Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 144 (2018) 60–70

Procedia Computer Science

www.elsevier.com/locate/procedia

INNS Conference on Big Data and Deep Learning 2018

# Domain Name Recommendation based on Neural Network

Kaoutar Benlamine[a,b], Nistor Grozavu[a], Younès Bennani[a], Rogovschi Nicoleta[c], Kamel Haddadou[b], Ahmed Amamou[b]

[a] *Sorbonne Paris Cité, Université Paris 13, 99 Av. J-B. Clément, 93430 Villetaneuse, France*
[b] *GANDI SAS, 63-65 Boulevard Masséna, 75013 Paris, France*
[c] *LIPADE, University Paris Descartes, 45 rue de Saintes Peres, 75005 Paris, France*

**Abstract**

The number of website is increasing speedily, and clients purchase their website from the enterprise that suggests them the best domain name with a good price. In order to give the relevant domain name, enterprise is always eager to have a good system of suggestion that suits the client request. Recommender system has been an effective key solution to guide users in a personalized way for discovering the domain name they might be interested in from a large space of possible suggestion. They have become fundamental applications that provides to users the best domain name that meet their needs and preferences. In this work, we used a recommender system based on neural network as it is capable to solve many complex tasks and gives better customer satisfaction. We proposed two approaches to recommend domain name, both of these approaches are based on neural network. The first one consists on discovering the similarity between the vocabulary of domain name, while the second one is finding the relevant Top Level Domain (TLD) corresponding to the context of the domain name. First experiments on GANDI datasets shows the effectiveness of the proposed approaches.



*Keywords:* Text Mining, Neural Network, Natural Language Processing.

## 1. Introduction

Recommender systems can be defined as programs which attempt to recommend the most suitable items, products or services to particular users by predicting a user's interest in an item based on related information about the items, the users and the interactions between items and users. The aim of developing recommender systems is to reduce information overload by retrieving the most relevant information and services from a huge amount of data, thereby providing personalized services. The most important feature of a recommender system is its ability to guess a user's

*E-mail address:* firstname.lastname@lipn.univ-paris13.fr

1877-0509 
10.1016/j.procs.2018.10.505

preferences and interests by analyzing the behavior of this user or the behavior of other users to generate personalized recommendations.

Various recommender system techniques have been proposed, and many sorts of recommender system software have been developed recently for a variety of applications. Researchers recognize that recommender systems offer great opportunities and challenges for business, government, education, and other domains, with more recent successful developments of recommender systems for real-world applications becoming apparent.

The past few decades have witnessed the tremendous successes of the use of neural network in many application domains such as speech recognition, image analysis and natural language processing. Applying neural network into recommender system has been gaining momentum thanks to its state-of-the-art performances and high-quality recommendations. In contrast to traditional recommendation models, neural network provides a better understanding of user's demands, item's characteristics and historical interactions between them.

In this paper, we suggest a system of recommendation of fully qualified domain name based on neural network. When a customer wants to purchase a domain name he rarely knows his qualified domain name, therefore, we will make a system of recommendation to suggest to the customer both the domain name and the extension that suits his recommend by referring to his localization and his research. The first part is the state of the art where we will present the existing methods concerning text similarity, through three approaches: Probabilistic-based approach, Prediction-based approach and Hybrid approach which is the combination of the two previous approaches. The second part is the description of the two models for suggesting domain name that are based on neural network: the first method tries to find similarity based on neighbors, while the second one predicts the TLD (Top Level Domain) corresponding to the context of domain name. The third part concerns the experiment results for the two models based on the real purchase database of GANDI.

## 2. State of the art

In this state of the art, we will present the most used methods to find text similarity, based on three approaches: 1) Probabilistic-based approach which is based on probabilistic models, and consists on searching in the whole corpus to find text similarity, 2) Prediction-based approach that more commonly known as embeddings or neural language models, which consists on looking for the neighbors of the text to find text similarity, 3) Hybrid approach which is the combination of the two previous approaches.

### *2.1. Probabilistic-based approach*

Probabilistic-Based approach is based on probabilistic models, this approach gives a sparse vector that can be easily interpretable by humans. However, the sparse vector can not be very efficient to detect the semantic meaning of the word.

#### *2.1.1. Latent Semantic Indexing*

Latent Semantic Indexing (LSI) [4] is a well-known method that allows to discover hidden concepts in a document. It represents the documents and terms in a way to express the semantic relationship between document/term, document/document or term/term. LSI constructs a term-document matrix from a large piece of text, this matrix describes the occurrences of terms in documents, it is a sparse matrix whose rows correspond to terms and whose columns correspond to documents. As an example of the weighting of the elements of the matrix is tf-idf (term frequency inverse document frequency): the weight of an element of the matrix is proportional to the number of times the terms appear in each document. After the construction of the occurrence matrix, LSI uses Singular Value Decomposition (SVD) to the term-document matrix in order to reduce the number of rows while preserving the similarity structure among columns. Then, the similarity between words could be compared using cosine similarity.

#### *2.1.2. Latent Dirichlet Allocation*

Latent Dirichlet allocation (LDA) [1] is a generative probabilistic model for collections of discrete data such as text corpora. LDA consider that documents are represented as random mixtures over latent topics, where each topic is characterized by a distribution over words. In other words, LDA allows to discover topics in text documents, where each topic is characterized by a list of words. The generative process for each document in the corpus is:

1. Choose a distribution over topics $\theta \sim Dir(\alpha)$
2. For each word in the document: choose a topic from the distribution over topics $z_n \sim Multinomial(\theta)$ then choose a word from the vocabulary $w_n \sim Multinomial(\phi_{z_n})$.

Each document gives the topic with certain probabilities, each topic in the document has some words with certain probabilities, these words are similar as they share the same context.

### *2.1.3. Global Vectors for Word Representation (Glove)*

Glove [16] is an unsupervised learning algorithm that gives vector representations of words. This algorithm is designed to take the advantages of the two major families of learning word vectors: global matrix factorization methods such as Latent Semantics Analysis [4] and local context window methods such as Skip gram model [13]. Glove uses ratios of co-occurrence probabilities, rather than the co-occurrence probabilities themselves, as the ratios between the probabilities of words appearing next to each other carry more information than considering those probabilities individually.

## *2.2. Prediction-based approach*

Prediction-based approach is more commonly known as embeddings or neural language models, this approach gives a dense vector that gives a good result in the semantic field of the word. However, this dense vector can not be easily interpretable by humans.

### *2.2.1. Word2vec*

Word2vec [12] is a model for learning vector representations of words, called "word Embeddings".
Word embeddings are a way to capture similarity across words based on the contexts in which they appear [17][10]. The intuition is that words with similar meanings often occur near each other in texts. There are two principal models in word2vec: Continuous Bag of Words (CBOW) that Predicts the current word based on the context and Skip gram that Predicts the context based on the current word.

In skip-gram model, we choose a window size C (often this window size is equal to five). Then, we train the neural network to do the following: Given the current word (input word), look at the C words nearby (C words behind and C words ahead). We train the neural network in the vocabulary in order to get the weights of the two matrix between the input/hidden layer ($W$) and hidden/output layer ($W'$).

Figure 1 shows the schema of the skip gram model. Let's consider that the vocabulary size is V, and the hidden layer size is N. The input is a one-hot encoded vector, which means for a given input word, only one out of V units, $x_1, ..., x_V$, will be 1, and all other units are 0. The units on adjacent layers are fully connected.

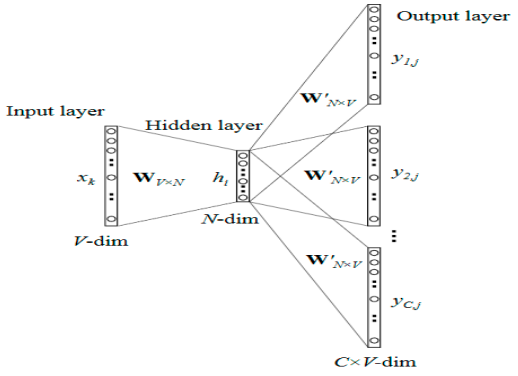


Fig. 1. The skip gram model

Finally, we will obtain two representations of each word in the vocabulary which are the rows of $W$ and $W'$. Our goal was to obtain the weights of two matrix $W_{V*N}$ and $W'_{N*V}$ to be able to get the vector representation of words.

After getting the vector representation of each word we can measure similarity between words in the vocabulary using cosine similarity.

### 2.2.2. *Paragraph Vector (Doc2vec)*

Paragraph Vector [9] is an unsupervised algorithm that models variable-length pieces of texts, such as sentences, paragraphs or documents. The algorithm is an adaptation of word2vec, while word2vec can generate word vector, paragraph vector can generate both document vector and word vector depending on a context (topic). Paragraph Vector has two approaches that are similar to the models of Word2Vec [12] (Continuous Bag of Words (CBOW) and Skip-gram).

- Distributed Memory Model of Paragraph Vectors (PV-DM)
  This approach is similar to CBOW, the only difference is that an additional input representing the paragraph id is added. PV-DM attempts to predict a word given its surrounding words and the context (topic) provided by the paragraph id. PV-DM uses the concept of memory, it remembers what is missing from the current context (topic) of the paragraph, which wasn't part of Word2vec. For example, a text that contains words about mathematics is more likely to use scientific words.
- Distributed Bag of Words version of Paragraph Vector (PV-DBOW)
  This model is similar to Skip-gram, the only difference is that it takes paragraph id as input and it predicts words in the window without any restriction on word order. PV-DBOW attempts to predict words randomly given the paragraph id.

Paragraph Vector has the potential to overcome the weaknesses of many machine learning algorithms as bag-of-words models [5]. Its robustness resides in: taking word order into account, taking the semantic of words into consideration, representing the text input as a variable length vector and learning from unlabeled data.

### 2.2.3. *Skip-Thought Vectors*

Skip-Thought Vectors [7] is a model that allows to get sentence vector and it doesn't require labeled data to train (unsupervised sentence vectors). This model inspires from skip-gram model [11] and attempts to extend it to sentences. So, instead of having a word to predict its context, Skip-Thought vectors have a sentence to predict the sentences around it. This model follows the encoder-decoder model that is so pervasive in neural machine translation [2]. Encoder maps words to a sentence vector and decoder is used to generate the surrounding sentences. Skip-thought model uses an RNN encoder with GRU activations and an RNN decoder with a conditional GRU. Gated Recurrent Unit (GRU) model [3] is simpler than Long Short Term Memory networks (LSTM) models [6], and has been shown to perform as well as LSTM on sequence modeling tasks.

There are one encoder and two decoders, all of them are constructed from recurrent neural networks (RNN). The encoder takes as input a sentence and outputs a vector. The two decoders take the vector as input: The first attempts to predict the previous sentence and the second attempts to predict the next sentence.

While training, Skip-Thought vectors uses two separate models: the first one "uni-skip" is unidirectional encoder (reads the sentence in the forward direction) while the other one "bi-skip" is bidirectional model with forward and backward encoders (reads the sentence forwards and backwards and concatenates the results). Another experiment "combine-skip" was added by using a combined model which consists of the concatenation of the vectors from uni-skip and bi-skip.

Figure 2 illustrates the model of Skip Thought, it shows the input sentence and the two predicted sentences. In this example, the input is the sentence triplet (a door confronted her, she stopped and tried to pull it open, it didn't budge).

Unattached arrows are connected to the encoder output. Colors indicate which components share parameters. $< eos >$ is the end of sentence token.

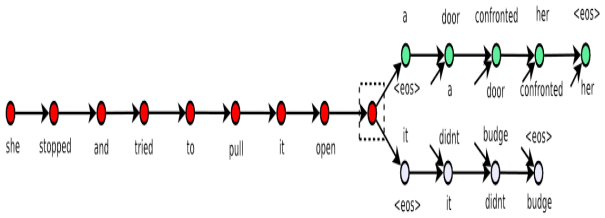


Fig. 2. The Skip Thought model

Skip-Thought framework proves to be a robust way of modeling the semantic content of sentences through the following tasks: semantic relatedness, paraphrase detection, image-sentence ranking and classification benchmarks.

### 2.3. Hybrid approach

The question is whether distributional word representations are best learned from Probabilistic-based models or from Prediction-based models?

Probabilistic-based approach tend to form documents as a sparse mixed-membership of topics while Prediction-based approach tend to model documents as dense vectors. The former is simpler to interpret as the vector is sparse while the second one is better in giving the semantic of the word but the vector is uninterpretable. To take advantages of both approaches, hybrid approach combines the best part of the two previous approaches.

#### 2.3.1. LDA2vec

Lda2vec [14] is a model that mixes two models: the dirichlet topic model (LDA) and word embeddings (word2vec). LDA [1] is able to create document (and topic) representations that are not so flexible but quite interpretable to humans. While word2vec [11] create vector representations of words that can model semantic similarity between words but the vectors are uninterpretable by humans. Thus, Lda2vec aims to solve this by embedding both words and document vectors into the same space and trains both representations simultaneously. Inspiring from the idea of Tomas Mikolov [12] that word vectors can be summed together to form a semantically meaningful combination of both words. We will add word and document vectors to each other to construct a meaningful context vector of them.

For example, taking the word *Germany* and a document about *airlines*. Then the predicted similar words for *Germany* will be *France*, *Spain*, and *Austria*. But if we apply lda2vec, instead of predicting tokens similar to *Germany* alone, the predictions will be similar to both the document and the word, which will give as a result: *Lufthansa*, *Condor Flugdienst*, and *Aero Lloyd*.

#### 2.3.2. Topic2vec

Topic2vec [15] is an approach that learns topic representations in the same semantic vector space with words. This model is inspired by word2vec, it is also separated in two models: CBOW which predicts both a word and topic vector using the context words, and Skip gram that predicts the context given the current word and its assigned topic by LDA.

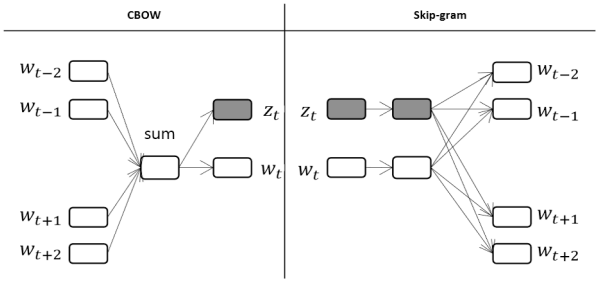


Fig. 3. The architectures of Topic2Vec where $(w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2})$ are context words and $w_t$ is the current word paired with a topic $z_t$

Topic2Vec aims at learning topic representations along with word representations. When training, given a word-topic sequence of a document $D = w_1 : z_1, ..., w_N : z_N$ , where $w_i$ is the word and $z_i$ is the topic inferred from LDA. The learning objective function for CBOW and Skip gram can be defined as follows:

$$F_{CBOW}(D) = \frac{1}{M}\sum_{i=1}^{M}(log\ p(w_i|w_{cxt}) + log\ p(z_i|w_{cxt}))$$

$$F_{Skip-gram}(D) = \frac{1}{M}\sum_{i=1}^{M}\sum_{-k\leqslant c\leqslant k}(log\ p(w_{i+c}|w_i) + log\ p(w_{i+c}|z_i))$$

Thus, topic representations are learned in the same semantic vector space with words.

## 3. The proposed approaches

In this section, we present two approaches: the first one aims to find similarity between the vocabulary while the second one is finding the TLD that corresponds to the context of domain name.

### *3.1. First approach*

This approach is based on training the neural network to do the following: looking for the neighbors for each word in the purchase database in order to find similarity between words of the domain name. First of all, we will preprocess on the purchase data; we will retrieve the purchase data to tokenize the domain name. Then, we will remove all the noise (stop words). After this, we will build the vocabulary which will be the fully qualified domain name which means the union of the domain name and TLD. After this, we will train this model on the purchase database which will allow us to get the word vector for each word in the vocabulary. Finally, we will make a scoring between the domain name and each TLD and we will suggest the TLD with a score above to $\gamma$.

#### *3.1.1. Learning phase*

We choose a window size $C$, which means, for each word in the vocabulary (domain name, TLD) we will look for the $C$ words behind and the $C$ words ahead. We will train the neural network to do the following: Given the input word, look at the C words nearby. The input and output for the first approach is a list of vectors that has the size of

the vocabulary, each word is represented as a one hot vector, which means, for a given input word, only one out of the words in the vocabulary will be 1, and all other words are 0). The vocabulary is composed of domain name and TLD.

**Learning phase**
Choose a window size $C$.
Look at the $C$ words nearby.
**parameter:** Parameter for the algorithm
$C$: window size, $n$: neurones, $h_i$: hidden layer, $e_r$: error threshold, $\gamma$: score threshold, $W$: weights (randomly initialized)
**Input :** A list of vectors that has the size of the vocabulary.
**Output :** The same list of input (vocabulary).

Choose a couple (Input, Output).
Calculate the state of the network by propagation.
Determine the cost.
Apply gradient descent.
Adjust the weights $W$.
Repeat until obtaining an error equal to $e_r$.
**Prediction phase**
Applying cosine similarity between the vector of domain name and each TLD.
Recommend the TLD with the score above of $\gamma$.

**Algorithm 1:** First approach

#### *3.1.2. Prediction phase*

After training the neural network, we will get the word vector for each word in the vocabulary (Domain name and TLD). The prediction of TLD will be done by making a score between the domain name and every TLD. Then, we will recommend the TLD with a score above $\gamma$ (fixed in the output).

### *3.2. Second approach*

This approach is based on constructing a neural network that takes as input the domain name and outputs a TLD. We will train the neural network on the purchase data in order to suggest the relevant TLD corresponding to the context of the domain name. This approach is divided into two parts: the first part is training the neural network to predict the context of the domain name while the second part is based on classifying the TLD using an unsupervised learning method. First of all, we will preprocess on the purchase data; we will retrieve the purchase data to tokenize the domain name. Then, we will remove all the noise (stop words). After this, we will build the vocabulary.

#### *3.2.1. Learning phase*

- `Learn to predict the context of the domain name`
  After building the vocabulary, we will train the neural network on that vocabulary, in order to find the context of the domain name. The neural network will be trained to do the following: Taking the domain name as input and outputs the TLD corresponding to the context of the domain name.

The input for the second approach is a list of vectors where each vector is represented as a one hot vector and each vector has the dimension of the vocabulary of the domain name. The output is a list of vectors where each vector is represented as a one hot vector and each vector has the dimension of the vocabulary of the TLD.

**Learning phase**
*Prediction of the context of domain name*
*Classification of TLD*
**parameter:** Parameter for the algorithm
$n$: neurones, $h_i$: hidden layer, $e_r$: error threshold, $W$: weights (randomly initialized).
**Input :** A list of vectors (Domain Name).
**Output :** A list of vectors (TLD).

Choose a batch of the couple (Input, Output).
Calculate the state of the network by propagation.
Determine the cost.
Apply gradient descent.
Adjust the weights.
Repeat until obtaining an error equal to $e_r$.
**Prediction phase**
Look for the cluster of the predicted TLD.
Apply cosine similarity between the predicted TLD and each TLD in the cluster.
Recommend the top $K$ TLD.

**Algorithm 2:** Second approach

- `Classification of TLD`
To classify TLDs, we will take the word vector for each TLD from the model of the first approach that we have already trained it. After getting the word vector for each TLD, we will calculate the TLD similarity matrix. Then, we will classify TLDs using an unsupervised learning method. We used an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables. This transformation is defined in such a way that the first component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. The resulting vectors are an uncorrelated orthogonal basis set.
K-means is an unsupervised machine learning algorithm that groups a dataset into a specified number (k) of clusters. The algorithm is somewhat naive as it clusters the data into k clusters, even if k is not the right number of clusters to use. Therefore, when using k-means clustering, users need some way to determine whether they are using the right number of clusters. So, here we will present how to choose the optimal number of clusters using elbow method [8], then we will apply k-means clustering.

  - Determine optimal number of clusters (K)
  The technique to determine the number of clusters(K), is called the elbow method. The idea of the elbow method is that it looks at the percentage of variance explained as a function of the number of clusters: One should choose a number of clusters so that adding another cluster doesn't give much better modeling of the data. Figure 4 shows an elbow, it is called an elbow method because the curve looks like a human arm and the elbow point gives us the optimum number of clusters. So, we will take the elbow point value as the final number of clusters, which is K=24.

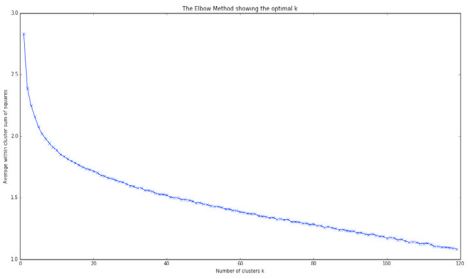


Fig. 4. The elbow method

– Applying K-means clustering
After choosing the right K (number of clusters), we will apply K-means clustering with K=24. Figure 5 shows the clusters of TLDs:

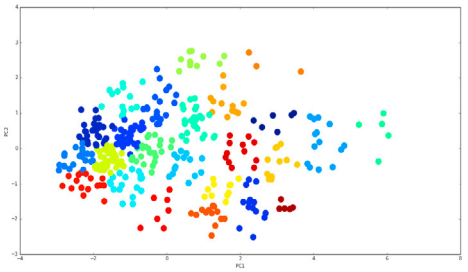


Fig. 5. Clusters of TLDs

*3.2.2. Prediction phase*

After training the neural network, we will use it to predict the TLD that corresponds to the context of the domain name. Then, we will search for the cluster that corresponds to the predicted TLD, and we will make cosine similarity between the predicted TLD and each TLD in the cluster, Then we will suggest the top *K* TLDs, which means the one who has high cosine similarity, and we will save the decreasing order as the client could see the relevant TLD in the beginning.

## 4. Experimental results

In this section, we will present the experimental results that we get from the two approaches. In this experiment, the sample was taking from Gandi database, the real purchases that has been done.

1. **First approach**
In this approach, we took a sample from the purchase database of Gandi that has 10 million purchase, and we test the accuracy. The accuracy was based on : if the predicted TLD coincide with the 10 TLD that has the highest score.

| Dataset | Domain Name | TLD | accuracy |
|---|---|---|---|
| 10M | 6.293.598 | 386 | 3,98% |

Table 1. The evaluation of the first approach

Indeed, the goal of this approach is to find similarity among the purchase data not the context of the domain name that will give the relevant TLD, that's why we took all the domain name that were purchased not only the one that were bought with GTLD (Generic Top Level Domain). This low accuracy is probably due to the domain name that has been purchased with ccTLD (Country Code Top Level Domain).

2. **Second approach**
In this approach we took a sample of 10.000 purchase domain name, to be able to make a setting on the neural network in order to choose the best parameters that will give the best results.

| Dataset | Domain Name | TLD |
|---|---|---|
| 10K | 7873 | 45 |

Table 2. The evaluation of the second approach

The activation function is a node that we add to the output end of any neural network. It is used to determine the output of neural network like yes or no. It maps the resulting values in between 0 to 1 or -1 to 1 etc. (depending upon the function). To train the neural network, we tested three activation function for the hidden layers : sigmoid function, tanh (hyperbolic tangent), Relu (Rectified Linear Unit).

| | Sigmoid $(\frac{1}{1+e^{-x}})$ | Relu $(\max(0, x))$ | tanh $(\frac{2}{1+e^{-2x}} - 1)$ |
|---|---|---|---|
| One hot vector | 96.9% | 96.76% | 96.89% |
| Vector using indexes | 81.71% | 80.67% | 81.08% |

Table 3. Accuracy by comparing different activation functions on 10K sample

Figure 6 shows the accuracy and the cross entropy using one hot vector with sigmoid function while figure 7 shows the accuracy and cross entropy using a vector with indexes with sigmoid function. It is noticeable that when the cross entropy goes down the accuracy increases until reaching a predefined threshold. The axis of abscissa represents the number of iteration while the axis of the ordinate represents the accuracy (left) and the cross entropy (right). Results with one hot vector is much better than using vector with indexes, as the accuracy is much better and also the number of iteration is lower than the one used by the vector with indexes.

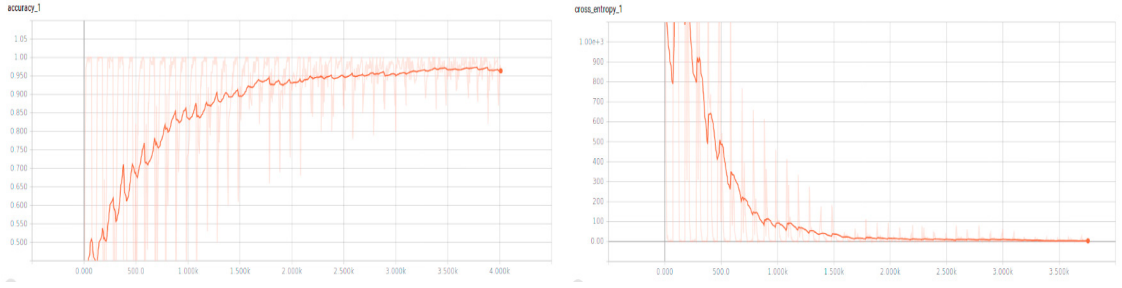


Fig. 6. Results of accuracy (left) and cross entropy (right) using one hot vector and sigmoid function

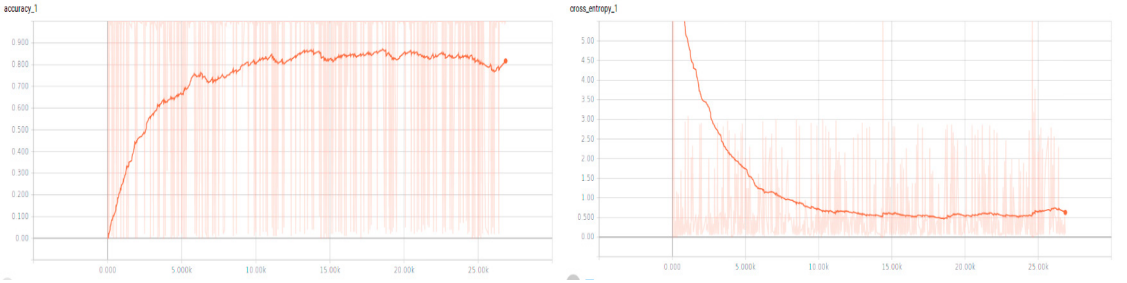


Fig. 7. Results of accuracy (left) and cross entropy (right) using vector with indexes and sigmoid function

## 5. Conclusion

In this work, we have proposed two approaches based on neural network to suggest a relevant domain name. The first approach consists on finding similarity between words of domain name, while the second one is predicting TLD corresponding to the context of domain name. Experiments on GANDI database has shown efficient results for the second approach.

## References

[1] Blei, D.M., Ng, A.Y., Jordan, M.I., 2003. Latent dirichlet allocation. Journal of machine Learning research 3, 993–1022.
[2] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y., 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078 .
[3] Chung, J., Gulcehre, C., Cho, K., Bengio, Y., 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555 .
[4] Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., Harshman, R., 1990. Indexing by latent semantic analysis. Journal of the American society for information science 41, 391.
[5] Harris, Z.S., 1954. Distributional structure. Word 10, 146–162.
[6] Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural computation 9, 1735–1780.
[7] Kiros, R., Zhu, Y., Salakhutdinov, R.R., Zemel, R., Urtasun, R., Torralba, A., Fidler, S., 2015. Skip-thought vectors, in: Advances in neural information processing systems, pp. 3294–3302.
[8] Kodinariya, T.M., Makwana, P.R., 2013. Review on determining number of cluster in k-means clustering. International Journal 1, 90–95.
[9] Le, Q., Mikolov, T., 2014. Distributed representations of sentences and documents, in: Proceedings of the 31st International Conference on Machine Learning (ICML-14), pp. 1188–1196.
[10] Meyer, D., 2016. How exactly does word2vec work? .
[11] Mikolov, T., Chen, K., Corrado, G., Dean, J., 2013a. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 .
[12] Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J., 2013b. Distributed representations of words and phrases and their compositionality, in: Advances in neural information processing systems, pp. 3111–3119.
[13] Mikolov, T., Yih, W.t., Zweig, G., 2013c. Linguistic regularities in continuous space word representations., in: hlt-Naacl, pp. 746–751.
[14] Moody, C.E., 2016. Mixing dirichlet topic models and word embeddings to make lda2vec. arXiv preprint arXiv:1605.02019 .
[15] Niu, L., Dai, X., Zhang, J., Chen, J., 2015. Topic2vec: learning distributed representations of topics, in: Asian Language Processing (IALP), 2015 International Conference on, IEEE. pp. 193–196.
[16] Pennington, J., Socher, R., Manning, C.D., 2014. Glove: Global vectors for word representation., in: EMNLP, pp. 1532–1543.
[17] Rong, X., 2014. word2vec parameter learning explained. arXiv preprint arXiv:1411.2738 .