Automated Custom Named Entity Recognition and Disambiguation

LEVON STEPANYAN
Akian College of Science and Engineering
American University of Armenia
40 Marshal Baghramyan Ave, Yerevan 0019
REPUBLIC OF ARMENIA
levon_stepanyan@edu.aua.am

Abstract: - Named Entity Recognition (NER) and Disambiguation are sub-tasks in Natural Language Processing (NLP) that seek to identify and classify named entities in the text into their designated categories. With recent advancements in Deep Learning it is possible to use attention mechanisms and recurrent networks in order to produce reliable NER predictions. The use of NER ranges from profanity detection to extracting meta-data from documents. However, the greatest shortcoming of the classical NER models is the limited number of predefined classes that are set in the task (i.e. Person (PER), Location (LOC), Companies/institutions (ORG) etc.). With this limitation in mind we proposed a novel fast approach (FastEnt) to tackle the task of identifying and detecting Custom Named Entities (CNE) that are not limited to definition.
The task was split into 2 parts, where we initially create a basis space of words using several examples of the entity we are trying to identify, by using search across the word representation found through FasText and Word2Vec. We further complete automated online scraping from several sources such as Reddit in order to obtain an annotated corpus that will be used in the modeling step.
After producing the Annotated corpus with the designated CNE we train a dilated convolutional neural network with recurrent mechanisms to complete NER on this new entity. We test our findings on classic NE's mentioned above and are able to reliably reproduce the State-of-the-art (SOTA) results and further show consistent results with this approach on several custom named entity tasks.

Key-Words: - NLP, NER, Neural Networks, CRF, Parallelization, databases, API.

1 Introduction

Before starting the background of Named Entity Recognition (NER) we need to introduce several concepts for further use. Firstly, the notion of ***named entity*** must be explicitly defined.

Definition 1: A named entity is a term for which one or many strings, such as words or phrases, stands (fairly) consistently for some referent.

This definition is closely associated to the concept of the rigid designator introduced by Kripke and Saul [1].

Definition 2: A rigid designator known as the absolute substantial term is a type of a term that designated a concept uniquely in the field of its existence while not identifying anything else outside of that field in the process.

The task of Named Entity Recognition involves identifying and pointing out the strings that fall into some named predetermined entity class(es), in the text. Although, attempts were made to create fixed rigid designators as entities for NER, in common practice one must deal with numerous referents that cannot be considered philosophically "rigid". This can be clearly shown by discussing the following example:

It was an interesting time for
the Ford Motor Company.

Here we can easily recognize that the string ***"Ford Motor Company"*** refers to the organization, yet we must not overlook the fact that the word ***"Ford"*** can easily refer to many entities.

2 Problem Formulation

The named entity recognition task can be conceptually separated into two problems: detection of probable strings of named entities and the classification of the detected strings by the type of entity they refer to.

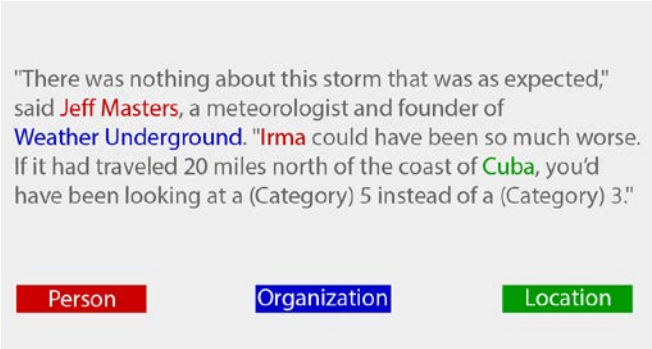

Fig.1. An example of classification by type.

The first part of the problem can be described as a segmentation task, where the model must try to identify the location of probable entities from the contiguous spans of tokens. The second phase of the problem requires choosing an ontology in order to be able to classify the segmented strings.
The results of these two tasks are evaluated using several metrics.

- Precision: Number of predicted entity string spans that line up **exactly** with spans in the evaluation data.
  $$precision = \frac{|\{releveant\ entitie\} \cap \{retrieved\ entities\}|}{|\{retrieved\ entities\}|}$$
- Recall: Number of names in the evaluation data that appear at exactly the same location in the predictions.
  $$recall = \frac{|\{releveant\ entitie\} \cap \{retrieved\ entities\}|}{|\{releveant\ entities\}|}$$
- F1: This metric is the harmonic mean of Precision and Recall.
  $$F_1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

2.1 Existing Solutions

Several common approaches have been used in order to implement NER systems. One set of solutions relies heavily on grammar-based rules and hand-crafted features, during the processes of segmentation and classification. The precision of this types of models are generally higher than compared to other approaches, however it must be noted that models lack significantly in recall and require months of work of experienced computational linguists [2].

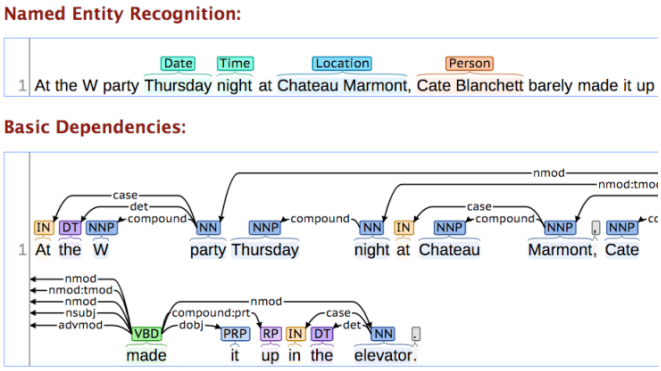

Fig.2. Basic dependencies in entity recognition

Usually NER is tackled using statistical and machine learning approaches. A frequent shortcoming of this kind of models is the requirement of a large hand annotated training and testing datasets. Some methods offer semi-supervised annotation modules in order to overcome this issue. The most prominent implementations of NER systems include a variation or an ensemble of the following methods:

1. Hidden Markov Models
2. Conditional Random Fields
3. Neural Networks (Convolution and Recurrence included)

All of these methods show significant promise and good results on the task. Each of the methods will be discussed in Methodology section. Even in the early implementations of NER systems such as *Zhou et al.* [3] we can see that the system based on Markov models achieves solid precision and recall on the predefined classes of CoNLL dataset [4]. With the advancement of hardware, more computationally intensive methods came into play. The efforts of Ling et al [5] and Lee, C. (2017) [6] signify the trend the current solutions are moving towards along with their results.

Eventually mixing the aforementioned techniques results in master systems such as *Explosion's Spacy*, *Stanford's CoreNLP* and several others.

2.2 Current Challenges

One of the most limiting factors of NER is the fixed number of classes of entities that the current systems use. Indeed, the commonly used CoNLL entity tags do not in any way cover the vast majority of probable entities that a user might be interested in segmenting. However, developing a corpora for custom named entity recognition is also a cumbersome task requiring an annotated dataset as mentioned in previous sub-section, thus effectively confining the researcher to hard and unnecessary

labor during the project. A probable solution will be offered to each of these problems in this article.

3 Methodology

3.1 Hidden Markov Models

The Hidden Markov Model is one of the most important machine learning models in speech and language processing. Hidden Markov Models (HMM) are doubly embedded stochastic processes, utilized for modelling diverse situations that are characterized by evolution of some events that depend on some internal factors. These internal factors are more commonly referred to as states, while the events are called observations. To grasp an intuition behind HMMs we can think of it as a closed system, that has n states, which is required to reside in on one of the states at a given fixed point in time and can also make transitions between those *n* states with some predetermined probability while emitting observations with some other predetermined probability set [7].

Definition 3: A model $\sigma(A, B, \pi, n, m)$ is called a Hidden Markov Model if:

1. *n* - signifies the number of states.
2. *m* - signifies the number of possible emissions.
3. π - is a vector, where an element π_i signifies the probability of being at state *i* during the time step 1.
4. *A* - is the transition matrix, where $A_{i,j}$ signifies the probability of transferring from state *i* to *j*.
5. *B* - is the emission matrix, where $B_{i,j}$ signifies the probability of emitting the symbol *j* while in state *i*.
6. $P(q_{t+1} = S_j | q_t = S_i) = P(q_{t+1} = S_j | q_t = S_i, q_{t-1} = S_k, q_{t-2} = S_l, \dots)$

where S_k are the states and q_t in the state at time t.

It must be noted that this definition asserts that only the present state matters when predicting the future states. In the context of named entity recognition our aim is to find an optimal tag sequence $T_1^n = t_1, t_2, \dots, t_n$ for the given token sequence $G_1^n = g_1, g_2, \dots, g_n$ that maximizes:

$$\log P(T_1^n | G_1^n) = \log P(T_1^n) + \log \frac{P(T_1^n, G_1^n)}{P(T_1^n) \cdot P(G_1^n)} \quad (1)$$

In order to simplify the equation further we can assume mutual information independence.

$$\log MI(T_1^n, G_1^n) = \sum_{i=1}^{n} MI(t_i, G_1^n) \quad (2)$$

can be written as:

$$\log \frac{P(T_1^n, G_1^n)}{P(T_1^n) \cdot P(G_1^n)} = \sum_{i=1}^{n} \log \frac{P(t_i, G_1^n)}{P(t_i)} \quad (3)$$

Plugging ***equation 2*** into ***equation 1*** yields the following general equation that we use:

$$\log P(T_1^n | G_1^n) = \log P(T_1^n) - \sum_{i=1}^{n} \log P(t_i) + \sum_{i=1}^{n} \log P(t_i | G_1^n) \quad (4)$$

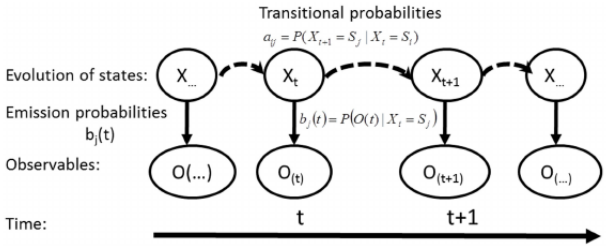

Fig.3. HMM model visualization - transition states

Hidden Markov Models conforming to the following rule completely or with slight variations were widely used in NER. However, as of today they are mostly used in combination with other techniques.

3.2 Conditional Random Fields

CRFs are a type of sequence modeling techniques that are used for structured prediction and have found their neat application inside NER. CRFs can be described as discriminative undirected probabilistic graph model used to encode known relations between the designated observation while constructing consistent interpretations.

A formal definition given by Lafferty, McCallum and Pereira [8] states that:

Definition 3: A CRF on observations *X* and random variables *Y* is defined as follows:

Let $G = (V, E)$ be a graph such that, $Y = (Y_v)_{v \in V}$, meaning that *Y* is indexed by the vertices of *G*.

Then (X, Y) is a conditional random field when the random variables Y_v, conditioned on *X*, obey the Markov property with respect to the graph: $p(Y_v | X\}, Y_w, w \neq v = p(Y_v | X\}, Y_w, w \sim v$, where $w \sim v$ means that *w* and *v* are neighbors in *G*.

Accordingly, we can conclude that CRF is an undirected graphical model whose nodes can be

explicitly divided into two disjoint sets X and Y and the conditional probability distribution $p(X|Y)$ can be modeled.

When applied to the NER task, CRFs share some common properties with HMMs. The tokens in the text can be conventionally labeled as the observation sequence while the named entities correspond to the tag sequence. We aim to model the conditional probability of a state sequence given the observation sequence. It can be mathematically described as follows:

$$P(S|O) = \frac{1}{Z_o} \exp\left(\sum_{t=1}^{T}\sum_{k} \lambda_k f_k(S_{l-1}, S_l, O, t)\right) \quad (5)$$

Where $f_k(S_{l-1}, S_l, O, t)$ is the feature function whose weight λ_k is tuned during the training process. We define the conditional probability of a label sequence based on total probability over the state sequences, i.e. $P(l|o) = \sum_{s:l(s)=l} P(s|o)$, where $l(s)$ is the sequence of labels corresponding to the states in the sequences. Z_0 is the normalization factor over all state sequences.

$$Z_o = \sum_{s} \exp\left(\sum_{t=1}^{T}\sum_{k} \lambda_k f_k(S_{l-1}, S_l, O, t)\right) \quad (6)$$

3.3 Neural Networks

Artificial Neural Networks are one of the most prominent tools for binary and multi-class classification problems. During several decades of development various architectures of NN arouse, such as Feedforward Neural Network (FNN - the basic ones), Recursive Neural Network (RNN) and Convolutional Neural Network (CNN). In this section we will overview the basics of the above structures. There are many useful and informative researches, which thoroughly elaborate on the aforementioned architectures [9], that is why this paper will be concise on the literature overview.

The most important concept to understand is a Deep Neural Network - this consists of several layers of artificial neurons which, in their turn, are basic computational nodes.

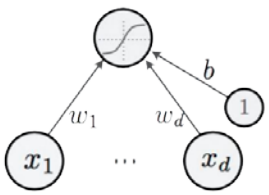

Fig.4. Single artificial neuron.

Each computational node (neuron) gets as an input a sequence of numbers x_i. Each x_i is multiplied of its corresponding weighting coefficient and a bias b is added to the sum of multiplications. This sum is given as an input to the neural node function, whose result is the final output of the neuron. There are several well-known choices for a neuron function, such as $tanh\ or\ sigmoid$. A single sigmoid neuron can't do much than to classify the data into two basic categories with a threshold. To create a more advanced classifier numerous neurons are combined - creating a layer, and several layers combined to form a classic FNN. A standard FNN consists of three categories of neural levels, i.e. input, hidden and output layers, as depicted below:

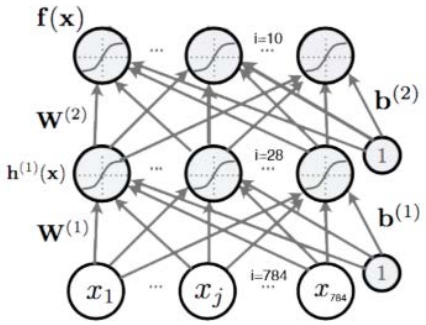

Fig.5. Multi-layer Neural Network

At the first place, the network is initialized with random weights and biases for each neural cell, which then are optimized in regards to the error function - thus trying to solve the classification problem.

The Convolutional NN is one step advanced in a sense that it has additional layers of convolution and pooling. Convolutional layers are different from ordinary neural layer in a way that each neuron will get as an input only a subset of the inputs. The basic concept of CNN would be to divide the neurons into subgroups, forming feature map - each of the subgroup will optimize itself in recognizing a specific "feature" in the data. Finally, pooling layers are used for filtering out the useless feature map subgroups, thus trying to get rid of useless "features" for our classifier.

4 Problem Solution

The design of the system created for end-to-end Custom Named Entity Recognition can effectively be segmented into three sub-modules: ***Dataset Generation, Automatic Annotation and Model Training***. Let us dive into each of these components separately.

4.1 Dataset Generation

One of the main issues during the creation of a new custom entity is the lack of understanding of what the entity must represent and what ideas and concepts it must cover. The viewpoint that we maintain is that any entity that can be described with a list of words can be completed using their synonyms. This intuition is akin to the concept of space spanning vectors in linear algebra. The descriptor words effectively try to span the concept behind the entity we are trying to construct, meaning that by creating a dataset comprised of the descriptors and their synonyms, we aim to create a complete space of ideas and words relevant to that entity. For this very purpose we use Word Embeddings in order to obtain words similar to the descriptors and use pre-processing techniques to get rid of irrelevant strings that made their way through to the initial dataset.

4.2 Word Embeddings

Word embeddings are the collective set of techniques that try to represent a single word string as a vector of some predefined dimension, thus creating a string to vector mapping for each of the words we are interested in. The most conventional methods of embedding today are the ***word2vec, GloVe, FastText and PoincareEmbeddings***.

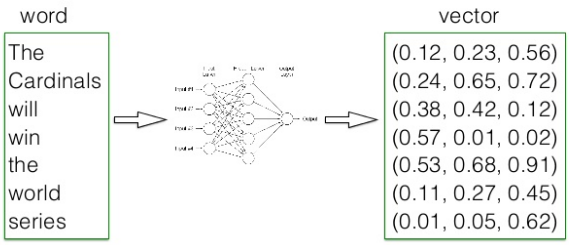

Fig.6. Word embedding transformation.

Originally created by Mikolov et al. 2013 [10], word2vec was the first neural embedding model that is intensely used by the researchers up till this day. There a are three different parameter learning types that it utilizes.

- One-word context - We are considering one word per one context. This approach is known as Continuous Bag Of Words (CBOW). The aim is to obtain an appropriate vector representation for the word.
- Multiple word context -Multiple context words are considered along the word itself. Here we consider the word in relation with other words in text. The aim is once again to obtain an appropriate vector representation for the word.
- Skip-gram - The reversed situation of multiple word context. Here we try to predict multiple context words given one word on the input.

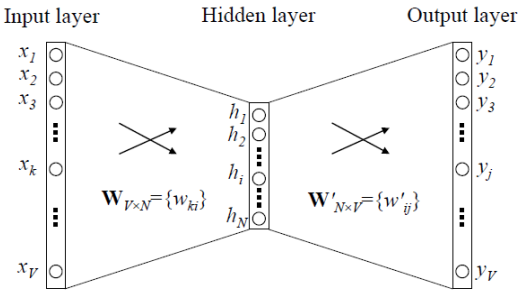

Fig.7. One Context Word

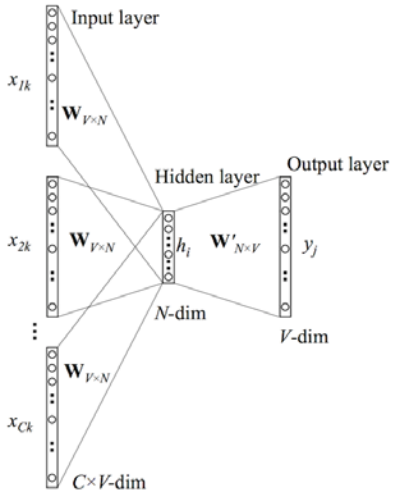

Fig.8. Multiple Context Words

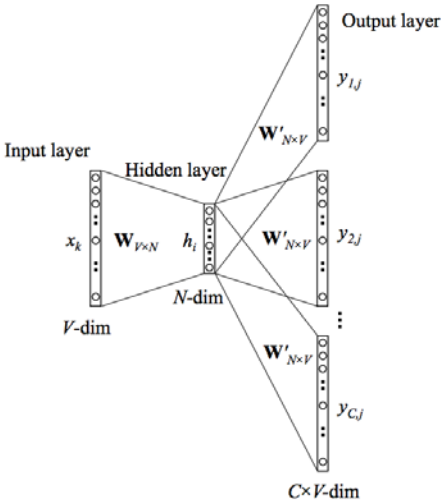

Fig.9. Skip-Gram

GloVe utilizes the structure of the whole observed corpus in order to capture the meaning of one word embedding. After training on global co-occurrence counts of words GloVe minimizes the least-squares error, consequently obtaining word

vectors with meaningful substructure. The design sufficiently preserves words similarities with vector distance.

The FastText model takes is quite alike word2vec, yet it adds new semantic features into the mix. It also considers the internal structure of words by splitting them into a bag of character n-grams and adding to them a whole word as a final feature. The complete set of operations is defined at Bojanowski et al [11].

The most recent advancement is Poincare's Embeddings that uses hyperbolic geometry for capturing hierarchical properties of the words that are hard to capture in Euclidean space. The use of hyperbolic geometry along with the Poincare ball allows to obtain and represent an interesting property that the distance from the root of the tree to its leaves grows exponentially with the addition of each new child. The whole optimization process with in-depth derivations and definitions can be found in the paper Nickel et al [12].

Using the aforementioned embeddings, we are able to obtain representation and compute similar words from those representations, which is the goal that we initially had in mind.

4.3 Preprocessing

It must be noted that even after utilizing word embeddings to get a dataset of similar words, we are still bound to have irrelevant words in that dataset. In order to get rid of them we must filter out the extraneous and unrelated words. The task has to be automated in order to avoid the time constraints and cost of human labor.

In order to complete this we can start by looking into the semantic structure and linguistic features of the words. A very simple example of preprocessing would be pseudo-deduplication, meaning that words that are almost semantically identical with only a slight set of differences i.e. "Erik", "Eric" will be removed. We can also look into the Part-Of-Speech (POS) tags of each word and determine the relevance of the word, for example knowing that usually adjectives are not a common part of the NER task we can delete them. The link to the documentation for the complete set of preprocessing tools is given in the next chapter.

4.4 Annotation

As previously mentioned, annotation is a very costly task that requires either long hours of human labor, or some supervised approaches to complete pseudo-annotation. In the system created in this research we aim to have a complete annotator system that is language independent and thoroughly deterministic.

4.4.1 Contextualization

After generating the raw word dataset of words as described in chapter 3, we have to put those words in several contexts in order to be able to get a trainable dataset for our further steps. We utilize several APIs (Reddit, Twitter, news etc.) in order to search for relevant sentences and paragraphs inside the comments, topics and texts within these networks. We try to construct optimized queries with adequate amount of filtering in order to retrieve the required amount of necessary content.

After this, the task of annotation shrinks down to finding the position of relevant designated sub-strings in the derived context.

4.4.2 Parallelization

It must be noted that although the queries to the APIs are optimized, completing every request iteratively one after another might take a significant amount of time and take a heavy toll on the CPU. To overcome this issue, we decided to implement a parallel computing scheme for the contextualization routine.

Idealistically, the speedup from parallelization must be linear, meaning that doubling the number of processing cores must split the time of computation in half. Yet, very few parallel algorithms achieve optimal speedup. Most of them have a near-linear speedup for small numbers of processing elements, decaying swiftly as the number of processing elements increases. The potential speedup of an algorithm after parallelization is given by Amdahl's law:

$$S_{latency}(s) = \frac{1}{1 - p + \frac{p}{s}} \quad (7)$$

- S - the speedup in latency of execution of the whole task
- s - is the speedup in latency of the execution of the parallelizable part of the task
- p -is the percentage of the execution time of the whole task concerning the parallelizable part of the task before parallelization.

Since $S_{latency} < \frac{1}{1-p}$, it shows that a small part of the program which cannot be parallelized will limit the overall speedup available from parallelization.

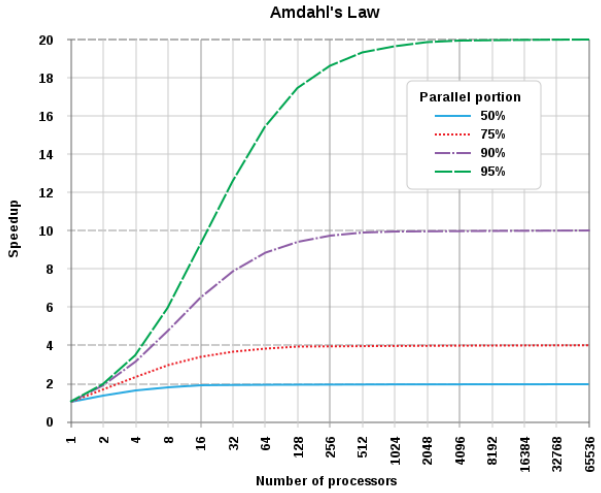

Fig.10. Visualization of Amdahl's Law

We make use of asynchronous threads that work concurrently in order to complete the process of contextualization in parallel.

4.4.3 CouchDB

In order to make the collected data editable and reusable we chose to create a structured database for each of the entities that we aim to create. After careful consideration CouchDB was chosen as the database system for storing the generated structures. Apache CouchDB is easy to use and has a vast focus on scalable architecture. It has a document-oriented NoSQL database architecture. JSON structures are used to store the data, JavaScript is used as the primary query language and MapReduce, and HTTP are utilized for an API. We must note significant difference between CouchDB and relational database structures. CouchDB does not store data and relationships in tables. Instead, each database is a collection of independent documents. Each document maintains its own data and self-contained schema.

In order to dive further in the discussion of CouchDB let define some key concepts.

Definition 4: Multi-version concurrency control, is a concurrency control method commonly used by database management systems to provide concurrent access to the database and in programming languages to implement transactional memory.

Definition 5: Eventual consistency is a consistency model used in distributed computing to achieve high availability that informally guarantees that, if no new updates are made to a given data item, eventually all accesses to that item will return the last updated value.

It is important to note that CouchDB implements both of these concepts, effectively allowing us to handle a high volume of concurrent reads and writes without conflict. A more in-depth analysis of the system can be found at Han et al [13].

4.5 Training the Model

After obtaining the contextualized and annotated data the most logical last step is to train a model for recognizing the designated entity. We make us of the *Spacy* pipeline in order train a Neural Network with iterated Dilated convolutions. The architecture and optimization of the network is fairly similar to the one described by Strubell et al [14].

4.6 Experiments

As a benchmark we choose to reconstruct the results obtained on classic Entity types PER and LOC. The dataset constructed for the task along with the full annotation set was able to reproduce the SOTA results obtained by Strubell et al [14] with marginal differences across several runs, thus proving the feasibility of using the dataset generation and annotation module for common entities.

In order to expand the experiment into custom entity types we choose to scrape the a set of dictionaries for entities that conformed to drugs (i.e. cocaine, heroine) and constructed an annotated test set for the task. During the dataset generation step we decided to use only 2 examples of the type for enriching and obtaining the full space that covered the entity. We further created an annotated training set that was completely independent of the testing set. After the completion the NER model was trained obtaining 85% on that unknown entity type. We further validated that the dataset enrichment module was able to produce a set of words/sentences that overlapped with the set of unique testing words that spanned the space at 93%. This proof is sufficient to show that the method performed incrementally well on this unknown task with sufficient results.

5 Conclusion

During the course of research an end-to-end framework for Custom Named Entity Recognition was developed. The system is named "Fastent" and can be found on GitHub or its very own website. An elaborate documentation was derived describing the processes of installation and the use of each of the sub-modules. The system has been tested on some

common NER tasks and the baselines should be reported in the documentation after the full scale of testing is complete. It must be noted that the modules in the system are *mostly* (only dataset generation requires explicit word vectors, which are initially supplied to be for English) language independent. Although a thorough routine was derived for complete custom NER, the system can be improved by the addition of Bidirectional LSTM models and more model training routines.

References:

[1] Kripke, S., "Identity and necessity.", *Perspectives in the Philosophy of Language*, 1971, pp. 93-126.

[2] Kapetanios E, Tatar D, Sacarea C., *Natural language processing: semantic aspects.* CRC Press, 2013.

[3] Zhou, GuoDong, and Jian Su. "Named entity recognition using an HMM-based chunk tagger." *In proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, 2002, pp. 473-480.

[4] Sang, E. F., & De Meulder, F., Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. 2003, arXiv preprint cs/0306050.

[5] Ling, W., Luís, T., Marujo, L., Astudillo, R.F., Amir, S., Dyer, C., Black, A.W. and Trancoso, I., Finding function in form: Compositional character models for open vocabulary word representation. 2015, arXiv preprint: 1508.02096.

[6] Lee, C., "LSTM-CRF models for named entity recognition.", *IEICE Transactions on Information and Systems*, 100(4), 2017, pp.882-887.

[7] Rabiner, Lawrence R. "A tutorial on hidden Markov models and selected applications in speech recognition." *Proceedings of the IEEE*, Vol 77, No. 2, 1989, pp. 257-286.

[8] Lafferty, J., McCallum, A., & Pereira, F. C., Conditional random fields: Probabilistic models for segmenting and labeling sequence data 2001.

[9] Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., & Dyer, C., Neural architectures for named entity recognition. 2016, arXiv preprint arXiv:1603.01360.

[10] Mikolov, T., Chen, K., Corrado, G., & Dean, J., Efficient estimation of word representations in vector space., 2013, preprint arXiv:1301.3781.

[11] Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T., Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics, No. 5, 2017, pp. 135-146.

[12] Nickel, M., & Kiela, D., Poincaré embeddings for learning hierarchical representations. *In Advances in neural information processing systems*, 2017, pp. 6338-6347.

[13] Han, J., Haihong, E., Le, G., & Du, J. Survey on NoSQL database in Pervasive computing and applications (ICPCA), 2011, pp. 363-366.

[14] Strubell, E., Verga, P., Belanger, D., & McCallum, A., Fast and accurate entity recognition with iterated dilated convolutions. 2017, preprint arXiv:1702.02098.