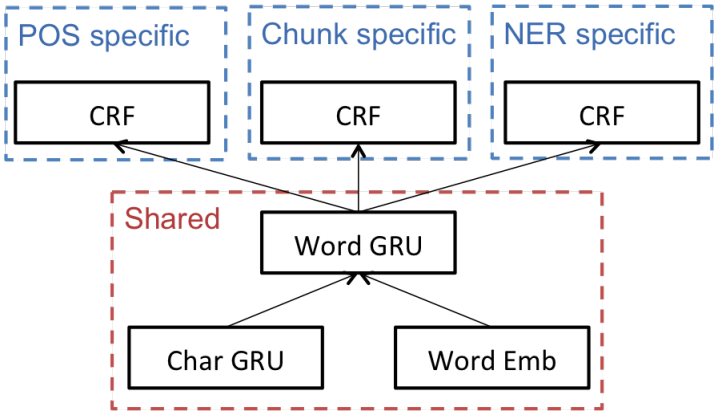


Figure 4: The Multi Task Cross Lingual Training model

is to build a neural NER model using the standard components for the problem: Word embedding for embedding words into vectors, Bidirectional LSTM for feature extraction and a Conditional Random Field sequence classifier.

However to introduce language independence to the model a new step; Word embedding transformation, is added to the pipeline, which will be able to transform the vectors of the words of a certain target language (Arabic in our case) into the space of the vectors of the source language (English in our case) which we trained our model on. This way the model will be independent of the language of the input and will only depend on the quality of the Word Embedding Transformation technique that is applied. Thus the model will be composed of four components:

1. Word Embedding
2. Word Embedding Transformation
3. Bidirectional LSTM Layer
4. Conditional Random Field

Figure 5 shows the architecture of the proposed neural architecture. First, each word in a sentence is embedded into its corresponding word vector using the word embedding model. If the word vectors are in a language different than the source language (English in this case) the word embedding transformation is invoked to transform the vector from the space of the target language to the space of the source language. The resulting word vector sequence is then used as the input to a bidirectional LSTM layer which encodes the input sequence. Finally a CRF layer is used to tag the sequence.

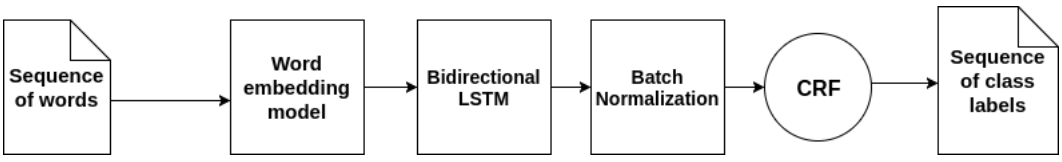


Figure 5: Architecture of the proposed model

4.1. Word Embedding

Because language is a sequence of words, while neural networks only work with numbers we have to transform our words into a numerical format. Vector space

models are the numerical representation of documents in which a vector is used as a representation for this document. An example of this is Bag of Words(BOW) [14].

In BOW models a vector is assigned for each document representing word counts for each word in the vocabulary in this document. Consider our vocabulary to only have the words: ["the", "boy", "girl", "rides", "bus", "and", "from"]. A representation of the string "the boy rides the bus and the girl rides the bus" would be [4, 1, 1, 2, 2, 1, 0] which is the count of each word in the vocabulary found in this string.

Vectors for words can also be computed using BOW by having a one hot representation over the vocabulary. For example the word "boy" would receive the vector [0, 1, 0, 0, 0, 0, 0]. this way we can have vectors representing every word in our vocabulary.

The limitation of this approach is that vectors are very sparse since for a word the vector will be mostly zeros except for a single value that corresponds to the word. These vectors also lack any information on the semantic meanings of these words since the position of the word in the vocabulary gives no information about similar words to it.

Due to these limitations distributed representation of words outperforms traditional BOW methods because they are dense vectors that can learn semantic and syntactic properties of these words.

A famous example for this taken from [15] is how their model learned a relationship between Country-Capital in the PCA projection of these vectors (Figure 6).

From the 2 dimensional PCA projection we can see that the model learned similar representation for countries (China, Russia, Japan, ...) and similar representation for capitals (Beijing, Moscow, Tokyo, ...) and the direction of the vector between a country-capital is similar. For example the relationship $model(\text{"}China\text{"}) - model(\text{"}Beijing\text{"}) \approx model(\text{"}Russia\text{"}) - model(\text{"}Moscow\text{"})$ can be seen from the projection.

These models are trained on huge corpora of data -Wikipedia for example- and the model learns from the context information of words the best representation that would capture semantic and syntactic information and embed it into the highly dense vector.

Our model employs the Word embedding model introduced in [16] which is a Skip Gram model [15] with subword information. Given a matrix $\mathbf{W}$, an entry $\mathbf{w}^i$ is the ith vector in the matrix which is the vector representation of the word i. The Skip Gram's objective is to find word representations that are useful for predicting the surrounding words in a sentence or a document. The model is trained to maximize the average log probability over a sequence of words $\mathbf{w_1}, \mathbf{w_2}, \mathbf{w_3}, ..., \mathbf{w_T}$ where c is the size of the training context (training window) and $\mathbf{w}_t$ is the center word.:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} \log p(\mathbf{w}_{t+j}|\mathbf{w}_t) \tag{1}$$

The basic skip gram model defines the probability of word $t+j$ given word t as $p(\mathbf{w}_{t+j}|\mathbf{w}_t)$ using the softmax function:

$$log(p(\mathbf{w}_O|\mathbf{w}_I)) = log(\frac{exp(v'^{\top}_{\mathbf{w}_O} v_{\mathbf{w}_I})}{\sum_{w=1}^{W} exp(v'^{\top}_{w} v_{w_I})}) \tag{2}$$

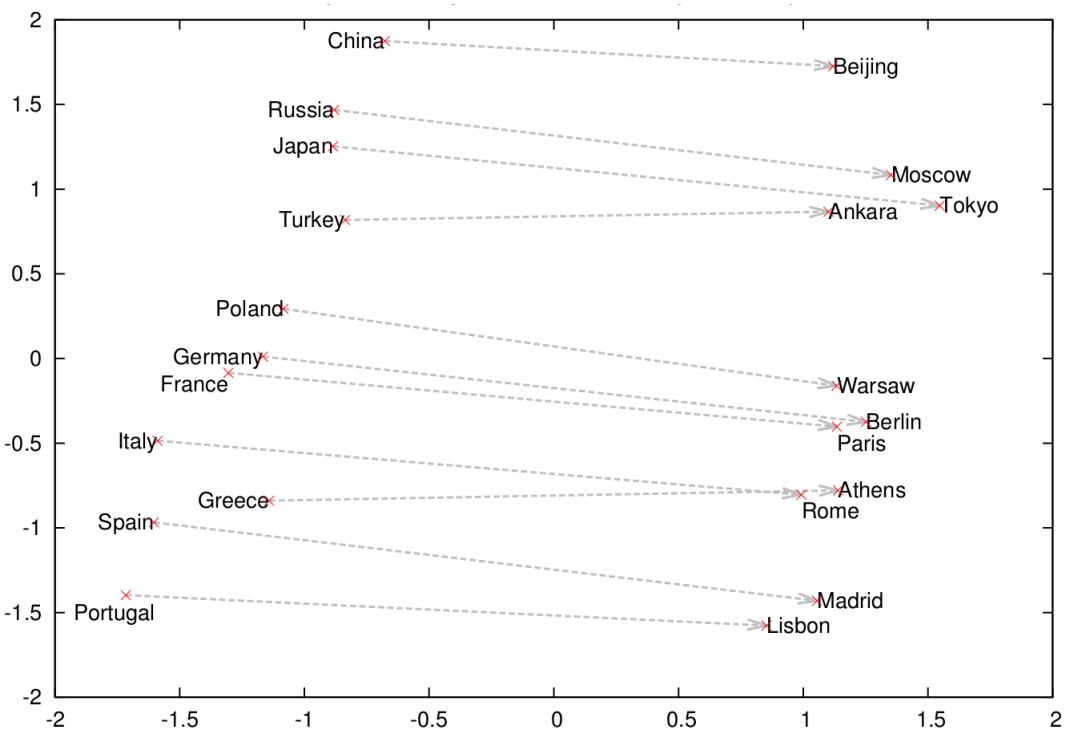


Figure 6: Country-Capital Relationship

where v_w and $v_w^{'}$ are the input and output vector representations of $\mathbf{w}$ and W is the number of words in the vocabulary. This can be simplified into:

$$v_{\mathbf{w}_O}^{'\top} v_{\mathbf{w}_I} - log(\sum_{w=1}^{W} exp(v_w^{'\top} v_{w_I})) \tag{3}$$

which consists of two terms, the dot product $v_{\mathbf{w}_O}^{'\top} v_{\mathbf{w}_I}$ which represents the similarity between the two vectors and the other term $log(\sum_{w=1}^{W} exp(v_w^{'\top} v_{w_I}))$ being the normalization term. Computing the normalization term is very expensive since it requires summing over all words in the vocabulary. This makes it impractical to compute the gradient of the softmax term $\bigtriangledown p(\mathbf{w}_O|\mathbf{w}_I)$ which is proportional to the size of the vocabulary W.

Negative sampling (NEG) is used as the objective for training the Skip Gram model instead of the softmax to overcome the problem of computing the normalization factor. NEG is defined as:

$$\log \sigma(v_{w_O}^{'\top} v_{w_I}) + \sum_{i=1}^{k} \mathop{\mathbb{E}}_{\mathbf{w}_i} \sim P_n(w)[log\, \sigma(-v_{w_i}^{'\top} v_{w_i})] \tag{4}$$

Negative sampling works by randomly selecting a set of k "negative" words drawn from a uni-gram distribution (P_n), These words are the words the model should not be predicting and instead of updating the weights for every word in the vocabulary that is not the target word, the optimizer will only update weights for this negative sample avoiding a huge computation that would make model training in-feasible.

In other words the vectors are trained by teaching the model to differentiate between the target word and noise.

So far the model is only capable of learning a distinct vector representation per word. However, this ignores the morphological features of words (prefixes, suffixes, n-grams, ...). Therefore, the scoring function needs to be adjusted to take this information into account.

A proper modification for the model would be to learn vector representations for words and for character n-grams instead of learning vectors for words only. The implication of this is that each word will be decomposed into a set of n-grams and the word itself, each element of this set has a distinct vector. this allows for n-gram vector sharing between words thus producing high quality vectors that take into account the structure of the word itself.

For $\varrho_w \subset 1, ..., G$ with G being the set of n-grams appearing in $\mathbf{W}$ and ϱ_w being the set of n-grams appearing in word w, a vector representation is associated with each n-gram, g, a word is represented by the sum of the vectors of its n-grams and a unique vector for the word itself, the scoring function is therefore:

$$s(w, c) = \sum_{g \in \varrho_w} z_g^\top v_c \tag{5}$$

Where z_g is a vector representation of each n-gram g.

For example the word "Good" can be represented by the set of character n-grams of length 3,4 "Goo", "ood", "Good" where every n-gram of which will have its own vector representation.

4.2. Word Embedding Transformation

The idea of this model is to be trained on a corpus from a source language (English) Figure 7a -since there are available annotated data- and test the model on a target language with less data (Arabic) Figure 7b, this wouldn't be possible if we use the word embedding of the two languages separately. This is due to them having different distributions and the model will not understand the target word embedding. However, we follow an approach [17] to transform the word embedding of our target language to our source language and then use it on our model Figure 7c. This approach is tested by training the model on an English dataset [18] and then evaluate the model on an Arabic dataset [3].

We can see in Figure 7a two English words projected on a 2 dimensional space and in Figure 7b the same two words in Arabic projected on their embedding space. We can clearly see that the two embedding spaces are different and that the two words do not overlap. The goal of Word Embedding Transformation is to find a feasible transformation that can transform the vectors of one of the two spaces into the other space in order to have words with the same meaning ideally overlapping Figure 7c.

The transformation can be performed by using a translation matrix that is capable of approximating target word embedding into source word embedding. Using a translation matrix is a powerful approach that will map one matrix to another by matrix multiplication.

Suppose we have a set of word embeddings that correspond to our target and source languages X, Y where $X \in R^{n*d}$ and $Y \in R^{n*d}$ such that n is the number of vectors and d is the dimension of each vector, our goal is to find a matrix W that approximates the function:

$$\sum_{i=1}^{n} ||Wx_i - y_i||^2 \tag{6}$$

Where x_i is our target language vector and y_i is the source language vector for the ith word. We can solve this equation using Stochastic Gradient Descent to find the optimal matrix W that minimizes the error function. Now by using this matrix W we can translate any vector from our target language X into our source language Y by multiplication with Wx_i.

However, this approach produces sub optimal results due to falling to local minima and being susceptible to learning rates, as well as being time consuming as you have to iterate over your full dataset multiple times, and being susceptible to the problems of training neural networks. another solution is favored over this one.

It is argued that the translation matrix *must* be an orthogonal matrix [19], to prove that let's first form the similarity matrix $S = Y(WX^T)$ where $\mathbf{Y}$ represents the word vectors matrix of our source language and $\mathbf{X}$ represents the matrix of our target language, the matrix $\mathbf{W}$ is the transformation matrix.

Thus the matrix $\mathbf{S}$ represents the dot product between every vector in the matrix $\mathbf{Y}$ and its corresponding vector in the transformed matrix WX^T.

Looking at the dimensions of the matrices $Y^{n*d}.W^{d*d}.X^{d*n}$ we notice that the similarity matrix's dimensions are S^{n*n} this represents for each vector in the target language its similarity with every vector in the source language. We can also have a reverse similarity matrix between the source vectors and the target vectors defined by $S' = XQY^T$.

For this to remain self consistent, i.e the two similarity matrices have the same values, because they both calculate the similarity between the two matrices $\mathbf{X}$ and $\mathbf{Y}$, we must have $S' = S^T$ but $S^T = XW^TY^T$ therefore $Q = W^T$ which means that if W maps the source language to the target language then W^T maps the target language back to the source language.

Given the matrix W we assume that $x \approx W^Ty$ and $y \approx Wx$ therefore $x \approx W^TWx$ therefore we can conclude that the transformation matrix W must be an orthogonal matrix satisfying $W^TW = I$, where I denotes the identity matrix.

We now modify our objective function to satisfy the orthogonality constraint on W to be:

$$\max_O \sum_{i=1}^{n} ||y_i^T W x_i||^2, \text{ subject to } W^TW = I \tag{7}$$

This solution uses Singular Value Decomposition(SVD) to find the translation matrix. SVD is favored over Principal Components Analysis (PCA) to avoid the cost of calculating the huge covariance matrix over the dictionaries.

The way SVD works is by factoring an $m * n$ matrix M into a product of three matrices:

$$M = U\Sigma V^T \tag{8}$$

Where U is $m * k$, Σ is $k * k$ and V^T is $k * n$. SVD is very common in the field of natural language processing, it is used in many algorithms including but not limited to:

- Latent Semantic Analysis
- Latent Semantic Indexing
- Visualization of Word embedding

We follow [17]'s method which uses SVD only. Given the source language dictionary Y_D and the target language dictionary X_D we normalize each vector by dividing it by its norm. We then multiply the two dictionaries together $M = Y_D^T X_D$ and then compute the SVD of M which is equal to $M = U\Sigma V^T$.

To map the two languages to a single space we multiply each dictionary with one of the decompositions, our similarity function will become

$$S = YUV^T X^T \tag{9}$$

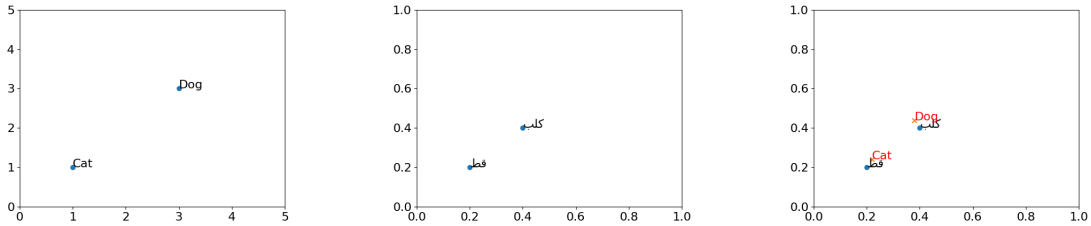
where

$$s_{ij} = y_i^T U V^T x_j \tag{10}$$

$$= (U^T y_i).(V^T x_j) \tag{11}$$

thus the two dictionaries are mapped to a single space.

This approach produces an exact solution (Equation 9) to our objective function (Equation 7).



(a) An example of two English words in their embedding space

(b) An example of their Arabic translation in their embedding space

(c) Ideally after the transformation the two Arabic words will overlap with the two English words

Figure 7: Word embedding transformation

4.3. Bi-LSTM encoding

Neural Networks have always had a strong ability to perform feature extraction even with time series or sequences they're still able to capture time related features due to their hierarchical nature and recurrence relations.

For this we will be using the Long-short term memory [20] algorithm to perform feature extraction from our word vectors sequence in order to convey temporal features to the classifier.

LSTM is the core of our model (Figure 8) since it will be responsible for extracting the useful features from the word vectors to help the classifier find the correct tags for the sequence.

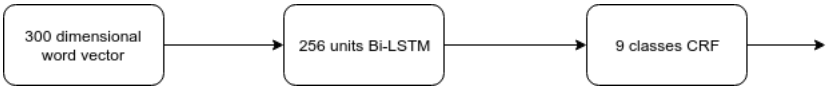


Figure 8: Model detailed architecture for each time step

The main strength of LSTM is its ability to handle long term dependencies in a sequence of steps while updating its state with information from context due to its gates that are responsible for updating cell states.

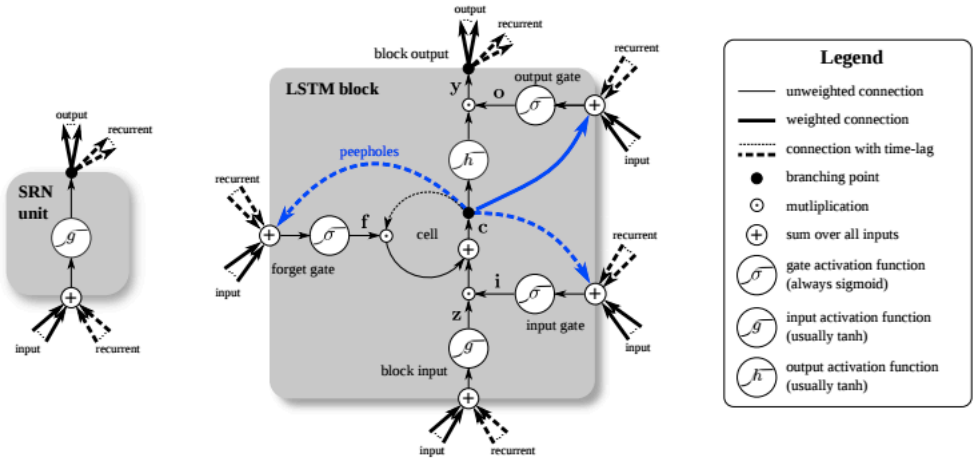


Figure 1. Detailed schematic of the Simple Recurrent Network (SRN) unit (left) and a Long Short-Term Memory block (right) as used in the hidden layers of a recurrent neural network.

Figure 9: LSTM architecture (courtesy of Skymind AI)

Each LSTM unit (Figure 9) holds a cell state C which flows from timestep $t-1$ to t where some modifications occur in between due to the LSTM gates, these gates are a way to optionally let information pass through. They are composed of a sigmoid layer that outputs a value between 0 and 1 and a multiplication operation in which multiplying by 0 means prevent the information flow and 1 means allow full information flow. The forget gate receives the hidden state at the previous time step h_{t-1} and the input x_t and decides how much information needs to be forgotten by the following equation:

$$f_t = sigmoid(W_f \,.\, [h_{t-1}, x_t] + b_f) \tag{12}$$

The input gate decides what new information need to be stored in the cell state, it follows the following equation:

$$i_t = sigmoid(W_i \,.\, [h_{t-1}, x_t] + b_i) \tag{13}$$

We also compute a candidate set of values to be added to the cell state:

$$\tilde{C}_t = tanh(W_c \,.\, [h_{t-1}, x_t] + b_c) \tag{14}$$

Following these computations is the update on the cell state which occurs by the equation:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{15}$$

Finally the model decides on what it should output according to its output gate:

$$o_t = sigmoid(W_o \,.\, [h_{t-1}, x_t] + b_o) \tag{16}$$

The output to the hidden state is then computed by:

$$o_t = tanh(C_t) \tag{17}$$

A Bi-directional LSTM layer (Figure 10) is added to our model instead of the classic Uni-directional LSTM since allowing the model to read text right-to-left and

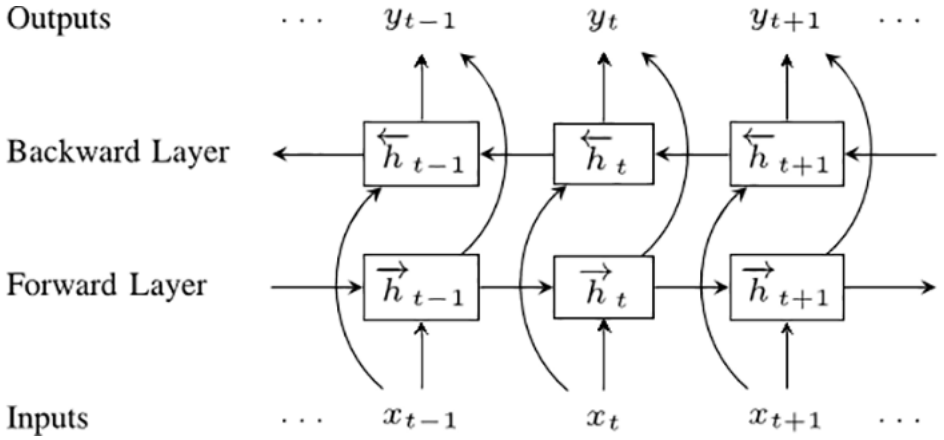


Figure 10: Bi-LSTM architecture [21]

left-to-right and aggregating their outputs would improve it significantly by peeking into the future as well as benefiting from the past [4] [5]. It will also make the model invariant to language writing direction. for example we train the model on English which is a left-to-right language and evaluate it on Arabic which is a right-to-left language.

We pass the h_t vector for every time step to the next layer which can be another layer of Bi-LSTM for an added layer of feature extraction or a Conditional Random Field classifier.

4.4. Conditional Random Field Classifier

Going from the features extracted by the Bi-LSTM to class labels requires a classifier that can take as input the features at each time step and produce as output the class label for each word.

For sequence classification the most common classifiers used in the domain of NER are:

- LSTM
- Hidden Markov Model
- Simple RNN
- Conditional Random Field

The CRF classifier is used for its strength and its current results in state of the art NER models [4] [5]. A huge strength of CRF is its ability to work with output classes that are dependent, an example of this is that for any I-*(Inside class) there must be a preceding B-*(Beginning class). CRF is capable of exploiting this dependency thus improving the results and avoiding unintuitive class outputs.

Let $h_{1:N}$ be our input to the CRF layer which are the features for each word in a sequence of length N where (1:N) denotes the words from index 1 to N. These features are calculated by h_t = Bi-LSTM(x_t). let $z_{1:N}$ be our class labels for the input sequence. The linear chain CRF model defines conditional probability of the class labels given the input by the equation:

$$p(z_{1:N}|h_{1:N}) = \frac{1}{Z} exp\,(\sum_{n=1}^{N}\sum_{i=1}^{F}\lambda_i f_i(z_{n-1}, z_n, h_{1:N}, n)) \tag{18}$$

In the equation the term Z is the normalization factor or the partition function which is used to make $p(z_{1:N}|h_{1:N})$ a valid probability, the term F is the number of feature functions we're going to use.

$$Z = \sum_{z_1:N} exp\,(\sum_{n=1}^{N}\sum_{i=1}^{F} \lambda_i f_i(z_{n-1}, z_n, h_{1:N}, n)) \tag{19}$$

In the probability equation we had feature functions f_i, each of which is a function that takes as input the current class label, the label of the previous class, the input and the current position and outputs a boolean value 0 or 1. λ_i is the feature weight for feature i, if this weight is large and positive then we emphasis that the probability of the current label is high if this feature is true. An example of a feature function: $f_1(z_{n-1}, z_n, h_{1:N}, n) = \begin{cases} 1, & \text{if } z_{n-1} = \text{B-PERS and } h_t = \text{Bi-LSTM(John)} \\ 0, & \text{Otherwise} \end{cases}$ A positive weight for this function would indicate that the model prefers the tag B-PERS for the current word, if the function was assigned a negative weight it would indicate that the model does not prefer the tag B-PERS for the current word.

In our case the features will be the output of the Bi-LSTM. In this case the Bi-LSTM layer(s) can be considered as the feature function we are using.

The appropriate objective for parameter learning of CRF is to maximize the conditional likelihood of the training data where m is the number of training examples:

$$\sum_{j=1}^{m} log\ p(z_{1:N}|h_{1:N}) \tag{20}$$

This learning procedure is conducted by computing the gradients for the objective function (Equation 20) and use the gradient in a gradient based optimization algorithm like Stochastic Gradient Descent.

Now that the model is trained we can use it to calculate the probability $p(z_{1:N}|h_{1:N})$ for any tag for each token in an input sequence. An approach to take the final sequence is by greedily taking the tag that has the highest probability. However this approach would have to check an exponential number of tags because it has to check K^N possible tags with K being the number of classes in our model. Another approach would be to use the Viterbi algorithm [22] which is a dynamic programming algorithm that works in polynomial time to find the optimal sequence of tags.

5. Experimental Results

First we define the metrics we used in our evaluation. We used Precision and Recall and their harmonic mean (F1-score).

Precision is defined as the number of true positives divided by the number of true positives and false positives. It is considered as a measure of the classifier's exactness.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{21}$$

Recall is defined as the number of true positives divided by the number of true positives and false negatives. It can be considered the sensitivity or the true positive rate.

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{22}$$

In our evaluation we use the F1-score which is a balance between them.

$$F1score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{23}$$

We trained our model on the WikiNER English dataset [18] by using the word embedding model [16] which is open source. We evaluated the model on English and noted its results then we tested it on the ANER dataset [3] with only applying the transformation and no fine tuning of the model.

The model was written using Keras [23], it was trained on an Ubuntu 16.04 system running on an Intel i7-7500U processor and accelerated by an Nvidia Geforce 940MX GPU. All the code for the experiment is available as open source on Github: Code on Github

The results of the model imply that the model succeeded at detecting many named entities in the Arabic text without being exposed to Arabic before. This approach is valid for any other language that we have a transformation matrix for. Recently 78 matrices for transformation [17] have been released as open source. These matrices can be used to transform the word embedding space from any of the 78 languages to English, which can be used to detect named entities in 78 different languages with only being trained on English.

Classification results per class for English			
	Precision	Recall	F1-Score
PER	0.88	0.93	0.91
MISC	0.80	0.73	0.76
LOC	0.86	0.85	0.86
ORG	0.82	0.72	0.76
Average scores	0.84	0.82	0.83

Table 1: Classification Report for Precision, Recall and F1-Scores on WikiNER

We can see from Table 1 that the model successfully learns to detect named entities in the English text. The model's accuracy depends on many factors among which the training data and the complexity of the model. The model seems to perform best on the persons class and seems to perform worst on the miscellaneous class.

Classification results per class for Arabic			
	Precision	Recall	F1-Score
PER	0.05	0.01	0.01
MISC	0.01	0.57	0.02
LOC	0.07	0.00	0.00
ORG	0.02	0.05	0.03
Average scores	0.05	0.07	0.02

Table 2: Classification Report for Precision, Recall and F1-Scores on ANER without Alignment

In Table 2 the model is tested on ANER which is the Arabic dataset without aligning the word vectors with the English word vectors. Recall that the model was never trained to understand Arabic and so testing the model on Arabic would

produce a score of 0 (or almost 0) which is the case as seen in the table. The model fails to capture any entities -Except for a very tiny number of entities that could be random-.

Classification results per class for Arabic			
	Precision	Recall	F1-Score
PER	0.68	0.52	0.59
MISC	0.05	0.06	0.06
LOC	0.80	0.44	0.57
ORG	0.32	0.08	0.12
Average scores	0.58	0.36	0.43

Table 3: Classification Report for Precision, Recall and F1-Scores on ANER with Alignment

The results in Table 3 shows that after alignment the model succeeds at detecting named entities in Arabic which it was never trained on. It detects the class persons with precision of 68% compared to 88% in English and it's also its top scoring class.

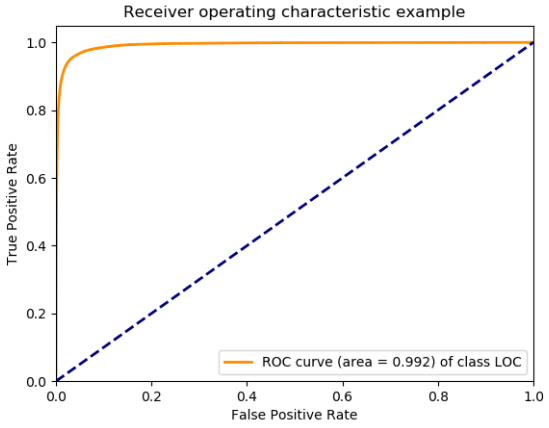


Figure 11: ROC curve for LOC class in English

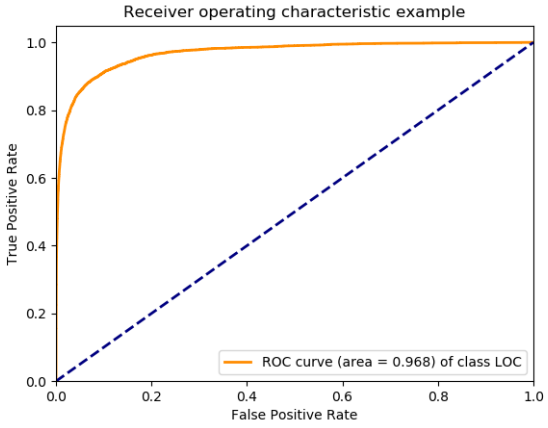


Figure 12: ROC curve for LOC class in Arabic

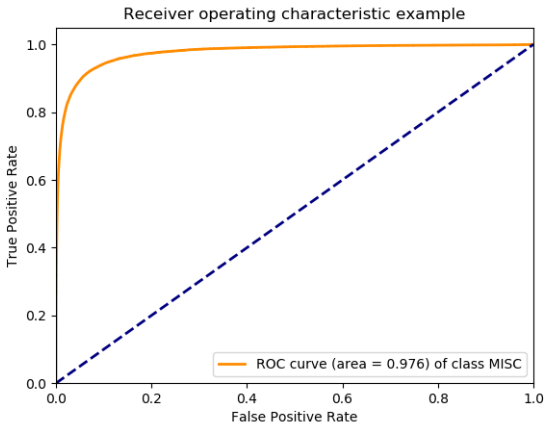


Figure 13: ROC curve for MISC class in English

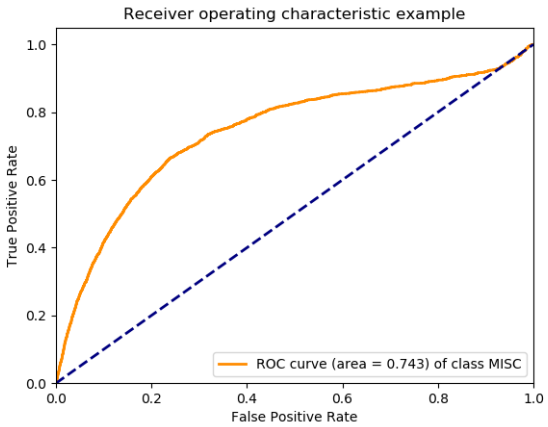


Figure 14: ROC curve for MISC class in Arabic

ROC figures and their corresponding AUC in Figures 11, 12, 13, 14, 15, 16, 17, 18 show that the proposed approach performs very well compared to random

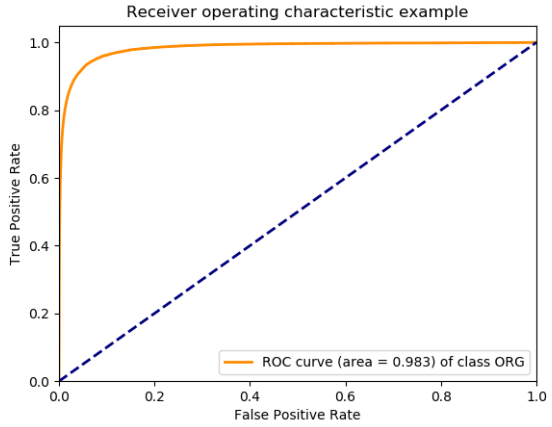


Figure 15: ROC curve for ORG class in English

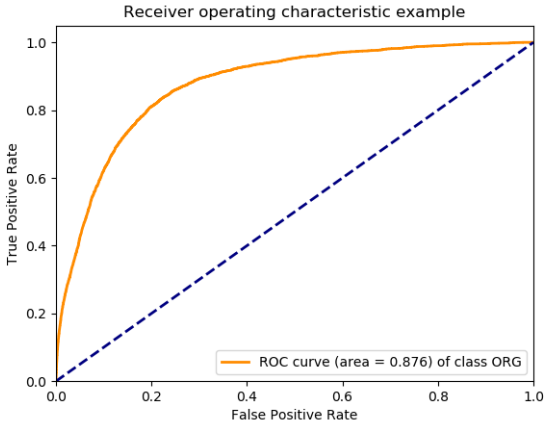


Figure 16: ROC curve for ORG class in Arabic

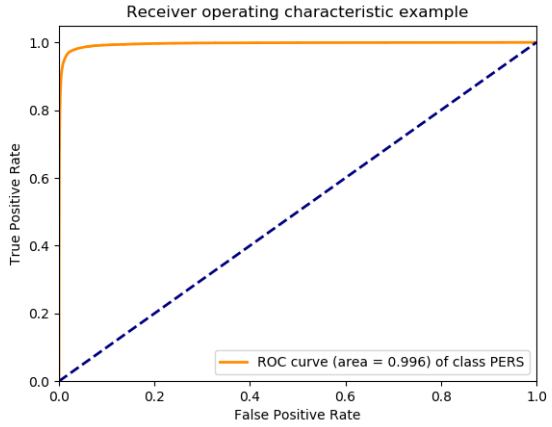


Figure 17: ROC curve for PERS class in English

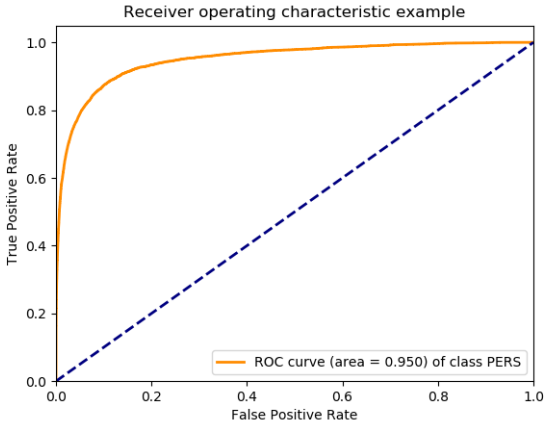


Figure 18: ROC curve for PERS class in Arabic

guessing and that the system is capable of classifying the aforementioned classes in both English and Arabic without the need for retraining the model on Arabic.

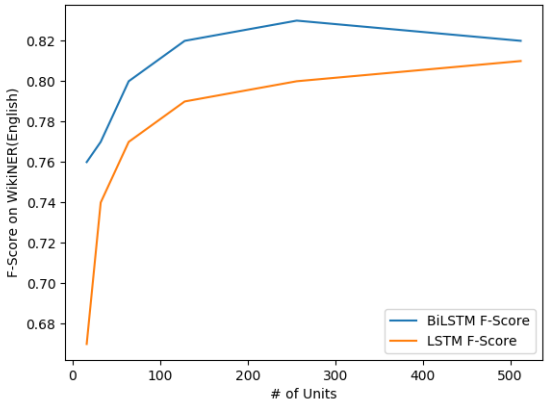


Figure 19: LSTM vs BiLSTM performance on English data

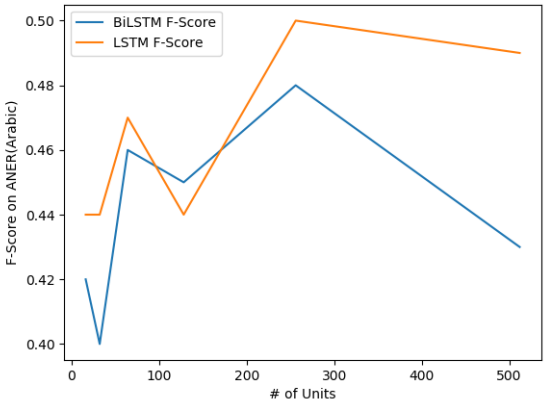


Figure 20: LSTM vs BiLSTM performance on aligned Arabic data

Hyper parameter selection for the model was performed empirically by iterating

over a finite set of parameters and choosing values that maximize our objective function (Grid search).

The main set of parameters to choose from were whether to use Bidirectional LSTM or Unidirectional LSTM and the number of units at the LSTM layer. These parameters were tested on both the English and Arabic models.

Figure 19 shows that BiLSTM performance is superior to LSTM performance on English data while it fluctuates on Arabic data according to Figure 20. It's also noted that the F-score values starts degrading just below 300 units. This makes the optimal choice for units to be around 256 units.

6. Conclusion

We exploit orthogonal transformation of word embeddings to create a language independent NER model that was trained on English and evaluated on Arabic without being explicitly exposed to Arabic before or fine tuned on an Arabic dataset. This paves the road to language independent NLP models that are capable of solving typical NLP tasks without being explicitly trained on a specific language.

References

[1] Jiafeng Guo, Gu Xu, Xueqi Cheng, and Hang Li. Named entity recognition in query. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 267–274. ACM, 2009.

[2] Diego Mollá, Menno Van Zaanen, Steve Cassidy, et al. Named entity recognition in question answering of speech data. 2007.

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

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

[5] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*, 2016.

[6] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

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

[8] Lance A. Ramshaw and Mitchell P. Marcus. Text chunking using transformation-based learning. *CoRR*, cmp-lg/9505040, 1995.

[9] Michal Ziemski, Marcin Junczys-Dowmunt, and Bruno Pouliquen. The united nations parallel corpus v1. 0. In *LREC*, 2016.

[10] Kareem Darwish. Named entity recognition using cross-lingual resources: Arabic as an example. In *ACL (1)*, pages 1558–1567, 2013.

[11] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Chris Bizer. DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*, 2014.

[12] Zhilin Yang, Ruslan Salakhutdinov, and William Cohen. Multi-task cross-lingual sequence tagging from scratch. *arXiv preprint arXiv:1603.06270*, 2016.

[13] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[14] Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

[15] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[16] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[17] Samuel L Smith, David HP Turban, Steven Hamblin, and Nils Y Hammerla. Offline bilingual word vectors, orthogonal transformations and the inverted softmax. *arXiv preprint arXiv:1702.03859*, 2017.

[18] Joel Nothman, Nicky Ringland, Will Radford, Tara Murphy, and James R. Curran. Learning multilingual named entity recognition from Wikipedia. *Artificial Intelligence*, 194:151–175, 2012.

[19] Chao Xing, Dong Wang, Chao Liu, and Yiye Lin. Normalized word embedding and orthogonal transform for bilingual word translation. In *HLT-NAACL*, pages 1006–1011, 2015.

[20] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[21] Alex Graves, Navdeep Jaitly, and Abdel rahman Mohamed. Hybrid speech recognition with deep bidirectional lstm. *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 273–278, 2013.

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

[23] François Chollet et al. Keras. https://github.com/fchollet/keras, 2015.