Department of Informatics
MSc in Data Science

Master Thesis

Named Entity Recognition (NER) and Graph Representation of job descriptions

Spantouri Natalia

Academic Supervisor: Prodromos Malakasiotis
Company Supervisor: Konstantinos Rapantzikos

Athens, October 2019

Abstract

In this thesis, we dealt with the problem of Named Entity Recognition for job descriptions and implemented two different models, Bi-LSTMs with a CRF layer (Flair framework) and BERT-base. We experimented on a dataset containing descriptions from job advertisements (ads) and their entities were labeled with six different category types: *clean title, technical skills, soft skills, language, field of study* and *education level*. In our results, the Flair framework surpassed the results of BERT-base.

In addition, we found a way to represent and connect job advertisements using a graph. In the proposed scheme, the job ads and all the possible entities, which can be found in an ad, are represented as nodes. Edges are added between ads and their entities. Using this representation and Node2Vec we extracted node embeddings and performed several queries to examine if the proposed graph scheme can be used to retrieve useful information. Indeed, we were able to extract similar advertisements, which were otherwise isolated.

Περίληψη

Στην παρούσα διπλωματική εργασία ασχοληθήκαμε με την δημιουργία ενός συστήματος αναγνώρισης ονομάτων οντοτήτων για αγγελίες εργασίας και πειραματιστήκαμε με δύο διαφορετικά μοντέλα, Bi-LTSMs με CRF (Flair framework) και BERT-base. Τα δεδομένα που χρησιμοποιήθηκαν ήταν αγγελίες εργασίας και οι οντότητες τους είχαν επισημειωθεί με έξι τύπους κατηγοριών: *τίτλος επαγγέματος, τεχνικές δεξιότητες, κοινωνικές δεξιότητες, γλώσσα, τομέας σπουδών* και *επίπεδο σπουδών*. Στα αποτελέσματα των πειραμάτων μας, το μοντέλο που χρησιμοποιεί το Flair framework είχε καλύτερη επίδοση από το BERT-base.

Επιπλέον, βρήκαμε ένα τρόπο αναπαράστασης και ένωσης των αγγελιών με τη χρήση γράφου. Στο προτεινόμενο σχήμα, οι αγγελίες εργασίας και όλες οι πιθανές οντότητες, που μπορούν να βρεθούν σε μια αγγελία, αναπαρίστανται με κόμβους. Ακμές προστίθενται μεταξύ των αγγελιών και των οντοτήτων. Χρησιμοποιώντας αυτή την αναπαράσταση και Node2Vec πήραμε αναπαραστάσεις των κόμβων και εκτελέσαμε διάφορα ερωτήματα για να εξετάσουμε αν το προτεινόμενο σχήμα γραφήματος μπορεί να χρησιμοποιηθεί για την ανάκτηση χρήσιμων πληροφοριών. Πράγματι, καταφέραμε να εξάγουμε κοινές αγγελίες, που αρχικά φαινόντουσαν διαφορετικές μεταξύ τους.

Acknowledgements

I would like to thank my supervisor, Prodromos Malakasiotis, for his valuable guidance and mentoring throughtout the duration of my thesis.

In addition, I would like to thank my company supervisors, Konstantinos Rapantzikos and Evangelos Anagnostopoulos for giving me the oportunity to explore an area of great interest.

Last but not least, I would like to thank my family and all the people close to me for their support and encouragement. Especially, Giannis Spantouris, for his support and patience.

Contents

Chapter 1

Introduction

Named Entity Recognition (NER) is a subtask of information extraction that seeks to locate named entities in texts and classify them in pre-defined categories such as persons, organizations, locations etc. NER is a valuable component for extracting information from text and can be used in various applications.

In the field of job recruiting, companies seek to find automated ways that can help them find candidates for their positions and can facilitate the procedure. Consequently, there is an increased need for automated systems that are able to classify and extract from job descriptions entities that the advertisements require (can be considered as the properties of the ad). Examples of possible entity types are: clean title (e.g. Data Analyst), technical skill (e.g. Microsoft Word), soft skill (e.g. decision making), language that a job advertisement require (e.g. English), field of study (e.g. Computer Science), education level (e.g. Master's Degree). NER can be used for extracting these entities and can simplify the procedure by using the results in combination with other models.

Moreover, a simple way to connect the individual entities with each other is with the use of a graph. A graph captures entities, attributes, and relationships between nodes. It is a way to model the information in order to represent a connected version of the same information that is otherwise isolated, distributed and disorganized. With this way all extracted information is connected together in order to identify meaningful pieces of information and relate to each other. Also, a graph excels at managing highly-connected data and complex queries and it can collect and organize data from multiple data sources.

Therefore, in combination with the NER model, it can be used to represent the entities, connect the advertisements and find similarities.

1.1 The purpose of the thesis

The purpose of this thesis is to study, implement and evaluate a Named Entity Recognition model for job descriptions in order to predict the following category types: *clean title, soft skill, technical skill, language, education level* and *field of study*. We use two different methods, Flair framework[1] and BERT-base[2] and compare their results.

Additionally, we use the job descriptions and the annotated entities from the dataset in order to create a graph and produce node embeddings using Node2Vec. The entities can be considered as the properties of the job ads. Our goal is to experiment with the node embeddings in order to see whether the graph can be used for running queries to retrieve similar job advertisements. More importantly, the graph can be enriched with nodes representing candidates which will be connected with the corresponding entities in a similar manner as the job ads. This will allow for performing queries to match ads with candidate employees.

[1] See https://github.com/zalandoresearch/flair

[2] See https://github.com/google-research/bert

1.2 Outline

The rest of the thesis is organized as follows:

- Chapter 2: Discussion about related work.
- Chapter 3: Description of the methods for NER, as well as the experiments conducted.
- Chapter 4: Description of the creation of the Graph and presentation of the experimental results.
- Chapter 5: Conclusions about the findings and proposals for future work.

Chapter 2

Related Work

Named Entity Recognition is a problem that has been widely studied in recent years and a considerable amount of literature has been published about the subject.

Huang et al. (2015) experimented with a variety of LSTM models and were the first to experiment with a Bi-LSTM-CRF model proving that could provide state of the art accuracy for NER models. They showed that the Bi-LSTM-CRF model can efficiently use both past and future input features due to the bidirectional LSTMs and that thanks to the CRF layer, it could use sentence level tag information.

Peters et al. (2017) proposed a general semi-supervised approach for adding pre-trained context embeddings from bidirectional language models and using them for tagging. They explored a semi-supervised approach that did not require additional labeled data and proved that context sensitive representation, captured by the language model embeddings, could be useful for tagging and could produce state of the art results.

Clark et al. (2018) proposed Cross-View Training (CVT), a semi-supervised learning algorithm that improves the representations of a Bi-LSTM sentence encoder using a mix of labeled and unlabeled data. The model for labeled and unlabeled data shared intermediate representations, which improved the whole model and achieved state of the art results.

Baevski et al. (2019) presented a new way for pre-training a bidirectional transformer model. The model solves a cloze-style word reconstruction task, where it must predict the center word given left-to-right and right-to-left context representations. This method provides significant performance gains in NER applications.

Chapter 3

Named Entity Recognition

3.1 Flair framework

Flair framework (Akbik et al. 2019a) is a simple framework containing state-of-the-art Natural Language Processing (NLP) models. It has simple interfaces that can be used to combine different embeddings, including their Flair embeddings, as well as BERT embeddings. Moreover, to facilitate the users it implements standard model training and hyper-parameter selection routines using the hyperopt library.[1]

3.1.1 Model

The NER model of the framework consists of Bi-LSTM with a CRF layer. Long Short Term Memory networks (LSTM) are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter et al. (1997), and were designed so as to resolve the vanishing gradients problem. The LSTM consists of a cell (the memory part of the LSTM unit) and three gates: an input gate, an output gate and a forget gate. The cell is responsible for keeping track of the dependencies between the elements in the input sequence. The input gate controls the extent to which a new value flows into the cell. The forget gate controls the extent to which a value remains in the cell. Finally, the output gate controls the extent to which the value in the cell is used to compute the output activation of the LSTM unit.

The LSTM block is described by the following set of equations, where σ stands for a sigmoid activation:

$$\text{Forget Gate: } f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \tag{3.1}$$

$$\text{Input Gate: } i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \tag{3.2}$$

$$\text{Output Gate: } o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \tag{3.3}$$

$$\text{Cell: } c_t = f_t \circ c_{t-1} + i_t \circ tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{3.4}$$

$$\text{Hidden State: } h_t = o_t \circ tanh(c_t) \tag{3.5}$$

As we want to have access to both past and future input features for a given time, the model uses a bidirectional LSTM network, which makes two passes over the sequence, one from left to right and one from right to left and we concatenate the result of the forward and the backward LSTM.

The Bi-LSTMs make predictions at token level, which means that their predictions ignore the neighboring tokens when making decisions. For this reason, Flair framework uses a Conditional Random Field (CRF) layer (Lafferty 2001), which observes the entire output in order to make predictions, making it possible to condition the prediction on neighboring words.

[1] See https://github.com/hyperopt/hyperopt

Without a CRF, a single linear layer would transform the output of the Bidirectional LSTM into scores for each tag. These are called emission scores and they represent the likelihood of the word being a certain tag. A CRF calculates also the transition scores, which are the likelihood of a word to be assigned with a certain tag considering that the previous word was assigned with as certain tag. Therefore, the transition scores measure the likelihood to transition from one tag to another. Figure 3.1 shows a simple representation of the network.

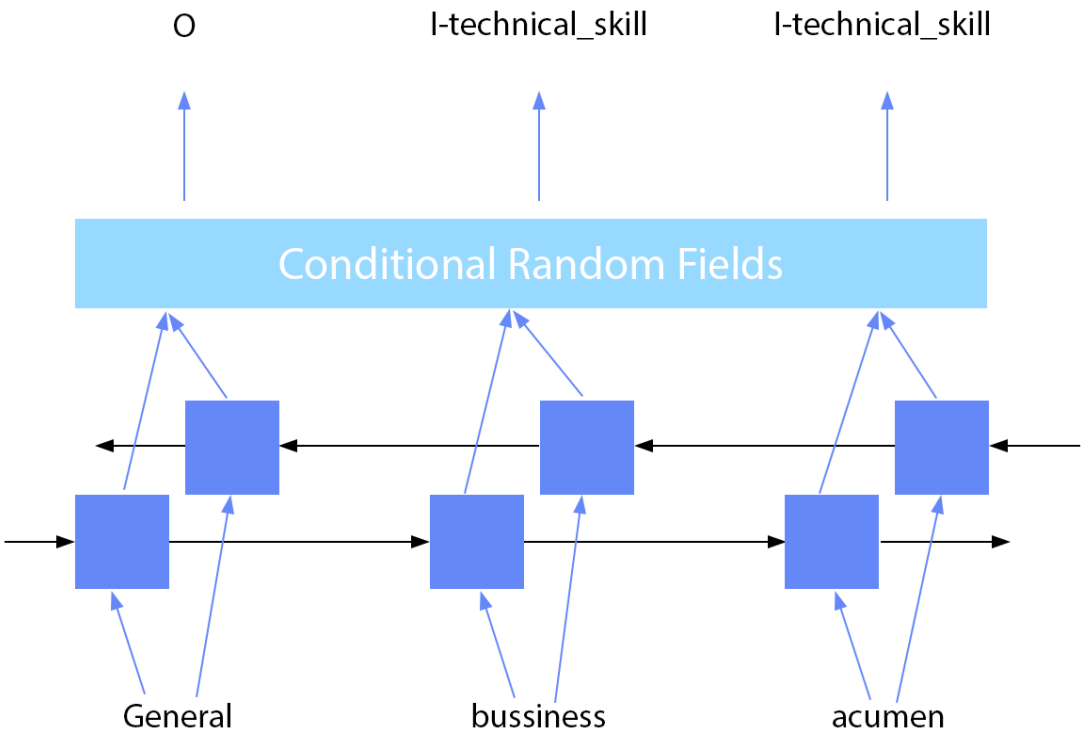

Figure 3.1: A Bi-LSTM with CRF network.

3.1.2 Pooled Flair Embeddings

The creators of the Flair framework, proposed the use of pooled contextualized embeddings (Akbik et al. 2019b). Contextual string embeddings are powerful embeddings that capture latent syntactic and semantic information that goes beyond standard word embeddings. The contextual embeddings are trained without any explicit notion of words and so basically they model words as sequences of characters and are contextualized by their surrounding text, meaning that the same word will have different embeddings depending on its contextual use. However, they suffer once they encounter rare words in an underspecified context. Thus, Akbik et al. (2019b) proposed a method in which they dynamically aggregate contextualized embeddings for each unique sequence of characters that they encounter as they process a dataset and then use a pooling operation to distill a global word representation from all contextualized instances in combination with the current contextualized representation as a new word embedding. Figure 3.2 shows an example of the above process.

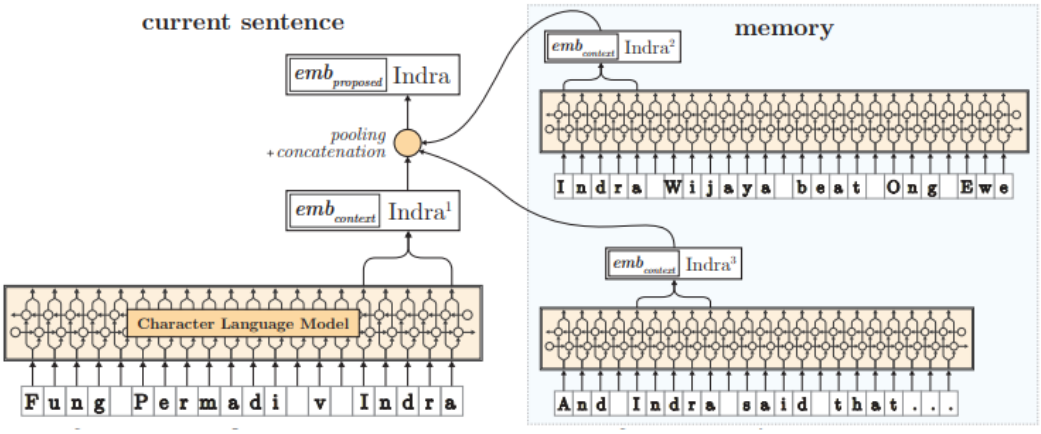

Figure 3.2: Example of how the embedding (*emb_proposed*) for the word "Indra" was generated in the example text segment "Fung Permadi v Indra". A contextual string embedding (*emb_context*) for this word was extracted and from the memory all embeddings that were produced for this string on previous sentences were retrieved. Finally, all local contextualized embeddings were pooled and concatenated to produce the final embedding.

3.2 BERT

BERT stands for Bidirectional Encoder Representations from Transformers (Devlin et al. 2018). BERT uses stacked Transformer encoders to turn each sequence of input embeddings to a sequence of context aware embeddings. Figure 3.3 shows the architecture of BERT. A word starts with its embedding representation (E1) from the embedding layer. Every layer performs a multi-headed attention computation on the word representation of the previous layer to create a new intermediate representation. T1 is the final output and Trm are the intermediate representations of the same token. The BERT-base model, which we use in this thesis, has 12 layers (transformer blocks), hidden size 768, 12 self-attention heads, and 110 million parameters.

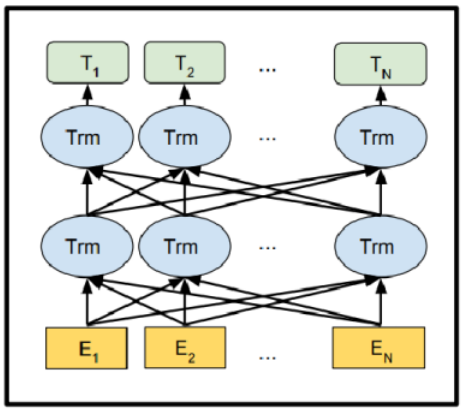

Figure 3.3: BERT Architecture.

As input, BERT takes the sum of token embeddings, segment and position embeddings. For BERT, sentence can be an arbitrary span of continuous text, rather than an actual linguistic sentence. A sequence refers to the input token sequence to BERT, which may be a single sentence or two sentences packed together (A and B). The positional embeddings represent the position of words in a sequence. BERT also takes segment embeddings as input. BERT can be trained on sentence pairs for tasks that take sentence pairs as input (e.g. question answering). It learns a unique embedding for A and B sentences to help the model distinguish between them. In our case, as we do not have a sentence pair task the segment embeddings correspond only to sentence A. The maximum sequence length of the input is 512 tokens. The input schema for BERT is shown in Figure 3.4.

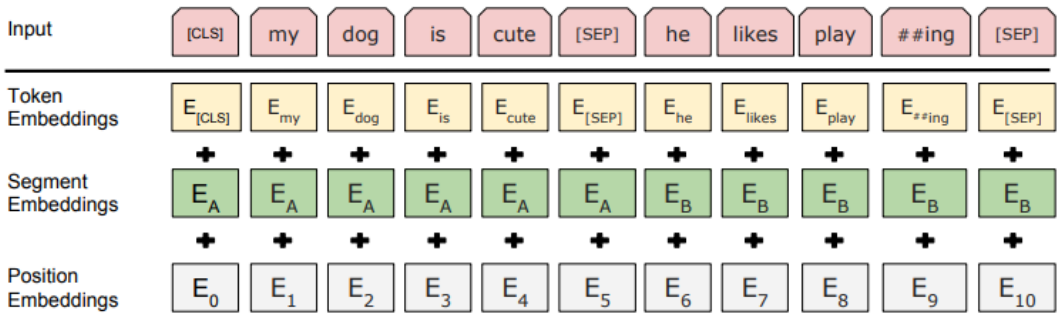

Figure 3.4: BERT Input Representation.

BERT has been pre-trained on Masked Language Modeling and Next Sentence Prediction. Language Modeling is the task of predicting the next word given a sequence of words. In masked language modeling instead of predicting every next token, a percentage of input tokens is masked at random and only those masked tokens are predicted. Next sentence prediction task is a binary classification task in which, given a pair of sentences, it is predicted if the second sentence is the actual next sentence of the first sentence.

BERT uses WordPiece tokenization. The vocabulary is initialized with all the individual characters in the language, and then the most frequent combinations of the existing words in the vocabulary are iteratively added. Any word that does not occur in the vocabulary is broken down into sub-words. For example, if “Ana”, “##ly”, and “##st” are present in the vocabulary but “Analyst” is out of the vocabulary then it will be broken down into “Ana” + “##ly” + “##st” respectively. (## is used to represent sub-words).

BERT depending on the task has to be fine-tuned. For each task, we plug the task specific inputs and outputs into BERT and fine-tune all the parameters. In NER, a tag must be predicted for every word in the input. The final hidden states (the transformer output) of every input token is fed to the classification layer to get a prediction for every token. Figure 3.5 shows fine tuning procedure of BERT for the NER task.

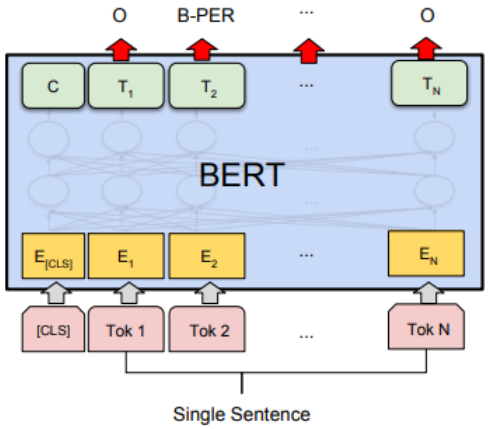

Figure 3.5: Illustaration of Fine-Tuning BERT for NER.

3.3 Dataset

The dataset that was used for the experiments consists of job descriptions. Specifically, it contains 3172 unique job advertisements (based on their id). From each job description, a sentence or a sequence of sentences have been randomly chosen and have been used for annotation. The number of annotated texts are 34493. There are six categories that an entity can be labeled as: *clean title, education level, field of study, soft skill, technical skill* and *language*. Each ad has the format shown in Listing 3.1.

```
{
  "job_raw_text": text of job advertisement,
  "meta": {
    "sentence_id": id of sentence used for the annotation
  },
  "text": the randomly chosen text used for the annotation,
  "tokens": [ Tokens of text
    {
      "text": token,
      "start": index of first character,
      "end":  index of last character,
      "id": index of token position in text
    },
  ],
  "_input_hash": hash field,
  "_task_hash": hash field,
  "spans": [
    {
      "start": index of first character,
      "end": index of last character,
      "token_start": index of start token position in text,
      "token_end":  index of end token position in text,
      "label": category type
    }
  ],
  "answer": possible values {accept, reject, ignore}
}
```

Listing 3.1: Example of a job ad in the dataset in JSON format.

3.3.1 Preprocessing

After an error analysis, it was observed that the dataset had duplicate texts from the same job description annotated and that the annotations were different in some cases. Therefore, we kept each text only once and we made the acceptance from the same texts with different annotations to keep the one with the most entities. After the preprocessing, we had 32866 different texts annotated.

Another problem that was observed is that in some specific cases the entities were inconsistently annotated. For example, some texts had the entity <bachelor's> as education level, while others did not include the <'s> in the annotation. Consequently, the system could be unfairly penalized in when predicting <bachelor> and <'s> as parts of the entity and the correct annotation contains only <bachelor>. For this reason, we tried to remove such inconsistencies from the dataset, e.g. by always including both <bachelor> and <'s> in the annotated entity.

3.3.2 Encoding

After the preprocessing, we performed IO encoding in each entity. This encoding, tags each token as either belonging in a particular type of named entity type X (I-X) or in no entity (O). Table 3.1 shows two examples.

The dataset was then split into three parts; Train used for training the models, Dev used for hyper-parameter tuning and Test used for final evaluation. Figure 3.6 shows the distribution of each category for each set. Tables 3.2 and 3.3 show the number of entities and tokens, respectively, per category for each split, as well as for the whole dataset.

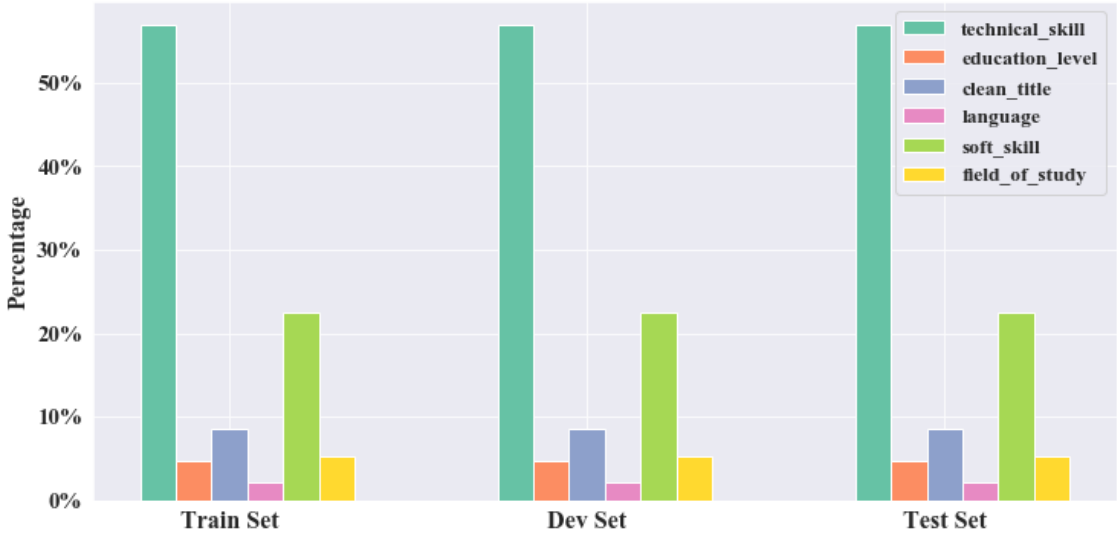

Figure 3.6: Dataset Categories' Distribution.

Tokens	IO encoding
Must	O
be	O
well	O
organized	I-soft_skill
and	O
detail	I-soft_skill
oriented	I-soft_skill
.	O

Tokens	IO encoding
Requires	O
a	O
bachelor	I-education_level
's	I-education_level
degree	O
with	O
a	O
strong	O
emphasis	O
on	O
natural	I-field_of_study
science	I-field_of_study
or	O
chemistry	I-field_of_study
.	O

Table 3.1: Two examples of the dataset using IO encoding.

	Train	Dev	Test	Total
Clean Title	2776	347	347	3470
Education Level	1550	195	193	1938
Field of Study	1743	219	217	2179
Soft Skill	7336	917	917	9170
Technical Skill	18591	2325	2323	23239
Language	680	86	85	851

Table 3.2: Number of entities for each category.

	Train	Dev	Test	Total
Clean Title	6665	851	804	8320
Education Level	2555	293	314	3162
Field of Study	2489	294	302	3085
Soft Skill	11960	1696	1484	15140
Technical Skill	34752	4245	4104	43101
Language	686	86	85	857
O	293031	18106	18138	329275

Table 3.3: Number of tokens for each category.

3.4 Experimental Setup

3.4.1 Flair framework

3.4.1.1 Early Stopping

The Flair framework was designed to stop the training after a specific number of epochs (variable max_epochs) and if the learning rate drops below a specific value (variable min_learning_rate). For this reason, as well as to avoid overfitting, we added early stopping to the model and set the patience to 10.

3.4.1.2 BERT Embeddings

The Flair framework allows users to use BERT embeddings. To be more specific it concatenates the layers of the BERT model that the user defines (default the last 4 layers). We used the cased model of BERT-base. BERT uses byte pair tokenization (e.g. the token "Analyst" could be split into the sub-words: "Ana", "##ly" and "##st".), therefore Flair framework implements different pooling operations (first and mean) to generate the final token representation. In addition, to the above pooling operations, we also added max pooling.

3.4.1.3 Hyper-parameter Tuning

The Flair framework uses the hyperopt package for parameter selection. The number of evaluation runs that hyperopt performed was set to 16 and the micro F1 score of the development set was used for choosing the best parameters. The hyper-parameters that were considered can be found in Appendix, while Table 3.4 shows their best values.

Hyper-Parameters	
Hidden Size	256
Dropout	0.25
RNN layers	3
Learning Rate	0.0001
Embeddings	WordEmbeddings("glove"), PooledFlairEmbeddings("news-forward", pooling = "min"), PooledFlairEmbeddings("news-backward", pooling = "min")

Table 3.4: Parameters used for training.

3.4.2 BERT-base

In NER, case information may be important so we used the cased version of BERT-base.

3.4.2.1 Embeddings

NER is a sequence tagging task, therefore we use BERT-base to produce one label per token. Since BERT uses sub-word units a word may be split in several tokens. As a consequence, we had to find a way to combine the results of the last hidden layer of BERT-base for each sub-word to generate the final token representation. We experimented with

three pooling operations: a) first pooling where we keep the results only of the first sub-word, b) mean pooling were the final representation was the average of all the sub-words, and c) max pooling where the token representation was the max of all sub-words.

3.4.2.2 Hyper-parameter Tuning

For BERT-base we performed a grid search, considering the hyper-parameters that can be found in Appendix. We used the micro F1 score of the entities of the development set for choosing the best parameters. We also used early stopping with patience 1. In table 3.5, we show the values of the best hyper-parameters.

Hyper-Parameters	
Learning rate	5e-05
Pooling Operation	"first"

Table 3.5: Parameters used for training.

3.5 Results

Table 3.6 shows the F1 score per category, as well as micro and macro averaged F1 across all categories. Table 3.7, shows three additional less strict metrics: a) micro averaged F1 across tokens, b) micro averaged F1 across entities where an entity is considered correct if at least one of its token is classified correctly, and c) micro averaged F1 across entities where an entity is considered correct if at least 50% of its tokens are classified correctly.

	Flair framework	BERT-base
clean_title	85.00	80.93
education_level	95.56	92.73
field_of_study	92.63	91.57
language	96.43	95.86
soft_skill	85.35	81.48
technical_skill	79.00	77.05
Micro Average F1	**82.89**	80.31
Macro Average F1	**88.99**	86.60

Table 3.6: F1 score Results on Test Set.

	Flair framework	BERT-base
Micro Average F1 tokens	**86.21**	86.19
Micro Average F1 50% tokens correctly labeled	**87.86**	87.28
Micro Average F1 at least 1 token correctly labeled	**88.49**	88.38

Table 3.7: Relaxed F1 scores Results on Test Set.

It can be observed that the Flair framework surpasses the BERT-base model in all the categories and the reported scores. One reason could be that the Flair framework model uses CRFs on top of the LSTMs, while BERT-base does not, which means that its predictions are at token level ignoring the neighboring tokens during inference. It should be clarified, that even though, one parameter considered in the hyper-parameter tuning for Flair framework were BERT embeddings, the use of CRFs on top of BERT-base is different from using frozen BERT embeddings-LSTMs-CRFs. Therefore, we cannot come to a conclusion about the results of BERT-base-CRFs.

To better understand the functionality of the examined models we performed an error analysis and identified two sources of errors: incorrect annotations (Tables 3.8 and 3.9) and entities with identical tokens but with different type (Table 3.10).

Text	True Labels	Flair framework Predictions	BERT-base Predictions
Sales	I-technical_skill	I-clean_title	I-clean_title
&	O	I-clean_title	I-clean_title
Business	I-clean_title	I-clean_title	I-clean_title
Development	I-clean_title	I-clean_title	I-clean_title
Director	I-clean_title	I-clean_title	I-clean_title
(	O	O	O
Japan	O	O	O
)	O	O	O

Table 3.8: Example of annotation error. “Sales” and “&” should have been annotated with I-clean_title as they are part of the job ad title.

Text	True Labels	Flair framework Predictions	BERT-base Predictions
Bachelor	I-education_level	I-education_level	I-education_level
's	I-education_level	I-education_level	I-education_level
degree	O	O	O
(	O	O	O
B.S.	O	I-education_level	I-education_level
)	O	O	O
or	O	O	O
equivalent	O	O	O
of	O	O	O
education	O	O	O
and	O	O	O
experience	O	O	O
.	O	O	O

Table 3.9: Example of annotation error. “B.S” should have been annotated with I-education_level as it is an abbreviation of Bachelor's Degree.

Text	True Labels	Flair framework Predictions	BERT-base Predictions
Computer	I-technical_skill	I-field_of_study	I-field_of_study
Science	I-technical_skill	I-field_of_study	I-field_of_study
background	O	O	O
and/or	O	O	O
Game	I-field_of_study	I-field_of_study	I-field_of_study
Design	I-field_of_study	I-field_of_study	I-field_of_study
Diploma	I-education_level	I-field_of_study	I-education_level
or	O	O	O
equivalent	O	O	O
.	O	O	O

Table 3.10: Example of entities with identical tokens but with different type. The models are not able to classify the tokens to the correct type.

Chapter 4

Representing Job Ads as a Graph

An additional objective of this thesis was to create a graph to represent the job descriptions and their entities. For this reason, we created a graph where every job ad and every entity are represented as nodes. Edges are added between the job ads and their corresponding entities. The dataset used for the creation of the graph is the same mentioned in the NER problem after the preprocessing.

4.1 Node2Vec

Node2Vec (Grover et al. 2016) is an algorithmic framework for representational learning on graphs. Node2Vec learns low-dimensional representations for nodes in a graph by optimizing a neighborhood preserving objective. It creates embeddings by producing random walks.

The embeddings, are learnt using a skip-gram model (Mikolov et al. 2013), which given a node tries to predict the surrounding nodes. Same way as a document is an ordered sequence of words, one could sample sequences of nodes from the underlying network and turn a network into an ordered sequence of nodes.

The objective of the algorithm is flexible, so it has the ability to learn representations that embed nodes from the same network community closely together, as well as to learn representations where nodes that share similar roles have similar embeddings. It can learn representations that organize nodes based on their network roles and/or communities they belong to by developing a family of biased random walks, which efficiently explore diverse neighborhoods of a given node. Figure 4.1 shows the embedding process of Node2Vec.

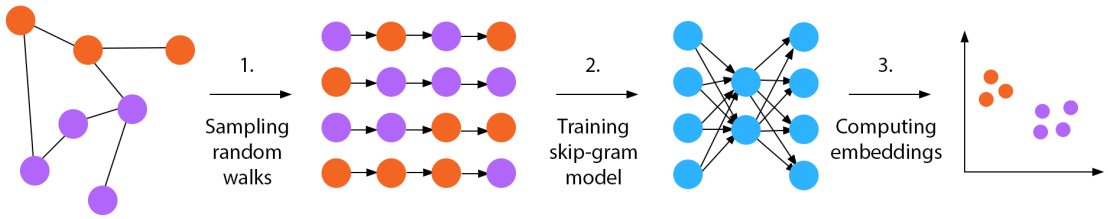

Figure 4.1: Embeddings Process of Node2Vec.

The sampling strategy, accepts 4 arguments:

- Number of walks: Number of random walks to be generated from each node in the graph
- Walk length: How many nodes are in each random walk
- p: Return hyper-parameter
- q: Inout hyper-aprameter

The parameters p and q are guiding the walk. Parameter p controls how fast the walk explores and parameter q how fast it leaves the neighborhood of the starting node. After transitioning to node v from t, the return hyper-parameter p and the inout hyper-parameter q, control the probability of a walk staying inward revisiting node (t), staying close to the preceding nodes (x_1), or moving outward farther away (x_2, x_3). Figure 4.2 shows the random walk procedure in Node2Vec.

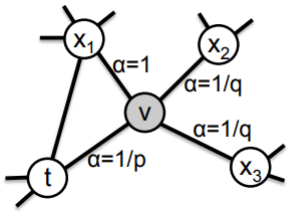

Figure 4.2: Illustration of the random walk procedure in Node2Vec.

4.2 Experimental Setup

4.2.1 Preprocessing

Each entity before being added on the graph and specifically all the tokens that an entity consists of, are joined together using the underscore and are lowercased. In order to avoid two words that are identically written but they have a different meaning to be confused, we also add the type of each entity (e.g. The entity “Data Analyst” becomes data_analyst_clean_title).

Another addition we made to the graph is to create nodes to represent entity types. Consequently it is possible to perform more abstract queries, e.g., a query that searches for job ads that require the knowledge of a language without specifying which particular language.

It was observed on the data that the job descriptions had different type of apostrophes. For example, <bachelor's> was written also as <bachelor’s>. For this reason, we replaced the different apostrophes found with the default one, in order for the job ads to be connected to the same entity and not different ones. Figure 4.3, shows a small part of the graph without all the edges for presentation reasons. Table 4.1 illustrates some statistics of the graph.

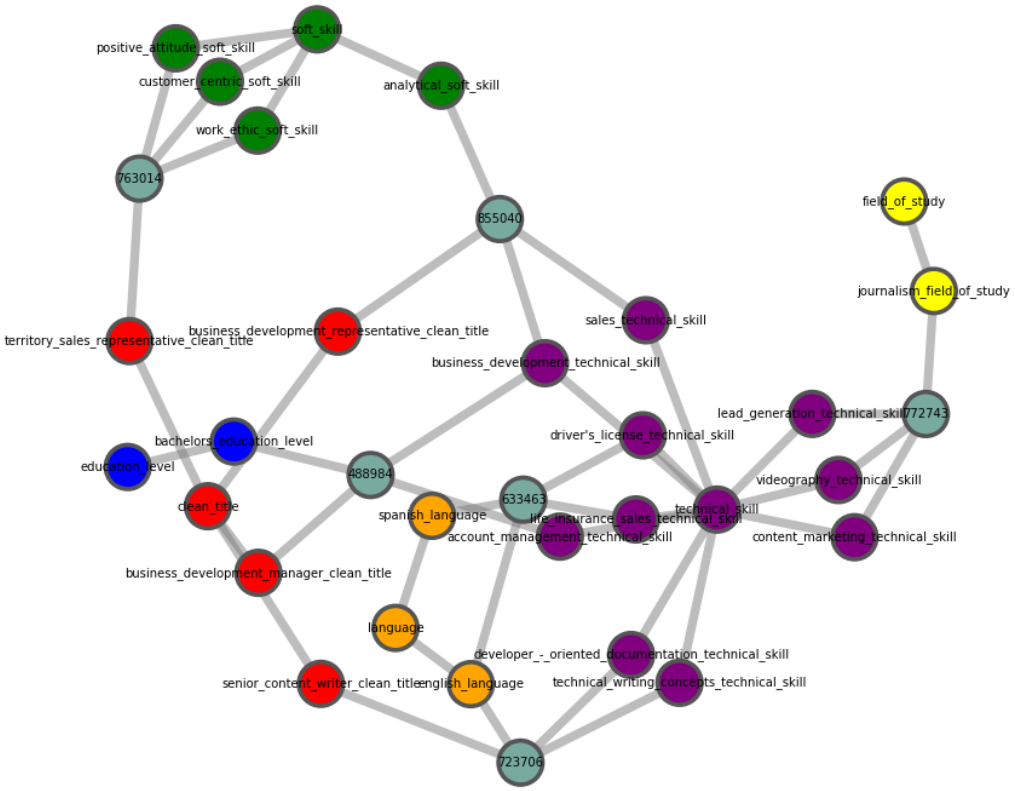

Figure 4.3: Small part of the Graph.

Number of Nodes	Number of Edges	Average Degree
17645	53967	6.1170

Table 4.1: Graph statistics.

4.2.2 Grouping Similar Entities

In our dataset exist many similar entities with different phrasings. This may result in an incomplete graph with missing connections between job ads. Therefore, we grouped these similar entities to enrich the graph with new connections using simple yet effective heuristics. We also performed some preliminary experiments with clustering but this did not produce satisfactory results.

Concerning the entity type *language* we grouped together entities that shared at least one word. This resulted in the groups of Table 4.2. We also grouped together entities with type *education level* (Table 4.3). After examining the data, we grouped together entities that required the same education level (e.g. bachelor). Specifically, we grouped together entities with a) different punctuation (e.g. bachelors and bachelor's), b) different phrasing (e.g. bachelor of science and bachelor's) and c) entities where one was the abbreviation of the other (e.g. bachelor's and b.s.).

Group Name	Group Contents
english_language	english_language, eng_language, enus_language egish_language, american_english_language
malay_language	malay_language, bahasa_malay_language, bahasa_language, indonesian_language, indonesian_bahasa_language,
portuguese_language	portuguese_language, brazilian_language, brazilian_portuguese_language
spanish_language	spanish_language, sp_language
arabic_language	arabic_language, arabic-_language

Table 4.2: Created groups for type *language*.

Group Name	Group Contents
requires_bachelor	•bachelor_'s_education_level, b.s._education_level, bachelor_of_science_education_level, b.a._education_level, 2:1_bachelor_education_level, b.s./b.a_education_level, bachelor's_education_level, bachelor_education_level, bachelor's_degree_education_level, b.sc_education_level, bachelor_of_arts_education_level, b.s_education_level, b.a_education_level, bachelors'_education_level, b.sc._education_level, bachelor_'s_education_level, bsc_education_level, bachelors_education_level
requires_master	master_education_level, masters_education_level, masters_of_science_education_level, m.sc_education_level, master_'s_education_level, post_-_master_'s_education_level, master_of_science_education_level, msc_education_level, m.a._education_level, m.s._education_level, masters_'_education_level, master's_degree_education_level, masters_of_taxation_education_level, masters'_education_level
requires_gcse	gcse_'s_education_level, gcse_education_level, gcse_level_education_level, gcses_education_level
requires_associates_degree	associate_'s_education_level, associate_education_level, associate_degree_education_level, associates_education_level
requires_high_school_diploma	h.s._diploma_education_level, hs_diploma_education_level, high_-_school_diploma_education_level, high_school_diploma_education_level, school_diploma_education_level

Table 4.3: Created groups for type *education level*.

4.3 Results

After creating the graph and the groups mentioned above, we extracted node embeddings with Node2Vec. We used the default parameter values apart from the number of random walks which was set to 25 since more than 90% of the nodes corresponding to job ads in our graph have a maximum degree of 25 (Figure 4.4).

Consequently using 25 random walks Node2Vec is able to explore almost every possible edge. We run Node2Vec two times, one without the created groups and one with the groups to see if the results changed and if grouping can improve the results.

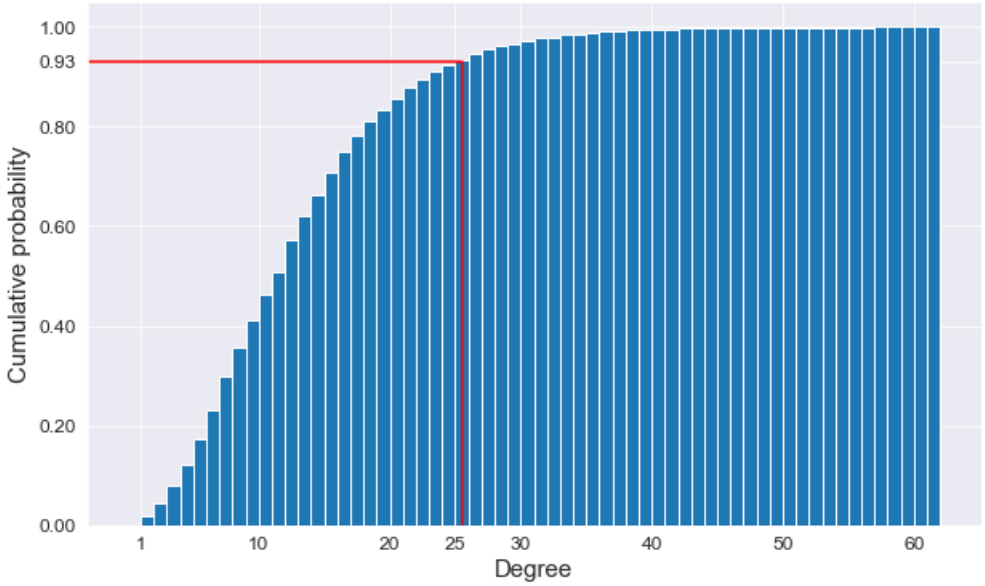

Figure 4.4: Cumulative Distribution of the degree of the job ad nodes.

One possible use of the graph would be to see how different entities can be grouped based on the ads they appear in. Table 4.4 shows the five most similar *technical skills* to "word_technical_skill" as retrieved from Node2Vec models trained with and without using entity grouping. Both models return, as expected, Microsoft Office programs.

Technical skill: word_technical_skill	
Model without groups	Model with groups
outlook_technical_skill	excel_technical_skill
excel_technical_skill	outlook_technical_skill
powerpoint_technical_skill	powerpoint_technical_skill
windows_operating_system_technical_skill	power_point_technical_skill
access_technical_skill	project_technical_skill

Table 4.4: Results of Graph when asking similar *technical skills*.

Another use of the graph could be to retrieve the most similar ads for a given job ad. Table 4.5 shows two examples. The cosine similarity of the original ad with the ad on the left is higher than with the ad on the right, which can be explained due to the fact that they share more common entities. One thing worth mentioning is that the models return ads that have entities similar to the original one but not the same (e.g. excel_technical_skill and microsoft_excel_technical_skill, business_mind_soft_skill and business_acumen_technical_skill). Thus, the graph succeeded on bringing the ads closer together.

In addition, Table 4.6 shows two examples indicating the importance of grouping similar entities. In both examples, grouping entities increase the similarity between job ads which were other wise very dissimilar due to the different phrasing of similar entities. Better grouping could increase more the similarities.

Ad Id: 411587 - Fraud & Risk Manager	
analyzing_data_technical_skill	fraud_technical_skill
attention_to_detail_soft_skill	identifying_trends_technical_skill
bachelor_'s_education_level (**requires_bachelor**)	management_technical_skill
business_mind_soft_skill	ms_word_technical_skill
decision_-_making_soft_skill	outlook_technical_skill
consumer_fraud_investigation_technical_skill	powerpoint_technical_skill
excel_technical_skill	problem_solving_soft_skill
finance_technical_skill	project_management_technical_skill
flexibility_soft_skill	sql_technical_skill
fraud_&_risk_manager_clean_title	

(a) Original ad and entities.

Ad Id 771181: Financial Planning Associate	Ad Id 907203: Night Shift: Operations Analyst
attention_to_detail_soft_skill	accounting_technical_skill
excel_technical_skill	bachelor_'s_education_level
financial_planning_associate_clean_title	communication_soft_skill
masters_education_level	finance_technical_skill
ms_word_technical_skill	microsoft_excel_technical_skill
outlook_technical_skill	operations_analyst_clean_title
powerpoint_technical_skill	problem_solving_soft_skill
qfa_technical_skill	sql_technical_skill

(b) Model without groups.

Ad Id 771181: Financial Planning Associate	Ad Id 687139: Director of Practice Operations
attention_to_detail_soft_skill	business_acumen_technical_skill
excel_technical_skill	cost_consciousness_soft_skill
financial_planning_associate_clean_title	customer_service_technical_skill
ms_word_technical_skill	decision_-_making_soft_skill
outlook_technical_skill	director_of_practice_operations_clean_title
powerpoint_technical_skill	planning_technical_skill
qfa_technical_skill	problem_solving_soft_skill
requires_master	project_management_technical_skill

(c) Model with groups.

Table 4.5: Example returning similar ads for each model.

Ad Id: 705835 - Freelance Study Coordinator	Ad Id: 602805 - Corporate Paralegal
bachelor_'s_education_level (**requires_bachelor**)	attention_to_detail_soft_skill
life_science_field_of_study	accuracy_soft_skill
clinical_research_technical_skill	microsoft_office_suite_technical_skill
study_coordinator_clean_title	ms_excel_2010_technical_skill
communication_skills_soft_skill	organizational_soft_skill
english_language	interpersonal_soft_skill
	b.a._education_level (**requires_bachelor**)
	communication_skills_soft_skill
	corporate_paralegal_clean_title
Cosine similarity without groups	**24.07**%
Cosine similarity with groups	**46.52**%

(a) First example.

Ad Id: 771181 - Financial Planning Associate	Ad Id: 549366 - Purchasing Agent
financial_planning_associate_clean_title	bsc_education_level (**requires_bachelor**)
ms_word_technical_skill	ba_education_level
excel_technical_skill	business_administration_field_of_study
outlook_technical_skill	msc_education_level (**requires_master**)
powerpoint_technical_skill	ma_education_level
qfa_technical_skill	ms_office_technical_skill
masters_education_level (**requires_master**)	negotiating_prices_and_terms _and_conditions_technical_skill
attention_to_detail_soft_skill	analytical_mindset_soft_skill
	purchasing_agent_clean_title
	communication_soft_skill
	interpersonal_skills_soft_skill
	organizational_soft_skill
	market_research_technical_skill
	data_analysis_technical_skill
	purchasing_technical_skill
Cosine similarity without groups	**12.48**%
Cosine similarity with groups	**37.02**%

(b) Second example.

Table 4.6: Example showing the difference in the cosine similarity when using the two models.

Finally, the graph could be used to retrieve entities of a specific type (e.g., *technical skill*) required by ads with a specified *clean title* (e.g., "data_analyst"). Table 4.7 shows an example.

Clean title: data_analyst	
Model without groups	Model with groups
sql	sql
looker	looker
ggplot2	ggplot2
shiny	shiny
r_markdown	r_markdown
jupyter_notebooks	jupyter_notebooks
etl	etl
aws_redshift_sql	aws_redshift_sql
kafka	kafka
matillion_etl	matillion_etl
python	python
r	r

Table 4.7: Results of Graph when asking for *technical skills* of a specific *clean title* (“data_analyst”).

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this thesis, we dealt with NER for job descriptions. We experimented with two models, BERT-base and Flair framework which uses Bi-LSTMs with a CRF layer. Our results showed that Flair framework outperformed BERT-base. One possible reason is that BERT-base produces worst results due to the fact that it does not use a CRF layer on top, therefore it does not take into account the neighboring words when it decides the tag of a certain word.

Additionally, we created a representation of the job descriptions with a graph. We created two different models one with the graph as it is, and one where we group some entities together. Specifically, we grouped together entities which were considered different due to their phrasing, despite having the same meaning. Both the resulting Node2Vec models (with and without grouping) can be used to retrieve similar ads as well as similar entities of a predefined entity type (e.g., *technical_skill*). Moreover, even though the grouping was rather simple, it managed to connect more ads together and increase their cosine similarity.

5.2 Future Work

As far as the NER task is concerned, an idea for future work is to use a CRF layer on top of BERT-base, which could improve significantly its results, considering it has a high score without even using a CRF layer. Another idea that could improve both models would be to train and use domain specific embeddings (embeddings from job advertisements) rather that the pre-trained general embeddings that were used in this thesis.

Concerning the graph, a possible improvement could be to enrich it by inserting ads and their entities as they are extracted by the NER model. This, however has the risk of inducing noise due to the prediction error of the NER model. Nevertheless, we expect the additional data to help Node2Vec produce richer representations. Moreover, we used the default parameters of Node2Vec, thus one improvement would be to fine tune the model. Additionally, as we showed, grouping entities together improved the performance of the model and increased the cosine similarity between the ads. Therefore, if more information for the advertisements can be found (e.g. function, industry, seniority, part-time vs full-time) and added, as well as for the entities, the results of Node2Vec would be improved. Finally, another idea is to train Node2Vec by considering not only the network's structure but also the text describing each node as shown by Kotitsas et al. (2019).

Bibliography

Akbik, Alan et al. (June 2019a). "FLAIR: An Easy-to-Use Framework for State-of-the-Art NLP". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 54–59. URL: https://www.aclweb.org/anthology/N19-4010.

Akbik, Alan et al. (2019b). "Pooled Contextualized Embeddings for Named Entity Recognition". In: *NAACL 2019, 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 724–728. URL: https://www.aclweb.org/anthology/N19-1078.

Baevski, Alexei et al. (2019). "Cloze-driven Pretraining of Self-attention Networks". In: *CoRR* abs/1903.07785. arXiv: 1903.07785. URL: http://arxiv.org/abs/1903.07785.

Clark, Kevin et al. (2018). "Semi-Supervised Sequence Modeling with Cross-View Training". In: *CoRR* abs/1809.08370. arXiv: 1809.08370. URL: http://arxiv.org/abs/1809.08370.

Devlin, Jacob et al. (2018). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805. arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

Goldberg, Yoav et al. (2017). *Neural Network Methods in Natural Language Processing*. Morgan & Claypool Publishers. ISBN: 1627052984, 9781627052986.

Grover, Aditya et al. (2016). "node2vec: Scalable Feature Learning for Networks". In: *CoRR* abs/1607.00653. arXiv: 1607.00653. URL: http://arxiv.org/abs/1607.00653.

Hochreiter, Sepp et al. (1997). "Long Short-Term Memory". In: *Neural Computation* 9.8, pp. 1735–1780.

Huang, Zhiheng et al. (2015). "Bidirectional LSTM-CRF Models for Sequence Tagging". In: *CoRR* abs/1508.01991. arXiv: 1508.01991. URL: http://arxiv.org/abs/1508.01991.

Kotitsas, Sotiris et al. (2019). "Embedding Biomedical Ontologies by Jointly Encoding Network Structure and Textual Node Descriptors". In: *CoRR* abs/1906.05939. arXiv: 1906.05939. URL: http://arxiv.org/abs/1906.05939.

Lafferty, John (2001). "Conditional random fields: Probabilistic models for segmenting and labeling sequence data". In: Morgan Kaufmann, pp. 282–289.

Mikolov, Tomas et al. (2013). *Efficient Estimation of Word Representations in Vector Space*. URL: http://arxiv.org/abs/1301.3781.

Peters, Matthew E. et al. (2017). "Semi-supervised sequence tagging with bidirectional language models". In: *CoRR* abs/1705.00108. arXiv: 1705.00108. URL: http://arxiv.org/abs/1705.00108.

Appendix

Models Hyper-Parameters

Table A.1 describes the hyper-parameters that were considered for Flair framework, Table A.2 describes the hyper-parameters that were considered for BERT-base, Table A.3 shows the possible values for the Flair framework and Table A.4 shows the possible values for BERT-base.

Hyper-Parameters	Description
Hidden Size	The number of hidden states in RNN
Dropout	The dropout probability of the RNN
RNN layers	The number of RNN layers
Learning Rate	The initial learning rate
Embeddings	The embeddings used in the tagger

Table A.1: Description of the hyper-parameters for Flair framework.

Hyper-Parameters	Description
Learning Rate	The learning rate
Pooling Operation	The operation used to the sub-tokens

Table A.2: Description of the hyper-parameters for BERT-base.

Hyper-Parameters	
Hidden Size	[128, 256]
Dropout	[0, 0.25, 0.5]
RNN layers	[1, 2, 3]
Learning Rate	[1e-4, 1e-3]
Embeddings	[WordEmbeddings(“glove”), StackedEmbeddings([PooledFlairEmbeddings(“news-forward”, pooling_operation = “min”), PooledFlairEmbeddings(“news-backward’, pooling_operation = “min”)]), StackedEmbeddings([WordEmbeddings(“glove”), PooledFlairEmbeddings(“news-forward”, pooling_operation = “min”) PooledFlairEmbeddings(“news-backward”, pooling_operation = “min”)]), BertEmbeddings(“bert-base-cased”, pooling_operation = “mean”), BertEmbeddings(“bert-base-cased”, pooling_operation = “first”), BertEmbeddings(“bert-base-cased”, pooling_operation = “max”)]

Table A.3: Hyper-Parameteres considered for Flair framework.

Hyper-Parameters	Values
Learning Rate	[2e-5, 3e-5, 5e-5]
Pooling Operation	[“first”, “mean”, “max”]

Table A.4: Hyper-Parameteres considered for BERT-base.