Efficient Multilingual Deep Learning Model for Keyword Categorization

Efficient Multilingual Deep Learning Model for Keyword Categorization

Mirko Polato[1][0000−0003−4890−5020], Denys Demchenko[2], Almatkhan Kuanyshkereyev[2], and Nicolò Navarin[1][0000−0002−4108−1754]

[1] University of Padova, Department of Mathematics, Padova, Italy
{mpolato,nnavarin}@math.unipd.it
[2] ID Ward, Nottingham, UK
{denys,almat}@id-ward.com

Abstract. Keywords categorization is an essential tool for SEO (Search Engine Optimization), digital marketers and online advertising. Keywords represent one of the most valuable pieces of information to infer the users' intents and interests. An effective keyword categorization method allows understanding what types of content are in the greatest demand and can help improve future content strategies or marketing/ad campaigns.
In this paper, we present a novel deep learning model for multilingual keyword categorization. The model relies on fastText multilingual word embeddings, and its architecture is inspired by the DeepSets model. Moreover, to make use of (training) words not included in the pre-trained fastText embeddings, we initialize them as the average embedding over-all the co-occurrent words. Then, we fine-tune these representations by allowing the network to back-propagate the error to the input.
We assess the quality of our proposal on a real-world dataset provided by a Spanish company where keywords are categorized upon the Google Product Taxonomy (GPT). Results on both level three and level six of the GPT show that our model can achieve high accuracy scores while being extremely efficient.

Keywords: Keyword Categorization · Word Embeddings · Deep Neural Networks · Deep Learning

1 Introduction

Online Advertising is a huge market that shows significant growth every year. Global advertising media owners revenue valued 613.9 billion US Dollars in 2019 and it is expected to grow to 762.76 billions US Dollars in 2024[3]. Many publishers use online advertising as a mean to keep content free of charge for users. Instead of a paywall, advertisers pay for their ads to be shown on the publisher's websites or apps. In order for such marketing campaigns to be effective and cost efficient, ads must be shown to people who are the advertiser's target audience.

[3] https://www.statista.com/statistics/236943/global-advertising-spending/

One of the approaches to targeted advertisement that can be deployed by companies is to identify user preferences and behaviour on certain products. Extracting and categorizing keywords from the content consumed by the user helps companies to find the information that is relevant to the user's interests and hobbies. This information is used to segment the publisher's audience in order for internet advertisement to be shown to targeted segments selected by the advertiser. This approach leads to increased effectiveness and reduction of costs of campaigns, as only relevant advertisement is shown to users. For such an approach to be deployed, there is the need to classify a substantial amount of keywords that are continuously generated by users from all over the world (thus in different languages). Therefore, the classification should be extremely scalable and efficient, while being able to consider multiple languages. Still, accuracy is a crucial aspect since too many errors may lead to a miss-specification of a user's interests. While on similar problems, simple machine learning approaches, such as Naïve Bayes, tend to achieve decent results, in the considered problem they fail to model the similarities among words across different languages, possibly resulting in not satisfying predictive performance.

Neural networks, instead, are extremely powerful models that, exploiting word embeddings, can consider intra-language and inter-language relationships among words. Moreover, they are one of the most efficient machine learning algorithms at prediction time.

In this paper, we present our deep learning based solution for the problem of large-scale multilingual keyword categorization on a dataset provided by *[anonymous for double blind submission]* in the context of one of the *[anonymous for double blind submission]* challenges. We exploit the recently proposed fastText [1] word embeddings, that allow to cope with the multi-lingual problem in an efficient and effective way. We then use set-based neural architectures that, relieving from the sequentiality of the input, allow for a strong parallelization on GPUs, being more efficient than sequence-based counterparts. Finally, we propose to fine-tune the pre-trained word embeddings to further improve the predictive performance of our model without impacting the final throughput.

2 Problem description

The problem we need to solve is keyword categorization (a.k.a., classification). Formally, given a labeled training set of keywords $\{(K_i, c_i)\}_{i=1}^{n}$, we aim at learning a function f such that it correctly classifies unseen keywords. For simplicity, a keyword K can be considered as a sequence of words, i.e., $K = \langle w_1, w_2, \ldots, w_k \rangle$. In the following sections, we will specify how keywords are divided into words.

The specific task at hand has some peculiar characteristics that make it particularly challenging:

– the keywords are categorized over the Google Product Taxonomy (GPT)[4] which currently has a total of 2454 categories arranged in a hierarchical way.

[4] https://developers.google.com/adwords/api/docs/appendix/verticals

However, in our case we get access only to labeled examples from the third (1026 categories) and sixth level (138 categories) of the taxonomy. Level 3 alone covers more than the 40% of the whole GPT;
- the keywords are in 17 different languages not equally distributed with unknown distribution;
- the required level of accuracy is high since wrongly classified keywords can lead to mistakes in the ad campaign by the company;
- the model should be very efficient, especially at prediction time, because the amount of keywords that has to be handled in production is huge (potentially millions of keywords every day).

2.1 Dataset

The dataset, provided by [anonymous for double blind submission], contains users' generated keywords that have been semi-automatically categorized over the GPT. Specifically, we have keywords from the 3$^{\text{rd}}$ and the 6$^{\text{th}}$ level of the Google taxonomy. Keywords come from 17 different languages namely english, spanish, italian, swedish, turkish, thai, japanese, korean, norvegian, vietnamise, dutch, russian, danish, polish, portoguese, french, and german. In the reminder of the paper, we will refer to the set of all languages with L. The details of the considered dataset are summarized in Table 1. From Table 1 we can notice that

Name	**GPT**	**#keywords**	**avg. length**	**#words**	**#categories**	**avg. kw.×cat.**
`lvl_3`	3	2.63 M	3.75 ± 1.50	382 511	1 026	2 566
`lvl_6`	6	3.35 M	3.85 ± 1.58	198 733	138	24 298.19

Table 1: Datasets' information: number of keywords, average keywords' length ($\pm$ standard deviation), number of different words, number of categories, and average number of examples per category.

the main difference between `lvl_3` and `lvl_6` is the average number of keywords per category which is one order of magnitude less in `lvl_3`. This difference, along with the number of classes, makes the classification on `lvl_3` particularly challenging.

All keywords passed through a simple pre-processing phase where they are lower cased and special characters are removed. With special characters we mean those characters in between two words that are for sure not part of the syntax of the word. For instance, in the keyword 'this+apple' (or 'this +apple') the '+' sign is removed. We make this conservative choice because we handle different languages that can make use of special characters, e.g., japanese ideograms, or the thai's characters.

For experimental purposes, we randomly divided the dataset into training and test set using a 80-20% proportion. Additionally, we created a validation set which is 10% of the size of the training set.

3 Background and Related works

One of the most prominent advances in representation learning for Natural Language Processing of the last decade was the introduction of the concept of word embeddings. In previous-generation machine learning approaches, words were generally encoded as one-hot vectors, discarding all information about the semantic similarity between them. With word embeddings, each word is represented by a (relatively small) vector. The mapping between words and their respective embeddings is generally learned in an unsupervised way. In more detail, the embeddings are initialized at random. Then, a self-supervised loss is exploited to refine such embeddings [14, 16, 18, 13, 3, 1]. The core advantage of this approach is that similar words tend to have embeddings that are spatially close. Word embeddings are usually trained on huge corpora, e.g. Wikipedia dumps in a specific language.

The training phase of deep learning models can exploit these similarities by encoding the input words with the corresponding pre-trained embedding (that is a form of transfer learning), resulting in less data-hungry models compared to training them from scratch.
Since the training of such word embeddings requires much computational effort, many pretrained word embeddings have been made available online [18, 16]. Most of the existing embeddings are computed separately for different languages. When dealing with datasets in multiple languages, the straightforward approach consists in developing a separate model for each of them. This approach, however, does not allow to transfer information from one language to the others, potentially harming the performance of the final system in languages where few training data is available.

Recently, several methods to learn bi/multilingual word embeddings have been proposed [5, 4, 21, 11, 10]. The goal is to have embeddings for words denoting similar concepts possibly in different languages that are close in a shared inter-lingual embedding space. For example, the learnt representation of the English word "home" should be similar to the one of the Italian word "casa". To perform learning, most of these approaches rely on sentence/document aligned corpora and/or seed parallel lexicon. Given two languages, the main idea is to jointly learn monolingual embeddings that are "aligned" in a common space via a regularization that leverages the aligned corpora/lexicon.

Nonetheless, given a new language, most of these techniques require to perform an entire new training which is computationally very demanding if the number of languages is high. For this reason, our solution relies on the approach proposed by Facebook AI Research [11, 6] in which no parallel data is needed, allowing to perform the alignment step efficiently starting from the mono-lingual embeddings alone. We discuss our solution in Section 4.4.

When considering texts from different sources w.r.t. the corpus used for the pretraining of word embeddings, there is the necessity to deal with novel words, i.e. words not frequent enough (or at all) in the pre-training data. We discuss this issue in Section 4.6.

3.1 Deep Learning architectures

When dealing with short texts such as the considered keywords, many deep learning approaches are possible to perform predictive tasks.

Considering the input keywors as a sequence of words $K = \langle w_1, w_2, \ldots, w_k \rangle$, the mainstream approaches are recurrent neural networks (RNNs) and Transformer architectures. In RNNs, the input sequence is presented to the network and processed sequentially. Each Recurrent Unit (RU) maintains a state (i.e. a memory), in the form of a hidden vector, that influences the transformation applied to the input element. After an element is processed, the RU produces the updated hidden state (to be used for the following element in the sequence) and an output that depends on all the elements in the sequence that have been presented to the RU so far (via the memory vector). Usually, when dealing with classification tasks, only the output produced by the last element is considered. A well-known version of RU are Long Short-Term memory Networks [8], that have been successfully applied to a wide range of problems [17, 2]. The main drawback of RNNs is that to learn the weights of the recurrent neurons, we have to resort to backpropagation through time [22], that does not allow to parallelize the processing of the elements in the sequence.

The Transformer architecture [20] leverages on the concept of attention to allow to process sequential data in a parallel fashion. Attention allows to explicitly model the dependencies among the different elements in the input sequence. Such models have been heavily applied in NLP, with BERT [3] being the best known example. Even though the approach of fine-tuning a pretrained BERT model is a common approach for text classification tasks, the model is relatively big and the necessity to store and run it makes it computationally unattractive for cases in which the computational efficiency is a key aspect.

4 Methods

In this section, we describe the details about the overall method we propose to efficiently solve the keyword classification task.

4.1 Classification pipeline overview

Besides accuracy, our main focus is the efficiency of the method, especially at prediction time. As a consequence, we keep the entire classification pipeline as simple as possible, including only the essential steps. Figure 1 summarizes the pipeline. The pre-processing step (see Section 2.1) consists in the removal of special characters and in the lower casing of the remaining ones. Then, the keywords are split (on the space character) into words. For each word, its embedding is retrieved from a look-up table. The look-up table consists of the pre-computed fastText's word embedding for all the 17 languages, with the addition of the embeddings of the training words for which we do not have a corresponding fastText embedding. Sections 4.2, 4.3 and 4.4 describe in detail how we construct the look-up table. At test/validation time, if a word is not included in

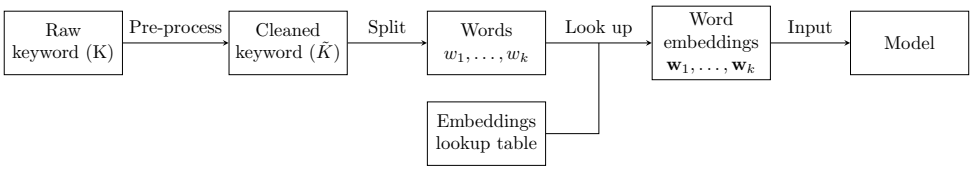

Fig. 1: Visual depiction of the classification pipeline.

the look-up table is ignored, i.e., a 0 vector is used as embedding. Finally, the keyword is fed into the model as a 2D tensor where the rows are the words and the columns are the latent features of the words. To set the dimension of the keyword's 2D tensor, we fixed a maximum number of words per keyword (in our case 10). If a keyword has less than 10 words, we add 0 padding vectors to fill the remaining rows.

4.2 Word embeddings

The literature offers a large set of state-of-the-art word embedding techniques. The most known (and used) ones are: word2vec [16], GloVe [18], fastText [1], and the more recent BERT [3]. In our model we employ the Facebook's fastText embeddings and the motivations of our choice are explained in the following.

FastText [1] takes the idea of word embeddings in word2vec [16] a step ahead and learns representations for character n-grams, and then represents words as the sum of the learnt n-gram vectors. fastText improves word2vec on several aspects, such as learning representation for internal word structures instead of the entire word, which gives fastText the ability of generating embeddings for out-of-vocabulary (OOV) words. Similar considerations also applies with respect to GloVe [18]. However, to generate OOV words there is the need of having the model of each language (or at least the embedding of all the n-grams) which is expensive especially in terms of space. Moreover, we only had access to the pre-computed word embeddings that Facebook released in the MUSE library[5]. Nonetheless, the fastText's strongest suit is its efficiency (hence its name). For instance, we can train fastText on more than one billion words in few minutes on a standard multi-core CPU. Additionally, fastText provides pre-computed word vectors for more than 150 languages, and for many of them Facebook released their multilingual version that we discuss in the following section.

The efficiency is the key aspect that made us prefer fastText over BERT which is arguably the current standard.

[5] https://github.com/facebookresearch/MUSE

4.3 Multilingual alignment

As introduced in Section 3, to deal with the multilingual nature of the dataset, we rely on the idea of word embeddings alignment. Specifically, we employ the (supervised) iterative Procrustes [19] alignment [15, 11]. The core concept of this technique is to align the embedding space of a source language with the one of a target language (used as the reference space). This alignment is made possible by the use of the so called anchor words, i.e., pairs of words that are semantically the same in the source and in the target language.

Formally, suppose we are given a set of m word pairs and their associated embeddings $\mathcal{P} \equiv \{\mathbf{x}_i, \mathbf{z}_i\}_{i=1}^{m}$, where $\mathbf{x}_i \in \mathbb{R}^{d_1}$ is the representation of word i in the source language, and $\mathbf{z}_i \in \mathbb{R}^{d_2}$ is the vector representation of the the corresponding word in the target language. The goal is to find a transformation matrix $\mathbf{W}^*$ such that $\mathbf{W}^*\mathbf{x}_i \approx \mathbf{z}_i$ by solving

$$\mathbf{W}^* = \arg \min_{\mathbf{W} \in \mathbb{R}^{d_2 \times d_1}} \|\mathbf{W}\mathbf{x}_i - \mathbf{z}_i\|^2.$$

There exists also an unsupervised version of this approach [11], where the matrix $\mathbf{W}$ is initially estimated using an adversarial approach, and then the set $\mathcal{P}$ is constructed using as anchor points the words that are the best matches according to $\mathbf{W}$. However, we employ the supervised approach because (i) for 14 out of 17 languages the aligned embeddings were already available, and (ii) for the remaining 3 languages (namely, japanese, thai and korean) the bi-lingual dictionaries (i.e., english-japanase, english-thai, and english-korean) of 5k words were also available. For these three languages we performed the aligned by using the MUSE library. Every language has been aligned using the English as the target language.

4.4 Inferring the language of a keyword

To select the correct embedding for a word, we need to recognize which language the keyword belongs to. A common approach to infer the language of a keyword is to identify the language(s) of each word in it and then take a consensus, e.g. a majority vote. The identification of the language of a word can be performed using dictionaries: if a word w is contained in the dictionary of the language l than w belongs to l. Note that with this procedure a specific word may be associated to more than one language. Notwithstanding its simplicity, this technique is neither perfect (e.g., a single keyword can contain words from multiple languages) nor very efficient because many dictionary look-ups have to be performed for each word in a keyword. Since we mainly focus on the efficiency, we propose a similar yet different approach that can be pre-computed only once before the training (Algorithm 1).

The idea is to build a single multi-language dictionary ($\mathcal{D}$) that associates to each word the embedding coming from the language in which the word is more frequent (relatively speaking). Formally, given a word w, the language l^*

Algorithm 1: Multilingual dictionary construction

Input: $D = [\mathcal{D}_1, \ldots, \mathcal{D}_L]$: list of embedding dictionaries, one per language
Output: $\mathcal{D}$: multilingual embedding dictionary

```
D ← {}
for D_i ∈ D do
    for w ∈ D_i do
        if w ∉ D then
            lang ← i
            for D_j ∈ D \ D_i do
                if w ∈ D_j ∧ rel_freq(w, j) > rel_freq(w, i) then
                    lang ← j
            D[w] ← D_lang[w]
return D
```

associated to w is defined as

$$l^* = \arg\max_{l \in L} \mathbb{P}(w|l)\mathbb{P}(l).$$

Clearly, the exact $\mathbb{P}(w|l)$ and $\mathbb{P}(l)$ are unknown, thus we approximate it by using dictionaries extracted from OpenSubtitles.org[6]. In Algorithm 1, rel_freq(w, l) is the function that estimates $\mathbb{P}(w|l)\mathbb{P}(l)$. The dictionary $\mathcal{D}$ represents the embeddings look-up table. The look-up table can be computed once, or every time the embeddings are updated. The overall complexity of Algorithm 1 is linear with respect to the number of unique words in all the dictionaries, i.e., $|\bigcup_{i=1}^{L} \mathcal{D}_i|$.

4.5 The learning models

To solve the keyword classification problem, we propose a series of neural network architectures with increasing complexity. All the proposed models take as input keywords represented as a 2D tensor in $\mathbb{R}^{m \times d}$, where m is the maximum number of words per keyword and d is the size of the embedding. In our case, $m = 10$ and $d = 300$.

Multi-Layer Perceptron The first model we propose, i.e., Multi-Layer Perceptron (MLP), is a simple yet effective baseline. The core assumption is that most of the complexity of the learning task has been already taken care of by the fastText model that computed the word embeddings. Moreover, being a simple model, the MLP is fast to train and it is also very efficient at prediction time. Formally, given a vectorial representation for a keyword $\mathbf{x}$, the MLP model with a single hidden layer can be defined by the following parametric function:

$$f_{\text{MLP}}(\mathbf{x}; \boldsymbol{\theta}) = \sigma(\mathbf{W}_2(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)^+ + \mathbf{b}_2),$$

[6] https://github.com/hermitdave/FrequencyWords

where $\boldsymbol{\theta} \equiv \{\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2\}$ are the trainable parameters of the network, $(\cdot)^+$ is the rectified linear unit (i.e., the ReLU activation function), and σ is the softmax function. The MLP model accepts as input a one dimensional vector, thus we need an aggregation function that transform the 2D input keyword representation into a 1D vector. To avoid losing much information, we opt for the sum as aggregation function and hence a keyword is represented as the sum of the embeddings of its words. Note that, differently from the average aggregation, using the sum allows to keep the padding as it is without adding any in-between operations (e.g., masking) since the 0 padding does not affect the sum.

The network parameters $\boldsymbol{\theta}$ are optimized using a Stochastic Gradient Descent (SGD) approach aiming at minimizing the cross entropy loss.

DeepSets Given the short nature of keywords, we argue that the sequential order of the words does not play a crucial role in learning a suitable keyword representation. This assumption allows to avoid the use of recurrent models that would harm the efficiency of both the training and prediction phase. However, using a simple aggregation function like the sum (see Section 4.5) may not be the best choice. For this reason, we propose to improve upon the MLP model by adding a further network that learns a word transformation that allows to better represent the keywords as sum of their words.

Specifically, we implemented a DeepSets [23] model. DeepSets are adapt in this context because we treat keywords as set of words and hence we need a network that is permutation invariant. Given a keyword $\mathbf{X}$ represented as a 2D tensor with word embeddings on the rows, the DeepSets can be formalized as:

$$f_{\text{DS}}(\mathbf{X}; \boldsymbol{\theta}', \boldsymbol{\theta}) = \rho\left(\sum_{\mathbf{x} \in \mathbf{X}} \phi(\mathbf{x}; \boldsymbol{\theta}'); \boldsymbol{\theta}\right),$$

where ϕ and ρ are two neural networks parametrized by $\boldsymbol{\theta}'$ and $\boldsymbol{\theta}$, respectively. In our model, we fixed $\rho(\cdot; \boldsymbol{\theta}) := f_{\text{MLP}}(\cdot; \boldsymbol{\theta})$, while

$$\phi(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{W}_{l+1}(\dots(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)^+ \dots)^+ + \mathbf{b}_{l+1}$$

is an l-layers feed forward network with a linear activation function in the output layer and with $\boldsymbol{\theta}' \equiv \{\mathbf{W}_i, \mathbf{b}_i\}_{i=1}^{l+1}$. The DeepSets model has been optimized in the same way as the MLP. Figure 2 provides a visual depiction of the architecture.

4.6 Learning representations for missing words

Not all the words appearing in a keyword have associated a pre-computed embedding and hence they must be handled differently. The most common approach is to simply ignore such words. However, in cases in which the text snippets for the considered prediction task are short, this approach may lead to an information loss that can negatively impact the overall predictive performance of the model. Unfortunately, this is the setting we consider in this paper, where the average length of keywords is 3.8 words (see Table 1).

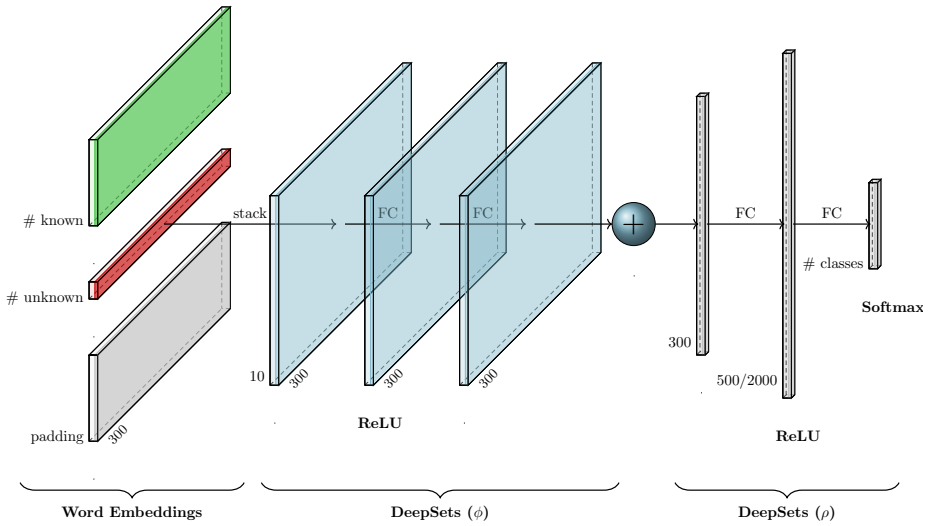

Fig. 2: The overall architecture of our model. The input keyword is represented as a 2D tensor including the embeddings of the known words (i.e. included in the pre-trained embeddings), the emebedding of the unknown/missing words (if any), and a 0 padding (if any). The 2D keyword representation is then fed into the ϕ network of the DeepSets model whose output (i.e., transformed word embeddings) is summed up and fed ρ network (i.e., Multi-Layer Perceptron) that performs the classification.

In such setting, it is important to maintain as much information as possible from the input, that is to discard as few words as possible. An approach that can be pursued is to recompute the word embeddings, augmenting the original corpus with the dataset at hand. This approach requires, in general, a huge computational effort, and since we deal with very short sentences there is no guarantees that the learned representation would be satisfactory.

For these reasons, we propose to approximate the embedding of the missing words using a two-step approach: (i) initialization, and (ii) fine-tuning.

Initialization In the first step, given a missing word w', we include it in $\mathcal{D}$ by initializing its embedding with:

$$\mathbf{w}' = \frac{1}{N} \sum_{\substack{K \in \mathcal{K} \\ w' \in K}} \sum_{w \in K \setminus \{w'\}} \mathcal{D}[w],$$

where $\mathcal{K}$ is the set of keywords in the training set, and N is the number of keywords in which w' appears. In other words, the embedding of w' is initialized as the average embedding of all the words that co-occur with w' in the keywords.

Note that this approach does not require any supervision (similarly to most methods computing word embeddings). For this reason, if additional unlabeled data coming from the same input distribution is available, it is possible to exploit it to build a better initialization for missing words.

Fine-tuning The just described initialization allows to not discard any word appearing in the training set, independently from the dictionary size of the pre-trained word embeddings. However, this initialization may be noisy and may not reflect well the information content of the words with respect to the task at hand. We thus propose to fine-tune the representations of missing words. In particular, we allow the error of the network to be back-propagated to the input embeddings of missing words during training. In this way, the word embeddings of missing words are specialized according to the considered classification task. Notice that, while a strongly non-linear transformation of the input should allow the network to learn appropriate representations without changing the input, in practice such transformations may lead to overfitting. In our experiments, we explored both solutions and find the fine-tuning approach to be the most effective.

Given the effectiveness of fine-tuning the word embeddings of missing words, we also explore the fine-tuning of all the word embeddings, not just the ones of missing words. We show in the experimental section that this step further improves the predictive results.

5 Experimental results

In this section we describe the performed experiments using the just described methodology.

5.1 Experimental setting

All the experiments reported in this section have been performed on the dataset described in Section 2.1. As stated before, we split the dataset into training, validation and test set having the following relative sizes 72%, 8%, 20%, respectively. We measured the models' performance in terms of accuracy that was expressively required in the challenge. The model hyper-parameters, e.g., number of layers and number of nodes per layer, have been selected according to the performance on the validation set. Each model has been trained for at most 20 epochs and the best performing model on the validation set has been retained for testing. Empirically, in both `lvl_3` and `lvl_6`, 20 epochs have shown of being a reasonable number of epochs since, on average, the models achieve their performance peak on the validation set around epoch 15.

The optimization was carried out using the Adam optimizer with learning rate $\eta = 10^{-3}$, $\epsilon = 10^{-8}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and 512 as batch size.

All the experiments have been performed using the Google Cloud Platform on a machine with 56GB of RAM, 8x Intel® Xeon® CPU @ 2.30GHz, and a single GPU Tesla P100 with 16GB of memory.

5.2 Evaluated models

We evaluated many different hyper-parametrization of the architectures described in Section 4.5. In the following we summarize the architectural details of each model:

- **mlp_only**: this model follows the MLP architecture described in Section 4.5. Missing words are only initialized (see Section 4.6) but not fine-tuned. When not differently stated, on `lvl_3` the size of the hidden layer is 2000, while on `lvl_6` is 500. We consider this model as our baseline.
- **ds1**, **ds2**: this is the DeepSets model. ds1 means single hidden layer and ds2 two hidden layers. Missing words are only initialized but not fine-tuned. When not differently stated the hidden layers of the DeepSets have size 300.
- **bMiss_mlp**: the same as mlp_only but with embedding fine-tuning (see Section 4.6) for the missing words.
- **bAll_mlp**: the same as bMiss_mlp but with embedding fine-tuning for all the words.
- **bMiss_ds1**, **bMiss_ds2**: this is the DeepSets model with the embedding fine-tuning for the missing words.
- **bAll_ds1**, **bAll_ds2**: same as the previous model but with embedding fine-tuning for all the words.

For each architecture we have empirically validated the size of the hidden layers on the validation set. We compare our models with a Naïve Bayes (**NB**) baseline, in which keywords are represented as Tf-Idf vectors.

5.3 Results on `lvl_6`

We evaluated several different architecture on `lvl_6`, and the achieved accuracy results on the test set are reported in Table 2. The table shows the performance of the models divided into three different groups: (*i*) models without the embedding fine-tuning, (*ii*) models with the embedding fine-tuning for the missing words, and (*iii*) models with the fine-tuning on all words. It is worth to underline that we omit from the table the results achieved by models with no missing words initialization nor fine-tuning because clearly inferior to all the other methods. For instance, the MLP architecture reaches 87.2% of accuracy.

As evident from the table, on each group the DeepSets model achieves higher accuracy than MLP (and NB) even though the performance with the fine-tuning on all words are comparable. We argue that most of the useful information to perform the task are directly encoded in the embedding during the training, leading to a kind of "good" overfitting. This is further noticeable by the fact that deeper DeepSets seem to not provide any significant gain in performance. In case of no embeddings fine-tuning, the DeepSets model outperform the MLP. Overall, the best accuracy has been achieved by the DeepSet model with a single hidden layer in the ϕ network and an hidden layer of size 2000 in the ρ network.

Model name	BP known	BP miss.	DS1	DS2	MLP	Accuracy
NB						84.2 %
mlp_only					✓	89.5 %
ds1			✓			91.8 %
ds2			✓	✓		**93.9** %
bMiss_mlp		✓			✓	91.8 %
bMiss_ds1		✓	✓			94.4 %
bMiss_ds2		✓	✓	✓		**94.8 %**
bAll_mlp	✓	✓			✓	96.0 %
bAll_ds1*	✓	✓	✓			**96.1** %
bAll_ds2	✓	✓	✓	✓		95.7 %

Table 2: Accuracy results on `lvl_6`. Results are divided into three groups w.r.t. the use of the embedding fine-tuning during training. The best results are highlighted in **bold**, while the overall best is underlined. (*) the ρ network has an hidden layer of size 2000.

5.4 Results on `lvl_3`

In order to save training time, we decided to use the results on `lvl_6` as a reference to select the architectures to evaluate on `lvl_3`. However, we also report the accuracy using the MLP, that is our baseline approach. The results on `lvl_3` are summarized in Table 5.4.

Model name	BP known	BP miss.	DS1	DS2	MLP	Accuracy
NB						75.1 %
mlp_only					✓	65.1 %
ds2			✓	✓		**73.7**%
bMiss_mlp		✓			✓	**78.6** %
bMiss_ds2		✓	✓	✓		76.6 %
bAll_mlp	✓	✓			✓	**82.3** %
bAll_ds1	✓	✓	✓			81.6 %

Table 3: Accuracy results on `lvl_3`. Results are divided into three groups w.r.t. the use of the embedding fine-tuning during training. The best results are highlighted in **bold**, while the overall best is underlined.

Likewise the experiments on `lvl_6`, when the fine-tuning is not used, the DeepSets model outperforms (+10%) the MLP model. However, when the embedding fine-tuning is considered (both on missing words and all words), the performance of the DeepSets model are inferior or comparable to the MLP model. We argue that this is due to overfitting caused by the smaller number of examples per class (see Table 1). Surprisingly, Naïve Bayes showed comparable or superior performance w.r.t. the not fine-tuned models, while it is significantly worse than the fine-tuned ones. It is also worth to notice that overall the accuracy is lower

on level three w.r.t. level six and this is reasonable given the number of classes which is 8 times more.

5.5 Time complexity

Efficiency-wise the proposed models are more than satisfactory from training to prediction. Clearly, the training phase is the most demanding one in terms of time and space. Time-wise the training process took at most 12 hours on a single GPU. The pre-processing phase, preformed on 8xCPU, takes around 8 seconds for 1M keywords, i.e., ∼125k keywords/sec. The inference times, performed using a single GPU, are summarized in Figure 3. Reasonably, the more complex

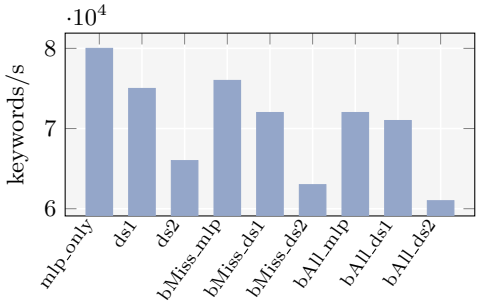

Fig. 3: Throughput of different architectures on `lvl_6`.

models are slower than the less complex ones. However all of them are capable of classifying more than 60k keywords per second. This means that one million keywords can be classified in at most 20 seconds, which is, efficiency-wise, an outstanding result. The same experiment has also been performed without the use of GPU. The relative efficiency is the same as the one in Figure 3, and in general the inference time is about 30% slower. The time complexity of the models on `lvl_3` are roughly the same as in `lvl_6`. Results are not reported for space reasons.

6 Conclusions and future work

In this paper, we proposed an efficient and effective keyword categorization system based on neural networks and word embeddings. The core computational advantage of our approach is that we consider the words in a keyword as a set and not a sequence of words. In this way, our models discard the sequentiality between words that with very short keywords (less than 4 words on average in both datsasets) is not much informative. This choice allows to obtain very fast models that, however, show good predictive performances.

In the future, we aim to further improve the classification model, for instance by considering attention-based models, such as set transformers [12] or other attention-based aggregation mechanism for sets [9]. Moreover, we will explore techniques to combine the predictions of the models at different levels of the taxonomy. The problem is not trivial since the probability estimates provided as output by neural networks (e.g., by the softmax function) are not representative of the true correctness likelihood [7]. A simple solution would be to identify a confidence threshold (on the validation set) that is used to decide if the considered keyword belongs to the current level or should be analyzed by models at deeper levels of the taxonomy. Although the output of the softmax function is not calibrated, it is consistent among different examples. Thus finding such threshold is a viable approach. A better approach would be to calibrate all the models at different taxonomy levels, still exploiting the validation set. This would enable to simply select the most probable prediction considering all taxonomy levels at the same time.

References

1. Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics **5**, 135–146 (2017)
2. Dai, H., Tian, Y., Dai, B., Skiena, S., Song, L., Financial, A.: Syntax-directed variational autoencoder for structurd data. In: ICLR (2018)
3. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. pp. 4171–4186. Association for Computational Linguistics, Minneapolis, Minnesota (2019)
4. Duong, L., Kanayama, H., Ma, T., Bird, S., Cohn, T.: Learning crosslingual word embeddings without bilingual corpora. In: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing. pp. 1285–1295. Association for Computational Linguistics, Austin, Texas (2016)
5. Gouws, S., Bengio, Y., Corrado, G.: Bilbowa: Fast bilingual distributed representations without word alignments. In: Bach, F., Blei, D. (eds.) Proceedings of the 32nd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 37, pp. 748–756. PMLR, Lille, France (2015)
6. Grave, E., Bojanowski, P., Gupta, P., Joulin, A., Mikolov, T.: Learning word vectors for 157 languages. In: Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018) (2018)
7. Guo, C., Pleiss, G., Sun, Y., Weinberger, K.Q.: On calibration of modern neural networks. In: Precup, D., Teh, Y.W. (eds.) Proceedings of the 34th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 70, pp. 1321–1330. PMLR (06–11 Aug 2017)
8. Hochreiter, S., Schmidhuber, J.: Long Short-Term Memory. Neural Computation **9**(8), 1735–1780 (1997)
9. Horn, M., Moor, M., Bock, C., Rieck, B., Borgwardt, K.: Set functions for time series (2020)

10. Lample, G., Conneau, A., Denoyer, L., Ranzato, M.: Unsupervised machine translation using monolingual corpora only. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. Association for Computational Linguistics (2018)
11. Lample, G., Conneau, A., Ranzato, M., Denoyer, L., Jégou, H.: Word translation without parallel data. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. Association for Computational Linguistics (2018)
12. Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., Teh, Y.W.: Set transformer: A framework for attention-based permutation-invariant neural networks. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 97, pp. 3744–3753. PMLR (09–15 Jun 2019)
13. Levy, O., Goldberg, Y.: Neural word embedding as implicit matrix factorization. In: Proceedings of the 27th International Conference on Neural Information Processing Systems. p. 2177–2185. NIPS'14, MIT Press, Cambridge, MA, USA (2014)
14. Li, Y., Xu, L., Tian, F., Jiang, L., Zhong, X., Chen, E.: Word embedding revisited: A new representation learning and explicit matrix factorization perspective. In: IJCAI (2015)
15. Mikolov, T., Le, Q.V., Sutskever, I.: Exploiting similarities among languages for machine translation. ArXiv **abs/1309.4168** (2013)
16. Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Proceedings of the 26th International Conference on Neural Information Processing Systems. p. 3111–3119. NIPS'13, Curran Associates Inc., Red Hook, NY, USA (2013)
17. Navarin, N., Vincenzi, B., Polato, M., Sperduti, A.: LSTM networks for data-aware remaining time prediction of business process instances. In: IEEE Symposium Series on Computational Intelligence, SSCI (2017)
18. Pennington, J., Socher, R., Manning, C.: GloVe: Global vectors for word representation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 1532–1543. Association for Computational Linguistics, Doha, Qatar (2014)
19. Schönemann, P.H.: A generalized solution of the orthogonal procrustes problem. Psychometrika **31**(1), 1–10 (1966)
20. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems. vol. 2017-Decem (2017)
21. Vulić, I., Moens, M.F.: Bilingual word embeddings from non-parallel document-aligned data applied to bilingual lexicon induction. In: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing. pp. 719–725. Association for Computational Linguistics, Beijing, China (2015)
22. Werbos, P.J.: Backpropagation Through Time: What It Does and How to Do It. Proceedings of the IEEE **78**(10), 1550–1560 (1990)
23. Zaheer, M., Kottur, S., Ravanbhakhsh, S., Póczos, B., Salakhutdinov, R., Smola, A.J.: Deep sets. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. p. 3394–3404. NIPS'17, Curran Associates Inc., Red Hook, NY, USA (2017)